



사용자 가이드

AWS Glue



AWS Glue: 사용자 가이드

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 브랜드 디자인은 타사 제품 또는 서비스와 함께, 당사 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon 계열사, 관련 업체 또는 Amazon의 지원 업체 여부에 상관없이 해당 소유자의 자산입니다.

Table of Contents

AWS Glue이란 무엇인가요?	1
AWS Glue 기능	2
AWS Glue의 혁신에 대해 알아보기	3
AWS Glue 시작하기	3
AWS Glue 액세스	4
관련 서비스	4
작동 방식	5
격리 중인 서버리스 ETL 작업 실행	6
개념	7
AWS Glue 용어	8
구성 요소	11
AWS Glue 콘솔	11
AWS Glue Data Catalog	12
AWS Glue 크롤러 및 분류자	13
AWS Glue ETL 연산	13
AWS Glue의 스트리밍 ETL	13
AWS Glue 작업 시스템	14
시각적 ETL 구성 요소	14
AWS Glue for Spark 및 AWS Glue for Ray	19
AWS Glue for Ray란 무엇인가요?	20
반구조화된 스키마를 관계형 스키마로 변환하기	21
AWS Glue 유형	23
AWS Glue 데이터 카탈로그 유형	23
Spark를 사용한 AWS Glue 스크립트의 유형	23
AWS Glue 크롤러 유형	24
시작하기	25
AWS Glue 사용 개요	25
IAM 권한 설정	27
다음 단계	32
시각적 ETL을 사용하기 위한 IAM 권한	32
AWS Glue Studio에서 노트북 시작하기	42
사용 프로필 설정	45
사용 프로필 관리	46
사용 프로필 및 작업	58

AWS Glue Data Catalog 시작하기	58
개요	59
1단계: 데이터베이스 생성	59
단계 2. 테이블 생성	61
다음 단계	62
데이터 스토어에 대한 네트워크 액세스 설정	65
AWS Glue에서 PyPI에 연결하도록 VPC 설정	66
VPC에서 DNS 설정	68
암호화 설정	69
개발에 대한 네트워킹 설정	73
개발 엔드포인트에 대한 네트워크 설정	73
노트북 서버용 Amazon EC2 설정	75
데이터 검색 및 카탈로그 작성	77
데이터 카탈로그 채우기	79
AWS Glue 크롤러 사용	80
수동으로 메타데이터 정의	176
다른 AWS 서비스와 통합	194
데이터 카탈로그 설정	195
트랜잭션 테이블 채우기 및 관리	198
Iceberg 테이블 생성	198
Iceberg 테이블 최적화	201
데이터 카탈로그 관리	213
스키마 업데이트 및 새 파티션 추가	214
열 통계를 사용한 쿼리 성능 최적화	222
데이터 카탈로그 암호화	233
Lake Formation을 사용한 데이터 카탈로그 보호	233
데이터 카탈로그 액세스	234
데이터 카탈로그 모범 사례	235
AWS Glue Schema Registry	236
스키마	237
레지스트리	239
스키마 버전 관리 및 호환성	240
오픈 소스 Serde 라이브러리	244
Schema Registry의 할당량	245
작동 방식	245
시작하기	247

AWS Glue Schema Registry와 통합	269
AWS Glue Schema Registry로 마이그레이션	295
데이터에 연결	297
AWS Glue 연결 속성	298
필수 연결 속성	299
JDBC 연결 속성	300
MongoDB 및 MongoDB Atlas 연결 속성	304
세일즈포스 연결 속성	305
Snowflake 연결	306
Vertica 연결	307
SAP HANA 연결	307
Azure SQL 연결	308
Teradata Vantage 연결	309
OpenSearch 서비스 연결	310
Azure Cosmos 연결	311
SSL 연결 속성	311
인증을 위한 Kafka 연결 속성	314
구글 BigQuery 커넥션	315
Vertica 연결	307
AWS Secrets Manager에 연결 자격 증명 저장	315
AWS Glue 연결 추가	316
Redshift에 연결	317
Azure Cosmos DB에 연결	321
Azure SQL에 연결	324
연결 대상 BigQuery	327
MongoDB에 연결	331
OpenSearch Service에 연결	335
세일즈포스에 연결	338
SAP HANA에 연결	349
Snowflake에 연결	353
Teradata에 연결	356
Vertica에 연결	360
커넥터 및 연결 사용	363
데이터 원본에 연결	390
자체 JDBC 드라이버를 사용하여 JDBC 연결 추가	398
AWS Glue 연결 테스트	402

AWS 호출이 VPC를 통과하도록 구성	403
VPC의 JDBC 데이터 스토어로 연결	404
탄력적 네트워크 인터페이스를 사용하여 VPC 데이터에 액세스	404
탄력적 네트워크 인터페이스 속성	405
MongoDB 또는 MongoDB Atlas 연결 사용	406
VPC 엔드포인트를 사용한 Amazon S3 데이터 스토어 크롤링	407
필수 조건	407
Amazon S3에 대한 연결 생성	408
Amazon S3에 대한 연결 테스트	411
Amazon S3 데이터 스토어의 크롤러 생성	413
Amazon S3 기반 데이터 카탈로그 테이블의 크롤러 생성	415
크롤러 실행	416
문제 해결	416
연결 문제 해결	416
자습서: AWS Glue Connector for Elasticsearch 사용	417
필수 조건	418
1단계: (선택 사항) OpenSearch 클러스터 정보에 대한 AWS 보안 암호 생성	418
2단계: 커넥터 구독	419
3단계: AWS Glue Studio에서 커넥터 활성화 및 연결 생성	420
4단계: ETL 작업에 대한 IAM 역할 구성	420
5단계: OpenSearch 연결을 사용하는 작업 생성	421
6단계: 작업 실행	422
대화형 세션을 사용하여 AWS Glue 작업 구축	424
AWS Glue 대화형 세션 개요	424
제한 사항	425
AWS Glue 대화형 세션 시작하기	425
대화형 세션을 로컬에서 설정하기 위한 사전 조건	425
주피터 및 AWS Glue 인터랙티브 세션 Jupyter 커널 설치	425
Jupyter 실행	425
세션 자격 증명 및 리전 구성	426
대화형 세션 평가판 업그레이드	427
SageMaker Studio에서 대화형 세션 사용	428
Microsoft Visual Studio Code에서 대화형 세션 사용	428
Jupyter 및 AWS Glue Studio 노트북용 AWS Glue 대화형 세션 구성	431
Jupyter 매직 소개	431
Jupyter용 AWS Glue 대화형 세션에서 지원되는 매직	432

세션 이름 지정	449
대화형 세션에 대한 IAM 역할 지정	449
명명된 프로파일을 사용하여 세션 구성	450
AWS Glue Ray 인터랙티브 세션용 (미리 보기)	451
콘솔의 Ray 인터랙티브 세션 AWS Glue Studio	451
Jupyter 커널을 사용하는 Ray 대화형 세션	452
Ray 대화형 세션 제한 시간 기본값	452
AWS Glue Ray 대화형 세션에서 지원되는 매직	453
IAM을 이용한 대화형 세션	454
대화형 세션에 사용되는 IAM 보안 주체	454
클라이언트 보안 주체 설정	455
런타임 역할 설정	455
세션을 비공개로 설정하세요. TagOnCreate	456
IAM 정책 고려 사항	461
스크립트 또는 노트북을 AWS Glue 작업으로 변환	462
스트리밍을 위한 AWS Glue 대화형 세션	462
스트리밍 세션 유형 전환	462
대화형 개발을 위한 샘플링 입력 스트림	462
대화형 세션에서 스트리밍 애플리케이션 실행	464
로컬로 개발 및 테스트	465
AWS Glue Studio를 사용하여 개발	465
대화형 세션을 사용하여 개발	466
Docker 이미지를 사용하여 개발	466
AWS Glue ETL 라이브러리를 통한 개발	477
dev 엔드포인트	485
개발 엔드포인트에서 대화형 세션으로 마이그레이션	486
개발 엔드포인트를 사용하여 스크립트 개발	488
노트북 관리	515
AWS Glue Studio를 사용하여 시각적 ETL 작업 구축	518
콘솔로 로그인합니다	518
AWS Glue Studio에서 작업을 생성하기 위한 다음 단계	518
AWS Glue Studio을(를) 사용한 시각적 ETL	519
AWS Glue Studio에서 작업 시작	519
작업 편집기 기능	521
AWS Glue 관리형 데이터 변환 노드 편집	529
사용자 지정 시각적 변환	590

AWS Glue Studio에서 데이터 레이크 프레임워크 사용	607
데이터 대상 노드 구성	618
작업 스크립트 편집 또는 업로드	623
작업 다이어그램에서 노드의 상위 노드 변경	627
작업 다이어그램에서 노드 삭제	628
AWS Glue 데이터 카탈로그 노드에 소스 및 대상 파라미터 추가	632
AWS Glue에서 Git 버전 제어 시스템 사용	634
AWS Glue Studio 노트북을 사용한 코드 작성	642
노트북 사용 개요	643
AWS Glue Studio에서 노트북을 사용하여 ETL 작업 생성	644
노트북 편집기 구성 요소	645
노트북 및 작업 스크립트 저장	646
노트북 세션 관리	647
CodeWhisperer 와 함께 사용 AWS Glue Studio notebooks	648
작업 실행 보기	649
작업 모니터링 대시보드에 액세스	649
작업 모니터링 대시보드 개요	649
작업 실행 보기	649
작업 실행 로그 보기	653
작업 실행의 세부 정보 보기	654
Spark 작업 실행에 대한 Amazon CloudWatch 지표 보기	657
Ray 작업 실행에 대한 Amazon CloudWatch 지표 보기	657
민감한 데이터 감지 및 처리	659
데이터를 스캔하는 방법 선택	660
탐지할 PII 엔터티 선택	661
탐지 민감도 수준 지정	665
식별된 PII 데이터로 수행할 작업 선택	665
세분화된 작업 오버라이드 추가	666
작업 관리	667
작업 실행 시작	668
작업 실행 예약	668
작업 일정 관리	669
작업 실행 중지	670
작업 보기	670
최근 작업 실행에 대한 정보 보기	671
작업 스크립트 보기	672

작업 속성 수정	672
작업 저장	675
작업 복제	677
작업 삭제	677
작업 사용	679
AWS Glue 버전	679
AWS Glue 버전	679
단축된 시작 시간으로 Spark ETL 작업 실행	692
AWS Glue 버전 3.0으로 AWS Glue for Spark 작업 마이그레이션	697
AWS Glue 버전 4.0으로 AWS Glue for Spark 작업 마이그레이션	705
AWS Glue for Ray(평가판)에서 AWS Glue for Ray로 마이그레이션	719
AWS Glue 버전 지원 정책	720
Spark 작업 사용	722
작업 파라미터	722
스파크 앤 잡스 PySpark	731
스트리밍 ETL 작업	857
FindMatches로 레코드 매칭	871
Spark 프로그램 마이그레이션	904
Ray 작업 사용	911
AWS Glue for Ray 시작하기	911
지원되는 Ray 런타임 환경	912
Ray 작업에서 작업자 고려	913
Ray 작업 파라미터	914
Ray 작업 지표	917
Python 셸 작업 속성 구성	918
제한 사항	918
Python 셸 작업에 대한 작업 속성 정의	918
Python 셸 작업에 지원되는 라이브러리	920
자체 Python 라이브러리 제공	922
AWS Glue에서 Python 셸 작업과 함께 AWS CloudFormation 사용	925
모니터링	926
AWS 태그	927
CloudWatch Events로 자동화	932
AWS Glue 리소스 모니터링	934
CloudTrail을 사용하여 로깅	937
작업 실행 상태	940

AWS Glue 스트리밍	943
스트리밍 사용 사례	943
AWS Glue 스트리밍을 사용하면 어떤 이점이 있나요?	944
스트리밍은 언제 사용하나요? AWS Glue	944
지원되는 데이터 원본	945
지원되는 데이터 대상	946
자습서: AWS Glue Studio를 사용하여 첫 번째 스트리밍 워크로드 구축	946
필수 조건	946
Amazon Kinesis의 스트리밍 데이터 소비	946
자습서: AWS Glue Studio 노트북을 사용하여 첫 번째 스트리밍 워크로드 구축	957
필수 조건	958
Amazon Kinesis의 스트리밍 데이터 소비	958
스트리밍 개념	965
AWS Glue 스트리밍 작업의 구조	965
Kafka 연결	969
Kinesis 연결	975
스트리밍 옵션	982
AWS Glue 스트리밍 자동 크기 조정	983
AWS Glue Studio에서 Auto Scaling 사용	983
AWS CLI 또는 SDK를 사용하여 Auto Scaling 사용 설정	984
작동 방식	985
유지 관리 기간	987
유지 관리 기간 설정	987
유지 관리 기간 동작	988
Job 모니터링	989
데이터 손실 처리	991
고급 AWS Glue 스트리밍 개념	992
스트림 처리 시 시간 고려 사항	992
기간 설정	992
최신 데이터 및 워터마크 처리	998
AWS Glue 스트리밍 작업 모니터링	1000
지표 시각화	1000
지표 심층 분석	1002
최고의 성능을 얻는 방법	1007
AWS Glue 데이터 품질	1009
이점 및 주요 특징	1009

작동 방식	1010
의 데이터 품질 AWS Glue Data Catalog	1010
AWS Glue ETL 작업의 데이터 품질	1010
AWS Glue 데이터 품질 진입점 비교	1011
고려 사항	1012
용어	1013
Limits	1014
데이터 품질에 대한 릴리스 노트 AWS Glue	1014
정식 출시: 새 기능	1014
2023년 11월 27일(미리 보기)	1014
2024년 3월 12일	1014
2024년 6월 26일	1015
AWS Glue Data Quality의 이상 탐지	1015
작동 방식	1015
분석기를 사용하여 데이터 검사	1016
DetectAnomaly 규칙 사용	1017
이상 탐지의 이점 및 사용 사례	1017
AWS Glue Data Quality에 대한 IAM 권한	1018
IAM 권한	1018
평가 실행을 예약하는 데 필요한 IAM 설정	1020
예제 IAM 정책	1021
Data Catalog에서 AWS Glue Data Quality 시작하기	1025
필수 조건	1025
단계별 예제	1026
규칙 권장 사항 생성	1026
모니터링 규칙 권장 사항	1027
권장 규칙 세트 편집	1028
새 규칙 세트 생성	1029
규칙 세트를 실행하여 데이터 품질 평가	1030
데이터 품질 점수 및 결과 보기	1031
관련 주제	1032
AWS Glue Studio에서 데이터 품질 평가	1032
이점	1033
AWS Glue Studio에서 ETL 작업에 대한 데이터 품질 평가	1033
데이터 품질 규칙 작성기	1038
이상 탐지 구성 및 인사이트 생성	1043

AWS Glue Studio 노트북의 ETL 작업에 대한 데이터 품질	1047
사전 조건	1047
AWS Glue Studio에서 ETL 작업 생성	1048
데이터 품질 정의 언어(DQDL) 참조	1052
구문	1054
규칙 유형 참조	1067
API를 사용하여 데이터 품질 측정 및 관리	1110
필수 조건	1110
AWS Glue Data Quality 권장 사용	1111
AWS Glue Data Quality 규칙 세트 사용	1114
AWS Glue Data Quality 실행 사용	1116
AWS Glue Data Quality 결과 사용	1120
알림, 배포 및 예약 설정	1122
Amazon EventBridge 통합에서 알림 및 알림 설정	1122
알림 및 알림을 CloudWatch 통합하여 설정합니다.	1130
데이터 품질 결과 쿼리	1131
데이터 품질 규칙 배포	1135
데이터 품질 규칙 예약	1135
AWS Glue 데이터 품질 오류 문제 해결	1136
오류: 누락된 모듈	1136
오류: 권한 부족	1137
오류: 규칙 세트가 고유하지 않음	1137
오류: 특수 문자가 있는 테이블	1137
오류: 큰 규칙 세트로 인한 오버플로	1137
오류: 규칙 상태가 실패임	1137
AnalysisException: 기본 데이터베이스가 있는지 확인할 수 없습니다.	1138
제공된 키 맵이 해당 데이터 프레임에 적합하지 않음	1138
자바.랭. RuntimeException : 데이터를 가져오지 못했습니다.	1138
시작 오류: 버킷에 대해 S3에서 다운로드하는 중 오류 발생	1139
InvalidInputException (상태: 400): DataQuality 규칙을 파싱할 수 없습니다.	1139
오류: Eventbridge에서 사용자가 설정한 일정에 따라 Glue DQ 작업을 트리거하지 않습니 다.	1140
CustomSQL 오류	1140
동적 규칙	1140
사용자 클래스에서의 예외: org.apache.spark.sql AnalysisException: org.apache.hadoop.hive.ql.metadata. HiveException	1142

UNCLASSIFIED_ERROR; IllegalArgumentException: 파싱 오류: 규칙이나 분석기가 제공되지 않았습 니다. 입력 시 실행 가능한 대안이 없습니다.	1143
Amazon Q 데이터 통합 AWS Glue	1144
Amazon Q란 무엇인가요?	1144
Amazon Q 데이터 통합 AWS Glue	1144
Amazon Q 데이터 통합을 사용하여 작업	1145
모범 사례	1146
서비스 개선	1147
고려 사항	1147
Amazon Q 데이터 통합 설정	1147
IAM 권한 구성	1148
지원되는 코드 생성	1150
상호 작용 예제	1151
Amazon Q 채팅 상호 작용	1151
AWS Glue 스튜디오 노트북 상호작용	1153
오케스트레이션	1157
트리거를 사용하여 작업 및 크롤러 시작	1157
AWS Glue 트리거	1157
트리거 추가	1160
트리거 활성화 및 비활성화	1164
블루프린트 및 워크플로를 사용하여 복잡한 ETL 활동 수행	1164
워크플로우 개요	1165
워크플로 수동 생성 및 구축	1169
EventBridge 이벤트로 워크플로 시작	1173
워크플로를 시작한 EventBridge 이벤트 보기	1180
워크플로 실행 및 모니터링	1181
워크플로 실행 중지	1184
워크플로 실행 복구 및 재개	1185
워크플로 실행 속성 가져오기 및 설정	1191
AWS Glue API를 사용하여 워크플로 쿼리	1192
블루프린트 및 워크플로 제한 사항	1196
블루프린트 오류 해결	1197
블루프린트 페르소나 및 역할에 대한 권한	1202
블루프린트 개발	1206
블루프린트 개요	1207
블루프린트 개발	1210

블루프린트 등록	1234
블루프린트 보기	1236
블루프린트 업데이트	1238
블루프린트에서 워크플로 생성	1240
블루프린트 실행 보기	1242
AWS Glue용 AWS CloudFormation	1244
샘플 데이터베이스	1246
샘플 데이터베이스, 테이블 및 파티션	1247
샘플 Grok 분류자	1251
샘플 JSON 분류자	1252
샘플 XML 분류자	1253
샘플 Amazon S3 크롤러	1254
샘플 연결	1256
샘플 JDBC 크롤러	1257
Amazon S3에서 Amazon S3로의 샘플 작업	1260
JDBC에서 Amazon S3로의 샘플 작업	1261
샘플 온디맨드 트리거	1263
일정이 정해진 샘플 트리거	1264
샘플 조건부 트리거	1265
샘플 기계 학습 변환	1266
샘플 데이터 품질 규칙 세트	1268
EventBridge 스케줄러에서 샘플 데이터 품질 규칙 세트	1269
샘플 개발 엔드포인트	1271
AWS Glue 프로그래밍 안내서	1273
사용자 지정 스크립트 제공	1273
AWS Glue for Spark	1274
자습서: Spark 스크립트 작성	1274
ETL 입력 PySpark	1287
Scala에서 ETL	1509
기능 및 최적화	1592
AWS Glue for Ray	1829
자습서: Ray 스크립트 작성	1829
AWS Glue for Ray에서 Ray Core 및 Ray Data 사용	1835
파일 및 Python 라이브러리 제공	1836
데이터에 연결	1841
SDK로 작업하기 AWS	1843

AWS Glue API	1845
보안	1867
- 데이터 유형 -	1867
DataCatalogEncryptionSettings	1868
EncryptionAtRest	1868
ConnectionPasswordEncryption	1868
EncryptionConfiguration	1869
S3Encryption	1870
CloudWatchEncryption	1870
JobBookmarksEncryption	1870
SecurityConfiguration	1871
GluePolicy	1871
- 작업 -	1872
GetDataCatalogEncryptionSettings (get_data_catalog_encryption_settings)	1872
PutDataCatalogEncryptionSettings (put_data_catalog_encryption_settings)	1873
PutResourcePolicy(put_resource_policy)	1874
GetResourcePolicy(get_resource_policy)	1875
DeleteResourcePolicy(delete_resource_policy)	1876
CreateSecurityConfiguration (create_security_configuration)	1877
DeleteSecurityConfiguration (delete_security_configuration)	1877
GetSecurityConfiguration (get_security_configuration)	1878
GetSecurityConfigurations (get_security_configurations)	1879
GetResourcePolicies(get_resource_policies)	1879
카탈로그	1880
데이터베이스 수	1881
표	1890
파티션	1928
연결	1953
사용자 정의 함수	1971
Athena 카탈로그 가져오기	1978
테이블 옵티마이저	1980
- 데이터 유형 -	1980
TableOptimizer	1980
TableOptimizerConfiguration	1981
TableOptimizerRun	1981
RunMetrics	1982

BatchGetTableOptimizerEntry	1982
BatchTableOptimizer	1983
BatchGetTableOptimizerError	1983
- 작업 -	1984
GetTableOptimizer (get_table_optimizer)	1984
BatchGetTableOptimizer (batch_get_table_optimizer)	1985
ListTableOptimizerRuns (목록_테이블_옵티마이저_실행)	1986
CreateTableOptimizer (테이블_옵티마이저 생성하기)	1988
DeleteTableOptimizer (테이블_옵티마이저 삭제)	1989
UpdateTableOptimizer (업데이트_테이블_옵티마이저)	1989
크롤러 및 분류자	1990
분류자	1991
크롤러	2005
열 통계값	2031
스케줄러	2038
ETL 스크립트 자동 생성	2041
- 데이터 유형 -	2041
CodeGenNode	2041
CodeGenNodeArg	2042
CodeGenEdge	2042
위치	2043
CatalogEntry	2043
MappingEntry	2044
- 작업 -	2044
CreateScript(create_script)	2045
GetDataflowGraph(get_dataflow_graph)	2045
GetMapping(get_mapping)	2046
GetPlan(get_plan)	2047
시각적 작업 API	2048
- 데이터 유형 -	2048
CodeGenConfigurationNode	2052
JDBC ConnectorOptions	2058
StreamingDataPreviewOptions	2059
AthenaConnectorSource	2060
JDBC ConnectorSource	2061
SparkConnectorSource	2061

CatalogSource	2062
MySQL CatalogSource	2063
PostgreSQL CatalogSource	2063
오라클 SQL CatalogSource	2063
마이크로소프트 SQL ServerCatalogSource	2064
CatalogKinesisSource	2064
DirectKinesisSource	2065
KinesisStreamingSourceOptions	2066
CatalogKafkaSource	2068
DirectKafkaSource	2069
KafkaStreamingSourceOptions	2069
RedshiftSource	2072
AmazonRedshiftSource	2072
AmazonRedshiftNodeData	2073
AmazonRedshiftAdvancedOption	2075
옵션	2075
S3 CatalogSource	2076
S3 SourceAdditionalOptions	2076
S3 CsvSource	2077
DirectJDBCSource	2079
S3 DirectSourceAdditionalOptions	2080
S3 JsonSource	2080
S3 ParquetSource	2082
S3 DeltaSource	2083
S3 CatalogDeltaSource	2084
CatalogDeltaSource	2084
S3 HudiSource	2085
S3 CatalogHudiSource	2086
CatalogHudiSource	2086
DynamoDB CatalogSource	2087
RelationalCatalogSource	2087
JDBC ConnectorTarget	2088
SparkConnectorTarget	2089
BasicCatalogTarget	2089
MySQL CatalogTarget	2090
PostgreSQL CatalogTarget	2090

오라클 SQL CatalogTarget	2091
마이크로소프트 SQL ServerCatalogTarget	2091
RedshiftTarget	2092
AmazonRedshiftTarget	2093
UpsertRedshiftTargetOptions	2093
S3 CatalogTarget	2093
S3 GlueParquetTarget	2094
CatalogSchemaChangePolicy	2095
S3 DirectTarget	2095
S3 HudiCatalogTarget	2096
S3 HudiDirectTarget	2097
S3 DeltaCatalogTarget	2098
S3 DeltaDirectTarget	2099
DirectSchemaChangePolicy	2100
ApplyMapping	2100
Mapping	2100
SelectFields	2101
DropFields	2102
RenameField	2102
Spigot	2103
조인	2103
JoinColumn	2104
SplitFields	2104
SelectFromCollection	2105
FillMissingValues	2105
Filter	2106
FilterExpression	2106
FilterValue	2107
CustomCode	2107
SparkSQL	2108
SqlAlias	2108
DropNullFields	2109
NullCheckBoxList	2109
NullValueField	2110
데이터 형식	2110
병합	2110

Union	2111
PIIDetection	2111
Aggregate	2112
DropDuplicates	2113
GovernedCatalogTarget	2113
GovernedCatalogSource	2114
AggregateOperation	2115
GlueSchema	2115
GlueStudioSchemaColumn	2115
GlueStudioColumn	2116
DynamicTransform	2116
TransformConfigParameter	2117
EvaluateDataQuality	2118
DQ ResultsPublishingOptions	2119
DQ StopJobOnFailureOptions	2119
EvaluateDataQualityMultiFrame	2119
레시피	2120
RecipeReference	2121
SnowflakeNodeData	2121
SnowflakeSource	2123
SnowflakeTarget	2124
ConnectorDataSource	2124
ConnectorDataTarget	2125
작업	2126
작업	2126
작업 실행	2151
트리거	2168
대화형 세션	2182
- 데이터 유형 -	2182
세션	2182
SessionCommand	2184
문	2185
StatementOutput	2186
StatementOutputData	2186
ConnectionsList	2186
- 작업 -	2187

CreateSession (생성_세션)	2187
StopSession (스톱_세션)	2190
DeleteSession (세션 삭제)	2191
GetSession (get_session)	2192
ListSessions (목록_세션)	2193
RunStatement (실행_명령문)	2194
CancelStatement (취소_진술)	2195
GetStatement (get_statement)	2196
ListStatements (목록_명령문)	2196
DevEndpoints	2197
- 데이터 유형 -	2198
DevEndpoint	2198
DevEndpointCustomLibraries	2201
- 작업 -	2202
CreateDevEndpoint(create_dev_endpoint)	2202
UpdateDevEndpoint(update_dev_endpoint)	2208
DeleteDevEndpoint(delete_dev_endpoint)	2209
GetDevEndpoint(get_dev_endpoint)	2210
GetDevEndpoints(get_dev_endpoints)	2210
BatchGetDevEndpoints(batch_get_dev_endpoints)	2211
ListDevEndpoints(list_dev_endpoints)	2212
Schema Registry	2213
- 데이터 유형 -	2213
RegistryId	2214
RegistryListItem	2214
MetadataInfo	2215
OtherMetadataValueListItem	2215
SchemaListItem	2216
SchemaVersionListItem	2217
MetadataKeyValuePair	2217
SchemaVersionErrorItem	2218
ErrorDetails	2218
SchemaVersionNumber	2218
Schemald	2219
- 작업 -	2219
CreateRegistry (레지스트리 생성)	2220

CreateSchema (생성_스키마)	2221
GetSchema (갯_스키마)	2225
ListSchemaVersions (목록_스키마_버전)	2227
GetSchemaVersion (get_스키마_버전)	2228
GetSchemaVersionsDiff (get_스키마_버전_diff)	2230
ListRegistries (리스트_레지스트리)	2231
ListSchemas (리스트_스키마)	2231
RegisterSchemaVersion (레지스터_스키마_버전)	2232
UpdateSchema (업데이트_스키마)	2234
CheckSchemaVersionValidity (스키마_버전_유효성 확인)	2235
UpdateRegistry (업데이트_레지스트리)	2236
GetSchemaByDefinition (get_schema_by_definition)	2237
GetRegistry (get_registry)	2238
PutSchemaVersionMetadata (put_스키마_버전_메타데이터)	2239
QuerySchemaVersionMetadata (쿼리_스키마_버전_메타데이터)	2241
RemoveSchemaVersionMetadata (스키마_버전_메타데이터 제거)	2242
DeleteRegistry (레지스트리 삭제)	2244
DeleteSchema (삭제_스키마)	2245
DeleteSchemaVersions (스키마_버전 삭제)	2246
워크플로	2247
- 데이터 유형 -	2247
JobNodeDetails	2248
CrawlerNodeDetails	2248
TriggerNodeDetails	2248
Crawl	2248
노드	2249
Edge	2250
워크플로	2250
WorkflowGraph	2251
WorkflowRun	2252
WorkflowRunStatistics	2253
StartingEventBatchCondition	2254
블루프린트	2254
BlueprintDetails	2255
LastActiveDefinition	2256
BlueprintRun	2256

- 작업 -	2257
CreateWorkflow(create_workflow)	2258
UpdateWorkflow(update_workflow)	2259
DeleteWorkflow(delete_workflow)	2261
GetWorkflow(get_workflow)	2261
ListWorkflows(list_workflows)	2262
BatchGetWorkflows(batch_get_workflows)	2263
GetWorkflowRun(get_workflow_run)	2263
GetWorkflowRuns(get_workflow_runs)	2264
GetWorkflowRunProperties(get_workflow_run_properties)	2265
PutWorkflowRunProperties(put_workflow_run_properties)	2266
CreateBlueprint(create_blueprint)	2267
UpdateBlueprint(update_blueprint)	2268
DeleteBlueprint(delete_blueprint)	2269
ListBlueprints(list_blueprints)	2270
BatchGetBlueprints(batch_get_blueprints)	2270
StartBlueprintRun(start_blueprint_run)	2271
GetBlueprintRun(get_blueprint_run)	2272
GetBlueprintRuns(get_blueprint_runs)	2273
StartWorkflowRun(start_workflow_run)	2274
StopWorkflowRun(stop_workflow_run)	2275
ResumeWorkflowRun(resume_workflow_run)	2275
사용 프로필	2277
- 데이터 유형 -	2277
ProfileConfiguration	2277
ConfigurationObject	2277
UsageProfileDefinition	2278
- 작업 -	2279
CreateUsageProfile (사용_프로필 생성)	2279
GetUsageProfile (사용량_프로필 가져오기)	2280
UpdateUsageProfile (업데이트_사용_프로필)	2281
DeleteUsageProfile (사용_프로필 삭제)	2282
ListUsageProfiles (목록_사용_프로필)	2282
기계 학습	2283
- 데이터 유형 -	2283
TransformParameters	2284

EvaluationMetrics	2284
MLTransform	2285
FindMatchesParameters	2288
FindMatchesMetrics	2289
ConfusionMatrix	2290
GlueTable	2290
TaskRun	2291
TransformFilterCriteria	2292
TransformSortCriteria	2293
TaskRunFilterCriteria	2294
TaskRunSortCriteria	2294
TaskRunProperties	2295
FindMatchesTaskRunProperties	2295
ImportLabelsTaskRunProperties	2296
ExportLabelsTaskRunProperties	2296
LabelingSetGenerationTaskRunProperties	2296
SchemaColumn	2296
TransformEncryption	2297
MLUserDataEncryption	2297
ColumnImportance	2298
- 작업 -	2298
CreateMLTransform(create_ml_transform)	2299
UpdateMLTransform(update_ml_transform)	2302
DeleteMLTransform(delete_ml_transform)	2304
GetMLTransform(get_ml_transform)	2305
GetMLTransforms(get_ml_transforms)	2308
ListMLTransforms(list_ml_transforms)	2309
StartMLEvaluationTaskRun(start_ml_evaluation_task_run)	2310
StartMLLabelingSetGenerationTaskRun(start_ml_labeling_set_generation_task_run)	2311
GetMLTaskRun(get_ml_task_run)	2312
GetMLTaskRuns(get_ml_task_runs)	2314
CancelMLTaskRun(cancel_ml_task_run)	2315
StartExportLabelsTaskRun(start_export_labels_task_run)	2316
StartImportLabelsTaskRun(start_import_labels_task_run)	2317
데이터 품질	2318
- 데이터 유형 -	2318

DataSource	2319
DataQualityRulesetListDetails	2319
DataQualityTargetTable	2320
DataQualityRulesetEvaluationRunDescription	2320
DataQualityRulesetEvaluationRunFilter	2321
DataQualityEvaluationRunAdditionalRunOptions	2321
DataQualityRuleRecommendationRunDescription	2322
DataQualityRuleRecommendationRunFilter	2322
DataQualityResult	2323
DataQualityAnalyzerResult	2324
DataQualityObservation	2325
MetricBasedObservation	2325
DataQualityMetricValues	2325
DataQualityRuleResult	2326
DataQualityResultDescription	2327
DataQualityResultFilterCriteria	2327
DataQualityRulesetFilterCriteria	2328
- 작업 -	2328
StartDataQualityRulesetEvaluationRun (시작_데이터_품질_규칙 세트_평가_실행)	2329
CancelDataQualityRulesetEvaluationRun (취소_데이터_품질_규칙 세트_평가_실행)	2331
GetDataQualityRulesetEvaluationRun (get_data_quality_rule set_valuation_run)	2331
ListDataQualityRulesetEvaluationRuns (목록_데이터_품질_규칙 세트_평가_실행)	2333
StartDataQualityRuleRecommendationRun (시작_데이터_품질_규칙_권장_실행)	2334
CancelDataQualityRuleRecommendationRun (취소_데이터_품질_규칙_권장_실행)	2335
GetDataQualityRuleRecommendationRun (get_data_quality_Rule_Recommendation_run)	2336
ListDataQualityRuleRecommendationRuns (목록_데이터_품질_규칙_권장_실행)	2338
GetDataQualityResult (데이터_품질_결과 가져오기)	2338
BatchGetDataQualityResult (배치_가져오기_데이터_품질_결과)	2340
ListDataQualityResults (목록_데이터_품질_결과)	2341
CreateDataQualityRuleset (데이터_품질_규칙 세트 생성하기)	2342
DeleteDataQualityRuleset (데이터_품질_규칙 세트 삭제)	2343
GetDataQualityRuleset (get_data_quality_rule set)	2344
ListDataQualityRulesets (목록_데이터_품질_규칙 세트)	2345
UpdateDataQualityRuleset (업데이트_데이터_품질_규칙 세트)	2346
민감한 데이터	2347
- 데이터 유형 -	2347

CustomEntityType	2347
- 작업 -	2348
CreateCustomEntityType(create_custom_entity_type)	2348
DeleteCustomEntityType(delete_custom_entity_type)	2350
GetCustomEntityType(get_custom_entity_type)	2350
BatchGetCustomEntityTypes(batch_get_custom_entity_types)	2351
ListCustomEntityTypes(list_custom_entity_types)	2352
API 태그 지정	2353
- 데이터 유형 -	2353
태그	2353
- 작업 -	2353
TagResource(tag_resource)	2354
UntagResource(untag_resource)	2355
GetTags(get_tags)	2355
공통 데이터 형식	2356
태그	2356
DecimalNumber	2357
ErrorDetail	2357
PropertyPredicate	2357
ResourceUri	2358
ColumnStatistics	2358
ColumnStatisticsError	2359
ColumnError	2359
ColumnStatisticsData	2359
BooleanColumnStatisticsData	2360
DateColumnStatisticsData	2361
DecimalColumnStatisticsData	2361
DoubleColumnStatisticsData	2362
LongColumnStatisticsData	2362
StringColumnStatisticsData	2363
BinaryColumnStatisticsData	2363
문자열 패턴	2363
예외	2365
AccessDeniedException	2365
AlreadyExistsException	2366
ConcurrentModificationException	2366

ConcurrentRunsExceededException	2366
CrawlerNotRunningException	2366
CrawlerRunningException	2367
CrawlerStoppingException	2367
EntityNotFoundException	2367
FederationSourceException	2368
FederationSourceRetryableException	2368
GlueEncryptionException	2368
IdempotentParameterMismatchException	2369
IllegalWorkflowStateException	2369
InternalServiceException	2369
InvalidExecutionEngineException	2369
InvalidInputException	2370
InvalidStateException	2370
InvalidTaskStatusTransitionException	2370
JobDefinitionErrorException	2370
JobRunInTerminalStateException	2371
JobRunInvalidStateTransitionException	2371
JobRunNotInTerminalStateException	2371
LateRunnerException	2372
NoScheduleException	2372
OperationTimeoutException	2372
ResourceNotReadyException	2372
ResourceNumberLimitExceededException	2373
SchedulerNotRunningException	2373
SchedulerRunningException	2373
SchedulerTransitioningException	2373
UnrecognizedRunnerException	2374
ValidationException	2374
VersionMismatchException	2374
AWS Glue API 코드 예제	2375
작업	2383
CreateCrawler	2383
CreateJob	2396
DeleteCrawler	2407
DeleteDatabase	2413

DeleteJob	2419
DeleteTable	2425
GetCrawler	2429
GetDatabase	2438
GetDatabases	2447
GetJob	2450
GetJobRun	2452
GetJobRuns	2459
GetTables	2468
ListJobs	2479
StartCrawler	2486
StartJobRun	2495
시나리오	2505
크롤러 및 작업 시작하기	2505
보안	2617
데이터 보호	2617
저장 중 암호화	2618
전송 중 암호화	2635
FIPS 규정 준수	2636
키 관리	2636
기타 AWS 서비스에서 AWS Glue 종속성	2636
개발 엔드포인트	2637
자격 증명 및 액세스 관리	2637
고객	2638
ID를 통한 인증	2639
정책을 사용한 액세스 관리	2642
AWS Glue가 IAM과 함께 작동하는 방식	2644
AWS Glue에 대한 IAM 권한 구성	2652
AWS Glue 액세스 제어 정책 예제	2683
AWS 관리형 정책	2708
리소스 ARN	2714
교차 계정 액세스 권한 부여	2721
문제 해결	2728
로그 및 모니터링	2730
규정 준수 확인	2731
복원력	2732

인프라 보안	2732
VPC 엔드포인트(AWS PrivateLink)	2733
공유 Amazon VPC	2735
AWS Glue 문제 해결	2736
AWS Glue 문제 해결 정보 모으기	2736
Spark 오류 해결	2737
오류: 사용 가능하지 않는 리소스	2738
오류: VPC에서 서브넷 ID용 S3 엔드포인트 또는 NAT 게이트웨이를 찾을 수 없습니다.	2738
오류: 보안 그룹의 인바운드 규칙이 필요합니다.	2738
오류: 보안 그룹의 아웃바운드 규칙이 필요합니다.	2739
오류: 전달된 역할에 AWS Glue 서비스에 대한 역할 수입 권한을 부여해야 하므로 작업 실행 이 실패했습니다.	2739
오류: DescribeVpcEndpoints 작업이 승인되지 않았습니니다. VPC ID vpc-id를 검증할 수 없습 니다.	2739
오류: DescribeRouteTables 작업이 승인되지 않았습니니다. 서브넷 ID를 확인할 수 없습니니다. VPC ID의 서브넷 ID: vpc-id	2739
오류: ec2 호출 실패: DescribeSubnets	2739
오류: ec2 호출 실패: DescribeSecurityGroups	2740
오류: AZ용 서브넷을 찾을 수 없습니다.	2740
오류: JDBC 대상으로 작성할 경우의 작업 실행 예외	2740
오류: Amazon S3: 객체의 스토리지 클래스에 대해 작업이 유효하지 않습니다.	2741
오류: Amazon S3 시간 제한	2741
오류: Amazon S3 액세스가 거부되었습니다.	2741
오류: Amazon S3 액세스 키 ID가 없습니다.	2741
오류: s3a:// URI를 사용해 Amazon S3에 액세스할 때 작업 실행이 실패합니다.	2742
오류: Amazon S3 서비스 토큰이 만료되었습니다.	2744
오류: 네트워크 인터페이스용 프라이빗 DNS를 찾을 수 없습니다.	2744
오류: 개발 엔드포인트 프로비저닝이 실패했습니다.	2744
오류: 노트북 서버 생성이 실패했습니다.	2744
오류: 로컬 노트북 시작이 실패했습니다.	2745
오류: 크롤러 실행이 실패했습니다.	2745
오류: 파티션이 업데이트되지 않았습니니다.	2745
오류: 버전 불일치로 인해 작업 북마크 업데이트 실패	2746
오류: 작업 북마크가 사용 설정된 경우 작업이 데이터를 재처리합니다.	2746
오류: 의 VPC 간 파일오버 동작 AWS Glue	2747
크롤러가 Lake Formation 보안 인증을 사용할 때 발생하는 크롤러 오류 문제 해결	2748

Ray 오류 해결	2750
Ray 작업 로그 검사	2750
Ray 작업 오류 해결	2751
AWS Glue 기계 학습 예외 사항	2753
CancelMLTaskRunActivity	2753
CreateMLTaskRunActivity	2753
DeleteMLTransformActivity	2754
GetMLTaskRunActivity	2755
GetMLTaskRunsActivity	2755
GetMLTransformActivity	2755
GetMLTransformsActivity	2755
GetSaveLocationForTransformArtifactActivity	2756
GetTaskRunArtifactActivity	2756
PublishMLTransformModelActivity	2757
PullLatestMLTransformModelActivity	2757
PutJobMetadataForMLTransformActivity	2758
StartExportLabelsTaskRunActivity	2758
StartImportLabelsTaskRunActivity	2758
StartMLEvaluationTaskRunActivity	2759
StartMLLabelingSetGenerationTaskRunActivity	2760
UpdateMLTransformActivity	2761
AWS Glue 할당량	2762
AWS Glue 퍼포먼스 개선	2763
작업 유형에 맞는 튜닝 전략	2763
Spark 성능 개선	2763
푸시다운을 사용한 읽기 최적화	2764
Amazon S3에 저장된 파일에서 조건자 푸시다운	2764
JDBC 소스를 사용할 때 푸시다운	2765
AWS Glue에서 푸시다운에 대한 참고 및 제한 사항	2768
AWS Glue에 Auto Scaling 사용	2768
요구 사항	2769
AWS Glue Studio에서 Auto Scaling 사용	983
AWS CLI 또는 SDK를 사용하여 Auto Scaling 사용 설정	984
Amazon CloudWatch 지표로 Auto Scaling 모니터링	2771
Spark UI로 Auto Scaling 모니터링	2772
Auto Scaling 작업 실행 DPU 사용량 모니터링	2772

제한 사항	2773
제한된 실행을 통한 워크로드 분할	2773
워크로드 분할 사용	2773
작업을 자동으로 실행하도록 AWS Glue 트리거 설정	2775
알려진 문제	2776
작업 간 데이터 액세스 방지	2776
문서 기록	2779
이전 업데이트	2824
AWS 용어집	2826
.....	mmdcccxxvii

AWS Glue이란 무엇인가요?

AWS Glue는 분석 사용자가 여러 소스의 데이터를 쉽게 검색, 준비, 이동, 통합할 수 있도록 하는 서버리스 데이터 통합 서비스입니다. 분석, 기계 학습 및 애플리케이션 개발에 사용할 수 있습니다. 또한 작성, 작업 실행, 비즈니스 워크플로 구현을 위한 추가 생산성 및 데이터 운영 도구도 포함됩니다.

AWS Glue를 사용하면 70개 이상의 다양한 데이터 소스를 검색하여 연결하고 중앙 집중식 데이터 카탈로그에서 데이터를 관리할 수 있습니다. 추출, 변환, 로드(ETL) 파이프라인을 시각적으로 생성, 실행, 모니터링하여 데이터 레이크에 데이터를 로드할 수 있습니다. 또한 Amazon Athena, Amazon EMR, Amazon Redshift Spectrum을 사용하여 카탈로그화된 데이터를 즉시 검색하고 쿼리할 수 있습니다.

AWS Glue는 주요 데이터 통합 기능을 단일 서비스로 통합합니다. 여기에는 데이터 검색, 최신 ETL, 정제, 변환, 중앙 집중식 카탈로그화가 포함됩니다. 또한 서버리스이므로 관리할 인프라가 없습니다. ETL, ELT, 스트리밍과 같은 모든 워크로드를 하나의 서비스에서 유연하게 지원하므로 AWS Glue는 다양한 워크로드 및 사용자 유형에 걸쳐 사용자를 지원합니다.

또한 AWS Glue를 사용하면 아키텍처 전반에 걸쳐 데이터를 쉽게 통합할 수 있습니다. AWS 분석 서비스 및 Amazon S3 데이터 레이크와 통합됩니다. AWS Glue는 개발자에서 비즈니스 사용자에게 이르기까지 모든 사용자가 사용하기 쉬운 통합 인터페이스 및 작업 작성 도구를 보유하고 있으며 다양한 기술 세트에 대한 맞춤형 솔루션을 제공합니다.

필요에 따라 확장할 수 있는 기능을 통해 AWS Glue는 데이터의 가치를 극대화하는 고부가가치 활동에 집중할 수 있도록 도와줍니다. 모든 데이터 크기에 맞게 확장되며 모든 데이터 유형 및 스키마 변형을 지원합니다. 민첩성을 높이고 비용을 최적화하기 위해 AWS Glue는 기본 제공 고가용성 및 사용한 만큼 지불하는 청구서를 제공합니다.

요금 정보는 [AWS Glue 요금](#)을 참조하세요.

AWS Glue Studio

AWS Glue Studio는 AWS Glue에서 데이터 통합 작업을 쉽게 생성, 실행, 모니터링할 수 있는 그래픽 인터페이스입니다. 데이터 변환 워크플로를 시각적으로 구성하고 AWS Glue의 Apache Spark 기반 서버리스 ETL 엔진에서 원활하게 실행할 수 있습니다.

AWS Glue Studio를 사용하면 데이터를 수집, 변환, 정리하는 작업을 생성하고 관리할 수 있습니다. 또한 AWS Glue Studio를 사용하여 작업 스크립트 문제를 해결하고 작업 스크립트를 편집할 수 있습니다.

주제

- [AWS Glue 기능](#)

- [AWS Glue의 혁신에 대해 알아보기](#)
- [AWS Glue 시작하기](#)
- [AWS Glue 액세스](#)
- [관련 서비스](#)

AWS Glue 기능

AWS Glue의 기능은 크게 세 가지 범주로 분류됩니다.

- 데이터 검색 및 구성
- 분석을 위한 데이터 변환, 준비, 정리
- 데이터 파이프라인 구축 및 모니터링

데이터 검색 및 구성

- 여러 데이터 스토어에 걸친 통합 및 검색 - AWS의 모든 데이터를 카탈로그로 분류하여 여러 데이터 소스 및 싱크에서 저장, 인덱싱, 검색할 수 있습니다.
- 데이터 자동 검색 - AWS Glue 크롤러를 사용하여 스키마 정보를 자동으로 추론하고 AWS Glue Data Catalog에 통합합니다.
- 스키마 및 권한 관리 - 데이터베이스와 테이블에 대한 액세스를 검증하고 제어합니다.
- 다양한 데이터 소스에 연결 - 데이터 레이크를 구축하기 위해 AWS Glue 연결을 사용하여 온프레미스와 AWS 모두에서 여러 데이터 소스를 활용합니다.

분석을 위한 데이터 변환, 준비, 정리

- 작업 캔버스 인터페이스를 사용하여 데이터를 시각적으로 변환 - 시각적 작업 편집기에서 ETL 프로세스를 정의하고 코드를 자동으로 생성하여 데이터를 추출, 변환, 로드할 수 있습니다.
- 간단한 작업 스케줄링으로 복잡한 ETL 파이프라인을 구축 - 일정에 따라 또는 요청 시 또는 이벤트를 기반으로 AWS Glue 작업을 호출합니다.
- 전송 중인 스트리밍 데이터 정리 및 변환 - 데이터를 지속적으로 사용하고 전송 중에 데이터를 정리 및 변환할 수 있습니다. 이렇게 하면 대상 데이터 스토어에서 몇 초 만에 분석할 수 있습니다.
- 기본 제공 기계 학습으로 데이터 중복 제거 및 정제 - FindMatches 기능을 사용하여 기계 학습 전문가가 아니더라도 분석을 위해 데이터를 정리하고 준비할 수 있습니다. 이 기능은 중복 데이터를 제거하고 서로 불완전하게 일치하는 레코드를 찾습니다.

- 기본 제공 작업 노트북 - AWS Glue 작업 노트북은 AWS Glue에서 최소한의 설정으로 서버리스 노트북을 제공하므로 빠르게 시작할 수 있습니다.
- ETL 코드 편집, 디버그, 테스트 - AWS Glue 대화형 세션을 통해 대화형 방식으로 데이터를 탐색하고 준비할 수 있습니다. 선택한 IDE 또는 노트북을 사용하여 대화형 방식으로 데이터를 탐색, 실험, 처리할 수 있습니다.
- 민감한 데이터의 정의, 탐지, 문제 해결 - AWS Glue 민감한 데이터 탐지를 통해 데이터 파이프라인과 데이터 레이크에서 민감한 데이터를 정의, 식별, 처리할 수 있습니다.

데이터 파이프라인 구축 및 모니터링

- 워크로드에 따라 자동 확장 - 워크로드에 따라 리소스를 동적으로 확장 및 축소할 수 있습니다. 이렇게 하면 필요한 경우에만 작업자를 작업에 할당합니다.
- 이벤트 기반 트리거로 작업 자동화 - 크롤러를 시작하거나 이벤트 기반 트리거를 사용하여 AWS Glue 작업을 수행하고 종속 작업 및 크롤러 체인을 설계합니다.
- 작업 실행 및 모니터링 - 원하는 엔진(Spark 또는 Ray)에서 AWS Glue 작업을 실행합니다. 자동 모니터링 도구, AWS Glue 작업 실행 인사이트, AWS CloudTrail 등을 사용하여 모니터링합니다. Apache Spark UI를 사용하여 Spark 지원 작업의 모니터링을 개선합니다.
- ETL 및 통합 활동을 위한 워크플로 정의 - 여러 크롤러, 작업, 트리거에 대한 ETL 및 통합 활동을 위한 워크플로를 정의합니다.

AWS Glue의 혁신에 대해 알아보기

AWS Glue의 최신 혁신에 대해 알아보고 고객이 조직 전체에서 셀프 서비스 데이터 준비를 지원하는 데 AWS Glue를 사용하는 방법을 알아보세요.

고객이 기존 설정 이상으로 AWS Glue 규모를 조정하는 방법과 작업 모니터링 및 성능을 위해 AWS Glue를 구성하는 방법도 알아보세요.

AWS Glue 시작하기

다음 단원부터 시작하는 것이 좋습니다.

- [AWS Glue 사용 개요](#)
- [AWS Glue개념](#)
- [AWS Glue에 대한 IAM 권한 설정](#)

- [AWS Glue Data Catalog 시작하기](#)
- [AWS Glue의 작업 작성](#)
- [AWS Glue 대화형 세션 시작하기](#)
- [AWS Glue에서 오케스트레이션](#)

AWS Glue 액세스

다음 인터페이스 중 하나를 사용하여 AWS Glue 작업을 생성하고, 액세스하고, 관리할 수 있습니다.

- AWS Glue 콘솔 - AWS Glue 작업을 생성, 확인 및 관리할 수 있는 웹 인터페이스를 제공합니다. 콘솔에 액세스하려면 [AWS Glue](#) 단원을 참조하십시오.
- AWS Glue Studio - AWS Glue 작업을 시각적으로 만들고 편집할 수 있는 그래픽 인터페이스를 제공합니다. 자세한 내용은 [AWS Glue Studio란 무엇인가요?](#)를 참조하세요.
- AWS CLI 참조의 AWS Glue 섹션 - AWS Glue에서 사용할 수 있는 AWS CLI 명령을 제공합니다. 자세한 내용은 [AWS Glue에 대한 AWS CLI 참조](#)를 참조하세요.
- AWS Glue API - 개발자를 위한 전체 API 레퍼런스를 제공합니다. 자세한 내용은 [AWS Glue API](#)를 참조하세요.

관련 서비스

AWS Glue의 사용자는 또한 다음을 사용합니다.

- [AWS Lake Formation](#) - AWS Glue Data Catalog의 리소스에 대한 세분화된 액세스 제어를 제공하는 권한 부여 계층인 서비스입니다.
- [AWS Glue DataBrew](#) - 코드를 작성하지 않고 데이터를 정리하고 정규화하는 데 사용할 수 있는 시각적 데이터 준비 도구입니다.

AWS Glue: 작동 방식

AWS Glue는 다른 AWS 서비스를 통해 추출, 변환, 로드 작업을 오케스트레이션하여 데이터 웨어하우스 및 데이터 레이크를 빌드하고 출력 스트림을 생성합니다. AWS Glue는 API 작업을 호출하여 데이터를 변환하고, 런타임 로그를 생성하고, 작업 로직을 저장하고, 작업 실행을 모니터링하는 데 도움이 되는 알림을 생성합니다. AWS Glue 콘솔은 이런 서비스를 관리형 애플리케이션으로 연결하여 ETL 작업을 생성하고 모니터링하는 데 집중할 수 있습니다. 콘솔은 사용자를 대신하여 관리 및 작업 개발 작업을 수행합니다. 자격 증명 및 기타 속성을 AWS Glue에 제공하여 데이터 원본에 액세스하고 데이터 대상에 작성합니다.

AWS Glue는 워크로드가 실행되도록 필요한 리소스를 프로비저닝 및 관리를 담당합니다. ETL 도구를 사용하기 위해서 인프라를 생성할 필요가 없는 이유는 AWS Glue가 대신하기 때문입니다. 리소스가 필요한 경우, 시작 시간을 줄이기 위해 AWS Glue는 인스턴스의 준비된 풀에서 인스턴스를 사용하여 워크로드를 실행합니다.

AWS Glue를 통해 데이터 카탈로그에서 테이블 정의를 사용하여 작업을 생성합니다. 작업은 변환을 수행하는 프로그램 로직을 포함하는 스크립트로 구성됩니다. 트리거를 사용하여 특정 이벤트의 일정에 따라 혹은 결과에 따라 작업을 시작합니다. 대상 데이터가 있는 곳과 어떤 소스 데이터가 대상으로 채우는지 결정합니다. 입력을 통해 AWS Glue는 소스에서 대상으로 데이터를 변환할 때 필요한 코드를 생성합니다. AWS Glue 콘솔 또는 API로 스크립트를 제공하여 데이터를 실행합니다.

데이터 소스 및 대상

AWS Glue for Spark를 사용하면 다음을 포함한 여러 시스템 및 데이터베이스에서 데이터를 읽고 쓸 수 있습니다.

- Amazon S3
- Amazon DynamoDB
- Amazon Redshift
- Amazon Relational Database Service(Amazon RDS)
- 타사 JDBC에서 액세스할 수 있는 데이터베이스
- MongoDB 및 Amazon DocumentDB(MongoDB와 호환)
- 기타 마켓플레이스 커넥터 및 Apache Spark 플러그인

데이터 스트림

AWS Glue for Spark는 다음 시스템에서 데이터를 스트리밍할 수 있습니다.

- Amazon Kinesis Data Streams
- Apache Kafka

AWS Glue는 여러 AWS 리전에서 사용할 수 있습니다. 자세한 내용은 Amazon Web Services 일반 참조의 [AWS Regions and endpoints](#)를 참조하세요.

주제

- [격리 중인 서버리스 ETL 작업 실행](#)
- [AWS Glue 개념](#)
- [AWS Glue 구성 요소](#)
- [AWS Glue for Spark 및 AWS Glue for Ray](#)
- [반구조화된 스키마를 AWS Glue가 포함된 관계형 스키마로 변환하기](#)
- [AWS Glue 타입 시스템](#)

격리 중인 서버리스 ETL 작업 실행

AWS Glue는 선택한 엔진(Spark 또는 Ray)을 사용하여 서버리스 환경에서 ETL 작업을 실행합니다. AWS Glue는 이런 작업을 가상 리소스에서 실행하여 자체 서비스 계정을 프로비저닝하고 관리합니다.

AWS Glue는 다음과 같이 하도록 설계되어 있습니다.

- 사용자 지점 데이터 분리
- 전송 및 저장 시 사용자 데이터 보호
- 사용자 요청에 대한 응답으로만 임시 범위가 축소된 자격 증명 또는 계정의 IAM 역할에 대한 사용자 동의에 따라 사용자 데이터에 액세스합니다.

ETL 작업을 프로비저닝하는 중에 가상 사설 클라우드(VPC)에 입력 데이터 원본 및 출력 데이터 대상을 제공합니다. 또한, 데이터 원본 및 대상에 필요한 IAM 역할, VPC ID, 서브넷 ID 및 보안 그룹을 제공합니다. 각 튜플(사용자 계정 ID, IAM 역할, 서브넷 ID 및 보안 그룹)을 위해 AWS Glue는 AWS Glue 서비스 계정 내 다른 모든 환경의 네트워크 및 관리 수준이 격리된 새로운 환경을 생성합니다.

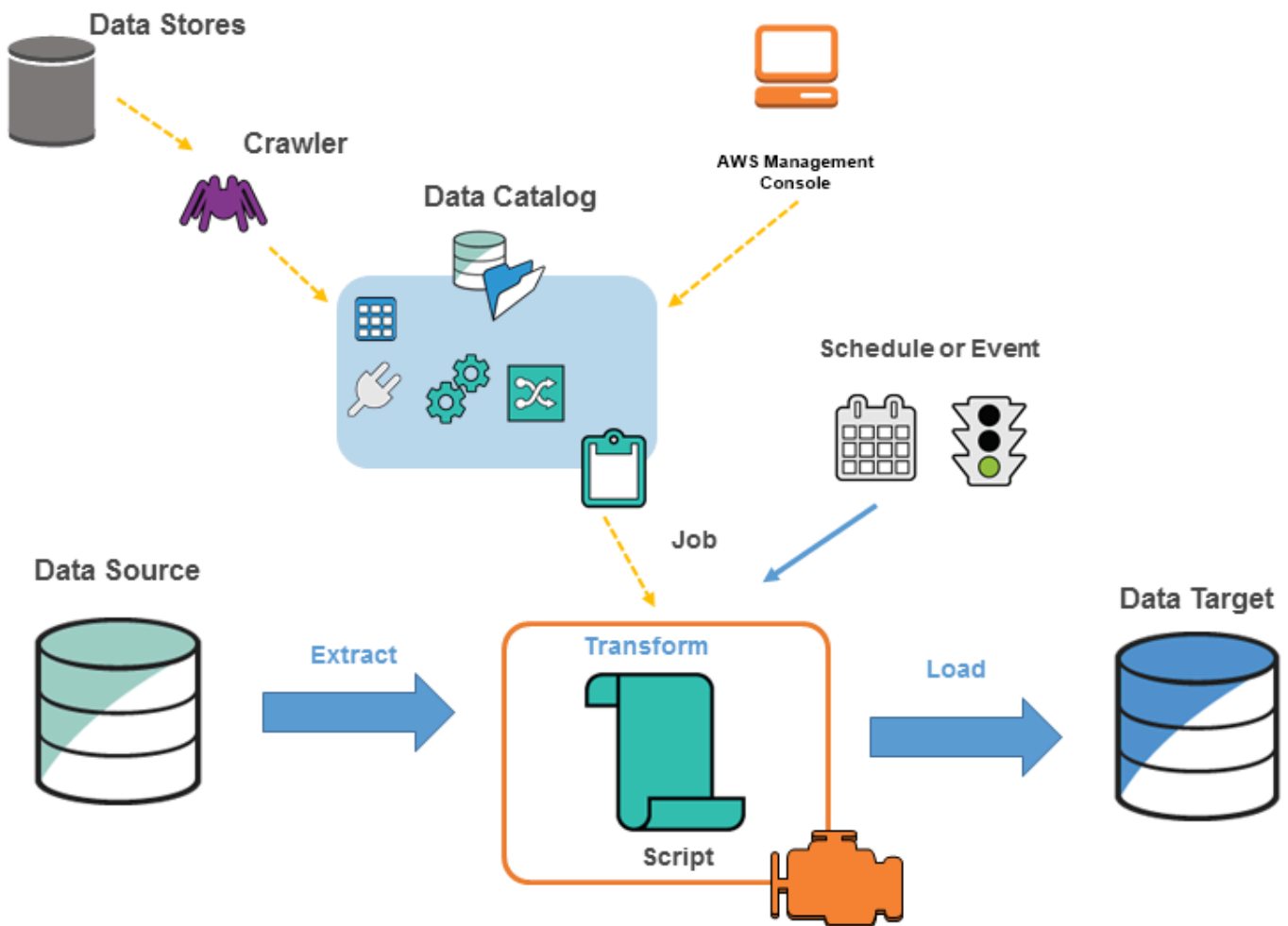
AWS Glue는 프라이빗 IP 주소를 사용하여 탄력적 네트워크 인터페이스를 서브넷에 생성합니다. 작업은 탄력적 네트워크 인터페이스를 사용하여 데이터 소스와 데이터 대상에 액세스합니다. VPC 및 네트워크 정책은 다음 한 가지를 제외한 작업 실행 환경 내부 및 외부에 있는 트래픽을 관리합니다. 한 가지 예외 사항은 AWS Glue 라이브러리를 직접 호출하여 프록시 트래픽에서 AWS Glue API 작업을 AWS

Glue VPC를 통해 가능하게 한 경우입니다. 모든 AWS Glue API 호출은 로깅됩니다. 따라서 데이터 소유자는 감사 로그를 계정으로 전달하는 [AWS CloudTrail](#)를 활성화하여 API 액세스를 감사할 수 있습니다.

ETL 작업을 실행하는 AWS Glue 관리형 환경은 뒤이어 다른 AWS 서비스가 따르는 동일한 보안 과정을 통해 보호됩니다. 사례 및 공유 보안 책임에 대한 개요는 [AWS 보안 프로세스 소개](#) 백서를 참조하세요.

AWS Glue 개념

다음 다이어그램은 AWS Glue 환경의 아키텍처를 나타냅니다.



AWS Glue에서 작업을 정의하여 데이터 원본에서 데이터 대상으로 데이터를 추출, 변환, 로드하는 데 필요한 작업을 수행합니다. 일반적으로 다음 작업을 수행할 수 있습니다.

- 데이터 스토어 소스의 경우 크롤러를 정의하여 메타데이터 테이블 정의로 AWS Glue Data Catalog를 채웁니다. 데이터 스토어에서 크롤러를 포인트하면 크롤러는 Data Catalog에서 테이블 정의를 생성합니다. 스트리밍 소스의 경우 Data Catalog 테이블을 수동으로 정의하고 데이터 스트림 속성을 지정합니다.

테이블 정의에 관련해 추가하자면 AWS Glue Data Catalog는 다른 필수 메타데이터를 포함하여 ETL 작업을 정의합니다. 작업을 정의하여 데이터를 변환할 경우 이 메타데이터를 사용합니다.

- AWS Glue는 스크립트를 생성하여 데이터를 변환할 있습니다. 또는 스크립트를 AWS Glue 콘솔이나 API에 제공합니다.
- 필요시 작업하거나 지정된 trigger가 발생하면 작업을 설정하여 시작할 수 있습니다. 트리거는 시간 기반 스케줄이나 이벤트일 수 있습니다.

작업이 진행되면 스크립트는 데이터 원본에서 데이터를 추출하고 데이터를 변환한 다음 데이터 대상으로 로드합니다. 스크립트는 AWS Glue의 Apache Spark 환경에서 실행됩니다.

Important

AWS Glue에 있는 테이블과 데이터베이스는 AWS Glue Data Catalog의 객체입니다. 이 객체들은 메타데이터를 포함하지만 데이터 스토어의 데이터는 포함하지 않습니다.

CSV와 같은 텍스트 기반 데이터를 처리하려면 AWS Glue용으로 **UTF-8**에 인코딩해야 합니다. 자세한 내용은 Wikipedia의 [UTF-8](#)을 참조하세요.

AWS Glue 용어

AWS Glue에서는 여러 구성 요소의 상호 작용에 의존하여 추출, 전환, 적재(ETL) 워크플로를 생성하고 관리합니다.

AWS Glue Data Catalog

AWS Glue의 영구적 메타데이터 스토어입니다. 테이블 정의, 작업 정의 및 기타 관리 정보를 포함하여 AWS Glue 환경을 관리합니다. 각 AWS 계정에는 리전당 AWS Glue Data Catalog 하나가 있습니다.

분류자

데이터 스키마를 결정합니다. AWS Glue는 CSV, JSON, AVRO, XML 등과 같은 일반 파일 형식에 대한 분류자를 제공합니다. JDBC 연결을 사용한 일반 관계형 데이터베이스 관리 시스템을 위한 분류자를 제공합니다. a grok 패턴을 사용하거나 XML 문서에 행 태그를 지정하여 자체 분류자를 작성할 수 있습니다.

연결

특정 데이터 스토어에 연결하는 데 필요한 속성을 포함하는 Data Catalog 객체입니다.

크롤러

데이터 스토어(소스 또는 대상)에 연결하는 프로그램은 분류자의 우선 순위 지정 목록을 통해 데이터의 스키마를 결정한 다음 AWS Glue Data Catalog에 메타데이터 테이블을 생성합니다.

데이터베이스

논리 그룹으로 구성된 일련의 연결된 Data Catalog 테이블 정의입니다.

데이터 스토어, 데이터 원본, 데이터 대상

데이터 스토어는 데이터를 영구적으로 저장하기 위한 리포지토리입니다. 예를 들면 Amazon S3 버킷과 관계형 데이터베이스가 있습니다. 데이터 원본은 프로세스 또는 변환의 입력값으로 사용되는 데이터 스토어입니다. 데이터 대상은 프로세스 또는 변환에서 쓰기를 수행하는 대상 데이터 스토어입니다.

개발 엔드포인트

AWS Glue ETL 스크립트를 개발하고 테스트하는 데 사용할 수 있는 환경입니다.

동적 프레임

구조 및 배열과 같은 중첩 데이터를 지원하는 분산 테이블입니다. 각 레코드는 자기 설명적이며 반정형 데이터가 있는 스키마 유연성을 위해 설계되었습니다. 각 레코드에는 데이터와 해당 데이터를 설명하는 스키마가 모두 포함되어 있습니다. ETL 스크립트에서 동적 프레임과 Apache Spark DataFrame을 모두 사용하여 이 두 프레임 간에 변환할 수 있습니다. 동적 프레임은 데이터 정리 및 ETL에 필요한 고급 변환 세트를 지원합니다.

작업

작업은 ETL 작업을 수행하는 데 필요한 비즈니스 로직입니다. 변환 스크립트, 데이터 원본 및 데이터 대상으로 구성됩니다. 작업은 트리거에 의해 시작되고, 트리거는 예약되거나 이벤트 기반으로 작동할 수 있습니다.

작업 성능 대시보드

AWS Glue는 ETL 작업에 대한 포괄적인 실행 대시보드를 제공합니다. 대시보드에는 특정 기간의 작업 실행에 대한 정보가 표시됩니다.

노트북 인터페이스

간편한 작업 작성 및 데이터 탐색을 위해 원클릭 설정으로 향상된 노트북 환경. 노트북과 연결이 자동으로 구성됩니다. Jupyter Notebook 기반의 노트북 인터페이스를 통해 AWS Glue 서버리스 Apache Spark ETL 인프라를 사용하여 스크립트와 워크플로를 대화형으로 개발, 디버깅, 배포할 수 있습니다. 노트북 환경에서 임시 쿼리, 데이터 분석, 시각화(예: 테이블 및 그래프)를 수행할 수도 있습니다.

Script

소스에서 데이터를 추출하고 변환하고 대상으로 로드하는 코드입니다. AWS Glue는 PySpark 또는 Scala 스크립트를 생성합니다.

표

데이터를 나타내는 메타데이터 정의입니다. 데이터가 Amazon Simple Storage Service(Amazon S3) 파일, Amazon Relational Database Service(Amazon RDS) 테이블 또는 다른 데이터 집합에 있는지에 관계없이 테이블은 데이터의 스키마를 정의합니다. AWS Glue Data Catalog의 테이블은 열 이름, 데이터 유형 정의, 파티션 정보 및 베이스 데이터 세트에 대한 기타 메타데이터로 구성됩니다. 데이터의 스키마는 AWS Glue 테이블 정의에 나타납니다. 실제 데이터는 파일 혹은 관계형 데이터베이스 테이블 어디에 있든지 기존 데이터 스토어에 남겨집니다. AWS Glue는 파일과 관계형 데이터베이스 테이블 목록을 AWS Glue Data Catalog에 작성합니다. ETL 작업을 생성할 경우 소스와 대상으로써 사용됩니다.

변환

다른 포맷으로 데이터를 바꾸는 코드 로직.

트리거

ETL 작업 시작. 트리거는 정해진 시간 혹은 이벤트 기반으로 정의할 수 있습니다.

시각적 작업 편집기

시각적 작업 편집기는 AWS Glue에서 추출, 전환, 적재(ETL) 작업을 쉽게 생성, 실행, 모니터링할 수 있게 해주는 그래픽 인터페이스입니다. 데이터 변환 워크플로를 시각적으로 구성하고 AWS Glue의 Apache Spark 기반 서버리스 ETL 엔진에서 원활하게 실행하며 작업의 각 단계에서 스키마와 데이터 결과를 검사할 수 있습니다.

작업자

AWS Glue를 사용하면 ETL 작업을 실행하는 데 걸리는 시간 요금만 결제하면 됩니다. 관리할 리소스와 사전 투자 비용이 없으며 시작 시간 또는 종료 시간에 대한 요금이 부과되지 않습니다. ETL 작업을 실행하는 데 사용된 데이터 처리 단위(DPU) 수에 따라 시간당 요금이 청구됩니다. 단일 데이터 처리 단위(DPU)를 작업자라고도 합니다. AWS Glue에는 작업 대기 시간 및 비용 요구 사항을 충족하는 구성을 선택하는 데 도움이 되는 세 가지 작업자 유형이 있습니다. 작업자는 표준, G.1X, G.2X 및 G.025X 구성으로 제공됩니다.

AWS Glue 구성 요소

AWS Glue는 콘솔 및 API 작업을 제공하여 워크로드를 추출, 변환 및 로드(ETL)하는 것을 설정하고 관리합니다. 특정 언어의 SDK 및 AWS Command Line Interface(AWS CLI)를 통해 API 작업을 사용할 수 있습니다. AWS CLI 사용에 대한 자세한 내용은 [AWS CLI Command Reference](#)를 참조하세요.

AWS Glue는 AWS Glue Data Catalog를 사용하여 데이터 원본, 변환 및 대상의 메타데이터를 저장합니다. Data Catalog는 Apache Hive Metastore의 드롭인 교체물입니다. AWS Glue Jobs system는 데이터의 ETL 작업을 정의, 일정 관리 및 실행하기 위한 관리된 인프라를 제공합니다. AWS Glue API에 대한 자세한 내용은 [AWS Glue API](#)을 참조하십시오.

AWS Glue 콘솔

AWS Glue 콘솔을 사용하여 ETL 워크플로우를 정의하고 관리할 수 있습니다. 콘솔은 AWS Glue Data Catalog 및 AWS Glue Jobs system에서 몇 가지 API 작업을 호출하여 다음 작업을 실행합니다.

- 작업, 테이블, 크롤러 및 연결과 같은 AWS Glue 객체를 정의합니다.
- 크롤러를 실행하는 일정
- 작업 트리거를 위한 이벤트 또는 일정을 정의합니다.
- AWS Glue 객체 목록을 검색하고 필터링합니다.
- 변환 스크립트를 편집합니다.

AWS Glue Data Catalog

AWS Glue Data Catalog는 AWS 클라우드에 있는 영구적 기술 메타데이터 스토어입니다.

각 AWS 계정에는 AWS 리전당 AWS Glue Data Catalog가 하나씩 있습니다. 각 데이터 카탈로그는 데이터베이스로 구성된 확장성이 뛰어난 테이블 모음입니다. 테이블은 Amazon RDS, Apache Hadoop 분산 파일 시스템, Amazon OpenSearch Service 등의 소스에 저장된 정형 또는 반정형 데이터 모음을 메타데이터로 표현한 것입니다. AWS Glue Data Catalog는 일정한 리포지토리를 제공합니다. 그러면 전혀 다른 시스템들이 메타데이터를 저장하고 탐색하여 데이터 사일로에서 데이터를 추적할 수 있습니다. 그런 다음 메타데이터를 사용하여 다양한 애플리케이션에서 일관된 방식으로 해당 데이터를 쿼리하고 변환할 수 있습니다.

데이터 카탈로그를 AWS Identity and Access Management 정책 및 Lake Formation과 함께 사용하여 테이블 및 데이터베이스에 대한 액세스를 제어합니다. 이렇게 하면 기업의 여러 그룹이 더 광범위한 조직에 데이터를 안전하게 게시하면서 매우 세분화된 방식으로 민감한 정보를 보호할 수 있습니다.

데이터 카탈로그는 CloudTrail 및 Lake Formation과 함께 스키마 변경 추적 및 데이터 액세스 제어와 함께 포괄적인 감사 및 거버넌스 기능도 제공합니다. 이렇게 하면 데이터가 부적절하게 수정되거나 실수로 공유되지 않도록 보장할 수 있습니다.

AWS Glue Data Catalog 보안 및 감사에 대한 자세한 내용은 다음을 참조하세요.

- AWS Lake Formation – 자세한 내용은 AWS Lake Formation 개발자 안내서의 [AWS Lake Formation이란 무엇입니까?](#)를 참조하세요.
- CloudTrail – 자세한 내용은 AWS CloudTrail 사용 설명서의 [CloudTrail이란 무엇입니까?](#)를 참조하세요.

다음은 AWS Glue Data Catalog를 사용하는 기타 AWS 서비스 및 오픈 소스 프로젝트입니다.

- Amazon Athena – 자세한 내용은 Amazon Athena 사용 설명서의 [테이블, 데이터베이스, 데이터 카탈로그 이해](#)를 참조하세요.
- Amazon Redshift Spectrum – 자세한 내용은 Amazon Redshift 데이터베이스 개발자 안내서의 [Amazon Redshift Spectrum을 사용하여 외부 데이터 쿼리](#)를 참조하세요.
- Amazon EMR – 자세한 내용은 Amazon EMR 관리 안내서의 [AWS Glue Data Catalog에 대한 Amazon EMR 액세스에 리소스 기반 정책 사용](#)을 참조하세요.
- Apache Hive 메타스토어용 AWS Glue Data Catalog 클라이언트 – 이 GitHub 프로젝트에 대한 자세한 내용은 [AWS Glue Data Catalog Client for Apache Hive Metastore](#)를 참조하세요.

AWS Glue 크롤러 및 분류자

AWS Glue는 모든 종류의 리포지토리에서 데이터를 스캔하고 분류하며, 스키마 정보를 추출하고, AWS Glue Data Catalog에서 자동적으로 메타데이터를 저장하는 크롤러를 설정할 수 있습니다. AWS Glue Data Catalog를 사용하여 ETL 작업을 관리할 수 있습니다.

크롤러 및 분류자 설정 방법에 대한 자세한 내용은 [크롤러를 사용하여 데이터 카탈로그 채우기](#) 단원을 참조하십시오. AWS Glue API를 사용하는 크롤러 및 분류자를 프로그래밍하는 방법에 대한 자세한 내용은 [크롤러 및 분류자 API](#) 단원을 참조하십시오.

AWS Glue ETL 연산

Data Catalog에서 메타데이터를 사용하여 AWS Glue는 다양한 ETL 작업을 실행할 때 사용하고 수정할 수 있는 AWS Glue 확장자를 붙여 Scala 또는 PySpark(Apache Spark용 Python API) 스크립트를 자동으로 생성할 수 있습니다. 예를 들어, 원 데이터를 추출, 정리 및 변환한 다음 결과가 쿼리되고 분석될 수 있는 다른 리포지토리에 결과를 저장할 수 있습니다. 이러한 스크립트는 CSV 파일을 관계 형식으로 변환하고 Amazon Redshift에 저장할 수 있습니다.

AWS Glue ETL 기능을 사용하는 방법에 대한 자세한 내용은 [Spark 스크립트 프로그래밍](#) 단원을 참조하십시오.

AWS Glue의 스트리밍 ETL

AWS Glue를 사용하면 지속적으로 실행되는 작업을 사용하여 스트리밍 데이터에 대해 ETL 작업을 수행할 수 있습니다. AWS Glue 스트리밍 ETL은 Apache Spark Structured Streaming 엔진을 기반으로 하며 Amazon Kinesis Data Streams, Apache Kafka 및 Amazon Managed Streaming for Apache Kafka (Amazon MSK)에서 스트림을 수집할 수 있습니다. 스트리밍 ETL은 스트리밍 데이터를 정리하고 변환하여 Amazon S3 또는 JDBC 데이터 스토어에 로드할 수 있습니다. AWS Glue의 스트리밍 ETL을 사용하여 IoT 스트림, 클릭스트림 및 네트워크 로그와 같은 이벤트 데이터를 처리합니다.

스트리밍 데이터 원본의 스키마를 알고 있는 경우 Data Catalog 테이블에서 지정할 수 있습니다. 그렇지 않은 경우 스트리밍 ETL 작업에서 스키마 감지를 사용할 수 있습니다. 그런 다음 작업은 들어오는 데이터에서 스키마를 자동으로 결정합니다.

스트리밍 ETL 작업은 AWS Glue 기본 제공 변환과 Apache Spark Structured Streaming에 대한 기본 변환을 모두 사용할 수 있습니다. 자세한 내용은 Apache Spark 웹 사이트의 [Operations on streaming DataFrames/Datasets](#)를 참조하세요.

자세한 내용은 [the section called “스트리밍 ETL 작업”](#) 단원을 참조하십시오.

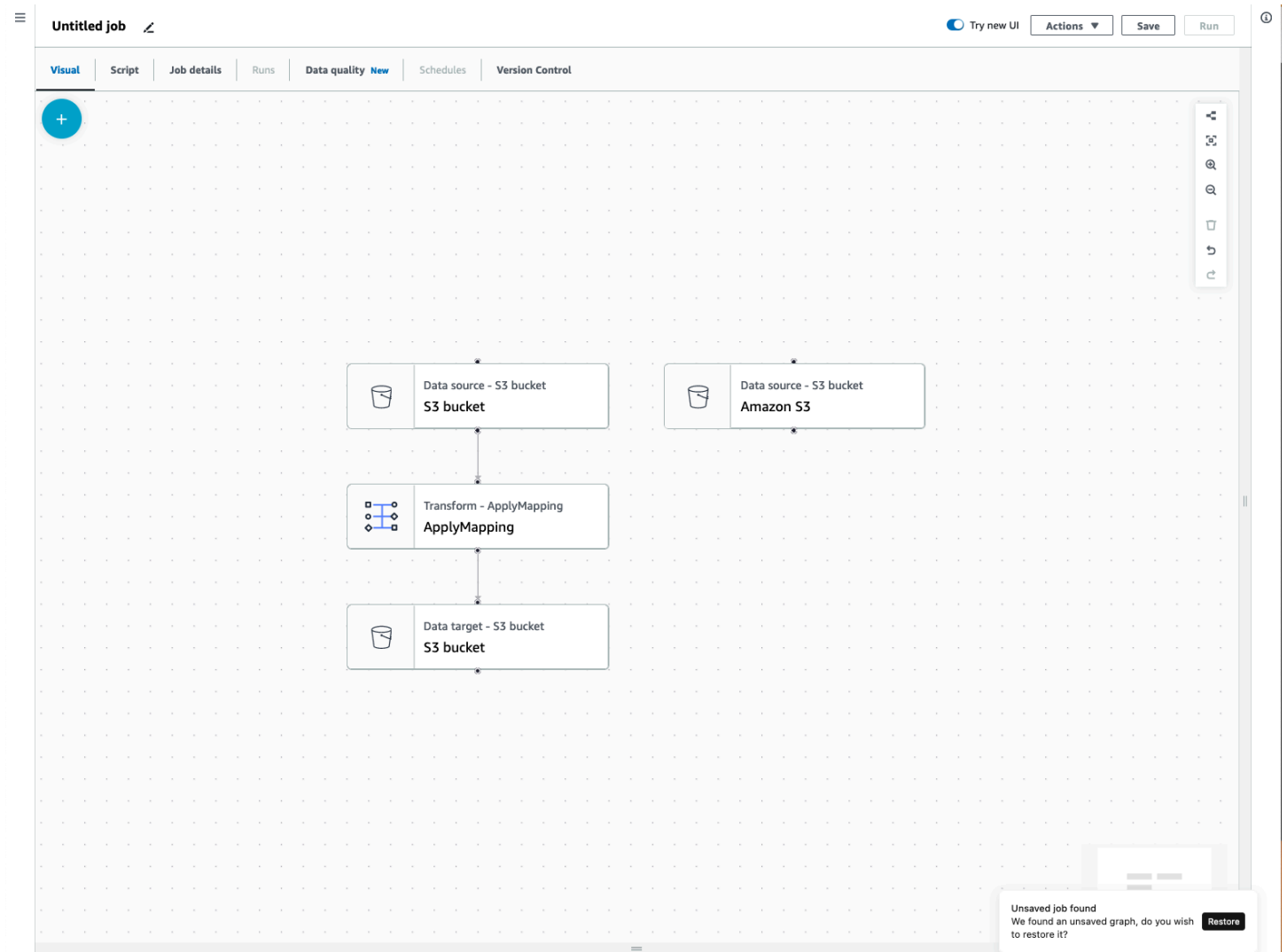
AWS Glue 작업 시스템

AWS Glue Jobs system는 관리된 인프라를 사용하여 ETL 워크플로우를 관리할 수 있습니다. 사용하는 스크립트를 통해 자동적으로 데이터를 다른 위치로 추출, 변환 및 전송할 수 있는 작업을 AWS Glue에 생성할 수 있습니다. 작업은 일정이 정해지고 모을 수 있고 새로운 데이터가 도착하는 것과 같은 이벤트에 의해 촉발될 수 있습니다.

AWS Glue Jobs system 사용에 대한 자세한 내용은 [AWS Glue 모니터링](#) 단원을 참조하십시오. AWS Glue Jobs system API를 사용하여 프로그래밍하는 것에 대한 자세한 내용은 [작업 API](#) 단원을 참조하십시오.

시각적 ETL 구성 요소

AWS Glue에서는 사용자가 조작할 수 있는 시각적 캔버스를 통해 ETL 작업을 생성할 수 있습니다.



ETL 작업 메뉴

캔버스 상단의 메뉴 옵션을 통해 작업에 대한 다양한 보기 및 구성 세부 정보에 액세스할 수 있습니다.

- 시각적 - 시각적 작업 편집기 캔버스입니다. 여기서 노드를 추가하여 작업을 생성할 수 있습니다.
- 스크립트 - ETL 작업의 스크립트 표현입니다. AWS Glue에서는 작업의 시각적 표현을 기반으로 스크립트를 생성합니다. 스크립트를 편집하거나 다운로드할 수도 있습니다.

Note

스크립트를 편집하려는 경우 작업 작성 환경이 스크립트 전용 모드로 영구적으로 전환됩니다. 이후에는 더 이상 시각적 편집기를 사용하여 작업을 편집할 수 없습니다. 스크립트 편집을 선택하기 전에 모든 작업 소스, 변환 및 대상을 추가하고 시각적 편집기를 사용하여 필요한 모든 변경 사항을 수행해야 합니다.

- 작업 세부 정보 - 작업 세부 정보 탭에서는 작업 속성을 설정하여 작업을 구성할 수 있습니다. 여기에는 기본 속성(예: 작업 이름 및 설명, IAM 역할, 작업 유형, AWS Glue 버전, 언어, 작업자 유형, 작업자 유형, 작업자 수, 작업 북마크, Flex 실행, 사용 중지 횟수, 작업 제한 시간 초과)과 함께, 연결, 라이브러리, 작업 파라미터, 태그와 같은 고급 속성도 있습니다.
- 실행 - 작업을 실행한 후 이 탭에 액세스하여 이전 작업 실행을 볼 수 있습니다.
- 데이터 품질 - 데이터 품질에서는 데이터 자산의 품질을 평가하고 모니터링합니다. 이 탭에서 데이터 품질을 사용하는 방법에 대해 자세히 알아보고 작업에 데이터 품질 변환을 추가할 수 있습니다.
- 일정 - 예약한 작업이 이 탭에 표시됩니다. 이 작업에 연결된 일정이 없는 경우 이 탭에는 액세스할 수 없습니다.
- 버전 제어 - 작업을 Git 리포지토리로 구성하여 작업에서 Git를 사용할 수 있습니다.

시각적 ETL 패널

캔버스에서 작업하는 경우 노드를 구성하거나 데이터를 미리 보고 출력 스키마를 보는 데 도움이 되는 여러 패널을 사용할 수 있습니다.

- 속성 - 캔버스에서 노드를 선택하면 속성 패널이 나타납니다.
- 데이터 미리 보기 - 데이터 미리 보기 패널에서는 데이터 출력의 미리 보기를 제공하므로 작업을 실행하기 전에 결정을 내리고 출력을 살펴볼 수 있습니다.
- 출력 스키마 - 출력 스키마 탭을 사용하면 변환 노드의 스키마를 보고 편집할 수 있습니다.

패널 크기 조정

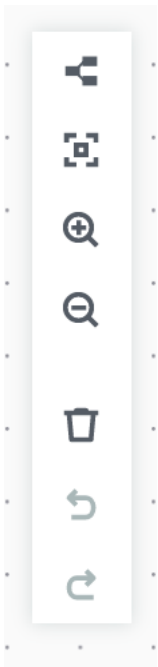
데이터 미리 보기 및 출력 스키마 탭이 포함된 하단 패널과 화면 오른쪽에 있는 속성 패널은 패널 가장 자리를 클릭한 상태로 좌우 또는 위아래로 끌어 크기를 조정할 수 있습니다.

- 속성 패널 - 화면 오른쪽의 캔버스 가장자리를 클릭한 상태로 끌어 속성 패널의 크기를 조정한 후 왼쪽으로 끌어 너비를 확장합니다. 기본적으로 패널은 축소된 상태이며, 노드를 선택하면 속성 패널이 기본 크기로 열립니다.
- 데이터 미리 보기 및 출력 스키마 패널 - 화면 하단에서 캔버스 하단 가장자리를 클릭한 상태로 끌어 하단 패널의 크기를 조정한 후 위로 끌어 높이를 확장합니다. 기본적으로 패널은 축소된 상태이며, 노드를 선택하면 하단 패널이 기본 크기로 열립니다.

작업 캔버스

시각적 ETL 캔버스에서 직접 노드를 추가, 제거, 이동/재정렬할 수 있습니다. 작업 캔버스는 데이터 소스로 시작하고 데이터 대상으로 끝낼 수 있는 완전한 기능을 갖춘 ETL 작업을 생성할 수 있는 작업 공간과 같습니다.

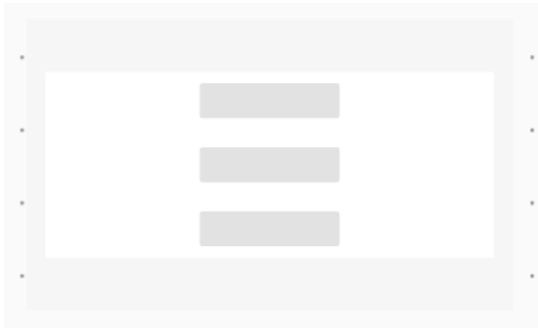
캔버스에서 노드에 대한 작업을 수행할 때 확대 및 축소, 노드 제거, 노드 간 연결 설정 또는 편집, 작업 흐름 방향 변경, 작업 실행 취소 또는 재실행 등을 도와주는 도구 모음이 있습니다.



부동 도구 모음은 캔버스의 오른쪽 상단 크기로 고정되어 있으며 다음과 같이 작업을 수행하는 여러 이미지를 포함합니다.

- 레이아웃 아이콘 - 도구 모음의 첫 번째 아이콘은 레이아웃 아이콘입니다. 기본적으로 시각적 작업의 방향은 위에서 아래 방향입니다. 노드를 왼쪽에서 오른쪽으로 가로로 정렬하여 시각적 작업의 방향을 재정렬할 수 있습니다. 레이아웃 아이콘을 다시 클릭하면 방향이 위에서 아래로 다시 바뀝니다.
- 가운데 재정렬 아이콘 - 가운데 재정렬 아이콘은 캔버스 보기를 가운데에 배치하여 변경합니다. 대규모 작업의 경우 이 아이콘을 사용하여 가운데 위치로 돌아갈 수 있습니다.
- 확대 아이콘 - 확대 아이콘은 캔버스의 노드 크기를 확대합니다.
- 축소 아이콘 - 축소 아이콘은 캔버스의 노드 크기를 축소합니다.
- 휴지통 아이콘 - 휴지통 아이콘은 시각적 작업에서 노드를 제거합니다. 먼저 노드를 선택해야 합니다.
- 실행 취소 아이콘 - 실행 취소 아이콘은 시각적 작업에서 수행한 마지막 작업을 되돌립니다.
- 다시 실행 아이콘 - 다시 실행 아이콘은 시각적 작업에서 수행한 마지막 작업을 반복합니다.

미니 맵 사용



리소스 패널

리소스 패널에는 사용자가 사용할 수 있는 모든 데이터 소스, 변환 작업 및 연결이 포함되어 있습니다. '+' 아이콘을 클릭하여 캔버스에서 리소스 패널을 엽니다. 그러면 리소스 패널이 열립니다.

리소스 패널을 닫으려면 리소스 패널의 오른쪽 상단에 있는 X를 클릭합니다. 그러면 패널을 다시 열 준비가 될 때까지 패널을 숨깁니다.

+ Add nodes ✕

▼ Popular transforms & data

Amazon S3 (source)	SQL Query
Amazon Redshift (source)	Aggregate
Change Schema	Custom Transform
Join	Filter

Transforms | **Data**

▼ Sources

- AWS Glue Data Catalog**
AWS Glue Data Catalog table as the data source.
- Amazon S3**
JSON, CSV, or Parquet files stored in S3.
- Amazon Kinesis**
Read from an Amazon Kinesis Data Stream.
- Apache Kafka**
Read from an Apache Kafka or Amazon MSK topic.
- Relational DB**
AWS Glue Data Catalog table with a relational database as the data source.
- Amazon Redshift**
Read your data from Amazon Redshift.
- MySQL**
AWS Glue Data Catalog table with MySQL as the data source.
- PostgreSQL**
AWS Glue Data Catalog table with PostgreSQL as the data source.
- Oracle SQL**
AWS Glue Data Catalog table with Oracle SQL as the data source.
- Microsoft SQL Server**
AWS Glue Data Catalog table with SQL Server as the data source.
- Amazon DynamoDB**
AWS Glue Data Catalog table with DynamoDB as the data source.
- Snowflake**
Read your data from Snowflake.

자주 사용하는 변환 및 데이터

패널 상단에는 자주 사용하는 변환 및 데이터 컬렉션이 있습니다. 이 노드는 일반적으로 AWS Glue에서 사용됩니다. 하나를 선택하여 캔버스에 추가합니다. 자주 사용하는 변환 및 데이터 제목 옆의 삼각형을 클릭하여 자주 사용하는 변환 및 데이터를 숨길 수도 있습니다.

자주 사용하는 변환 및 데이터 섹션 아래에서 변환 및 데이터 소스 노드를 검색할 수 있습니다. 내용을 입력하면 결과가 표시됩니다. 검색 쿼리를 길게 입력할수록 결과 목록이 작아집니다. 검색 결과는 노드 이름 및/또는 설명으로 채워집니다. 노드를 선택하여 캔버스에 추가합니다.

변환 및 데이터

노드를 변환 및 데이터로 구성하는 두 개의 탭이 있습니다.

변환 - 변환 탭을 선택하면 사용 가능한 모든 변환을 선택할 수 있습니다. 변환을 선택하여 캔버스에 추가합니다. 변환 목록 하단에서 변환 추가를 선택할 수도 있습니다. 그러면 [사용자 지정 시각적 변환](#) 생성에 관한 설명서의 새 페이지가 열립니다. 단계를 따라 직접 변환을 생성할 수 있습니다. 그러면 변환이 사용 가능한 변환 목록에 표시됩니다.

데이터 - 데이터 탭에는 소스 및 대상의 모든 노드가 포함되어 있습니다. 소스 또는 대상 제목 옆의 삼각형을 클릭하여 소스 및 대상을 숨길 수 있습니다. 삼각형을 다시 클릭하면 숨긴 소스 및 대상을 다시 표시할 수 있습니다. 소스 또는 대상 노드를 선택하여 캔버스에 추가합니다. 연결 관리를 선택하여 새 연결을 추가할 수도 있습니다. 그러면 콘솔에서 커넥터 페이지가 열립니다.

AWS Glue for Spark 및 AWS Glue for Ray

AWS Glue on Apache Spark(AWS Glue ETL)에서 PySpark를 사용하여 대규모 데이터를 처리하는 Python 코드를 작성할 수 있습니다. Spark는 이 문제에 대한 친숙한 솔루션이지만 Python 중심 배경을 갖춘 데이터 엔지니어는 이 전환이 직관적이지 않다고 생각할 수 있습니다. Spark DataFrame 모델은 이 모델의 기반이 되는 Scala 언어와 Java 런타임을 반영하는 “Pythonic”이 원활하지 않습니다.

AWS Glue에서는 Python 셸 작업을 사용하여 기본 Python 데이터 통합을 실행할 수 있습니다. 이러한 작업은 단일 Amazon EC2 인스턴스에서 실행되며 해당 인스턴스의 용량에 따라 제한됩니다. 따라서 처리할 수 있는 데이터의 처리량이 제한되고 빅 데이터를 처리할 때 유지 관리 비용이 많이 듭니다.

AWS Glue for Ray를 사용하면 Spark 학습에 많은 투자를 하지 않고도 Python 워크로드를 확장할 수 있습니다. Ray가 더 잘 작동하는 특정 시나리오를 활용할 수 있습니다. 선택권을 제공하여 Spark와 Ray의 장점을 모두 활용할 수 있습니다.

AWS Glue ETL과 AWS Glue for Ray는 기본적으로 다르므로 서로 다른 기능을 지원합니다. 지원되는 기능은 설명서를 참조하세요.

AWS Glue for Ray란 무엇인가요?

Ray는 Python을 중심으로 워크로드를 확장하는 데 사용될 수 있는 오픈 소스 분산 계산 프레임워크입니다. Ray에 대한 자세한 내용은 [Ray 웹사이트](#)를 참조하세요. AWS Glue Ray 작업 및 대화형 세션을 활용하면 Ray를 AWS Glue 내에서 사용할 수 있습니다.

AWS Glue for Ray를 사용하여 여러 시스템에서 병렬로 실행되는 계산을 위한 Python 스크립트를 작성할 수 있습니다. Ray 작업 및 대화형 세션에서는 pandas와 같은 친숙한 Python 라이브러리를 사용하여 워크플로를 쉽게 작성하고 실행할 수 있습니다. Ray 데이터 세트에 대한 자세한 내용은 Ray 설명서의 [Ray 데이터 세트](#)를 참조하세요. Pandas에 대한 자세한 내용은 [Pandas 웹 사이트](#)를 참조하세요.

AWS Glue for Ray를 사용하면 단 몇 줄의 코드만으로 엔터프라이즈 규모의 빅 데이터에 대해 Pandas 워크플로를 실행할 수 있습니다. AWS Glue 콘솔 또는 AWS SDK에서 Ray 작업을 생성할 수 있습니다. AWS Glue 대화형 세션을 열어 서버리스 Ray 환경에서 코드를 실행할 수도 있습니다. AWS Glue Studio의 시각적 작업은 아직 지원되지 않습니다.

AWS Glue for Ray 작업을 통해 일정에 따라 또는 Amazon EventBridge의 이벤트에 대한 응답으로 스크립트를 실행할 수 있습니다. 작업은 스크립트의 상태와 신뢰성을 이해하는 데 도움이 되는 로그 정보 및 CloudWatch의 모니터링 통계를 저장합니다. AWS Glue 작업 시스템에 대한 자세한 내용은 [the section called "Ray 작업 사용"](#) 섹션을 참조하세요.

AWS Glue for Ray 대화형 세션(평가판)을 사용하면 프로비저닝된 동일한 리소스에 대해 코드 조각을 차례로 실행할 수 있습니다. 이를 사용하여 스크립트를 효율적으로 프로토타이핑하고 개발하거나 대화형 애플리케이션을 직접 빌드할 수 있습니다. AWS Management Console에서 AWS Glue Studio 노트북의 AWS Glue 대화형 세션을 사용할 수 있습니다. 자세한 내용은 [AWS Glue Studio 및 AWS Glue 을\(를\) 사용하여 노트북 사용](#)을 참조하세요. Jupyter 커널을 통해 사용할 수도 있습니다. Jupyter 커널을 사용하면 VSCode와 같은 Jupyter Notebook을 지원하는 기존 코드 편집 도구에서 대화형 세션을 실행할 수 있습니다. 자세한 내용은 [the section called "AWS Glue Ray 인터랙티브 세션용 \(미리 보기\)"](#) 단원을 참조하십시오.

Ray는 부하에 따라 실시간으로 재구성하는 시스템 클러스터에 처리를 분산하여 Python 코드 크기 조정 작업을 자동화합니다. 따라서 특정 워크로드의 가격 대비 성능이 향상될 수 있습니다. Ray 작업에서는 auto Scaling이 AWS Glue 작업 모델에 기본적으로 제공되므로 이 기능을 최대한 활용할 수 있습니다. Ray 작업은 AWS Graviton에서 실행되므로 전반적인 가격 대비 성능이 향상됩니다.

비용 절감 이외에 기본 auto Scaling을 사용하여 클러스터 유지 관리, 조정 및 관리에 시간을 투자하지 않고도 Ray 워크로드를 실행할 수 있습니다. Pandas와 같은 친숙한 기존 오픈 소스 라이브러리와 AWS SDK for Pandas를 사용할 수 있습니다. 그러면 AWS Glue for Ray에서 개발하는 동안 반복 속도

가 향상됩니다. AWS Glue for Ray를 사용하면 비용 효율적인 데이터 통합 워크로드를 신속하게 개발하여 실행할 수 있습니다.

반구조화된 스키마를 AWS Glue가 포함된 관계형 스키마로 변환하기

일반적으로 반구조화된 데이터를 관계형 테이블로 변환하고자 할 것입니다. 개념적으로는 계층적 스키마를 관계형 스키마로 평면화하는 것입니다. AWS Glue는 이 변환을 바로 실행할 수 있습니다.

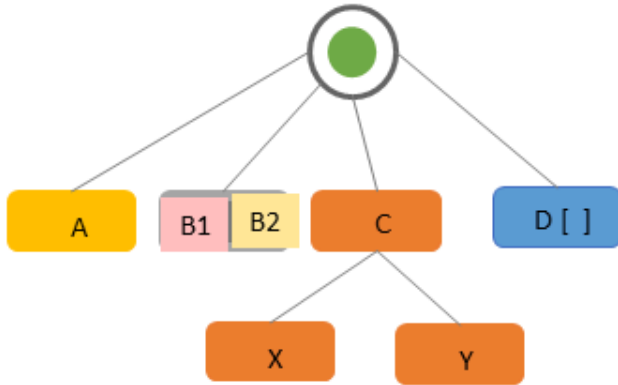
반구조화된 데이터는 일반적으로 데이터 내 개체를 식별하기 위해서 마크업을 포함합니다. 고정된 스키마없이 중첩된 데이터 구조를 가질 수 있습니다. 반구조화된 데이터에 대한 자세한 내용은 Wikipedia에서 [반구조화된 데이터](#)를 참조하십시오.

관계형 데이터는 행과 열로 조직된 테이블이 보여줍니다. 테이블간 관계는 기본 키(PK)와 외래 키(FK) 관계에 의해 표현됩니다. 자세한 내용은 Wikipedia의 [관계형 데이터베이스](#)를 참조하십시오.

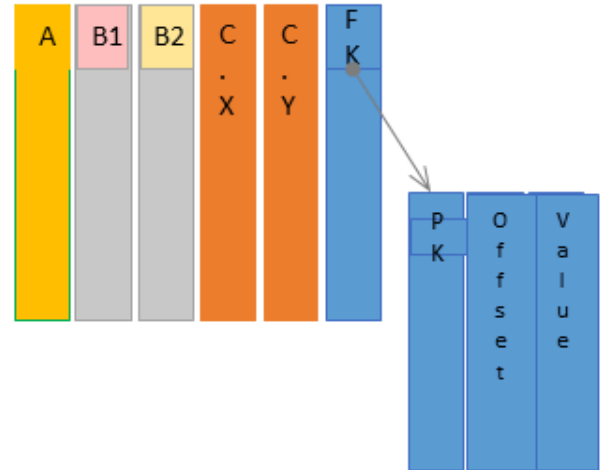
AWS Glue는 크롤러를 사용하여 반구조화된 데이터용 스키마를 유추합니다. ETL(추출, 변환 및 로드) 작업을 사용하여 데이터를 관계형 스키마로 변환합니다. 예를 들어, JSON 데이터를 Amazon Simple Storage Service(Amazon S3) 소스 파일에서 Amazon Relational Database Service (Amazon RDS) 테이블로 구문 분석해야 할 경우가 있습니다. AWS Glue가 어떻게 스키마간 차이를 다루는지 알면 변환 절차를 이해할 수 있습니다.

이 다이어그램은 AWS Glue가 어떻게 반구조화된 스키마를 관계형 스키마로 변환하는지 보여줍니다.

Semi-structured schema



Relational schema



다이어그램은 다음을 보여 줍니다.

- 단일 값 A는 관계형 열로 바로 변환됩니다.
- B1 및 B2인 값 페어는 두 개의 관계형 열로 변환됩니다.
- 하위 X 및 Y가 있는 C 구조는 두 개의 관계형 열로 변환됩니다.
- D[] 배열은 다른 관계형 테이블을 가리키는 외래 키(FK)가 있는 관계형 열로 변환합니다. 기본 키(PK)와 함께 두 번째 관계형 테이블은 배열 아이템의 오프셋과 값을 포함하는 열을 보유하고 있습니다.

AWS Glue 타입 시스템

AWS Glue는 여러 유형 시스템을 사용하여 매우 다양한 방식으로 데이터를 저장하는 데이터 시스템에 다양한 인터페이스를 제공합니다. 이 문서에서는 AWS Glue 유형 시스템 및 데이터 표준을 명확하게 설명합니다.

AWS Glue 데이터 카탈로그 유형

데이터 카탈로그는 다양한 데이터 시스템인 메타스토어에 저장된 테이블 및 필드의 레지스트리입니다. AWS Glue 크롤러 및 AWS Glue with Spark 작업과 같은 AWS Glue 구성 요소는 데이터 카탈로그에 기록할 때 필드 유형을 추적하기 위한 내부 유형 시스템을 사용하여 작성합니다. 이러한 값은 AWS Glue Console에서 테이블 스키마의 데이터 유형 열에 표시됩니다. 이 유형 시스템은 Apache Hive의 유형 시스템을 기반으로 합니다. Apache Hive 유형 시스템에 대한 자세한 내용은 Apache Hive 위키의 [유형](#)을 참조하세요. 특정 유형 및 지원에 대한 자세한 내용은 스키마 빌더의 일부로 AWS Glue 콘솔에 예제가 제공됩니다.

검증, 호환성 및 기타 용도

데이터 카탈로그는 유형 필드에 기록된 유형을 검증하지 않습니다. AWS Glue 구성 요소가 데이터 카탈로그를 읽고 쓸 때 서로 호환됩니다. AWS 또한 Glue 구성 요소는 Hive 유형과의 높은 수준의 호환성을 유지하는 것을 목표로 합니다. 그러나 AWS Glue 구성 요소는 모든 Hive 유형과의 호환성을 보장하지 않습니다. 이를 통해 데이터 카탈로그의 테이블로 작업할 때 Athena DDL과 같은 도구와의 상호 운용성이 가능합니다.

데이터 카탈로그는 유형을 검증하지 않으므로 다른 서비스에서는 데이터 카탈로그를 사용하여 Hive 유형 시스템 또는 기타 시스템을 엄격하게 준수하는 시스템을 사용하여 유형을 추적할 수 있습니다.

Spark를 사용한 AWS Glue 스크립트의 유형

AWS Glue with Spark 스크립트가 데이터셋을 해석하거나 변환할 때 스크립트에서 사용된 데이터셋을 메모리에 그대로 표시합니다. DynamicFrame DynamicFrame의 목표는 Spark DataFrame의 목표와 유사합니다. Spark가 데이터에 대한 변환을 예약하고 실행할 수 있도록 데이터셋을 모델링하는 것입니다. toDF 및 fromDF 메서드를 제공하여 DynamicFrame의 유형 표현이 DataFrame과 상호 호환되도록 보장합니다.

유형 정보가 유추되거나 DataFrame에 제공될 수 있는 경우 달리 문서화되지 않는 한 이를 유추하거나 DynamicFrame에 제공할 수 있습니다. 특정 데이터 형식에 최적화된 리더 또는 라이터를 제공하는 경우, Spark가 사용자의 데이터를 읽거나 쓸 수 있는 경우 제공된 리더 및 라이터는 문서화된 제한에

따라 데이터를 읽거나 쓸 수 있습니다. 리더 및 라이터에 관한 자세한 내용은 [the section called “데이터 포맷 옵션”](#) 섹션을 참조하세요.

선택 유형

DynamicFrames는 디스크에서 행 간에 유형이 일치하지 않을 수 있는 값인 데이터 세트의 필드를 모델링하는 메커니즘을 제공합니다. 예를 들어 필드는 특정 행에는 문자열로 저장된 숫자를, 다른 행에는 정수를 포함할 수 있습니다. 이 메커니즘은 Choice라고 하는 인 메모리 유형입니다. Choice 열을 구체적인 유형으로 해석하기 위한 ResolveChoice 메서드와 같은 변환을 제공합니다. AWS Glue ETL은 일반적인 작업 과정에서 Choice 유형을 데이터 카탈로그에 기록하지 않습니다. Choice 유형은 데이터 세트의 DynamicFrame 메모리 모델 컨텍스트에서만 존재합니다. 선택 유형 사용 예는 [the section called “데이터 준비 예”](#) 섹션을 참조하세요.

AWS Glue 크롤러 유형

크롤러는 데이터 세트에 대해 일관되고 사용 가능한 스키마를 생성한 다음 다른 AWS Glue 구성 요소 및 Athena에서 사용할 수 있도록 Data Catalog에 저장하는 것을 목표로 합니다. 크롤러는 데이터 카탈로그의 이전 섹션 [the section called “AWS Glue 데이터 카탈로그 유형”](#)에서 정의된 유형을 다룹니다. 열에 두 개 이상의 유형 값이 포함된 '선택' 유형 시나리오에서 사용 가능한 유형을 생성하기 위해 크롤러는 잠재적 유형을 모델링하는 struct 유형을 만듭니다.

AWS Glue 시작하기

다음 섹션에서는 AWS Glue 설정에 대한 정보를 제공합니다. AWS Glue 사용을 시작하기 위해 모든 설정 섹션이 필요한 것은 아닙니다. VPC 환경을 사용하여 데이터 스토어에 액세스하거나 대화형 세션을 사용하는 경우 IAM 권한, 암호화, DNS를 설정하는 데 필요한 지침을 사용할 수 있습니다.

주제

- [AWS Glue 사용 개요](#)
- [AWS Glue에 대한 IAM 권한 설정](#)
- [AWS Glue 사용 프로필 설정](#)
- [AWS Glue Data Catalog 시작하기](#)
- [데이터 스토어에 대한 네트워크 액세스 설정](#)
- [AWS Glue에서 암호화 설정](#)
- [AWS Glue의 개발에 대한 네트워킹 설정](#)

AWS Glue 사용 개요

AWS Glue로 AWS Glue Data Catalog에 메타데이터를 저장합니다. 이 메타데이터를 사용하여 데이터 원본을 변환하고 데이터 웨어하우스 또는 데이터 레이크를 로드하는 ETL 작업을 오케스트레이션할 수 있습니다. 다음 단계는 일반 작업 흐름과 AWS Glue와 작업할 때 정한 선택에 대해 설명합니다.

Note

다음 단계를 사용하거나 1~3단계를 자동으로 수행하는 워크플로를 생성할 수 있습니다. 자세한 내용은 [the section called “블루프린트 및 워크플로를 사용하여 복잡한 ETL 활동 수행”](#) 단원을 참조하십시오.

1. 테이블 정의로 AWS Glue Data Catalog를 채웁니다.

콘솔에서 영구 데이터 스토어의 경우 크롤러를 추가하여 AWS Glue Data Catalog를 채울 수 있습니다. 테이블 목록 또는 크롤러 목록에서 [Add crawler(크롤러 추가)] 마법사를 시작합니다. 크롤러에 액세스하도록 하나 이상의 스토어를 선택합니다. 일정을 생성하여 크롤러 실행 빈도수를 결정합니다. 데이터 스트림의 경우 테이블 정의를 수동으로 생성하고 스트림 속성을 정의할 수 있습니다.

데이터 스키마를 추론하는 사용자 분류자를 제공할 수도 있습니다. grok 패턴을 사용하여 사용자 지정 분류자를 생성합니다. 하지만 AWS Glue는 사용자 분류자가 데이터를 인식하지 않으면 크롤러에서 자동적으로 사용된 기본 설정 분류자를 제공합니다. 크롤러를 정의할 때 분류자를 선택할 필요가 없습니다. AWS Glue의 분류자에 대한 자세한 내용은 [AWS Glue의 크롤러로 분류자 추가](#) 단원을 참조하십시오.

몇 가지 데이터 스토어 유형을 크롤하려면 인증 및 위치 정보가 제공되는 연결이 필요합니다. 필요하다면 AWS Glue 콘솔에 필요한 정보를 제공하는 연결을 생성할 수 있습니다.

크롤러가 데이터 스토어를 읽고 데이터 정의와 이름 붙여진 테이블을 AWS Glue Data Catalog에 생성합니다. 이런 테이블은 선택에 따라 데이터베이스에 조직됩니다. 수동으로 생성한 테이블로 Data Catalog를 채울 수도 있습니다. 이 방법을 통해 스키마 및 기타 메타데이터를 제공하여 Data Catalog에 테이블 정의를 생성합니다. 이런 방법은 조금 짜증나고 오류가 쉽게 날 수 있기 때문에 크롤러가 테이블 정의를 생성하는 방법이 더 좋습니다.

테이블 정의로 AWS Glue Data Catalog를 채우는 방법에 대한 자세한 내용은 [테이블 생성](#) 단원을 참조하십시오.

2. 원본에서 대상으로의 데이터 변환을 설명하는 작업을 정의합니다.

일반적으로 다음과 같은 선택을 통해 작업을 생성합니다.

- AWS Glue Data Catalog에서 작업의 소스로 사용할 테이블을 선택합니다. 작업은 이 테이블 정의를 사용하여 데이터 원본에 액세스하고 데이터 포맷을 해석합니다.
- AWS Glue Data Catalog에서 작업의 대상으로 사용할 테이블 또는 위치를 선택합니다. 작업은 이 정보를 사용하여 데이터 스토어에 액세스합니다.
- AWS Glue에 스크립트를 생성하여 소스를 대상으로 변환하도록 지시합니다. AWS Glue는 코드를 생성하여 기본 제공 변환을 직접 호출한 다음 소스 스키마에서 대상 스키마 형식으로 데이터를 전환합니다. 이 변환은 데이터 복사, 열 이름 바꾸기 및 데이터 필터링 후 필요에 따라 데이터 변환의 작업을 실행합니다. AWS Glue 콘솔의 스크립트를 수정할 수 있습니다.

AWS Glue의 작업을 정의하는 방법에 대한 자세한 내용은 [AWS Glue Studio를 사용하여 시각적 ETL 작업 구축](#) 단원을 참조하십시오.

3. 작업을 실행하여 데이터를 변환합니다.

필요할 때 작업하거나, 트리거 유형 중 하나에 따라 시작할 수 있습니다.

- Cron 일정에 따른 트리거.

- 트리거는 이벤트 기반입니다. 예를 들어, 작업을 성공적으로 완료했다면 AWS Glue 작업을 시작할 수 있습니다.
- 필요 시 트리거는 작업을 시작합니다.

AWS Glue의 트리거에 대한 자세한 내용은 [트리거를 사용하여 작업 및 크롤러 시작](#) 단원을 참조하십시오.

4. 예정된 크롤러와 촉발된 작업을 모니터링합니다.

AWS Glue 콘솔을 사용하여 다음을 봅니다.

- 작업 실행 상세 정보 및 오류
- 크롤러 실행 상세 정보 및 오류
- AWS Glue 동작에 대한 알림

AWS Glue의 크롤러 및 작업 모니터링에 대한 자세한 내용은 [AWS Glue 모니터링](#) 단원을 참조하십시오.

AWS Glue에 대한 IAM 권한 설정

이 주제의 지침은 AWS Glue에 대해 AWS Identity and Access Management(IAM) 권한을 빠르게 설정하는 데 도움이 됩니다. 다음 작업을 완료합니다.

- IAM 자격 증명에 AWS Glue 리소스에 대한 액세스 권한을 부여합니다.
- 작업 실행, 데이터 액세스, AWS Glue Data Quality 작업 실행을 위한 서비스 역할을 생성합니다.

AWS Glue에 대한 IAM 권한을 사용자 지정하는 데 사용할 수 있는 자세한 지침은 [AWS Glue에 대한 IAM 권한 구성](#) 섹션을 참조하세요.

AWS Management Console에서 AWS Glue에 대한 IAM 권한을 설정하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 시작하기를 선택합니다.
3. AWS Glue에 대한 계정 준비에서 IAM 권한 설정을 선택합니다.
4. AWS Glue 권한을 부여하려는 IAM 자격 증명(역할 또는 사용자)을 선택합니다. AWS Glue에서는 이 자격 증명에 [AWSGlueConsoleFullAccess](#) 관리형 정책을 연결합니다. 이러한 권한을 수동으로 설정하거나 기본 서비스 역할만 설정하려는 경우 이 단계를 건너뛸 수 있습니다.

5. 다음을 선택합니다.
6. 역할 및 사용자에게 필요한 Amazon S3 액세스 수준을 선택합니다. 이 단계에서 선택한 옵션은 선택한 모든 자격 증명에 적용됩니다.
 - a. S3 위치 선택에서 액세스 권한을 부여하려는 Amazon S3 위치를 선택합니다.
 - b. 다음으로, 이전에 선택한 위치에 대해 자격 증명에 읽기 전용(권장) 또는 읽기 및 쓰기 액세스 권한을 부여할지 선택합니다. AWS Glue에서는 선택한 위치 및 읽기 또는 쓰기 권한의 조합을 기반으로 자격 증명에 권한 정책을 추가합니다.

다음 테이블에는 Amazon S3 액세스를 위해 AWS Glue에서 연결하는 권한이 나와 있습니다.

사용자 선택 항목...	AWS Glue에서 연결하는 항목...
변경 없음	해당 권한이 없습니다. AWS Glue에서는 자격 증명 권한은 변경하지 않습니다.
특정 Amazon S3 위치에 대한 액세스 권한 부여(읽기 전용)	<p>선택한 IAM 자격 증명에 포함된 인라인 정책입니다. 자세한 내용은 IAM 사용 설명서의 인라인 정책을 참조하세요.</p> <p>AWS Glue에서는 AWSGlueConsole <i><Role/User></i> InlinePolicy-read-specific-access- <i><UUID></i> 규칙을 사용하여 정책 이름을 지정합니다. 예: AWSGlueConsoleRoleInlinePolicy-read-specific-access-123456780123 .</p> <p>다음은 지정된 Amazon S3 위치에 대한 읽기 전용 액세스 권한을 부여하기 위해 AWS Glue에서 연결하는 인라인 정책의 예제입니다.</p> <pre data-bbox="911 1703 1507 1877"> { "Version": "2012-10-17", "Statement": [{ </pre>

사용자 선택 항목...	AWS Glue에서 연결하는 항목...
	<pre> "Effect": "Allow", "Action": ["s3:Get*", "s3:List*"], "Resource": ["arn:aws: s3:::DOC-EXAMPLE-BUCKET/*"] }]</pre>

사용자 선택 항목...	AWS Glue에서 연결하는 항목...
<p>특정 Amazon S3 위치에 대한 액세스 권한 부여(읽기 및 쓰기)</p>	<p>선택한 IAM 자격 증명에 포함된 인라인 정책입니다. 자세한 내용은 IAM 사용 설명서의 인라인 정책을 참조하세요.</p> <p>AWS Glue에서는 AWSGlueConsole <i><Role/User></i> InlinePolicy-read-and-write-specific-access- <i><UUID></i> 규칙을 사용하여 정책 이름을 지정합니다. 예: AWSGlueConsoleRoleInlinePolicy-read-and-write-specific-access-123456780123 .</p> <p>다음은 지정된 Amazon S3 위치에 대한 읽기 및 쓰기 액세스 권한을 부여하기 위해 AWS Glue에서 연결하는 인라인 정책의 예제입니다.</p> <pre data-bbox="917 1031 1507 1831"> { "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["s3:Get*", "s3:List*", "s3:*Object*"], "Resource": ["arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*", "arn:aws:s3:::DOC-EXAMPLE-BUCKET2/*"] }] } </pre>

사용자 선택 항목...	AWS Glue에서 연결하는 항목...
Amazon S3에 대한 전체 액세스 권한 부여 (읽기 전용)	AmazonS3ReadOnlyAccess 관리형 IAM 정책입니다. 자세한 내용은 AWS 관리형 정책: AmazonS3ReadOnlyAccess 를 참조하세요.
Amazon S3에 대한 전체 액세스 권한 부여 (읽기 및 쓰기)	AmazonS3FullAccess 관리형 IAM 정책입니다. 자세한 내용은 AWS 관리형 정책: AmazonS3FullAccess 를 참조하세요.

- 다음을 선택합니다.
- 사용자 계정의 기본 AWS Glue 서비스 역할을 선택합니다. 서비스 역할은 사용자를 대신하여 다른 AWS 서비스의 리소스에 액세스하기 위해 AWS Glue에서 사용하는 IAM 역할입니다. 자세한 내용은 [AWS Glue의 서비스 역할](#) 단원을 참조하십시오.
 - 표준 AWS Glue 서비스 역할을 선택하면 AWS Glue는 AWS 계정에서 이름이 `AWSGlueServiceRole`이고 다음 관리형 정책이 연결된 새 IAM 역할을 생성합니다. 사용자 계정에 이름이 `AWSGlueServiceRole`인 IAM 역할이 이미 있는 경우 AWS Glue는 이러한 정책을 기존 역할에 연결합니다.
 - [AWSGlueServiceRole](#)
 - [AmazonS3FullAccess](#)
 - 기존 IAM 역할을 선택하면 AWS Glue는 역할을 기본값으로 설정하지만 역할에 권한을 추가하지는 않습니다. AWS Glue에 대한 서비스 역할로 사용하도록 역할을 구성했는지 확인합니다. 자세한 내용은 [1단계: AWS Glue 서비스를 위한 IAM 정책 생성](#) 및 [2단계: AWS Glue에 대한 IAM 역할 생성](#) 단원을 참조하세요.
- 다음을 선택합니다.
- 마지막으로, 선택한 권한을 검토하고 변경 사항 적용을 선택합니다. 변경 사항을 적용하면 AWS Glue는 선택한 자격 증명에 IAM 권한을 추가합니다. 새 권한은 IAM 콘솔(<https://console.aws.amazon.com/iam/>)에서 확인하거나 수정할 수 있습니다.

이제 AWS Glue에 대한 최소 IAM 권한 설정을 완료했습니다. 프로덕션 환경에서는 사용 사례에 사용할 AWS 리소스를 확보할 수 있도록 [AWS Glue의 보안](#) 및 [AWS Glue의 ID 및 액세스 관리](#)에 익숙해지는 것이 좋습니다.

다음 단계

이제 IAM 권한을 설정했으므로 AWS Glue 사용을 시작하기 위해 다음 주제를 탐색해볼 수 있습니다.

- [Getting Started with AWS Glue in AWS Skill Builder](#)
- [AWS Glue Data Catalog 시작하기](#)

AWS Glue Studio에 대한 설정

시각적 ETL에 대한 AWS Glue을(를) 처음 사용하는 경우 이 섹션의 작업을 완료합니다.

주제

- [AWS Glue Studio 사용자에게 필요한 IAM 권한 검토](#)
- [ETL 작업에 필요한 IAM 권한 검토](#)
- [AWS Glue Studio에 대한 IAM 권한 설정](#)
- [ETL 작업에 사용할 VPC 구성](#)

AWS Glue Studio 사용자에게 필요한 IAM 권한 검토

AWS Glue Studio를 사용하려면 사용자가 다양한 AWS 리소스에 액세스할 수 있어야 합니다. 사용자는 Amazon S3 버킷, IAM 정책 및 역할, AWS Glue Data Catalog 객체를 보고 선택할 수 있어야 합니다.

AWS Glue 서비스 권한

AWS Glue Studio는 AWS Glue 서비스의 작업과 리소스를 사용합니다. AWS Glue Studio를 효과적으로 사용하려면 사용자에게 이러한 작업과 리소스에 대한 권한이 필요합니다. AWS Glue Studio 사용자에게 AWSGlueConsoleFullAccess 관리형 정책을 부여하거나 더 작은 권한 집합으로 사용자 지정 정책을 생성할 수 있습니다.

Important

보안 모범 사례에 따라 Amazon S3 버킷 및 Amazon CloudWatch 로그 그룹에 대한 액세스를 추가로 제한하는 정책을 강화하여 액세스를 제한하는 것이 좋습니다. Amazon S3 정책 예제는 [IAM 정책 작성하기: Amazon S3 버킷으로의 액세스를 보장하는 방법](#)을 참조하세요.

AWS Glue Studio에 대한 사용자 지정 IAM 정책 생성

AWS Glue Studio에 대한 더 작은 권한 집합을 포함하는 사용자 지정 정책을 생성할 수 있습니다. 이 정책에서는 객체 또는 작업의 하위 집합에 대한 권한을 부여할 수 있습니다. 사용자 지정 정책을 생성할 때는 다음 정보를 사용합니다.

AWS Glue Studio API를 사용하려면 IAM 권한 내의 작업 정책에 `glue:UseGlueStudio`를 포함합니다. `glue:UseGlueStudio`를 사용하면 시간이 지남에 따라 API에 더 많은 작업이 추가되는 경우에도 모든 AWS Glue Studio 작업을 수행할 수 있습니다.

DAG(방향성 비순환 그래프) 작업

- `CreateDag`
- `UpdateDag`
- `GetDag`
- `DeleteDag`

작업

- `SaveJob`
- `GetJob`
- `CreateJob`
- `DeleteJob`
- `GetJobs`
- `UpdateJob`

작업 실행 옵션

- `StartJobRun`
- `GetJobRuns`
- `BatchStopJobRun`
- `GetJobRun`
- `QueryJobRuns`
- `QueryJobs`
- `QueryJobRunsAggregated`

스키마 작업

- GetSchema
- GetInferredSchema

데이터베이스 작업

- GetDatabases

계획 작업

- GetPlan

테이블 작업

- SearchTables
- GetTables
- GetTable

연결 작업

- CreateConnection
- DeleteConnection
- UpdateConnection
- GetConnections
- GetConnection

매핑 작업

- GetMapping

S3 프록시 작업

- ListBuckets
- ListObjectsV2

- GetBucketLocation

보안 구성 작업

- GetSecurityConfigurations

스크립트 작업

- CreateScript(AWS Glue에서 같은 이름의 API와 다름)

AWS Glue Studio API에 액세스

AWS Glue Studio에 액세스하려면 IAM 권한 내의 작업 정책 목록에 `glue:UseGlueStudio`를 추가합니다.

아래 예제에서는 `glue:UseGlueStudio`가 작업 정책에 포함되어 있지만 AWS Glue Studio API는 개별적으로 식별되지 않습니다. 그 이유는 `glue:UseGlueStudio`를 포함하면 IAM 권한 내의 개별 AWS Glue Studio API를 지정하지 않아도 내부 API에 대한 액세스 권한이 사용자에게 자동으로 부여되기 때문입니다.

이 예제에서 나열된 추가 작업 정책(예: `glue:SearchTables`)은 AWS Glue Studio API가 아니며, 따라서 필요에 따라 IAM 권한에 포함되어야 합니다. 부여할 Amazon S3 액세스 수준을 지정할 때 Amazon S3 프록시 작업을 포함할 수도 있습니다. 아래 예제 정책에서는 AWS Glue Studio를 열고, 시각적 작업을 생성하고 선택한 IAM 역할에 충분한 액세스 권한이 있는 경우 이를 저장/실행하는 액세스 권한을 제공합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "glue:UseGlueStudio",
        "iam:ListRoles",
        "iam:ListUsers",
        "iam:ListGroups",
        "iam:ListRolePolicies",
        "iam:GetRole",
        "iam:GetRolePolicy",

```

```

        "glue:SearchTables",
        "glue:GetConnections",
        "glue:GetJobs",
        "glue:GetTables",
        "glue:BatchStopJobRun",
        "glue:GetSecurityConfigurations",
        "glue>DeleteJob",
        "glue:GetDatabases",
        "glue:CreateConnection",
        "glue:GetSchema",
        "glue:GetTable",
        "glue:GetMapping",
        "glue:CreateJob",
        "glue>DeleteConnection",
        "glue:CreateScript",
        "glue:UpdateConnection",
        "glue:GetConnection",
        "glue:StartJobRun",
        "glue:GetJobRun",
        "glue:UpdateJob",
        "glue:GetPlan",
        "glue:GetJobRuns",
        "glue:GetTags",
        "glue:GetJob",
        "glue:QueryJobRuns",
        "glue:QueryJobs",
        "glue:QueryJobRunsAggregated"
    ],
    "Resource": "*"
},
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/AWSGlueServiceRole*",
    "Condition": {
        "StringLike": {
            "iam:PassedToService": [
                "glue.amazonaws.com"
            ]
        }
    }
}
}

```

```
    ]
  }
```

노트북 및 데이터 미리 보기 권한

데이터 미리 보기와 노트북을 사용하면 작업을 실행하지 않고도 작업의 모든 스테이지(읽기, 변환, 쓰기)에서 데이터 샘플을 볼 수 있습니다. 데이터에 액세스할 때 사용할 AWS Glue Studio에 대한 AWS Identity and Access Management(IAM) 역할을 지정합니다. IAM 역할은 수입 가능하도록 설계되었으며 암호 또는 액세스 키와 같은 표준 장기 자격 증명이 연결되어 있지 않습니다. 대신, AWS Glue Studio가 역할을 수입할 때 IAM은 임시 보안 자격 증명을 제공합니다.

데이터 미리 보기와 노트북 명령이 올바르게 작동하려면 이름이 `AWSGlueServiceRole` 문자열로 시작하는 역할을 사용합니다. 역할에 다른 이름을 사용하려는 경우 `iam:passrole` 권한을 추가하고 IAM에서 해당 역할에 대한 정책을 구성해야 합니다. 자세한 내용은 [역할에 대해 'AWSGlueServiceRole*'이라는 이름이 아닌 IAM 정책 생성 단원을 참조하십시오.](#)

Warning

역할이 노트북에 대한 `iam:passrole` 권한을 부여하고 역할 체인을 구현하는 경우 사용자가 의도하지 않게 노트북에 액세스하게 될 수 있습니다. 노트북에 대한 액세스 권한이 부여된 사용자를 모니터링할 수 있는 감사는 현재 구현되어 있지 않습니다.

IAM ID가 데이터 미리 보기 세션을 생성하지 못하게 하려면 [the section called "ID가 데이터 미리 보기 세션을 생성하지 못하게 하기"](#) 섹션의 다음 예제를 참조하세요.

Amazon CloudWatch 권한

AWS Glue에서 원시 데이터를 수집한 후 판독이 가능한 지표로 실시간에 가깝게 처리하는 Amazon CloudWatch를 사용하여 AWS Glue Studio 작업을 모니터링할 수 있습니다. 기본적으로 AWS Glue 지표 데이터는 CloudWatch에 자동으로 전송됩니다. 자세한 내용은 Amazon CloudWatch User Guide의 [What Is Amazon CloudWatch?](#)와 AWS Glue Developer Guide의 [AWS Glue Metrics](#)를 참조하세요.

CloudWatch 대시보드에 액세스하려면 AWS Glue Studio에 액세스하는 사용자에게 다음 중 하나가 필요합니다.

- `AdministratorAccess` 정책
- `CloudWatchFullAccess` 정책

- 다음과 같은 특정 권한 중 하나 이상을 포함하는 사용자 지정 정책:
 - 대시보드 보기를 위한 `cloudwatch:GetDashboard` 및 `cloudwatch:ListDashboards`
 - 대시보드를 생성하거나 수정하는 `cloudwatch:PutDashboard`
 - 대시보드를 삭제하는 `cloudwatch>DeleteDashboards`

정책을 사용하여 IAM 사용자의 권한 변경에 대한 자세한 내용은 IAM User Guide의 [Changing Permissions for an IAM User](#)를 참조하세요.

ETL 작업에 필요한 IAM 권한 검토

AWS Glue Studio를 사용하여 작업을 생성하는 경우 작업은 생성 시 지정한 IAM 역할의 권한을 갖게 됩니다. 이 IAM 역할은 데이터 원본에서 데이터를 추출하고, 대상에 데이터를 쓰고, AWS Glue 리소스에 액세스할 수 있는 권한이 있어야 합니다.

작업을 위해 생성하는 역할의 이름이 `AWSGlueServiceRole` 문자열로 시작해야 AWS Glue Studio에서 올바르게 사용할 수 있습니다. 예를 들어 역할 이름을 `AWSGlueServiceRole-FlightDataJob`으로 지정할 수 있습니다.

데이터 원본 및 데이터 대상 권한

AWS Glue Studio 작업에서 사용하는 모든 소스, 대상, 스크립트, 임시 디렉터리를 위해 작업이 Amazon S3에 액세스할 수 있어야 합니다. 특정 Amazon S3 리소스에 대한 세분화된 액세스를 제공하는 정책을 생성할 수 있습니다.

- 데이터 원본은 `s3:ListBucket` 및 `s3:GetObject` 권한을 요구합니다.
- 데이터 대상은 `s3:ListBucket`, `s3:PutObject` 및 `s3>DeleteObject` 권한을 요구합니다.

Amazon Redshift를 데이터 원본으로 선택하면 클러스터 권한에 대한 역할을 제공할 수 있습니다. Amazon Redshift 클러스터에 대해 실행되는 작업은 임시 자격 증명을 사용하여 임시 스토리지용 Amazon S3에 액세스하는 명령을 실행합니다. 작업이 1시간 이상 실행되면 이러한 자격 증명이 만료되어 작업이 실패합니다. 이 문제를 방지하기 위해 임시 자격 증명을 사용하여 작업에 필요한 권한을 부여하는 역할을 Amazon Redshift 클러스터 자체에 할당할 수 있습니다. 자세한 내용은 AWS Glue Developer Guide의 [Moving Data to and from Amazon Redshift](#)를 참조하세요.

작업이 Amazon S3 이외의 데이터 원본 또는 대상을 사용하는 경우 이러한 데이터 원본 및 대상에 액세스하기 위해 작업에서 사용하는 IAM 역할에 필요한 권한을 연결해야 합니다. 자세한 내용은 AWS Glue Developer Guide의 [Setting Up Your Environment to Access Data Stores](#)를 참조하세요.

데이터 스토어에 커넥터 및 연결을 사용하는 경우 [the section called “커넥터 사용에 필요한 권한”](#)에 설명된 대로 추가 권한이 필요합니다.

작업 삭제에 필요한 권한

AWS Glue Studio의 콘솔에서 여러 작업을 선택하여 삭제할 수 있습니다. 이 작업을 수행하려면 `glue:BatchDeleteJob` 권한이 있어야 합니다. 이는 작업 삭제에 `glue>DeleteJob` 권한이 필요한 AWS Glue 콘솔과 다릅니다.

AWS Key Management Service 권한

AWS Key Management Service(AWS KMS)에서 서버 측 암호화를 사용하는 Amazon S3 소스 및 대상에 액세스하려는 경우 작업에 사용되는 AWS Glue Studio 역할에 작업에서 데이터를 복호화할 수 있도록 하는 정책을 연결합니다. 작업 역할에는 `kms:ReEncrypt`, `kms:GenerateDataKey` 및 `kms:DescribeKey` 권한이 필요합니다. 또한 작업 역할에는 AWS KMS 고객 마스터 키(CMK)로 암호화된 Amazon S3 객체를 업로드하거나 다운로드할 수 있는 `kms:Decrypt` 권한이 필요합니다.

AWS KMS CMK를 사용하면 추가 요금이 부과됩니다. 자세한 내용은 AWS Key Management Service Developer Guide의 [AWS Key Management Service Concepts - Customer Master Keys \(CMKs\)](#) 및 [AWS Key Management Service Pricing](#)을 참조하세요.

커넥터 사용에 필요한 권한

AWS Glue 사용자 정의 커넥터 및 연결을 사용하여 데이터 스토어에 액세스하는 경우 AWS Glue ETL 작업을 실행하는 데 사용되는 역할에 연결된 추가 권한이 필요합니다.

- AWS Marketplace에서 구입한 커넥터 액세스를 위한 AWS 관리형 정책 `AmazonEC2ContainerRegistryReadOnly`.
- `glue:GetJob` 및 `glue:GetJobs` 권한.
- 연결에 사용되는 보안 암호 액세스를 위한 AWS Secrets Manager 권한. 예제 IAM 정책을 보려면 [예: 보안 암호 값을 검색할 수 있는 권한](#)을 참조하세요.

AWS Glue ETL 작업이 Amazon VPC를 실행하는 VPC 내에서 실행되는 경우 VPC는 [the section called “ETL 작업에 사용할 VPC 구성”](#)에 설명된 대로 구성되어야 합니다.

AWS Glue Studio에 대한 IAM 권한 설정

AWS 관리자 사용자를 사용하여 역할을 생성하고 사용자 및 작업 역할에 정책을 할당할 수 있습니다.

`AWSGlueConsoleFullAccess` AWS 관리형 정책을 사용하여 AWS Glue Studio 콘솔 사용에 필요한 권한을 제공할 수 있습니다.

자체 정책을 생성하려면 AWS Glue Developer Guide의 [Create an IAM Policy for the AWS Glue Service](#)에 설명된 단계를 따릅니다. 이전에 [AWS Glue Studio 사용자에게 필요한 IAM 권한 검토](#)에서 설명한 IAM 권한을 포함합니다.

주제

- [AWS Glue Studio 사용자에게 정책을 연결합니다.](#)
- [역할에 대해 'AWSGlueServiceRole*'이라는 이름이 아닌 IAM 정책 생성](#)

AWS Glue Studio 사용자에게 정책을 연결합니다.

AWS Glue Studio 콘솔에 로그인하는 모든 AWS 사용자에게 특정 리소스에 액세스할 수 있는 권한이 있어야 합니다. 사용자에게 IAM 정책을 할당하여 해당 권한을 제공합니다.

AWSGlueConsoleFullAccess 관리형 정책을 사용자에게 연결

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 목록에서 AWSGlueConsoleFullAccess 옆의 확인란을 선택합니다. [Filter] 메뉴와 검색 상자를 사용하여 정책 목록을 필터링할 수 있습니다.
4. 정책 조치를 선택한 후 연결을 선택합니다.
5. 정책을 연결하려는 사용자를 선택합니다. 필터 메뉴와 검색 상자를 사용하면 보안 주체 개체 목록을 필터링할 수 있습니다. 정책을 추가할 사용자를 선택한 다음 [Attach policy(정책 추가)]를 선택합니다.
6. 필요에 따라 이전 단계를 반복하여 사용자에게 추가 정책을 연결합니다.

역할에 대해 'AWSGlueServiceRole*'이라는 이름이 아닌 IAM 정책 생성

AWS Glue Studio에서 사용하는 역할에 대한 IAM 정책 구성

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 새 IAM 정책을 추가합니다. 기존 정책에 추가하거나 새 IAM 인라인 정책을 생성할 수 있습니다.

IAM 정책 생성

 1. 정책을 선택한 후 정책 생성을 선택합니다. 시작 버튼이 표시되면 이 버튼을 선택한 다음 정책 생성을 선택합니다.

2. [Create Your Own Policy] 옆의 [Select]를 선택합니다.
 3. 정책 이름에 나중에 쉽게 참조할 수 있는 값을 입력합니다. 선택적으로, 설명에 설명을 입력합니다.
 4. 정책 문서에 다음 형식의 정책 설명을 입력한 다음, 정책 생성을 선택합니다.
3. 다음 블록을 복사하여 'Statement' 배열 아래의 정책에 붙여넣습니다. 이때 *my-interactive-session-role-prefix*를 AWS Glue에 대한 권한과 연결할 모든 공통 역할의 접두사로 바꿉니다.

```
{
  "Action": [
    "iam:PassRole"
  ],
  "Effect": "Allow",
  "Resource": "arn:aws:iam::*:role/my-interactive-session-role-prefix",
  "Condition": {
    "StringLike": {
      "iam:PassedToService": [
        "glue.amazonaws.com "
      ]
    }
  }
}
```

다음은 정책에 포함된 Version 및 Statement 배열의 전체 예입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/my-interactive-session-role-prefix",
      "Condition": {
        "StringLike": {
          "iam:PassedToService": [
            "glue.amazonaws.com "
          ]
        }
      }
    }
  ]
}
```

```

    }
  }
]
}

```

4. 사용자에게 정책을 활성화하려면 사용자를 선택합니다.
5. 정책을 연결하려는 사용자를 선택합니다.

ETL 작업에 사용할 VPC 구성

Amazon Virtual Private Cloud(Amazon VPC)를 사용하면 AWS 클라우드 내에서 논리적으로 격리된 자체 영역에 Virtual Private Cloud(VPC)라고 하는 가상 네트워크를 정의할 수 있습니다. 인스턴스와 같은 AWS 리소스를 VPC에서 시작할 수 있습니다. VPC는 고객의 자체 데이터 센터에서 운영하는 기존 네트워크와 매우 유사하지만 AWS의 확장 가능한 인프라를 사용한다는 이점을 제공합니다. 해당 IP 주소 범위를 선택하고, 서브넷을 만든 후 라우팅 테이블, 네트워크 게이트웨이 및 보안 설정을 구성하여 VPC를 구성할 수 있습니다. VPC의 인스턴스를 인터넷에 연결합니다. VPC를 사내 데이터 센터에 연결하여 AWS 클라우드에서 데이터 센터를 확장할 수 있습니다. 각의 서브넷에서 리소스를 보호하기 위해 보안 그룹 및 네트워크 액세스 제어 목록을 포함한 다중 보안 계층을 사용할 수 있습니다. 자세한 내용은 [Amazon VPC 사용 설명서](#)를 참조하세요.

커넥터를 사용할 때 VPC 내에서 실행되도록 AWS Glue ETL 작업을 구성할 수 있습니다. 필요에 따라 다음에 대해 VPC를 구성해야 합니다.

- AWS에 없는 데이터 스토어에 대한 퍼블릭 네트워크 액세스. 작업이 액세스한 모든 데이터 스토어는 VPC 서브넷에서 사용 가능해야 합니다.
- 작업에서 VPC 리소스와 퍼블릭 인터넷에 모두 액세스해야 할 경우 VPC 내부 네트워크 주소 변환(NAT) 게이트웨이가 VPC에 있어야 합니다.

자세한 내용은 AWS Glue Developer Guide의 [Setting Up Your Environment to Access Data Stores](#)를 참조하세요.

AWS Glue Studio에서 노트북 시작하기

AWS Glue Studio를 통해 노트북을 시작하는 경우 데이터를 탐색하고 몇 초 만에 작업 스크립트 개발을 시작할 수 있도록 모든 구성 단계가 완료됩니다.

다음 섹션에서는 ETL 작업에 대해 AWS Glue Studio에서 노트북을 사용하기 위해 역할을 생성하고 적절한 권한을 부여하는 방법을 설명합니다.

주제

- [IAM 역할에 권한 부여](#)

IAM 역할에 권한 부여

AWS Glue Studio 설정은 노트북을 사용하기 위한 전제 조건입니다.

AWS Glue에서 노트북을 사용하려면 역할에 다음이 필요합니다.

- sts:AssumeRole 작업(및 태그 지정을 원할 경우 sts:TagSession 작업)을 수행하기 위한 AWS Glue와의 신뢰 관계
- 노트북, AWS Glue 및 대화형 세션에 대한 모든 API 작업을 포함하는 IAM 정책
- 전달 역할에 대한 IAM 정책(역할이 노트북에서 대화형 세션으로 전달될 수 있어야 하기 때문)

예를 들어 새 역할을 생성할 때 역할에 AWSGlueConsoleFullAccessRole과 같은 표준 AWS 관리 형 정책을 추가한 후 노트북 작업에 대한 새 정책과 IAM PassRole 정책에 대한 다른 새 정책을 추가할 수 있습니다.

AWS Glue와의 신뢰 관계에 필요한 작업

노트북 세션을 시작할 때 노트북에 전달되는 역할의 신뢰 관계에 sts:AssumeRole을 추가해야 합니다. 세션에 태그가 포함되어 있는 경우 sts:TagSession 작업도 전달해야 합니다. 이러한 작업이 없으면 노트북 세션을 시작할 수 없습니다.

예:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "glue.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

노트북에 대한 API 작업을 포함하는 정책

다음 샘플 정책은 노트북에 대한 필수적인 AWS IAM 권한을 설명합니다. 새 역할을 생성하는 경우 다음을 포함하는 정책을 생성합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:StartNotebook",
        "glue:TerminateNotebook",
        "glue:GlueNotebookRefreshCredentials",
        "glue:DeregisterDataPreview",
        "glue:GetNotebookInstanceStatus",
        "glue:GlueNotebookAuthorize"
      ],
      "Resource": "*"
    }
  ]
}
```

다음 IAM 정책을 사용하여 특정 리소스에 대한 액세스를 허용할 수 있습니다.

- **AwsGlueSessionUserRestrictedNotebookServiceRole**: 세션을 제외한 모든 AWS Glue 리소스에 대한 전체 액세스 권한을 제공합니다. 사용자가 자신과 연결된 노트북 세션만 생성하고 사용할 수 있도록 허용합니다. 이 정책에는 다른 AWS 서비스에서 AWS Glue 리소스를 관리하는 데 AWS Glue에서 필요한 기타 권한도 포함됩니다.
- **AwsGlueSessionUserRestrictedNotebookPolicy**: 사용자가 자신과 연결된 노트북 세션만 생성하고 사용할 수 있도록 허용하는 권한을 제공합니다. 이 정책에는 제한된 AWS Glue 세션 역할을 사용자가 전달할 수 있도록 명시적으로 허용하는 권한도 포함되어 있습니다.

역할을 전달하는 IAM 정책

역할이 있는 노트북을 생성하면 해당 역할이 대화형 세션으로 전달되므로 두 위치에서 동일한 역할을 사용할 수 있습니다. 따라서 iam:PassRole 권한은 역할의 정책의 일부여야 합니다.

다음 예제를 사용하여 역할에 대한 새 정책을 생성합니다. 계정 번호를 사용자의 고유한 역할 이름으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::090000000210:role/<role_name>"
    }
  ]
}
```

AWS Glue 사용 프로필 설정

클라우드 플랫폼 사용의 주요 이점 중 하나는 유연성입니다. 그러나 컴퓨팅 리소스를 이렇게 쉽게 만들 수 있기 때문에 관리하지 않고 가드레일을 사용하지 않으면 클라우드 비용이 급증할 위험이 있습니다. 따라서 관리자는 높은 인프라 비용을 피하는 동시에 사용자가 불필요한 마찰 없이 작업할 수 있도록 균형을 맞춰야 합니다.

관리자는 AWS Glue 사용 프로필을 사용하여 개발자, 테스터, 제품 팀 등 계정 내 다양한 사용자 계층에 대해 서로 다른 프로필을 만들 수 있습니다. 각 프로필은 다양한 유형의 사용자에게 할당할 수 있는 고유한 매개 변수 집합입니다. 예를 들어 개발자에게는 더 많은 작업자가 필요하고 최대 작업자 수가 더 많을 수 있는 반면, 제품 팀에는 필요한 작업자 수가 적고 타임아웃 또는 유희 타임아웃 값이 낮아질 수 있습니다.

작업 및 작업 실행 동작의 예

사용자 A가 프로필 A를 사용하여 작업을 생성했다고 가정해 보겠습니다. 작업은 특정 매개변수 값과 함께 저장됩니다. 프로필 B를 가진 사용자 B가 작업을 실행하려고 합니다.

사용자 A가 작업을 작성할 때 특정 작업자 수를 설정하지 않은 경우 사용자 A 프로필에 설정된 기본값이 적용되어 작업 정의와 함께 저장되었습니다.

사용자 B가 작업을 실행하면 해당 작업에 대해 저장된 모든 값을 사용하여 작업이 실행됩니다. 사용자 B의 프로필이 더 제한적이고 작업자 수가 많은 상태에서 실행할 수 없는 경우에는 작업 실행이 실패합니다.

리소스로서의 사용 프로필

AWS Glue 사용 프로필은 Amazon 리소스 이름 (ARN) 으로 식별되는 리소스입니다. 조치 기반 및 리소스 기반 권한 부여를 비롯한 모든 기본 IAM (Identity 및 Access Management) 제어가 적용됩니다. 관리

자는 AWS Glue 리소스를 생성하는 사용자의 IAM 정책을 업데이트하여 프로필을 사용할 수 있는 액세스 권한을 부여해야 합니다.

The screenshot shows the AWS Glue console interface for managing usage profiles. It includes a sidebar with navigation options like 'Tables', 'Stream schema registries', and 'Usage profiles'. The main content area is titled 'Usage profiles (1/9)' and contains a table with columns for Name, Status, Description, and Created on (UTC). The table lists several profiles, some assigned and some not assigned. A 'Create usage profile' button is visible at the top right of the main content area.

Name	Status	Description	Created on (UTC)
dev-profile-1	Assigned	-	April 30, 2024, 02:19:53
dev-profile-2	Not assigned	I edited the description and default workers	April 25, 2024, 22:10:17
product-profile-1	Not assigned	-	April 30, 2024, 02:19:02
product-profile-2	Assigned	-	May 7, 2024, 20:39:18
tester-profile-1	Assigned	test description has been edited	May 7, 2024, 20:55:25
tester-profile-2	Assigned	glue testing profile	May 7, 2024, 21:20:13
test	Assigned	I edited this successfully again	April 25, 2024, 20:28:48
test profile	Not assigned	Description I edited this	April 30, 2024, 17:17:53

주제

- [사용 프로필 생성 및 관리](#)
- [사용 프로필 및 작업](#)

사용 프로필 생성 및 관리

AWS Glue 사용 프로필 생성

관리자는 사용 프로필을 만든 다음 이를 다양한 사용자에게 할당해야 합니다. 사용 프로필을 만들 때 다양한 작업 및 세션 매개변수에 허용되는 값 범위뿐 아니라 기본값도 지정합니다. 작업 또는 대화형 세션에 대해 하나 이상의 매개 변수를 구성해야 합니다. 작업에 매개 변수 값이 제공되지 않을 때 사용할 기본값을 사용자 정의하거나, 이 프로필을 사용할 때 사용자가 매개 변수 값을 제공하는 경우 검증을 위한 범위 제한 또는 허용 값 세트를 설정할 수 있습니다.

기본값은 관리자가 작업 작성자를 지원하기 위해 설정한 모범 사례입니다. 사용자가 새 작업을 생성하고 시간 제한 값을 설정하지 않으면 사용 프로필의 기본 제한 시간이 적용됩니다. 작성자에게 프로필이 없는 경우 AWS Glue 서비스 기본값이 적용되어 작업 정의에 저장됩니다. 런타임 시 프로필에 설정된 제한 (최소, 최대, 허용된 작업자) 을 AWS Glue 적용합니다.

매개변수가 구성되면 다른 모든 매개변수는 선택 사항입니다. 작업 또는 대화형 세션에 대해 사용자 정의할 수 있는 매개 변수는 다음과 같습니다.

- **작업자 수** — 컴퓨팅 리소스의 과도한 사용을 방지하려면 작업자 수를 제한하십시오. 기본값, 최소값, 최대값을 설정할 수 있습니다. 최소값은 1입니다.
- **작업자 유형** — 워크로드와 관련된 작업자 유형을 제한합니다. 기본 유형을 설정하고 사용자 프로필에 작업자 유형을 허용할 수 있습니다.
- **타임아웃** — 작업 또는 대화형 세션이 종료되기 전에 실행 및 리소스를 소비할 수 있는 최대 시간을 정의합니다. 타임아웃 값을 설정하여 작업이 오래 실행되지 않도록 하세요.

기본값, 최소값, 최대값을 분 단위로 설정할 수 있습니다. 최소값은 1 (분) 입니다. AWS Glue 기본 제한 시간은 2880분이지만 사용 프로필에서 원하는 기본값을 설정할 수 있습니다.

'기본값' 값을 설정하는 것이 가장 좋습니다. 이 값은 사용자가 값을 설정하지 않은 경우 작업 또는 세션 생성에 사용됩니다.

- **휴식 제한 시간** — 셸이 실행된 후 제한 시간이 초과되기 전에 대화형 세션이 비활성화되는 시간 (분)을 정의합니다. 작업이 완료된 후 종료할 대화형 세션의 휴식 제한 시간을 정의합니다. 휴식 타임아웃 범위는 타임아웃 한도 이내여야 합니다.

기본값, 최소값, 최대값을 분 단위로 설정할 수 있습니다. 최소값은 1 (분) 입니다. AWS Glue 기본 제한 시간은 2880분이지만 사용 프로필에서 원하는 기본값을 설정할 수 있습니다.

'기본값' 값을 설정하는 것이 가장 좋습니다. 이 값은 사용자가 값을 설정하지 않은 경우 세션 생성에 사용됩니다.

관리자로서 AWS Glue 사용자 프로필을 만들려면 (콘솔)

1. 왼쪽 탐색 메뉴에서 비용 관리를 선택합니다.
2. 사용자 프로필 생성을 선택합니다.
3. 사용 프로필의 사용 프로필 이름을 입력합니다.
4. 다른 사람들이 사용 프로필의 용도를 쉽게 알 수 있도록 설명 (선택 사항) 을 입력합니다.
5. 프로필에 매개변수를 하나 이상 정의하십시오. 양식의 모든 필드는 매개변수입니다. 세션 휴식 제한 시간 최소값을 예로 들 수 있습니다.
6. 사용 프로필에 적용되는 선택적 태그를 모두 정의하십시오.
7. 저장을 선택합니다.

AWS Glue X

Getting started
ETL jobs
 Visual ETL
 Notebooks
 Job run monitoring
Data Catalog tables
Data connections
Workflows (orchestration)

▼ **Data Catalog**
Databases
 Tables
Stream schema registries
 Schemas
Connections
Crawlers
 Classifiers
Catalog settings

▼ **Data Integration and ETL**
ETL jobs
 Visual ETL
 Notebooks
 Job run monitoring
Interactive Sessions
Data classification tools
 Sensitive data detection
 Record Matching
Triggers
Workflows (orchestration)
 Blueprints
Security configurations
Cost management *New*

▶ **Legacy pages**

What's New [↗](#)
Documentation [↗](#)
AWS Marketplace

Enable compact mode

AWS Glue > Usage profiles > Create usage profile

Create usage profile [Info](#)

When you create a usage profile, you can assign it to AWS IAM roles and users to control cloud costs.

Name and description

Usage profile name

Usage profile description - optional

Descriptions can be up to 2048 characters long.

Parameter configurations [Info](#)

⚠ Please configure at least one parameter for jobs or interactive sessions to create a usage profile. Once a parameter is configured, all other parameters are optional.

▼ **Customize parameter configurations for jobs**

Number of workers

The number of workers of a defined worker_type that are allocated. Customize the number of workers to avoid excessive use of compute resources.

Default	Minimum	Maximum
<input type="text" value="10"/>	<input type="text" value="1"/>	<input type="text" value="20"/>

Between minimum and maximum Minimum allowed value: 1

Worker type

The type of predefined worker that is allocated when a job runs. Select the relevant worker types for your workloads.

Default worker type

Allowed worker types

Timeout

The maximum time in minutes that an interactive session run can consume resources before it is terminated. Set up a timeout value to avoid long running sessions.

Default (minutes)	Minimum (minutes)	Maximum (minutes)
<input type="text" value="2880"/>	<input type="text" value="1"/>	<input type="text" value="4000"/>

Between minimum and maximum Minimum allowed value: 1

사용 프로파일 (AWS CLI) 을 만들려면

1. 다음 명령을 입력합니다.

```
aws glue create-usage-profile --name profile-name --configuration file://config.json --tags list-of-tags
```

여기서 config.json은 대화형 세션 (SessionConfiguration) 및 작업 () 에 대한 매개변수 값을 정의할 수 있습니다. JobConfiguration

```
//config.json (There is a separate blob for session/job configuration
{
  "SessionConfiguration": {
    "timeout": {
      "DefaultValue": "2880",
      "MinValue": "100",
      "MaxValue": "4000"
    },
    "idleTimeout": {
      "DefaultValue": "30",
      "MinValue": "10",
      "MaxValue": "4000"
    },
    "workerType": {
      "DefaultValue": "G.2X",
      "AllowedValues": [
        "G.2X",
        "G.4X",
        "G.8X"
      ]
    },
    "numberOfWorkers": {
      "DefaultValue": "10",
      "MinValue": "1",
      "MaxValue": "10"
    }
  },
  "JobConfiguration": {
    "timeout": {
      "DefaultValue": "2880",
      "MinValue": "100",
      "MaxValue": "4000"
    },
    "workerType": {
      "DefaultValue": "G.2X",
      "AllowedValues": [
        "G.2X",
        "G.4X",
        "G.8X"
      ]
    }
  }
}
```

```

    ]
  },
  "numberOfWorkers": {
    "DefaultValue": "10",
    "MinValue": "1",
    "MaxValue": "10"
  }
}
}

```

2. 다음 명령을 입력하여 생성된 사용 프로필을 확인합니다.

```
aws glue get-usage-profile --name profile-name
```

응답:

```

{
  "ProfileName": "foo",
  "Configuration": {
    "SessionConfiguration": {
      "numberOfWorkers": {
        "DefaultValue": "10",
        "MinValue": "1",
        "MaxValue": "10"
      },
      "workerType": {
        "DefaultValue": "G.2X",
        "AllowedValues": [
          "G.2X",
          "G.4X",
          "G.8X"
        ]
      },
      "timeout": {
        "DefaultValue": "2880",
        "MinValue": "100",
        "MaxValue": "4000"
      },
      "idleTimeout": {
        "DefaultValue": "30",
        "MinValue": "10",
        "MaxValue": "4000"
      }
    }
  }
}

```



```

    },
    "JobConfiguration": {
      "numberOfWorkers": {
        "DefaultValue": "10",
        "MinValue": "1",
        "MaxValue": "10"
      },
      "workerType": {
        "DefaultValue": "G.2X",
        "AllowedValues": [
          "G.2X",
          "G.4X",
          "G.8X"
        ]
      },
      "timeout": {
        "DefaultValue": "2880",
        "MinValue": "100",
        "MaxValue": "4000"
      }
    },
    "CreatedOn": "2024-01-19T23:15:24.542000+00:00"
  }
}

```

사용 프로필을 관리하는 데 사용되는 추가 CLI 명령:

- `aws 글루 list-usage-profiles`
- `aws glue update-usage-profile --## ### ## --## ##: //config.json`
- `aws glue delete-usage-profile --name ### 이름`

사용 프로필 편집

관리자는 자신이 만든 사용 프로필을 편집하여 작업 및 대화형 세션의 프로필 매개변수 값을 변경할 수 있습니다.

사용자 프로필을 편집하려면:

관리자로서 AWS Glue 사용자 프로필을 편집하려면 (콘솔)

1. 왼쪽 탐색 메뉴에서 비용 관리를 선택합니다.

2. 편집 권한이 있는 사용자 프로필을 선택하고 편집을 선택합니다.
3. 필요에 따라 프로필을 변경합니다. 기본적으로 이미 값이 있는 매개변수가 확장됩니다.
4. [편집 내용 저장] 을 선택합니다.

aws Services Search for services, features, blogs, docs, and more [Alt+S] N. Virginia MyRole/AWSUser @ 0123-4567-8901

AWS Glue > Usage profiles > dev-profile-1 > Edit

Edit dev-profile-1

Name and description

Usage profile name

Usage profile description - optional

Descriptions can be up to 2048 characters long.

▼ Parameter configurations for jobs [Info](#)
Configure usage restrictions for AWS Glue jobs. Each parameter has a default value preconfigured for different types of jobs.

▼ Number of workers
The number of workers of a defined worker_type that are allocated. Customize number of workers to avoid excessive use of compute resources.

Default: Minimum: Maximum:
Between minimum and maximum Minimum allowed value: 1

▼ Worker type
The type of a unit capable of performing operational processes dictated by its fleet management system. Select the relevant worker types for your wo

Default worker type:

Allowed worker types:

▶ Timeout
The maximum time in minutes that a job run can consume resources before it is terminated and. Setup timeout values to avoid long running jobs.

▼ Parmeter configurations for sessions [Info](#)
Configure usage restrictions for AWS Glue interactive sessions. Each parameter has a default value preconfigured for different types of interactive sessions.

▶ Number of workers
The number of workers of a defined worker_type that are allocated. Customize number of workers to avoid excessive use of compute resources.

▶ Worker type
The type of a unit capable of performing operational processes dictated by its fleet management system. Select the relevant worker types for your workloads.

▼ Idle timeout
The number of minutes of inactivity after which an interactive session will timeout after a cell has been executed. Define idle-timeout for sessions to terminate after the work completed.

Default (minutes): Minimum (minutes): Maximum (minutes):
Between minimum and maximum Minimum allowed value: 1

▶ Timeout
The maximum time in minutes that an interactive session run can consume resources before it is terminated. Setup timeout values to avoid long running sessions.

▶ Tags - optional
Tags are user-defined key-value pairs that provide metadata to organize and classify your AWS resources.

사용 프로필 (AWS CLI) 을 편집하려면

- 다음 명령을 입력합니다. 위의 create 명령에 표시된 것과 동일한 --configuration 파일 구문이 사용됩니다.

```
aws glue update-usage-profile --name profile-name --configuration file://
config.json
```

여기서 config.json은 대화형 세션 (SessionConfiguration) 및 jobs () 의 매개 변수 값을 정의합니다. JobConfiguration

사용 프로필 할당

사용 프로필 페이지의 사용을 상태 열에는 사용 프로필이 사용자에게 할당되었는지 여부가 표시됩니다. 상태를 마우스로 가리키면 할당된 IAM 엔티티가 표시됩니다.

관리자는 리소스를 생성하는 사용자/역할에 AWS Glue 사용 프로필을 할당할 수 있습니다. AWS Glue 사용 프로필 할당은 다음 두 가지 작업의 조합입니다.

- 키로 IAM 사용자/역할 태그를 업데이트한 다음 glue:UsageProfile
- 사용자/역할의 IAM 정책 업데이트

AWS Glue Studio를 사용하여 작업/대화형 세션을 생성하는 사용자의 경우 관리자는 다음 역할에 태그를 지정합니다.

- 작업 제한의 경우 관리자는 로그인한 콘솔 역할에 태그를 지정합니다.
- 대화형 세션에 대한 제한 사항의 경우 관리자는 사용자가 노트북을 만들 때 제공하는 역할에 태그를 지정합니다.

다음은 관리자가 리소스를 생성하는 IAM 사용자/역할에 대해 업데이트해야 하는 예제 정책입니다.

AWS Glue

```
{
  "Effect": "Allow",
  "Action": [
    "glue:GetUsageProfile"
  ],
  "Resource": [
```

```

    "arn:aws:glue:us-east-1:123456789012:usageProfile/foo"
  ]
}

```

AWS Glue AWS Glue 사용 프로필에 지정된 값을 기반으로 작업, 작업 실행 및 세션 요청을 검증하고 요청이 허용되지 않는 경우 예외를 발생시킵니다. 동기 API의 경우 사용자에게 오류가 발생합니다. 비 동기 경로의 경우 입력 매개변수가 사용자/역할에 할당된 프로필에 허용되는 범위를 벗어났다는 오류 메시지와 함께 실패한 작업 실행이 생성됩니다.

사용자/역할에 사용 프로필을 할당하려면:

1. (ID 및 Access Management) IAM 콘솔을 엽니다.
2. 왼쪽 탐색창에서 [사용자] 또는 [역할] 을 선택합니다.
3. 사용자 또는 역할을 선택합니다.
4. 태그 탭을 선택합니다.
5. 새 태그 추가를 선택합니다.
6. 사용 프로필 이름의 **glue:UsageProfile**키와 값이 포함된 태그를 추가합니다.
7. 변경 사항 저장(Save changes)을 선택합니다

The screenshot shows the AWS IAM console interface for an **AWSGlueServiceRole**. The **Tags (1)** tab is selected, displaying a table with one tag:

Key	Value
glue:UsageProfile	foo

Other visible details include the role's ARN: `arn:aws:iam:::role/service-role/AWSGlueServiceRole` and a maximum session duration of 1 hour.

할당된 사용 프로필 보기

사용자는 할당된 사용 프로필을 보고 API 호출을 통해 AWS Glue 작업 및 세션 리소스를 생성하거나 작업을 시작할 때 이를 사용할 수 있습니다.

프로필 권한은 IAM 정책에서 제공됩니다. 발신자 정책에 `glue:UsageProfile` 권한이 있는 한 사용자는 프로필을 볼 수 있습니다. 그렇지 않으면 액세스 거부 오류가 발생합니다.

할당된 사용자 프로필을 보려면:

1. 왼쪽 탐색 메뉴에서 비용 관리를 선택합니다.
2. 볼 권한이 있는 사용자 프로필을 선택합니다.

Usage profile "dev-provile-1" successfully updated. Usage profile "dev-provile-1" successfully updated. To assign it to IAM roles or users, go to AWS IAM service through the "Open AWS IAM" button and tag the IAM role or user with key: glue:UsageProfile and value: dev-profile-1.

[Open AWS IAM](#)

AWS Glue > Usage profiles > dev-profile-1

dev-profile-1

[Edit](#) [Delete](#)

Usage profile details

Usage profile name dev-profile-1	Status Assigned	Created on October 18, 2023, 14:32 (UTC+3:30)
-------------------------------------	--------------------	--

Usage profile description
A long description of the flow. Long description of the flow. Long description of the flow. Long description of the flow. Long description of the flow.

Assigned IAM roles (8)

Find IAM roles

- AmazonSageMakerServiceCatalogProductsCloudformationRole
- GlueRedshiftDevRole
- GlueRedshiftTestRole
- GlueRedshiftTestRole-2
- GlueEMRRole
- GlueEMRDevRole
- GlueTestRole
- GlueAppFlowRole

Assigned IAM users (100)

Find IAM users

- glue-dev-user-1
- glue-dev-user-2
- glue-dev-user-3
- glue-test-user-1
- glue-test-user-2
- glue-test-user-3
- glue-product-user-1
- glue-product-user-1

Parameter configurations for jobs

Number of workers			Worker type	
Default	Minimum	Maximum	Default type	Allowed types
10	1	20	G.2X	-

Timeout (minutes)		
Default	Minimum	Maximum
2880	100	4000

Parameter configurations for sessions

Number of workers			Worker type	
Default	Minimum	Maximum	Default type	Allowed types
10	1	20	G.1X	G.1X, G.4X, G.8X

Timeout (minutes)			Idle timeout (minutes)		
Default	Minimum	Maximum	Default	Minimum	Maximum
2880	100	4000	30	10	200

Tags (3)

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Find tags

Key	Value
Key-1	Value-1
Key-2	Value-2
Key-3	Value-3

Manage tags

사용 프로파일 및 작업

사용 프로파일 포함된 작업 작성

작업을 작성하는 동안에는 사용 프로파일에 설정된 제한 및 기본값이 적용됩니다. 저장 시 작업에 프로파일 이 할당됩니다.

사용 프로파일로 작업 실행

작업 실행을 시작하면 AWS Glue 발신자 프로파일에 설정된 제한이 적용됩니다. 직접 전화를 건 사람이 없는 경우 Glue는 작성자가 작업에 할당한 프로파일의 한도를 적용합니다.

Note

AWS Glue 워크플로우 또는 AWS Glue 트리거에 따라 일정에 따라 작업이 실행되면 작성자가 작업에 할당된 프로파일 이 적용됩니다.

외부 서비스 (Step Functions, MWAA) 또는 StartJobRun API에서 작업을 실행하는 경우 호출자의 프로파일 제한이 적용됩니다.

AWS Glue 워크플로 또는 AWS Glue 트리거의 경우 기존 작업을 업데이트하여 새 프로파일 이름을 저장해야 합니다. 그래야 예약된 실행에 대해 런타임에 프로파일 제한 (최소, 최대, 허용된 작업자 수) 이 적용됩니다.

작업에 할당된 사용 프로파일 보기

작업에 할당된 프로파일 (런타임 시 예약된 AWS Glue 워크플로 또는 AWS Glue 트리거와 함께 사용됨) 을 보려면 작업 세부 정보 탭을 볼 수 있습니다. 작업 실행 세부 정보 탭에서 과거 실행에 사용된 프로파일 을 볼 수도 있습니다.

작업에 첨부된 사용 프로파일 업데이트 또는 삭제

작업에 할당된 프로파일은 업데이트 시 변경됩니다. 작성자에게 사용 프로파일 이 할당되지 않은 경우 이전에 작업에 첨부된 모든 프로파일 이 사용자 프로파일에서 제거됩니다.

AWS Glue Data Catalog 시작하기

AWS Glue Data Catalog는 영구적 기술 메타데이터 스토어입니다. AWS 클라우드에서 메타데이터를 저장 및 공유하고 주석을 다는 데 사용할 수 있는 관리형 서비스입니다. 자세한 내용은 [AWS Glue Data Catalog](#) 단원을 참조하십시오.

AWS Glue 콘솔 및 일부 사용자 인터페이스가 최근에 업데이트되었습니다.

개요

이 자습서를 사용하여 Amazon S3 버킷을 데이터 원본으로 사용하는 첫 번째 AWS Glue 데이터 카탈로그를 생성할 수 있습니다.

이 자습서에서는 AWS Glue 콘솔을 사용하여 다음을 수행합니다.

1. 데이터베이스 생성
2. 테이블 생성
3. Amazon S3 버킷을 데이터 소스로 사용

이 단계를 완료하면 Amazon S3 버킷을 데이터 원본으로 사용하여 AWS Glue 데이터 카탈로그를 채울 수 있습니다.

1단계: 데이터베이스 생성

시작하려면 AWS Management Console에 로그인한 후 [AWS Glue 콘솔](#)을 엽니다.

AWS Glue 콘솔을 사용하여 데이터베이스 생성

1. AWS Glue 콘솔에서 왼쪽 메뉴의 Data catalog(데이터 카탈로그)에서 Databases(데이터베이스)를 선택합니다.
2. 데이터베이스 추가(Add database)를 선택합니다.
3. 데이터베이스 생성 페이지에서 데이터베이스의 이름을 입력합니다. 위치 - 선택 사항 섹션에서 데이터 카탈로그의 클라이언트가 사용할 URI 위치를 설정합니다. 이 정보를 몰라도 데이터베이스 생성을 계속할 수 있습니다.
4. (선택 사항). 데이터베이스에 대한 설명을 입력합니다.
5. 데이터베이스 생성을 선택합니다.

축하합니다. AWS Glue 콘솔을 사용하여 첫 번째 데이터베이스를 설정했습니다. 새 데이터베이스가 사용 가능한 데이터베이스 목록에 나타납니다. 데이터베이스(Databases) 대시보드에서 데이터베이스 이름을 선택하여 데이터베이스를 편집할 수 있습니다.

다음 단계

데이터베이스를 생성하는 다른 방법:

방금 AWS Glue 콘솔을 사용하여 데이터베이스를 생성했지만 데이터베이스를 생성하는 다른 방법이 있습니다.

- 크롤러를 사용하여 자동으로 데이터베이스와 테이블을 생성할 수 있습니다. 크롤러를 사용하여 데이터베이스를 설정하려면 [AWS Glue 콘솔에서 크롤러 작업을 참조하세요](#).
- AWS CloudFormation 템플릿을 사용할 수 있습니다. [AWS Glue Data Catalog 템플릿을 사용하여 AWS Glue 리소스 생성](#)을 참조하세요.
- AWS Glue 데이터베이스 API 작업을 사용하여 데이터베이스를 생성할 수도 있습니다.

create 작업을 사용하여 데이터베이스를 생성하려면 DatabaseInput(필수) 파라미터를 포함하여 요청을 구조화합니다.

예:

다음은 CLI, Boto3 또는 DDL을 사용하여 자습서에서 사용한 S3 버킷의 동일한 flight_data.csv 파일을 기반으로 테이블을 정의하는 방법의 예입니다.

CLI

```
aws glue create-database --database-input "{\"Name\":\"clidb\"}"
```

Boto3

```
glueClient = boto3.client('glue')

response = glueClient.create_database(
    DatabaseInput={
        'Name': 'boto3db'
    }
)
```

데이터베이스 API 데이터 유형, 구조 및 작업에 대한 자세한 내용은 [데이터베이스 API](#)를 참조하세요.

다음 단계

다음 섹션에서는 테이블을 생성하고 데이터베이스에 추가합니다.

데이터 카탈로그에 대한 설정과 권한을 탐색할 수도 있습니다. [AWS Glue 콘솔에서 데이터 카탈로그 설정 관련 작업](#)을 참조하세요.

단계 2. 테이블 생성

이 단계에서는 AWS Glue 콘솔을 사용하여 테이블을 생성합니다.

1. AWS Glue 콘솔에서 왼쪽 메뉴에서 테이블(Tables)을 선택합니다.
2. 테이블 추가를 선택합니다.
3. Table details(테이블 세부 정보)에 테이블 이름을 입력하여 테이블 속성을 설정합니다.
4. Databases(데이터베이스) 섹션의 드롭다운 메뉴에서 1단계에서 생성한 데이터베이스를 선택합니다.
5. Add a data store(데이터 스토어 추가) 섹션에서 S3는 소스 유형으로 기본 선택됩니다.
6. Data is located in(데이터 위치)에서 Specified path in another account(다른 계정의 지정된 경로)를 선택합니다.
7. Include path(포함 경로) 입력 필드의 경로를 복사하여 붙여 넣습니다.


```
s3://crawler-public-us-west-2/flight/2016/csv/
```
8. Data format(데이터 형식) 섹션의 Classification(분류)에서 CSV를 선택하고 Delimiter(구분 기호)에서 comma (,)(쉼표(,))를 선택합니다. 다음을 선택합니다.
9. 스키마를 정의하라는 메시지가 나타납니다. 스키마는 데이터 레코드의 구조와 포맷을 정의합니다. 열 추가(Add column)를 선택합니다. 자세한 내용은 [스키마 레지스트리](#)를 참조하세요.
10. 열 속성을 지정합니다.
 - a. 열 이름을 입력합니다.
 - b. 열 유형(Column type)에 기본적으로 '문자열(string)'이 이미 선택되어 있습니다.
 - c. 열 번호(Column number)에 기본적으로 '1'이 이미 선택되어 있습니다.
 - d. 추가를 선택합니다.
11. 파티션 인덱스를 추가하라는 메시지가 나타납니다. 이는 선택 사항입니다. 이 단계를 건너뛰려면 다음(Next)을 선택합니다.
12. 테이블 속성의 요약이 표시됩니다. 모든 것이 예상대로 표시되면 생성을 선택합니다. 그렇지 않으면 뒤로(Back)를 선택하고 필요에 따라 편집합니다.

축하합니다. 성공적으로 테이블을 수동으로 생성하고 데이터베이스에 연결했습니다. 새로 생성된 테이블은 테이블(Tables) 대시보드에 나타납니다. 대시보드에서 모든 테이블을 수정하고 관리할 수 있습니다.

자세한 내용은 [AWS Glue 콘솔에서 테이블 관련 작업을 참조하세요](#).

다음 단계

다음 단계

이제 데이터 카탈로그가 채워졌으므로 AWS Glue에서 작업 작성을 시작할 수 있습니다. [AWS Glue Studio를 사용하여 시각적 ETL 작업 구축을 참조하세요](#).

콘솔을 사용하는 것 외에도 다음과 같은 다른 방법으로 데이터 카탈로그에서 테이블을 정의할 수 있습니다.

- [크롤러 생성 및 실행](#)
- [AWS Glue의 크롤러로 분류자 추가](#)
- [AWS Glue 테이블 API 사용](#)
- [AWS Glue Data Catalog 템플릿 사용](#)
- [Apache Hive 메타스토어 마이그레이션](#)
- [AWS CLI, Boto3 또는 데이터 정의 언어\(DDL\) 사용](#)

다음은 CLI, Boto3 또는 DDL을 사용하여 자습서에서 사용한 S3 버킷의 동일한 flight_data.csv 파일을 기반으로 테이블을 정의하는 방법의 예입니다.

AWS CLI 명령을 구조화하는 방법은 관련 설명서를 참조하세요. CLI 예제에는 'aws glue create-table --table-input' 값에 대한 JSON 구문이 포함되어 있습니다.

CLI

```
{
  "Name": "flights_data_cli",
  "StorageDescriptor": {
    "Columns": [
      {
        "Name": "year",
        "Type": "bigint"
      },
      {
        "Name": "quarter",
```

```

        "Type": "bigint"
    }
],
"Location": "s3://crawler-public-us-west-2/flight/2016/csv",
"InputFormat": "org.apache.hadoop.mapred.TextInputFormat",
"OutputFormat":
"org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat",
"Compressed": false,
"NumberOfBuckets": -1,
"SerdeInfo": {
    "SerializationLibrary":
"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
    "Parameters": {
        "field.delim": ",",
        "serialization.format": ","
    }
}
},
"PartitionKeys": [
    {
        "Name": "mon",
        "Type": "string"
    }
],
"TableType": "EXTERNAL_TABLE",
"Parameters": {
    "EXTERNAL": "TRUE",
    "classification": "csv",
    "columnsOrdered": "true",
    "compressionType": "none",
    "delimiter": ",",
    "skip.header.line.count": "1",
    "typeOfData": "file"
}
}

```

Boto3

```

import boto3

glue_client = boto3.client("glue")

```

```
response = glue_client.create_table(  
    DatabaseName='sampledb',  
    TableInput={  
        'Name': 'flights_data_manual',  
        'StorageDescriptor': {  
            'Columns': [{  
                'Name': 'year',  
                'Type': 'bigint'  
            }], {  
                'Name': 'quarter',  
                'Type': 'bigint'  
            }  
        }],  
        'Location': 's3://crawler-public-us-west-2/flight/2016/csv',  
        'InputFormat': 'org.apache.hadoop.mapred.TextInputFormat',  
        'OutputFormat':  
        'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat',  
        'Compressed': False,  
        'NumberOfBuckets': -1,  
        'SerdeInfo': {  
            'SerializationLibrary':  
            'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe',  
            'Parameters': {  
                'field.delim': ',',  
                'serialization.format': ','  
            }  
        },  
        'PartitionKeys': [{  
            'Name': 'mon',  
            'Type': 'string'  
        }],  
        'TableType': 'EXTERNAL_TABLE',  
        'Parameters': {  
            'EXTERNAL': 'TRUE',  
            'classification': 'csv',  
            'columnsOrdered': 'true',  
            'compressionType': 'none',  
            'delimiter': ',',  
            'skip.header.line.count': '1',  
            'typeOfData': 'file'  
        }  
    }  
)
```

DDL

```
CREATE EXTERNAL TABLE `sampledb`.`flights_data` (  
  `year` bigint,  
  `quarter` bigint)  
PARTITIONED BY (  
  `mon` string)  
ROW FORMAT DELIMITED  
  FIELDS TERMINATED BY ','  
STORED AS INPUTFORMAT  
  'org.apache.hadoop.mapred.TextInputFormat'  
OUTPUTFORMAT  
  'org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat'  
LOCATION  
  's3://crawler-public-us-west-2/flight/2016/csv/'  
TBLPROPERTIES (  
  'classification'='csv',  
  'columnsOrdered'='true',  
  'compressionType'='none',  
  'delimiter'=',',  
  'skip.header.line.count'='1',  
  'typeOfData'='file')
```

데이터 스토어에 대한 네트워크 액세스 설정

작업을 추출, 변환 및 로드(ETL)하려면 AWS Glue는 데이터 스토어에 액세스할 수 있어야 합니다. 작업이 Virtual Private Cloud(VPC) 서브넷에서 실행될 필요가 없으면(예를 들어, 데이터를 Amazon S3에서 Amazon S3로 변환하는 것) 추가 구성이 필요 없습니다.

VPC 서브넷에서 작업을 실행해야 하는 경우(예: 프라이빗 서브넷의 JDBC 데이터 스토어에서 데이터 변환) AWS Glue는 작업이 VPC 내의 다른 리소스에 안전하게 연결할 수 있도록 [탄력적 네트워크 인터페이스](#)를 설정합니다. 지정한 서브넷의 IP 주소 범위에 속하는 프라이빗 IP 주소가 이 탄력적 네트워크 인터페이스에 할당됩니다. 퍼블릭 IP 주소가 할당되지 않습니다. AWS Glue 연결에 지정된 보안 그룹은 탄력적 네트워크 인터페이스 각각에 적용됩니다. 자세한 내용은 [AWS Glue에서 Amazon RDS 데이터 스토어에 대해 JDBC를 연결하도록 Amazon VPC 설정](#) 단원을 참조하십시오.

작업이 액세스한 모든 JDBC 데이터 스토어는 VPC 서브넷에서 사용 가능해야 합니다. VPC 안에서 Amazon S3에 액세스하려면 [VPC 엔드포인트](#)가 필요합니다. 작업에서 VPC 리소스와 퍼블릭 인터넷에 모두 액세스해야 할 경우 VPC 내부 네트워크 주소 변환(NAT) 게이트웨이가 VPC에 있어야 합니다.

작업 또는 개발 엔드포인트는 한 번에 하나의 VPC(와 서브넷)에만 액세스할 수 있습니다. 다른 VPC의 데이터 스토어로 액세스하려면 다음과 같은 옵션이 있습니다.

- VPC 피어링을 사용하여 데이터 스토어에 액세스합니다. VPC 피어링에 대한 자세한 내용은 [VPC 피어링 기초](#) 단원을 참조하십시오.
- Amazon S3 버킷을 중개 스토리지 위치로 사용합니다. 작업 1 출력값인 Amazon S3를 작업 2의 입력값으로 하여 작업을 두 가지로 분할합니다.

Amazon VPC를 사용하여 Amazon Redshift 데이터 스토어에 연결하는 방법에 대한 자세한 내용은 [the section called “Redshift 구성”](#) 섹션을 참조하세요.

Amazon VPC를 사용하여 Amazon RDS 데이터 스토어에 연결하는 방법에 대한 자세한 내용은 [the section called “Amazon RDS 데이터 스토어에 연결하도록 Amazon VPC 설정”](#) 섹션을 참조하세요.

Amazon VPC에 필요한 규칙을 설정한 후 데이터 스토어에 연결하는 데 필요한 속성을 사용하여 AWS Glue에서 연결을 생성합니다. 연결에 대한 자세한 정보는 [데이터에 연결](#) 섹션을 참조하세요.

Note

AWS Glue DNS 환경을 설정하도록 합니다. 자세한 내용은 [VPC에서 DNS 설정](#) 단원을 참조하십시오.

주제

- [AWS Glue에서 PyPI에 연결하도록 VPC 설정](#)
- [VPC에서 DNS 설정](#)

AWS Glue에서 PyPI에 연결하도록 VPC 설정

Python Package Index(PyPI)는 Python 프로그래밍 언어용 소프트웨어 리포지토리입니다. 이 주제에서는 pip 설치 패키지(세션 생성자가 `--additional-python-modules` 플래그를 사용하여 지정함) 사용을 지원하는 데 필요한 세부 정보를 다룹니다.

커넥터와 함께 AWS Glue 대화형 세션을 사용하면 커넥터에 지정된 서브넷을 통해 VPC 네트워크를 사용하게 됩니다. 따라서 특별 구성을 설정하지 않으면 AWS 서비스 및 기타 네트워크 대상을 사용할 수 없습니다.

이 문제의 해결 방법은 다음과 같습니다.

- 세션에서 연결할 수 있는 인터넷 게이트웨이 사용.
- 패키지 세트의 종속성에서 전이 폐쇄 특성을 포함하는 PyPI/simple 리포지토리로 S3 버킷의 설정 및 사용.
- PyPI를 미러링하고 VPC에 연결하는 CodeArtifact 리포지토리 사용.

인터넷 게이트웨이 설정

기술적 측면은 [NAT 게이트웨이 사용 사례](#)에 자세히 설명되어 있지만 `--additional-python-modules` 사용에 관한 다음 요구 사항도 참고하세요. 특히, VPC의 구성에 따라 결정되는 `pypi.org`에 대한 액세스 권한이 `--additional-python-modules`에 필요합니다. 다음과 같은 요구 사항을 확인합니다.

1. 사용자 세션의 경우 `pip install`을 통해 추가 python 모듈을 설치해야 합니다. 세션에서 커넥터를 사용하는 경우 구성이 영향을 받을 수 있습니다.
2. `--additional-python-modules`에서 커넥터를 사용할 때 세션이 시작되면 커넥터의 `PhysicalConnectionRequirements`에 연결된 서브넷이 `pypi.org`에 도달하기 위한 네트워크 경로를 제공해야 합니다.
3. 구성이 올바른지 확인해야 합니다.

대상 PyPI/simple 리포지토리를 호스팅하도록 Amazon S3 버킷 설정

이 예제는 Amazon S3에서 패키지 세트 및 해당 종속성에 대한 PyPI 미러를 설정합니다.

패키지 세트에 대한 PyPI 미러를 설정하려면:

```
# pip download all the dependencies
pip download -d s3pypi --only-binary :all: plotly ggplot
pip download -d s3pypi --platform manylinux_2_17_x86_64 --only-binary :all: pycopg2-binary
# create and upload the pypi/simple index and wheel files to the s3 bucket
s3pypi -b test-domain-name --put-root-index -v s3pypi/*
```

기존 아티팩트 리포지토리가 이미 있는 경우, 위와 같이 Amazon S3 버킷의 예제 URL 대신 제공할 수 있는 pip 사용을 위한 인덱스 URL이 있을 것입니다.

몇 가지 예제 패키지에서 사용자 지정 index-url을 사용하려면:

```
%%configure
{
  "--additional-python-modules": "psycopy2_binary==2.9.5",
  "python-modules-installer-option": "--no-cache-dir --verbose --index-url https://
test-domain-name.s3.amazonaws.com/ --trusted-host test-domain-name.s3.amazonaws.com"
}
```

VPC에 연결된 pypi의 CodeArtifact 미리 설정

미러를 설정하려면:

1. 커넥터에서 사용하는 서브넷과 동일한 리전에서 리포지토리를 생성합니다.

Public upstream repositories를 선택하고 pypi-store를 선택합니다.

2. 서브넷에 VPC에서 리포지토리로의 액세스를 제공합니다.

3. python-modules-installer-option을 사용하여 올바른 --index-url을 지정합니다.

```
%%configure
{
  "--additional-python-modules": "psycopy2_binary==2.9.5",
  "python-modules-installer-option": "--no-cache-dir --verbose --index-url https://
test-domain-name.s3.amazonaws.com/ --trusted-host test-domain-name.s3.amazonaws.com"
}
```

자세한 내용은 [VPC에서 CodeArtifact 사용](#)을 참조하세요.

VPC에서 DNS 설정

도메인 이름 시스템(DNS)은 인터넷에서 사용되는 이름을 해당 IP 주소로 확인할 때 기준이 됩니다.

DNS 호스트 이름은 컴퓨터를 고유적으로 지정하는 이름으로서, 호스트 이름과 도메인 이름으로 구성됩니다. DNS 서버는 DNS 호스트 이름을 해당 IP 주소로 확인합니다.

VPC에 DNS를 설정하려면 DNS 호스트 이름과 DNS 확인이 모두 VPC에서 활성화되었는지 확인합니다. VPC 네트워크 속성 enableDnsHostnames 및 enableDnsSupport는 true로 설정되어 있어야

합니다. 이러한 속성을 보고 수정하려면 <https://console.aws.amazon.com/vpc/>의 VPC 콘솔로 이동합니다.

자세한 내용은 [VPC에서 DNS 사용하기](#) 단원을 참조하세요. 또한 AWS CLI를 사용하고 [modify-vpc-attribute](#) 명령을 호출하여 VPC 네트워크 속성을 구성할 수 있습니다.

Note

Route 53를 사용한다면 구성이 DNS 네트워크 속성을 재정의하지 않도록 확인합니다.

AWS Glue에서 암호화 설정

다음 예제 작업 흐름에는 AWS Glue로 암호화를 사용할 때 구성하는 옵션이 강조되어 있습니다. 이 예제는 특정 AWS Key Management Service(AWS KMS) 키 사용을 설명하지만 사용자의 특정 요구에 따라 다른 설정을 선택할 수도 있습니다. 이 작업 흐름에는 AWS Glue 설정 시 암호화에 관한 옵션만 강조 표시되어 있습니다.

1. AWS Glue 콘솔 사용자가 모든 AWS Glue API 작업(예: "glue:*")을 허용하는 권한 정책을 사용하지 않을 경우에는 다음 작업이 허용되는지 확인하십시오.
 - "glue:GetDataCatalogEncryptionSettings"
 - "glue:PutDataCatalogEncryptionSettings"
 - "glue:CreateSecurityConfiguration"
 - "glue:GetSecurityConfiguration"
 - "glue:GetSecurityConfigurations"
 - "glue>DeleteSecurityConfiguration"
2. 암호화된 카탈로그에 액세스하거나 이에 쓰는 모든 클라이언트(즉, 콘솔 사용자, 크롤러, 작업 또는 개발 엔드포인트)에는 다음 권한이 필요합니다.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt",
      "kms:Encrypt"
    ]
  }
}
```

```

    "Resource": "<key-arns-used-for-data-catalog>"
  }
}

```

3. 암호화된 연결 암호에 액세스하는 사용자 또는 역할은 다음 권한이 필요합니다.

```

{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt"
    ],
    "Resource": "<key-arns-used-for-password-encryption>"
  }
}

```

4. 암호화된 데이터를 Amazon S3에 기록하는 추출, 변환, 로드 작업 역할은 다음 권한이 있어야 합니다.

```

{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:Encrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": "<key-arns-used-for-s3>"
  }
}

```

5. 암호화된 Amazon CloudWatch Logs를 기록하는 ETL 작업 또는 크롤러는 키 및 IAM 정책에 다음 권한이 있어야 합니다.

키 정책(IAM 정책 아님)에서 다음을 수행합니다.

```

{
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.region.amazonaws.com"
  },

```

```

"Action": [
  "kms:Encrypt*",
  "kms:Decrypt*",
  "kms:ReEncrypt*",
  "kms:GenerateDataKey*",
  "kms:Describe*"
],
"Resource": "<arn of key used for ETL/crawler cloudwatch encryption>"
}

```

키 정책에 대한 자세한 내용은 AWS Key Management Service Developer Guide의 [Using Key Policies in AWS KMS](#)를 참조하세요.

IAM 정책에서 `logs:AssociateKmsKey` 권한을 첨부합니다.

```

{
  "Effect": "Allow",
  "Principal": {
    "Service": "logs.region.amazonaws.com"
  },
  "Action": [
    "logs:AssociateKmsKey"
  ],
  "Resource": "<arn of key used for ETL/crawler cloudwatch encryption>"
}

```

6. 암호화된 작업 북마크를 사용하는 ETL 작업은 다음 권한이 있어야 합니다.

```

{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:Encrypt"
    ],
    "Resource": "<key-arns-used-for-job-bookmark-encryption>"
  }
}

```

7. AWS Glue 콘솔의 탐색 창에서 설정을 선택합니다.

- a. [데이터 카탈로그 설정(Data catalog settings)] 페이지에서 [메타데이터 암호화(Metadata encryption)]를 선택하여 Data Catalog를 암호화합니다. 이 옵션은 사용자가 선택한 AWS KMS 키로 Data Catalog의 모든 객체를 암호화합니다.
- b. AWS KMS 키의 경우 aws/glue를 선택합니다. 사용자가 생성한 AWS KMS 키를 선택할 수도 있습니다.

⚠ Important

AWS Glue에서는 대칭 고객 마스터 키(CMK)만 지원합니다. AWS KMS 키(key) 목록에는 대칭 키만 표시됩니다. 그러나 Choose a AWS KMS key ARN(KMS 키 ARN 선택)을 선택하면 콘솔에서 모든 키 유형의 ARN을 입력할 수 있습니다. 대칭 키에 대한 ARN만 입력해야 합니다.

암호화가 활성화된 경우 Data Catalog에 액세스하는 클라이언트는 AWS KMS 권한이 있어야 합니다.

8. 탐색 창에서 보안 구성을 선택합니다. 보안 구성은 AWS Glue 프로세스를 구성할 때 사용할 수 있는 보안 속성 세트입니다. 그런 다음 보안 구성 추가를 선택합니다. 구성에서 다음 옵션 중 하나를 선택합니다.
 - a. S3 암호화를 선택합니다. 암호화 모드로 SSE-KMS를 선택합니다. AWS KMS키는 aws/s3을 선택합니다(사용자에게 이 키를 사용할 권한이 있어야 합니다). 이제 작업이 Amazon S3에 기록한 데이터로 AWS 관리형 AWS Glue AWS KMS 키를 사용할 수 있습니다.
 - b. CloudWatch Logs 암호화를 선택하고 CMK를 선택합니다. 사용자에게 이 키를 사용할 권한이 있어야 합니다. 자세한 내용은 AWS Key Management Service Developer Guide의 [Encrypt Log Data in CloudWatch Logs Using AWS KMS](#)를 참조하세요.

⚠ Important

AWS Glue에서는 대칭 고객 마스터 키(CMK)만 지원합니다. AWS KMS 키(key) 목록에는 대칭 키만 표시됩니다. 그러나 Choose a AWS KMS key ARN(KMS 키 ARN 선택)을 선택하면 콘솔에서 모든 키 유형의 ARN을 입력할 수 있습니다. 대칭 키에 대한 ARN만 입력해야 합니다.

- c. 고급 속성을 선택하고 작업 북마크 암호화를 선택합니다. AWS KMS키는 aws/glue를 선택합니다(사용자에게 이 키를 사용할 권한이 있어야 합니다). 이제 Amazon S3에 기록된 작업 북마크를 AWS Glue AWS KMS 키로 암호화할 수 있습니다.

9. 탐색 창에서 연결을 선택합니다.

- a. 연결 추가를 선택하여 ETL 작업의 대상인 Java Database Connectivity(JDBC) 데이터 스토어에 대한 연결을 생성합니다.
- b. Secure Sockets Layer(SSL) 암호화를 사용하도록 강제하려면 Require SSL connection(SSL 연결 요구)을 선택하고 연결을 테스트합니다.

10. 탐색 창에서, 작업을 선택합니다.

- a. 작업 추가를 선택하여 데이터를 변환하는 작업을 만듭니다.
- b. 작업 정의에서 생성한 보안 구성을 선택합니다.

11. AWS Glue 콘솔에서 온디맨드 작업을 실행합니다. 작업에서 작성한 Amazon S3 데이터, 작업에서 작성한 CloudWatch Logs 및 작업 북마크가 모두 암호화되어 있는지 확인합니다.

AWS Glue의 개발에 대한 네트워킹 설정

AWS Glue로 추출, 변환, 로드(ETL) 스크립트를 실행하기 위해서 개발 엔드포인트를 사용하여 스크립트를 개발하고 테스트할 수 있습니다. 개발 엔드포인트는 AWS Glue 버전 2.0 작업과 함께 사용하도록 지원되지 않습니다. 버전 2.0 이상의 경우 선호하는 개발 방법은 AWS Glue 커널 중 하나에서 Jupyter Notebook을 사용하는 것입니다. 자세한 내용은 [the section called “AWS Glue 대화형 세션 시작하기”](#) 단원을 참조하십시오.

개발 엔드포인트에 대한 네트워크 설정

개발 엔드포인트를 설정할 때 Virtual Private Cloud(VPC), 서브넷 및 보안 그룹을 지정합니다.

Note

AWS Glue DNS 환경을 설정하도록 합니다. 자세한 내용은 [VPC에서 DNS 설정](#) 단원을 참조하십시오.

AWS Glue를 가능하게 하여 필요한 리소스에 액세스하기 위해서는 행을 서브넷 라우팅 테이블에 추가하여 Amazon S3 접두사 목록과 VPC 엔드포인트를 연결합니다. 접두사 목록 ID는 VPC 트래픽을 허용하여 VPC 엔드포인트를 통해 AWS 서비스로 액세스할 수 있는 아웃바운드 보안 그룹을 생성하는 데 필요합니다. 로컬 시스템에서 이 개발 엔드포인트와 연결된 노트북 서버를 쉽게 연결하려면 행을 라우팅 테이블에 추가하여 인터넷 게이트웨이 ID를 추가합니다. 자세한 내용은 [VPC 엔드포인트](#)를 참조하십시오. 서브넷 라우팅 테이블을 업데이트하여 다음 테이블과 비슷하게 만듭니다.

대상 주소	대상		
10.0.0.0/16	로컬		
Amazon S3용 pl-id	vpce-id		
0.0.0.0/0	igw-xxxx		

AWS Glue가 구성 요소 간에 통신하려면 보안 그룹이 모든 TCP 포트에 자기 참조 인바운드 규칙을 지정해야 합니다. 자기 참조 규칙을 생성하여 VPC에서 동일한 보안 그룹으로 소스를 제한하고 모든 네트워크로 공개되지 않도록 합니다. VPC의 기본 보안 그룹은 ALL Traffic의 자기 참조 인바운드 액세스 규칙이 있습니다.

보안 그룹을 설정하려면

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/ec2/>에서 Amazon EC2 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 [Security Groups]를 선택합니다.
3. 목록에서 기존 보안 그룹을 선택하거나 [Create Security Group]를 선택하여 개발 엔드포인트와 함께 사용합니다.
4. 보안 그룹 화면에서 [Inbound] 탭으로 이동합니다.
5. 자기 참조 규칙을 추가하여 AWS Glue 구성 요소를 허용하여 통신합니다. 특히, [Type(유형)] All TCP의 규칙을 추가하고 확인합니다. [Protocol(프로토콜)]은 TCP이고 [Port Range(포트 범위)]는 모든 포트를 포함하고 포트의 [Source(원본)]은 [Group ID(그룹 ID)]이라는 동일한 보안 그룹입니다.

인바운드 규칙은 다음과 비슷하게 보입니다.

유형	프로토콜	포트 범위	소스
모든 TCP	TCP	0~65535	<i>security-group</i>

다음은 자기 참조 인바운드 규칙 예제입니다.

6. 아웃바운드 트래픽 규칙도 추가합니다. 아웃바운드 트래픽을 모든 포트로 열거나 자기 [Type(유형)] All TCP 의 자기 참조 규칙을 생성합니다. [Protocol(프로토콜)]는 TCP이고 [Port Range(포트 범위)]는 모든 포트를 포함하고 포트의 [Source(원본)]은 [Group ID(그룹 ID)]이라는 동일한 보안 그룹입니다.

아웃바운드 규칙은 다음 규칙 중 하나와 비슷합니다.

유형	프로토콜	포트 범위	대상
모든 TCP	TCP	0~65535	<i>security-group</i>
모든 트래픽	ALL	ALL	0.0.0.0/0

노트북 서버용 Amazon EC2 설정

개발 엔드포인트를 통해 노트북 서버를 생성하여 Jupyter 노트북에서 ETL 스크립트를 테스트합니다. 노트북 통신을 가능하게 하려면 HTTPS(port 443)와 SSH(port 22)의 인바운드 규칙으로 보안 그룹을 지정합니다. 규칙의 소스가 0.0.0.0/0 또는 노트북에 연결하는 시스템의 IP 주소인지 확인합니다.

보안 그룹을 설정하려면

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/ec2/>에서 Amazon EC2 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 [Security Groups]를 선택합니다.
3. 목록에서 기존 보안 그룹을 선택하거나 [Create Security Group]를 선택하여 노트북 서버와 함께 사용합니다. 개발 엔드포인트와 연결된 보안 그룹은 노트북 서버를 생성하는 데도 사용할 수 있습니다.
4. 보안 그룹 화면에서 [Inbound] 탭으로 이동합니다.
5. 이와 비슷한 인바운드 규칙을 추가합니다.

유형	프로토콜	포트 범위	소스
SSH	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

다음은 보안 그룹에 대한 인바운드 규칙 예제를 소개합니다.

Security Group: sg-19e1b768

Description

Inbound

Outbound

Tags

Edit

Type <small>i</small>	Protocol <small>i</small>	Port Range <small>i</small>	Source <small>i</small>
SSH	TCP	22	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0

AWS Glue에서 데이터 검색 및 카탈로그 작성

AWS Glue Data Catalog는 조직의 데이터 세트에 대한 메타데이터를 저장하는 중앙 집중식 리포지토리입니다. 즉, 데이터 소스의 위치, 스키마 및 런타임 메트릭에 대한 인덱스로 동작합니다. 메타데이터는 메타데이터 테이블에 저장되며, 여기서 각 테이블은 단일 데이터 스토어를 나타냅니다.

데이터 소스를 자동으로 스캔하고 메타데이터를 추출하는 크롤러를 사용하여 데이터 카탈로그를 채울 수 있습니다. 크롤러는 AWS 내부(AWS 기반) 및 외부 데이터 소스에 연결할 수 있습니다.

지원되는 데이터 소스에 대한 자세한 내용은 [어떤 데이터 스토어를 크롤할 수 있나요?](#) 섹션을 참조하세요.

특정 요구 사항에 따라 테이블 구조, 스키마 및 파티셔닝 구조를 정의하여 데이터 카탈로그에서 수동으로 테이블을 생성할 수도 있습니다.

수동으로 메타데이터 테이블을 생성하는 것에 대한 자세한 내용은 [수동으로 메타데이터 정의](#) 섹션을 참조하세요.

데이터 카탈로그에서 이 정보를 사용하여 ETL 작업을 생성하고 모니터링할 수 있습니다. 데이터 카탈로그는 다른 AWS 분석 서비스와 통합되어 데이터 소스에 대한 통합 보기를 제공하므로 데이터를 더 쉽게 관리하고 분석할 수 있습니다.

- Amazon Athena - SQL을 사용하여 Amazon S3 데이터에 대한 테이블 메타데이터를 데이터 카탈로그에 저장하고 쿼리합니다.
- AWS Lake Formation - 세분화된 데이터 액세스 정책을 중앙에서 정의하고 관리하며 데이터 액세스를 감사합니다.
- Amazon EMR - 빅 데이터 처리를 위해 데이터 카탈로그에 정의된 데이터 소스에 액세스합니다.
- Amazon SageMaker - 기계 학습 모델을 빠르고 확실하게 구축하고, 학습시키고, 배포합니다.

데이터 카탈로그의 주요 기능

다음은 데이터 카탈로그의 주요 기능입니다.

메타데이터 리포지토리

데이터 카탈로그는 중앙 메타데이터 리포지토리로 작동하며, 데이터 소스의 위치, 스키마 및 속성에 대한 정보를 저장합니다. 이 메타데이터는 기존 관계형 데이터베이스 카탈로그와 유사하게 데이터베이스 및 테이블로 구성됩니다.

자동 데이터 검색 기능

AWS Glue 크롤러는 새 데이터 소스나 업데이트된 데이터 소스를 자동으로 검색하고 카탈로그를 작성하여 수동 메타데이터 관리에 따른 오버헤드를 줄이고 데이터 카탈로그가 최신 상태로 유지되도록 보장합니다. 데이터 카탈로그는 데이터 소스의 카탈로그를 작성함으로써 사용자와 애플리케이션이 조직 내에서 사용 가능한 데이터 자산을 더 쉽게 발견하고 이해할 수 있도록 도와주며 데이터 재사용과 협업을 촉진시킵니다.

데이터 카탈로그는 Amazon S3, Amazon RDS, Amazon Redshift, Apache Hive 등을 비롯한 매우 다양한 데이터 소스를 지원합니다. 또한 AWS Glue 크롤러를 사용하여 이러한 소스에서 메타데이터를 자동으로 추론하고 저장할 수 있습니다.

자세한 내용은 [크롤러를 사용하여 데이터 카탈로그 채우기](#) 단원을 참조하십시오.

스키마 관리

데이터 카탈로그는 스키마 추론, 진화 및 버전 관리를 포함하여 데이터 소스의 스키마를 자동으로 캡처하고 관리합니다. 데이터 카탈로그에서 AWS Glue ETL 작업을 사용하여 스키마 및 파티션을 업데이트할 수 있습니다.

테이블 최적화

AWS 분석 서비스(예: Amazon Athena 및 Amazon EMR)와 AWS Glue ETL 작업에서 읽기 성능을 향상시키기 위해 데이터 카탈로그는 데이터 카탈로그의 Iceberg 테이블에 대해 관리형 압축(작은 Amazon S3 객체를 큰 객체로 압축하는 프로세스)을 제공합니다. AWS Glue 콘솔, AWS Lake Formation 콘솔, AWS CLI 또는 AWS API를 사용하여 데이터 카탈로그에 있는 개별 Iceberg 테이블에 대한 압축을 활성화하거나 비활성화할 수 있습니다.

자세한 내용은 [Iceberg 테이블 최적화](#) 단원을 참조하십시오.

열 통계값

추가 데이터 파이프라인을 설정하지 않고도 Parquet, ORC, JSON, ION, CSV 및 XML과 같은 데이터 형식의 데이터 카탈로그 테이블에 대한 열 수준 통계를 계산할 수 있습니다. 열 통계는 열 내 값에 대한 통찰력을 얻어 데이터 프로필을 이해하는 데 도움이 됩니다. 데이터 카탈로그는 최소값, 최대값, 총 null 값, 총 고유 값, 값의 평균 길이, 실제 값의 총 발생 횟수 등과 같은 열 값에 대한 통계 생성을 지원합니다.

자세한 내용은 [열 통계를 사용한 쿼리 성능 최적화](#) 단원을 참조하십시오.

데이터 계보

데이터 카탈로그는 데이터에 대해 수행된 변환 및 작업의 기록을 유지 관리하며, 데이터 계보 정보를 제공합니다. 이 계보 정보는 감사, 규정 준수 및 데이터 출처 이해에 유용합니다.

다른 AWS 서비스와의 통합

데이터 카탈로그는 AWS Lake Formation, Amazon Athena, Amazon Redshift Spectrum, Amazon EMR 등과 같은 다른 AWS 서비스와 원활하게 통합됩니다. 이 통합을 통해 일관된 단일 메타데이터 계층을 사용하여 다양한 데이터 스토어 간에서 데이터를 쿼리하고 분석할 수 있습니다.

보안 및 액세스 제어

AWS Glue는 AWS Lake Formation과 통합되어 Data Catalog 리소스에 대한 세분화된 액세스 제어를 지원하므로 조직의 정책 및 요구 사항에 따라 데이터 자산에 대한 사용 권한을 관리하고 액세스를 보호할 수 있습니다. AWS Glue는 AWS Key Management Service(AWS KMS)와 통합되어 데이터 카탈로그에 저장된 메타데이터를 암호화합니다.

주제

- [AWS Glue 데이터 카탈로그 채우기](#)
- [트랜잭션 테이블 채우기 및 관리](#)
- [데이터 카탈로그 관리](#)
- [데이터 카탈로그 액세스](#)
- [AWS Glue 데이터 카탈로그 모범 사례](#)
- [AWS Glue Schema Registry](#)

AWS Glue 데이터 카탈로그 채우기

다음 방법을 사용하여 AWS Glue Data Catalog를 채울 수 있습니다.

- AWS Glue 크롤러 - AWS Glue 크롤러를 통해 데이터베이스, 데이터 레이크 및 스트리밍 데이터와 같은 데이터 소스를 자동으로 검색하고 카탈로그화할 수 있습니다. 크롤러는 매우 다양한 데이터 소스의 메타데이터를 자동으로 검색하고 추론할 수 있으므로 데이터 카탈로그를 채울 때 가장 일반적이고 권장되는 방법입니다.
- 메타데이터 수동 추가 - AWS Glue 콘솔, Lake Formation 콘솔, AWS CLI 또는 AWS Glue API를 사용하여 데이터베이스, 테이블 및 연결 세부 정보를 수동으로 정의하고 데이터 카탈로그에 추가할 수 있습니다. 수동 입력은 크롤링할 수 없는 데이터 소스의 카탈로그를 작성하려는 경우에 유용합니다.
- 다른 AWS 서비스와의 통합 - AWS Lake Formation 및 Amazon Athena와 같은 서비스의 메타데이터로 데이터 카탈로그를 채울 수 있습니다. 이러한 서비스는 데이터 카탈로그에서 데이터 소스를 검색하고 등록할 수 있습니다.

- 기존 메타데이터 리포지토리에서 채우기 - Apache Hive Metastore와 같은 기존 메타데이터 저장소가 있는 경우 AWS Glue를 사용하여 해당 메타데이터를 데이터 카탈로그로 가져올 수 있습니다. 자세한 내용은 GitHub의 [Hive 메타스토어와 AWS Glue Data Catalog 간의 마이그레이션](#)을 참조하세요.

주제

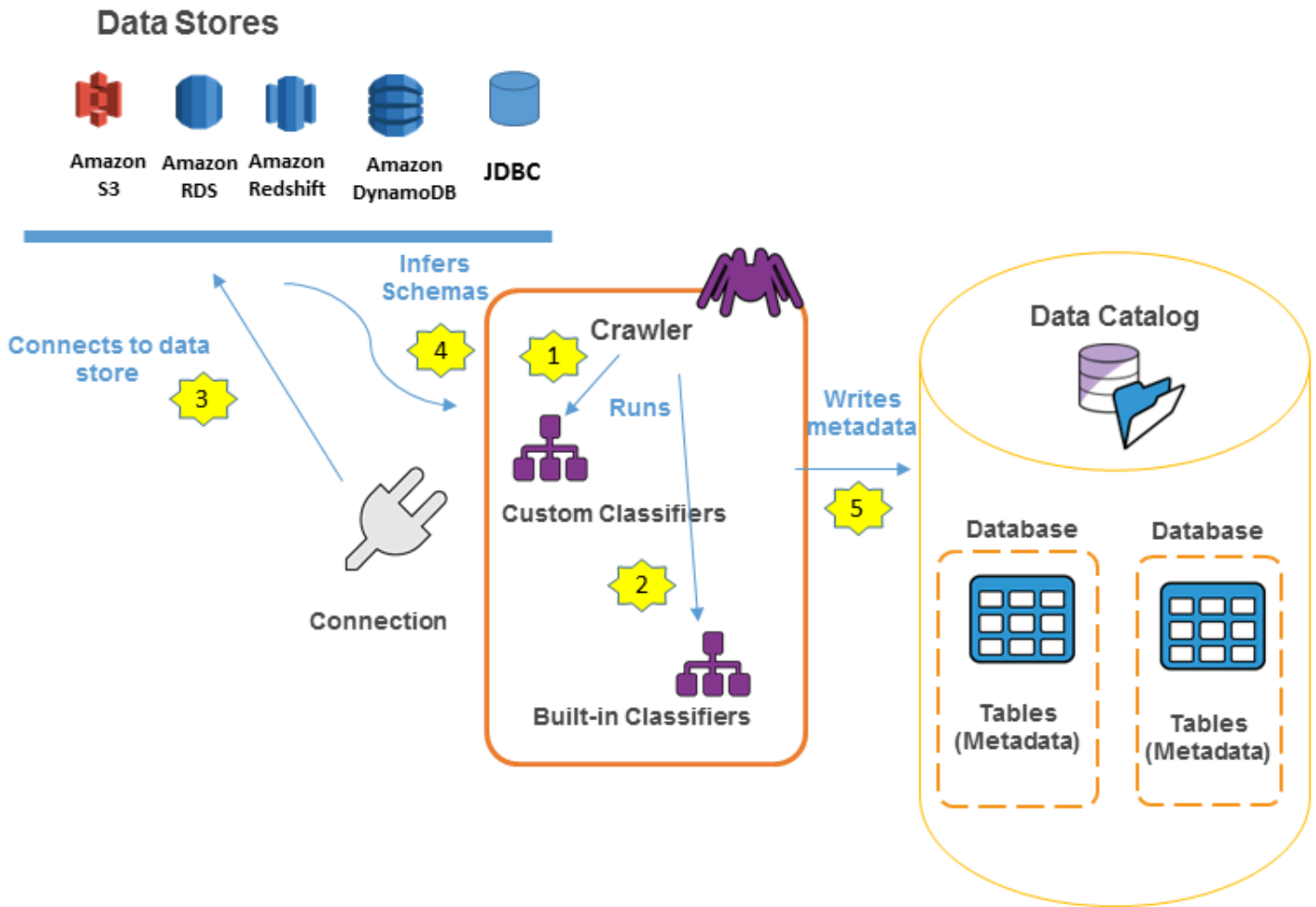
- [크롤러를 사용하여 데이터 카탈로그 채우기](#)
- [수동으로 메타데이터 정의](#)
- [다른 AWS 서비스와 통합](#)
- [데이터 카탈로그 설정](#)

크롤러를 사용하여 데이터 카탈로그 채우기

AWS Glue 크롤러를 사용하면 데이터베이스 및 테이블을 사용하여 AWS Glue Data Catalog를 채울 수 있습니다. 대부분의 AWS Glue 사용자가 사용하는 기본적인 방법입니다. 크롤러는 단일 실행으로 여러 데이터 스토어를 크롤할 수 있습니다. 완료 시 크롤러는 데이터 카탈로그에서 하나 이상의 테이블을 생성하거나 업데이트합니다. AWS Glue에서 정의한 추출, 변환, 로드 작업은 이러한 데이터 카탈로그 테이블을 원본 및 대상으로 사용합니다. ETL 작업은 원본 및 대상 데이터 카탈로그 테이블에 지정된 데이터 스토어에서 읽기와 쓰기를 수행합니다.

워크플로

다음 워크플로 다이어그램은 AWS Glue 크롤러가 데이터 스토어와 다른 요소와 상호 작용하여 Data Catalog를 채우는 방법을 보여줍니다.



다음은 크롤러가 AWS Glue Data Catalog을 채우는 방법에 대한 일반적인 워크플로입니다.

1. 크롤러는 선택한 사용자 지정 분류자를 실행하여 데이터의 형식 및 스키마를 추론합니다. 사용자 지정 분류자에 대한 코드를 제공하면 지정한 순서대로 실행됩니다.

첫 번째 사용자 분류자가 데이터 구조를 성공적으로 인식하는 과정은 테이블의 스키마를 생성하는데 사용됩니다. 하위 목록에 있는 사용자 분류자는 건너뜁니다.

2. 어떠한 사용자 지정 분류자도 데이터 스키마와 일치하지 않는다면 기본 설정 분류자는 데이터 스키마를 인식할 시도를 합니다. 기본 설정 분류자의 예는 JSON을 인식하는 분류자입니다.
3. 크롤러를 데이터 스토어로 연결합니다. 어떤 데이터 스토어는 크롤러 액세스 연결 속성을 요구합니다.
4. 추론된 스키마는 데이터 때문에 생성됩니다.

5. 크롤러는 메타데이터를 Data Catalog로 작성합니다. 테이블 정의는 데이터 스토어의 데이터에 대한 메타데이터를 포함합니다. 테이블은 Data Catalog에서 테이블 컨테이너인 데이터베이스에 작성됩니다. 테이블 속성은 테이블 스키마를 추론한 분류자에 의해 생성된 라벨인 분류자를 포함합니다.

주제

- [크롤러 작동 방식](#)
- [어떤 데이터 스토어를 크롤할 수 있나요?](#)
- [크롤러는 파티션 생성 시기를 어떻게 결정하나요?](#)
- [크롤러 사전 조건](#)
- [크롤러 구성](#)
- [AWS Glue의 크롤러로 분류자 추가](#)
- [AWS Glue 크롤러 일정 관리](#)
- [크롤러 결과 및 세부 정보 보기](#)
- [크롤러 동작 사용자 지정](#)
- [튜토리얼: AWS Glue 크롤러 추가](#)

크롤러 작동 방식

크롤러가 실행되면 데이터 스토어에서 정보를 얻기 위한 다음 작업을 실행합니다.

- 데이터를 분류하여 원시 데이터의 포맷, 스키마 및 관련 속성 결정 - 분류 결과는 사용자 정의 분류자를 생성하여 구성할 수 있습니다.
- 데이터를 테이블 혹은 파티션으로 분류합니다. - 데이터는 크롤러 발견을 기반으로 분류합니다.
- 메타데이터를 데이터 카탈로그에 작성합니다 - 크롤러가 어떻게 테이블과 파티션을 추가하고 업데이트, 삭제하는지 구성합니다.

크롤러를 정의할 때 스키마를 추론할 수 있도록 데이터 포맷을 평가하는 하나 이상의 분류자를 선택합니다. 크롤러가 실행되면 목록의 첫 번째 분류자가 성공적으로 데이터 스토어를 인식하고 테이블의 스키마를 생성합니다. 기본 제공 분류자를 사용하거나 사용자가 직접 정의할 수 있습니다. 크롤러를 정의하기 전에 별도의 작업에서 사용자 지정 분류자를 정의합니다. AWS Glue는 기본 제공 분류자를 제공하여 일반 파일에서 JSON, CSV 및 Apache Avro를 포함하는 포맷으로 스키마를 추론합니다. AWS Glue의 기본 제공 분류자의 현재 목록은 [AWS Glue의 기본 설정 분류자](#) 섹션을 참조하세요.

크롤러가 생성하는 메타데이터는 크롤러를 정의할 때 데이터베이스에 포함됩니다. 크롤러가 데이터베이스를 지정하지 않으면 테이블은 기본 데이터베이스로 배치합니다. 또한, 각 테이블은 처음으로 데이터 스토어를 성공적으로 인식하는 분류자로 채워진 분류 열이 있습니다.

크롤된 파일이 압축되면 크롤러는 반드시 다운로드하고 실행해야 합니다. 크롤러가 실행되면 크롤러는 파일 정보를 얻어 파일 포맷 및 압축 유형을 결정하고 파일 속성을 데이터 카탈로그에 작성합니다. Apache Parquet과 같은 일부 파일 형식은 파일이 작성한 대로 파일 일부를 압축할 수 있습니다. 이런 파일의 압축 데이터는 파일의 내부 구성 요소이고, AWS Glue는 데이터 카탈로그에 테이블을 쓸 때 `compressionType` 속성을 채우지 않습니다. 반대로 전체 파일이 gzip처럼 압축 알고리즘을 통해 압축된 후 `compressionType` 속성은 테이블이 데이터 카탈로그에 작성될 때 채워집니다.

크롤러는 생성하는 테이블 이름을 만듭니다. AWS Glue Data Catalog에 저장된 테이블 이름은 다음 규칙을 따릅니다.

- 영숫자와 밑줄(_)만 허용됩니다.
- 사용자 지정 접두사는 64자보다 길 수 없습니다.
- 이름 최대 길이는 128자보다 길 수 없습니다. 크롤러는 이름이 제한 범위 내에 있도록 이름을 줄입니다.
- 테이블 이름이 복제된다면 크롤러는 이름에 해시 문자열 접미사를 추가합니다.

크롤러가 일정에 따라 한 번을 초과하여 실행된다면 데이터 스토어에서 새로운 또는 변화된 파일 혹은 테이블을 찾습니다. 크롤러 출력값은 과거 실행에서 찾은 새로운 테이블과 파티션을 포함합니다.

어떤 데이터 스토어를 크롤할 수 있나요?

크롤러는 다음과 같은 파일 기반 및 테이블 기반 데이터 스토어를 크롤할 수 있습니다.

크롤러가 사용하는 액세스 유형	데이터 스토어
네이티브 클라이언트	<ul style="list-style-type: none"> • Amazon Simple Storage Service(S3) • Amazon DynamoDB • Delta Lake 2.0.x • Apache Iceberg 1.5 • Apache Hudi 0.14
JDBC	Amazon Redshift

크롤러가 사용하는 액세스 유형	데이터 스토어
	Snowflake Amazon Relational Database Service(Amazon RDS) 내 또는 Amazon RDS 외부: <ul style="list-style-type: none"> • Amazon Aurora • MariaDB • Microsoft SQL Server • MySQL • Oracle • PostgreSQL
MongoDB 클라이언트	<ul style="list-style-type: none"> • MongoDB • MongoDB Atlas • Amazon DocumentDB(MongoDB 호환)

Note
 현재 AWS Glue는 데이터 스트림에 대한 크롤러를 지원하지 않습니다.

JDBC, MongoDB, MongoDB Atlas 및 Amazon DocumentDB(MongoDB 호환) 데이터 스토어의 경우 크롤러가 데이터 스토어에 연결하는 데 사용할 수 있는 AWS Glue 연결을 지정해야 합니다. Amazon S3의 경우 선택적으로 네트워크 유형의 연결을 지정할 수 있습니다. 연결은 자격 증명, URL, Amazon Virtual Private Cloud 정보 등의 연결 정보를 저장하는 데이터 카탈로그 객체입니다. 자세한 내용은 [데이터에 연결](#) 단원을 참조하십시오.

크롤러에서 지원하는 드라이버 버전은 다음과 같습니다.

제품	크롤러 지원 드라이버
PostgreSQL	42.2.1

제품	크롤러 지원 드라이버
Amazon Aurora	네이티브 크롤러 드라이버와 동일
MariaDB	8.0.13
Microsoft SQL Server	6.1.0
MySQL	8.0.13
Oracle	11.2.2
Amazon Redshift	4.1
Snowflake	3.13.20
MongoDB	4.7.2
MongoDB Atlas	4.7.2

다음은 다양한 데이터 스토어에 대한 참고 사항입니다.

Amazon S3

사용자 계정 또는 다른 계정에서 경로를 크롤링하도록 선택할 수 있습니다. 폴더의 모든 Amazon S3 파일이 동일한 스키마를 보유하고 있으면 크롤러는 테이블 하나를 생성합니다. 또한 Amazon S3 객체가 분할된 경우 메타데이터 테이블이 하나만 생성되고 해당 테이블에 대한 데이터 카탈로그에 파티션 정보가 추가됩니다.

Amazon S3 및 Amazon DynamoDB

크롤러는 AWS Identity and Access Management(IAM) 역할을 사용하여 데이터 스토어에 대한 액세스 권한을 보유합니다. 크롤러에 전달하는 역할은 크롤된 Amazon S3 경로 및 Amazon DynamoDB 테이블에 대한 액세스 권한을 보유해야 합니다.

Amazon DynamoDB

AWS Glue 콘솔을 사용하여 크롤러를 정의하는 경우 DynamoDB 테이블을 지정할 수 있습니다. AWS Glue API를 사용하는 경우 테이블 목록을 지정할 수 있습니다. 크롤러 실행 시간을 줄이기 위해 데이터의 작은 샘플만 크롤링하도록 선택할 수 있습니다.

Delta Lake

각 Delta Lake 데이터 스토어에 대해 해당하는 Delta 테이블을 생성하는 방식을 지정합니다.

- 기본 테이블 생성: Delta 트랜잭션 로그의 쿼리를 직접 지원하는 쿼리 엔진과 통합할 수 있습니다. 자세한 내용은 [Delta Lake 테이블 쿼리하기](#)를 참조하세요.
- Symlink 테이블 생성: 지정된 구성 파라미터를 기반으로 파티션 키로 분할된 매니페스트 파일을 포함하는 `_symlink_manifest` 폴더를 생성합니다.

Iceberg

각 Iceberg 데이터 스토어에 대해 Iceberg 테이블의 메타데이터가 포함된 Amazon S3 경로를 지정합니다. 크롤러가 Iceberg 테이블 메타데이터를 검색하면 이를 데이터 카탈로그에 등록합니다. 크롤러가 테이블을 최신 상태로 유지하도록 예약을 설정할 수 있습니다.

데이터 스토어에 대해 다음 파라미터를 정의할 수 있습니다.

- 제외: 특정 폴더를 건너뛴 수 있습니다.
- 최대 이동 깊이: 크롤러가 Amazon S3 버킷에서 크롤링할 수 있는 깊이 제한을 설정합니다. 최대 이동 깊이의 기본값은 10이고 설정할 수 있는 최대 이동 깊이는 20입니다.

Hudi

각 Hudi 데이터 스토어에 대해 Hudi 테이블의 메타데이터가 포함된 Amazon S3 경로를 지정합니다. 크롤러가 Hudi 테이블 메타데이터를 검색하면 이를 데이터 카탈로그에 등록합니다. 크롤러가 테이블을 최신 상태로 유지하도록 예약을 설정할 수 있습니다.

데이터 스토어에 대해 다음 파라미터를 정의할 수 있습니다.

- 제외: 특정 폴더를 건너뛴 수 있습니다.
- 최대 이동 깊이: 크롤러가 Amazon S3 버킷에서 크롤링할 수 있는 깊이 제한을 설정합니다. 최대 이동 깊이의 기본값은 10이고 설정할 수 있는 최대 이동 깊이는 20입니다.

Note

논리적 유형으로 `millis`를 사용하는 타임스탬프 열은 Hudi 0.13.1 및 타임스탬프 유형과의 비호환성으로 인해 `bigint`로 해석됩니다. 향후 Hudi 릴리스에서 이 문제의 해결 방법이 제공될 수 있습니다.

Hudi 테이블은 다음과 같이 분류되며 각각 특정한 의미를 함축합니다.

- 쓸 때 복사(CoW): 데이터가 열 기반 형식(Parquet)으로 저장되며, 업데이트마다 쓰기 중에 새 버전의 파일을 만듭니다.
- 읽을 때 병합(MoR): 데이터가 열 기반 형식(Parquet)과 행 기반(Avro) 형식을 조합하여 저장됩니다. 업데이트는 행 기반 delta 파일에 기록되며 새 버전의 열 형식 파일을 작성할 때 필요에 따라 압축됩니다.

CoW 데이터 세트를 사용하면 레코드에 대한 업데이트가 있을 때마다 레코드가 포함된 파일이 업데이트된 값으로 다시 작성됩니다. MoR 데이터 세트를 사용하면 Hudi는 업데이트가 있을 때마다 변경된 레코드에 대한 행만 씁니다. MoR은 읽기 수행이 적고 쓰기 또는 변경이 많은 워크로드에 더 적합합니다. CoW는 자주 변경되지 않는 데이터에서 읽기 수행이 많은 워크로드에 더 적합합니다.

Hudi는 데이터 액세스를 위해 세 가지 쿼리 유형을 제공합니다.

- 스냅샷 쿼리: 지정된 커밋 또는 압축 작업 시 테이블의 최신 스냅샷을 보는 쿼리입니다. MoR 테이블의 경우 스냅샷 쿼리는 쿼리 시의 최신 파일 슬라이스의 기본 파일과 델타 파일을 병합하여 테이블의 최신 상태를 나타냅니다.
- 증분의 쿼리: 이 쿼리는 지정된 커밋 및 압축 이후 테이블에 기록된 새 데이터만 볼 수 있습니다. 이는 변경 스트림을 효과적으로 제공하여 증분 데이터 파이프라인을 사용할 수 있도록 합니다.
- 읽기 최적화 쿼리: MoR 테이블의 경우 쿼리가 압축된 최신 데이터를 표시합니다. CoW 테이블의 경우 이 쿼리는 커밋된 최신 데이터를 보여줍니다.

쓸 때 복사(CoW) 테이블의 경우 크롤러는 ReadOptimized serde

`org.apache.hudi.hadoop.HoodieParquetInputFormat`을 사용하는 단일 테이블을 데이터 카탈로그에서 생성합니다.

읽을 때 병합(MoR) 테이블의 경우 크롤러는 동일한 테이블 위치에 대한 두 개의 테이블을 데이터 카탈로그에서 생성합니다.

- ReadOptimized serde `org.apache.hudi.hadoop.HoodieParquetInputFormat`을 사용하는 `_ro` 접미사가 있는 테이블.
- 스냅샷 쿼리를 허용하는 RealTime Serde를 사용하는 `_rt` 접미사가 포함된 테이블 (`org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat`).

MongoDB 및 Amazon DocumentDB(MongoDB와 호환)

MongoDB 버전 3.2 이상이 지원됩니다. 크롤러 실행 시간을 줄이기 위해 데이터의 작은 샘플만 크롤링하도록 선택할 수 있습니다.

관계형 데이터베이스

인증에는 데이터베이스 사용자 이름과 암호가 사용됩니다. 데이터베이스 엔진의 유형에 따라 데이터베이스, 스키마 및 테이블처럼 어떤 객체를 크롤할 것인지 선택합니다.

Snowflake

Snowflake JDBC 크롤러는 테이블, 외부 테이블, 뷰 및 구체화된 뷰의 크롤링을 지원합니다. 구체화된 뷰 정의는 채워지지 않습니다.

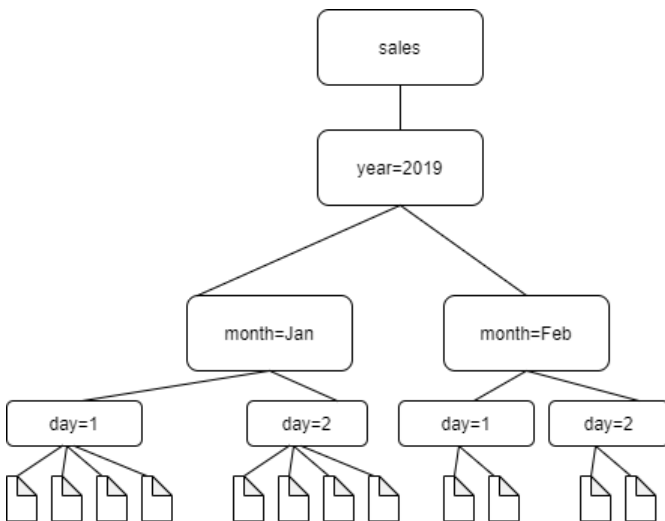
Snowflake 외부 테이블의 경우 크롤러는 Amazon S3 위치를 가리키는 경우에만 크롤링합니다. 크롤러는 테이블 스키마 이외에 Amazon S3 위치, 파일 형식 및 출력을 Data Catalog 테이블의 테이블 파라미터로 크롤링합니다. 파티션을 나눈 외부 테이블의 파티션 정보는 채워지지 않습니다.

ETL은 현재 Snowflake 크롤러를 사용하여 생성한 데이터 카탈로그 테이블에서 지원되지 않습니다.

크롤러는 파티션 생성 시기를 어떻게 결정하나요?

AWS Glue 크롤러가 Amazon S3를 스캔하고 버킷에서 여러 폴더를 발견하면 폴더 구조의 테이블 루트 및 테이블의 파티션에 해당하는 폴더를 결정합니다. Amazon S3 접두사 또는 폴더 이름을 기반의 테이블 이름. 크롤할 폴더 수준을 가리키는 [추가 경로]를 제공합니다. 폴더 수준의 다수 스키마가 유사한 경우, 크롤러는 개별 테이블 대신 테이블 파티션을 생성합니다. 크롤러가 개별 테이블에 영향을 주는 방법은 크롤러를 정의할 때 각 테이블의 루트 폴더를 데이터 스토어로 추가하는 것입니다.

예를 들어 다음과 같은 Amazon S3 폴더 구조를 고려합니다.



4개의 최하위 수준 폴더에 대한 경로는 다음과 같습니다.

```

S3://sales/year=2019/month=Jan/day=1
S3://sales/year=2019/month=Jan/day=2
S3://sales/year=2019/month=Feb/day=1
S3://sales/year=2019/month=Feb/day=2
  
```

크롤러 대상이 Sales로 설정되고 day=*n* 폴더의 모든 파일이 동일한 포맷(예: JSON, 암호화되지 않음)이고 동일하거나 매우 유사한 스키마를 갖는다고 가정합니다. 크롤러는 파티션 키가 year, month 및 day인 4개의 파티션이 있는 단일 테이블을 생성합니다.

다음 예에서는 다음과 같은 Amazon S3 구조를 고려합니다.

```
s3://bucket01/folder1/table1/partition1/file.txt
s3://bucket01/folder1/table1/partition2/file.txt
s3://bucket01/folder1/table1/partition3/file.txt
s3://bucket01/folder1/table2/partition4/file.txt
s3://bucket01/folder1/table2/partition5/file.txt
```

table1과 table2 아래의 파일 스키마가 유사하고 [포함 경로(Include path)]가 s3://bucket01/folder1/인 크롤러에 데이터 스토어 1개가 정의된 경우, 크롤러는 파티션 키 열 2개로 테이블 하나를 생성합니다. 첫 번째 파티션 키 열에는 table1과 table2가 포함되고 두 번째 파티션 키 열에는 table1 파티션의 경우 partition1~partition3, table2 파티션의 경우 partition4 및 partition5가 포함됩니다. 두 데이터 스토어로 크롤러를 정의하여 두 개별 테이블을 생성합니다. 이 예제에서는 첫 번째 Include path(추가 경로)를 s3://bucket01/folder1/table1/로 두 번째는 s3://bucket01/folder1/table2로 정의합니다.

Note

Amazon Athena에서 각 테이블은 모든 객체가 들어 있는 Amazon S3 접두사에 해당합니다. 객체들이 다른 스키마를 가지고 있으면 Athena는 동일한 접두사 내 다른 객체를 다른 테이블로 인식하지 못합니다. 크롤러가 동일한 Amazon S3 접두사의 여러 테이블을 생성하면 이와 같은 현상이 발생할 수 있습니다. 이는 어떤 결과 없이 Athena의 쿼리로 이끕니다. Athena가 테이블을 올바르게 인식하고 쿼리할 수 있도록 Amazon S3 폴더 구조에서 서로 다른 테이블 스키마마다 별도의 [포함 경로(Include path)]를 사용하여 크롤러를 생성합니다. 자세한 내용은 [AWS Glue와 함께 Athena를 사용할 때의 모범 사례](#)와 이 [AWS 지식 센터 문서](#)를 참조하세요.

크롤러 사전 조건

크롤러는 정의할 때 지정한 AWS Identity and Access Management(IAM) 역할의 권한을 수임합니다. IAM 역할은 데이터 스토어에서 데이터를 추출하여 데이터 카탈로그에 작성할 수 있는 권한이 있어야 합니다. AWS Glue 콘솔은 AWS Glue 보안 주체 서비스를 위한 신뢰할 수 있는 정책과 연관된 IAM 역할만 목록에 기록합니다. 콘솔에서 IAM 역할을 IAM 정책을 통해 생성하여 크롤러에 의해 액세스된

Amazon S3 데이터 스토어로 액세스할 수 있습니다. AWS Glue에 역할을 부여하는 것에 대한 자세한 내용은 [Glue에 대한 ID 기반 정책 AWS](#) 단원을 참조하십시오.

Note

Delta Lake 데이터 스토어를 크롤링할 때는 Amazon S3 위치에 대한 읽기/쓰기 권한을 가지고 있어야 합니다.

크롤러의 경우 역할을 생성하고 다음 정책을 연결할 수 있습니다.

- 데이터 카탈로그에 필요한 권한을 부여하는 AWSGlueServiceRole AWS 관리형 정책
- 데이터 원본에 대한 권한을 부여하는 인라인 정책입니다.
- 역할에 대한 iam:PassRole 권한을 부여하는 인라인 정책입니다.

더 빠른 접근 방식은 AWS Glue 콘솔 크롤러 마법사가 역할을 생성하도록 하는 것입니다. 생성하는 역할은 특히 크롤러를 위한 것이며 AWSGlueServiceRole AWS 관리형 정책과 지정된 데이터 원본에 대한 필수 인라인 정책을 포함합니다.

크롤러에 대한 기존 역할을 지정하는 경우 AWSGlueServiceRole 정책 또는 이에 상응하는 것(또는 이 정책의 범위가 축소된 버전)과 필수 인라인 정책이 포함되어 있는지 확인합니다. 예를 들어 Amazon S3 데이터 스토어의 경우 인라인 정책은 최소한 다음과 같습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket/object*"
      ]
    }
  ]
}
```

Amazon DynamoDB 데이터 스토어의 경우 정책은 최소한 다음과 같습니다.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:Scan"
      ],
      "Resource": [
        "arn:aws:dynamodb:region:account-id:table/table-name*"
      ]
    }
  ]
}
```

크롤러가 AWS Key Management Service(AWS KMS) 암호화 Amazon S3 데이터를 읽는 경우 IAM 역할이 AWS KMS 키에 대한 권한을 복호화해야 합니다. 자세한 내용은 [2단계: AWS Glue에 대한 IAM 역할 생성](#) 단원을 참조하십시오.

크롤러 구성

크롤러가 데이터 스토어로 액세스하고 메타데이터를 추출하고 AWS Glue Data Catalog의 테이블 정의를 생성합니다. AWS Glue 콘솔의 [크롤러(Crawlers)] 창에는 생성한 모든 크롤러가 나열됩니다. 목록은 크롤러 마지막 실행의 상태와 지표를 보여줍니다.

Note

자체 JDBC 드라이버 버전을 가져오기로 선택한 경우 AWS Glue 크롤러는 AWS Glue작업 및 Amazon S3 버킷의 리소스를 소비하여 제공된 드라이버가 사용자 환경에서 실행되도록 합니다. 리소스의 추가 사용량은 계정에 반영됩니다. 또한 자체 JDBC 드라이버를 제공한다고 해서 크롤러가 해당 드라이버의 모든 기능을 사용할 수 있는 것은 아닙니다. 드라이버는 [AWS Glue 연결 추가](#)에 설명된 속성으로 제한됩니다.

크롤러를 구성하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다. 탐색 창에서 크롤러를 선택합니다.

2. 크롤러 생성을 선택하고 크롤러 추가 마법사의 지침을 따릅니다. 마법사는 크롤러 생성에 필요한 단계를 안내합니다. 사용자 지정 분류기를 추가하여 스키마를 정의하려면 [AWS Glue의 크롤러로 분류자 추가](#) 섹션을 참조하세요.

1단계: 크롤러 속성 설정

크롤러에 대한 이름과 설명(선택 사항)을 입력합니다. 선택에 따라 태그 키와 선택 사항인 태그 값으로 크롤러에 태그를 지정할 수 있습니다. 생성된 태그 키는 읽기 전용입니다. 일부 리소스에서 태그를 이용하면 리소스를 정리하고 식별하는 데 도움이 됩니다. 자세한 내용은 AWS Glue의 AWS 태그를 참조하세요.

명칭

이름은 문자(A-Z), 숫자(0-9), 하이픈(-) 또는 밑줄(_) 포함할 수 있으며 최대 255자로 지정할 수 있습니다.

설명

설명은 최대 2048자까지 입력이 가능합니다.

Tags

태그를 사용하여 리소스를 정리하고 식별할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- [AWS Glue의 AWS 태그](#)

2단계: 데이터 소스 및 분류자 선택

데이터 소스 구성

데이터가 AWS Glue 테이블에 이미 매핑되어 있습니까?에 적절한 옵션인 '아직' 또는 '예'를 선택하세요. 기본적으로 '아직'이 선택됩니다.

크롤러는 크롤의 원본으로 데이터 스토어에 직접 액세스하거나 데이터 카탈로그의 기존 테이블을 원본으로 사용할 수 있습니다. 크롤러가 기존 카탈로그 테이블을 사용하면 해당 카탈로그 테이블에 지정된 데이터 스토어가 크롤됩니다.

- 아직 안 됨: 크롤링할 데이터 소스를 하나 이상 선택합니다. 크롤러는 서로 다른 유형(Amazon S3, JDBC 등)의 여러 데이터 스토어를 크롤할 수 있습니다.

한 번에 하나의 데이터 스토어만 구성할 수 있습니다. 연결 정보를 제공하고 경로를 포함하고 패턴을 제외하면 다른 데이터 스토어를 추가할 수 있습니다.

- 예: AWS Glue 데이터 카탈로그에서 기존 테이블을 선택합니다. 카탈로그 테이블은 크롤할 데이터 스토어를 지정합니다. 크롤러는 단일 실행으로 카탈로그 테이블만 크롤할 수 있습니다. 다른 원본 유형에서는 혼합할 수 없습니다.

카탈로그 테이블을 원본으로 지정하는 일반적인 이유는 (데이터 스토어의 구조를 이미 알고 있었으므로) 테이블을 수동으로 생성했으며 크롤러가 새 파티션을 추가하는 등 테이블을 업데이트된 상태로 유지하기 때문입니다. 다른 이유에 대한 설명은 [크롤러를 사용하여 수동으로 생성된 Data Catalog 테이블 업데이트](#) 단원을 참조하십시오.

기존 테이블을 크롤러 원본 유형으로 지정하면 다음 조건이 적용됩니다.

- 데이터베이스 이름은 선택 사항입니다.
- Amazon S3 또는 Amazon DynamoDB 데이터 스토어를 지정하는 카탈로그 테이블만 허용됩니다.
- 크롤러가 실행되면 새 카탈로그 테이블이 생성되지 않습니다. 기존 테이블은 새 파티션을 추가하는 등 필요에 따라 업데이트됩니다.
- 데이터 스토어에서 찾은 삭제된 객체는 무시됩니다. 카탈로그 테이블은 삭제되지 않습니다. 대신 크롤러는 로그 메시지를 작성합니다. (SchemaChangePolicy.DeleteBehavior=LOG)
- 각 Amazon S3 경로에 대해 단일 스키마를 생성하는 크롤러 구성 옵션은 기본적으로 사용되며 사용 중지할 수 없습니다. (TableGroupingPolicy=CombineCompatibleSchemas) 자세한 내용은 [각 Amazon S3 포함 경로에 대해 단일 스키마를 생성하는 방법](#) 섹션을 참조하십시오.
- 다른 원본 유형(예: Amazon S3 또는 Amazon DynamoDB)과 원본으로 카탈로그 테이블을 혼합할 수 없습니다.

데이터 소스

크롤러가 스캔할 데이터 소스 목록을 선택하거나 추가합니다.

(선택 사항) JDBC를 데이터 소스로 선택하면 드라이버 정보가 저장되는 연결 액세스를 지정할 때 자체 JDBC 드라이버를 사용할 수 있습니다.

포함 경로

크롤러에 무엇을 포함하거나 제외할 것인지 평가할 때 크롤러는 필요한 포함 경로를 평가하는 것으로 시작합니다. Amazon S3, MongoDB, MongoDB Atlas, Amazon DocumentDB(MongoDB 호환) 및 관계형 데이터 스토어의 경우 포함 경로를 지정해야 합니다.

Amazon S3 데이터 스토어의 경우

이 계정에서 경로를 지정할지 혹은 다른 계정에서 경로를 지정할지 선택한 다음 Amazon S3 경로 선택을 위해 찾아봅니다.

Amazon S3 데이터 스토어의 경우 경로 포함 구문은 `bucket-name/folder-name/file-name.ext`입니다. 버킷에 있는 모든 객체를 크롤하려면 추가 경로에서 버킷 이름을 지정합니다. 제외 패턴은 포함 경로와 상대적입니다.

Delta Lake 데이터 스토어의 경우

`s3://bucket/prefix/object`와 같이 Delta 테이블에 대한 Amazon S3 경로를 하나 이상 지정합니다.

Iceberg 또는 Hudi 데이터 스토어의 경우

Iceberg 또는 Hudi 테이블 메타데이터가 포함된 폴더를 포함하는 Amazon S3 경로를 하나 이상 지정합니다(`s3://bucket/prefix` 형식).

Hudi 데이터 스토어의 경우 Hudi 폴더는 루트 폴더의 하위 폴더에 있을 수 있습니다. 크롤러는 경로 아래에 있는 모든 폴더에서 Hudi 폴더를 스캔합니다.

JDBC 데이터 스토어의 경우

데이터베이스 제품에 따라 `<database>/<schema>/<table>` 또는 `<database>/<table>`을 입력합니다. Oracle Database 및 MySQL은 경로의 스키마를 지원하지 않습니다. `<schema>` 또는 `<table>` 대신에 백분율 문자(%)를 사용할 수 있습니다. 예를 들어 시스템 식별자(SID)가 `orcl`인 Oracle Database의 경우 `orcl/%`를 입력하여 연결에 이름이 지정된 사용자가 액세스할 수 있는 모든 테이블을 가져옵니다.

Important

이 필드는 대/소문자를 구분합니다.

MongoDB, MongoDB Atlas 또는 Amazon DocumentDB 데이터 스토어의 경우

`database/collection`을 입력합니다.

MongoDB, MongoDB Atlas 및 Amazon DocumentDB(MongoDB와 호환)의 경우 구문은 `database/collection`입니다.

JDBC 데이터 스토어의 구문은 `database-name/schema-name/table-name` 혹은 `database-name/table-name`입니다. 구문은 데이터베이스 엔진이 데이터베이스 내에서 스키마를 지원할

지 여부에 따라 다릅니다. 예를 들어, MySQL 또는 Oracle과 같은 데이터베이스는 추가 경로에서 `schema-name`를 지정할 필요가 없습니다. 퍼센트 표시(%)를 추가 경로 내 스키마 또는 테이블로 바꿔 데이터베이스 내 모든 스키마 혹은 모든 테이블을 나타냅니다. 퍼센트 표시(%)를 추가 경로 내 데이터베이스로 바꿀 수 없습니다.

최대 이동 깊이(Iceberg 또는 Hudi 데이터 스토어만 해당)

크롤러가 Amazon S3 경로에서 Iceberg 또는 Hudi 메타데이터 폴더를 검색하기 위해 이동할 수 있는 Amazon S3 경로의 최대 깊이를 정의합니다. 이 파라미터의 목적은 크롤러 실행 시간을 제한하는 것입니다. 기본값은 10이고, 최댓값은 20입니다.

제외 패턴

이렇게 하면 크롤에서 특정 파일이나 테이블을 제외할 수 있습니다. 제외 경로는 포함 경로와 상대적입니다. 예를 들어 테이블에서 JDBC 데이터 스토어에서 테이블을 제외하기 위해서 테이블 이름을 제외 경로에 입력합니다.

크롤러는 JDBC URI 연결 문자열을 포함한 AWS Glue를 사용하여 JDBC 데이터 스토어에 연결합니다. 크롤러는 AWS Glue 연결의 JDBC 사용자 이름 및 비밀번호를 사용하여 데이터베이스 엔진의 객체에 액세스만 합니다. 크롤러는 JDBC 연결을 통한 액세스할 수 있는 테이블을 생성만 합니다. JDBC URI로 데이터베이스 엔진에 크롤러가 액세스한 후, 추가 경로는 데이터 카탈로그에 생성된 데이터베이스 엔진의 테이블을 결정하는데 사용됩니다. 예를 들어 MySQL의 경우, `MyDatabase/%`의 추가 경로를 지정하면 `MyDatabase` 내 모든 테이블은 데이터 카탈로그에 생성됩니다. Amazon Redshift에 액세스할 경우, `MyDatabase/%`의 추가 경로를 지정하면 데이터베이스 `MyDatabase`의 모든 스키마 내 모든 테이블은 데이터 카탈로그에 생성됩니다. `MyDatabase/MySchema/%`의 추가 경로를 지정하면 `MyDatabase` 데이터베이스의 모든 테이블과 `MySchema` 스키마가 생성됩니다.

추가 경로를 지정한 후, 크롤러에서 객체를 제외합니다. 그렇지 않으면 추가 경로는 하나 이상의 Unix 스타일 `glob`이 패턴을 제외하는 것을 지정하여 객체를 추가합니다. 이런 패턴은 추가 경로에 적용되어 어떤 객체가 제외되어야 하는지 결정합니다. 또한 이런 패턴은 크롤러가 생성한 테이블의 속성으로 저장됩니다. AWS Glue `create_dynamic_frame.from_catalog` 등의 PySpark 확장은 테이블 속성을 읽고 제외 패턴이 정의한 객체를 제외합니다.

AWS Glue는 제외 패턴에서 `glob` 패턴의 다음 종류를 지원합니다.

제외 패턴	설명
*.csv	.csv로 끝나는 현재 폴더의 객체 이름을 나타내는 Amazon S3 경로에 해당합니다.

제외 패턴	설명
.	점을 포함한 모든 객체 이름과 해당합니다.
*.{csv,avro}	.csv 또는 .avro로 끝나는 객체 이름과 해당합니다.
foo.?	하나의 문자 확장에 따라 foo.으로 시작하는 객체 이름과 해당합니다.
myfolder/*	/myfolder/mysource 와 같은 myfolder에서 하위 폴더의 한 수준의 객체와 해당합니다.
myfolder/**	/myfolder/mysource/data 와 같은 myfolder에서 하위 폴더의 두 가지 수준의 객체와 해당합니다.
myfolder/**	/myfolder/mysource/mydata 및 /myfolder/mysource/data 와 같은 myfolder의 모든 하위 폴더의 객체와 해당합니다.
myfolder**	myfolder 아래 파일뿐만 아니라 하위 폴더 myfolder도 일치시킵니다(예: /myfolder 와 /myfolder/mydata.txt).
Market*	Market_us 및 Market_fr 와 같은 Market으로 시작하는 이름의 JDBC 데이터베이스에 있는 테이블과 해당합니다.

AWS Glue는 다음과 같이 glob가 패턴을 제외하도록 해석합니다.

- Amazon S3 키를 폴더 계층으로 구분하기 위해 슬래시(/) 문자가 사용됩니다.
- 별표(*) 문자는 폴더의 경계를 넘지 않고 0개 이상의 이름 구성 요소 문자와 해당합니다.
- 이중 별표(**) 문자는 폴더 또는 스키마의 경계를 넘지 않고 0개 이상의 이름 구성 요소 문자와 해당합니다.
- 물음표(?) 문자는 이름 구성 요소의 정확히 1문자에 해당합니다.

- 역 슬래시(\) 문자는 특수 문자로 해석될 수 있는 이스케이프 문자로 사용됩니다. 표현식 \\는 하나의 백 슬래시에 해당하고 \{는 왼쪽 중괄호에 해당합니다.
- 대괄호([])는 문자 모음 중 한 문자의 이름 구성 요소에 해당하는 괄호 표현식을 생성합니다. 예를 들어, [abc]는 a, b 혹은 c에 해당합니다. 하이픈(-)은 범위를 지정하는 데 사용되기 때문에 [a-z]는 a에서 z까지의 범위를 지정합니다 (포괄적). 이런 형식은 혼합될 수 있어 [abce-g]는 a, b, c, e, f, 또는 g에 해당됩니다. 대괄호([) 다음 문자가 느낌표(!)라면, 괄호 표현식이 취소됩니다. 예를 들어, [!a-c]는 a, b 또는 c를 제외한 모든 문자와 대응합니다.

괄호 표현식 내에 *, ? 및 \는 자체로 대응됩니다. 취소할 때 하이픈이 괄호 내 첫 번째 문자이거나 ! 다음 첫 번째 문자이면 하이픈(-) 문자는 자체로 대응합니다.

- 부 패턴이 그룹 내에서 대등하면 중괄호({ })는 그룹과 대응하는 부 패턴 그룹을 포함합니다. 쉼표(,) 문자는 부 패턴을 구분하는 기호입니다. 그룹은 중첩될 수 없습니다.
- 파일명의 첫 번째 마침표 혹은 점은 매칭 작업에서 일반 문자로 취급합니다. 예를 들어 *가 파일명 .hidden에 해당하는 패턴을 제외합니다.

Example Amazon S3 제외 패턴

각 제외 패턴은 추가 경로에 대응하여 평가됩니다. 예를 들어 다음과 같은 Amazon S3 디렉터리 구조가 있다고 가정하겠습니다.

```
/mybucket/myfolder/
  departments/
    finance.json
    market-us.json
    market-emea.json
    market-ap.json
  employees/
    hr.json
    john.csv
    jane.csv
    juan.txt
```

추가 경로 s3://mybucket/myfolder/에 따르면 다음 예는 패턴을 제외한 결과입니다.

제외 패턴	결과
departments/**	departments 아래에 있는 모든 파일 및 폴더를 제외하고 employees 폴더 및 하위 폴더를 포함합니다.

제외 패턴	결과
departments/market*	market-us.json , market-emea.json 및 market-ap.json 을 제외합니다.
** .csv	.csv로 끝나는 이름을 갖는 myfolder의 모든 하위 객체를 제외합니다.
employees/*.csv	employees 폴더 내 모든 .csv 파일을 제외합니다

Example Amazon S3 파티션의 하위 집합 제외

일별로 데이터가 분할되었다면 개별 Amazon S3 파티션에 1년의 각 일자가 존재합니다. 따라서 2015년 1월이면 31 파티션이 있습니다. 이제, 1월 첫째 주 데이터만 크롤하고자 한다면 1일부터 7 일까지만 남겨놓고 모든 파티션을 제외합니다.

```
2015/01/{[!0],0[8-9]}**, 2015/0[2-9]**, 2015/1[0-2]**
```

이 glob 형식을 살펴보겠습니다. 첫 번째 부분인 2015/01/{[!0],0[8-9]}**는 2015년 01월 중 08일과 09뿐만 아니라 0으로 시작하지 않는 모든 일자를 제외합니다. "**"는 일수 형식의 접미사로 사용된다는 것을 기억하고 폴더 경계를 더 낮은 수준의 폴더로 지나갑니다. "*"가 사용되면 하위 폴더 수준은 제외되지 않습니다.

두 번째 부분인 2015/0[2-9]**는 2015년 중 02월에서 09월까지의 일자를 제외합니다.

세 번째 부분인 2015/1[0-2]**는 2015년 중 10월, 11월 및 12월의 일자를 제외합니다.

Example JDBC 제외 패턴

다음과 같은 스키마 구조로 JDBC 데이터베이스를 크롤한다고 가정하겠습니다.

```
MyDatabase/MySchema/
  HR_us
  HR_fr
  Employees_Table
  Finance
```



```
Market_US_Table
Market_EMEA_Table
Market_AP_Table
```

추가 경로 MyDatabase/MySchema/%에 따르면 다음 예는 패턴을 제외한 결과입니다.

제외 패턴	결과
HR*	HR로 이름이 시작되는 테이블을 제외합니다.
Market_*	Market_로 이름이 시작되는 테이블을 제외합니다.
**_Table	_Table로 이름이 끝나는 모든 테이블을 제외합니다.

추가 크롤러 소스 파라미터

소스 유형마다 다른 추가 파라미터 집합이 필요합니다. 다음은 불안정한 목록입니다.

연결

AWS Glue 연결을 선택하거나 추가합니다. 연결에 관한 자세한 내용은 [데이터에 연결](#) 단원을 참조하십시오.

추가 메타데이터 - 선택 사항(JDBC 데이터 스토어에만 해당)

크롤러가 크롤링할 추가 메타데이터 속성을 선택합니다.

- **댓글:** 관련 테이블 수준 및 열 수준 설명을 크롤링합니다.
- **원시 유형:** 테이블 열의 원시 데이터 유형을 추가 메타데이터에 유지합니다. 기본 동작으로 크롤러는 원시 데이터 유형을 Hive 호환 유형으로 변환합니다.

JDBC 드라이버 클래스 이름 - 선택 사항 (JDBC 데이터 스토어에만 해당)

크롤러가 데이터 소스에 연결하는 데 사용할 사용자 지정 JDBC 드라이버 클래스 이름을 입력합니다.

- **Postgres:** org.postgresql.Driver
- **MySQL:** com.mysql.jdbc.Driver, com.mysql.cj.jdbc.Driver
- **Redshift:** com.amazon.redshift.jdbc.Driver, com.amazon.redshift.jdbc42.Driver

- Oracle: oracle.jdbc.driver.OracleDriver
- SQL Server: com.microsoft.sqlserver.jdbc.SQLServerDriver

JDBC 드라이버 S3 경로 - 선택 사항(JDBC 데이터 스토어에만 해당)

.jar 파일에 대한 기존 Amazon S3 경로를 선택합니다. 크롤러를 데이터 소스에 연결할 때 사용자 지정 JDBC 드라이버를 사용하는 경우 .jar 파일이 여기에 저장됩니다.

데이터 샘플링 사용(Amazon DynamoDB, MongoDB, MongoDB Atlas Amazon DocumentDB 데이터 스토어만 해당)

데이터 샘플만 크롤링할지 여부를 선택합니다. 선택하지 않으면 전체 테이블이 크롤링됩니다. 테이블이 높은 처리량 테이블이 아닌 경우 모든 레코드를 스캔하는 데 시간이 오래 걸릴 수 있습니다.

쿼리를 위한 테이블 생성(Delta Lake 데이터 스토어에만 해당)

Delta Lake 테이블 생성 방법을 선택합니다.

- 기본 테이블 생성: Delta 트랜잭션 로그의 쿼리를 직접 지원하는 쿼리 엔진과 통합할 수 있습니다.
- Symlink 테이블 생성: 지정된 구성 파라미터를 기반으로 파티션 키로 분할된 매니페스트 파일을 포함하는 symlink 매니페스트 폴더를 생성합니다.

스캔 속도 - 선택 사항(DynamoDB 데이터 스토어에만 해당)

크롤러에서 사용할 DynamoDB 테이블 읽기 용량 단위의 비율을 지정합니다. 읽기 용량 단위는 DynamoDB에서 정의한 용어이며, 초당 해당 테이블에서 수행할 수 있는 읽기 수에 대한 속도 제한 기 역할을 하는 숫자 값입니다. 0.1과 1.5 사이의 값을 입력합니다. 값을 지정하지 않으면 기본값은 프로비저닝된 테이블의 경우 0.5%이고 온디맨드 테이블의 경우 구성된 최대 용량의 1/4입니다. 프로비저닝된 용량 모드만 AWS Glue 크롤러와 함께 사용해야 합니다.

Note

DynamoDB 데이터 스토어의 경우, 테이블의 읽기 및 쓰기 처리를 위한 프로비저닝된 용량 모드를 설정합니다. AWS Glue 크롤러는 온디맨드 용량 모드와 함께 사용하지 않아야 합니다.

네트워크 연결 - 선택 사항(Amazon S3 데이터 스토어에만 해당)

필요에 따라 이 Amazon S3 대상과 함께 사용할 네트워크 연결을 포함합니다. 각 크롤러는 하나의 네트워크 연결로 제한되므로 다른 Amazon S3 대상도 동일한 연결을 사용하게 됩니다(비워 두면 연결 없음).

연결에 관한 자세한 내용은 [데이터에 연결](#) 단원을 참조하십시오.

파일의 하위 집합과 샘플 크기만 샘플링(Amazon S3 데이터 스토어에만 해당)

데이터 집합의 샘플 파일을 크롤링할 때 크롤링할 각 리프 폴더의 파일 수를 지정합니다. 이 기능이 설정되어 있으면 이 데이터 집합의 모든 파일을 크롤링하는 대신 크롤러가 크롤링할 각 리프 폴더의 일부 파일을 무작위로 선택합니다.

샘플링 크롤러는 데이터 포맷에 대한 사전 지식이 있고 폴더의 스키마가 변경되지 않는다는 것을 알고 있는 고객에게 가장 적합합니다. 이 기능을 설정하면 크롤러 런타임이 크게 감소합니다.

유효한 값은 1~249의 정수입니다. 지정하지 않으면 모든 파일이 크롤링됩니다.

후속 크롤러 실행

이 필드는 모든 Amazon S3 데이터 소스에 영향을 주는 글로벌 필드입니다.

- 모든 하위 폴더 크롤링: 이후에 크롤링할 때마다 모든 폴더를 다시 크롤링합니다.
- 새 하위 폴더만 크롤링: 마지막 크롤링된 이후에 추가된 Amazon S3 폴더만 크롤링됩니다. 스키마가 호환되는 경우 새 파티션이 기존 테이블에 추가됩니다. 자세한 내용은 [the section called “새 파티션을 추가하기 위한 점진적인 크롤링”](#) 단원을 참조하십시오.
- 이벤트 기반 크롤링: Amazon S3 이벤트를 사용하여 크롤링할 폴더를 제어합니다. 자세한 내용은 [the section called “Amazon S3 이벤트 알림을 사용하여 크롤링 가속화”](#) 단원을 참조하십시오.

사용자 지정 분류자 - 선택 사항

크롤러를 정의하기 전에 사용자 지정 분류자를 정의합니다. 분류자는 지정된 파일이 크롤러가 처리할 수 있는 형식인지 여부를 확인합니다. 처리할 수 있는 형식인 경우, 분류자는 해당 데이터 형식과 일치하는 StructType 객체의 형태로 스키마를 생성합니다.

자세한 내용은 [AWS Glue의 크롤러로 분류자 추가](#) 단원을 참조하십시오.

3단계: 보안 설정 구성

IAM 역할

크롤러는 이 역할을 수임합니다. AWS 관리형 정책 `AWSGlueServiceRole`과 비슷한 권한이 있어야 합니다. Amazon S3 및 DynamoDB 소스의 경우 데이터 스토어에 액세스할 수 있는 권한도 있어야 합니다. 크롤러가 AWS Key Management Service(AWS KMS)로 암호화된 Amazon S3 데이터를 읽는 경우 역할에 AWS KMS 키에 대한 복호화 권한이 있어야 합니다.

Amazon S3 데이터 스토어의 경우 역할에 연결된 추가 권한은 다음과 유사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket/object*"
      ]
    }
  ]
}
```

Amazon DynamoDB 데이터 스토어의 경우 역할에 연결된 추가 권한은 다음과 유사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:Scan"
      ],
      "Resource": [
        "arn:aws:dynamodb:region:account-id:table/table-name*"
      ]
    }
  ]
}
```

자체 JDBC 드라이버를 추가하려면 추가 권한을 추가해야 합니다.

- CreateJob, DeleteJob, GetJob, GetJobRun, StartJobRun 작업에 대한 권한을 부여합니다.
- Amazon S3 작업, s3:DeleteObjects, s3:GetObject, s3:ListBucket, s3:PutObject에 대한 권한을 부여합니다.

Note

Amazon S3 버킷 정책이 비활성화된 경우에는 `s3:ListBucket`이 필요하지 않습니다.

- 서비스 보안 주체에 Amazon S3 정책의 버킷 및 폴더에 대한 액세스 권한을 부여합니다.

Amazon S3 정책 예제:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-name/driver-parent-folder/driver.jar",
        "arn:aws:s3:::bucket-name"
      ]
    }
  ]
}
```

AWS Glue에서는 Amazon S3 버킷에서 JDBC 드라이버와 동일한 수준에 있는 `_crawler` 및 `_glue_job_crawler` 폴더를 생성합니다. 예를 들어 드라이버 경로가 `<s3-path/driver_folder/driver.jar>`인 경우 다음 폴더가 아직 없으면 폴더가 생성됩니다.

- `<s3-path/driver_folder/_crawler>`
- `<s3-path/driver_folder/_glue_job_crawler>`

선택적으로 크롤러에 보안 구성을 추가하여 저장 데이터 암호화 옵션을 지정할 수 있습니다.

자세한 내용은 [2단계: AWS Glue에 대한 IAM 역할 생성](#) 및 [AWS Glue의 ID 및 액세스 관리 단원을](#) 참조하세요.

Lake Formation 구성 - 선택 사항

크롤러가 Lake Formation 보안 인증을 사용하여 데이터 소스를 크롤링하도록 허용합니다.

Use Lake Formation credentials for crawling S3 data source(Lake Formation 보안 인증을 사용하여 S3 데이터 소스 크롤링)을 선택하면 크롤러가 Lake Formation 보안 인증을 사용하여 데이터 소스를 크롤링할 수 있습니다. 데이터 소스가 다른 계정에 속하는 경우 등록된 계정 ID를 제공해야 합니다. 그렇지 않으면 크롤러는 계정과 연결된 데이터 소스만 크롤링합니다. Amazon S3 및 데이터 카탈로그 데이터 소스에만 해당됩니다.

보안 구성 - 선택 사항

설정에는 보안 구성이 포함됩니다. 자세한 내용은 다음 자료를 참조하세요.

- [AWS Glue에서 작성한 데이터 암호화](#)

Note

크롤러에 보안 구성이 설정된 후에는 이를 변경할 수는 있지만 제거할 수는 없습니다. 크롤러의 보안 수준을 낮추려면 구성 내에서 보안 기능을 DISABLED로 명시적으로 설정하거나 새 크롤러를 생성하세요.

4단계: 출력 및 일정 설정

출력 구성

옵션에는 크롤러가 감지된 스키마 변경 사항, 데이터 스토어에서 삭제된 객체 등을 처리하는 방법이 포함됩니다. 자세한 내용은 [크롤러 동작 사용자 지정](#) 단원을 참조하세요.

크롤러 예약

필요에 따라 크롤러를 실행하거나 AWS Glue에서 크롤러 및 작업을 위한 시간 기반 일정을 정의합니다. 일정 정의는 Unix식 cron 구문을 사용합니다. 자세한 내용은 [AWS Glue 크롤러 일정 관리](#) 단원을 참조하십시오.

5단계: 검토 및 생성

구성한 크롤러 설정을 검토하고 크롤러를 생성합니다.

AWS Glue의 크롤러로 분류자 추가

분류자는 데이터 스토어의 데이터를 읽습니다. 데이터 포맷을 인식하면 스키마를 생성합니다. 분류자는 확실성 숫자를 반환하여 포맷 인식이 어땠는지 보여줍니다.

AWS Glue에서 기본 분류자 세트를 제공하지만 사용자 지정 분류자를 생성할 수도 있습니다. AWS Glue는 크롤러 정의에 지정된 순서로 사용자 정의 분류자를 먼저 호출합니다. 사용자 분류자에 반환된 결과에 따라 AWS Glue는 기본 설정 분류자도 시작할 수 있습니다. 분류자가 진행 중에 `certainty=1.0`을 반환하면 정확한 스키마를 생성할 수 있다는 100% 확실성을 나타냅니다. 이후 AWS Glue가 해당 분류자의 출력을 사용합니다.

어떠한 분류자도 `certainty=1.0`을 반환하지 않으면, AWS Glue는 높은 확실성이 있는 분류자 출력을 사용합니다. 어떠한 분류자도 `0.0`을 초과한 확실성을 반환하지 않으면, AWS Glue는 UNKNOWN의 기본 분류 문자열을 반환합니다.

언제 분류자를 사용하나요?

데이터 스토어를 크롤하여 AWS Glue Data Catalog에 메타데이터 테이블을 정의할 때 분류자를 사용합니다. 분류자의 순서가 있는 세트를 통해 크롤러를 설치할 수 있습니다. 크롤러가 분류자를 시작할 때 분류자는 데이터가 인식되었는지 결정합니다. 분류자가 데이터를 인식하지 못하거나 100% 확실성이 없다면 크롤러는 목록에서 다음 분류자를 시작하여 데이터를 인식하는지 결정합니다.

AWS Glue 콘솔을 사용한 분류자 생성에 대한 자세한 내용은 [AWS Glue 콘솔에서 분류자 관련 작업을 참조하십시오](#).

사용자 지정 분류자

분류자의 출력은 파일 분류 혹은 포맷(예를 들어 json) 및 파일 스키마를 보여주는 문자열을 포함합니다. 사용자 지정 분류자의 경우, 분류자 유형을 기반으로 스키마를 생성하는 로직을 정의합니다. 분류자 유형은 grok 패턴, XML 태그 및 JSON에 따른 스키마 정의를 포함합니다.

분류자 정의를 변경하면 분류자를 사용하여 이전에 크롤링된 어떤 데이터도 재분류가 되지 않습니다. 크롤러는 이전에 크롤링된 데이터를 계속 추적합니다. 새로운 데이터는 업데이트된 스키마 결과를 낼 수 있는 업데이트된 분류자로 분류됩니다. 데이터 스키마가 연관되면 크롤러를 실행할 때 분류자를 계정에 업데이트하여 스키마를 변경합니다. 부정확한 분류자를 바로 잡기 위해 데이터를 재분류하려면 업데이트된 분류자로 새로운 데이터를 생성합니다.

AWS Glue의 사용자 지정 분류자를 생성하는 방법에 대한 자세한 내용은 [사용자 지정 분류자 작성](#) 섹션을 참조하십시오.

Note

데이터 포맷이 기본 설정 분류자 중 하나로 인식되면 사용자 지정 분류자를 생성할 필요가 없습니다.

AWS Glue의 기본 설정 분류자

AWS Glue는 JSON, CSV, 웹 로그 및 다수 데이터베이스 시스템을 포함한 다양한 포맷의 기본 설정 분류자를 제공합니다.

AWS Glue가 100% 확실성의 입력 데이터 포맷에 맞는 사용자 지정 분류자를 찾을 수 없다면 다음 테이블에 보여진 것처럼 기본 설정 분류자를 시작합니다. 기본 설정 분류자는 포맷이 일치한지(certainty=1.0) 일치하지 않는지(certainty=0.0) 보여주는 결과를 반환합니다. certainty=1.0이 있는 첫 번째 분류자는 Data Catalog의 메타데이터 테이블의 분류 문자열 및 스키마를 제공합니다.

분류자 유형	분류 문자열	참고
Apache Avro	avro	파일 시작 부분의 스키마를 읽어 포맷을 결정합니다.
Apache ORC	orc	파일 메타데이터를 읽어 포맷을 결정합니다.
Apache Parquet	parquet	파일 끝 부분의 스키마를 읽어 포맷을 결정합니다.
JSON	json	파일 시작 부분을 읽어 포맷을 결정합니다.
이진수 JSON	bson	파일 시작 부분을 읽어 포맷을 결정합니다.
XML	xml	파일 시작부를 읽어 형식을 확인합니다. AWS Glue는 문서의 XML 태그에 기반하여 테이블 스키마를 확인합니다. 문서의 행을 지정하기 위한 사용자 지정 분류자를 생성하는 방법에 대한 자세한 내용은 XML 사용자 지정 분류자 작성 단원을 참조하십시오.
Amazon Ion	ion	파일 시작 부분을 읽어 포맷을 결정합니다.

분류자 유형	분류 문자열	참고
통합된 Apache Log	combined_apache	grok 패턴을 통해 로그 포맷을 결정합니다.
Apache log	apache	grok 패턴을 통해 로그 포맷을 결정합니다.
Linux kernel log	linux_kernel	grok 패턴을 통해 로그 포맷을 결정합니다.
Microsoft log	microsoft_log	grok 패턴을 통해 로그 포맷을 결정합니다.
Ruby log	ruby_logger	파일 시작 부분을 읽어 포맷을 결정합니다.
Squid 3.x log	squid	파일 시작 부분을 읽어 포맷을 결정합니다.
Redis monitor log	redismonlog	파일 시작 부분을 읽어 포맷을 결정합니다.
Redis log	redislog	파일 시작 부분을 읽어 포맷을 결정합니다.
CSV	csv	coma(,) 혹은 파이프(), 탭(\t), 세미콜론(;), Ctrl+A(\u0001)의 다음 구획 문자를 확인합니다. Ctrl-A는 Start Of Heading용 유니코드 관리 문자입니다.
Amazon Redshift	redshift	JDBC 연결을 사용하여 메타데이터를 가져옵니다.
MySQL	mysql	JDBC 연결을 사용하여 메타데이터를 가져옵니다.
PostgreSQL	postgresql	JDBC 연결을 사용하여 메타데이터를 가져옵니다.
Oracle 데이터베이스	oracle	JDBC 연결을 사용하여 메타데이터를 가져옵니다.
Microsoft SQL Server	sqlserver	JDBC 연결을 사용하여 메타데이터를 가져옵니다.
Amazon DynamoDB	dynamodb	DynamoDB 테이블에서 데이터를 읽습니다.

다음 형식의 파일은 분류될 수 있습니다.

- ZIP(단일 파일만 있는 아카이브에 지원됨). 다른 서비스에서는 Zip이 잘 지원되지 않습니다(아카이브 때문에).
- bzip
- GZIP
- LZ4
- Snappy(표준 및 Hadoop 네이티브 Snappy 형식 모두에서 지원됨)

기본 설정 CSV 분류자

기본 설정 CSV 분류자는 CSV 파일 내용을 파싱하여 AWS Glue 테이블 스키마를 결정합니다. 이 분류자는 다음 구분 기호를 확인합니다.

- 쉼표(,)
- 파이프(|)
- 탭(\t)
- 세미콜론(;)
- Ctrl-A (\u0001)

Ctrl-A는 Start Of Heading용 유니코드 관리 문자입니다.

CSV로써 분류되려면 테이블 스키마는 적어도 데이터의 두 개의 열과 두 개의 행이 있어야 합니다. CSV 분류자는 휴리스틱 숫자를 사용하여 헤더가 주어진 파일에 존재하는지 결정합니다. 분류자가 데이터의 첫 번째 행에서 헤더를 결정하지 못하면 열 헤더는 col1, col2, col3 등으로 보여집니다. 기본 설정 CSV 분류자는 다음 파일의 특성을 평가하여 헤더를 추론할지 여부를 결정합니다.

- 잠재적 헤더의 모든 열은 STRING(문자열) 데이터 유형으로 파싱합니다.
- 마지막 열을 제외하면 잠재적 헤더의 모든 열은 150 문자보다 적습니다. 후행 구분 기호를 허용하기 위해서는 마지막 열이 파일 전체로 비어 있을 수 있습니다.
- 잠재적 헤더의 모든 열은 열 이름 때문에 AWS Glue regex 요구 사항과 일치해야 합니다.
- 헤더는 데이터 행과 많이 달라야 합니다. 이를 결정하기 위해서는 하나 이상의 행이 STRING(문자열) 유형 아닌 것으로써 파싱해야 합니다. 모든 열이 문자열 유형이면 데이터 첫 번째 행은 헤더로써 사용되는 후속 행과 많이 다르지 않습니다.

Note

기본 설정 CSV 분류자는 원하는 AWS Glue 테이블을 생성하지 않으면 다음 대안 중 하나 이상을 사용할 수 있습니다.

- Data Catalog의 열 이름을 변경하고 SchemaChangePolicy를 LOG에 설정하고 파티션 출력 구성을 InheritFromTable로 설정하여 추후에 실행할 크롤러에 대비합니다.
- 사용자 지정 grok 분류자를 생성하여 데이터를 파싱하고 원하는 열을 지정합니다.
- 기본 설정 CSV 분류자는 유형 추론의 좋은 선택인 직렬화 라이브러리로서 LazySimpleSerDe를 참조하는 테이블을 생성합니다. 하지만 CSV 데이터가 따옴표로 묶인 문자열을 포함하면 테이블 정의를 편집하고 SerDe 라이브러리를 OpenCSVSerDe로 변경합니다. 추론된 유형을 STRING(문자열)로 조절하고 SchemaChangePolicy를 로그에 설치하고 파티션 출력 구성을 InheritFromTable로 설치하여 추후에 실행할 크롤러에 대비합니다. SerDe 라이브러리에 대한 자세한 내용은 Amazon Athena User Guide의 [SerDe Reference](#)를 참조하세요.

사용자 지정 분류자 작성

AWS Glue에서 데이터를 분류할 수 있도록 사용자 지정 분류자를 제공할 수 있습니다. You can create a custom classifier using a grok 패턴, XML 태그, JavaScript Object Notation(JSON) 또는 CSV(쉼표로 분리된 값)를 이용하여 사용자 지정 분류자를 생성할 수 있습니다. AWS Glue 크롤러가 사용자 지정 분류자를 호출합니다. 분류자가 데이터를 인식하면 크롤러에 데이터 분류 및 스키마를 반환합니다. 데이터가 내장된 분류자와 일치하지 않거나 크롤러가 생성한 테이블을 사용자 지정하기 원할 경우 사용자 지정 분류자를 정의해야 할 수도 있습니다.

AWS Glue 콘솔을 사용한 분류자 생성에 대한 자세한 내용은 [AWS Glue 콘솔에서 분류자 관련 작업을 참조하십시오](#).

AWS Glue는 사용자가 지정한 순서대로 내장된(기본) 분류자에 앞서 사용자 지정 분류자를 실행합니다. 크롤러가 데이터와 일치하는 분류자를 찾으면, 사용자의 AWS Glue Data Catalog에 작성되는 테이블 정의에 분류 문자열과 스키마를 사용합니다.

주제

- [Grok 사용자 지정 분류자 작성](#)
- [XML 사용자 지정 분류자 작성](#)
- [JSON 사용자 지정 분류자 작성](#)

• CSV 사용자 지정 분류자 작성

Grok 사용자 지정 분류자 작성

Grok는 텍스트 데이터의 패턴 일치 구문 분석에 사용하는 도구입니다. Grok 패턴은 한 번에 한 줄씩 데이터를 일치시킬 때 사용하는 명명된 정규식(regex) 세트입니다. AWS Glue는 grok 패턴을 사용하여 사용자 데이터의 스키마를 추정합니다. Grok 패턴과 사용자 데이터가 일치하는 경우 AWS Glue는 이 패턴을 사용해 데이터의 구조를 결정하고 이를 필드로 매핑합니다.

AWS Glue는 많은 기본 패턴을 제공하지만, 사용자가 직접 정의할 수도 있습니다. 기본 패턴을 사용하여 grok 패턴을 생성하거나 사용자 지정 분류자 정의에서 패턴을 사용자 지정할 수 있습니다. 사용자 지정 텍스트 파일 형식을 분류할 수 있도록 grok 패턴을 조정할 수 있습니다.

Note

AWS Glue grok 사용자 지정 분류자는 AWS Glue Data Catalog에서 생성된 테이블에 대해 GrokSerDe 직렬화 라이브러리를 사용합니다. AWS Glue Data Catalog를 Amazon Athena, Amazon EMR 또는 Redshift Spectrum에 사용하려는 경우 이러한 서비스에 대한 설명서에서 GrokSerDe 지원에 대한 정보를 확인하세요. 현재는 Amazon EMR 및 Redshift Spectrum에서 GrokSerDe로 생성된 테이블을 쿼리할 때 문제가 발생할 수 있습니다.

다음은 grok 패턴 구성 요소의 기본적인 구문입니다.

```
%{PATTERN:field-name}
```

명명된 PATTERN과 일치하는 데이터를 string이라는 기본 데이터 유형을 가진 스키마의 field-name 열로 매핑합니다. 선택적으로, 필드의 데이터 형식을 결과 스키마에서 byte, boolean, double, short, int, long 또는 float로 캐스트할 수 있습니다.

```
%{PATTERN:field-name:data-type}
```

예를 들면, num 필드를 int 데이터 유형으로 캐스트하기 위해 이 패턴을 사용할 수 있습니다.

```
%{NUMBER:num:int}
```

패턴을 다른 패턴으로 구성할 수 있습니다. 예를 들어 월, 해당 월의 날, 시간(예: Feb 1 06:25:43)의 패턴으로 정의되는 SYSLOG 타임스탬프의 패턴이 있을 수 있습니다. 이 데이터에서 다음 패턴을 정의할 수 있습니다.

```
SYSLOGTIMESTAMP %{MONTH} +%{MONTHDAY} %{TIME}
```

Note

Grok 패턴은 한 번에 한 줄씩만 처리할 수 있습니다. 여러 줄로 된 패턴을 지원하지 않습니다. 또 패턴 내부의 줄 바꿈도 지원하지 않습니다.

AWS Glue의 사용자 지정 분류자 값

Grok 분류자를 정의할 때 사용자 지정 분류자를 생성하기 위해 AWS Glue에 다음 값을 제공합니다.

명칭

분류자의 이름.

분류

special-logs 같이 분류되는 데이터의 형식을 설명하기 위해 작성하는 텍스트 문자열입니다.

Grok 패턴

일치 여부를 결정하기 위해 데이터 스토어에 적용하는 패턴 세트입니다. 이 패턴은 AWS Glue [기본 패턴](#) 및 사용자가 정의한 사용자 지정 패턴입니다.

다음은 grok 패턴의 예입니다.

```
%{TIMESTAMP_IS08601:timestamp} \[%{MESSAGEPREFIX:message_prefix}\]  
%{CRAWLERLOGLEVEL:loglevel} : %{GREEDYDATA:message}
```

데이터가 TIMESTAMP_IS08601와 일치하면 스키마 열인 timestamp가 생성됩니다. 동작은 예제의 다른 명명 패턴과 유사합니다.

사용자 지정 패턴

선택적으로 사용자가 정의한 사용자 지정 패턴입니다. 사용자 데이터를 분류하는 grok 패턴이 이 패턴을 참조합니다. 데이터에 적용되는 grok 패턴에서 이런 사용자 지정 패턴을 참조시킬 수 있습

니다. 각 사용자 지정 구성 요소 패턴은 별개 줄로 구성되어 있어야 합니다. [정규식\(regex\)](#) 구문을 사용하여 패턴을 정의합니다.

다음은 사용자 지정 패턴 사용에 대한 예입니다.

```
CRAWLERLOGLEVEL (BENCHMARK|ERROR|WARN|INFO|TRACE)
MESSAGEPREFIX .*-.*-.*-.*-.*
```

첫 번째 사용자 지정 명명 패턴인 CRAWLERLOGLEVEL은 데이터가 열거된 문자열 중 하나와 일치할 때 일치됩니다. 두 번째 사용자 지정 패턴인 MESSAGEPREFIX은 메시지 접두사 문자열과 일치할 시도합니다.

AWS Glue는 계속해서 생성 시간, 마지막 업데이트 시간, 분류자 버전을 추적합니다.

AWS Glue 기본 설정 패턴

AWS Glue는 사용자 지정 분류자 구축에 사용할 수 있는 많은 범용 패턴을 제공합니다. 분류자 정의의 grok pattern에 명명 패턴을 추가합니다.

다음 목록은 각 패턴이 한 줄씩 구성되어 있습니다. 각 줄의 패턴 이름에는 정의가 적용됩니다. [정규식\(regex\)](#) 구문은 패턴을 정의하는 데 사용됩니다.

```
#<noLOC>&GLU;</noLOC> Built-in patterns
USERNAME [a-zA-Z0-9._-]+
USER %{USERNAME:UNWANTED}
INT (?:[+-]?(?:[0-9]+))
BASE10NUM (?![0-9.+~])(?>[+-]?(?:[0-9]+(?:\.[0-9]+)?)|(?:\.[0-9]+)))
NUMBER (?:%{BASE10NUM:UNWANTED})
BASE16NUM (?![0-9A-Fa-f])(?:[+-]?(?:0x)?(?:[0-9A-Fa-f]+))
BASE16FLOAT \b(?![0-9A-Fa-f.])?(?:[+-]?(?:0x)?(?:[0-9A-Fa-f]+(?:\.[0-9A-Fa-f]*)?))|
(?:\.[0-9A-Fa-f]+))\b
BOOLEAN (?i)(true|false)

POSINT \b(?:[1-9][0-9]*)\b
NONNEGINT \b(?:[0-9]+)\b
WORD \b\w+\b
NOTSPACE \S+
SPACE \s*
DATA .*?
GREEDYDATA .*
```

```

#QUOTEDSTRING (?:(?<!\\)(?:\"(?:\\.|[^\"])*"|(?:(?:\\.|[^\'])*'|(?:`(?:\\.|[^\`])*`)))
QUOTEDSTRING (?>(?!\\)(?>\"(?>\\.|[^\"]+)+\"|\"\"|(?>'(?>\\.|[^\']+)+')|' '|(?>`(?)>\\.|[^\`]+)+`)|` `))
UUID [A-Fa-f0-9]{8}-(?:[A-Fa-f0-9]{4}-){3}[A-Fa-f0-9]{12}

# Networking
MAC (?:%{CISCOMAC:UNWANTED}|%{WINDOWSMAC:UNWANTED}|%{COMMONMAC:UNWANTED})
CISCOMAC (?:(?:[A-Fa-f0-9]{4}\.){2}[A-Fa-f0-9]{4})
WINDOWSMAC (?:(?:[A-Fa-f0-9]{2}-){5}[A-Fa-f0-9]{2})
COMMONMAC (?:(?:[A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2})
IPV6 ((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|(([0-9A-Fa-f]{1,4}:){6}(:[0-9A-Fa-f]{1,4}|
([0-9A-Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d)(\.((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d))
{3})|:))|([0-9A-Fa-f]{1,4}:){5}((:[0-9A-Fa-f]{1,4}){1,2})|:|((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d)
(\.((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d))){3})|:))|([0-9A-Fa-f]{1,4}:){4}(((:[0-9A-Fa-f]{1,4}){1,3})|
((:[0-9A-Fa-f]{1,4})?:((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d)(\.((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d))
{3}))|:))|([0-9A-Fa-f]{1,4}:){3}(((:[0-9A-Fa-f]{1,4}){1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d)
(\.((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d))){3}))|:))|([0-9A-Fa-f]{1,4}:){2}(((:[0-9A-Fa-f]{1,4}){1,5})|
((:[0-9A-Fa-f]{1,4}){0,3}:((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d)(\.((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d)
{3}))|:))|([0-9A-Fa-f]{1,4}:){1}(((:[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d)
(\.((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d))){3}))|:))|((:[0-9A-Fa-f]{1,4}){1,7})|((:[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d)
(\.((25[0-5]|2[0-4]\d|1\d\d|1[0-9]?\d))){3}))|:)))(%.+)?
IPV4 (?<![0-9])(?::(?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|[0-1]?[0-9]{1,2}))(![0-9])
IP (?:%{IPV6:UNWANTED}|%{IPV4:UNWANTED})
HOSTNAME \b(?:[0-9A-Za-z][0-9A-Za-z-_]{{0,62}}(?:\.(?:[0-9A-Za-z][0-9A-Za-z-_]
{{0,62}}))*(\.?\|b)
HOST %{{HOSTNAME:UNWANTED}}
IPORHOST (?:%{{HOSTNAME:UNWANTED}}|%{{IP:UNWANTED}})
HOSTPORT (?:%{{IPORHOST}}:%{{POSINT:PORT}})

# paths
PATH (?:%{{UNIXPATH}}|%{{WINPATH}})
UNIXPATH (?>/(?>[\w_!$@:.,~-]+|\\\.)*+
#UNIXPATH (?<![\w\|])(?:/[^\|s?]*)*+
TTY (?:/dev/(pts|tty(?:[pq]))?)(\w+)?/?(?:[0-9]+))
WINPATH (?>[A-Za-z]+:|\\)(?:\\[^\|?]*)*+
URIPROTO [A-Za-z]+(\+[A-Za-z+])?
URIHOST %{{IPORHOST}}(?::%{{POSINT:port}})?
# uripath comes loosely from RFC1738, but mostly from what Firefox
# doesn't turn into %XX

```

```

URIPATH (?:/[A-Za-z0-9$.+!*'(){}~,~;=@#%_\-]*)+
#URIPARAM \?(?:[A-Za-z0-9]+(?:=(?:[^\&]*))?(?:&(?:[A-Za-z0-9]+(?:=(?:[^\&]*))?)?)*)?
URIPARAM \?[A-Za-z0-9$.+!*'|(){}~,~@#%&/=;~?_\-[\]]*
URIPATHPARAM %{URIPATH}(?:%{URIPARAM})?
URI %{URIPROTO}://(?:%{USER}(?::[^\@]*)?@)?(?:%{URIHOST})?(?:%{URIPATHPARAM})?

# Months: January, Feb, 3, 03, 12, December
MONTH \b(?:Jan(?:uary)?|Feb(?:ruary)?|Mar(?:ch)?|Apr(?:il)?|May|Jun(?:e)?|Jul(?:y)?|
Aug(?:ust)?|Sep(?:tember)?|Oct(?:ober)?|Nov(?:ember)?|Dec(?:ember)?)\b
MONTHNUM (?:0?[1-9]|1[0-2])
MONTHNUM2 (?:0[1-9]|1[0-2])
MONTHDAY (?:0?[1-9])|(?:[12][0-9])|(?:3[01])|[1-9])

# Days: Monday, Tue, Thu, etc...
DAY (?:Mon(?:day)?|Tue(?:sday)?|Wed(?:nesday)?|Thu(?:rsday)?|Fri(?:day)?|
Sat(?:urday)?|Sun(?:day)?)

# Years?
YEAR (?:>\d\d){1,2}
# Time: HH:MM:SS
#TIME \d{2}:\d{2}(?::\d{2}(?:\.\d+)?)?
# TIME %{POSINT<24}:%{POSINT<60}(?::%{POSINT<60}(?:\.%{POSINT})?)?
HOUR (?:2[0123]|[01]?[0-9])
MINUTE (?:[0-5][0-9])
# '60' is a leap second in most time standards and thus is valid.
SECOND (?:0?[0-5]?[0-9]|60)(?:[:.,][0-9]+)?
TIME (?!<[0-9])%{HOUR}:%{MINUTE}(?::%{SECOND})(?![0-9])
# timestamp is YYYY/MM/DD-HH:MM:SS.UUUU (or something like it)
DATE_US %{MONTHNUM}[/-]%{MONTHDAY}[/-]%{YEAR}
DATE_EU %{MONTHDAY}[./-]%{MONTHNUM}[./-]%{YEAR}
DATESTAMP_US %{DATE_US}[- ]%{TIME}
DATESTAMP_EU %{DATE_EU}[- ]%{TIME}
ISO8601_TIMEZONE (?:Z|[+-%]{HOUR}(?::%{MINUTE}))
ISO8601_SECOND (?:%{SECOND}|60)
TIMESTAMP_ISO8601 %{YEAR}-%{MONTHNUM}-%{MONTHDAY}[T ]%{HOUR}:%{MINUTE}(?::??
%{SECOND})?%{ISO8601_TIMEZONE}?
TZ (?:[PMCE][SD]T|UTC)
DATESTAMP_RFC822 %{DAY} %{MONTH} %{MONTHDAY} %{YEAR} %{TIME} %{TZ}
DATESTAMP_RFC2822 %{DAY}, %{MONTHDAY} %{MONTH} %{YEAR} %{TIME} %{ISO8601_TIMEZONE}
DATESTAMP_OTHER %{DAY} %{MONTH} %{MONTHDAY} %{TIME} %{TZ} %{YEAR}
DATESTAMP_EVENTLOG %{YEAR}%{MONTHNUM2}%{MONTHDAY}%{HOUR}%{MINUTE}%{SECOND}
CISCOTIMESTAMP %{MONTH} %{MONTHDAY} %{TIME}

# Syslog Dates: Month Day HH:MM:SS

```



```

SYSLOGTIMESTAMP %{{MONTH}} +{{MONTHDAY}} %{{TIME}}
PROG (?:[\\w._/%-]+)
SYSLOGPROG %{{PROG:program}}(?:\\[{{POSINT:pid}}\\])?
SYSLOGHOST %{{IPORHOST}}
SYSLOGFACILITY <{{NONNEGINT:facility}}.{{NONNEGINT:priority}}>
HTTPDATE %{{MONTHDAY}}/%{{MONTH}}/%{{YEAR}}:{{TIME}} %{{INT}}

# Shortcuts
QS %{{QUOTEDSTRING:UNWANTED}}

# Log formats
SYSLOGBASE %{{SYSLOGTIMESTAMP:timestamp}} (?:%{{SYSLOGFACILITY}} )?%{{SYSLOGHOST:logsource}}
%{{SYSLOGPROG}}:

MESSAGESLOG %{{SYSLOGBASE}} %{{DATA}}

COMMONAPACHELOG %{{IPORHOST:clientip}} %{{USER:ident}} %{{USER:auth}}
\\[{{HTTPDATE:timestamp}}\\] "(?:%{{WORD:verb}} %{{NOTSPACE:request}}(?: HTTP/
%{{NUMBER:httpversion}})?|%{{DATA:rawrequest}})" %{{NUMBER:response}} (?:%{{Bytes:bytes=
%{{NUMBER}}|-})
COMBINEDAPACHELOG %{{COMMONAPACHELOG}} %{{QS:referrer}} %{{QS:agent}}
COMMONAPACHELOG_DATATYPED %{{IPORHOST:clientip}} %{{USER:ident;boolean}} %{{USER:auth}}
\\[{{HTTPDATE:timestamp;date;dd/MMM/yyyy:HH:mm:ss Z}}\\] "(?:%{{WORD:verb;string}}
%{{NOTSPACE:request}}(?: HTTP/%{{NUMBER:httpversion;float}})?|%{{DATA:rawrequest}})"
%{{NUMBER:response;int}} (?:%{{NUMBER:bytes;long}}|-)

# Log Levels
LOGLEVEL ([A|a]lert|ALERT|[T|t]race|TRACE|[D|d]ebug|DEBUG|[N|n]otice|NOTICE|[I|i]nfo|
INFO|[W|w]arn(?:ing)?|WARN(?:ING)?|[E|e]rr(?:or)?|ERR(?:OR)?|[C|c]rit(?:ical)?|
CRIT(?:ICAL)?|[F|f]atal|FATAL|[S|s]evere|SEVERE|EMERG(?:ENCY)?|[E|e]merg(?:ency)?)

```

XML 사용자 지정 분류자 작성

XML는 파일에서 태그를 사용해 문서의 구조를 정의합니다. XML 사용자 지정 분류자의 경우 사용자가 행을 정의할 때 사용하는 태그 이름을 지정할 수 있습니다.

AWS Glue의 사용자 지정 분류자 값

XML 분류자를 정의할 때 사용자 지정 분류자를 생성하기 위해 AWS Glue에 다음 값을 제공합니다. 이 분류자의 분류 필드를 xml1으로 설정합니다.

명칭

분류자의 이름.

행 태그

XML 문서의 테이블 행을 정의하는 XML 태그 이름입니다. 꺾쇠 괄호 < >는 사용하지 않습니다. 이름은 태그에 대한 XML 규칙을 따라야 합니다.

Note

행 데이터가 포함된 요소는 자기 복제를 하는 비어 있는 요소가 될 수 없습니다. 예를 들어 이 빈 요소는 AWS Glue에서 구문 분석되지 않습니다.

```
<row att1="xx" att2="yy" />
```

비어 있는 요소를 다음과 같이 작성할 수 없습니다.

```
<row att1="xx" att2="yy"> </row>
```

AWS Glue는 계속해서 생성 시간, 마지막 업데이트 시간, 분류자 버전을 추적합니다.

예를 들어 다음과 같은 XML 파일이 있다고 가정하겠습니다. 작성자와 제목 열만 포함된 AWS Glue 테이블을 생성하려면 AWS Glue 콘솔에서 Row tag(행 태그)가 AnyCompany인 분류자를 생성합니다. 그런 다음 이 사용자 지정 분류자를 사용하는 크롤러를 추가해 실행합니다.

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <AnyCompany>
      <author>Rivera, Martha</author>
      <title>AnyCompany Developer Guide</title>
    </AnyCompany>
  </book>
  <book id="bk102">
    <AnyCompany>
```

```

    <author>Stiles, John</author>
    <title>Style Guide for AnyCompany</title>
  </AnyCompany>
</book>
</catalog>

```

JSON 사용자 지정 분류자 작성

JSON은 데이터 교환 형식입니다. 이름 값 쌍이나 순서가 지정된 값 목록을 가진 데이터 구조를 정의합니다. JSON 사용자 지정 분류자의 경우 데이터 구조에 JSON 경로를 지정해 사용자 테이블의 스키마를 정의할 수 있습니다.

AWS Glue의 사용자 지정 분류자 값

JSON 분류자를 정의할 때 사용자 지정 분류자를 생성하기 위해 AWS Glue에 다음 값을 제공합니다. 이 분류자의 분류 필드를 json으로 설정합니다.

명칭

분류자의 이름.

JSON 경로

테이블 스키마 정의에 사용하는 객체를 가리키는 JSON 경로입니다. JSON 경로 작성 시 점이나 대괄호를 사용할 수 없습니다. 다음의 연산자가 지원됩니다:

설명

JSON 객체의 루트 요소입니다. 모든 경로 표현식은

와일드카드 문자로 시작됩니다. JSON 경로의 경우 위치와 무관하게 이름이나 숫자를 사용해야 합니다.

점으로 표기된 하위 필드. JSON 객체의 하위 필드를 지정합니다.

괄호로 표기된 하위 필드. JSON 객체의 하위 필드를 지정합니다. 하위 필드 1개만 지정할 수 있습니다.

어레이 인덱스. 인덱스 별로 어레이 값을 지정합니다.

AWS Glue는 계속해서 생성 시간, 마지막 업데이트 시간, 분류자 버전을 추적합니다.

Example 배열에서 레코드를 가져오는 데 JSON 분류자 사용

JSON 데이터가 레코드 배열이라고 가정하겠습니다. 예를 들어, 파일의 처음 몇 줄은 다음과 비슷할 수 있습니다.

```
[
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:ak",
    "name": "Alaska"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:al\cd:1",
    "name": "Alabama's 1st congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:al\cd:2",
    "name": "Alabama's 2nd congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:al\cd:3",
    "name": "Alabama's 3rd congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:al\cd:4",
    "name": "Alabama's 4th congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:al\cd:5",
    "name": "Alabama's 5th congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:al\cd:6",
    "name": "Alabama's 6th congressional district"
  },
],
```

```
[
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:al\cd:7",
    "name": "Alabama's 7th congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:ar\cd:1",
    "name": "Arkansas's 1st congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:ar\cd:2",
    "name": "Arkansas's 2nd congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:ar\cd:3",
    "name": "Arkansas's 3rd congressional district"
  },
  {
    "type": "constituency",
    "id": "ocd-division\country:us\state:ar\cd:4",
    "name": "Arkansas's 4th congressional district"
  }
]
```

기본 JSON 분류자를 사용하는 크롤러를 실행할 때는 스키마 정의에 전체 파일이 사용됩니다. JSON 경로를 지정하지 않기 때문에 크롤러는 데이터를 하나의 객체로 처리하며, 이것이 배열입니다. 예를 들어, 스키마의 모양은 다음과 같을 수 있습니다.

```
root
|-- record: array
```

하지만 JSON 어레이의 각 레코드를 토대로 스키마를 생성하려면 사용자 지정 JSON 분류자를 생성하고 JSON 경로를 `$[*]`로 지정합니다. 이 JSON 경로를 지정하는 경우, 분류자는 어레이의 12개 레코드를 모두 조사해 스키마를 결정합니다. 그 결과로 생성된 스키마에는 다음 예제와 유사하게 각 객체의 필드가 분리되어 포함됩니다.

```
root
```

```
|-- type: string
|-- id: string
|-- name: string
```

Example 파일의 일부만을 검사하는 데 JSON 분류자 사용

JSON 데이터가 <http://everypolitician.org>에서 가져온 예제 JSON 파일 `s3://awsglue-datasets/examples/us-legislators/all/areas.json`의 패턴을 따른다고 가정합니다. JSON 파일의 예제 객체는 다음과 같습니다.

```
{
  "type": "constituency",
  "id": "ocd-division/country:us/state:ak",
  "name": "Alaska"
}
{
  "type": "constituency",
  "identifiers": [
    {
      "scheme": "dmoz",
      "identifier": "Regional/North_America/United_States/Alaska/"
    },
    {
      "scheme": "freebase",
      "identifier": "\/m\/0hjy"
    },
    {
      "scheme": "fips",
      "identifier": "US02"
    },
    {
      "scheme": "quora",
      "identifier": "Alaska-state"
    },
    {
      "scheme": "britannica",
      "identifier": "place/Alaska"
    },
    {
      "scheme": "wikidata",
      "identifier": "Q797"
    }
  ]
}
```

```

],
"other_names": [
  {
    "lang": "en",
    "note": "multilingual",
    "name": "Alaska"
  },
  {
    "lang": "fr",
    "note": "multilingual",
    "name": "Alaska"
  },
  {
    "lang": "nov",
    "note": "multilingual",
    "name": "Alaska"
  }
],
"id": "ocd-division/country:us/state:ak",
"name": "Alaska"
}

```

기본 JSON 분류자를 사용하는 크롤러를 실행할 때는 스키마를 생성하는 데 전체 파일이 사용됩니다. 그러면 다음과 같은 스키마가 생성될 수 있습니다.

```

root
|-- type: string
|-- id: string
|-- name: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string

```

하지만 "id" 객체만을 사용하는 스키마를 생성하려면 사용자 지정 JSON 분류자를 생성하고 JSON 경로를 \$.id로 지정합니다. 그러면 스키마는 "id" 필드에만 토대를 둡니다.

```
root
|-- record: string
```

이 스키마에서 추출한 데이터의 첫 몇 줄은 다음과 같습니다.

```
{"record": "ocd-division/country:us/state:ak"}
{"record": "ocd-division/country:us/state:al/cd:1"}
{"record": "ocd-division/country:us/state:al/cd:2"}
{"record": "ocd-division/country:us/state:al/cd:3"}
{"record": "ocd-division/country:us/state:al/cd:4"}
{"record": "ocd-division/country:us/state:al/cd:5"}
{"record": "ocd-division/country:us/state:al/cd:6"}
{"record": "ocd-division/country:us/state:al/cd:7"}
{"record": "ocd-division/country:us/state:ar/cd:1"}
{"record": "ocd-division/country:us/state:ar/cd:2"}
{"record": "ocd-division/country:us/state:ar/cd:3"}
{"record": "ocd-division/country:us/state:ar/cd:4"}
{"record": "ocd-division/country:us/state:as"}
{"record": "ocd-division/country:us/state:az/cd:1"}
{"record": "ocd-division/country:us/state:az/cd:2"}
{"record": "ocd-division/country:us/state:az/cd:3"}
{"record": "ocd-division/country:us/state:az/cd:4"}
{"record": "ocd-division/country:us/state:az/cd:5"}
{"record": "ocd-division/country:us/state:az/cd:6"}
{"record": "ocd-division/country:us/state:az/cd:7"}
```

JSON 파일의 "identifier" 값이 중첩된 객체를 토대로 스키마를 생성하려면 사용자 지정 JSON 분류자를 생성하고 JSON 경로를 \$.identifiers[*].identifier로 지정합니다. 이 스키마는 이전 예제와 유사하지만, JSON 파일에서 전혀 다른 객체에 토대를 두고 있습니다.

이런 스키마는 다음과 같습니다.

```
root
|-- record: string
```

테이블에서 데이터의 첫 몇 줄은 스키마가 "identifier" 객체에 토대를 두고 있다는 점을 보여줍니다.


```

{"record": "Regional/North_America/United_States/Alaska/"}
{"record": "/m/0hjy"}
{"record": "US02"}
{"record": "5879092"}
{"record": "4001016-8"}
{"record": "destination/alaska"}
{"record": "1116270"}
{"record": "139487266"}
{"record": "n79018447"}
{"record": "01490999-8dec-4129-8254-eef6e80fadc3"}
{"record": "Alaska-state"}
{"record": "place/Alaska"}
{"record": "Q797"}
{"record": "Regional/North_America/United_States/Alabama/"}
{"record": "/m/0gyh"}
{"record": "US01"}
{"record": "4829764"}
{"record": "4084839-5"}
{"record": "161950"}
{"record": "131885589"}

```

JSON 파일의 "name" 배열의 "other_names" 필드 같이 다른 중첩된 객체를 토대로 테이블을 생성하려면 사용자 지정 JSON 분류자를 생성하고 JSON 경로를 \$.other_names[*].name로 지정합니다. 이 스키마는 이전 예제와 유사하지만, JSON 파일에서 전혀 다른 객체에 토대를 두고 있습니다. 이런 스키마는 다음과 같습니다.

```

root
|-- record: string

```

이 테이블의 처음 몇 줄 데이터는 "name" 배열 내 "other_names" 객체의 데이터에 근거함을 보여줍니다.

```

{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Аляска"}
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Alaska"}

```

```

{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "#####"}
{"record": "#####"}
{"record": "#####"}
{"record": "Alaska"}
{"record": "Alyaska"}
{"record": "Alaska"}
{"record": "Alaska"}
{"record": "Штат Аляска"}
{"record": "Аляска"}
{"record": "Alaska"}
{"record": "#####"}

```

CSV 사용자 지정 분류자 작성

사용자 지정 CSV 분류자를 사용하면 사용자 지정 csv 분류자 필드의 각 열에 대한 데이터 형식을 지정할 수 있습니다. 콤마로 분리된 각 열의 데이터 유형을 지정할 수 있습니다. 데이터 유형을 지정하면 크롤러가 추론한 데이터 유형을 재정의하고 데이터가 적절하게 분류되도록 할 수 있습니다.

분류자에서 CSV를 처리하기 위한 Serde를 설정할 수 있습니다. 이는 데이터 카탈로그에서 적용됩니다.

사용자 지정 분류기를 만들면 분류기를 다른 크롤러에 다시 사용할 수도 있습니다.

- 헤더만 있는(데이터 없음) csv 파일의 경우, 정보가 충분하지 않기 때문에 이러한 파일은 UNKNOWN으로 분류됩니다. 열 제목 옵션에서 CSV에 '제목 포함'을 지정하고 데이터 유형을 제공하면 이러한 파일을 올바르게 분류할 수 있습니다.

사용자 지정 CSV 분류자를 사용하여 다양한 유형의 CSV 데이터 스키마를 추론할 수 있습니다. 분류자에 제공할 수 있는 사용자 지정 속성에는 구분 기호, CSV SerDe 옵션, 헤더에 대한 옵션, 데이터에서 특정 검증을 수행할 수 있는지 여부가 포함되어 있습니다.

AWS Glue의 사용자 지정 분류자 값

CSV 분류자를 정의할 때 분류자를 생성하기 위해 AWS Glue에 다음 값을 제공합니다. 이 분류자의 분류 필드를 csv으로 설정합니다.

분류자 이름

분류자의 이름.

CSV Serde

분류자에서 CSV를 처리하기 위한 Serde를 설정합니다. 이는 데이터 카탈로그에서 적용됩니다. 옵션은 Open CSV SerDe, Lazy Simple SerDe, 없음과 같습니다. 크롤러에서 감지하려는 경우 없음 값을 지정할 수 있습니다.

열 구분 기호

행의 열 입력 항목 각각을 구분하는 것을 나타내기 위한 사용자 지정 기호입니다. 유니코드 문자를 제공합니다. 구분 기호를 입력할 수 없는 경우 구분 기호를 복사하여 붙여넣을 수 있습니다. 이 방법은 시스템에서 지원하지 않는 문자(일반적으로 □로 표시)를 포함하여 인쇄 가능한 문자에도 사용할 수 있습니다.

인용 기호

단일 열 값에 내용을 결합하는 것을 나타내기 위한 사용자 지정 기호입니다. 열 구분 기호와 달라야 합니다. 유니코드 문자를 제공합니다. 구분 기호를 입력할 수 없는 경우 구분 기호를 복사하여 붙여넣을 수 있습니다. 이 방법은 시스템에서 지원하지 않는 문자(일반적으로 □로 표시)를 포함하여 인쇄 가능한 문자에도 사용할 수 있습니다.

열 제목

열 제목을 CSV 파일에서 어떻게 탐지해야 하는지에 대한 행동을 표시합니다. 사용자 지정 CSV 파일에 열 제목이 포함되어 있는 경우에는 쉼표로 구분된 열 제목 목록을 입력합니다.

처리 옵션: 단일 열에서 파일 허용

오직 하나의 열만 포함하는 파일을 처리할 수 있도록 합니다.

처리 옵션: 열 값을 식별하기 전에 공백 트리밍

열 값의 유형을 식별하기 전에 값의 트리밍 여부를 지정합니다.

사용자 지정 데이터 유형 - 선택 사항

쉼표로 분리된 사용자 지정 데이터 유형을 입력합니다. CSV 파일의 사용자 지정 데이터 유형을 지정합니다. 사용자 지정 데이터 유형은 지원되는 데이터 유형이어야 합니다. 지원되는 데이터 유형은 “바이너리”, “부울”, “날짜”, “십진수”, “더블”, “플로트”, “정수”, “롱”, “쇼트”, “문자열”, “타임스탬프”입니다. 지원되지 않는 데이터 유형은 오류를 표시합니다.

AWS Glue 콘솔에서 분류자 관련 작업

분류자는 데이터의 스키마를 결정합니다. 사용자 지정 분류자를 작성하고 AWS Glue에서 분류자를 가리킬 수 있습니다.

분류자 보기

AWS Glue 콘솔을 <https://console.aws.amazon.com/glue/>에서 열고 [분류자(Classifiers)] 탭을 선택하여 생성한 모든 분류자에 대한 목록을 봅니다.

분류자 목록은 각 분류자에 대한 다음 속성을 표시합니다.

- 분류자 - 분류자 이름입니다. 분류자를 생성할 때는 분류자 이름을 제공해야 합니다.
- [분류(Classification)] - 분류자가 유추한 테이블의 분류자 유형입니다.
- [마지막 업데이트(Last updated)] - 이 분류자가 업데이트된 마지막 시간입니다.

분류자 관리

AWS Glue 콘솔의 [분류자(Classifiers)] 목록에서 분류자를 추가, 편집 및 삭제할 수 있습니다. 목록에서 분류자 이름을 선택하여 분류자에 대한 더 자세한 정보를 알아봅니다. 세부 정보는 분류자를 생성할 때 정의한 정보를 포함합니다.

분류자 생성

AWS Glue 콘솔에서 분류자를 추가하려면 [분류자 추가(Add classifier)]를 선택합니다. 분류자를 정의할 때 다음에 대한 값을 제공합니다.

- [분류자 이름(Classifier name)] - 분류자에 대한 고유 이름을 입력합니다.
- [분류자 유형(Classifier type)] - 분류자가 유추한 테이블의 분류자 유형입니다.
- [마지막 업데이트(Last updated)] - 이 분류자가 업데이트된 마지막 시간입니다.

분류자 이름

분류자 고유 이름을 입력합니다.

분류자 유형

생성할 분류자의 유형을 선택합니다.

선택한 분류기 유형에 따라 분류기에 대해 다음 속성을 구성합니다.

Grok

- 분류

분류된 데이터의 포맷 또는 유형을 설명하거나 사용자 정의 레이블을 제공합니다.

- Grok 패턴

데이터를 구조화 스키마로 구분 분석하는 데 사용됩니다. Grok 패턴은 데이터 스토어의 포맷을 보여주는 이름이 붙여진 패턴으로 구성됩니다. AWS Glue에서 제공하는 명명된 기본 제공 패턴과 [사용자 정의 패턴(Custom patterns)] 필드에 작성하고 포함하는 사용자 정의 패턴을 사용하여 이 grok 패턴을 작성합니다. Grok 디버거 결과가 AWS Glue 결과와 정확하게 일치하지 않더라도 grok 디버거를 통한 몇 가지 샘플 데이터를 사용하여 패턴을 시도해 볼 수 있습니다. Grok 디버거는 웹에서 볼 수 있습니다. AWS Glue가 제공한 이름이 붙여진 기본 설정 패턴은 웹에서 사용 가능한 grok 패턴과 일반적으로 호환됩니다.

반복적으로 이름이 붙여진 패턴을 추가하여 Grok 패턴을 만들고 디버거에서 결과를 확인합니다. 이 활동은 AWS Glue 크롤러가 grok 패턴을 실행할 때 신뢰를 줘 데이터가 구문 분석될 수 있습니다.

- 사용자 지정 패턴

Grok 분류자의 경우, 이것은 여러분이 작성하는 Grok pattern(Grok 패턴)의 조건부 빌딩 블록입니다. 기본 설정 패턴이 데이터를 구분 분석하지 못할 경우, 사용자 지정 패턴을 작성해야 할 수도 있습니다. 이런 사용자 지정 패턴은 이 필드에서 정의되고 Grok pattern(Grok 패턴) 필드에서 참조됩니다. 각 사용자 지정 패턴은 개별 라인에서 정의됩니다. 기본 설정 패턴과 같이 [regular expression\(정규식\) \(regex\)](#) 구문을 사용하는 이름이 붙여진 패턴 정의로 구성됩니다.

예를 들어, 다음은 뒤이어 정규식 정의가 있는 MESSAGEPREFIX 이름을 붙여 데이터를 적용하고 데이터가 패턴을 따르는지 결정합니다.

```
MESSAGEPREFIX .*-.*-.*-.*-.*
```

XML

- 행 태그

XML 분류자를 위해, 이것은 XML 문서의 테이블 행을 정의하는 XML 태그 이름입니다. < > 꺾쇠 괄호없이 이름을 입력합니다. 이름은 태그에 대한 XML 규칙을 따라야 합니다.

자세한 내용은 [XML 사용자 지정 분류자 작성](#) 단원을 참조하십시오.

JSON

- JSON 경로

JSON 분류자를 위해, 이것은 생성된 테이블 행을 정의하는 객체, 배열 및 값까지의 JSON 경로입니다. AWS Glue가 지원한 연산자를 사용하여 점 혹은 괄호 JSON 구문 이름을 입력합니다.

자세한 내용은 [JSON 사용자 지정 분류자 작성](#)의 연산자 목록을 참조하십시오.

CSV

- 열 구분 기호

행의 열 입력 항목 각각을 구분하는 것을 나타내기 위한 단일 문자 또는 기호입니다. 목록에서 구분 기호를 선택하거나 Other를 선택하여 사용자 정의 구분 기호를 입력합니다.

- 인용 기호

단일 열 값에 내용을 결합하는 것을 나타내기 위한 단일 문자 또는 기호입니다. 열 구분 기호와 달라야 합니다. 목록에서 따옴표 기호를 선택하거나 Other를 선택하여 사용자 정의 따옴표 문자를 입력합니다.

- 열 제목

열 제목을 CSV 파일에서 어떻게 탐지해야 하는지에 대한 행동을 표시합니다. Has headings, No headings 또는 Detect headings를 선택할 수 있습니다. 사용자 지정 CSV 파일에 열 제목이 포함되어 있는 경우에는 쉼표로 구분된 열 제목 목록을 입력합니다.

- 단일 열이 있는 파일 허용

CSV로 분류하려면 데이터는 적어도 데이터의 두 개의 열과 두 개의 행이 있어야 합니다. 하나의 열만 포함하는 파일의 처리를 허용하려면 이 옵션을 사용합니다.

- 열 값을 식별하기 전에 공백 트리밍

이 옵션은 열 값의 유형을 식별하기 전에 값의 트리밍 여부를 지정합니다.

- 사용자 지정 데이터 유형

(선택 사항) - 쉼표로 구분된 목록에 사용자 지정 데이터 유형을 입력합니다. 지원되는 데이터 유형은 “바이너리”, “부울”, “날짜”, “십진수”, “더블”, “플로트”, “정수”, “롱”, “쇼트”, “문자열”, “타임스탬프”입니다.

- CSV Serde

(선택 사항) - 분류자에서 CSV를 처리하기 위한 Serde입니다. 이는 데이터 카탈로그에서 적용됩니다. Open CSV SerDe, Lazy Simple SerDe 또는 None 중에서 선택합니다. 크롤러에서 감지하려는 경우 None 값을 지정할 수 있습니다.

자세한 내용은 [사용자 지정 분류자 작성](#) 단원을 참조하십시오.

AWS Glue 크롤러 일정 관리

요구에 따라 정기 일정에 따라 AWS Glue 크롤러를 실행합니다. 크롤러 일정은 con 포맷에 나타납니다. 자세한 내용은 Wikipedia의 [cron](#)을 참조하십시오.

일정에 따라 크롤러를 생성할 경우 1주일에 어떤 요일 어떤 시간에 실행할지 빈도수와 같은 특정 제약점을 지정할 수 있습니다. 이 제약점은 CRON을 기반으로 합니다. 크롤러 일정을 설정하고자 한다면 CRON의 기능 및 제약점을 고려해야 합니다. 예를 들어, 매월 31일에 크롤러를 실행하고자 한다면 매월 31일이 없다는 점을 유의하기 바랍니다.

각 크롤러의 크롤링은 최대 12개월 동안만 유효합니다

작업 및 크롤러의 일정을 관리하기 위해 cron을 사용하는 방법에 대한 자세한 내용은 [작업 및 크롤러를 위한 시간 기반 일정](#) 섹션을 참조하세요.

크롤러 결과 및 세부 정보 보기

크롤러가 성공적으로 실행되면 데이터 카탈로그에 테이블 정의가 생성됩니다. 탐색 창에서 [테이블 (Tables)] 탭을 선택하여 지정한 데이터베이스의 크롤러가 생성한 테이블을 살펴봅니다.

다음과 같이 크롤러 자체와 관련된 정보를 볼 수 있습니다.

- AWS Glue 콘솔의 [크롤러(Crawlers)] 페이지에는 크롤러에 대한 다음 속성이 표시됩니다.

속성	설명
이름	크롤러를 생성하는 경우, 고유 이름을 붙여야 합니다.
상태	크롤러는 준비, 시작, 중지, 예약 및 예약 중지 상태일 수 있습니다. 실행 중인 크롤러는 시작에서 중지까지 진행합니다. 크롤러 예약을 재개하거나 중단할 수 있습니다.

속성	설명
Schedule	필요에 따라 크롤러를 실행하거나 일정 빈도수를 선택합니다. 크롤러 예약에 대한 자세한 내용은 크롤러 일정 관리 를 참조하십시오.
마지막 실행(Last run)	크롤러가 마지막으로 실행된 날짜 및 시간입니다.
로그(Log)	크롤러의 마지막 실행에 대한 사용 가능한 로그에 대한 링크.
마지막 실행 이후의 테이블 변경 사항(Tables changes from last run)	크롤러의 마지막 실행에 따라 업데이트된 AWS Glue Data Catalog의 테이블 수입입니다.

- 크롤러의 기록을 보려면 탐색 창에서 Crawlers(크롤러)를 선택하여 생성한 크롤러를 확인합니다. 사용 가능한 크롤러 목록에서 크롤러를 선택합니다. Crawler runs(크롤러 실행) 탭에서 크롤러 속성과 크롤러 기록을 볼 수 있습니다.

Crawler runs(크롤러 실행) 탭에는 Start time (UTC)(시작 시간(UTC)), End time (UTC)(종료 시간(UTC)), Duration(기간), Status(상태), DPU hours(DPU 시간) 및 Table changes(테이블 변경)를 비롯하여 크롤러가 실행된 각 시간에 대한 정보가 표시됩니다.

크롤러 실행 탭은 크롤러 기록 기능의 시작 날짜 이후 발생한 크롤만 표시하고 최대 12개월의 크롤만 유지합니다. 이전의 크롤은 반환되지 않습니다.

- 추가 정보를 보려면 크롤러 세부 정보 페이지에서 탭을 선택합니다. 각 탭에는 크롤러 관련 정보가 표시됩니다.
 - Schedule(일정): 크롤러에 대해 생성된 모든 일정이 여기에 표시됩니다.
 - Data sources(데이터 소스): 크롤러에서 스캔한 모든 데이터 소스가 여기에 표시됩니다.
 - Classifiers(분류자): 크롤러에 할당된 모든 분류자가 여기에 표시됩니다.
 - Tags(태그): 생성되고 AWS 리소스에 할당된 모든 태그가 여기에 표시됩니다.

크롤러가 데이터 카탈로그 테이블에 설정한 파라미터

이러한 테이블 속성은 AWS Glue 크롤러가 설정합니다. 사용자가 classification 및 compressionType 속성을 소비할 것으로 기대합니다. 테이블 크기 추정치를 비롯한 기타 속성은 내부 계산에 사용되며, 당사는 이러한 속성의 정확성이나 고객 사용 사례에 대한 적용 가능성을 보장하지

않습니다. 이러한 파라미터를 변경하면 크롤러의 동작이 변경될 수 있지만 이 워크플로는 지원되지 않습니다.

속성 키	속성 값
UPDATED_BY_CRAWLER	업데이트를 수행하는 크롤러의 이름입니다.
connectionName	크롤러가 데이터 스토어에 연결하는 데 사용되는 데이터 카탈로그의 연결 이름입니다.
recordCount	파일 크기 및 헤더를 기반으로 테이블의 레코드 수를 추정합니다.
skip.header.line.count	헤더를 건너뛰기 위해 행을 건너뛰었습니다. CSV로 분류된 테이블에 설정합니다.
CrawlerSchemaSerializerVersion	내부용
classification	크롤러에서 추론한 데이터 형식입니다. AWS Glue에 의해 지원되는 형식에 대한 자세한 내용은 the section called “AWS Glue의 기본 설정 분류자” 단원을 참조하세요.
CrawlerSchemaDeserializerVersion	내부용
sizeKey	테이블의 파일 크기를 합산하여 크롤링했습니다.
averageRecordSize	테이블 열의 평균 크기(바이트).
compressionType	테이블의 데이터에 사용되는 압축 유형입니다. AWS Glue 크롤러가 지원하는 압축 유형에 대한 자세한 내용은 the section called “AWS Glue의 기본 설정 분류자” 을 참조하세요.
typeOfData	file, table, 또는 view.
objectCount	테이블의 Amazon S3 경로 아래에 있는 객체 수입니다.

이러한 추가 테이블 속성은 Snowflake 데이터 스토어의 AWS Glue 크롤러에 의해 설정됩니다.

속성 키	속성 값
aws:RawTableLastAltered	Snowflake 테이블의 마지막으로 변경된 타임스탬프를 기록합니다.
ViewOriginalText	SQL 문을 봅니다.
ViewExpandedText	Base64 형식으로 인코딩된 SQL 문을 볼 수 있습니다.
ExternalTable:S3Location	Snowflake 외부 테이블의 Amazon S3 위치입니다.
ExternalTable:FileFormat	Snowflake 외부 테이블의 Amazon S3 파일 형식입니다.

이러한 추가 테이블 속성은 Amazon Redshift, Microsoft SQL Server, MySQL, PostgreSQL 및 Oracle과 같은 JDBC 유형 데이터 스토어의 AWS Glue 크롤러에 의해 설정됩니다.

속성 키	속성 값
aws:RawType	크롤러가 데이터 카탈로그에 데이터를 저장하면 데이터 유형이 Hive 호환 유형으로 변환되므로 기본 데이터 유형에 대한 정보가 손실되는 경우가 많습니다. 크롤러는 aws:RawType 파라미터를 출력하여 기본 수준 데이터 유형을 제공합니다.
aws:RawColumnComment	설명이 데이터베이스의 열과 연결되는 경우 크롤러는 카탈로그 테이블에 해당 설명을 출력합니다. 주석 문자열은 255바이트로 잘립니다. Microsoft SQL Server에서는 설명이 지원되지 않습니다.
aws:RawTableComment	설명이 데이터베이스의 테이블과 연결되는 경우 크롤러는 카탈로그 테이블에 해당 설명을 출력합니다. 설명 문자열은 255바이트로 잘립니다.

속성 키	속성 값
	Microsoft SQL Server에서는 설명이 지원되지 않습니다.

크롤러 동작 사용자 지정

크롤러가 실행되면 이전 크롤러와 다른 스키마와 파티션을 초래하는 데이터 스토어의 변화가 발생합니다. AWS Management Console 또는 AWS Glue API를 사용하여 크롤러가 특정 변화에 따라 어떤 절차를 밟는지 알아봅니다.

주제

- [새 파티션을 추가하기 위한 점진적인 크롤링](#)
- [파티션 인덱스 크롤러 구성 옵션 설정](#)
- [Amazon S3 이벤트 알림을 사용하여 크롤링 가속화](#)
- [크롤러가 기존 스키마를 변경하지 않도록 예방하는 방법](#)
- [각 Amazon S3 포함 경로에 대해 단일 스키마를 생성하는 방법](#)
- [테이블 위치와 파티션 수준을 지정하는 방법](#)
- [크롤러가 생성할 수 있는 최대 테이블 수를 지정하는 방법](#)
- [Delta Lake 데이터 스토어에 대한 구성 옵션을 지정하는 방법](#)
- [Lake Formation 자격 증명을 사용하도록 크롤러를 구성하는 방법](#)

Console

AWS Glue 콘솔을 사용하여 크롤러를 정의하면 크롤러의 행태를 구성할 수 있는 몇 가지 옵션이 있습니다. AWS Glue 콘솔을 사용하여 크롤러를 추가하는 방법에 대한 자세한 내용은 [크롤러 구성](#) 단원을 참조하십시오.

이전에 크롤된 데이터 스토어에 반해 크롤러가 실행되면 스키마가 변했거나 데이터 스토어의 어떤 객체가 삭제되었다는 점을 알게 됩니다. 크롤러 로그는 스키마로 변합니다. 크롤러의 원본 유형에 따라 스키마 변경 정책과 상관없이 새 테이블과 파티션이 생성될 수 있습니다.

스키마가 변하면 크롤러가 무엇을 하는지 지정하기 위해서 콘솔의 다음 작업 중 하나를 선택할 수 있습니다.

- Data Catalog에서 테이블 정의 업데이트 - 새로운 열을 추가하고 누락 열을 제거한 다음 AWS Glue Data Catalog의 기존 열 정의를 수정합니다. 크롤러가 설정하지 않은 메타데이터를 제거합니다. 이것이 기본 설정입니다.
- 새 열만 추가 - Amazon S3 데이터 스토어의 매핑인 테이블을 위해 새로운 열이 검색되면 추가하지만 Data Catalog에 존재하는 열의 유형을 제거하거나 변형하지 않습니다. Data Catalog의 현재 열이 올바르고 크롤러가 기존 열의 유형을 제거하거나 변하지 않기 원한다면 이 옵션을 선택합니다. 분류, 요약 유형, CSV 구분자 등 기본 Amazon S3 테이블 속성이 변한다면 테이블이 사용되지 않음이라고 표시합니다. Data Catalog에 존재하는 입력 및 출력 포맷을 유지합니다. 크롤러에 설정된 피라미터이면 SerDe 파라미터를 업데이트합니다. 다른 모든 데이터 스토어는 기존 열 정의를 변경합니다.
- 변경 사항을 무시하고, 에서 테이블을 업데이트하지 않음 - 새로운 테이블과 파티션만이 생성됩니다.

중분 크롤링에 대한 기본 설정입니다.

크롤러는 새롭고 변한 파티션을 발견할 수 있습니다. 기본적으로 새 파티션이 추가되고 기존 파티션이 변했다면 업데이트됩니다. 또한 AWS Glue 콘솔에서 크롤러 구성 옵션을 [테이블의 메타데이터로 모든 신규 및 기존 파티션 업데이트(Update all new and existing partitions with metadata from the table)]로 설정할 수 있습니다. 이 옵션이 설정되면 파티션은 상위 테이블에서 분류, 입력 포맷, 출력 포맷, SerDe 정보, 스키마 등의 메타데이터 속성을 상속합니다. 테이블 속성의 모든 변경 사항은 파티션으로 전파됩니다. 이 구성 옵션이 기존 크롤러에 설정되면 기존 파티션은 크롤러가 그 다음번에 실행될 때 상위 테이블의 속성과 일치하기 위해서 업데이트됩니다.

삭제된 객체가 데이터 스토어에 존재한다면 크롤러가 무엇을 하는지 지정하기 위해서 다음 작업 중 하나를 선택할 수 있습니다.

- Data Catalog에서 테이블 및 파티션 삭제
- 변경 사항을 무시하고 Data Catalog에서 테이블을 업데이트하지 않음

중분 크롤링에 대한 기본 설정입니다.

- Data Catalog에서 사용되지 않는 테이블로 표시(Mark the table as deprecated in the Data Catalog) - 이것은 기본 설정입니다.

AWS CLI

```
aws glue create-crawler \
--name "your-crawler-name" \
```

```
--role "your-iam-role-arn" \
--database-name "your-database-name" \
--targets 'S3Targets=[{Path="s3://your-bucket-name/path-to-data"}]' \
--configuration '{"Version": 1.0, "CrawlerOutput": {"Partitions":
{"AddOrUpdateBehavior": "InheritFromTable"}, "Tables": {"AddOrUpdateBehavior":
"MergeNewColumns"}}}'
```

API

AWS Glue API를 사용하여 크롤러를 정의하면 크롤러를 구성할 수 있는 몇 가지 필드 중에서 선택할 수 있습니다. 크롤러 API의 SchemaChangePolicy는 스키마가 변하고 객체가 삭제되면 크롤러는 무엇을 할지 결정합니다. 크롤러 로그 스키마가 실행되면서 변합니다.

크롤러 구성 옵션을 보여주는 샘플 Python 코드

```
import boto3
import json

# Initialize a boto3 client for AWS Glue
glue_client = boto3.client('glue', region_name='us-east-1') # Replace 'us-east-1'
with your desired AWS region

# Define the crawler configuration
crawler_configuration = {
    "Version": 1.0,
    "CrawlerOutput": {
        "Partitions": {
            "AddOrUpdateBehavior": "InheritFromTable"
        },
        "Tables": {
            "AddOrUpdateBehavior": "MergeNewColumns"
        }
    }
}

configuration_json = json.dumps(crawler_configuration)
# Create the crawler with the specified configuration
response = glue_client.create_crawler(
    Name='your-crawler-name', # Replace with your desired crawler name
    Role='crawler-test-role', # Replace with the ARN of your IAM role for Glue
    DatabaseName='default', # Replace with your target Glue database name
    Targets={
```

```

    'S3Targets': [
      {
        'Path': "s3://your-bucket-name/path/", # Replace with your S3 path
to the data
      },
    ],
    # Include other target types like 'JdbcTargets' if needed
  },
  Configuration=configuration_json,
  # Include other parameters like Schedule, Classifiers, TablePrefix,
  SchemaChangePolicy, etc., as needed
)

print(response)a

```

크롤러가 실행되면 새로운 테이블과 파티션은 스키마 변화 정책과 무관하게 항상 생성됩니다. SchemaChangePolicy 구조의 UpdateBehavior 필드에서 다음 작업 중 하나를 선택하여 크롤러가 변경된 테이블 스키마를 찾았을 때 수행할 작업을 결정할 수 있습니다.

- UPDATE_IN_DATABASE - AWS Glue Data Catalog에서 테이블을 업데이트합니다. 새로운 열을 추가하고 누락 열을 제거하며 기존 열의 정의를 수정합니다. 크롤러가 설정하지 않은 메타데이터를 제거합니다.
- LOG - 변경 사항을 무시하고 Data Catalog에서 테이블을 업데이트하지 않습니다.

중분 크롤링에 대한 기본 설정입니다.

크롤러 API Configuration 필드에 제공된 JSON 객체를 사용하여 SchemaChangePolicy 구조를 재정의할 수도 있습니다. JSON 객체는 키값 페어를 포함하여 정책을 설정하고 기존 열을 업데이트하지 않으면서 새로운 열만 추가합니다. 예를 들어, 문자열로써 다음 JSON 객체를 제공합니다.

```

{
  "Version": 1.0,
  "CrawlerOutput": {
    "Tables": { "AddOrUpdateBehavior": "MergeNewColumns" }
  }
}

```

이 옵션은 AWS Glue 콘솔의 [새 열만 추가(Add new columns only)] 옵션에 해당합니다. Amazon S3 데이터 스토어만 크롤링하여 생성된 테이블의 SchemaChangePolicy 구조를 재정의합니다. Data Catalog(사실 원본)에 메타데이터가 존재하고 유지하고자 한다면 이 옵션을 선택합니다. 중복된 데이터 유형을 포함하여 새로운 열에 맞닥뜨리면 이 새로운 열을 추가합니다. 하지만 기존 열은 제거되지 않고 유형도 변하지 않습니다. Amazon S3 테이블 속성이 크게 변하면 테이블의 사용 중지를 표시하고 비호환성 문제를 해결하라는 경고문을 실행합니다. 이 옵션은 증분 크롤러에는 적용할 수 없습니다.

이전에 크롤된 데이터 스토어에 반해 크롤러가 실행되면 새롭거나 변한 파티션을 발견하게 됩니다. 기본적으로 새 파티션이 추가되고 기존 파티션이 변했다면 업데이트됩니다. 또한 크롤러 구성 옵션을 InheritFromTable(AWS Glue 콘솔의 [테이블의 메타데이터로 모든 신규 및 기존 파티션 업데이트(Update all new and existing partitions with metadata from the table)] 옵션에 해당)로 설정할 수 있습니다. 이 옵션이 설정되면 파티션은 분류, 입력 포맷, 출력 포맷, SerDe 정보, 스키마 등 상위 테이블에서 메타데이터 속성을 물려받습니다. 모든 상위 테이블 속성의 변경 사항은 파티션으로 전파됩니다.

이 구성 옵션이 기존 크롤러에 설정되면 기존 파티션은 크롤러가 그 다음번에 실행될 때 상위 테이블의 속성과 일치하기 위해서 업데이트됩니다. 이 행동은 크롤러 API Configuration 필드에 설정됩니다. 예를 들어, 문자열로써 다음 JSON 객체를 제공합니다.

```
{
  "Version": 1.0,
  "CrawlerOutput": {
    "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" }
  }
}
```

크롤러 API Configuration 필드는 다양한 구성 옵션을 설정할 수 있습니다. 예를 들어, 파티션과 테이블 모두의 크롤러 출력을 구성하기 위해서 다음 JSON 객체를 나타내는 문자열을 제공할 수 있습니다.

```
{
  "Version": 1.0,
  "CrawlerOutput": {
    "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },
    "Tables": {"AddOrUpdateBehavior": "MergeNewColumns" }
  }
}
```

삭제된 객체가 데이터 스토어에 존재한다면 크롤러가 무엇을 할지 결정하기 위해서 다음 행동 중 하나를 선택할 수 있습니다. 크롤러 API의 SchemaChangePolicy 구조에 있는 DeleteBehavior 필드는 크롤러가 삭제된 객체를 검색할 때의 동작을 설정합니다.

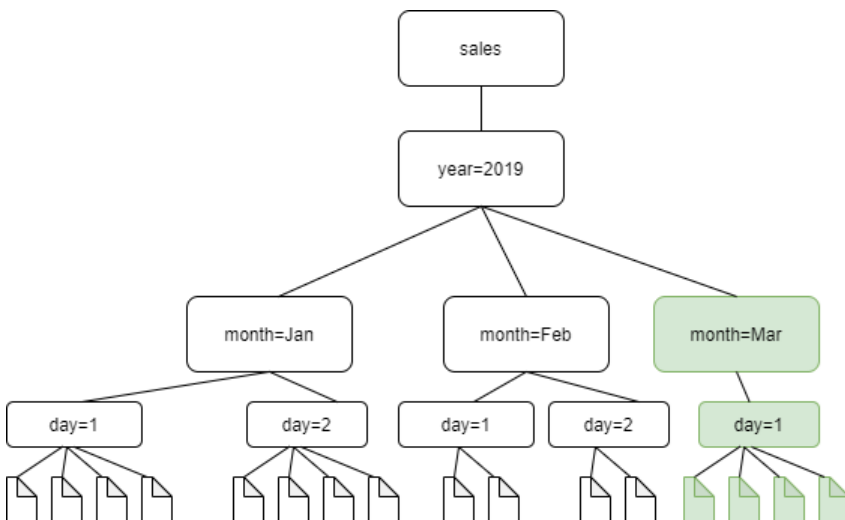
- DELETE_FROM_DATABASE – Data Catalog에서 테이블 및 파티션을 삭제합니다.
- LOG - 변경 사항을 무시합니다. Data Catalog를 업데이트하지 않습니다. 대신 로그 메시지를 작성합니다.
- DEPRECATE_IN_DATABASE – Data Catalog에서 해당 테이블을 사용 중지로 표시합니다. 이것이 기본 설정입니다.

새 파티션을 추가하기 위한 점진적인 크롤링

크롤러는 새 파티션을 추가하는 옵션을 제공하므로 안정적인 테이블 스키마가 있는 증분 데이터 세트의 크롤링이 더 빨라집니다. 각 크롤링 중에 새 파티션이 추가되는 예약된 크롤러에 많이 사용됩니다. 이 옵션을 켜면 먼저 크롤러가 초기 스키마 및 파티션 구조를 기록할 수 있도록 대상 데이터 세트에서 완전한 크롤링이 실행됩니다. 다시 크롤링하는 중에 스키마가 호환되는 경우 새 파티션이 기존 테이블에 추가됩니다. 첫 크롤링 실행 후 스키마가 변경되지 않으며 데이터 카탈로그에 새 테이블이 추가되지 않습니다.

Amazon S3 데이터 소스를 설정할 때 이 옵션을 사용할 수 있습니다. CreateCrawler API에서 RecrawlBehavior이(가) 포함된 RecrawlPolicy을(를) "Crawl_New_Folders"로 설정하거나 후속 크롤러가 콘솔에서 새 하위 폴더만 크롤링으로 실행할 수 있습니다.

[the section called “크롤러는 파티션 생성 시기를 어떻게 결정하나요?”](#)의 예를 계속하면 다음 다이어그램은 3월(March)의 파일이 추가되었음을 보여줍니다.



RecrawlBehavior을(를) "Crawl_New_Folders" 옵션으로 설정하면 새 폴더인 month=Mar만 크롤링됩니다.

규칙 및 제한

이 옵션이 설정되어 있으면 크롤러를 편집할 때 Amazon S3 대상 데이터 스토어를 변경할 수 없습니다. 이 옵션은 특정 크롤러 구성 설정에 영향을 줍니다. 설정하면 크롤러의 업데이트 동작 및 삭제 동작이 LOG에 기록됩니다. 이는 다음을 의미합니다.

- 스키마가 호환되지 않는 객체를 발견하면 크롤러는 데이터 카탈로그에 객체를 추가하지 않고 이 세부 정보를 CloudWatch Logs에 로그로 추가합니다.
- 데이터 카탈로그의 삭제된 개체는 업데이트되지 않습니다.

자세한 내용은 [the section called “크롤러 동작 사용자 지정” 단원을 참조하십시오.](#)

파티션 인덱스 크롤러 구성 옵션 설정

데이터 카탈로그는 특정 파티션을 효율적으로 조회할 수 있도록 파티션 인덱스를 지원합니다. 자세한 내용은 [AWS Glue에서 파티션 인덱스 작업](#)을 참조하세요. AWS Glue 크롤러는 기본적으로 Amazon S3 및 Delta Lake 대상에 대한 파티션 인덱스를 생성합니다.

크롤러를 정의하면 자동으로 파티션 인덱스 생성 옵션이 출력 및 예약 설정 페이지의 고급 옵션 아래에서 기본적으로 활성화됩니다.

이 옵션을 비활성화하려면 콘솔에서 파티션 인덱스 자동 생성 확인란의 선택을 취소하면 됩니다. 크롤러 API를 사용하여 Configuration에서 CreatePartitionIndex 를 설정하여 이 옵션을 비활성화할 수도 있습니다. 기본값은 true입니다.

파티션 인덱스에 대한 사용 참고 사항

- 크롤러에서 생성한 테이블에는 기본적으로 partition_filtering.enabled 변수가 없습니다. 자세한 내용은 [AWS Glue 파티션 인덱싱 및 필터링](#)을 참조하세요.
- 암호화된 파티션에 대한 파티션 인덱스 생성은 지원되지 않습니다.

Amazon S3 이벤트 알림을 사용하여 크롤링 가속화

Amazon S3 또는 데이터 카탈로그 대상의 객체를 나열하는 대신 Amazon S3 이벤트를 사용하여 변경 사항을 찾도록 크롤러를 구성할 수 있습니다. 이 기능은 전체 Amazon S3 또는 데이터 카탈로그 대상을 나열하는 대신 Amazon S3 이벤트를 통해 이벤트를 트리거한 하위 폴더의 모든 파일을 나열하여 두 크롤링 간의 변경 사항을 식별하므로 다시 크롤링하는 시간이 단축됩니다.

첫 번째 크롤링은 대상의 모든 Amazon S3 객체를 나열합니다. 첫 번째 크롤링이 성공한 후에는 수동으로 또는 정해진 일정에 따라 다시 크롤링하도록 선택할 수 있습니다. 크롤러는 모든 객체를 나열하지 않고 해당 이벤트의 객체만 나열합니다.

Amazon S3 이벤트 기반 크롤러로 이동할 경우의 이점은 다음과 같습니다.

- 대상의 모든 객체를 나열할 필요 없이 객체가 추가되거나 삭제되는 특정 폴더가 나열되므로 더 빠르게 다시 크롤링할 수 있습니다.
- 객체가 추가되거나 삭제되는 특정 폴더가 나열되므로 전체 크롤링 비용이 절감됩니다.

Amazon S3 이벤트 크롤링은 크롤러 일정에 따라 SQS 대기열의 Amazon S3 이벤트를 사용하여 실행됩니다. 대기열에 이벤트가 없으면 비용이 발생하지 않습니다. SQS 대기열이나 여러 소비자가 동일한 이벤트를 필요로 하는 경우 SNS와 SQS의 조합으로 바로 이동하도록 Amazon S3 이벤트를 구성할 수 있습니다. 자세한 내용은 [the section called "Amazon S3 이벤트 알림을 위해 계정 설정"](#) 단원을 참조하십시오.

이벤트 모드로 크롤러를 생성하고 구성한 후 첫 번째 크롤링은 Amazon S3 또는 데이터 카탈로그 대상을 모두 나열하여 목록 모드로 실행됩니다. "크롤링이 Amazon S3 이벤트를 사용하여 실행되고 있습니다."라는 로그는 첫 번째 크롤링이 성공적으로 끝난 후 크롤링이 Amazon S3 이벤트를 사용하여 작동하고 있음을 확인합니다.

Amazon S3 이벤트 크롤링을 생성하고 크롤링에 영향을 줄 수 있는 크롤러 속성을 업데이트하면 크롤링이 목록 모드로 작동하고 "크롤링이 S3 이벤트 모드로 실행되고 있지 않습니다."라는 로그가 추가됩니다.

Note

사용할 수 있는 최대 메시지 수는 크롤링당 10,000개입니다.

카탈로그 대상

대상이 데이터 카탈로그인 경우 크롤러는 변경 내용을 사용하여 데이터 카탈로그의 기존 테이블을 업데이트합니다(예: 테이블의 추가 파티션).

주제

- [Amazon S3 이벤트 알림을 위해 계정 설정](#)
- [Amazon S3 이벤트 크롤러에서 암호화 사용](#)

Amazon S3 이벤트 알림을 위해 계정 설정

이 섹션에서는 Amazon S3 이벤트 알림을 위해 계정을 설정하는 방법을 설명하고 스크립트 또는 AWS Glue 콘솔을 사용하여 계정을 설정하는 지침을 제공합니다.

필수 조건

다음 설정 태스크를 완료합니다. 괄호 안의 값은 스크립트에서 구성 가능한 설정을 가리킵니다.

1. Amazon S3 버킷(s3_bucket_name)을 생성합니다.
2. 식별된 버킷의 경로인 크롤러 대상('test1'과 같은 folder_name)을 식별합니다.
3. 크롤러 이름(crawler_name)을 준비합니다.
4. SNS 주제 이름(sns_topic_name)을 준비합니다. 크롤러 이름과 같을 수 있습니다.
5. 크롤러를 실행해야 하고 S3 버킷이 있는 AWS 리전(region)을 준비합니다.
6. 이메일을 사용하여 Amazon S3 이벤트를 가져오는 경우 선택적으로 이메일 주소 (subscribing_email)를 준비합니다.

CloudFormation 스택을 사용하여 리소스를 생성할 수도 있습니다. 다음 단계를 완료합니다.

1. 미국 동부(버지니아 북부)에서 CloudFormation 스택을 [시작](#)합니다.
2. 파라미터 아래에 Amazon S3 버킷의 이름(계정 번호 포함)을 입력합니다.
3. I acknowledge that AWS CloudFormation might create IAM resources with custom names을 선택합니다.
4. Create stack를 선택합니다.

제한:

- Amazon S3 대상이든 Data Catalog 대상이든 크롤러는 단일 대상만 지원합니다.
- 프라이빗 VPC의 SQS는 지원되지 않습니다.
- Amazon S3 샘플링은 지원되지 않습니다.
- 크롤러 대상은 Amazon S3 대상의 폴더이거나 데이터 카탈로그 대상에 대한 하나 이상의 AWS Glue 데이터 카탈로그 테이블이어야 합니다.
- 'everything' 경로 와일드카드 는 지원되지 않습니다(s3://%).
- 데이터 카탈로그 대상의 경우 모든 카탈로그 테이블은 Amazon S3 이벤트 모드에 대해 동일한 Amazon S3 버킷을 가리켜야 합니다.

- 데이터 카탈로그 대상의 경우 카탈로그 테이블은 Delta Lake 형식(_symlink 폴더 포함 또는 카탈로그 테이블의 InputFormat 확인)의 Amazon S3 위치를 가리키지 않아야 합니다.

Amazon S3 이벤트 기반 크롤러를 사용하려면 S3 대상 및 SQS의 스토어와 동일한 접두사에서 필터링된 이벤트를 사용하여 S3 버킷에서 이벤트 알림을 사용해야 합니다. 콘솔을 통해 [연습: 알림용 버킷 구성](#) 단계를 수행하거나 [the section called “SQS를 생성하고 대상의 Amazon S3 이벤트를 구성하는 스크립트”](#)를 사용하여 SQS와 이벤트 알림을 설정할 수 있습니다.

SQS 정책

크롤러가 사용하는 역할에 연결해야 하는 다음 SQS 정책을 추가합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "sqs:DeleteMessage",
        "sqs:GetQueueUrl",
        "sqs:ListDeadLetterSourceQueues",
        "sqs:ReceiveMessage",
        "sqs:GetQueueAttributes",
        "sqs:ListQueueTags",
        "sqs:SetQueueAttributes",
        "sqs:PurgeQueue"
      ],
      "Resource": "arn:aws:sqs:{region}:{accountID}:cfn-sqs-queue"
    }
  ]
}
```

SQS를 생성하고 대상의 Amazon S3 이벤트를 구성하는 스크립트

사전 조건이 충족되었는지 확인한 후 다음 Python 스크립트를 실행하여 SQS를 생성할 수 있습니다. 구성 가능한 설정을 사전 조건에서 준비된 이름으로 바꿉니다.

Note

스크립트를 실행한 후 SQS 콘솔에 로그인하여 생성된 SQS의 ARN을 찾습니다.

Amazon SQS에서 다른 소비자가 메시지를 수신하고 처리할 수 없도록 차단하는 기간인 가시성 시간 제한을 설정합니다. 가시성 시간 제한을 크롤링 실행 시간과 거의 동일하게 설정합니다.

```
#!/venv/bin/python
import boto3
import botocore

#-----Start : READ ME FIRST -----#
# 1. Purpose of this script is to create the SQS, SNS and enable S3 bucket
notification.
# The following are the operations performed by the scripts:
# a. Enable S3 bucket notification to trigger 's3:ObjectCreated:' and
's3:ObjectRemoved:' events.
# b. Create SNS topic for fan out.
# c. Create SQS queue for saving events which will be consumed by the crawler.
# SQS Event Queue ARN will be used to create the crawler after running the
script.
# 2. This script does not create the crawler.
# 3. SNS topic is created to support FAN out of S3 events. If S3 event is also used by
another
purpose, SNS topic created by the script can be used.
# 1. Creation of bucket is an optional step.
# To create a bucket set create_bucket variable to true.
# 2. The purpose of crawler_name is to easily locate the SQS/SNS.
# crawler_name is used to create SQS and SNS with the same name as crawler.
# 3. 'folder_name' is the target of crawl inside the specified bucket 's3_bucket_name'
#
#-----End : READ ME FIRST -----#

#-----#
# Start : Configurable settings #
#-----#

#Create
region = 'us-west-2'
s3_bucket_name = 's3eventtestuswest2'
folder_name = "test"
crawler_name = "test33S3Event"
sns_topic_name = crawler_name
sqs_queue_name = sns_topic_name
create_bucket = False
```

```
#-----#
# End : Configurable settings  #
#-----#

# Define aws clients
dev = boto3.session.Session(profile_name='myprofile')
boto3.setup_default_session(profile_name='myprofile')
s3 = boto3.resource('s3', region_name=region)
sns = boto3.client('sns', region_name=region)
sqs = boto3.client('sqs', region_name=region)
client = boto3.client("sts")
account_id = client.get_caller_identity()["Account"]
queue_arn = ""

def print_error(e):
    print(e.message + ' RequestId: ' + e.response['ResponseMetadata']['RequestId'])

def create_s3_bucket(bucket_name, client):
    bucket = client.Bucket(bucket_name)
    try:
        if not create_bucket:
            return True
        response = bucket.create(
            ACL='private',
            CreateBucketConfiguration={
                'LocationConstraint': region
            },
        )
        return True
    except botocore.exceptions.ClientError as e:
        print_error(e)
        if 'BucketAlreadyOwnedByYou' in e.message: # we own this bucket so continue
            print('We own the bucket already. Lets continue...')
            return True
    return False

def create_s3_bucket_folder(bucket_name, client, directory_name):
    s3.put_object(Bucket=bucket_name, Key=(directory_name + '/'))

def set_s3_notification_sns(bucket_name, client, topic_arn):
    bucket_notification = client.BucketNotification(bucket_name)
    try:
```

```
response = bucket_notification.put(
    NotificationConfiguration={
        'TopicConfigurations': [
            {
                'Id' : crawler_name,
                'TopicArn': topic_arn,
                'Events': [
                    's3:ObjectCreated:*',
                    's3:ObjectRemoved:*',
                ],
                'Filter' : {'Key': {'FilterRules': [{'Name': 'prefix',
'Value': folder_name}]}}
            },
        ]
    }
)
return True
except botocore.exceptions.ClientError as e:
    print_error(e)
return False

def create_sns_topic(topic_name, client):
    try:
        response = client.create_topic(
            Name=topic_name
        )
        return response['TopicArn']
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return None

def set_sns_topic_policy(topic_arn, client, bucket_name):
    try:
        response = client.set_topic_attributes(
            TopicArn=topic_arn,
            AttributeName='Policy',
            AttributeValue='''{
                "Version": "2008-10-17",
                "Id": "s3-publish-to-sns",
                "Statement": [{
                    "Effect": "Allow",
                    "Principal": { "AWS" : "*" },
```

```
        "Action": [ "SNS:Publish" ],
        "Resource": "%s",
        "Condition": {
            "StringEquals": {
                "AWS:SourceAccount": "%s"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws:s3:*:*:%s"
            }
        }
    }
}]]
)''' % (topic_arn, account_id, bucket_name)
)
return True
except botocore.exceptions.ClientError as e:
    print_error(e)

return False

def subscribe_to_sns_topic(topic_arn, client, protocol, endpoint):
    try:
        response = client.subscribe(
            TopicArn=topic_arn,
            Protocol=protocol,
            Endpoint=endpoint
        )
        return response['SubscriptionArn']
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return None

def create_sqs_queue(queue_name, client):
    try:
        response = client.create_queue(
            QueueName=queue_name,
        )
        return response['QueueUrl']
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return None
```



```
def get_sqs_queue_arn(queue_url, client):
    try:
        response = client.get_queue_attributes(
            QueueUrl=queue_url,
            AttributeNames=[
                'QueueArn',
            ]
        )
        return response['Attributes']['QueueArn']
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return None

def set_sqs_policy(queue_url, queue_arn, client, topic_arn):
    try:
        response = client.set_queue_attributes(
            QueueUrl=queue_url,
            Attributes={
                'Policy': '''{
                    "Version": "2012-10-17",
                    "Id": "AllowSNSPublish",
                    "Statement": [
                        {
                            "Sid": "AllowSNSPublish01",
                            "Effect": "Allow",
                            "Principal": "*",
                            "Action": "SQS:SendMessage",
                            "Resource": "%s",
                            "Condition": {
                                "ArnEquals": {
                                    "aws:SourceArn": "%s"
                                }
                            }
                        }
                    ]
                }''' % (queue_arn, topic_arn)
            }
        )
        return True
    except botocore.exceptions.ClientError as e:
        print_error(e)
    return False
```

```

if __name__ == "__main__":
    print('Creating S3 bucket %s.' % s3_bucket_name)
    if create_s3_bucket(s3_bucket_name, s3):
        print('\nCreating SNS topic %s.' % sns_topic_name)
        topic_arn = create_sns_topic(sns_topic_name, sns)
        if topic_arn:
            print('SNS topic created successfully: %s' % topic_arn)

            print('Creating SQS queue %s' % sqs_queue_name)
            queue_url = create_sqs_queue(sqs_queue_name, sqs)
            if queue_url is not None:
                print('Subscribing sqs queue with sns.')
                queue_arn = get_sqs_queue_arn(queue_url, sqs)
                if queue_arn is not None:
                    if set_sqs_policy(queue_url, queue_arn, sqs, topic_arn):
                        print('Successfully configured queue policy.')
                        subscription_arn = subscribe_to_sns_topic(topic_arn, sns,
'sqs', queue_arn)

                        if subscription_arn is not None:
                            if 'pending confirmation' in subscription_arn:
                                print('Please confirm SNS subscription by visiting the
subscribe URL.')

                            else:
                                print('Successfully subscribed SQS queue: ' +
queue_arn)

                            else:
                                print('Failed to subscribe SNS')
                        else:
                            print('Failed to set queue policy.')
                    else:
                        print("Failed to get queue arn for %s" % queue_url)
                # ----- End subscriptions to SNS topic -----

            print('\nSetting topic policy to allow s3 bucket %s to publish.' %
s3_bucket_name)
            if set_sns_topic_policy(topic_arn, sns, s3_bucket_name):
                print('SNS topic policy added successfully.')
                if set_s3_notification_sns(s3_bucket_name, s3, topic_arn):
                    print('Successfully configured event for S3 bucket %s' %
s3_bucket_name)

                    print('Create S3 Event Crawler using SQS ARN %s' % queue_arn)
                else:
                    print('Failed to configure S3 bucket notification.')
            else:

```

```

        print('Failed to add SNS topic policy.')
    else:
        print('Failed to create SNS topic.')

```

콘솔을 사용하여 Amazon S3 이벤트 알림용 크롤러 설정(Amazon S3 대상)

Amazon S3 대상에 대한 AWS Glue 콘솔을 사용하여 Amazon S3 이벤트 알림용 크롤러를 설정하려면

1. 크롤러 속성을 설정합니다. 자세한 내용을 알아보려면 [AWS Glue 콘솔에서 크롤러 구성 옵션 설정](#)을 참조하세요.
2. Data source configuration(데이터 소스 구성) 섹션에 Is your data already mapped to AWS Glue tables?라는 메시지가 표시됩니다.

기본적으로 Not yet(아직)이 이미 선택되어 있습니다. Amazon S3 데이터 소스를 사용 중이고 데이터가 AWS Glue 테이블에 아직 매핑되지 않았으므로 이 항목을 기본값으로 둡니다.

3. Data sources(데이터 소스) 섹션에서 Add a data source(데이터 소스 추가)를 선택합니다.

Step 1
Set crawler properties

Step 2
Choose data sources and classifiers

Step 3
Configure security settings

Step 4
Set output and scheduling

Step 5
Review and create

Choose data sources and classifiers

Data source configuration

Is your data already mapped to Glue tables?

Not yet
Select one or more data sources to be crawled.

Yes
Select existing tables from your Glue Data Catalog.

Data sources (0) Edit Remove Add a data source

The list of data sources to be scanned by the crawler.

Type	Data source	Parameters
You don't have any data sources.		

Add a data source

► **Custom classifiers - optional**

A classifier checks whether a given file is in a format the crawler can handle. If it is, the classifier creates a schema in the form of a StructType object that matches that data format.

Cancel Previous Next

4. Add data source(데이터 소스 추가) 모달에서 Amazon S3 데이터 소스를 구성합니다.
 - Data source(데이터 소스): 기본적으로 Amazon S3가 선택됩니다.
 - Network connection(네트워크 연결)(선택 사항): Add new connection(새 연결 추가)을 선택합니다.

- Location of Amazon S3 data(Amazon S3 데이터 위치): 기본적으로 In this account(이 계정에서)가 선택됩니다.
- Amazon S3 path(Amazon S3 경로): 폴더와 파일이 크롤링되는 Amazon S3 경로를 지정합니다.
- Subsequent crawler runs(후속 크롤러 실행): 크롤러에 대한 Amazon S3 이벤트 알림을 사용하려면 Crawl based on events(이벤트 기반 크롤링)를 선택합니다.
- Include SQS ARN(SQS ARN 포함): 유효한 SQS ARN을 포함하는 데이터 스토어 파라미터를 지정합니다. (예: arn:aws:sqs:region:account:sqs).
- Include dead-letter SQS ARN(배달 못한 편지 SQS ARN 포함)(선택 사항): 유효한 Amazon 배달 못한 편지 SQS ARN을 지정합니다. (예: arn:aws:sqs:region:account:deadLetterQueue).
- Add an Amazon S3 data source(Amazon S3 데이터 소스 추가)를 선택합니다.

The screenshot shows the 'Add data source' dialog in the AWS Glue console. The dialog is titled 'Add data source' and has a close button (X) in the top right corner. It is divided into several sections:

- Data source:** A dropdown menu is set to 'S3'.
- Network connection - optional:** A dropdown menu is set to 'None', with a refresh button (C) to its right.
- Clear selection:** A button to clear the current selection.
- Add new connection:** A button with an external link icon to add a new network connection.
- Location of S3 data:** Two radio buttons are present: 'In this account' (selected) and 'In a different account'.
- S3 path:** A text input field contains 's3://test', with 'View' and 'Browse' buttons to its right.
- Subsequent crawler runs:** A text input field is present. Below it, three radio buttons are shown: 'Crawl all sub-folders', 'Crawl new sub-folders only', and 'Crawl based on events' (selected).
- Include SQS ARN:** A text input field contains 'arn:aws:sqs:region:account:sqs'.
- Include dead-letter SQS ARN - optional:** A text input field contains 'arn:aws:sqs:region:account:deadLetterQueue'.
- Sample only a subset of files:** An unchecked checkbox.
- Exclude files matching pattern:** An unchecked checkbox.

At the bottom of the dialog, there are 'Cancel' and 'Add an S3 data source' buttons.

AWS CLI를 사용하여 Amazon S3 이벤트 알림용 크롤러 설정

다음은 Amazon S3 대상 버킷에서 SQS 대기열을 생성하고 이벤트 알림을 설정하기 위한 Amazon S3 AWS CLI 직접 호출 예제입니다.

```
S3 Event AWS CLI
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
create-queue.json
...
{
  "Policy": {
    "Version": "2012-10-17",
    "Id": "example-ID",
    "Statement": [
      {
        "Sid": "example-statement-ID",
        "Effect": "Allow",
        "Principal": {
          "Service": "s3.amazonaws.com"
        },
        "Action": [
          "SQS:SendMessage"
        ],
        "Resource": "SQS-queue-ARN",
        "Condition": {
          "ArnLike": {
            "aws:SourceArn": "arn:aws:s3:*:*:awsexamplebucket1"
          },
          "StringEquals": {
            "aws:SourceAccount": "bucket-owner-account-id"
          }
        }
      }
    ]
  }
}
...
aws s3api put-bucket-notification-configuration --bucket customer-data-pdx --
notification-configuration file://s3-event-config.json
s3-event-config.json
...
{
  "QueueConfigurations": [
```

```

    {
      "Id": "s3event-sqs-queue",
      "QueueArn": "arn:aws:sqs:{region}:{account}:queuename",
      "Events": [
        "s3:ObjectCreated:*",
        "s3:ObjectRemoved:*"
      ],
      "Filter": {
        "Key": {
          "FilterRules": [
            {
              "Name": "Prefix",
              "Value": "/json"
            }
          ]
        }
      }
    }
  ]
}
...

```

Create Crawler:

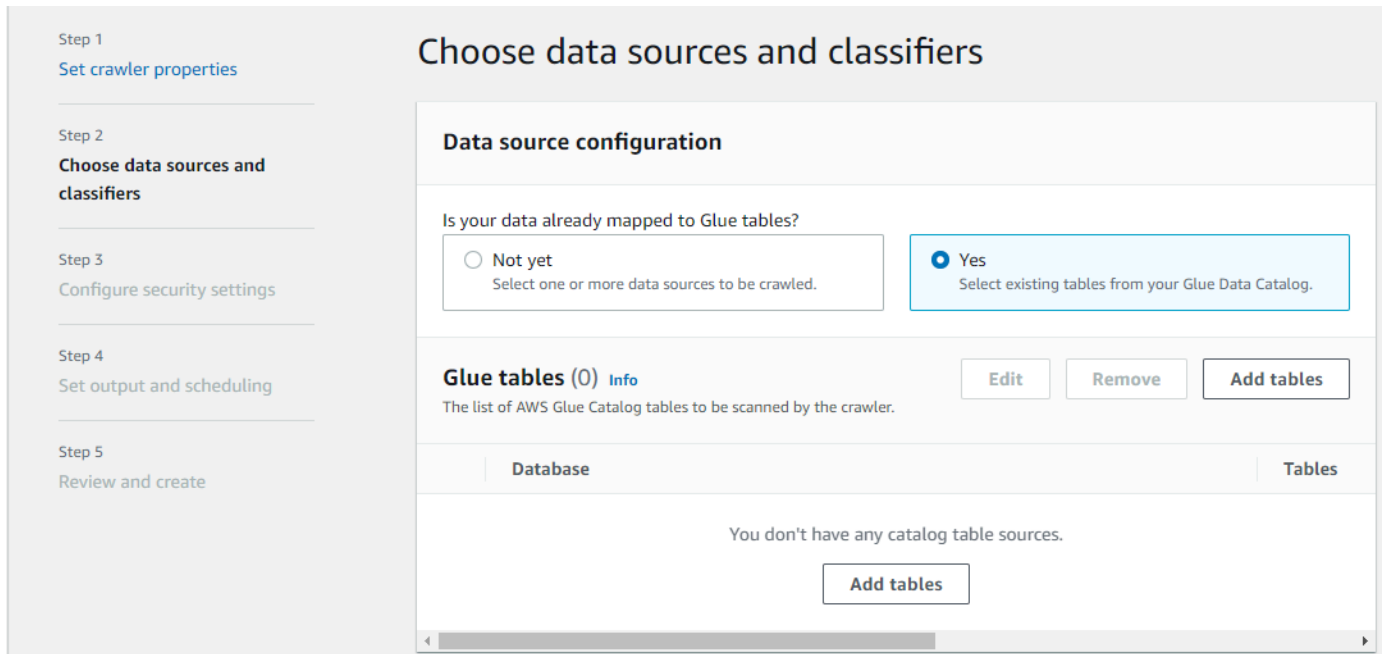
콘솔을 사용하여 Amazon S3 이벤트 알림용 크롤러 설정(데이터 카탈로그 대상)

카탈로그 대상이 있는 경우, AWS Glue 콘솔을 사용하여 Amazon S3 이벤트 알림용 크롤러를 설정합니다.

1. 크롤러 속성을 설정합니다. 자세한 내용을 알아보려면 [AWS Glue 콘솔에서 크롤러 구성 옵션 설정](#)을 참조하세요.
2. Data source configuration(데이터 소스 구성) 섹션에 Is your data already mapped to AWS Glue tables?라는 메시지가 표시됩니다.

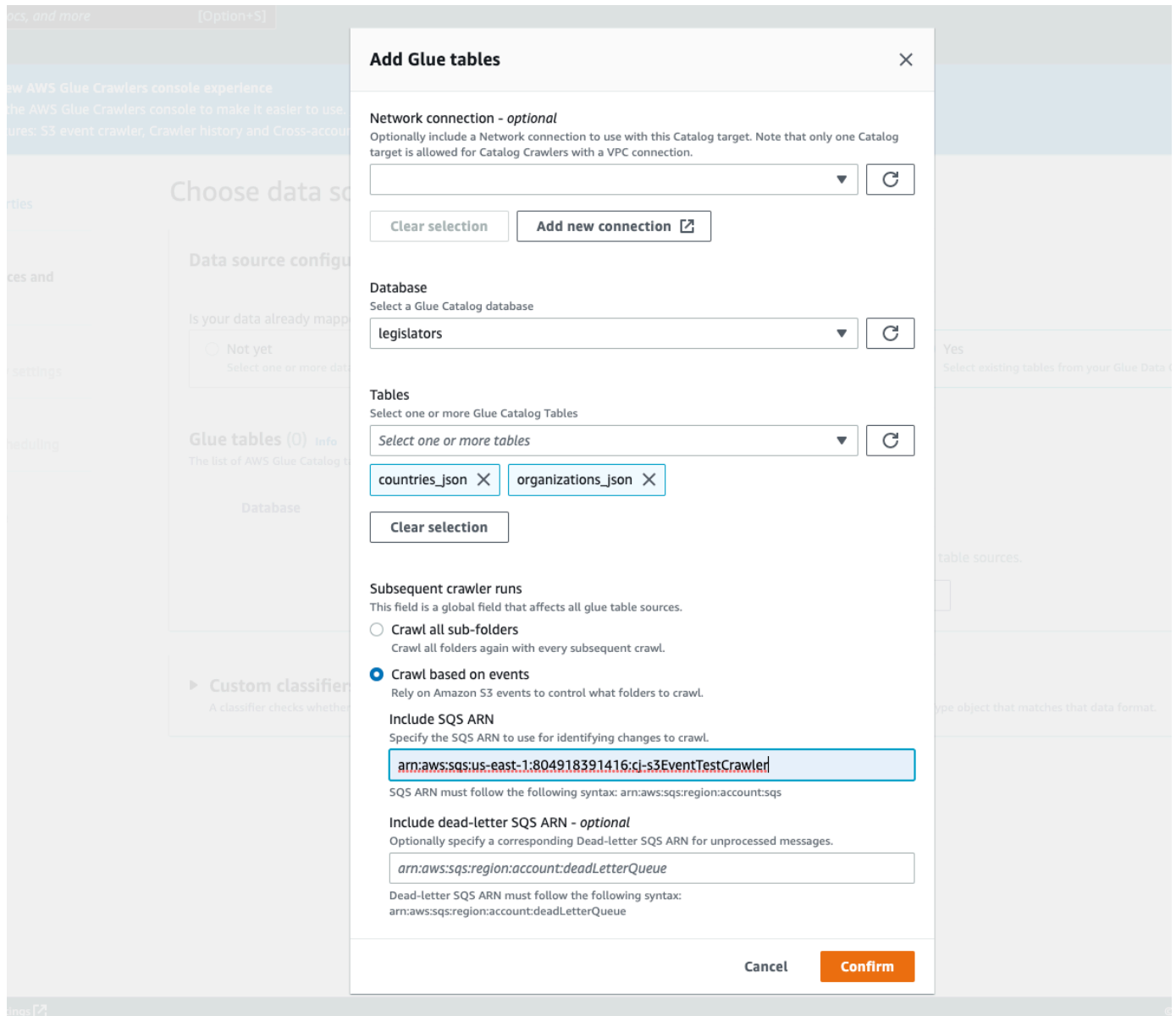
데이터 카탈로그의 기존 테이블을 데이터 소스로 선택하려면 예(Yes)를 선택합니다.

3. Glue 테이블(Glue tables) 섹션에서 테이블 추가(Add tables)를 선택합니다.



4. 테이블 추가(Add table) 모달에서 데이터베이스 및 테이블을 구성합니다.

- Network connection(네트워크 연결)(선택 사항): Add new connection(새 연결 추가)을 선택합니다.
- 데이터베이스(Database): 데이터 카탈로그의 데이터베이스를 선택합니다.
- 테이블(Tables): 데이터 카탈로그의 해당 데이터베이스에서 하나 이상의 테이블을 선택합니다.
- Subsequent crawler runs(후속 크롤러 실행): 크롤러에 대한 Amazon S3 이벤트 알림을 사용하려면 Crawl based on events(이벤트 기반 크롤링)를 선택합니다.
- Include SQS ARN(SQS ARN 포함): 유효한 SQS ARN을 포함하는 데이터 스토어 파라미터를 지정합니다. (예: `arn:aws:sqs:region:account:sqs`).
- Include dead-letter SQS ARN(배달 못한 편지 SQS ARN 포함)(선택 사항): 유효한 Amazon 배달 못한 편지 SQS ARN을 지정합니다. (예: `arn:aws:sqs:region:account:deadLetterQueue`).
- 확인(Confirm)을 선택합니다.



Amazon S3 이벤트 크롤러에서 암호화 사용

이 섹션에서는 SQS에서만 또는 SQS와 Amazon S3 둘 다에서 암호화를 사용하는 방법을 설명합니다.

주제

- [SQS에서만 암호화 사용 설정](#)
- [SQS와 Amazon S3 둘 다에서 암호화 사용 설정](#)
- [FAQ](#)

SQS에서만 암호화 사용 설정

Amazon SQS는 기본값으로 전송 중 암호화를 제공합니다. 선택 사항인 서버 측 암호화(SSE)를 대기열에 추가하려면 편집 패널에서 [고객 마스터 키\(CMK\)](#)를 첨부할 수 있습니다. 즉, SQS가 SQS 서버에 저장된 모든 고객 데이터를 암호화합니다.

고객 마스터 키(CMK) 생성

1. 키 관리 서비스(KMS)(Key Management Service (KMS)) > 고객 관리형 키(Customer Managed Keys) > 키 생성(Create key)을 선택합니다.
2. 단계에 따라 고유한 별칭과 설명을 추가합니다.
3. 이 키의 사용을 허용하려는 해당 IAM 역할을 추가합니다.
4. [사용자 지정 키 정책](#)이 Amazon SNS에 충분한 키 사용 권한을 부여하도록 키 정책의 '문(Statement)' 목록에 또 다른 문을 추가합니다.

```
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "*"
  }
]
```

대기열에서 서버 측 암호화(SSE) 사용

1. Amazon SQS > 대기열(Queues) > sqs_queue_name > 암호화(Encryption) 탭을 선택합니다.
2. 편집(Edit)을 선택하고 암호화(Encryption) 드롭다운까지 아래로 스크롤합니다.
3. 사용(Enabled)을 선택하여 SSE를 추가합니다.
4. 이름이 alias/aws/sqs인 기본 키가 아니라 앞서 생성한 CMK를 선택합니다.

▼ **Encryption - Optional**
 Amazon SQS provides in-transit encryption by default. To add at-rest encryption to your queue, enable server-side encryption. [Info](#)

Server-side encryption

Disabled

Enabled

Customer master key [Info](#)

alias/sqs-key ▼

이 키를 추가하면 암호화(Encryption) 탭이 추가한 키로 업데이트됩니다.

SNS subscriptions | Lambda triggers | Dead-letter queue | Monitoring | Tagging | Access policy | **Encryption**

Encryption Edit

Amazon SQS provides encryption in-transit by default. You can also add Server-Side Encryption (SSE) to your queue, which means that SQS encrypts all customer data at-rest on SQS servers. [Info](#)

CMK alias alias/sqs-key	Data key reuse period 5 Minutes
----------------------------	------------------------------------

Note

Amazon SQS는 최대 메시지 보존 기간 넘게 대기열에 유지된 메시지를 자동으로 삭제합니다. 기본 메시지 보존 기간은 4일입니다. 누락 이벤트를 방지하려면 SQS MessageRetentionPeriod를 최대값인 14일로 변경합니다.

SQS와 Amazon S3 둘 다에서 암호화 사용 설정

SQS에서 서버 측 암호화(SSE) 사용

1. [the section called “SQS에서만 암호화 사용 설정”](#) 단원의 단계를 따르세요.
2. CMK 설정의 마지막 단계에서 Amazon S3에 충분한 키 사용 권한을 부여합니다.

'문(Statement)' 목록에 다음을 붙여넣습니다.

```
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "s3.amazonaws.com"
```

```

    },
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "*"
  }
]

```

Amazon S3 버킷에서 서버 측 암호화(SSE) 사용 설정

1. [the section called “SQS에서만 암호화 사용 설정”](#) 단원의 단계를 따르세요.
2. 다음 중 하나를 수행하십시오.
 - 전체 S3 버킷에서 SSE를 사용하려면 대상 버킷의 속성(Properties) 탭으로 이동합니다.

여기서 SSE를 사용하도록 설정하고 사용할 암호화 유형을 선택할 수 있습니다. Amazon S3에서 자동으로 생성, 관리, 사용하는 암호화 키가 제공됩니다. 또는 KMS에서 키를 선택할 수도 있습니다.

Edit default encryption

Default encryption
Automatically encrypt new objects stored in this bucket. [Learn more](#)

Server-side encryption

Disable

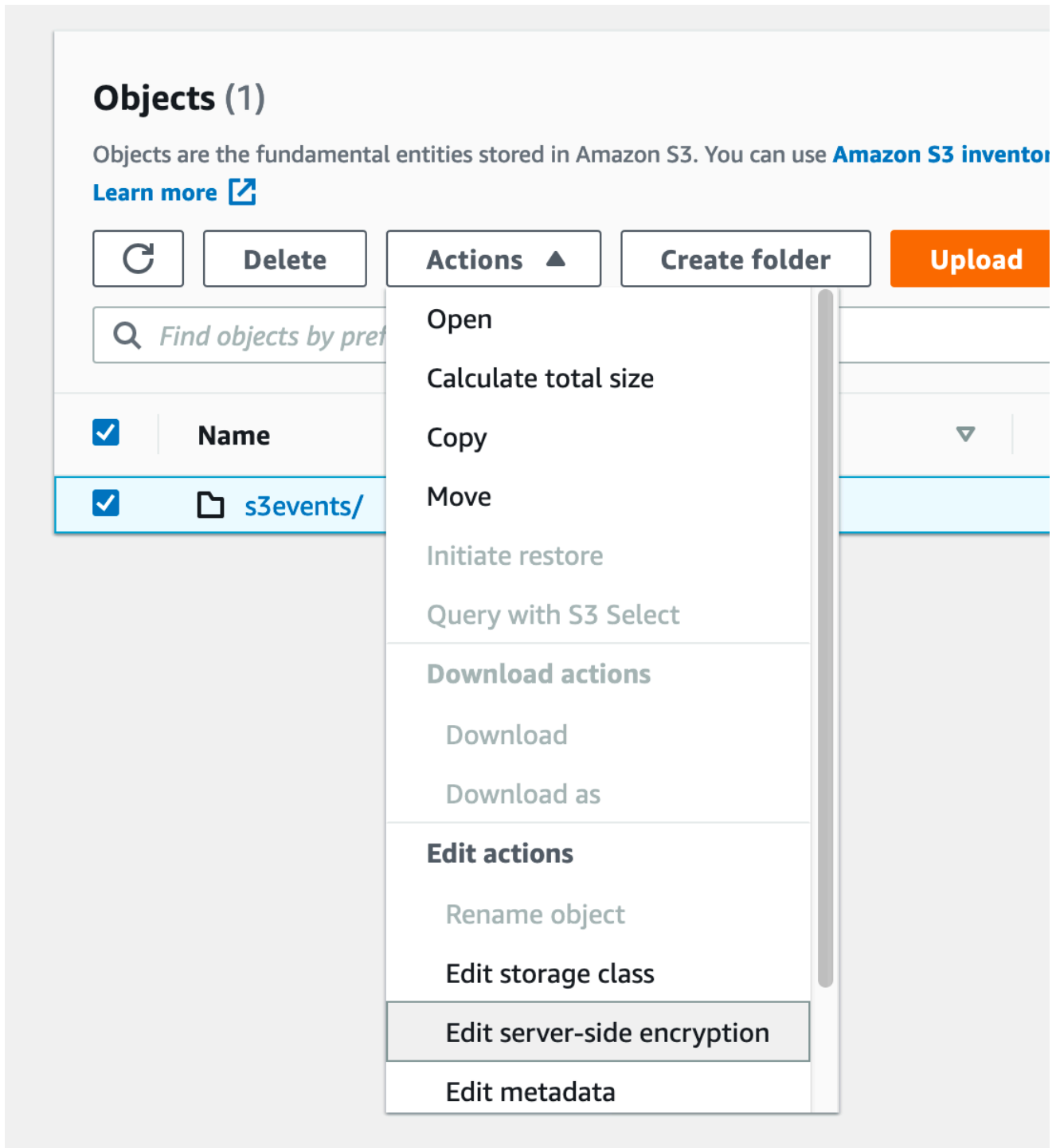
Enable

Encryption key type
To upload an object with a customer-provided encryption key (SSE-C), use the AWS CLI, AWS SDK, or Amazon S3 REST API.

Amazon S3 key (SSE-S3)
An encryption key that Amazon S3 creates, manages, and uses for you. [Learn more](#)

AWS Key Management Service key (SSE-KMS)
An encryption key protected by AWS Key Management Service (AWS KMS). [Learn more](#)

- 특정 폴더에서 SSE를 사용하려면 대상 폴더 옆의 확인란을 클릭하고 작업(Actions) 드롭다운 아래에 있는 서버 측 암호화 편집(Edit server-side encryption)을 선택합니다.



FAQ

Amazon SNS 주제에 게시한 메시지가 서버 측 암호화(SSE)를 사용하는 구독한 Amazon SQS 대기열로 전달되지 않는 이유는 무엇입니까?

Amazon SQS 대기열에서 다음을 사용하고 있는지 다시 확인합니다.

1. 고객 관리형인 [고객 마스터 키\(CMK\)](#). SQS에서 제공하는 기본 키가 아닙니다.
2. Amazon SNS에 충분한 키 사용 권한을 부여하는 [사용자 지정 키 정책](#)이 (1)의 CMK에 포함되어 있습니다.

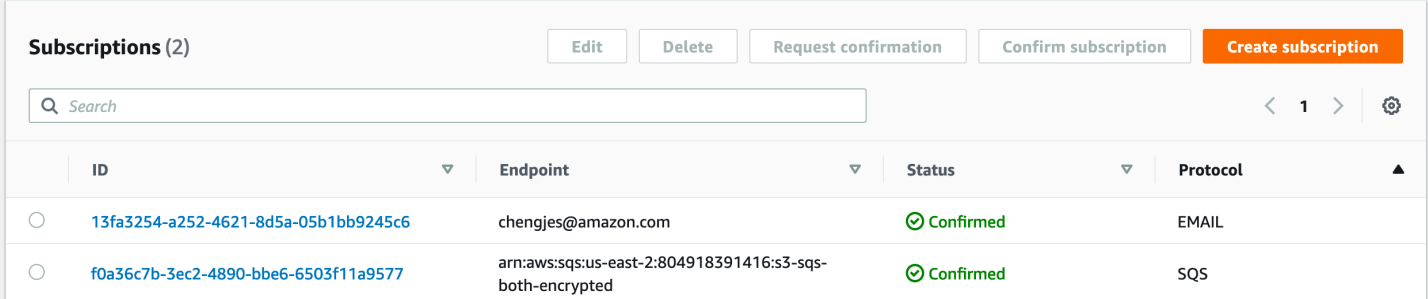
자세한 내용은 지식 센터에서 [이 문서를 참조](#)하세요.

이메일 알림을 구독했지만 Amazon S3 버킷을 편집할 때 이메일 업데이트가 수신되지 않습니다.

이메일에서 '구독 확인(Confirm Subscription)' 링크를 클릭하여 이메일 주소를 확인했는지 확인합니다. SNS 주제 아래에 있는 구독(Subscriptions) 테이블을 검사하여 확인 상태를 확인할 수 있습니다.

Amazon SNS > 주제(Topics) > sns_topic_name > 구독 테이블(Subscriptions table)을 선택합니다.

사전 조건 스크립트를 수행한 경우 sns_topic_name은 sqs_queue_name과 같습니다. 예를 들면 다음과 같아야 합니다.



ID	Endpoint	Status	Protocol
13fa3254-a252-4621-8d5a-05b1bb9245c6	chengjes@amazon.com	Confirmed	EMAIL
f0a36c7b-3ec2-4890-bbe6-6503f11a9577	arn:aws:sqs:us-east-2:804918391416:s3-sqs-both-encrypted	Confirmed	SQS

SQS 대기열에서 서버 측 암호화를 사용하도록 설정한 후 추가한 폴더 중 일부만 테이블에 표시됩니다. 일부 parquet가 누락되는 이유는 무엇입니까?

SQS 대기열에서 SSE를 사용하도록 설정하기 전에 Amazon S3 버킷을 변경한 경우 크롤러가 해당 변경 사항을 픽업하지 않을 수 있습니다. S3 버킷에 대한 모든 업데이트를 크롤링했는지 확인하려면 목록 모드로 크롤러를 다시 실행합니다('모든 폴더 크롤링(Crawl All Folders)'). 또 다른 옵션은 S3 이벤트가 사용되는 새 크롤러를 생성하여 새로 시작하는 것입니다.

크롤러가 기존 스키마를 변경하지 않도록 예방하는 방법

크롤러가 Amazon S3 테이블 정의의 기존 필드를 중복 업데이트하지 않도록 하려면 콘솔의 옵션을 선택하여 [새 열만 추가(Add new columns only)] 혹은 구성 옵션 MergeNewColumns를 설정합니다. Partitions.AddOrUpdateBehavior가 InheritFromTable에 중복되지 않는 한 이는 테이블과 파티션에 적용됩니다.

크롤러가 실행될 때 테이블 스키마가 전혀 변하지 않도록 하려면 LOG에 스키마 변경 정책을 설정합니다. 테이블에서 물려받은 파티션 스키마를 설정할 수 있는 구성 옵션도 설정할 수 있습니다.

콘솔에서 크롤러를 구성하는 중이면 다음 작업을 선택할 수 있습니다.

- 변경 사항을 무시하고 Data Catalog에서 테이블을 업데이트하지 않음
- [Update all new and existing partitions with metadata from the table(테이블에서 메타데이터가 있는 새 파티션과 기존 파티션을 모두 업데이트합니다)]

API를 사용하여 크롤러를 구성하면 다음 파라미터를 설정합니다.

- SchemaChangePolicy 구조의 UpdateBehavior 필드를 LOG로 설정합니다.
- 예를 들어, 크롤러 API의 다음 JSON 객체를 나타내는 문자열로 Configuration 필드를 설정합니다.

```
{
  "Version": 1.0,
  "CrawlerOutput": {
    "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" }
  }
}
```

각 Amazon S3 포함 경로에 대해 단일 스키마를 생성하는 방법

기본적으로 크롤러가 Amazon S3에 저장된 데이터용 테이블을 정의하면 데이터 호환성과 스키마 유사성을 모두 고려합니다. 고려되는 데이터 호환성 요인에는 데이터가 동일 포맷(예: JSON), 동일 압축 포맷(예: GZIP), Amazon S3 경로의 구조 및 기타 데이터 속성인지 여부가 포함됩니다. 스키마 유사성은 개별 Amazon S3 객체의 스키마가 얼마나 근접하게 유사한지를 측정합니다.

가능하면 공통 테이블 정의에 호환 스키마를 결합(CombineCompatibleSchemas)하도록 크롤러를 구성할 수 있습니다. 이 옵션을 사용하는 경우 크롤러가 데이터 호환성을 고려하지만, 지정된 포함 경로에서 Amazon S3 객체를 평가할 때 특정 스키마의 유사성을 무시합니다.

스키마를 결합하기 위해 콘솔에서 크롤러를 구성하려는 경우 크롤러 옵션 Create a single schema for each S3 path(각 S3 경로에 대해 단일 스키마 생성)를 선택합니다.

API를 사용하여 크롤러를 구성할 때는 다음 구성 옵션을 설정합니다.

- 예를 들어, 크롤러 API의 다음 JSON 객체를 나타내는 문자열로 Configuration 필드를 설정합니다.

```
{
```

```

"Version": 1.0,
"Grouping": {
  "TableGroupingPolicy": "CombineCompatibleSchemas" }
}

```

이 옵션의 이해를 돕기 위해 포함 경로가 `s3://bucket/table1/`인 크롤러를 정의한다고 가정해 보겠습니다. 이 크롤러는 실행 시 다음과 같은 특성을 가진 JSON 파일 두 개를 찾습니다.

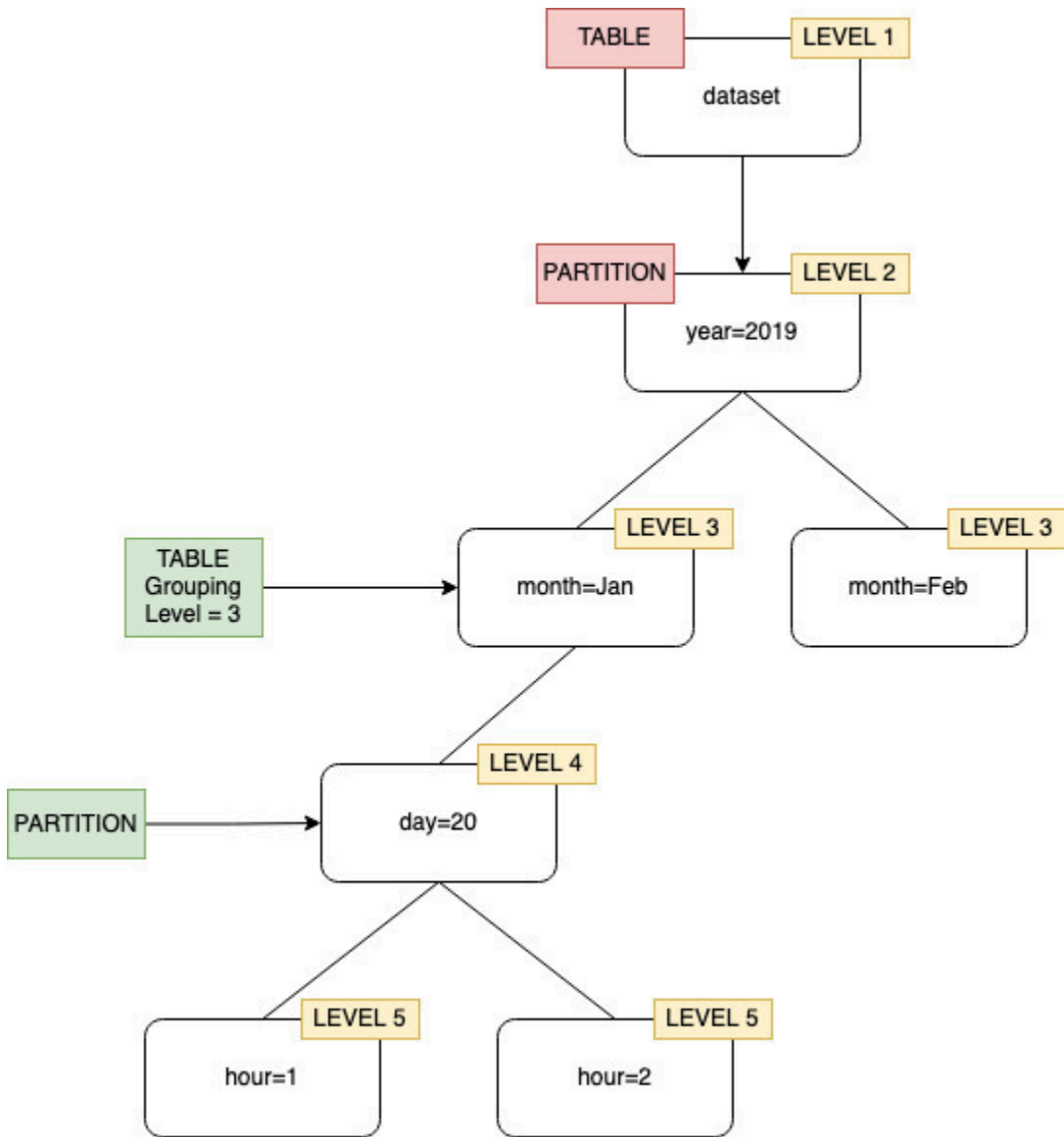
- 파일 1 – `S3://bucket/table1/year=2017/data1.json`
- 파일 내용 – `{“A”: 1, “B”: 2}`
- 스키마 – `A:int, B:int`
- 파일 2 – `S3://bucket/table1/year=2018/data2.json`
- 파일 내용 – `{“C”: 3, “D”: 4}`
- 스키마 – `C: int, D: int`

스키마가 충분히 비슷하지 않으므로 기본적으로 크롤러가 `year_2017` 및 `year_2018`이라는 두 테이블을 생성합니다. 하지만 `Create a single schema for each S3 path`(각 S3 경로에 대해 단일 스키마 생성) 옵션을 선택했으며 데이터가 호환되는 경우, 크롤러가 한 개 테이블을 생성합니다. 이 테이블에는 `A:int,B:int,C:int,D:int` 및 `partitionKey year:string`이라는 스키마가 있습니다.

테이블 위치와 파티션 수준을 지정하는 방법

기본적으로 크롤러가 Amazon S3에 저장된 데이터에 대한 테이블을 정의할 때 크롤러는 스키마를 병합하고 최상위 테이블(`year=2019`)을 생성하려고 시도합니다. 경우에 따라 크롤러가 폴더 `month=Jan`에 대한 테이블을 생성할 것으로 예상할 수 있지만 대신 형제 폴더(`month=Mar`)가 동일한 테이블에 병합되었기 때문에 크롤러가 파티션을 생성합니다.

테이블 수준 크롤러 옵션을 사용하면 크롤러에 테이블의 위치와 파티션 생성 방법을 유연하게 알릴 수 있습니다. [테이블 수준(Table level)]을 지정하면 Amazon S3 버킷에서 해당 절대 수준으로 테이블이 생성됩니다.



콘솔에서 크롤러를 구성할 때 [테이블 수준(Table level)] 크롤러 옵션에 대한 값을 지정할 수 있습니다. 값은 테이블 위치(데이터 집합의 절대 수준)를 나타내는 양의 정수여야 합니다. 최상위 폴더의 수준은 1입니다. 예를 들어 경로 mydataset/year/month/day/hour의 경우 수준을 3으로 설정하면 mydataset/year/month 위치에 테이블이 생성됩니다.

Console

Step 1
Set crawler properties

Step 2
Choose data sources and classifiers

Step 3
Configure security settings

**Step 4
Set output and scheduling**

Step 5
Review and create

Set output and scheduling

Output configuration

Target database

↕
↻

Clear selection
Add database ↗

Table name prefix - optional

Maximum table threshold - optional

This field sets the maximum number of tables the crawler is allowed to generate. In the event that this number is surpassed, the crawl will fail with an error. If not set, the crawler will automatically generate the number of tables depending on the data schema.

▼ Advanced options

S3 schema grouping

Create a single schema for each S3 path

By default, when a crawler defines tables for data stored in S3, it considers both data compatibility and schema similarity. Select this check box to group compatible schemas into a single table definition across all S3 objects under the provided include path. Other criteria will still be considered to determine proper grouping.

Table level - optional

The value must be a positive integer that indicates table location (the absolute level in the dataset). The level for the top level folder is 1. For example, for the path mydataset/a/b, if the level is set to 3, the table is created at location mydataset/a/b.

API

API를 사용하여 크롤러를 구성하는 경우 다음 JSON 객체의 문자열 표현으로 Configuration 필드를 설정합니다. 예를 들면 다음과 같습니다.

```
configuration = jsonencode(
{
  "Version": 1.0,
  "Grouping": {
    TableLevelConfiguration = 2
  }
})
```

CloudFormation

이 예제에서는 콘솔에서 사용 가능한 테이블 레벨(Table level) 옵션을 CloudFormation 템플릿 내에서 설정합니다.

```
"Configuration": "{
```

```
\"Version\":1.0,
\"Grouping\":{\"TableLevelConfiguration\":2}
}"
```

크롤러가 생성할 수 있는 최대 테이블 수를 지정하는 방법

선택적으로 AWS Glue 콘솔 또는 CLI를 통해 `TableThreshold`를 지정하여 크롤러가 생성할 수 있는 최대 테이블 수를 지정할 수 있습니다. 크롤링 중 크롤러가 탐지한 테이블이 이 입력 값보다 크면 크롤링이 실패하고 데이터 카탈로그에 데이터가 기록되지 않습니다.

이 파라미터는 크롤러에 의해 탐지되고 생성되는 테이블이 예상보다 훨씬 큰 경우에 유용합니다. 다음과 같은 여러 가지 이유가 있을 수 있습니다.

- AWS Glue 작업을 사용하여 Amazon S3 위치를 채우면 폴더와 같은 수준에 빈 파일이 생길 수 있습니다. 이러한 경우 이 Amazon S3 위치에서 크롤러를 실행하면 파일 및 폴더가 동일한 수준에 있기 때문에 크롤러가 여러 테이블을 생성합니다.
- `"TableGroupingPolicy": "CombineCompatibleSchemas"`를 설정하지 않으면 예상보다 많은 테이블이 생성될 수 있습니다.

`TableThreshold`를 0보다 큰 정수 값으로 지정합니다. 이 값은 크롤러별로 구성됩니다. 즉, 모든 크롤링에 대해 이 값이 고려됩니다. 예: 크롤러의 `TableThreshold` 값이 5로 설정되어 있습니다. 각 크롤링에서 AWS Glue는 탐지된 테이블 수를 이 테이블 임계값(5)과 비교하여 탐지된 테이블 수가 5보다 작으면 테이블을 데이터 카탈로그에 쓰고 AWS Glue 그렇지 않으면 데이터 카탈로그에 쓰지 않고 크롤링에 실패합니다.

콘솔

AWS 콘솔을 사용하여 `TableThreshold` 설정

Set output and scheduling

Output configuration Info

Target database
 ▼ | ↻

Table name prefix - optional

Maximum table threshold - optional
This field sets the maximum number of tables the crawler is allowed to generate. In the event that this number is surpassed, the crawl will fail with an error. If not set, the crawler will automatically generate the number of tables depending on the data schema.

▶ Advanced options

CLI

AWS CLI를 사용하여 TableThreshold 설정

```

{"Version":1.0,
"CrawlerOutput":
{"Tables":{"AddOrUpdateBehavior":"MergeNewColumns",
"TableThreshold":5}}};

```

테이블 경로를 식별하고 데이터를 정리하는 데 도움이 되도록 오류 메시지가 기록됩니다. 다음은 테이블 수가 제공된 테이블 임계값보다 커서 크롤러가 실패한 경우 계정의 예제 로그입니다.

```
Table Threshold value = 28, Tables detected - 29
```

CloudWatch에서는 탐지된 모든 테이블 위치를 INFO 메시지로 기록합니다. 오류가 실패 원인으로 기록됩니다.

```

ERROR com.amazonaws.services.glue.customerLogs.CustomerLogService - CustomerLogService
received CustomerFacingException with message
The number of tables detected by crawler: 29 is greater than the table threshold value
provided: 28. Failing crawler without writing to Data Catalog.
com.amazonaws.services.glue.exceptions.CustomerFacingInternalException: The number of
tables detected by crawler: 29 is greater than the table threshold value provided:
28.
Failing crawler without writing to Data Catalog.

```

Delta Lake 데이터 스토어에 대한 구성 옵션을 지정하는 방법

Delta Lake 데이터 스토어에 대해 크롤러를 구성할 때 이러한 구성 파라미터를 지정합니다.

연결

필요에 따라 이 Amazon S3 대상과 함께 사용할 네트워크 연결을 선택하거나 추가합니다. 연결에 관한 자세한 내용은 [데이터에 연결](#) 단원을 참조하십시오.

쿼리용 테이블 생성

Delta Lake 테이블 생성 방법을 선택합니다.

- 기본 테이블 생성: Delta 트랜잭션 로그의 쿼리를 직접 지원하는 쿼리 엔진과 통합할 수 있습니다.
- Symlink 테이블 생성: 지정된 구성 파라미터를 기반으로 파티션 키로 분할된 매니페스트 파일을 포함하는 symlink 매니페스트 폴더를 생성합니다.

쓰기 매니페스트 활성화(Delta Lake 소스에 대해 Symlink 테이블 생성을 선택한 경우에만 구성 가능)

Delta Lake 트랜잭션 로그에서 테이블 메타데이터 또는 스키마 변경 사항을 감지할지 여부를 선택합니다. 해당 항목은 매니페스트 파일을 다시 생성합니다. Delta Lake SET TBLPROPERTIES에서 자동 매니페스트 업데이트를 구성한 경우에는 이 옵션을 선택해서는 안 됩니다.

Delta Lake 테이블 경로 포함

`s3://bucket/prefix/object`와 같이 Delta 테이블에 대한 Amazon S3 경로를 하나 이상 지정합니다.

Add data source ✕

Data source
Choose the source of data to be crawled.

Delta Lake ▼

Connection - *optional*
Select a connection to access the data sources below.

▼

↻

Clear selection

Add new connection [↗](#)

Include delta lake table paths
Browse for or enter an existing S3 path.

s3://bucket/prefix/object

Remove

Add new delta table path

Enable write manifest
When enabled, if the crawler detects table metadata or schema changes in the Delta Lake transaction log, it regenerates the manifest file. You should not choose this option if you configured automatic manifest updates with Delta Lake SET TBLPROPERTIES.

Cancel

Add a Delta Lake data source

Lake Formation 자격 증명을 사용하도록 크롤러를 구성하는 방법

AWS Lake Formation 자격 증명을 사용하여 동일한 AWS 계정 또는 다른 AWS 계정 내의 기본 Amazon S3 위치가 있는 데이터 카탈로그 테이블 또는 Amazon S3 데이터 스토어에 액세스하도록 크롤러를 구성할 수 있습니다. 크롤러와 데이터 카탈로그 테이블이 동일한 계정에 있는 경우 기존 데이터

카탈로그 테이블을 크롤러의 대상으로 구성할 수 있습니다. 현재 데이터 카탈로그 테이블을 크롤러의 대상으로 사용할 때 단일 카탈로그 테이블이 있는 단일 카탈로그 대상만 허용됩니다.

Note

데이터 카탈로그 테이블을 크롤러 대상으로 정의할 때 데이터 카탈로그 테이블의 기본 위치가 Amazon S3 위치인지 확인합니다. Lake Formation 자격 증명을 사용하는 크롤러는 기본 Amazon S3 위치가 있는 데이터 카탈로그 대상만 지원합니다.

크롤러와 등록된 Amazon S3 위치 또는 데이터 카탈로그 테이블이 동일한 계정에 있는 경우 필요한 설정(계정 내 크롤링)


크롤러가 Lake Formation 자격 증명을 사용하여 데이터 스토어 또는 데이터 카탈로그 테이블에 액세스할 수 있도록 하려면 Lake Formation에 데이터 위치를 등록해야 합니다. 또한 크롤러의 IAM 역할에는 Amazon S3 버킷이 등록된 대상에서 데이터를 읽을 수 있는 권한이 있어야 합니다.

AWS Management Console 또는 AWS Command Line Interface(AWS CLI)를 사용하여 다음 구성 단계를 완료할 수 있습니다.

AWS Management Console

1. 크롤러 소스에 액세스하도록 크롤러를 구성하기 전에 데이터 스토어 또는 데이터 카탈로그의 데이터 위치를 Lake Formation에 등록합니다. Lake Formation 콘솔(<https://console.aws.amazon.com/lakeformation/>)에서 크롤러가 정의된 AWS 계정에서 Amazon S3 위치를 데이터 레이크의 루트 위치로 등록합니다. 자세한 내용을 알아보려면 [Registering an Amazon S3 location](#)(Amazon S3 위치 등록)을 참조하세요.
2. 크롤러가 Lake Formation의 대상에서 데이터를 읽을 수 있도록 크롤러 실행에 사용되는 IAM 역할에 Data location(데이터 위치) 권한을 부여합니다. 자세한 내용을 알아보려면 [Granting data location permissions \(same account\)](#)(데이터 위치 권한 부여(동일한 계정))를 참조하세요.
3. 출력 데이터베이스로 지정된 데이터베이스에 크롤러 역할 액세스 권한(Create)을 부여합니다. 자세한 내용을 알아보려면 [Granting database permissions using the Lake Formation console and the named resource method](#)(Lake Formation 콘솔 및 명명된 리소스 메서드를 사용하여 데이터베이스 권한 부여)를 참조하세요.
4. IAM 콘솔(<https://console.aws.amazon.com/iam/>)에서 크롤러에 대한 IAM 역할을 생성합니다. 역할에 `lakeformation:GetDataAccess` 정책을 추가합니다.

5. AWS Glue 콘솔(<https://console.aws.amazon.com/glue/>)에서 크롤러를 구성하는 동안 Use Lake Formation credentials for crawling Amazon S3 data source(Amazon S3 데이터 소스 크롤링에 Lake Formation 자격 증명 사용) 옵션을 선택합니다.

 Note

accountId 필드는 계정 내 크롤링의 선택 사항입니다.

AWS CLI

```
aws glue --profile demo create-crawler --debug --cli-input-json '{
  "Name": "prod-test-crawler",
  "Role": "arn:aws:iam::111122223333:role/service-role/AWSGlueServiceRole-prod-
test-run-role",
  "DatabaseName": "prod-run-db",
  "Description": "",
  "Targets": {
    "S3Targets":[
      {
        "Path": "s3://crawl-testbucket"
      }
    ]
  },
  "SchemaChangePolicy": {
    "UpdateBehavior": "LOG",
    "DeleteBehavior": "LOG"
  },
  "RecrawlPolicy": {
    "RecrawlBehavior": "CRAWL_EVERYTHING"
  },
  "LineageConfiguration": {
    "CrawlerLineageSettings": "DISABLE"
  },
  "LakeFormationConfiguration": {
    "UseLakeFormationCredentials": true,
    "AccountId": "111122223333"
  },
  "Configuration": {
    "Version": 1.0,
    "CrawlerOutput": {
      "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },

```

```

        "Tables": {"AddOrUpdateBehavior": "MergeNewColumns" }
    },
    "Grouping": { "TableGroupingPolicy": "CombineCompatibleSchemas" }
},
"CrawlerSecurityConfiguration": "",
"Tags": {
    "KeyName": ""
}
}'

```

크롤러와 등록된 Amazon S3 위치가 다른 계정에 있는 경우 필요한 설정(크로스 계정 크롤링)

크롤러가 Lake Formation 자격 증명을 사용하여 다른 계정의 데이터 스토어에 액세스할 수 있도록 하려면 먼저 Lake Formation에 Amazon S3 데이터 위치를 등록해야 합니다. 그리고 다음 단계에 따라 크롤러의 계정에 데이터 위치 권한을 부여합니다.

AWS Management Console 또는 AWS CLI를 사용하여 다음 단계를 완료할 수 있습니다.

AWS Management Console

1. Amazon S3 위치가 등록된 계정(계정 B)에서
 - a. Lake Formation에 Amazon S3 경로를 등록합니다. 자세한 내용을 알아보려면 [Registering an Amazon S3 location](#)(Amazon S3 위치 등록)을 참조하세요.
 - b. 크롤러가 실행될 계정(계정 A)에 Data location(데이터 위치) 권한을 부여합니다. 자세한 내용을 알아보려면 [Granting data location permissions](#)(데이터 위치 권한 부여)를 참조하세요.
 - c. 기본 위치를 대상 Amazon S3 위치로 사용하여 Lake Formation에 빈 데이터베이스를 생성합니다. 자세한 내용을 알아보려면 [Creating a database](#)(데이터베이스 생성)를 참조하세요.
 - d. 계정 A(크롤러가 실행될 계정)에 이전 단계에서 생성한 데이터베이스에 대한 액세스 권한을 부여합니다. 자세한 내용을 알아보려면 [Granting database permissions](#)(데이터베이스 권한 부여)를 참조하세요.
2. 크롤러가 생성되어 실행될 계정(계정 A)에서
 - a. AWS RAM 콘솔을 사용하여 외부 계정(계정 B)에서 공유된 데이터베이스를 수락합니다. 자세한 내용을 알아보려면 [AWS Resource Access Manager에서 리소스 공유 초대 수락](#)을 참조하세요.
 - b. 크롤러에 대한 IAM 역할을 생성합니다. 역할에 lakeformation:GetDataAccess 정책을 추가합니다.

- c. Lake Formation 콘솔(<https://console.aws.amazon.com/lakeformation/>)에서 크롤러가 Lake Formation의 대상에서 데이터를 읽을 수 있도록 크롤러 실행에 사용되는 IAM 역할에 대상 Amazon S3 위치에 대한 Data location(데이터 위치) 권한을 부여합니다. 자세한 내용을 알아보려면 [Granting data location permissions](#)(데이터 위치 권한 부여)를 참조하세요.
- d. 공유 데이터베이스에 리소스 링크를 생성합니다. 자세한 내용을 알아보려면 [Create a resource link](#)(리소스 링크 만들기)를 참조하세요.
- e. 공유 데이터베이스 및 (Describe) 리소스 링크에 대한 크롤러 역할 액세스 권한(Create)을 부여합니다. 리소스 링크는 크롤러의 출력에 지정됩니다.
- f. AWS Glue 콘솔(<https://console.aws.amazon.com/glue/>)에서 크롤러를 구성하는 동안 Use Lake Formation credentials for crawling Amazon S3 data source(Amazon S3 데이터 소스 크롤링에 Lake Formation 자격 증명 사용) 옵션을 선택합니다.

크로스 계정 크롤링의 경우 대상 Amazon S3 위치가 Lake Formation에 등록된 AWS 계정 ID를 지정합니다. 계정 내 크롤링의 경우 accountId 필드는 선택 사항입니다.

AWS CLI

```
aws glue --profile demo create-crawler --debug --cli-input-json '{
  "Name": "prod-test-crawler",
```

```
"Role": "arn:aws:iam::111122223333:role/service-role/AWSGlueServiceRole-prod-
test-run-role",
  "DatabaseName": "prod-run-db",
  "Description": "",
  "Targets": {
    "S3Targets":[
      {
        "Path": "s3://crawl-testbucket"
      }
    ]
  },
  "SchemaChangePolicy": {
    "UpdateBehavior": "LOG",
    "DeleteBehavior": "LOG"
  },
  "RecrawlPolicy": {
    "RecrawlBehavior": "CRAWL_EVERYTHING"
  },
  "LineageConfiguration": {
    "CrawlerLineageSettings": "DISABLE"
  },
  "LakeFormationConfiguration": {
    "UseLakeFormationCredentials": true,
    "AccountId": "111111111111"
  },
  "Configuration": {
    "Version": 1.0,
    "CrawlerOutput": {
      "Partitions": { "AddOrUpdateBehavior": "InheritFromTable" },
      "Tables": {"AddOrUpdateBehavior": "MergeNewColumns" }
    },
    "Grouping": { "TableGroupingPolicy": "CombineCompatibleSchemas" }
  },
  "CrawlerSecurityConfiguration": "",
  "Tags": {
    "KeyName": ""
  }
}'
```

Note

- Lake Formation 자격 증명을 사용하는 크롤러는 Amazon S3 및 데이터 카탈로그 대상에 대해서만 지원됩니다.
- Lake Formation 자격 증명 벤딩을 사용하는 대상의 경우 기본 Amazon S3 위치가 동일한 버킷에 속해야 합니다. 예를 들어, 고객은 모든 대상 위치가 동일한 버킷(bucket1) 아래에 있는 한 여러 대상(s3://bucket1/folder1, s3://bucket1/folder2)을 사용할 수 있습니다. 서로 다른 버킷(s3://bucket1/folder1, s3://bucket2/folder2)을 지정할 수 없습니다.
- 현재 데이터 카탈로그 대상 크롤러의 경우 단일 카탈로그 테이블이 있는 단일 카탈로그 대상만 허용됩니다.

튜토리얼: AWS Glue 크롤러 추가

이 AWS Glue 시나리오에서는 주요 항공사의 도착 데이터를 분석하여 월별 출발 공항의 인기도를 계산해야 합니다. Amazon S3에 저장된 CSV 포맷의 2016년 항공편 데이터가 있습니다. 데이터를 변환하고 분석하기 전에 해당 메타데이터를 AWS Glue Data Catalog에 분류합니다.

이 튜토리얼에서는 Amazon S3 비행 로그에서 메타데이터를 유추하고 Data Catalog에 테이블을 생성하는 크롤러를 추가해 보겠습니다.

주제

- [필수 조건](#)
- [1단계: 크롤러 추가](#)
- [2단계: 크롤러 실행](#)
- [3단계: AWS Glue Data Catalog 객체 보기](#)

필수 조건

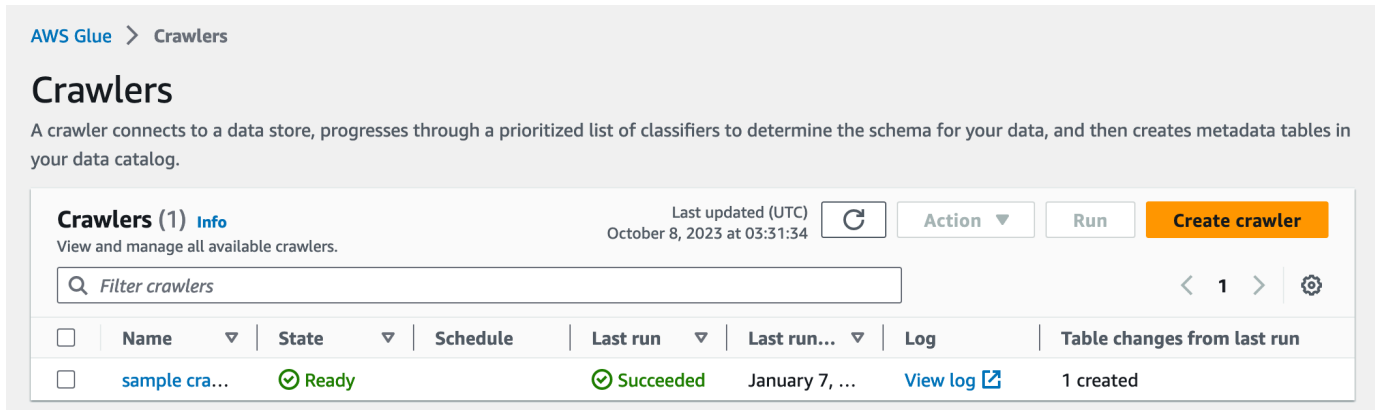
이 튜토리얼에서는 AWS 계정이 있고 AWS Glue에 대한 액세스 권한이 있다고 가정합니다.

1단계: 크롤러 추가

다음 단계에 따라 Amazon S3 저장된 CSV 파일에서 메타데이터를 추출하는 크롤러를 구성하고 실행합니다.

Amazon S3에 저장된 파일을 읽는 크롤러를 생성하려면

1. AWS Glue 서비스 콘솔의 왼쪽 메뉴에서 [크롤러(Crawlers)]를 선택합니다.
2. 크롤러 페이지에서 크롤러 생성을 선택합니다. 그러면 크롤러 세부 정보를 묻는 일련의 페이지가 시작됩니다.



3. [크롤러 이름(Crawler name)] 필드에 **Flights Data Crawler**를 입력하고 [다음(Next)]을 선택합니다.

크롤러는 분류자를 호출하여 데이터의 스키마를 추론합니다. 이 튜토리얼에서는 기본적으로 CSV에 기본 제공 분류자를 사용합니다.

4. 크롤러 소스 유형으로 [데이터 스토어(Data stores)]를 선택하고 [다음(Next)]을 선택합니다.
5. 이제 크롤러가 데이터를 가리키도록 하겠습니까. [데이터 스토어 추가(Add a data store)] 페이지에서 Amazon S3 데이터 스토어를 선택합니다. 이 튜토리얼에서는 연결을 사용하지 않으므로 [연결(Connection)] 필드가 표시되면 비워 둡니다.

[데이터 크롤링(Crawl data in)] 옵션으로 [다른 계정으로 지정된 경로(Specified path in another account)]를 선택합니다. 그런 다음 [포함 경로(Include path)]에 크롤러가 항공편 데이터를 찾을 수 있는 경로인 **s3://crawler-public-us-east-1/flight/2016/csv**를 입력합니다. 경로를 입력하면 이 필드의 제목이 [포함 경로(Include path)]로 바뀝니다. 다음을 선택합니다.

6. 단일 크롤러로 여러 데이터 스토어를 크롤링할 수 있습니다. 그러나 이 튜토리얼에서는 단일 데이터 스토어만 사용하므로 [아니오(No)]를 선택한 후 [다음(Next)]을 선택합니다.
7. 크롤러는 데이터 스토어에 액세스하고 AWS Glue Data Catalog에서 객체를 생성할 수 있는 권한이 필요합니다. 이러한 권한을 구성하려면 [IAM 역할 생성(Create an IAM role)]을 선택합니다. IAM 역할 이름은 **AWSGlueServiceRole-**로 시작하고 필드에 역할 이름의 마지막 부분을 입력합니다. **CrawlerTutorial**을 입력한 다음 [다음(Next)]을 선택합니다.

Note

IAM 역할을 생성하려면 AWS 사용자에게 `CreateRole`, `CreatePolicy` 및 `AttachRolePolicy` 권한이 있어야 합니다.

마법사는 `AWSGlueServiceRole-CrawlerTutorial1`이라는 IAM 역할을 생성하고 AWS 관리형 정책 `AWSGlueServiceRole`을 이 역할에 연결하고 Amazon S3 위치 `s3://crawler-public-us-east-1/flight/2016/csv`에 대한 읽기 액세스를 허용하는 인라인 정책을 추가합니다.

8. 크롤러에 대한 일정을 생성합니다. [빈도(Frequency)]에서 [온디맨드로 실행(Run on demand)]을 선택하고 [다음(Next)]을 선택합니다.
9. 크롤러가 Data Catalog에 테이블을 생성합니다. 테이블은 Data Catalog의 데이터베이스에 포함됩니다. 먼저 [데이터베이스 추가(Add database)]를 선택하여 데이터베이스를 생성합니다. 팝업 창에서 데이터베이스 이름으로 **test-flights-db**를 입력하고 [생성(Create)]을 선택합니다.

그런 다음 [테이블에 추가된 접두사(Prefix added to tables)]에 **flights**를 입력합니다. 나머지 옵션에 기본값을 사용하고 [다음(Next)]을 선택합니다.

10. [크롤러 추가(Add crawler)] 마법사에서 선택 내용을 확인합니다. 실수가 있는 경우 [뒤로(Back)]를 클릭하여 이전 페이지로 돌아가서 변경할 수 있습니다.

정보를 검토한 후 [마침(Finish)]을 선택하여 크롤러를 생성합니다.

2단계: 크롤러 실행

크롤러를 생성하면 마법사가 크롤러 보기 페이지로 이동합니다. 온디맨드 일정으로 크롤러를 생성하기 때문에 크롤러를 실행할 수 있는 옵션이 제공됩니다.

크롤러를 실행하려면

1. 이 페이지 상단 근처에 있는 배너를 통해 크롤러가 생성되었음을 알리고 지금 실행할 것인지 묻습니다. [지금 실행?(Run it now?)]을 선택하여 크롤러를 실행합니다.

배너가 변경되어 크롤러에 대해 “실행 시도 중(Attempting to run)” 및 “실행 중(Running)” 메시지가 표시됩니다. 크롤러 실행이 시작되면 배너가 사라지고 크롤러 디스플레이가 업데이트되어 크롤러에 대해 시작 중(Starting) 상태가 표시됩니다. 잠시 후 새로 고침 아이콘을 클릭하여 테이블에 표시된 크롤러의 상태를 업데이트할 수 있습니다.

2. 크롤러가 완료되면 크롤러의 변경 사항을 설명하는 새 배너가 나타납니다. `test-flights-db` 링크를 선택하여 Data Catalog 객체를 볼 수 있습니다.

3단계: AWS Glue Data Catalog 객체 보기

크롤러는 소스 위치에서 데이터를 읽고 Data Catalog에 테이블을 생성합니다. 테이블은 해당 스키마를 포함한 데이터를 나타내는 메타데이터 정의입니다. Data Catalog의 테이블에는 데이터가 없습니다. 대신 이러한 테이블을 작업 정의에서 소스 또는 대상으로 사용합니다.

크롤러가 생성한 Data Catalog 객체를 보려면

1. 왼쪽 탐색의 [데이터 카탈로그(Data catalog)]에서 [데이터베이스(Databases)]를 선택합니다. 여기에서 크롤러에 의해 생성된 `flights-db` 데이터베이스를 볼 수 있습니다.
2. 왼쪽 탐색의 [데이터 카탈로그(Data catalog)] 및 [데이터베이스(Databases)] 아래에서 [테이블(Tables)]을 선택합니다. 여기에서 크롤러가 생성한 `flightscsv` 테이블을 볼 수 있습니다. 테이블 이름을 선택하면 테이블 설정, 파라미터 및 속성을 볼 수 있습니다. 이 보기에서 아래로 스크롤하면 테이블의 열 및 데이터 유형에 대한 정보인 스키마를 볼 수 있습니다.
3. 테이블 보기 페이지에서 [파티션 보기(View partitions)]를 선택하면 데이터에 대해 생성된 파티션을 볼 수 있습니다. 첫 번째 열은 파티션 키입니다.

수동으로 메타데이터 정의

AWS Glue 데이터 카탈로그는 데이터 소스 및 데이터 세트에 대한 메타데이터를 저장하는 중앙 리포지토리입니다. 크롤러가 지원되는 데이터 소스의 메타데이터를 자동으로 크롤링하고 채울 수 있지만 특정 시나리오에서는 데이터 카탈로그에서 메타데이터를 수동으로 정의해야 할 수 있습니다.

- 지원되지 않는 데이터 형식 - 크롤러가 지원하지 않는 데이터 소스가 있는 경우 데이터 카탈로그에서 해당 데이터 소스의 메타데이터를 수동으로 정의해야 합니다.
- 사용자 지정 메타데이터 요구 사항 - AWS Glue 크롤러에서는 사전 정의된 규칙 및 규약을 기반으로 메타데이터를 유추합니다. AWS Glue 크롤러 추론 메타데이터에 포함되지 않는 특정 메타데이터 요구 사항이 있는 경우 필요에 맞게 메타데이터를 수동으로 정의할 수 있습니다.
- 데이터 거버넌스 및 표준화 - 데이터 거버넌스, 규정 준수 또는 보안상의 이유로 메타데이터 정의에 대한 보다 세밀한 제어가 필요할 수 있습니다. 메타데이터를 수동으로 정의하면 메타데이터가 조직의 표준 및 정책을 준수하는지 확인할 수 있습니다.

- 장래 데이터 수집을 위한 자리 표시자 - 즉시 사용할 수 없거나 액세스할 수 없는 데이터 소스가 있는 경우 빈 스키마 테이블을 자리 표시자로 생성할 수 있습니다. 데이터 소스를 사용할 수 있게 되면 사전 정의된 구조를 유지하면서 실제 데이터로 테이블을 채울 수 있습니다.

메타데이터를 수동으로 정의하려면 AWS Glue 콘솔, Lake Formation 콘솔, AWS Glue API 또는 AWS Command Line Interface(AWS CLI)를 사용할 수 있습니다. 데이터베이스, 테이블 및 파티션을 생성하고 열 이름, 데이터 유형, 설명 및 기타 속성과 같은 메타데이터 속성을 지정할 수 있습니다.

데이터베이스 생성

데이터베이스는 AWS Glue에서 메타데이터 테이블을 구성하는 데 사용됩니다. AWS Glue Data Catalog에 테이블을 정의할 때 테이블을 데이터베이스에 추가합니다. 테이블은 하나의 데이터베이스에만 있을 수 있습니다.

데이터베이스는 다양한 데이터 스토어의 데이터를 정의하는 테이블을 포함할 수 있습니다. 이 데이터에는 Amazon Simple Storage Service(Amazon S3)의 객체와 Amazon Relational Database Service의 관계형 테이블이 포함될 수 있습니다.

Note

AWS Glue Data Catalog에서 데이터베이스를 삭제할 때 데이터베이스의 모든 테이블도 삭제됩니다.

데이터베이스 목록을 보려면 AWS Management Console에 로그인하고 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다. [데이터베이스(Databases)]를 선택한 다음 목록에서 데이터베이스 이름을 선택하여 상세 정보를 봅니다.

AWS Glue 콘솔의 [데이터베이스(Databases)] 탭에서 데이터베이스를 추가, 편집 및 삭제할 수 있습니다.

- 새로운 데이터베이스를 생성하려면 [Add database(데이터베이스 추가)]를 선택하고 이름과 설명을 제공해야 합니다. Apache Hive와 같은 다른 메타데이터 스토어의 호환성을 고려해 이름은 소문자로 합니다.

Note

Amazon Athena에서 데이터베이스로 액세스하고자 할 경우, 공급자 이름을 영숫자 문자와 밑줄 표시로 생성합니다. 자세한 내용을 알아보려면 [테이블, 데이터베이스 및 열의 이름을 참조하세요](#).

- 데이터베이스에 대한 설명을 편집하려면 데이터베이스 이름 옆에 있는 확인란을 선택하고 Edit(편집)을 선택합니다.
- 데이터베이스를 삭제하려면 데이터베이스 이름 옆에 있는 확인란을 선택하고 Remove(제거)를 선택합니다.
- 데이터베이스에 포함된 테이블 목록을 표시하려면 데이터베이스 이름을 선택합니다. 그러면 데이터베이스 속성에 데이터베이스의 모든 테이블이 표시됩니다.

크롤러가 작성하는 데이터베이스를 변경하려면 크롤러 정의를 변경해야 합니다. 자세한 내용은 [크롤러를 사용하여 데이터 카탈로그 채우기](#) 단원을 참조하십시오.

데이터베이스 리소스 링크

AWS Glue 콘솔이 최근에 업데이트되었습니다. 현재 버전의 콘솔에서는 데이터베이스 리소스 링크를 지원하지 않습니다.

Data Catalog에는 데이터베이스에 대한 리소스 링크도 포함될 수 있습니다. 데이터베이스 리소스 링크는 로컬 또는 공유 데이터베이스에 대한 링크입니다. 현재 AWS Lake Formation에서만 리소스 링크를 생성할 수 있습니다. 데이터베이스에 대한 리소스 링크를 생성한 후에는 데이터베이스 이름을 사용할 모든 위치에 리소스 링크 이름을 사용할 수 있습니다. 사용자가 소유하거나 사용자와 공유된 데이터베이스와 함께 데이터베이스 리소스 링크는 `glue:GetDatabases()`에 의해 반환되고 AWS Glue 콘솔의 [데이터베이스(Databases)] 페이지에 항목으로 나타납니다.

Data Catalog에는 테이블 리소스 링크도 포함될 수 있습니다.

리소스 링크에 대한 자세한 내용은 AWS Lake Formation Developer Guide의 [Creating Resource Links](#)를 참조하세요.

테이블 생성

크롤러를 실행하여 데이터의 목록을 데이터 스토어로 가져오는 것이 권장되는 방법이지만 메타데이터 테이블을 수동으로 AWS Glue Data Catalog에 추가할 수도 있습니다. 이 접근 방식을 사용하면 메타데

이더 정의를 더 세밀하게 제어하고 특정 요구 사항에 따라 메타데이터 정의를 사용자 지정할 수 있습니다.

또한 다음과 같은 방법으로 데이터 카탈로그에 수동으로 테이블을 추가할 수 있습니다.

- AWS Glue 콘솔을 사용하여 AWS Glue Data Catalog에 테이블을 수동으로 생성합니다. 자세한 내용은 [AWS Glue 콘솔에서 테이블 관련 작업](#) 단원을 참조하십시오.
- [AWS Glue API](#)에서 CreateTable 작업을 사용하여 AWS Glue Data Catalog에 테이블을 생성합니다. 자세한 내용은 [CreateTable 액션 \(파이썬: create_table\)](#) 단원을 참조하십시오.
- AWS CloudFormation 템플릿을 사용합니다. 자세한 내용은 [AWS Glue용 AWS CloudFormation](#) 단원을 참조하십시오.

콘솔 또는 API를 사용하여 테이블을 수동으로 정의할 때 데이터 원본에 있는 데이터의 유형과 형식을 나타내는 분류 필드의 값과 테이블 스키마를 지정합니다. 크롤러가 테이블을 생성하면 데이터 형식 및 스키마는 기본 제공 분류자 또는 사용자 지정 분류자에 의해 결정됩니다. AWS Glue 콘솔을 사용하여 테이블 생성에 대한 자세한 내용은 [AWS Glue 콘솔에서 테이블 관련 작업](#) 단원을 참조하십시오.

주제

- [테이블 파티션](#)
- [테이블 리소스 링크](#)
- [크롤러를 사용하여 수동으로 생성된 Data Catalog 테이블 업데이트](#)
- [데이터 카탈로그 테이블 속성](#)
- [AWS Glue 콘솔에서 테이블 관련 작업](#)
- [AWS Glue에서 파티션 인덱스 작업](#)

테이블 파티션

Amazon Simple Storage Service(Amazon S3) 폴더의 AWS Glue 테이블 정의는 분할된 테이블을 설명할 수 있습니다. 예를 들어, 쿼리 성능을 향상시키려면 파티션된 테이블이 매월 데이터를 키로써 매월 이름을 사용하여 다른 파일로 나뉘어야 합니다. AWS Glue에서 테이블의 파티션된 키가 테이블 정의에 포함됩니다. AWS Glue가 Amazon S3 폴더의 데이터를 평가하여 테이블을 분류하면 개별 테이블인지 또는 추가된 파티션된 테이블인지 결정합니다.

테이블의 모든 파티션을 로드하는 대신 테이블에 파티션 인덱스를 만들어 파티션의 하위 집합을 가져올 수 있습니다. 파티션 인덱스 작업에 대한 자세한 내용은 [AWS Glue에서 파티션 인덱스 작업](#) 섹션을 참조하세요.

AWS Glue가 Amazon S3 폴더의 파티션된 테이블을 생성하는 것으로 간주하려면 모든 조건이 true여야 합니다.

- AWS Glue가 결정함에 따라 파일의 스키마는 비슷합니다.
- 파일의 데이터 형식은 동일합니다.
- 파일의 압축 형식은 동일합니다.

예를 들어, iOS 및 안드로이드 앱 세일 데이터 모두를 저장할 수 있는 my-app-bucket이라는 Amazon S3 버킷을 소유할 수 있습니다. 데이터는 연, 월, 일별로 분할되어 있습니다. iOS 및 안드로이드 세일 데이터 파일은 동일한 스키마, 데이터 포맷 및 압축 포맷을 가지고 있습니다. AWS Glue Data Catalog에서 AWS Glue 크롤러는 연, 월, 일별 파티션 키와 함께 하나의 테이블 정의를 생성합니다.

my-app-bucket의 다음 Amazon S3 목록은 몇 가지 파티션을 보여줍니다. = 부호는 파티션 키 값을 지정할 때 사용됩니다.

```
my-app-bucket/Sales/year=2010/month=feb/day=1/iOS.csv
my-app-bucket/Sales/year=2010/month=feb/day=1/Android.csv
my-app-bucket/Sales/year=2010/month=feb/day=2/iOS.csv
my-app-bucket/Sales/year=2010/month=feb/day=2/Android.csv
...
my-app-bucket/Sales/year=2017/month=feb/day=4/iOS.csv
my-app-bucket/Sales/year=2017/month=feb/day=4/Android.csv
```

테이블 리소스 링크

AWS Glue 콘솔이 최근에 업데이트되었습니다. 현재 버전의 콘솔에서는 테이블 리소스 링크를 지원하지 않습니다.

Data Catalog에는 테이블에 대한 리소스 링크도 포함될 수 있습니다. 테이블 리소스 링크는 로컬 또는 공유 테이블에 대한 링크입니다. 현재 AWS Lake Formation에서만 리소스 링크를 생성할 수 있습니다. 테이블에 대한 리소스 링크를 생성한 후에는 테이블 이름을 사용할 모든 위치에 리소스 링크 이름을 사용할 수 있습니다. 사용자가 소유하거나 사용자와 공유된 테이블과 함께 테이블 리소스 링크는 `glue:GetTables()`에 의해 반환되고 AWS Glue 콘솔의 [데이터베이스(Databases)] 페이지에 항목으로 나타납니다.

Data Catalog에는 데이터베이스 리소스 링크도 포함될 수 있습니다.

리소스 링크에 대한 자세한 내용은 AWS Lake Formation Developer Guide의 [Creating Resource Links](#)를 참조하세요.

크롤러를 사용하여 수동으로 생성된 Data Catalog 테이블 업데이트

AWS Glue Data Catalog 테이블을 수동으로 생성한 다음 AWS Glue 크롤러를 사용하여 업데이트된 상태로 유지할 수 있습니다. 일정에 따라 실행되는 크롤러는 새 파티션을 추가하고 스키마 변경 내용으로 테이블을 업데이트할 수 있습니다. 이는 Apache Hive 메타스토어에서 마이그레이션된 테이블에도 적용됩니다.

이렇게 하려면 크롤러를 정의할 때 하나 이상의 데이터 스토어를 크롤의 원본으로 지정하는 대신 하나 이상의 기존 Data Catalog 테이블을 지정합니다. 그런 다음 크롤러는 카탈로그 테이블에 지정된 데이터 스토어를 크롤합니다. 이 경우 새 테이블이 생성되지 않습니다. 대신 수동으로 생성된 테이블이 업데이트됩니다.

카탈로그 테이블을 수동으로 생성하고 카탈로그 테이블을 크롤러 원본으로 지정할 수 있는 다른 이유는 다음과 같습니다.

- 카탈로그 테이블 이름을 선택하고 카탈로그 테이블 이름 지정 알고리즘을 사용하지 않습니다.
- 파티션 감지를 방해할 수 있는 형식의 파일이 데이터 원본 경로에 잘못 저장되는 경우 새 테이블이 생성되지 않도록 합니다.

자세한 내용은 [2단계: 데이터 소스 및 분류자 선택](#) 단원을 참조하십시오.

데이터 카탈로그 테이블 속성

AWS CLI에 알려진 테이블 속성 또는 매개 변수는 검증되지 않은 키 및 값 문자열입니다. 테이블에서 사용자 고유의 속성을 설정하여 AWS Glue 외부에서 데이터 카탈로그를 사용할 수 있도록 지원할 수 있습니다. 데이터 카탈로그를 사용하는 다른 서비스도 마찬가지로 사용할 수 있습니다. AWS Glue는 작업 또는 크롤러를 실행할 때 일부 테이블 속성을 설정합니다. 달리 설명하지 않는 한 이러한 속성은 내부용이므로 현재 형태로 계속 존재하도록 지원하지 않으며, 이러한 속성을 수동으로 변경하는 경우 제품 작동을 지원하지 않습니다.

AWS Glue 크롤러가 설정하는 테이블 속성에 대한 자세한 내용은 [the section called “크롤러가 데이터 카탈로그 테이블에 설정한 파라미터”](#)을 참조하세요.

AWS Glue 콘솔에서 테이블 관련 작업

AWS Glue Data Catalog에서 테이블은 데이터 스토어의 데이터를 표현하는 메타데이터 정의입니다. 크롤러를 실행하여 테이블을 생성하거나 AWS Glue 콘솔을 사용하여 수동으로 테이블을 생성할

수 있습니다. AWS Glue 콘솔의 [테이블(Tables)] 목록에는 테이블의 메타데이터 값이 표시됩니다. ETL(Extract, Transform, and Load) 작업을 생성할 시 테이블 정의를 사용하여 소스 및 타겟을 명시합니다.

Note

AWS Management Console이 최근에 변경되었으므로 [SearchTables](#) 권한을 가지려면 기존 IAM 역할을 수정해야 할 수 있습니다. 새 역할 생성을 위해 SearchTables API 권한이 이미 기본값으로 추가되었습니다.

시작하려면 AWS Management Console에 로그인하고 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다. [Tables] 탭을 선택하고 [Add tables] 버튼을 사용하여 크롤러 혹은 수동 입력 속성으로 테이블을 생성합니다.

콘솔에서 테이블 추가

크롤러를 사용하여 테이블을 추가하려면 [Add tables]에서 [Add tables using a crawler]를 선택합니다. 다음, [Add crawler] 마법사 지시에 따릅니다. 크롤러가 실행되면 테이블은 AWS Glue Data Catalog에 추가됩니다. 자세한 내용은 [크롤러를 사용하여 데이터 카탈로그 채우기](#) 단원을 참조하십시오.

요구되는 속성을 알면 테이블 마법사를 통해 Data Catalog에 Amazon Simple Storage Service(Amazon S3) 테이블 정의를 생성할 수 있습니다. [Add tables]에서 [Add table manually]를 선택하고 [Add table] 마법사 지시를 따릅니다.

수동으로 콘솔을 통해 테이블을 추가할 때는 다음을 참조하세요.

- Amazon Athena에서 테이블에 액세스하고자 할 경우, 공급자 이름을 영숫자 문자와 밑줄 표시로 생성합니다. 자세한 내용은 [Athena 이름](#)을 참조하세요.
- 원본 데이터의 위치는 Amazon S3 경로여야 합니다.
- 데이터의 데이터 형식은 마법사의 형식과 일치해야 합니다. 해당 분류와 SerDe, 기타 테이블 속성은 자동으로 선택한 형식을 기반으로 장착됩니다. 다음 형식을 사용하여 테이블을 정의할 수 있습니다.

Avro

아파치 아브로 JSON 이진 형식

CSV

값 분리 가치. 콤마 혹은 파이프, 세미콜론, 탭, Ctrl+A의 구획 문자를 명시합니다.

JSON

JavaScript Object Notation(JSON)입니다.

XML

확장형 마크업 언어 형식 데이터에서 열을 정의하는 XML 태그를 명시합니다. 컬럼은 열 태그들 내에서 정의합니다.

PARQUET

Apache Parquet 컬럼 방식 스토리지

ORC

Optimized Row Columnar(ORC) 파일 형식입니다. Hive 데이터를 효율적으로 저장하도록 설계된 형식입니다.

- 테이블의 파티션 키를 정의할 수 있습니다.
- 현재, 콘솔을 사용하여 생성한 파티션된 테이블은 ETL 작업 시 사용할 수 없습니다.

테이블 속성

다음은 테이블이 갖는 몇 가지 중요한 특징입니다.

명칭

테이블 생성 시 이름이 결정되고 변경할 수 없습니다. 테이블 이름은 많은 AWS Glue 작업을 참조 하십시오.

데이터베이스

컨테이너 객체는 테이블이 위치한 곳입니다. 이 객체는 AWS Glue Data Catalog 내 존재하는 테이블 조직을 포함하고 데이터 스토어의 조직과 다를 수 있습니다. 데이터베이스를 삭제하면 데이터베이스에 포함된 모든 테이블도 Data Catalog에서 삭제됩니다.

설명

테이블에 대한 설명입니다. 테이블 내용을 이해할 수 있도록 설명을 적을 수 있습니다.

테이블 형식

표준 AWS Glue 테이블 또는 Apache Iceberg 형식의 테이블 생성을 지정합니다.

압축 활성화

테이블에 있는 작은 Amazon S3 객체를 더 큰 객체로 압축하려면 압축 활성화를 선택합니다.

IAM 역할

압축을 실행하기 위해 서비스는 사용자를 대신하여 IAM 역할을 맡습니다. 드롭다운을 사용하여 IAM 역할을 선택할 수 있습니다. 압축 기능을 활성화하는 데 필요한 권한이 역할에 있는지 확인합니다.

IAM 역할에 필요한 권한에 대해 알아보려면 [테이블 최적화 필수 조건](#) 섹션을 참조하십시오.

위치

이 테이블 정의는 데이터 스토어의 데이터 위치를 표현하는 포인터를 표시합니다.

분류

테이블 생성 시 분류 값이 제공됩니다. 일반적으로 크롤러가 소스 데이터 형식을 실행하고 명시하면 이것이 적합니다.

최종 업데이트 날짜

Data Catalog에 이 테이블이 업데이트된 시간 및 날짜(UTC).

데이터 추가됨

Data Catalog에 이 테이블이 추가된 시간 및 날짜(UTC).

Deprecated

기존 데이터 스토어에 Data Catalog의 테이블이 존재하지 않고 AWS Glue가 이를 인식하면 데이터 카탈로그에 이 테이블을 사용 중단 상태로 표시합니다. 사용 중단된 테이블을 참조하는 작업을 실행하면 그 작업은 실행되지 않을 수 있습니다. 사용 중단된 테이블을 참조하는 작업을 편집하여 소스와 타겟으로써 제거합니다. 더 이상 필요하지 않은 사용 중단된 테이블은 삭제하는 것이 좋습니다.

연결

AWS Glue가 데이터 스토어에 연결하고자 할 경우, 연결 이름은 테이블과 관련이 있습니다.

테이블 세부 정보 보기 및 편집

목록에서 테이블 이름을 선택하고 [Action, View details]를 선택하여 존재하는 테이블의 세부 정보를 열람합니다.

테이블 세부 정보는 테이블 속성과 스키마를 포함합니다. 이 화면은 테이블을 정의하기 위한 열 이름, 데이터 유형, 파티션 키 열 등을 포함한 테이블 스키마를 표시합니다. 복잡한 유형의 열은 [속성 보기 (View properties)]를 선택하여 다음 예처럼 필드 구조의 세부 사항을 표시합니다.

```
{
  "StorageDescriptor":
    {
      "cols": {
        "FieldSchema": [
          {
            "name": "primary-1",
            "type": "CHAR",
            "comment": ""
          },
          {
            "name": "second ",
            "type": "STRING",
            "comment": ""
          }
        ]
      },
      "location": "s3://aws-logs-111122223333-us-east-1",
      "inputFormat": "",
      "outputFormat": "org.apache.hadoop.hive.q1.io.HiveIgnoreKeyTextOutputFormat",
      "compressed": "false",
      "numBuckets": "0",
      "SerDeInfo": {
        "name": "",
        "serializationLib": "org.apache.hadoop.hive.serde2.OpenCSVSerde",
        "parameters": {
          "separatorChar": "|"
        }
      },
      "bucketCols": [],
      "sortCols": [],
      "parameters": {},
      "SkewedInfo": {},
      "storedAsSubDirectories": "false"
    },
  "parameters": {
    "classification": "csv"
  }
}
```

이러한 속성에 대한 StorageDescriptor와 같은 자세한 내용은 [StorageDescriptor 구조](#)를 참조하십시오.

[Edit schema]를 선택하여 열을 추가 및 제거하고 열 이름과 데이터 유형을 바꿔 테이블 스키마를 바꿀 수 있습니다.

[Compare versions]를 선택하고 테이블 스키마의 두 가지 다른 버전을 나란히 비교하여 스키마를 포함한 테이블의 다른 유형들을 비교할 수 있습니다. 자세한 내용은 [테이블 스키마 버전 비교](#) 단원을 참조하십시오.

Amazon S3 파티션을 구성하는 파일을 표시하려면 [파티션 보기(View partition)]를 선택합니다. Amazon S3 테이블의 [키(Key)] 열은 원본 데이터 스토어 테이블의 파티션에 사용된 파티션 키를 표시합니다. 파티셔닝은 날짜, 위치, 출발점과 같이 키 열 값을 기준으로 테이블을 관련 부분으로 나눕니다. 파티션에 대한 보다 자세한 내용은 "hive partitioning"으로 웹서치를 하시기 바랍니다.

Note

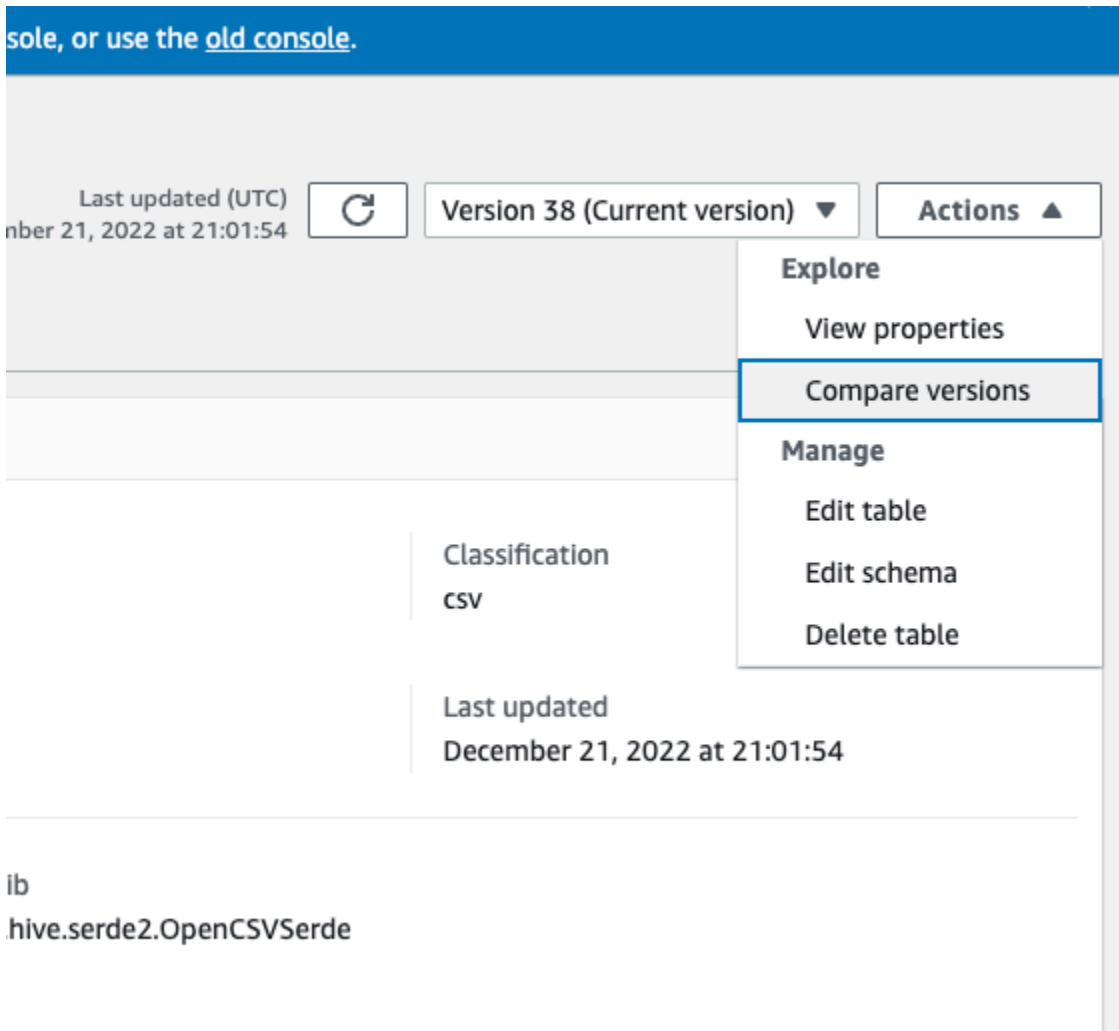
콘솔의 [Explore table] 자습서를 통해 테이블 세부 정보를 보기 위한 단계들을 알아보십시오.

테이블 스키마 버전 비교

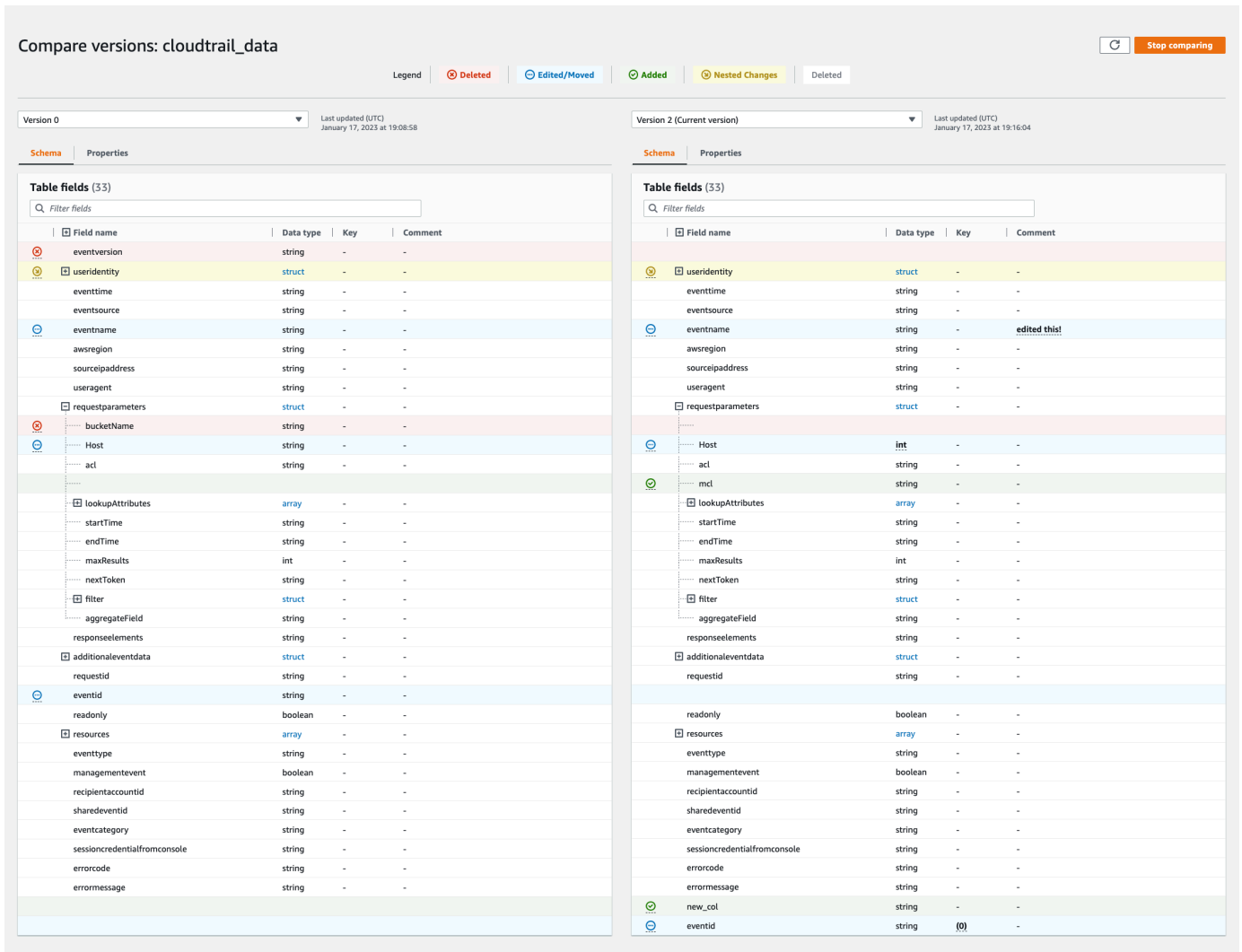
두 버전의 테이블 스키마를 비교할 때 중첩된 행을 확장 및 축소하여 중첩된 행 변경 내용을 비교하고, 두 버전의 스키마를 나란히 비교하고, 테이블 속성을 나란히 볼 수 있습니다.

버전을 비교하려면

1. AWS Glue Console에서 테이블, 작업을 선택한 다음 버전 비교를 선택합니다.



2. 버전 드롭다운 메뉴를 선택하여 비교할 버전을 선택합니다. 스키마를 비교할 때 스키마 탭은 주황색으로 강조 표시됩니다.
3. 두 버전 간에 테이블을 비교하면 화면 왼쪽과 오른쪽에 테이블 스키마가 표시됩니다. 이렇게 하면 열 이름, 데이터 유형, 키 및 설명 필드를 나란히 비교하여 변경 내용을 시각적으로 확인할 수 있습니다. 변경 사항이 있을 경우 색상 아이콘에 적용된 변경 유형이 표시됩니다.
 - 삭제됨 - 빨간색 아이콘으로 표시되면 이전 버전의 테이블 스키마에서 해당 열이 제거된 위치를 나타냅니다.
 - 편집됨 또는 이동됨 - 파란색 아이콘으로 표시되면 새 버전의 테이블 스키마에서 열이 수정 또는 이동된 위치를 나타냅니다.
 - 추가됨 - 녹색 아이콘으로 표시되면 해당 열이 새 버전의 테이블 스키마에 추가된 위치를 나타냅니다.
 - 중첩된 변경 내용 - 노란색 아이콘으로 표시되면 중첩된 열에 변경 내용이 포함된 위치를 나타냅니다. 확장할 열을 선택하면 삭제, 편집, 이동 또는 추가된 열이 표시됩니다.



4. 필터 필드 검색 창을 사용하면 여기에 입력한 문자를 기반으로 필드를 표시할 수 있습니다. 테이블 버전 중 하나에 열 이름을 입력하면 필터링된 필드가 두 테이블 버전 모두에 표시되어 변경 사항이 발생한 위치를 보여 줍니다.
5. 속성을 비교하려면 속성 탭을 선택합니다.
6. 버전 비교를 중지하려면 비교 중지를 선택하여 테이블 목록으로 돌아갑니다.

AWS Glue에서 파티션 인덱스 작업

시간이 지남에 따라 수십만 개의 파티션이 테이블에 추가됩니다. [GetPartitions API](#)는 테이블의 파티션을 가져오는 데 사용됩니다. API는 요청에 제공된 표현식과 일치하는 파티션을 반환합니다.

Country, Category, Year, Month, creationDate 키로 분할된 sales_data 테이블을 예로 들어 보겠습니다. 2020년 2020-08-15 이후에 도서 카테고리에서 판매된 모든 상품에 대한 판매 데이터를

얻으려면 데이터 카탈로그에 "Category = 'Books' 및 creationDate > '2020-08-15'"라는 표현으로 GetPartitions 요청을 해야 합니다.

테이블에 파티션 인덱스가 없으면 AWS Glue는 테이블의 모든 파티션을 로드한 다음 GetPartitions 요청에서 사용자가 제공한 쿼리 표현식을 사용하여 로드된 파티션을 필터링합니다. 인덱스가 없는 테이블에서 파티션 수가 증가하면 쿼리를 실행하는 데 더 많은 시간이 걸립니다. 인덱스를 사용하면 GetPartitions 쿼리는 테이블의 모든 파티션을 로드하는 대신 파티션의 하위 집합을 가져오려고 시도합니다.

주제

- [파티션 인덱스 정보](#)
- [파티션 인덱스로 테이블 생성](#)
- [기존 테이블에 파티션 인덱스 추가](#)
- [테이블의 파티션 인덱스 설명](#)
- [파티션 인덱스 사용에 대한 제한 사항](#)
- [최적화된 GetPartitions 호출에 인덱스 사용](#)
- [엔진과의 통합](#)

파티션 인덱스 정보

파티션 인덱스를 생성할 때 지정된 테이블에 이미 존재하는 파티션 키 목록을 지정합니다. 파티션 인덱스는 테이블에 정의된 파티션 키의 하위 목록입니다. 테이블에 정의된 파티션 키의 순열에 대해 파티션 인덱스를 생성할 수 있습니다. 위의 sales_data 테이블에 대해 가능한 인덱스는 (country, category, creationDate), (country, category, year), (country, category), (country), (category, country, year, month) 등입니다.

Data Catalog는 인덱스 생성 시 제공된 순서대로 파티션 값을 연결합니다. 인덱스는 파티션이 테이블에 추가될 때 일관되게 구축됩니다. 문자열(string, char, varchar), 숫자(int, bigint, long, tinyint, smallint), 날짜(yyyy-MM-dd) 열 유형에 대한 인덱스를 생성할 수 있습니다.

지원되는 데이터 유형

- 날짜 - ISO 형식의 날짜(예: YYYY-MM-DD)입니다. 예를 들어 날짜 2020-08-15입니다. 형식은 하이픈(-)을 사용하여 연도, 월, 일을 구분합니다. 인덱싱에 허용되는 날짜 범위는 0000-01-01에서 9999-12-31까지입니다.
- String - 작은따옴표 또는 큰따옴표로 묶인 문자열 리터럴입니다.
- Char - 길이가 1~255자로 지정된 고정 길이 문자 데이터입니다(예: char(10)).

- Varchar - 길이가 1~65535자로 지정된 가변 길이 문자 데이터입니다(예: varchar(10)).
- 숫자 - int, bigint, long, tinyint, smallint

숫자, 문자열, 날짜 데이터 유형에 대한 인덱스는 =, >, >=, <, <=, between 연산자를 지원합니다. 인덱싱 솔루션은 현재 AND 논리 연산자만 지원합니다. "LIKE", "IN", "OR" 및 "NOT" 연산자가 있는 하위 표현식은 인덱스를 사용하여 필터링하는 표현식에서 무시됩니다. 무시된 하위 표현식에 대한 필터링은 인덱스 필터링을 적용한 후 가져온 파티션에서 수행됩니다.

테이블에 추가된 각 파티션에 대해 해당 인덱스 항목이 생성됩니다. 'n'개의 파티션이 있는 테이블의 경우 1개의 파티션 인덱스는 'n'개의 파티션 인덱스 항목을 생성합니다. 동일한 테이블의 'm' 파티션 인덱스는 'm*n' 파티션 인덱스 항목이 됩니다. 각 파티션 인덱스 항목은 데이터 카탈로그 스토리지에 대한 현재 AWS Glue 가격 정책에 따라 요금이 부과됩니다. 스토리지 객체 요금에 대한 자세한 내용은 [AWS Glue 요금](#)을 참조하세요.

파티션 인덱스로 테이블 생성

테이블 생성 중 파티션 인덱스를 생성할 수 있습니다. CreateTable 요청은 [PartitionIndex 객체](#) 목록을 입력으로 사용합니다. 지정된 테이블에 최대 3개의 파티션 인덱스를 생성할 수 있습니다. 각 파티션 인덱스에는 테이블에 대해 정의된 이름과 partitionKeys 목록이 필요합니다. 테이블에 생성된 인덱스는 [GetPartitionIndexes API](#)를 사용하여 가져올 수 있습니다.

기존 테이블에 파티션 인덱스 추가

파티션 인덱스를 기존 테이블에 추가하려면 CreatePartitionIndex 작업을 사용합니다. 단, CreatePartitionIndex 작업당 PartitionIndex 하나를 생성할 수 있습니다. 인덱스가 생성되는 동안 테이블을 계속 사용할 수 있으므로 인덱스를 추가해도 테이블의 가용성에는 영향이 없습니다.

추가된 파티션의 인덱스 상태는 CREATING으로 설정되고 인덱스 데이터 생성이 시작됩니다. 인덱스 생성 프로세스가 성공하면 indexStatus가 ACTIVE로 업데이트되고 실패한 프로세스에 대해 인덱스 상태가 FAILED로 업데이트됩니다. 인덱스 생성은 여러 가지 이유로 실패할 수 있으며 GetPartitionIndexes 작업을 사용하여 실패 세부 정보를 검색할 수 있습니다. 가능한 실패는 다음과 같습니다.

- ENCRYPTED_PARTITION_ERROR - 암호화된 파티션이 있는 테이블에 대한 인덱스 생성은 지원되지 않습니다.
- INVALID PARTITION TYPE DATA_ERROR - partitionKey 값이 해당 partitionKey 데이터 유형에 대해 유효한 값이 아닐 때 관찰됩니다. 예: 'int' 데이터 유형을 가진 partitionKey는 'foo' 값을 갖습니다.

- MISSING_PARTITION_VALUE_ERROR - indexedKey에 대한 partitionValue가 없을 때 관찰됩니다. 이는 테이블이 일관되게 분할되지 않은 경우에 발생할 수 있습니다.
- UNSUPPORTED_PARTITION_CHARACTER_ERROR - 인덱싱된 파티션 키의 값에 \u0000, \u0001 또는 \u0002 문자가 포함될 때 관찰됩니다.
- INTERNAL_ERROR - 인덱스를 생성하는 동안 내부 오류가 발생했습니다.

테이블의 파티션 인덱스 설명

테이블에 생성된 파티션 인덱스를 가져오려면 GetPartitionIndexes 연산을 사용합니다. 응답은 현재 각 인덱스 상태(IndexStatus)와 함께 테이블의 모든 인덱스를 반환합니다.

파티션 인덱스의 IndexStatus는 다음 중 하나가 됩니다.

- CREATING - 현재 인덱스를 생성 중이며, 아직 사용할 수 없습니다.
- ACTIVE - 인덱스를 사용할 준비가 되었습니다. 요청은 인덱스를 사용하여 최적화된 쿼리를 수행할 수 있습니다.
- DELETING - 현재 인덱스를 삭제하고 있으며 더 이상 사용할 수 없습니다. 활성 상태의 인덱스는 상태를 ACTIVE에서 DELETING으로 이동하는 DeletePartitionIndex 요청을 사용하여 삭제할 수 있습니다.
- FAILED - 기존 테이블에 대한 인덱스 생성에 실패했습니다. 각 테이블은 마지막 10개의 실패한 인덱스를 저장합니다.

기존 테이블에 생성된 인덱스의 가능한 상태 전환은 다음과 같습니다.

- CREATING → ACTIVE → DELETING
- CREATING → FAILED

파티션 인덱스 사용에 대한 제한 사항

파티션 인덱스를 생성한 후에는 테이블 및 파티션 기능에 대한 다음 변경 사항을 확인합니다.

새 파티션 생성(인덱스 추가 후)

테이블에 파티션 인덱스가 생성된 후 테이블에 추가된 모든 새 파티션은 인덱싱된 키에 대한 데이터 유형 검사에 대해 검증됩니다. 인덱싱된 키의 파티션 값은 데이터 유형 포맷에 대해 검증됩니다. 데이터 유형 검사가 실패하면 파티션 생성 작업이 실패합니다. sales_data 테이블의 경우 범주가 string 유

형이고 연도가 int 유형인 키(category, year)에 대한 인덱스가 생성되면 YEAR 값이 "foo"인 새 파티션 생성이 실패합니다.

인덱스가 사용되면 U+0000, U+00001 및 U+0002 문자가 있는 인덱스 키 값이 있는 파티션 추가가 실패하기 시작합니다.

테이블 업데이트

테이블에 파티션 인덱스가 생성되면 기존 파티션 키의 파티션 키 이름을 수정할 수 없으며 인덱스에 등록된 키의 유형이나 순서를 변경할 수 없습니다.

최적화된 GetPartitions 호출에 인덱스 사용

인덱스가 있는 테이블에서 GetPartitions를 호출할 때 표현식을 포함할 수 있으며 해당하는 경우 Data Catalog는 가능한 경우 인덱스를 사용합니다. 인덱스의 첫 번째 키는 필터링에 사용할 인덱스에 대한 표현식에 전달되어야 합니다. 필터링의 인덱스 최적화는 최선의 노력으로 적용됩니다. Data Catalog는 가능한 한 인덱스 최적화를 사용하려고 시도하지만 인덱스가 누락되거나 지원되지 않는 연산자의 경우 모든 파티션을 로드하는 기존 구현으로 폴백합니다.

위의 sales_data 테이블에 대해 인덱스 [Country, Category, Year]를 추가합니다. 표현식에 "Country"가 전달되지 않으면 등록된 인덱스가 인덱스를 사용하여 파티션을 필터링할 수 없습니다. 다양한 쿼리 패턴을 지원하기 위해 최대 3개의 인덱스를 추가할 수 있습니다.

몇 가지 예제 표현식을 사용하여 인덱스가 어떻게 작동하는지 살펴보겠습니다.

Expressions	인덱스 사용 방법
Country = 'US'	파티션을 필터링하는 데 인덱스가 사용됩니다.
Country = 'US' 및 Category = 'Shoes'	파티션을 필터링하는 데 인덱스가 사용됩니다.
Category = 'Shoes'	표현식에 "country"가 제공되지 않으므로 인덱스는 사용되지 않습니다. 응답을 반환하기 위해 모든 파티션이 로드됩니다.
Country = 'US', Category = 'Shoes' 및 Year > '2018'	파티션을 필터링하는 데 인덱스가 사용됩니다.
Country = 'US', Category = 'Shoes', Year > '2018' 및 month = 2	country = "US", category = "shoes", year > 2018인 모든 파티션을 가져오는 데 인덱스가 사

Expressions	인덱스 사용 방법
	용됩니다. 그런 다음 월 표현식에 대한 필터링이 수행됩니다.
Country = 'US' AND Category = 'Shoes' OR Year > '2018'	표현식에 OR 연산자가 있으므로 인덱스가 사용되지 않습니다.
Country = 'US' AND Category = 'Shoes' AND (Year = 2017 OR Year = '2018')	country = "US" 및 category = "shoes"인 모든 파티션을 가져오는 데 인덱스가 사용된 다음 연도 표현식에 대한 필터링이 수행됩니다.
Country in ('US', 'UK') AND Category = 'Shoes'	IN 연산자는 현재 지원되지 않으므로 필터링에 인덱스가 사용되지 않습니다.
Country = 'US' AND Category in ('Shoes', 'Books')	country = "US"인 모든 파티션을 가져오는 데 인덱스가 사용된 다음 범주 표현식에 대한 필터링이 수행됩니다.
Country = 'US' AND Category in ('Shoes', 'Books') AND (creationDate > '2023-9-01')	country = "US" 및 creationDate > '2023-9-01'인 모든 파티션을 가져오는 데 인덱스가 사용된 다음 범주 표현식에 대한 필터링이 수행됩니다.

엔진과의 통합

Redshift Spectrum, Amazon EMR 및 AWS Glue ETL Spark DataFrames는 인덱스가 AWS Glue에서 ACTIVE 상태인 후 파티션을 가져 오기 위해 인덱스를 사용할 수 있습니다. [Athena](#) 및 [AWS Glue ETL 동적 프레임](#)에서 쿼리 개선을 위해 인덱스를 활용하려면 추가 단계를 따라야 합니다.

파티션 필터링 사용 설정

Athena에서 파티션 필터링을 사용하려면 다음과 같이 테이블 속성을 업데이트해야 합니다.

1. AWS Glue 콘솔의 데이터 카탈로그에서 테이블을 선택합니다.
2. 테이블을 선택합니다.
3. 작업에서 테이블 편집을 선택합니다.
4. 테이블 속성에서 다음을 추가합니다.
 - 키 - partition_filtering.enabled

- 값 - true

5. 적용을 선택합니다.

또는 Athena에서 [ALTER TABLE SET PROPERTIES](#) 쿼리를 실행하여 이 파라미터를 설정할 수도 있습니다.

```
ALTER TABLE partition_index.table_with_index
SET TBLPROPERTIES ('partition_filtering.enabled' = 'true')
```

다른 AWS 서비스와 통합

AWS Glue 크롤러를 사용하여 AWS Glue Data Catalog를 채울 수 있지만 데이터 카탈로그와 자동으로 통합되어 채울 수 있는 여러 AWS 서비스가 있습니다. 다음 섹션에서는 AWS 서비스가 지원하며 데이터 카탈로그를 채울 수 있는 특정 사용 사례에 대한 자세한 정보를 제공합니다.

주제

- [AWS Lake Formation](#)
- [Amazon Athena](#)

AWS Lake Formation

AWS Lake Formation은 AWS에서 안전한 데이터 레이크를 더 쉽게 설정할 수 있게 해주는 서비스입니다. Lake Formation은 AWS Glue에 기반하며, Lake Formation과 AWS Glue는 동일한 AWS Glue Data Catalog를 공유합니다. Amazon S3 데이터 위치를 Lake Formation에 등록하고, Lake Formation 콘솔을 사용하여 AWS Glue 데이터 카탈로그에 데이터베이스 및 테이블을 생성하고, 데이터 액세스 정책을 정의하고, 중앙에서 데이터 레이크 전체의 데이터 액세스를 감사할 수 있습니다. Lake Formation의 세분화된 액세스 제어를 사용하여 기존 데이터 카탈로그 리소스와 Amazon S3 데이터 위치를 관리할 수 있습니다.

Lake Formation에 데이터를 등록하면 IAM 보안 주체, AWS 계정, AWS 조직 및 조직 단위 간에서 데이터 카탈로그 리소스를 안전하게 공유할 수 있습니다.

Lake Formation을 사용하여 데이터 카탈로그 리소스를 생성하는 방법에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 [데이터 카탈로그 테이블 및 데이터베이스 생성](#)을 참조하세요.

Amazon Athena

Amazon Athena는 데이터 카탈로그를 사용하여 AWS 계정의 Amazon S3 데이터에 대한 테이블 메타데이터를 저장하고 검색합니다. Athena 쿼리 엔진은 테이블 메타데이터를 통해 쿼리하려는 데이터를 찾고, 읽고, 처리할 방법을 파악합니다.

Athena CREATE TABLE 문을 직접 사용하여 AWS Glue Data Catalog를 채울 수 있습니다. 크롤러를 실행할 필요 없이 데이터 카탈로그에서 스키마 및 파티션 메타데이터를 수동으로 정의하고 채울 수 있습니다.

1. Athena 콘솔에서 데이터 카탈로그에 테이블 메타데이터를 저장할 데이터베이스를 생성합니다.
2. CREATE EXTERNAL TABLE 문을 사용하여 데이터 소스의 스키마를 정의합니다.
3. 데이터가 분할된 경우 PARTITIONED BY 절을 사용하여 파티션 키를 정의합니다.
4. LOCATION 절을 사용하여 실제 데이터 파일이 저장되는 Amazon S3 경로를 지정합니다.
5. CREATE TABLE 문을 실행하세요.

이 쿼리는 데이터를 실제로 크롤링하지 않고 정의된 스키마와 파티션을 기반으로 데이터 카탈로그에 테이블 메타데이터를 생성합니다.

이제 Athena에서 이 테이블을 쿼리하면 데이터 카탈로그의 메타데이터를 사용하여 Amazon S3의 데이터 파일에 액세스하고 쿼리합니다.

자세한 내용은 Amazon Athena 사용 설명서의 [데이터베이스 및 테이블 생성](#)을 참조하세요.

데이터 카탈로그 설정

데이터 카탈로그 설정에는 계정의 데이터 카탈로그에 대한 암호화 및 사용 권한 옵션을 설정하는 옵션이 있습니다.

Data catalog settings

Last updated (UTC)
January 1, 1970 at 00:00:00



Choose encryption and permission options for your accounts data catalog.

Encryption options

- Metadata encryption**
Enable at-rest encryption for metadata stored in the data catalog.
- Encrypt connection passwords**
When enabled, the password you provide when you create a connection is encrypted with the given KMS key.

Permissions

Add a policy to define fine-grained access control of the data catalog.

1	
---	--

JSON Ln 1, Col 1 Errors: 0 Warnings: 0

Cancel **Save**

Data Catalog의 세분화된 액세스 제어를 변경하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 암호화 옵션을 선택합니다.

- 메타데이터 암호화 - 이 확인란을 선택하여 데이터 카탈로그의 메타데이터를 암호화합니다. 메타데이터는 지정된 AWS Key Management Service(AWS KMS) 키를 사용하여 암호화됩니다. 자세한 내용은 [데이터 카탈로그 암호화](#) 단원을 참조하십시오.
- 연결 암호 암호화 - 이 확인란을 선택하여 연결이 생성되거나 업데이트될 때 AWS Glue 연결 객체의 암호를 암호화할 수 있습니다. 암호는 지정한 AWS KMS 키를 사용하여 암호화됩니다. 암호가 반환될 때 암호화됩니다. 이 옵션은 Data Catalog의 모든 AWS Glue 연결에 대한 전역 설정입니다. 이 확인란을 선택 취소하면 이전에 암호화된 암호가 생성되었거나 업데이트되었을 때 사용된 키를 사용하여 암호화된 상태로 유지됩니다. AWS Glue 연결에 대한 자세한 정보는 [데이터에 연결](#) 섹션을 참조하세요.

이 옵션을 활성화하는 경우 AWS KMS 키를 선택하거나 Enter a key ARN(키 ARN 입력)을 선택하고 키에 대한 Amazon 리소스 이름(ARN)을 제공합니다.

`arn:aws:kms:region:account-id:key/key-id` 형식에 ARN 이름을 입력합니다. 또한 `arn:aws:kms:region:account-id:alias/alias-name` 같은 키 별칭으로 ARN을 제공할 수도 있습니다.

Important

이 옵션이 선택되어 있는 경우 연결을 생성하거나 업데이트하는 사용자 또는 역할에 지정된 KMS 키에 대한 `kms:Encrypt` 권한이 있어야 합니다.

자세한 내용은 [연결 암호 암호화](#) 단원을 참조하십시오.

3. [설정(Settings)]을 선택하고 [권한(Permissions)] 편집기에서 계정에 대한 Data Catalog의 세분화된 액세스 제어를 변경할 정책 설명을 추가합니다. 한 번에 정책 하나만 Data Catalog에 연결할 수 있습니다. JSON 리소스 정책을 이 컨트롤에 붙여 넣을 수 있습니다. 자세한 내용은 [Glue 내 리소스 기반 정책 AWS](#) 단원을 참조하십시오.
4. [저장(Save)]을 선택하여 변경 사항으로 Data Catalog를 업데이트합니다.

또한 AWS Glue API 작업을 사용하여 리소스 정책을 설정하고, 가져오고, 삭제할 수 있습니다. 자세한 내용은 [AWS Glue의 보안 API](#) 단원을 참조하십시오.

트랜잭션 테이블 채우기 및 관리

[Apache Iceberg](#), [Apache Hudi](#) 및 Linux Foundation [Delta Lake](#)는 Apache Spark에서 대규모 데이터 분석 및 데이터 레이크 워크로드를 처리하도록 설계된 오픈 소스 테이블 형식입니다.

다음과 같은 방법을 사용하여 AWS Glue Data Catalog에서 Iceberg, Hudi 및 Delta Lake 테이블을 채울 수 있습니다.

- AWS Glue 크롤러 - AWS Glue 크롤러는 데이터 카탈로그에서 Iceberg, Hudi 및 Delta Lake 테이블 메타데이터를 자동으로 검색하고 채울 수 있습니다. 자세한 내용은 [크롤러를 사용하여 데이터 카탈로그 채우기](#) 단원을 참조하십시오.
- AWS Glue ETL 작업 - ETL 작업을 생성하여 Iceberg, Hudi 및 Delta Lake 테이블에 데이터를 쓰고 데이터 카탈로그에서 해당 메타데이터를 채울 수 있습니다. 자세한 내용은 [AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용](#)을 참조하세요.
- AWS Glue 콘솔, AWS Lake Formation 콘솔, AWS CLI 또는 API - AWS Glue 콘솔, Lake Formation 콘솔 또는 API를 사용하여 데이터 카탈로그에서 Iceberg 테이블 정의를 생성하고 관리할 수 있습니다.

주제

- [Apache Iceberg 테이블 생성](#)
- [Iceberg 테이블 최적화](#)

Apache Iceberg 테이블 생성

Amazon S3에 상주하는 데이터를 사용하여 AWS Glue Data Catalog에서 Apache Parquet 데이터 형식을 사용하는 Apache Iceberg 테이블을 생성할 수 있습니다. 데이터 카탈로그의 테이블은 데이터 스토어의 데이터를 표현하는 메타데이터 정의입니다. 기본적으로 AWS Glue는 Iceberg v2 테이블을 생성합니다. v1과 v2 테이블의 차이점은 Apache Iceberg 설명서의 [Format version changes](#)(포맷 버전 변경 사항)을 참조하세요.

[Apache Iceberg](#)는 매우 큰 분석 데이터 세트를 위한 오픈 테이블 형식입니다. Iceberg를 사용하면 스키마를 쉽게 변경할 수 있습니다(이를 스키마 진화라고도 함). 다시 말해서 사용자는 기본 데이터를 손상시키지 않고 데이터 테이블에서 열을 추가하거나, 이름을 바꾸거나, 제거할 수 있습니다. 또한 Iceberg는 사용자가 시간 경과에 따른 데이터 변경 사항을 추적할 수 있는 데이터 버전 관리를 지원합니다. 이를 통해 사용자는 과거 버전의 데이터에 액세스하여 데이터를 쿼리하고 업데이트와 삭제 사이의 데이터 변화를 분석할 수 있습니다.

AWS Glue 또는 Lake Formation 콘솔을 사용하거나 AWS Glue API의 CreateTable 작업을 사용하여 데이터 카탈로그에 Iceberg 테이블을 생성할 수 있습니다. 자세한 내용은 [CreateTable 작업\(Python: create_table\)](#)을 참조하세요.

데이터 카탈로그에서 Iceberg 테이블을 생성할 때 Amazon S3에서 테이블 형식과 메타데이터 파일 경로를 지정해야 읽기 및 쓰기를 수행할 수 있습니다.

Lake Formation을 사용하면 AWS Lake Formation에 Amazon S3 데이터 위치를 등록할 때 세분화된 액세스 제어 권한을 사용하여 Iceberg 테이블을 보호할 수 있습니다. Lake Formation에 등록되지 않은 Amazon S3의 원본 데이터와 메타데이터의 경우 액세스 권한이 Amazon S3 및 AWS Glue 작업에 대한 IAM 권한 정책에 의해 결정됩니다. 자세한 내용은 [권한 관리](#)를 참조하세요.

Note

데이터 카탈로그는 파티션 생성 및 Iceberg 테이블 속성 추가를 지원하지 않습니다.

필수 조건

데이터 카탈로그에서 Iceberg 테이블을 생성하고 Lake Formation 데이터 액세스 권한을 설정하려면 다음 요구 사항을 완료해야 합니다.

1. Lake Formation에 등록된 데이터 없이 Iceberg 테이블을 생성하는 데 필요한 권한.

데이터 카탈로그에서 테이블을 생성하는 데 필요한 권한 외에도 테이블 생성자는 다음 권한이 필요합니다.

- 리소스 `arn:aws:s3:::{bucketName}`에 대한 `s3:PutObject`
- 리소스 `arn:aws:s3:::{bucketName}`에 대한 `s3:GetObject`
- 리소스 `arn:aws:s3:::{bucketName}`에 대한 `s3:DeleteObject`

2. Lake Formation에 등록된 데이터를 사용하여 Iceberg 테이블을 생성하는 데 필요한 권한.

Lake Formation을 사용하여 데이터 레이크의 데이터를 관리하고 보호하려면 테이블을 위한 데이터가 있는 Amazon S3 위치를 Lake Formation에 등록합니다. 이는 Lake Formation이 Athena, Redshift Spectrum 및 Amazon EMR과 같은 AWS 분석 서비스에 보안 인증 정보를 벤딩하여 데이터에 액세스할 수 있도록 하기 위함입니다. Amazon S3 위치 등록에 대한 자세한 내용은 [데이터 레이크에 Amazon S3 위치 추가](#) 섹션을 참조하세요.

Lake Formation에 등록된 기본 데이터를 읽고 쓰는 보안 주체는 다음과 같은 권한이 필요합니다.

- `lakeformation:GetDataAccess`
- `DATA_LOCATION_ACCESS`

위치에 대한 데이터 위치 권한이 있는 보안 주체는 모든 하위 위치에 대한 위치 권한도 갖습니다.

데이터 위치 권한에 대한 자세한 내용은 [기본 데이터 액세스 제어](#) 섹션을 참조하세요.

압축을 활성화하려면 서비스가 데이터 카탈로그의 테이블을 업데이트할 권한이 있는 IAM 역할을 맡아야 합니다. 자세한 내용은 [테이블 최적화 필수 조건](#) 을 참조하세요.

Iceberg 테이블 생성

이 페이지에 설명된 대로 AWS Glue 또는 Lake Formation 콘솔을 사용하거나 AWS Command Line Interface를 사용하여 Iceberg v1 및 v2 테이블을 생성할 수 있습니다. 또는 AWS Glue 크롤러를 사용하여 Iceberg 테이블을 생성할 수도 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 [데이터 카탈로그 및 크롤러](#)를 참조하세요.

Iceberg 테이블을 생성하려면

Console

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 데이터 카탈로그에서 테이블을 선택하고 테이블 생성 버튼을 사용하여 다음 속성을 지정합니다.
 - 테이블 이름 - 테이블 이름을 입력합니다. Athena를 사용하여 테이블에 액세스하는 경우 Amazon Athena 사용 설명서의 [이름 지정 팁](#)을 사용하세요.
 - 데이터베이스 - 기존 데이터베이스를 선택하거나 새 데이터베이스를 생성합니다.
 - 설명 - 테이블에 대한 설명입니다. 테이블 내용을 이해할 수 있도록 설명을 적을 수 있습니다.
 - 테이블 형식 - 테이블 형식으로 Apache Iceberg를 선택합니다.
 - 압축 활성화 - 테이블의 작은 Amazon S3 객체를 더 큰 객체로 압축하려면 압축 활성화를 선택합니다.
 - IAM 역할 - 압축을 실행하기 위해 서비스는 사용자를 대신하여 IAM 역할을 맡습니다. 드롭다운을 사용하여 IAM 역할을 선택할 수 있습니다. 압축 기능을 활성화하는 데 필요한 권한이 역할에 있는지 확인합니다.

필요한 권한에 대해 알아보려면 [테이블 최적화 필수 조건](#) 섹션을 참조하세요.

- 위치 - 메타데이터 테이블을 저장하는 Amazon S3의 폴더 경로를 지정합니다. Iceberg가 읽기 및 쓰기를 수행하려면 데이터 카탈로그에 메타데이터 파일과 위치가 필요합니다.
- 스키마 - 열 추가를 선택하여 열과 열의 데이터 유형을 추가합니다. 빈 테이블을 생성하고 나중에 스키마를 업데이트할 수 있습니다. 데이터 카탈로그는 Hive 데이터 유형을 지원합니다. 자세한 내용은 [Hive 데이터 유형](#)을 참조하세요.

Iceberg를 사용하면 테이블을 생성한 후 스키마와 파티션을 개선할 수 있습니다. [Athena 쿼리](#)를 사용하여 테이블 스키마를 업데이트하고 [Spark 쿼리](#)를 사용하여 파티션을 업데이트할 수 있습니다.

AWS CLI

```
aws glue create-table \
  --database-name iceberg-db \
  --region us-west-2 \
  --open-table-format-input '{
    "IcebergInput": {
      "MetadataOperation": "CREATE",
      "Version": "2"
    }
  }' \
  --table-input '{"Name":"test-iceberg-input-demo",
    "TableType": "EXTERNAL_TABLE",
    "StorageDescriptor":{
      "Columns":[
        {"Name":"col1", "Type":"int"},
        {"Name":"col2", "Type":"int"},
        {"Name":"col3", "Type":"string"}
      ],
      "Location":"s3://DOC_EXAMPLE_BUCKET_ICEBERG/"
    }
  }'
```

Iceberg 테이블 최적화

Apache Iceberg와 같은 오픈 테이블 형식을 사용하는 Amazon S3 데이터 레이크는 데이터를 Amazon S3 객체로 저장합니다. 데이터 레이크 테이블에 수천 개의 작은 Amazon S3 객체가 있으면 Iceberg 테이블의 메타데이터 오버헤드가 증가하고 읽기 성능에 영향을 줍니다. Amazon Athena, Amazon EMR,

AWS Glue ETL 작업과 같은 AWS 분석 서비스를 통한 읽기 성능 향상을 위해 AWS Glue Data Catalog은 Data Catalog의 Iceberg 테이블에 대한 관리형 압축(작은 Amazon S3 객체를 더 큰 객체로 압축하는 프로세스)을 제공합니다. Lake Formation 콘솔, AWS Glue 콘솔, AWS CLI 또는 AWS API를 사용하여 데이터 카탈로그에 있는 개별 Iceberg 테이블에 대한 압축을 활성화하거나 비활성화할 수 있습니다.

테이블 옵티마이저는 테이블 파티션을 지속적으로 모니터링하여 파일 수 및 파일 크기가 임계값을 초과할 경우 압축 프로세스를 시작합니다. Iceberg 테이블은 `write.target-file-size-bytes` 속성에 지정된 파일 크기가 128MB~512MB 범위 내에 있는 경우 압축에 적합합니다. 데이터 카탈로그에서 테이블에 5개 이상의 파일이 있고 각 파일이 `write.target-file-size-bytes` 속성의 75%보다 작은 경우 압축 프로세스가 시작됩니다.

예를 들어 `write.target-file-size-bytes` 속성에 파일 크기 임계값이 512MB(앞서 설명한 128MB~512MB 범위 이내)로 설정된 테이블이 있고 이 테이블에는 10개의 파일이 포함되어 있다고 가정합니다. 파일 10개 중 6개가 384MB(0.75×512)보다 작은 경우 데이터 카탈로그가 압축을 트리거합니다.

데이터 카탈로그는 동시 쿼리를 방해하지 않고 압축을 수행합니다. 데이터 카탈로그는 Parquet 형식의 테이블에 대해서만 데이터 압축을 지원합니다.

지원되는 데이터 유형, 압축 형식 및 제한은 [관리형 데이터 압축에 지원되는 형식 및 제한 사항](#) 섹션을 참조하십시오.

주제

- [테이블 최적화 필수 조건](#)
- [압축 활성화](#)
- [압축 비활성화](#)
- [압축 세부 정보 보기](#)
- [Amazon CloudWatch 지표 보기](#)
- [옵티마이저 삭제](#)
- [관리형 데이터 압축에 지원되는 형식 및 제한 사항](#)

테이블 최적화 필수 조건

테이블 최적화 프로그램은 테이블에 압축 기능을 사용할 때 사용자가 지정하는 AWS Identity and Access Management (IAM) 역할의 권한을 갖습니다. IAM 역할에는 데이터 카탈로그의 데이터를 읽고 메타데이터를 업데이트할 수 있는 권한이 있어야 합니다. IAM 역할을 생성하여 다음 인라인 정책을 연결할 수 있습니다.

- Lake Formation에 등록되지 않은 데이터에 대한 위치에 대한 Amazon S3 읽기/쓰기 권한을 부여하는 다음 인라인 정책을 추가하세요. 이 정책에는 데이터 카탈로그의 테이블을 업데이트하고 AWS Glue에게 Amazon CloudWatch 로그에 로그를 추가하고 지표를 게시할 수 있는 권한도 포함되어 있습니다. Lake Formation에 등록되지 않은 Amazon S3의 원본 데이터는 액세스 권한이 Amazon S3 및 AWS Glue 작업에 대한 IAM 권한 정책에 의해 결정됩니다.

다음 인라인 정책에서는 Amazon S3 버킷 이름이 있는 `bucket-name`, `aws-account-id`, `region`을 데이터 카탈로그의 유효한 AWS 계정 번호, 리전, `database_name`을 데이터베이스 이름으로, `table_name`은 테이블 이름으로 대체하십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::<bucket-name>/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::<bucket-name>"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "glue:UpdateTable",
        "glue:GetTable"
      ],
      "Resource": [
        "arn:aws:glue:<region>:<aws-account-id>:table/<database-name>/<table-
name>",
```

```

        "arn:aws:glue:<region>:<aws-account-id>:database/<database-name>",
        "arn:aws:glue:<region>:<aws-account-id>:catalog"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/iceberg-compaction/logs:*"
}
]
}

```

- Lake Formation에 등록된 데이터에 대해 압축을 활성화하려면 다음 정책을 사용하십시오.

압축 역할에 테이블에 대한 IAM_ALLOWED_PRINCIPALS 그룹 권한이 부여되지 않은 경우, 해당 역할에는 테이블에 대한 Lake Formation ALTER, DESCRIBE, INSERT 및 DELETE 권한이 필요합니다.

Amazon S3 버킷의 Lake Formation 등록에 대한 자세한 내용은 [데이터 레이크에 Amazon S3 위치 추가](#) 섹션을 참조하세요.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lakeformation:GetDataAccess"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "glue:UpdateTable",
        "glue:GetTable"
      ],
      "Resource": [

```

```

    "arn:aws:glue:<region>:<aws-account-id>:table/<databaseName>/<tableName>",
    "arn:aws:glue:<region>:<aws-account-id>:database/<database-name>",
    "arn:aws:glue:<region>:<aws-account-id>:catalog"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "logs:CreateLogGroup",
    "logs:CreateLogStream",
    "logs:PutLogEvents"
  ],
  "Resource": "arn:aws:logs:<region>:<aws-account-id>:log-group:/aws-glue/iceberg-compaction/logs:*"
}
]
}

```

- (선택 사항) [서버 측 암호화](#)를 사용하여 암호화된 Amazon S3 버킷의 데이터로 Iceberg 테이블을 압축하려면 압축 역할에 Amazon S3 객체를 해독하고 암호화된 버킷에 객체를 쓰기 위한 새 데이터 키를 생성할 수 있는 권한이 필요합니다. 다음 텍스트를 AWS KMS 키 정책에 추가합니다. 버킷 수준 암호화만 지원합니다.

```

{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::<aws-account-id>:role/<compaction-role-name>"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*"
}

```

- (선택 사항) Lake Formation에 등록된 데이터 위치의 경우, 위치를 등록하는 데 사용되는 역할에는 Amazon S3 객체를 해독하고 암호화된 버킷에 객체를 쓰기 위한 새 데이터 키를 생성할 수 있는 권한이 필요합니다. 자세한 내용을 알아보려면 [암호화된 Amazon S3 위치 등록](#)을 참조하십시오.
- (선택 사항) AWS KMS 키가 다른 AWS 계정에 저장되어 있는 경우 압축 역할에 다음 권한을 포함해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": ["arn:aws:kms:<REGION>:<KEY_OWNER_ACCOUNT_ID>:key/<KEY_ID>"] ]
    ]
  ]
}
```

- 압축을 실행하는 데 사용하는 역할에는 해당 역할에 대한 iam:PassRole 권한이 있어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::<account-id>:role/<compaction-role-name>"
      ]
    ]
  ]
}
```

- 압축 프로세스를 실행하는 IAM 역할을 수입하도록 AWS Glue 서비스의 역할에 다음 신뢰 정책을 추가합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
```

```

    "Principal": {
      "Service": "glue.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
}

```

압축 활성화

Lake Formation 콘솔, AWS Glue 콘솔, AWS CLI 또는 AWS API를 사용하여 데이터 카탈로그의 Apache Iceberg 테이블을 압축할 수 있습니다. 새 테이블의 경우 Apache Iceberg를 테이블 형식으로 선택하고 테이블을 생성할 때 압축을 활성화할 수 있습니다. 압축 기능은 새 테이블에 대해 기본적으로 비활성화되어 있습니다.

Console

압축 기능 활성화하는 방법

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 열고 데이터 레이크 관리자, 테이블 작성자 또는 테이블에 대한 `glue:UpdateTable` 및 `lakeformation:GetDataAccess` 권한을 부여 받은 사용자로 로그인합니다.
2. 탐색 창의 데이터 카탈로그에서 테이블을 선택합니다.
3. 테이블 페이지에서 압축을 활성화하려는 열린 테이블 형식의 테이블을 선택한 다음 작업 메뉴에서 압축 활성화를 선택합니다.
4. 테이블을 선택하고 테이블 세부정보 페이지를 열어 압축을 활성화할 수도 있습니다. 페이지 하단에서 테이블 최적화 탭을 선택하고 압축 활성화를 선택합니다.
5. 그런 다음 [테이블 최적화 필수 조건](#) 섹션에 표시된 권한을 사용하여 드롭다운에서 기존 IAM 역할을 선택합니다.

새 IAM 역할 생성 옵션을 선택하면 서비스가 압축을 실행하는 데 필요한 권한이 있는 사용자 지정 역할을 생성합니다.

아래 단계에 따라 기존 IAM 역할을 업데이트하십시오.

- a. IAM 역할에 대한 권한 정책을 업데이트하려면 IAM 콘솔에서 컴팩션을 실행하는 데 사용되는 IAM 역할로 이동합니다.
- b. 권한 추가 섹션에서 정책 생성을 선택합니다. 새로 열린 브라우저 창에서 역할에 사용할 새 정책을 생성합니다.

- c. 정책 생성 페이지에서 JSON 탭을 선택합니다. 필수 조건에 표시된 JSON 코드를 정책 편집기 필드에 복사합니다.

AWS CLI

다음 예제는 압축 기능을 활성화하는 방법을 보여줍니다. 계정 ID를 유효한 AWS 계정 ID로 바꿉니다. 데이터베이스 이름과 테이블 이름을 실제 Iceberg 테이블 이름 및 데이터베이스 이름으로 바꿉니다. `roleArn`을 IAM 역할의 AWS 리소스 이름(ARN)과 압축 실행에 필요한 권한이 있는 IAM 역할의 이름으로 바꿉니다.

```
aws glue create-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --table-optimizer-configuration
  '{"roleArn":"arn:aws:iam::123456789012:role/compaction_role", "enabled":'true'}' \
  --type compaction
```

AWS API

`CreateTableOptimizer` 작업을 호출하여 테이블의 압축을 활성화합니다.

압축을 활성화하면 테이블 최적화 탭에 다음과 같은 압축 세부 정보가 표시됩니다(약 15~20분 후).

시작 시간

Lake Formation 내에서 압축 프로세스가 시작된 시간입니다. 값은 UTC 시간으로 표시된 타임스탬프입니다.

종료 시간

압축 프로세스가 데이터 카탈로그에서 끝나는 시간입니다. 값은 UTC 시간으로 표시된 타임스탬프입니다.

상태 표시기

압축 실행의 상태입니다. 값은 성공 또는 실패입니다.

압축된 파일 수

압축된 총 파일 수입니다.

압축된 바이트 수

압축된 총 바이트 수입니다.

압축 비활성화

AWS Glue 콘솔 또는 AWS CLI를 사용하여 특정 Apache Iceberg 테이블에 대한 자동 압축을 비활성화할 수 있습니다.

Console

1. 데이터 카탈로그를 선택하고 테이블을 선택합니다. 테이블 목록에서 압축을 비활성화하려는 열린 테이블 형식의 테이블을 선택합니다.
2. Iceberg 테이블을 선택하고 작업에서 압축 비활성화를 선택할 수 있습니다.

테이블 세부정보 페이지 하단 섹션에서 압축 비활성화를 선택하여 테이블의 압축을 비활성화할 수도 있습니다.

3. 확인 메시지에서 압축 비활성화를 선택합니다. 압축을 나중에 다시 활성화할 수 있습니다.

확인하면 압축이 비활성화되고 테이블의 압축 상태가 Off로 다시 켜집니다.

AWS CLI

다음 예제에서 계정 ID를 유효한 AWS 계정 ID로 바꿉니다. 데이터베이스 이름과 테이블 이름을 실제 Iceberg 테이블 이름 및 데이터베이스 이름으로 바꿉니다. roleArn을 IAM 역할의 AWS 리소스 이름(ARN)과 압축 실행에 필요한 권한이 있는 IAM 역할의 실제 이름으로 대체합니다.

```
aws glue update-table-optimizer \
  --catalog-id 123456789012 \
  --database-name iceberg_db \
  --table-name iceberg_table \
  --table-optimizer-configuration
  '{"roleArn":"arn:aws:iam::123456789012:role/compaction_role", "enabled":'false'}'\
  --type compaction
```

AWS API

UpdateTableOptimizer 작업을 호출하여 특정 테이블에 대한 압축을 비활성화합니다.

압축 세부 정보 보기

Lake Formation 콘솔, AWS CLI 또는 AWS API 작업을 사용하여 Apache Iceberg의 압축 상태를 볼 수 있습니다.

Console

Iceberg 테이블의 압축 상태를 보려면(콘솔)

- Lake Formation 콘솔의 데이터 카탈로그에서 테이블을 선택하여 Iceberg 테이블의 압축 상태를 볼 수 있습니다. 압축 상태 필드는 압축 실행 상태를 표시합니다. 표 기본 설정을 사용하여 표 형식 및 압축 상태를 표시할 수 있습니다.
- 특정 테이블의 압축 실행 기록을 보려면 AWS Glue Data Catalog에서 테이블을 선택하고 테이블 세부 정보를 볼 테이블을 선택합니다. 테이블 최적화 탭에는 테이블의 압축 기록이 표시됩니다.

AWS CLI

AWS CLI를 사용하여 압축 세부 정보를 볼 수 있습니다.

다음 예제에서 계정 ID를 유효한 AWS 계정 ID로, 데이터베이스 이름과 테이블 이름을 실제 Iceberg 데이터베이스 이름과 테이블 이름으로 바꿉니다.

- 테이블의 마지막 압축 실행 세부 정보를 가져오려면

```
aws get-table-optimizer \  
  --catalog-id 123456789012 \  
  --database-name iceberg_db \  
  --table-name iceberg_table \  
  --type compaction
```

- 다음 예제를 사용하여 특정 테이블에 대한 옵티마이저 기록을 검색할 수 있습니다.

```
aws list-table-optimizer-runs \  
  --catalog-id 123456789012 \  
  --database-name iceberg_db \  
  --table-name iceberg_table \  
  --type compaction
```

- 다음 예제에서는 압축 실행 결과를 검색하는 방법과 여러 최적화 프로그램의 구성 세부 정보를 보여줍니다. 최대 20개의 옵티마이저를 지정할 수 있습니다.


```
aws glue batch-get-table-optimizer \
--entries '[{"catalogId":"123456789012", "databaseName":"iceberg_db",
"tableName":"iceberg_table", "type":"compaction"}]'
```

AWS API

- `GetTableOptimizer` 작업을 사용하여 옵티마이저의 마지막 실행 세부 정보를 검색합니다.
- `ListTableOptimizerRuns` 작업을 사용하여 특정 테이블에서 지정된 옵티마이저의 기록을 검색할 수 있습니다. API 호출 한 번으로 옵티마이저 20개를 지정할 수 있습니다.
- `BatchGetTableOptimizer` 작업을 사용하여 계정의 여러 옵티마이저에 대한 구성 세부 정보를 검색할 수 있습니다. 이 작업은 교차 계정 호출을 지원하지 않습니다.

Amazon CloudWatch 지표 보기

압축을 성공적으로 실행한 후 서비스는 압축 작업 성능에 대한 Amazon CloudWatch 지표를 생성합니다. CloudWatch 지표로 이동하여 지표, 모든 지표를 선택할 수 있습니다. 특정 네임스페이스(예: AWS Glue), 테이블 이름 또는 데이터베이스 이름을 기준으로 지표를 필터링할 수 있습니다.

자세한 내용은 Amazon CloudWatch 사용 설명서의 [사용 가능한 지표 보기](#)를 참조하세요.

- 압축된 바이트 수
- 압축된 파일 수
- 작업에 할당된 DPU 수
- 작업 기간(시간)

옵티마이저 삭제

AWS CLI 또는 AWS API 작업을 사용하여 테이블의 옵티마이저 및 관련 메타데이터를 삭제할 수 있습니다.

다음 AWS CLI 명령을 실행하여 테이블의 압축 기록을 삭제합니다.

```
aws glue delete-table-optimizer \
--catalog-id 123456789012 \
--database-name iceberg_db \
```

```
--table-name iceberg_table \  
--type compaction
```

DeleteTableOptimizer 작업을 사용하여 테이블의 옵티마이저를 삭제합니다.

관리형 데이터 압축에 지원되는 형식 및 제한 사항

Amazon Athena, Amazon EMR 및 AWS Glue ETL 작업과 같은 AWS 분석 서비스를 통한 읽기 성능 향상을 위해 데이터 카탈로그의 Iceberg 테이블에 대한 관리형 압축 (작은 Amazon S3 객체를 큰 객체로 압축하는 프로세스) 을 AWS Glue Data Catalog 제공합니다.

데이터 압축은 암호화된 테이블에서 데이터를 읽는 것을 비롯하여, 데이터 읽기 및 쓰기를 위한 다양한 데이터 형식 및 압축 형식을 지원합니다.

데이터 압축은 다음을 지원합니다.

- 파일 유형 - Parquet
- 데이터 유형 - 부울, 정수, 롱, 플롯, 더블, 문자열, 십진수, 날짜, 시간, 타임스탬프, 문자열, UUID, 바이너리
- 압축 - zstd, gzip, snappy, 비압축
- 암호화 - 데이터 압축은 기본 Amazon S3 암호화(SSE-S3) 및 서버 측 KMS 암호화(SSE-KMS)만 지원합니다.
- 빈 팩 압축
- 스키마 진화
- 대상 파일 크기를 가진 테이블 (쓰기) target-file-size-bytes 128MB ~ 512MB의 포함 범위 내에 있는 빙산 구성의 속성)
- 리전
 - 아시아 태평양(도쿄)
 - 아시아 태평양(서울)
 - 아시아 태평양(뭄바이)
 - 아시아 태평양(싱가포르)
 - 유럽(아일랜드)
 - 유럽(런던)
 - 유럽(프랑크푸르트)
 - 미국 동부(버지니아 북부)
 - 미국 동부(오하이오)

- 미국 서부(캘리포니아 북부)
- 남아메리카(상파울루)
- 기본 데이터를 저장하는 Amazon S3 버킷이 다른 계정에 있는 경우 데이터 카탈로그가 있는 계정에서 압축을 실행할 수 있습니다. 이렇게 하려면 압축 역할에 Amazon S3 버킷에 대한 액세스 권한이 필요합니다.

데이터 압축은 현재 다음을 지원하지 않습니다.

- 파일 유형 - Avro, ORC
- 데이터 유형 - 고정
- 압축 - brotli, lz4
- 파티션 사양이 진화하는 동안 파일을 압축합니다.
- 일반 정렬 또는 z순서 정렬
- 파일 병합 또는 삭제 - 압축 프로세스에서는 관련된 삭제 파일이 있는 데이터 파일을 건너뛵니다.
- 교차 계정 테이블에서의 압축 - 교차 계정 테이블에서는 압축을 실행할 수 없습니다.
- 교차 리전 테이블에서의 압축 - 교차 리전 테이블에서는 압축을 실행할 수 없습니다.
- 리소스 링크에서 압축 활성화
- Amazon S3 버킷에 대한 VPC 엔드포인트
- [DynamoDB 잠금](#) 관리자 — 데이터 압축을 사용할 때 다른 데이터 로드 작업에서 `org.apache.iceberg.aws.dynamodb`를 사용해서는 안 됩니다. `lock-impl DynamoDbLockManager`.

데이터 카탈로그 관리

AWS Glue Data Catalog는 Amazon S3 데이터 세트의 구조 및 운영 메타데이터를 저장하는 중앙 메타데이터 리포지토리입니다. 데이터 카탈로그를 효과적으로 관리하는 것은 데이터 품질, 성능, 보안 및 거버넌스를 유지 관리하는 데 매우 중요합니다.

이러한 데이터 카탈로그 관리 모범 사례를 이해하고 적용하면 데이터 환경의 진화에 발맞춰 메타데이터가 정확하고, 뛰어난 성능을 유지하고, 안전하며, 잘 통제되도록 보장할 수 있습니다.

이 섹션에서는 데이터 카탈로그 관리의 다음과 같은 측면을 다룹니다.

- 테이블 스키마 및 파티션 업데이트: 데이터가 진화함에 따라 데이터 카탈로그에 정의된 테이블 스키마나 파티션 구조를 업데이트해야 할 수 있습니다. AWS Glue ETL을 사용하여 프로그래밍 방식으로

업데이트하는 방법에 대한 자세한 내용은 [AWS Glue ETL 작업을 사용하여 데이터 카탈로그에서 스키마 업데이트 및 새 파티션 추가](#) 섹션을 참조하세요.

- 열 통계 관리: 정확한 열 통계는 쿼리 계획을 최적화하고 성능을 개선하는 데 도움이 됩니다. 열 통계를 생성, 업데이트 및 관리하는 방법에 대한 자세한 정보는 [열 통계를 사용한 쿼리 성능 최적화](#) 섹션을 참조하세요.
- 데이터 카탈로그 암호화: 민감한 메타데이터를 보호하려면 AWS Key Management Service(AWS KMS)를 사용하여 데이터 카탈로그를 암호화할 수 있습니다. 이 섹션에서는 데이터 카탈로그에 대한 암호화를 활성화하고 관리하는 방법을 설명합니다.
- AWS Lake Formation으로 데이터 카탈로그 보호: Lake Formation은 데이터 레이크 보안 및 액세스 제어에 대한 포괄적인 접근 방식을 제공합니다. Lake Formation을 사용하면 데이터 카탈로그 및 기본 데이터에 대한 액세스를 보호하고 관리할 수 있습니다.

주제

- [AWS Glue ETL 작업을 사용하여 데이터 카탈로그에서 스키마 업데이트 및 새 파티션 추가](#)
- [열 통계를 사용한 쿼리 성능 최적화](#)
- [데이터 카탈로그 암호화](#)
- [Lake Formation을 사용한 데이터 카탈로그 보호](#)

AWS Glue ETL 작업을 사용하여 데이터 카탈로그에서 스키마 업데이트 및 새 파티션 추가

추출, 변환 및 로드(ETL) 작업은 대상 데이터 스토어에 새 테이블 파티션을 생성할 수 있습니다. 데이터 집합 스키마는 시간이 지남에 따라 진화하면서 AWS Glue Data Catalog 스키마에서 벗어날 수 있습니다. AWS Glue 이제 ETL 작업에서 여러 기능을 사용해 ETL 스크립트 내에서 Data Catalog의 스키마 및 파티션을 업데이트할 수 있습니다. 이러한 기능을 통해 크롤러를 다시 실행할 필요 없이 Data Catalog의 ETL 작업 결과를 볼 수 있습니다.

새 파티션

AWS Glue Data Catalog의 새 파티션을 보려면 다음 중 하나를 수행할 수 있습니다.

- 작업이 완료된 후 크롤러를 다시 실행하고, 크롤러가 완료되면 콘솔에서 새 파티션을 확인합니다.
- 작업이 완료된 후 크롤러를 다시 실행하지 않고 콘솔에서 바로 새 파티션을 확인합니다. 다음 예제와 같이 몇 줄의 코드를 ETL 스크립트에 추가하여 이 기능을 활성화할 수 있습니다. 이 코드는 새 파티

션이 생성될 때 작업 실행 중에 Data Catalog를 업데이트하는 enableUpdateCatalog 인수를 사용합니다.

방법 1

옵션 인수에서 enableUpdateCatalog 및 partitionKeys를 전달합니다.

Python

```
additionalOptions = {"enableUpdateCatalog": True}
additionalOptions["partitionKeys"] = ["region", "year", "month", "day"]

sink = glueContext.write_dynamic_frame_from_catalog(frame=last_transform,
    database=<target_db_name>,

    table_name=<target_table_name>, transformation_ctx="write_sink",

    additional_options=additionalOptions)
```

Scala

```
val options = JsonOptions(Map(
    "path" -> <S3_output_path>,
    "partitionKeys" -> Seq("region", "year", "month", "day"),
    "enableUpdateCatalog" -> true))
val sink = glueContext.getCatalogSink(
    database = <target_db_name>,
    tableName = <target_table_name>,
    additionalOptions = options)sink.writeDynamicFrame(df)
```

방법 2

getSink()에서 enableUpdateCatalog 및 partitionKeys를 전달하고 DataSink 객체에서 setCatalogInfo()를 호출합니다.

Python

```
sink = glueContext.getSink(
    connection_type="s3",
    path="<S3_output_path>",
    enableUpdateCatalog=True,
```

```
partitionKeys=["region", "year", "month", "day"])
sink.setFormat("json")
sink.setCatalogInfo(catalogDatabase=<target_db_name>,
catalogTableName=<target_table_name>)
sink.writeFrame(last_transform)
```

Scala

```
val options = JsonOptions(
  Map("path" -> <S3_output_path>,
    "partitionKeys" -> Seq("region", "year", "month", "day"),
    "enableUpdateCatalog" -> true))
val sink = glueContext.getSink("s3", options).withFormat("json")
sink.setCatalogInfo(<target_db_name>, <target_table_name>)
sink.writeDynamicFrame(df)
```

이제 크롤러를 다시 실행할 필요 없이 Data Catalog에서 AWS Glue ETL 작업 자체를 사용해 새 카탈로그 테이블을 생성하고, 수정된 스키마로 기존 테이블을 업데이트하고, 새 테이블 파티션을 추가할 수 있습니다.

테이블 스키마 업데이트

Data Catalog 테이블의 스키마를 덮어쓰려면 다음 중 하나를 수행할 수 있습니다.

- 작업이 완료되면 크롤러를 다시 실행하고 크롤러가 테이블 정의도 업데이트하도록 구성되어 있는지 확인합니다. 크롤러가 완료되면 콘솔의 새 파티션과 스키마 업데이트를 확인합니다. 자세한 내용은 [API를 사용해 크롤러 구성](#)을 참조하십시오.
- 작업이 완료된 후 크롤러를 다시 실행하지 않고 콘솔에서 바로 수정된 스키마를 확인합니다. 다음 예제와 같이 몇 줄의 코드를 ETL 스크립트에 추가하여 이 기능을 활성화할 수 있습니다. 이 코드에서는 `enableUpdateCatalog`가 `true`로, `updateBehavior`가 `UPDATE_IN_DATABASE`로 설정되어 있는데, 이를 통해 작업 실행 중에 Data Catalog에서 스키마를 덮어쓰고 새 파티션을 추가하게 됩니다.

Python

```
additionalOptions = {
    "enableUpdateCatalog": True,
    "updateBehavior": "UPDATE_IN_DATABASE"}
additionalOptions["partitionKeys"] = ["partition_key0", "partition_key1"]
```

```
sink = glueContext.write_dynamic_frame_from_catalog(frame=last_transform,
    database=<dst_db_name>,
    table_name=<dst_tbl_name>, transformation_ctx="write_sink",
    additional_options=additionalOptions)
job.commit()
```

Scala

```
val options = JsonOptions(Map(
    "path" -> outputPath,
    "partitionKeys" -> Seq("partition_0", "partition_1"),
    "enableUpdateCatalog" -> true))
val sink = glueContext.getCatalogSink(database = nameSpace, tableName = tableName,
    additionalOptions = options)
sink.writeDynamicFrame(df)
```

또한 테이블 스키마를 덮어쓰지 않고 새 파티션을 추가하려는 경우에는 updateBehavior 값을 LOG로 설정하면 됩니다. updateBehavior의 기본값은 UPDATE_IN_DATABASE이므로 명시적으로 정의하지 않으면 테이블 스키마를 덮어씁니다.

enableUpdateCatalog가 true로 설정되어 있지 않으면 updateBehavior에서 선택한 옵션에 관계 없이 ETL 작업이 Data Catalog의 테이블을 업데이트하지 않습니다.

새 테이블 생성

동일한 옵션을 사용해 Data Catalog에서 새 테이블을 생성할 수도 있습니다. setCatalogInfo를 사용해 데이터베이스 및 새 테이블 이름을 지정할 수 있습니다.

Python

```
sink = glueContext.getSink(connection_type="s3", path="s3://path/to/data",
    enableUpdateCatalog=True, updateBehavior="UPDATE_IN_DATABASE",
    partitionKeys=["partition_key0", "partition_key1"])
sink.setFormat("<format>")
sink.setCatalogInfo(catalogDatabase=<dst_db_name>, catalogTableName=<dst_tbl_name>)
sink.writeFrame(last_transform)
```

Scala

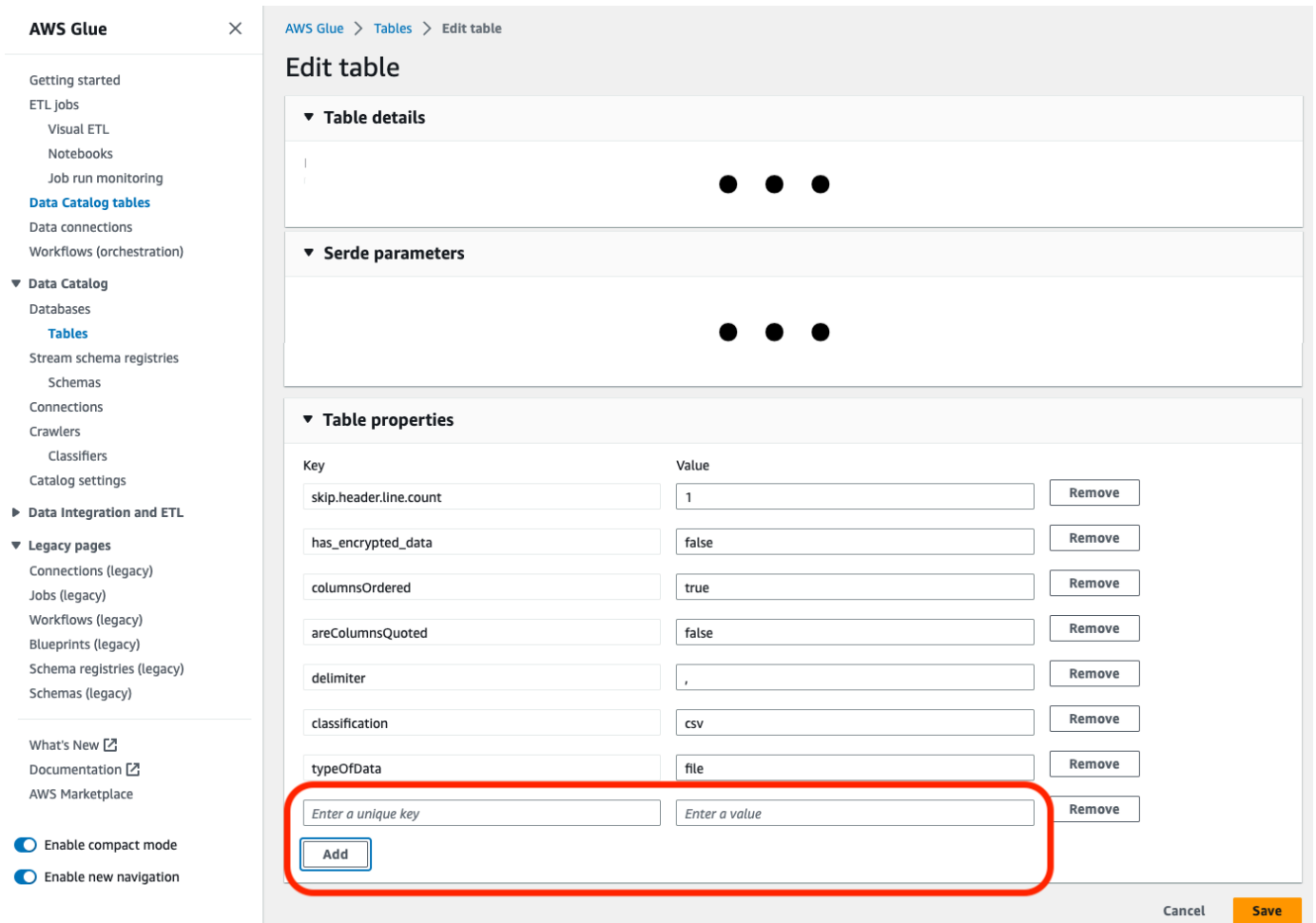
```
val options = JsonOptions(Map(
```

```
"path" -> outputPath,
"partitionKeys" -> Seq("<partition_1>", "<partition_2>"),
"enableUpdateCatalog" -> true,
"updateBehavior" -> "UPDATE_IN_DATABASE"))
val sink = glueContext.getSink(connectionType = "s3", connectionOptions =
options).withFormat("<format>")
sink.setCatalogInfo(catalogDatabase = "<dst_db_name>", catalogTableName =
"<dst_tbl_name>")
sink.writeDynamicFrame(df)
```

제한 사항

다음 제한 사항에 유의하십시오.

- Amazon Simple Storage Service(Amazon S3) 대상만 지원됩니다.
- enableUpdateCatalog 기능은 관리 테이블에서 지원되지 않습니다.
- json, csv, avro 및 parquet 형식만 지원됩니다.
- parquet 분류로 테이블을 생성하거나 업데이트하려면 Dynamic Frames에 대해 AWS Glue 최적화 parquet 라이터를 사용해야 합니다. 이는 다음 중 하나를 통해 수행할 수 있습니다.
- 카탈로그의 기존 테이블을 parquet 분류로 업데이트하는 경우 테이블을 업데이트하기 전에 테이블에서 "useGlueParquetWriter" 테이블 속성이 true로 설정되어 있어야 합니다. AWS Glue API/SDK, 콘솔 또는 Athena DDL 문을 통해 이 속성을 설정할 수 있습니다.



카탈로그 테이블 속성이 설정되면 다음 코드 스니펫을 사용하여 카탈로그 테이블을 새 데이터로 업데이트할 수 있습니다.

```
glueContext.write_dynamic_frame.from_catalog(
    frame=frameToWrite,
    database="dbName",
    table_name="tableName",
    additional_options={
        "enableUpdateCatalog": True,
        "updateBehavior": "UPDATE_IN_DATABASE"
    }
)
```

- 테이블이 카탈로그에 아직 없는 경우 connection_type="s3"를 통해 스크립트에서 getSink() 메서드를 사용하여 Amazon S3에 데이터를 쓰는 작업과 함께 테이블과 해당 파티션을 카탈로그에 추가할 수 있습니다. 워크플로에 적절한 partitionKeys 및 compression을 제공합니다.

```
s3sink = glueContext.getSink(
    path="s3://bucket/folder",
    connection_type="s3",
    updateBehavior="UPDATE_IN_DATABASE",
    partitionKeys=[],
    compression="snappy",
    enableUpdateCatalog=True
)

s3sink.setCatalogInfo(
    catalogDatabase="dbName", catalogTableName="tableName"
)

s3sink.setFormat("parquet", useGlueParquetWriter=true)
s3sink.writeFrame(frameToWrite)
```

- 이 glueparquet 형식 값은 AWS Glue parquet 작성기를 활성화하는 기존 방법입니다.
- updateBehavior를 LOG로 설정하면 DynamicFrame 스키마가 Data Catalog 테이블의 스키마에 정의된 열의 하위 집합과 동일하거나 해당 집합을 포함하고 있는 경우에만 새 파티션이 추가됩니다.
- 분할되지 않은 테이블에는 스키마 업데이트가 지원되지 않습니다("partitionKeys" 옵션 사용 안 함).
- partitionKeys는 ETL 스크립트에 전달된 파라미터와 Data Catalog 테이블 스키마의 partitionKeys 간에 동일한 순서로 동일해야 합니다.
- 이 기능은 현재 업데이트 스키마가 종첩된 테이블 업데이트/생성을 아직 지원하지 않습니다(예: 구조체 내부 배열).

자세한 내용은 [the section called "AWS Glue for Spark"](#) 단원을 참조하십시오.

ETL 작업의 MondoDB 연결 작업

MongoDB 연결을 생성한 다음 해당 연결을 AWS Glue 작업에서 사용하면 됩니다. 자세한 내용은 AWS Glue 프로그래밍 안내서의 [the section called "MongoDB 연결"](#)를 참조하십시오. 연결 url, username 및 password는 MongoDB 연결에 저장됩니다. 다른 옵션은 glueContext.getCatalogSource의 additionalOptions 파라미터를 사용하여 ETL 작업 스크립트에서 지정하면 됩니다. 이외 옵션으로 다음과 같은 항목이 포함됩니다.

- database: (필수 사항) 읽을 MongoDB 데이터베이스입니다.
- collection: (필수 사항) 읽을 MongoDB 컬렉션입니다.

ETL 작업 스크립트 내에 database 및 collection 정보를 배치하면 여러 작업에 같은 조건을 사용할 수 있습니다.

1. MongoDB 데이터 원본에 대해 AWS Glue Data Catalog 연결을 생성합니다. 연결 파라미터에 대한 설명은 ["connectionType": "mongodb"](#)를 참조하세요. 연결은 콘솔, API 또는 CLI를 사용해 생성하면 됩니다.
2. AWS Glue Data Catalog에 데이터베이스를 생성하여 여기에 MongoDB 데이터의 테이블 정의를 저장합니다. 자세한 정보는 [데이터베이스 생성](#)을 참조하세요.
3. MongoDB로 연결되는 연결의 정보를 사용해 MongoDB의 데이터를 크롤링하는 크롤러를 생성합니다. 이 크롤러가 AWS Glue Data Catalog에 테이블을 생성하며 이것이 작업에서 사용하는 MongoDB 데이터베이스 내 테이블을 설명합니다. 자세한 정보는 [크롤러를 사용하여 데이터 카탈로그 채우기](#)을 참조하세요.
4. 사용자 정의 스크립트를 사용하여 작업을 생성합니다. 작업은 콘솔, API 또는 CLI를 사용해 생성하면 됩니다. 자세한 내용은 [AWS Glue의 작업 추가](#)를 참조하십시오.
5. 작업의 데이터 대상을 선택합니다. 데이터 대상을 나타내는 테이블은 Data Catalog에서 정의할 수 있고, 아니면 작업이 실행될 때 대상 테이블을 생성할 수도 있습니다. 작업을 작성하면 대상 위치를 선택합니다. 대상에 연결이 요구되면, 연결도 작업에서 참조됩니다. 작업이 여러 데이터 대상이 필요하다면 나중에 스크립트를 편집하여 추가할 수 있습니다.
6. 작업과 생성된 스크립트의 인수를 제공하여 작업 처리 환경을 사용자 지정합니다.

다음은 Data Catalog에서 정의한 테이블 구조를 기반으로 한 MongoDB 데이터베이스에서 DynamicFrame을 생성하는 예입니다. 이 코드는 additionalOptions를 사용하여 추가적인 데이터 원본 정보를 제공합니다.

Scala

```
val resultFrame: DynamicFrame = glueContext.getCatalogSource(
    database = catalogDB,
    tableName = catalogTable,
    additionalOptions = JsonOptions(Map("database" -> DATABASE_NAME,
    "collection" -> COLLECTION_NAME))
).getDynamicFrame()
```

Python

```
glue_context.create_dynamic_frame_from_catalog(
    database = catalogDB,
    table_name = catalogTable,
```

```
additional_options = {"database": "database_name",
                      "collection": "collection_name"})
```

7. 작업을 실행합니다. 온디맨드여도 되고, 트리거를 통해서도 됩니다.

열 통계를 사용한 쿼리 성능 최적화

추가 데이터 파이프라인을 설정하지 않고도 Parquet, ORC, JSON, ION, CSV 및 XML과 같은 데이터 형식의 AWS Glue Data Catalog 테이블에 대한 열 수준 통계를 계산할 수 있습니다. 열 통계는 열 내 값에 대한 통찰력을 얻어 데이터 프로필을 이해하는 데 도움이 됩니다. 데이터 카탈로그는 최소값, 최대값, 총 null 값, 총 고유 값, 값의 평균 길이, 실제 값의 총 발생 횟수 등과 같은 열 값에 대한 통계 생성을 지원합니다.

Amazon Redshift 및 Amazon Athena와 같은 AWS 분석 서비스에서는 이러한 열 통계를 사용하여 쿼리 실행 계획을 생성하고 쿼리 성능을 향상시키는 최적의 계획을 선택할 수 있습니다.

AWS Glue 콘솔 또는 AWS CLI를 사용하여 열 통계 생성 작업을 실행하도록 구성할 수 있습니다. 프로세스를 시작하면 백그라운드에서 AWS Glue가 Spark 작업을 시작하고 데이터 카탈로그의 AWS Glue 테이블 메타데이터를 업데이트합니다. AWS Glue 콘솔 또는 AWS CLI를 사용하거나 [GetColumnStatisticsForTable](#) API 작업을 직접적으로 호출하여 열 통계를 볼 수 있습니다.

Note

Lake Formation 권한을 사용하여 테이블에 대한 액세스를 제어하는 경우 열 통계 작업에서 맡은 역할을 수행하려면 통계를 생성하기 위한 전체 테이블 액세스 권한이 필요합니다.

주제

- [열 통계 생성을 위한 사전 요구 사항](#)
- [컬럼 통계 생성](#)
- [열 통계 보기](#)
- [컬럼 통계 업데이트](#)
- [컬럼 통계 삭제](#)
- [컬럼 통계 태스크 실행 보기](#)
- [열 통계 작업 실행 중지](#)
- [고려 사항 및 제한](#)

열 통계 생성을 위한 사전 요구 사항

열 통계를 생성하거나 업데이트하기 위해 통계 생성 작업은 사용자를 대신하여 AWS Identity and Access Management(IAM) 역할을 수입합니다. 역할에 부여된 권한에 따라 열 통계 생성 작업은 Amazon S3 데이터 스토어에서 데이터를 읽을 수 있습니다.

Note

Lake Formation에서 관리하는 테이블에 대한 통계를 생성하려면 통계 생성에 사용되는 IAM 역할에 전체 테이블 액세스 권한이 필요합니다.

역할 기반 액세스 제어를 사용하려면 아래 정책에 나열된 권한으로 IAM 역할을 생성한 후 열 통계 생성 작업에 추가해야 합니다.

열 통계 생성을 위한 IAM 역할을 생성하는 방법

1. IAM 역할을 생성하려면 [AWS Glue의 IAM 역할 생성](#)을 참조하십시오.
2. 기존 역할을 업데이트하려면 IAM 콘솔에서 열 통계 생성 프로세스에 사용되는 IAM 역할로 이동합니다.
3. 권한 추가 탭에서 정책 연결을 선택합니다. 새로 열린 브라우저 창에서 AWSGlueServiceRole AWS 관리형 정책을 선택합니다.
4. Amazon S3 데이터 위치에서 데이터를 읽을 수 있는 권한도 포함해야 합니다.

권한 추가 섹션에서 정책 생성을 선택합니다. 새로 열린 브라우저 창에서 역할에 사용할 새 정책을 생성합니다.

5. 정책 생성 페이지에서 JSON 탭을 선택합니다. 다음 JSON 코드를 정책 편집기 필드에 복사합니다.

Note

다음 정책에서는 계정 ID를 유효한 AWS 계정으로 바꾸고, `region`을 테이블의 리전으로, `bucket-name`을 Amazon S3 버킷 이름으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

        "Sid": "S3BucketAccess",
        "Effect": "Allow",
        "Action": [
            "s3:ListBucket",
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::<bucket-name>/*",
            "arn:aws:s3:::<bucket-name>"
        ]
    }
]
}

```

6. (선택 사항) Lake Formation 권한을 사용하여 데이터에 대한 액세스를 제공하는 경우 IAM 역할에 `lakeformation:GetDataAccess` 권한이 필요합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LakeFormationDataAccess",
      "Effect": "Allow",
      "Action": "lakeformation:GetDataAccess",
      "Resource": [
        "*"
      ]
    }
  ]
}

```

Amazon S3 데이터 위치가 Lake Formation에 등록되어 있고 열 통계 생성 작업이 맡는 IAM 역할에 테이블에 대한 `IAM_ALLOWED_PRINCIPALS` 그룹 권한이 부여되지 않은 경우, 이 역할에는 Lake Formation `ALTER`와 테이블에 대한 `DESCRIBE` 권한이 필요합니다. Amazon S3 버킷을 등록하는 데 사용되는 역할에는 테이블에 대한 Lake Formation `INSERT` 및 `DELETE` 권한이 필요합니다.

Amazon S3 데이터 위치가 Lake Formation에 등록되지 않았고 IAM 역할에 테이블에 대한 `IAM_ALLOWED_PRINCIPALS` 그룹 권한이 부여되지 않은 경우, 해당 역할에는 테이블에 대한 Lake Formation `ALTER`, `DESCRIBE`, `INSERT`, `DELETE` 권한이 필요합니다.

7. (선택 사항) 암호화된 Amazon CloudWatch Logs를 기록하는 열 통계 생성 작업에는 키 정책 내 다음 권한이 있어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "CWLogsKmsPermissions",
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents",
      "logs:AssociateKmsKey"
    ],
    "Resource": [
      "arn:aws:logs:<region>:111122223333:log-group:/aws-glue:*"
    ]
  },
  {
    "Sid": "KmsPermissions",
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt",
      "kms:Encrypt"
    ],
    "Resource": [
      "arn:aws:kms:<region>:111122223333:key/"arn of key used for ETL cloudwatch encryption"
    ],
    "Condition": {
      "StringEquals": {
        "kms:ViaService": ["glue.<region>.amazonaws.com"]
      }
    }
  }
  ]
}
```

8. 열 통계를 실행하는 데 사용하는 역할에는 해당 역할에 대한 iam:PassRole 권한이 있어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": [
      "arn:aws:iam::111122223333:role/<columnstats-role-name>"
    ]
  }]
}
```

9. 열 통계 생성을 위한 IAM 역할을 생성할 때 해당 역할에는 서비스가 역할을 수임할 수 있도록 하는 다음과 같은 신뢰 정책도 포함되어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "TrustPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "glue.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
    }
  ]
}
```

컬럼 통계 생성

다음 단계에 따라 AWS Glue 콘솔 또는 AWS CLI를 사용하여 데이터 카탈로그에서 통계 생성을 관리하십시오.

Console

콘솔을 사용하여 열 통계를 생성하는 방법

1. AWS Glue 콘솔(<https://console.aws.amazon.com/glue/>)에 로그인합니다.

2. 데이터 카탈로그 테이블을 선택합니다.
3. 목록에서 테이블을 선택합니다.
4. 작업 메뉴에서 통계 생성을 선택합니다.

테이블 페이지 하단 섹션의 열 통계 탭에서 통계 생성 버튼을 선택해도 됩니다.

5. 통계 생성 페이지에서 다음 옵션을 지정합니다.

Generate statistics

Generate column statistics for the table to improve query performance and potentially save costs. [View pricing](#)

Choose columns

Table (All columns)
Generate statistics for all columns.

Selected columns
Choose the columns to generate statistics.

Row sampling options

We recommend to use all rows to compute accurate column statistics. You can use sampling when the dataset is potentially large and approximate results are acceptable.

All rows
Generate column statistics on entire data.

Sample rows
Generate approximate statistics using sample rows.

IAM role

To generate statistics, the IAM role assumed by the job should have necessary permissions. [Learn more](#)

Choose an existing IAM role

▼
↻
View

[Create new IAM role](#)

▶ **Security configuration - optional**

Enable at-rest encryption with a security configuration.

Cancel
Generate statistics

- 표(모든 열) - 표의 모든 열에 대한 통계를 생성하려면 이 옵션을 선택합니다.
- 선택한 열 - 특정 열에 대한 통계를 생성하려면 이 옵션을 선택합니다. 드롭다운 목록에서 를 선택합니다.
- 모든 행 - 테이블에서 모든 행을 선택하여 정확한 통계를 생성합니다.
- 샘플 행 - 테이블에서 특정 비율의 행만 선택하여 통계를 생성합니다. 기본값은 모든 행입니다. 위쪽 및 아래쪽 화살표를 사용하여 백분율 값을 늘리거나 줄입니다.

i Note

정확한 통계를 계산하려면 표에 모든 행을 포함하는 것이 좋습니다. 대략적인 값이 허용되는 경우에만 샘플 행을 사용하여 열 통계를 생성하십시오.

6. (선택 사항) 다음으로 로그에 대해 저장 중 암호화를 활성화하는 보안 구성을 선택합니다.

7. 통계 생성을 선택하여 프로세스를 실행합니다.

AWS CLI

다음 예제에서는 DatabaseName, TableName, ColumnNameList의 값을 실제 데이터베이스, 테이블 및 열 이름으로 바꿉니다. 계정 ID를 유효한 AWS 계정으로 바꾸고, 역할 이름을 통계를 생성하는 데 사용하는 IAM 역할의 이름으로 바꾸십시오.

```
aws glue start-column-statistics-task-run --input-cli-json file://input.json
{
  "DatabaseName": "<test-db>",
  "TableName": "<test-table>",
  "ColumnNameList": [
    "<column1>",
    "<column2>",
  ],
  "Role": "arn:aws:iam::<123456789012>:role/<Stats-Role>",
  "SampleSize": 10.0
}
```

[StartColumnStatisticsTaskRun](#) 작업을 호출하여 열 통계를 생성해도 됩니다.

열 통계 보기

통계를 성공적으로 생성한 후 Data Catalog는 비용 기반 최적화 프로그램이 쿼리를 실행할 때 최적의 선택을 할 수 있도록 Amazon Athena과 Amazon Redshift에 이 정보를 저장합니다. 통계는 열의 유형에 따라 달라집니다.

AWS Management Console

테이블의 열 통계를 보는 방법

- 열 통계 작업을 실행한 후 테이블 세부정보 페이지의 열 통계 탭에 테이블의 통계가 표시됩니다.

AWS Glue > Tables > pentest_orders_xml

pentest_orders_xml Last updated (UTC) October 25, 2023 at 19:14:47 Version 15 (Current version) Actions

Table overview | Data quality New

Table details | Advanced properties

Name pentest_orders_xml	Description -	Database pentest_db	Classification XML
Location s3://kietduon-column-statistics-table/orders.xml	Connection -	Deprecated -	Last updated October 25, 2023 at 19:14:47
Input format -	Output format -	Serde serialization lib -	

Schema | Partitions | Indexes | Column statistics - new

Column statistics (9) Last updated (UTC) November 6, 2023 at 21:50:40 Stop View all runs Generate statistics

Get an overview of the data profile. We estimate the approximate number of distinct values in a data set with 5% average relative error.

Find columns

Column name	Last updated (UTC)	Average length	Distinct values	Max length	Null values	Max value	Min value	True values	False values
o_clerk	October 25, 2023 at 19:14:	15.00	919	15	-	-	-	-	-
o_comment	October 25, 2023 at 19:14:	88.38	3156	124559	-	-	-	-	-
o_custkey	October 25, 2023 at 19:14:	-	919	-	-	1499	1	-	-
o_order-priority	October 25, 2023 at 19:14:	8.45	5	15	-	-	-	-	-
o_orderdate	October 25, 2023 at 19:14:	10.00	1790	10	-	-	-	-	-
o_orderkey	October 25, 2023 at 19:14:	-	3098	-	-	12451	1	-	-
o_orderstatus	October 25, 2023 at 19:14:	1.00	3	1	-	-	-	-	-
o_ship-priority	October 25, 2023 at 19:14:	-	1	-	-	-	-	-	-
o_totalprice	October 25, 2023 at 19:14:	-	3062	-	-	422359.65	974.04	-	-

다음과 같은 통계를 사용할 수 있습니다.

- 열 이름: 통계 생성에 사용되는 열 이름
- 최종 업데이트: 통계가 생성된 날짜 및 시각
- 평균 길이: 열에 있는 값의 평균 길이
- 고유 값: 열의 고유 값 총 수입니다. 상대 오차가 5% 인 열의 고유 값 수를 추정합니다.
- 최대값: 열에서 최고 값입니다.
- 최소값: 열에서 최저 값입니다.
- 최대 길이: 열에서 최고 값의 길이입니다.
- 열의 null 값의 수입니다.
- 열의 true 값의 수입니다.
- 열의 false 값의 수입니다.

AWS CLI

다음 예제에서는 AWS CLI를 사용하여 열 통계를 검색하는 방법을 보여줍니다.

```
aws glue get-column-statistics-for-table \
  --database-name <test_db> \
```

```
--table-name <test_tble> \  
--column-names <col1>
```

또한 [GetColumnStatisticsForTable](#) API 작업을 사용하여 열 통계를 볼 수 있습니다.

컬럼 통계 업데이트

통계를 최신 상태로 유지하면 쿼리 플래너가 최적의 계획을 선택할 수 있으므로 쿼리 성능이 향상됩니다. 열 통계를 새로 고치려면 AWS Glue 콘솔에서 통계 생성 작업을 명시적으로 실행해야 합니다. 데이터 카탈로그는 통계를 자동으로 새로 고치지 않습니다.

콘솔에서 AWS Glue의 통계 생성 기능을 사용하지 않는 경우 [UpdateColumnStatisticsForTable](#) API 작업 또는 AWS CLI를 사용하여 열 통계를 수동으로 업데이트할 수 있습니다. 다음 예에서는 AWS CLI를 이용하여 열 통계를 업데이트하는 방법을 보여 줍니다.

```
aws glue update-column-statistics-for-table --cli-input-json:
```

```
{  
  "CatalogId": "111122223333",  
  "DatabaseName": "test_db",  
  "TableName": "test_table",  
  "ColumnStatisticsList": [  
    {  
      "ColumnName": "col1",  
      "ColumnType": "Boolean",  
      "AnalyzedTime": "1970-01-01T00:00:00",  
      "StatisticsData": {  
        "Type": "BOOLEAN",  
        "BooleanColumnStatisticsData": {  
          "NumberOfTrues": 5,  
          "NumberOfFalses": 5,  
          "NumberOfNulls": 0  
        }  
      }  
    }  
  ]  
}
```

컬럼 통계 삭제

[DeleteColumnStatisticsForTable](#) API 작업 또는 AWS CLI를 사용하여 열 통계를 삭제할 수 있습니다. 다음 예제에서는 AWS Command Line Interface(AWS CLI)을 사용하여 열 통계를 삭제하는 방법을 보여줍니다.

```
aws glue delete-column-statistics-for-table \
  --database-name test_db \
  --table-name test_table \
  --column-name col1
```

컬럼 통계 태스크 실행 보기

열 통계 작업을 실행한 후 AWS Glue 콘솔, AWS CLI 또는 [GetColumnStatisticsTaskRuns](#) 작업을 사용하여 테이블에 대한 작업 실행 세부 정보를 탐색할 수 있습니다.

Console

열 통계 작업 실행 세부 정보를 보는 방법

1. AWS Glue 콘솔에서 데이터 카탈로그에서 테이블을 선택합니다.
2. 열 통계가 있는 테이블을 선택합니다.
3. 테이블 세부정보 페이지에서 열 통계를 선택합니다.
4. 실행 보기를 선택합니다.

지정된 테이블과 관련된 모든 실행에 대한 정보를 볼 수 있습니다.

AWS Glue > Tables > path1 > All column statistics runs

All runs (1) Last updated (UTC) November 16, 2023 at 00:21:44

View all column statistic runs

< 1 >

Run ID	Status	Start time (UTC)	End time (UTC)	Duration	Column selection	Row sampling
f6a7b304-ad59-49d1-9	Running	November 16, 2023 at 00:21:44	-	-	All columns	100%

AWS CLI

다음 예제에서는 DatabaseName 및 TableName의 값을 실제 데이터베이스 및 테이블 이름으로 바꿉니다.

```
aws glue get-column-statistics-task-runs --input-cli-json file://input.json
{
  "DatabaseName": "<test-db>",
  "TableName": "<test-table>"
}
```

열 통계 작업 실행 중지

AWS Glue 콘솔을 AWS CLI 사용하거나 [StopColumnStatisticsTaskRun](#) 작업을 사용하여 테이블에 대한 열 통계 작업 실행을 중지할 수 있습니다.

Console

열 통계 작업을 중지하려면 작업을 실행하십시오

1. AWS Glue 콘솔에서 데이터 카탈로그에서 테이블을 선택합니다.
2. 열 통계 작업 실행이 진행 중인 테이블을 선택합니다.
3. 테이블 세부정보 페이지에서 열 통계를 선택합니다.
4. 중지를 선택합니다.

실행이 완료되기 전에 작업을 중지하면 테이블에 대한 열 통계가 생성되지 않습니다.

AWS CLI

다음 예제에서는 DatabaseName 및 TableName의 값을 실제 데이터베이스 및 테이블 이름으로 바꿉니다.

```
aws glue stop-column-statistics-task-run --input-cli-json file://input.json
{
  "DatabaseName": "<test-db>",
  "TableName": "<test-table>"
}
```

고려 사항 및 제한

열 통계 생성에는 다음 고려 사항 및 제한 사항이 적용됩니다.

고려 사항

- 샘플링을 사용하여 통계를 생성하면 실행 시간이 줄어들지만 통계가 정확하지 않을 수도 있습니다.
- 각 열 통계 실행에는 전체 데이터 세트를 처리해야 합니다.
- 데이터 카탈로그에는 통계의 서로 다른 버전이 저장되지 않습니다.
- 테이블당 한 번에 하나의 통계 생성 작업만 실행할 수 있습니다.
- Data Catalog에 등록된 고객 AWS KMS 키를 사용하여 테이블을 암호화하는 경우 AWS Glue는 동일한 키를 사용하여 통계를 암호화합니다.

열 통계 작업은 통계 생성을 지원합니다.

- IAM 역할에 전체 테이블 권한(IAM 또는 Lake Formation)이 있는 경우.
- IAM 역할에 Lake Formation 하이브리드 액세스 모드를 사용하는 테이블에 대한 권한이 있는 경우.

열 통계 작업은 다음에 대한 통계 생성을 지원하지 않습니다.

- Lake Formation 셀 기반 액세스 제어 기능이 있는 테이블.
- 트랜잭션 데이터 레이크 - Linux Foundation Delta Lake, Apache Iceberg, Apache Hudi.
- 페더레이션된 데이터베이스의 테이블 - Hive 메타스토어, Amazon Redshift 데이터 공유
- 중첩된 열, 배열, 구조체 데이터 유형.
- 다른 계정에서 공유한 테이블.

데이터 카탈로그 암호화

AWS Key Management Service(AWS KMS)에서 관리하는 암호화 키를 사용하여 AWS Glue Data Catalog에 저장된 메타데이터를 유희 시에 보호할 수 있습니다. 데이터 카탈로그 설정을 사용하여 새 데이터 카탈로그에 대한 데이터 카탈로그 암호화를 활성화할 수 있습니다. 필요에 따라 기존 데이터 카탈로그의 암호화를 활성화하거나 비활성화할 수 있습니다. 활성화하면 AWS Glue는 카탈로그에 기록되는 모든 새 메타데이터를 암호화하지만 기존 메타데이터는 암호화되지 않은 상태로 유지됩니다.

데이터 카탈로그 암호화에 대한 자세한 내용은 [데이터 카탈로그 암호화](#) 섹션을 참조하세요.

Lake Formation을 사용한 데이터 카탈로그 보호

AWS Lake Formation은 AWS에서 안전한 데이터 레이크를 더 쉽게 설정할 수 있게 해주는 서비스입니다. 세분화된 액세스 제어 권한을 정의하여 데이터 레이크를 생성하고 안전하게 관리할 수 있는 중앙

위치를 제공합니다. Lake Formation은 데이터 카탈로그를 사용하여 테이블 정의, 스키마 정보, 데이터 액세스 제어 설정 등과 같은 데이터 레이크에 대한 메타데이터를 저장하고 검색합니다.

Lake Formation에 메타데이터 테이블 또는 데이터베이스의 Amazon S3 데이터 위치를 등록하고 이를 사용하여 데이터 카탈로그 리소스에 대한 메타데이터 수준 권한을 정의할 수 있습니다. 또한 통합 분석 엔진 대신 Lake Formation을 사용하여 Amazon S3에 저장된 기본 데이터에 대한 스토리지 액세스 권한을 관리할 수 있습니다.

자세한 내용은 [AWS Lake Formation이란 무엇인가요?](#)를 참조하세요.

데이터 카탈로그 액세스

AWS Glue Data Catalog를 사용하면 데이터를 검색하고 파악할 수 있습니다. 데이터 카탈로그는 스키마 정의, 데이터 유형, 위치 및 기타 메타데이터를 일관되게 유지 관리할 수 있는 방법을 제공합니다. 다음 방법을 사용하여 데이터 카탈로그에 액세스할 수 있습니다.

- AWS Glue 콘솔 - 웹 기반 사용자 인터페이스인 AWS Glue 콘솔을 통해 데이터 카탈로그에 액세스하고 관리할 수 있습니다. 콘솔을 사용하면 데이터베이스, 테이블 및 관련 메타데이터를 찾고 검색할 수 있을 뿐만 아니라 메타데이터 정의를 생성, 업데이트 및 삭제할 수 있습니다.
- AWS Glue 크롤러 - 크롤러는 데이터 소스를 자동으로 스캔하고 데이터 카탈로그를 메타데이터로 채우는 프로그램입니다. 크롤러를 생성하고 실행하여 Amazon S3, Amazon RDS, Amazon DynamoDB, Amazon CloudWatch와 MySQL, PostgreSQL 등 같은 JDBC 호환 관계형 데이터베이스 뿐만 아니라 Snowflake, Google BigQuery 등과 같은 여러 비 AWS 소스에서 데이터를 검색하고 카탈로그화할 수 있습니다.
- AWS Glue API - AWS Glue API를 사용하여 프로그래밍 방식으로 데이터 카탈로그에 액세스할 수 있습니다. 이러한 API를 사용하면 다른 애플리케이션과 서비스에서 프로그래밍 방식으로 데이터 카탈로그와 상호 작용하여 자동화 및 통합을 수행할 수 있습니다.
- AWS Command Line Interface(AWS CLI) - 명령줄에서 AWS CLI를 사용하여 데이터 카탈로그에 액세스하고 관리할 수 있습니다. CLI는 메타데이터 정의를 생성, 업데이트 및 삭제하는 명령과 메타데이터 정보를 쿼리하고 검색하는 명령을 제공합니다.
- 다른 AWS 서비스와의 통합 - 데이터 카탈로그는 다른 많은 AWS 서비스와 통합되므로 이를 통해 카탈로그에 저장된 메타데이터에 액세스하고 활용할 수 있습니다. 예를 들어 Amazon Athena를 사용하면 데이터 카탈로그의 메타데이터를 사용하여 데이터 소스를 쿼리할 수 있으며, AWS Lake Formation을 사용하면 데이터 카탈로그 리소스에 대한 데이터 액세스 및 거버넌스를 관리할 수 있습니다.

AWS Glue 데이터 카탈로그 모범 사례

이 섹션에서는 AWS Glue Data Catalog를 효과적으로 관리하고 활용하기 위한 모범 사례에 대해 다룹니다. 특히 효율적인 크롤러 사용법, 메타데이터 구성, 보안, 성능 최적화, 자동화, 데이터 거버넌스, 다른 AWS 서비스와의 통합 등의 사례를 강조합니다.

- **효과적인 크롤러 사용** - 정기적으로 크롤러를 실행하여 데이터 소스의 변경 사항을 반영하고 데이터 카탈로그를 최신 상태로 유지합니다. 자주 변경되는 데이터 소스에는 증분 크롤링을 사용하여 성능을 개선할 수 있습니다. 변경 사항이 감지되면 자동으로 새 파티션을 추가하거나 스키마를 업데이트 하도록 크롤러를 구성하세요.
- **메타데이터 테이블 구성 및 이름 지정** - 데이터 카탈로그에서 데이터베이스 및 테이블에 대해 일관된 이름 지정 규칙을 설정합니다. 더 나은 구성을 위해 관련 데이터 소스를 논리적 데이터베이스 또는 폴더로 그룹화합니다. 각 테이블의 목적과 내용을 알 수 있는 설명이 포함된 이름을 사용합니다.
- **효과적인 스키마 관리** - AWS Glue 크롤러의 스키마 추론 기능을 활용합니다. 스키마 변경 사항을 적용하기 전에 검토하고 업데이트하여 다운스트림 애플리케이션이 중단되지 않도록 하세요. 스키마 진화 기능을 사용하면 스키마 변경 사항을 원활하게 처리할 수 있습니다.
- **데이터 카탈로그 보호** - 데이터 카탈로그에서 저장 및 전송 중인 데이터를 암호화할 수 있습니다. 세분화된 액세스 제어 정책을 구현하여 민감한 데이터에 대한 액세스를 제한하세요. 데이터 카탈로그 사용 권한 및 활동 로그를 정기적으로 감사하고 검토하세요.
- **다른 AWS 서비스와의 통합** - 데이터 카탈로그를 Amazon Athena, Redshift Spectrum, AWS Lake Formation 등과 같은 서비스를 위한 중앙 집중식 메타데이터 계층으로 사용하세요. AWS Glue ETL 작업을 활용하면 데이터를 변환하고 다양한 데이터 스토어로 로드하는 동시에 데이터 카탈로그에 메타데이터를 유지 관리할 수 있습니다.
- **성능 모니터링 및 최적화** - Amazon CloudWatch 메트릭을 사용하여 크롤러 및 ETL 작업의 성능을 모니터링하세요. 데이터 카탈로그에서 대규모 데이터셋을 분할하여 쿼리 성능을 개선하세요. 자주 액세스하는 메타데이터에 대한 성능 최적화를 구현하세요.
- **AWS Glue 설명서 및 모범 사례에 대한 최신 정보 확인** - AWS Glue 설명서 및 AWS Glue 리소스에 대한 최신 업데이트, 모범 사례 및 권장 사항을 정기적으로 확인하세요. AWS Glue 웨비나, 워크숍 및 기타 이벤트에 참석하여 전문가로부터 배우고 새로운 특징과 기능에 대한 최신 정보를 받아보세요.

AWS Glue Schema Registry

Note

AWS Glue Console의 다음 리전에서는 AWS Glue Schema Registry가 지원되지 않습니다. 해당 리전은 아시아 태평양(자카르타) 및 중동(UAE)입니다.

AWS Glue Schema Registry는 데이터 스트림 스키마를 중앙에서 검색, 제어 및 발전시킬 수 있는 새로운 기능입니다. 스키마는 데이터 레코드의 구조와 포맷을 정의합니다. AWS Glue Schema Registry를 사용하면 Apache Kafka, [Amazon Managed Streaming for Apache Kafka](#), [Amazon Kinesis Data Streams](#), [Apache Flink](#)용 [Amazon Managed Service for Apache Flink](#), [AWS Lambda](#)와의 편리한 통합을 사용하여 데이터 스트리밍 애플리케이션에서 스키마를 관리하고 적용할 수 있습니다.

AWS Glue 스키마 레지스트리는 AVRO(v1.10.2) 데이터 형식, [Everit 라이브러리](#)를 사용한 JSON 스키마 검증이 포함된 스키마에 대한 [JSON 스키마 포맷](#)을 사용하는 JSON 데이터 형식(사양 Draft-04, Draft-06 및 Draft-07), extensions 또는 groups 지원이 없는 프로토콜 버퍼(Protobuf) 버전 proto2 및 proto3, 기타 데이터 형식과 언어를 제공할 예정인 Java 언어 지원 등을 지원합니다. 지원되는 기능에는 호환성, 메타데이터를 통한 스키마 소싱, 스키마 자동 등록, IAM 호환성, 저장 및 데이터 전송을 줄이기 위한 선택적 ZLIB 압축이 포함됩니다. AWS Glue Schema Registry는 서버리스이며 무료입니다.

생산자와 소비자 간의 데이터 포맷 계약으로 스키마를 사용하면 데이터 거버넌스가 개선되고 데이터 품질이 향상되며 데이터 소비자가 호환 가능한 업스트림 변경 사항에 탄력적으로 대처할 수 있습니다.

Schema Registry를 사용하면 직렬화 및 역직렬화를 위해 서로 다른 시스템이 스키마를 공유할 수 있습니다. 예를 들어 데이터 생산자와 소비자가 있다고 가정합니다. 생산자는 데이터를 게시할 때 스키마를 알고 있습니다. Schema Registry는 Amazon MSK 또는 Apache Kafka와 같은 특정 시스템용 serializer 및 deserializer를 제공합니다.

자세한 내용은 [Schema Registry 작동 방식](#) 단원을 참조하십시오.

주제

- [스키마](#)
- [레지스트리](#)
- [스키마 버전 관리 및 호환성](#)
- [오픈 소스 Serde 라이브러리](#)
- [Schema Registry의 할당량](#)

- [Schema Registry 작동 방식](#)
- [Schema Registry 시작하기](#)
- [AWS Glue Schema Registry와 통합](#)
- [서드 파티 Schema Registry에서 AWS Glue Schema Registry로 마이그레이션](#)

스키마

스키마는 데이터 레코드의 구조와 포맷을 정의합니다. 스키마는 신뢰할 수 있는 데이터 게시, 소비 또는 저장을 위한 버전 지정 사양입니다.

Avro에 대한 이 예제 스키마에서 포맷과 구조는 레이아웃 및 필드 이름으로 정의되고 필드 이름 포맷은 데이터 유형(예: string, int)으로 정의됩니다.

```
{
  "type": "record",
  "namespace": "ABC_Organization",
  "name": "Employee",
  "fields": [
    {
      "name": "Name",
      "type": "string"
    },
    {
      "name": "Age",
      "type": "int"
    },
    {
      "name": "address",
      "type": {
        "type": "record",
        "name": "addressRecord",
        "fields": [
          {
            "name": "street",
            "type": "string"
          },
          {
            "name": "zipcode",
            "type": "int"
          }
        ]
      }
    }
  ]
}
```

```

    }
  }
]
}

```

이 JSON용 JSON 스키마 Draft-07 예에서 포맷은 [JSON 스키마 조직](#)에서 정의합니다.

```

{
  "$id": "https://example.com/person.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Person",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string",
      "description": "The person's first name."
    },
    "lastName": {
      "type": "string",
      "description": "The person's last name."
    },
    "age": {
      "description": "Age in years which must be equal to or greater than zero.",
      "type": "integer",
      "minimum": 0
    }
  }
}

```

Protobuf에 대한 이 예제에서는 [프로토콜 버퍼 언어 버전 2\(proto2\)](#)로 형식을 정의합니다.

```

syntax = "proto2";

package tutorial;

option java_multiple_files = true;
option java_package = "com.example.tutorial.protos";
option java_outer_classname = "AddressBookProtos";

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}

```

```

enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
}

message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2 [default = HOME];
}

repeated PhoneNumber phones = 4;
}

message AddressBook {
    repeated Person people = 1;
}

```

레지스트리

레지스트리는 스키마의 논리적 컨테이너입니다. 레지스트리를 사용하면 스키마를 구성하고 애플리케이션에 대한 액세스 제어를 관리할 수 있습니다. 레지스트리에는 레지스트리 내의 스키마 작업에 대한 다양한 액세스 권한을 구성하고 설정할 수 있는 Amazon 리소스 이름(ARN)이 있습니다.

기본 레지스트리를 사용하거나 필요한 만큼 새 레지스트리를 생성할 수 있습니다.

AWS Glue Schema Registry 계층 구조

- RegistryName: [string]
 - RegistryArn: [AWS ARN]
 - CreatedTime: [timestamp]
 - UpdatedTime: [timestamp]
- SchemaName: [string]
 - SchemaArn: [AWS ARN]
 - DataFormat: [Avro, Json 또는 Protobuf]
 - Compatibility: [예]: BACKWARD, BACKWARD_ALL, FORWARD, FORWARD_ALL, FULL, FULL_ALL, NONE, DISABLED]

- Status: [예: PENDING, AVAILABLE, DELETING]
 - SchemaCheckpoint: [integer]
 - CreatedTime: [timestamp]
 - UpdatedTime: [timestamp]
- SchemaVersion: [string]
 - SchemaVersionNumber: [integer]
 - Status: [예: PENDING, AVAILABLE, DELETING]
 - SchemaDefinition: [string, Value: JSON]
 - CreatedTime: [timestamp]
- SchemaVersionMetadata: [list]
 - MetadataKey: [string]
 - MetadataInfo
 - MetadataValue: [string]
 - CreatedTime: [timestamp]

스키마 버전 관리 및 호환성

각 스키마에는 여러 버전이 있을 수 있습니다. 버전 관리는 스키마에 적용되는 호환성 규칙에 의해 관리됩니다. 새 스키마 버전 등록 요청은 성공하기 전에 Schema Registry에서 이 규칙에 대해 검사합니다.

체크포인트로 표시된 스키마 버전은 스키마의 새 버전 등록 호환성을 결정하는 데 사용됩니다. 스키마가 처음 생성되면 기본 체크포인트가 첫 번째 버전이 됩니다. 스키마가 더 많은 버전으로 발전함에 따라 CLI/SDK를 사용하여 일련의 제약 조건을 준수하는 UpdateSchema API를 사용하는 스키마 버전으로 체크포인트를 변경할 수 있습니다. 콘솔에서 스키마 정의 또는 호환성 모드를 편집하면 기본적으로 체크포인트가 최신 버전으로 변경됩니다.

호환성 모드를 사용하면 시간이 지남에 따라 스키마가 어떻게 발전할 수 있는지 여부를 제어할 수 있습니다. 이러한 모드는 데이터를 생성하고 소비하는 애플리케이션 간의 계약을 형성합니다. 새 버전의 스키마가 레지스트리에 제출되면 스키마 이름에 적용된 호환성 규칙을 사용하여 새 버전을 수락할 수 있는지 여부를 결정합니다. NONE, DISABLED, BACKWARD, BACKWARD_ALL, FORWARD, FORWARD_ALL, FULL, FULL_ALL의 8가지 호환 모드가 있습니다.

Avro 데이터 포맷에서 필드는 선택 사항이거나 필수일 수 있습니다. 선택적 필드는 Type이 null을 포함하는 필드입니다. 필수 필드에는 Type으로 null이 없습니다.

Protobuf 데이터 형식에서 필드가 proto2 구문에서는 선택 사항(반복 포함)이거나 필수일 수 있지만 proto3 구문에서는 모든 필드가 선택 사항입니다(반복 포함). 모든 호환성 규칙은 프로토콜 버퍼 사양에 대한 이해와 [Google 프로토콜 버퍼 설명서](#)의 지침에 따라 결정됩니다.

- NONE: 호환 모드가 적용되지 않습니다. 개발 시나리오에서 또는 스키마에 적용할 호환성 모드를 모르는 경우 이 선택 사항을 사용할 수 있습니다. 추가된 모든 새 버전은 호환성 검사를 거치지 않고 수락됩니다.
- DISABLED: 이 호환성 선택 항목은 특정 스키마에 대한 버전 관리를 방지합니다. 새 버전을 추가할 수 없습니다.
- BACKWARD: 이 호환성 선택 항목은 소비자가 현재 및 이전 스키마 버전을 모두 읽을 수 있도록 하므로 권장됩니다. 이 선택 항목을 사용하여 필드를 삭제하거나 선택적 필드를 추가할 때 이전 스키마 버전과의 호환성을 확인할 수 있습니다. BACKWARD의 일반적인 사용 사례는 애플리케이션이 가장 최근 스키마에 대해 생성된 경우입니다.

AVRO

예를 들어 이름(필수), 성(필수), 이메일(필수) 및 전화번호(선택 사항)로 정의된 스키마가 있다고 가정합니다.

다음 스키마 버전에서 필수 이메일 필드를 제거하면 성공적으로 등록됩니다. BACKWARD 호환성을 위해서는 소비자가 현재 및 이전 스키마 버전을 읽을 수 있어야 합니다. 이전 메시지의 추가 이메일 필드가 무시되므로 소비자는 새 스키마를 읽을 수 있습니다.

필수 필드를 추가하는 제안된 새 스키마 버전이 있는 경우(예: 우편 번호) 이는 BACKWARD 호환성으로 성공적으로 등록되지 않습니다. 새 버전의 소비자는 필수 우편 번호 필드가 누락되어 스키마 변경 전에 이전 메시지를 읽을 수 없습니다. 그러나 새 스키마에서 우편 번호 필드가 선택 사항으로 설정된 경우 소비자가 선택적 우편 번호 필드 없이 이전 스키마를 읽을 수 있으므로 제안된 버전이 성공적으로 등록됩니다.

JSON

예를 들어 이름(선택 사항), 성(선택 사항), 이메일(선택 사항) 및 전화 번호(선택 사항)로 정의된 스키마 버전이 있다고 가정합니다.

다음 스키마 버전이 선택적 전화번호 속성을 추가하는 경우 원래 스키마 버전이 `additionalProperties` 필드를 `false`로 설정하여 추가 속성을 허용하지 않는 한 성공적으로 등록

됩니다. BACKWARD 호환성을 위해서는 소비자가 현재 및 이전 스키마 버전을 읽을 수 있어야 합니다. 소비자는 전화번호 속성이 존재하지 않는 원래 스키마로 생성된 데이터를 읽을 수 있습니다.

선택적 전화 번호 속성을 추가하는 제안된 새 스키마 버전이 있는 경우 원래 스키마 버전이 `additionalProperties` 필드를 `true`로 설정하면(즉, 추가 속성을 허용할 때) BACKWARD 호환성으로 성공적으로 등록되지 않습니다. 새 버전의 소비자는 다른 유형(예: 숫자 대신 문자열)의 전화 번호 속성이 있는 데이터를 읽을 수 없으므로 스키마가 변경되기 전에 이전 메시지를 읽을 수 없습니다.

PROTOBUF

예를 들어 `proto2` 구문에서 `first name`(필수), `last name`(필수), `email`(필수), `phone number`(선택) 필드를 포함하는 `Message Person`으로 정의된 스키마 버전이 있다고 가정합니다.

AVRO 시나리오에서처럼 다음 스키마 버전에서 필수 `email` 필드를 제거하면 성공적으로 등록됩니다. BACKWARD 호환성을 위해서는 소비자가 현재 및 이전 스키마 버전을 읽을 수 있어야 합니다. 이전 메시지의 추가 `email` 필드가 무시되므로 소비자는 새 스키마를 읽을 수 있습니다.

필수 항목(예: `zip code`)을 추가하는 제안된 새 스키마 버전이 있는 경우, 이는 이전 버전과의 호환성으로 성공적으로 등록되지 않습니다. 새 버전의 소비자는 필수 `zip code` 필드가 누락되어 스키마 변경 전에 이전 메시지를 읽을 수 없습니다. 그러나 새 스키마에서 `zip code` 필드가 선택 사항으로 설정된 경우 소비자가 선택적 `zip code` 필드 없이 이전 스키마를 읽을 수 있으므로 제안된 버전이 성공적으로 등록됩니다.

gRPC 사용 사례의 경우 새 RPC 서비스 또는 RPC 메서드 추가는 이전 버전과 호환되는 변경 사항입니다. 예를 들어 RPC 서비스 `MyService`에서 두 개의 RPC 메서드 `Foo` 및 `Bar`를 사용하여 정의한 스키마 버전이 있다고 가정합니다.

다음 스키마 버전에서 `Baz`라는 새 RPC 메서드를 추가하는 경우 성공적으로 등록됩니다. 새로 추가된 RPC 메서드 `Baz`가 선택 사항이므로 소비자는 이전 버전과의 호환성에 따라 원래 스키마로 생성된 데이터를 읽을 수 있습니다.

기존 RPC 메서드 `Foo`를 제거하는 제안된 새 스키마 버전이 있는 경우, 이는 이전 버전과의 호환성으로 성공적으로 등록되지 않습니다. 새 버전의 소비자는 gRPC 애플리케이션에서 RPC 메서드 `Foo`가 없는 데이터를 이해하거나 읽을 수 없으므로 스키마 변경 전에 이전 메시지를 읽을 수 없습니다.

- **BACKWARD_ALL**: 이 호환성 선택을 통해 소비자는 현재 및 모든 이전 스키마 버전을 모두 읽을 수 있습니다. 이 선택 항목을 사용하여 필드를 삭제하거나 선택적 필드를 추가할 때 모든 이전 스키마 버전과의 호환성을 확인할 수 있습니다.

- FORWARD: 이 호환성 선택을 통해 소비자는 현재 및 후속 스키마 버전을 모두 읽을 수 있지만 반드시 이후 버전은 아닙니다. 이 선택 항목 사용하여 필드를 추가하거나 선택적 필드를 삭제할 때 마지막 스키마 버전과의 호환성을 확인할 수 있습니다. FORWARD의 일반적인 사용 사례는 애플리케이션이 이전 스키마에 대해 생성되었고 더 최신 스키마를 처리할 수 있어야 하는 경우입니다.

AVRO

예를 들어 이름(필수), 성(필수), 이메일(선택 사항)로 정의된 스키마 버전이 있다고 가정합니다.

필수 항목(예: 전화번호)을 추가하는 새 스키마 버전이 있는 경우 성공적으로 등록됩니다.

FORWARD 호환성을 위해서는 소비자가 이전 버전을 사용하여 새 스키마로 생성된 데이터를 읽을 수 있어야 합니다.

필수 이름 필드를 삭제하는 제안된 스키마 버전이 있는 경우 FORWARD 호환성으로 성공적으로 등록되지 않습니다. 이전 버전의 소비자는 필수 이름 필드가 누락되어 제안된 스키마를 읽을 수 없습니다. 그러나 이름 필드가 원래 선택 사항인 경우 제안된 새 스키마는 선택 사항인 이름 필드가 없는 새 스키마를 기반으로 소비자가 데이터를 읽을 수 있으므로 성공적으로 등록됩니다.

JSON

예를 들어 이름(선택 사항), 성(선택 사항), 이메일(선택 사항) 및 전화 번호(선택 사항)로 정의된 스키마 버전이 있다고 가정합니다.

선택적 전화 번호 속성을 제거하는 새 스키마 버전이 있는 경우 새 스키마 버전에서 `additionalProperties` 필드를 `false`로 설정하여 추가 속성을 허용하지 않는 한 성공적으로 등록됩니다. FORWARD 호환성을 위해서는 소비자가 이전 버전을 사용하여 새 스키마로 생성된 데이터를 읽을 수 있어야 합니다.

선택적 전화번호 속성을 삭제하는 제안된 스키마 버전이 있는 경우 새 스키마 버전이 `additionalProperties` 필드를 `true`로 설정하면 즉, 추가 속성을 허용할 때 FORWARD 호환성으로 성공적으로 등록되지 않습니다. 이전 버전의 소비자는 다른 유형의 전화번호 속성(예: 숫자 대신 문자열)을 가질 수 있으므로 제안된 스키마를 읽을 수 없습니다.

PROTOBUF

예를 들어 `proto2` 구문에서 `first name`(필수), `last name`(필수), `email`(선택 사항) 필드를 포함하는 `Message Person`으로 정의된 스키마 버전이 있다고 가정합니다.

AVRO 시나리오처럼 필수 항목(예: phone number)을 추가하는 새 스키마 버전이 있는 경우 성공적으로 등록됩니다. FORWARD 호환성을 위해서는 소비자가 이전 버전을 사용하여 새 스키마로 생성된 데이터를 읽을 수 있어야 합니다.

필수 first name 필드를 삭제하는 제안된 스키마 버전이 있는 경우 FORWARD 호환성으로 성공적으로 등록되지 않습니다. 이전 버전의 소비자는 필수 first name 필드가 누락되어 제안된 스키마를 읽을 수 없습니다. 그러나 first name 필드가 원래 선택 사항인 경우 제안된 새 스키마는 선택 사항인 first name 필드가 없는 새 스키마를 기반으로 소비자가 데이터를 읽을 수 있으므로 성공적으로 등록됩니다.

gRPC 사용 사례의 경우 RPC 서비스 또는 RPC 메서드 제거는 다음 버전과 호환되는 변경 사항입니다. 예를 들어 RPC 서비스 MyService에서 두 개의 RPC 메서드 Foo 및 Bar를 사용하여 정의한 스키마 버전이 있다고 가정합니다.

다음 스키마 버전에서 Foo라는 기존 RPC 메서드를 삭제하는 경우 소비자가 이전 버전을 사용하여 새 스키마로 생성된 데이터를 읽을 수 있으므로 FORWARD 호환성에 따라 성공적으로 등록됩니다. Baz라는 RPC 메서드를 추가하는 제안된 새 스키마 버전이 있는 경우 FORWARD 호환성으로 성공적으로 등록되지 않습니다. 이전 버전의 소비자는 Baz라는 RPC 메서드가 누락되어 제안된 스키마를 읽을 수 없습니다.

- FORWARD_ALL: 이 호환성 선택을 통해 소비자는 새로 등록된 스키마의 생산자가 작성한 데이터를 읽을 수 있습니다. 필드를 추가하거나 선택적 필드를 삭제하고 모든 이전 스키마 버전과의 호환성을 확인해야 할 때 이 선택 사항을 사용할 수 있습니다.
- FULL: 이 호환성 선택을 통해 소비자는 이전 또는 다음 버전의 스키마를 사용하여 생산자가 작성한 데이터를 읽을 수 있지만 이전 또는 이후 버전은 아닙니다. 이 선택 항목 사용하여 선택적 필드를 추가하거나 제거할 때 마지막 스키마 버전과의 호환성을 확인할 수 있습니다.
- FULL_ALL: 이 호환성 선택을 통해 소비자는 모든 이전 스키마 버전을 사용하여 생산자가 작성한 데이터를 읽을 수 있습니다. 이 선택 항목을 사용하여 선택적 필드를 추가하거나 제거할 때 모든 이전 스키마 버전과의 호환성을 확인할 수 있습니다.

오픈 소스 Serde 라이브러리

AWS는 데이터 직렬화 및 역직렬화를 위한 프레임워크로 오픈 소스 Serde 라이브러리를 제공합니다. 이러한 라이브러리의 오픈 소스 설계를 통해 일반적인 오픈 소스 애플리케이션 및 프레임워크가 프로젝트에서 이러한 라이브러리를 지원할 수 있습니다.

Serde 라이브러리의 작동 방식에 대한 자세한 내용은 [Schema Registry 작동 방식](#) 섹션을 참조하세요.

Schema Registry의 할당량

AWS에서 제한이라고도 하는 할당량은 AWS 계정의 리소스, 작업, 항목의 최댓값입니다. 다음은 AWS Glue의 Schema Registry에 대한 소프트 제한입니다.

스키마 버전 메타데이터 키-값 페어

AWS 리전별로 SchemaVersion당 최대 10개의 키-값 페어가 있을 수 있습니다.

[QuerySchemaVersionMetadata 액션 \(Python: 쿼리 스키마 버전 메타데이터\)](#) 또는 [PutSchemaVersionMetadata 액션 \(Python: put_schema_version 메타데이터\)](#) API를 사용하여 키-값 메타데이터 페어를 보거나 설정할 수 있습니다.

다음은 AWS Glue의 Schema Registry에 대한 하드 제한입니다.

레지스트리

이 계정의 경우 AWS 리전당 최대 100개의 레지스트리를 보유할 수 있습니다.

SchemaVersion

이 계정에 대해 AWS 리전당 최대 10,000개의 스키마 버전을 보유할 수 있습니다.

각 새 스키마는 새 스키마 버전을 생성하므로 각 스키마에 버전이 하나만 있는 경우 이론상 지역별로 계정당 최대 10,000개의 스키마를 보유할 수 있습니다.

스키마 페이로드

스키마 페이로드의 크기 제한은 170KB입니다.

Schema Registry 작동 방식

이 섹션에서는 Schema Registry의 직렬화 및 역직렬화 프로세스의 작동 방식에 대해 설명합니다.

1. 스키마 등록: 스키마가 레지스트리에 이미 존재하지 않는 경우 대상 이름과 동일한 스키마 이름(예: test_topic, test_stream, prod_firehose)으로 스키마를 등록하거나 생산자가 스키마에 대한 사용자 정의 이름을 제공할 수 있습니다. 생산자는 또한 소스: msk_kafka_topic_A와 같은 메타데이터로 스키마에 키-값 페어를 추가하거나 스키마 생성 시 스키마에 AWS 태그를 적용할 수 있습니다. 스키마가 등록되면 Schema Registry는 스키마 버전 ID를 serializer에 반환합니다. 스키마가 존재하지만 serializer가 존재하지 않는 새 버전을 사용하는 경우 Schema Registry는 스키마 참조를 호환성 규칙으로 확인하여 새 버전을 새 버전으로 등록하기 전에 새 버전이 호환되는지 확인합니다.

스키마를 등록하는 방법에는 수동 등록과 자동 등록의 두 가지가 있습니다. AWS Glue 콘솔이나 CLI/SDK를 통해 수동으로 스키마를 등록할 수 있습니다.

serializer 설정에서 자동 등록이 켜져 있으면 스키마의 자동 등록이 수행됩니다. 생산자 구성에 REGISTRY_NAME이 제공되지 않으면 자동 등록은 기본 레지스트리(default-registry) 아래에 새 스키마 버전을 등록합니다. 자동 등록 속성 지정에 대한 자세한 내용은 [SerDe 라이브러리 설치](#) 섹션을 참조하세요.

2. Serializer는 스키마에 대해 데이터 레코드를 검증합니다. 데이터를 생성하는 애플리케이션이 해당 스키마를 등록한 경우 Schema Registry serializer는 애플리케이션이 생성하는 레코드가 등록된 스키마와 일치하는 필드 및 데이터 유형으로 구성되었는지 확인합니다. 레코드의 스키마가 등록된 스키마와 일치하지 않는 경우 serializer는 예외를 반환하고 애플리케이션은 레코드를 대상으로 전달하지 못합니다.

스키마가 없고 생산자 구성을 통해 스키마 이름이 제공되지 않은 경우 스키마는 주제 이름(Apache Kafka 또는 Amazon MSK의 경우) 또는 스트림 이름(Kinesis Data Streams의 경우)과 동일한 이름으로 생성됩니다.

모든 레코드에는 스키마 정의와 데이터가 있습니다. 스키마 정의는 Schema Registry의 기존 스키마 및 버전에 대해 쿼리됩니다.

기본적으로 생산자는 등록된 스키마의 스키마 정의 및 스키마 버전 ID를 캐시합니다. 레코드의 스키마 버전 정의가 캐시에서 사용 가능한 것과 일치하지 않는 경우 생산자는 Schema Registry를 사용하여 스키마를 검증하려고 시도합니다. 스키마 버전이 유효하면 해당 버전 ID와 정의가 생산자에서 로컬로 캐시됩니다.

[SerDe 라이브러리 설치](#)의 3단계에서 선택적 생산자 속성 내에서 기본 캐시 기간(24시간)을 조정할 수 있습니다.

3. 레코드 직렬화 및 전달: 레코드가 스키마를 준수하는 경우 serializer는 각 레코드를 스키마 버전 ID로 장식하고 선택한 데이터 포맷(AVRO, JSON, Protobuf 또는 다른 포맷 곧 제공 예정)을 기반으로 레코드를 직렬화하고 레코드를 압축(선택적 생산자 구성), 목적지로 전달합니다.
4. 소비자는 데이터를 역직렬화합니다. 이 데이터를 읽는 소비자는 레코드 페이로드에서 스키마 버전 ID를 구문 분석하는 Schema Registry deserializer 라이브러리를 사용합니다.
5. Deserializer는 Schema Registry에서 스키마를 요청할 수 있습니다. Deserializer가 특정 스키마 버전 ID를 가진 레코드를 처음 본 경우에는 스키마 버전 ID를 사용하여 deserializer가 Schema Registry에서 스키마를 요청하고 소비자에서 로컬로 스키마를 캐시합니다. Schema Registry가 레코드를 역직렬화할 수 없는 경우 소비자는 레코드에서 데이터를 기록하고 계속 진행하거나 애플리케이션을 중지할 수 있습니다.

6. Deserializer는 스키마를 사용하여 레코드를 역직렬화합니다. Deserializer가 Schema Registry에서 스키마 버전 ID를 검색할 때 deserializer는 레코드를 압축 해제하고(생산자가 보낸 레코드가 압축된 경우) 스키마를 사용하여 레코드를 역직렬화합니다. 이제 애플리케이션이 레코드를 처리합니다.

Note

암호화: 클라이언트는 HTTPS를 통한 TLS 암호화를 사용하여 전송 중 데이터를 암호화하는 API 호출을 통해 Schema Registry와 통신합니다. Schema Registry에 저장된 스키마는 항상 서비스 관리형 AWS Key Management Service(AWS KMS) 키를 사용하여 저장 시 암호화됩니다.

Note

사용자 권한 부여: 스키마 레지스트리는 자격 증명 기반 IAM 정책을 지원합니다.

Schema Registry 시작하기

다음 섹션은 Schema Registry를 설정하고 사용하는 방법에 대한 개요 및 설명입니다. Schema Registry 개념 및 구성 요소에 대한 자세한 내용은 [AWS Glue Schema Registry](#) 섹션을 참조하세요.

주제

- [SerDe 라이브러리 설치](#)
- [AWS Glue Schema Registry API에 AWS CLI 사용](#)
- [레지스트리 생성](#)
- [JSON에 대한 특정 레코드\(JAVA POJO\) 처리](#)
- [스키마 생성](#)
- [스키마 또는 레지스트리 업데이트](#)
- [스키마 또는 레지스트리 삭제](#)
- [Serializer에 대한 IAM 예](#)
- [Deserializer에 대한 IAM 예](#)
- [AWS PrivateLink를 사용한 프라이빗 연결](#)
- [Amazon CloudWatch 지표에 액세스](#)

- [Schema Registry용 샘플 AWS CloudFormation 템플릿](#)

SerDe 라이브러리 설치

Note

전제 조건: 다음 단계를 완료하기 전에 Amazon Managed Streaming for Apache Kafka(Amazon MSK) 또는 Apache Kafka 클러스터가 실행 중이어야 합니다. 생산자와 소비자는 Java 8 이상에서 실행 중이어야 합니다.

SerDe 라이브러리는 데이터 직렬화 및 역직렬화를 위한 프레임워크를 제공합니다.

데이터를 생성하는 애플리케이션("serializer"로 통칭)을 위한 오픈 소스 serializer를 설치합니다. Serializer는 직렬화, 압축 및 Schema Registry와의 상호 작용을 처리합니다. Serializer는 Amazon MSK와 같은 Schema Registry 호환 대상에 기록되는 레코드에서 스키마를 자동으로 추출합니다. 마찬가지로 데이터를 사용하는 애플리케이션에 오픈 소스 deserializer를 설치합니다.

생산자와 소비자에 라이브러리를 설치하려면

1. 생산자와 소비자의 pom.xml 파일 내에서 아래 코드를 통해 이 종속성을 추가합니다.

```
<dependency>
  <groupId>software.amazon.glue</groupId>
  <artifactId>schema-registry-serde</artifactId>
  <version>1.1.5</version>
</dependency>
```

또는 [AWS Glue Schema Registry Github 리포지토리](#)를 복제할 수 있습니다.

2. 다음 필수 속성으로 생산자를 설정합니다.

```
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
  StringSerializer.class.getName()); // Can replace StringSerializer.class.getName()
with any other key serializer that you may use
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
  GlueSchemaRegistryKafkaSerializer.class.getName());
props.put(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2");
properties.put(AWSSchemaRegistryConstants.DATA_FORMAT, "JSON"); // OR "AVRO"
```

기존 스키마가 없으면 자동 등록을 설정해야 합니다(다음 단계). 적용하려는 스키마가 있는 경우 "my-schema"를 스키마 이름으로 바꿉니다. 또한 스키마 자동 등록이 꺼져 있는 경우 "registry-name"을 제공해야 합니다. 스키마가 "default-registry" 아래에 생성된 경우 레지스트리 이름을 생략할 수 있습니다.

3. (선택 사항) 이러한 선택적 생산자 속성을 설정합니다. 자세한 속성 설명은 [ReadMe 파일](#)을 참조하세요.

```
props.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, "true"); // If
not passed, uses "false"
props.put(AWSSchemaRegistryConstants.SCHEMA_NAME, "my-schema"); // If not passed,
uses transport name (topic name in case of Kafka, or stream name in case of Kinesis
Data Streams)
props.put(AWSSchemaRegistryConstants.REGISTRY_NAME, "my-registry"); // If not passed,
uses "default-registry"
props.put(AWSSchemaRegistryConstants.CACHE_TIME_TO_LIVE_MILLIS, "86400000"); // If
not passed, uses 86400000 (24 Hours)
props.put(AWSSchemaRegistryConstants.CACHE_SIZE, "10"); // default value is 200
props.put(AWSSchemaRegistryConstants.COMPATIBILITY_SETTING, Compatibility.FULL); //
Pass a compatibility mode. If not passed, uses Compatibility.BACKWARD
props.put(AWSSchemaRegistryConstants.DESCRPTION, "This registry is used for several
purposes."); // If not passed, constructs a description
props.put(AWSSchemaRegistryConstants.COMPRESSION_TYPE,
AWSSchemaRegistryConstants.COMPRESSION.ZLIB); // If not passed, records are sent
uncompressed
```

자동 등록은 기본 레지스트리("default-registry") 아래에 스키마 버전을 등록합니다. 이전 단계에서 SCHEMA_NAME을 지정하지 않으면 주제 이름이 SCHEMA_NAME으로 유추됩니다.

호환성 모드에 대한 자세한 내용은 [스키마 버전 관리 및 호환성](#) 섹션을 참조하세요.

4. 다음 필수 속성으로 소비자를 설정합니다.

```
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
GlueSchemaRegistryKafkaDeserializer.class.getName());
props.put(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2"); // Pass an AWS ##
props.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
AvroRecordType.GENERIC_RECORD.getName()); // Only required for AVRO data format
```

5. (선택 사항) 이러한 선택적 소비자 속성을 설정합니다. 자세한 속성 설명은 [ReadMe 파일](#)을 참조하세요.

```
properties.put(AWSSchemaRegistryConstants.CACHE_TIME_TO_LIVE_MILLIS, "86400000"); //
  If not passed, uses 86400000
props.put(AWSSchemaRegistryConstants.CACHE_SIZE, "10"); // default value is 200
props.put(AWSSchemaRegistryConstants.SECONDARY_DESERIALIZER,
  "com.amazonaws.services.schemaregistry.deserializers.external.ThirdPartyDeserializer"); //
  For migration fall back scenario
```

AWS Glue Schema Registry API에 AWS CLI 사용

AWS Glue Schema Registry API에 AWS CLI를 사용하려면 AWS CLI를 최신 버전으로 업데이트해야 합니다.

레지스트리 생성

기본 레지스트리를 사용하거나 AWS Glue API 또는 AWS Glue 콘솔을 사용하여 필요한 만큼 새 레지스트리를 생성할 수 있습니다.

AWS Glue API

다음 단계를 사용하여 AWS Glue API로 이 태스크를 수행할 수 있습니다.

새 레지스트리를 추가하려면 [CreateRegistry 액션 \(Python: 레지스트리 만들기\)](#) API를 사용합니다. 생성할 레지스트리의 이름으로 RegistryName을 지정합니다. 최대 길이는 255이며 문자, 숫자, 하이픈, 밑줄, 달러 기호 또는 해시 표시만 포함합니다.

[URI 주소 여러 줄 문자열 패턴](#)과 일치하는 2048바이트 이하의 문자열로 Description을 지정합니다.

선택적으로 키-값 페어의 맵 배열로 레지스트리에 대해 하나 이상의 Tags를 지정합니다.

```
aws glue create-registry --registry-name registryName1 --description description
```

레지스트리가 생성되면 API 응답의 RegistryArn에서 볼 수 있는 Amazon 리소스 이름(ARN)이 할당됩니다. 이제 레지스트리를 생성했으므로 해당 레지스트리에 대해 하나 이상의 스키마를 생성합니다.

AWS Glue 콘솔

AWS Glue 콘솔에서 새 레지스트리를 추가하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [스키마 레지스트리(Schema registries)]를 선택합니다.
3. [레지스트리 추가(Add registry)]를 선택합니다.
4. 문자, 숫자, 하이픈 또는 밑줄로 구성된 레지스트리의 [레지스트리 이름(Registry name)]을 입력합니다. 이 이름은 변경할 수 없습니다.
5. 레지스트리에 대한 [설명(Description)](선택 사항)을 입력합니다.
6. 선택적으로 레지스트리에 하나 이상의 태그를 적용합니다. [새 태그 추가(Add new tag)]를 선택하고 [태그 키(Tag key)]와 [태그 값(Tag value)](선택 사항)을 지정합니다.
7. [레지스트리 추가(Add registry)]를 선택합니다.

AWS Glue

Data catalog

Databases

Tables

Connections

Crawlers

Classifiers

Schema registries

Schemas

Settings

ETL

AWS Glue Studio

New

Workflows

Jobs

ML Transforms

Triggers

Dev endpoints

Notebooks

Schema registries > Add registry

Add a new schema registry

Add a schema registry to store one or multiple new related schemas.

Registry name
Name can't be changed post creation.

Only letters (A-Z), numbers (0-9), hyphens (-), underscores (_), dollar signs (\$), or hash marks (#) allowed. 255 characters maximum.

Description - optional

2048 characters maximum.

Registry tags - optional
No tags defined.

You can add up to 50 more tags.

레지스트리가 생성되면 [스키마 레지스트리(Schema registries)]의 목록에서 레지스트리를 선택하여 볼 수 있는 Amazon 리소스 이름(ARN)이 할당됩니다. 이제 레지스트리를 생성했으므로 해당 레지스트리에 대해 하나 이상의 스키마를 생성합니다.

JSON에 대한 특정 레코드(JAVA POJO) 처리

POJO(Plain Old Java Object)를 사용하고 객체를 레코드로 전달할 수 있습니다. 이는 AVRO에서 특정 레코드의 개념과 유사합니다. [mbknor-jackson-jsonschema](#)는 전달된 POJO에 대한 JSON 스키마를 생성할 수 있습니다. 이 라이브러리는 JSON 스키마에 추가 정보를 삽입할 수도 있습니다.

AWS Glue Schema Registry 라이브러리는 스키마에 삽입된 "className" 필드를 사용하여 완전히 분류된 클래스 이름을 제공합니다. "className" 필드는 deserializer에서 해당 클래스의 객체로 역직렬화하는 데 사용됩니다.

Example class :

```
@JsonSchemaDescription("This is a car")
@JsonSchemaTitle("Simple Car Schema")
@Builder
@AllArgsConstructor
@EqualsAndHashCode
// Fully qualified class name to be added to an additionally injected property
// called className for deserializer to determine which class to deserialize
// the bytes into
@JsonSchemaInject(
    strings = {@JsonSchemaString(path = "className",
        value =
            "com.amazonaws.services.schemaregistry.integrationtests.generators.Car")})
)
// List of annotations to help infer JSON Schema are defined by https://github.com/
mbknor/mbknor-jackson-jsonSchema
public class Car {
    @JsonProperty(required = true)
    private String make;

    @JsonProperty(required = true)
    private String model;

    @JsonSchemaDefault("true")
    @JsonProperty
    public boolean used;

    @JsonSchemaInject(ints = {@JsonSchemaInt(path = "multipleOf", value = 1000)})
```

```

    @Max(200000)
    @JsonProperty
    private int miles;

    @Min(2000)
    @JsonProperty
    private int year;

    @JsonProperty
    private Date purchaseDate;

    @JsonProperty
    @JsonFormat(shape = JsonFormat.Shape.NUMBER)
    private Date listedDate;

    @JsonProperty
    private String[] owners;

    @JsonProperty
    private Collection<Float> serviceChecks;

    // Empty constructor is required by Jackson to deserialize bytes
    // into an Object of this class
    public Car() {}
}

```

스키마 생성

AWS Glue API 또는 AWS Glue 콘솔을 사용하여 스키마를 생성할 수 있습니다.

AWS Glue API

다음 단계를 사용하여 AWS Glue API로 이 태스크를 수행할 수 있습니다.

새 스키마를 추가하려면 [CreateSchema 액션 \(Python: 생성_스키마\)](#) API를 사용합니다.

스키마에 대한 레지스트리를 나타내려면 RegistryId 구조를 지정합니다. 또는 기본 레지스트리를 사용하려면 RegistryId를 생략합니다.

문자, 숫자, 하이픈, 밑줄로 구성된 SchemaName을 지정하고 DataFormat은 **AVRO** 또는 **JSON**으로 지정합니다. DataFormat은 스키마에 한 번 설정되면 변경할 수 없습니다.

Compatibility 모드를 지정합니다.

- [뒤로(권장)(Backward (recommended))] - 소비자가 현재 버전과 이전 버전을 모두 읽을 수 있습니다.
- [모두 뒤로(Backward all)] - 소비자가 현재 및 모든 이전 버전을 읽을 수 있습니다.
- [앞으로(Forward)] - 소비자가 현재 및 후속 버전을 모두 읽을 수 있습니다.
- [모두 앞으로(Forward all)] - 소비자가 현재 버전과 모든 후속 버전을 읽을 수 있습니다.
- [전체(Full)] - 뒤로 및 앞으로의 조합입니다.
- [모두 전체(Full all)] - 모두 뒤로 및 모두 앞으로의 조합입니다.
- [없음(None)] - 호환성 확인이 수행되지 않습니다.
- [사용 중지됨(Disabled)] - 이 스키마에 대한 버전 관리를 방지합니다.

선택적으로 스키마에 Tags를 지정합니다.

Avro, JSON 또는 Protobuf 데이터 포맷으로 스키마를 정의하려면 SchemaDefinition을(를) 지정합니다. 예제를 참조하세요.

Avro 데이터 포맷의 경우:

```
aws glue create-schema --registry-id RegistryName="registryName1" --schema-name
testschema --compatibility NONE --data-format AVRO --schema-definition "{\"type\":
\"record\", \"name\": \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"},
{\"name\": \"f2\", \"type\": \"string\"} ]}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName1" --schema-name testschema --compatibility
NONE --data-format AVRO --schema-definition "{\"type\": \"record\", \"name\": \"r1\",
\"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type\":
\"string\"} ]}"
```

JSON 데이터 포맷의 경우:

```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name
testSchemaJson --compatibility NONE --data-format JSON --schema-definition "{\"$schema
\": \"http://json-schema.org/draft-07/schema#\", \"type\": \"object\", \"properties\":
{\"f1\": {\"type\": \"string\"}}}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-
east-2:901234567890:registry/registryName" --schema-name testSchemaJson --compatibility
```

```
NONE --data-format JSON --schema-definition "{\"$schema\": \"http://json-schema.org/draft-07/schema#\", \"type\": \"object\", \"properties\": {\"f1\": {\"type\": \"string\"}}}"
```

Protobuf 데이터 형식의 경우:

```
aws glue create-schema --registry-id RegistryName="registryName" --schema-name testSchemaProtobuf --compatibility NONE --data-format PROTOBUF --schema-definition "syntax = \"proto2\"; package org.test; message Basic { optional int32 basic = 1;}"
```

```
aws glue create-schema --registry-id RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName" --schema-name testSchemaProtobuf --compatibility NONE --data-format PROTOBUF --schema-definition "syntax = \"proto2\"; package org.test; message Basic { optional int32 basic = 1;}"
```

AWS Glue 콘솔

AWS Glue 콘솔을 사용하여 새 스키마를 추가하려면

1. AWS 관리 콘솔에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [스키마(Schemas)]를 선택합니다.
3. [스키마 추가(Add schema)]를 선택합니다.
4. 문자, 숫자, 하이픈, 밑줄, 달러 기호 또는 해시마크로 구성된 [스키마 이름(Schema name)]을 입력합니다. 이 이름은 변경할 수 없습니다.
5. 드롭다운 메뉴에서 스키마가 저장될 [레지스트리(Registry)]를 선택합니다. 상위 레지스트리는 생성 후에 변경할 수 없습니다.
6. [데이터 포맷(Data format)]을 [Apache Avro] 또는 [JSON]으로 둡니다. 이 포맷은 이 스키마의 모든 버전에 적용됩니다.
7. [호환성 모드(Compatibility mode)]를 선택합니다.
 - [뒤로(권장)(Backward (recommended))] - 수신자가 현재 버전과 이전 버전을 모두 읽을 수 있습니다.
 - [모두 뒤로(Backward All)] - 수신자가 현재 및 모든 이전 버전을 읽을 수 있습니다.
 - [앞으로(Forward)] - 발신자가 현재 버전과 이전 버전을 모두 쓸 수 있습니다.
 - [모두 앞으로(Forward All)] - 발신자가 현재 버전과 모든 이전 버전을 모두 쓸 수 있습니다.
 - [전체(Full)] - 뒤로 및 앞으로의 조합입니다.
 - [모두 전체(Full All)] - 모두 뒤로 및 모두 앞으로의 조합입니다.

- [없음(None)] - 호환성 확인이 수행되지 않습니다.
- [사용 중지됨(Disabled)] - 이 스키마에 대한 버전 관리를 방지합니다.

8. 최대 250자의 레지스트리에 대한 선택적 [설명(Description)]을 입력합니다.

The screenshot shows the AWS Glue console interface for adding a new schema. The left sidebar contains a navigation menu with categories like Data catalog, ETL, Security, and Tutorials. The main content area is titled 'Add a new schema' and includes the following sections:

- Schema name:** A text input field with a placeholder 'Enter a schema name...'. Below it, a note states: 'Only letters (A-Z), numbers (0-9), hyphens (-), underscores (_), dollar signs (\$), or hash marks (#) allowed. 255 characters maximum.'
- Registry:** A dropdown menu with a placeholder 'Select a registry...'. To its right are a refresh button and an 'Add new registry' button. A note above the dropdown says: 'Parent registry can't be changed post creation.'
- Data format:** A dropdown menu currently set to 'Apache Avro'. A note above it says: 'Glue schemas only support Apache Avro for now, which offers the compatibility options below. [Learn more](#)'
- Compatibility mode:** A dropdown menu set to 'Backward (default) - consumer can read both current and previous version'. A note above it says: 'Compatibility may be changed post creation and affects data senders and/or receivers.'
- Warning box:** A blue box with an information icon and the title 'Backward compatibility [Learn more](#)'. The text inside reads: 'This compatibility choice allows consumers to read both the current and the previous schema version. This means that for instance, a new schema version cannot drop data fields or change the type of these fields, so they can't be read by consumers using the previous version.'
- Description - optional:** A large text area with a placeholder 'Enter a schema description...'. Below it, a note states: '2048 characters maximum.'

9. 선택적으로 스키마에 하나 이상의 태그를 적용합니다. [새 태그 추가(Add new tag)]를 선택하고 [태그 키(Tag key)]와 [태그 값(Tag value)](선택 사항)을 지정합니다.

10 [첫 번째 스키마 버전(First schema version)] 상자에 초기 스키마를 입력하거나 붙여넣습니다.

Avro 포맷은 [Avro 데이터 포맷 작업](#) 섹션을 참조하세요.

JSON 포맷은 [JSON 데이터 포맷 작업](#) 섹션을 참조하세요.

11.선택적으로 [메타데이터 추가(Add metadata)]를 선택하여 스키마 버전에 주석을 달거나 분류할 버전 메타데이터를 추가합니다.

12[스키마 및 버전 생성(Create schema and version)]을 선택합니다.

AWS Glue

- Data catalog
- Databases
 - Tables
 - Connections
- Crawlers
- Classifiers
- Schema registries
 - Schemas**
- Settings
- ETL
- AWS Glue Studio
 - New
 - Blueprints
 - Workflows
 - Jobs
 - ML Transforms
 - Triggers
- Dev endpoints
 - Notebooks
- Security
 - Security configurations
- Tutorials
- Add crawler
- Explore table
- Add job
- Resources

Schema tags - optional
No tags defined.

[Add new tag](#)

You can add up to 50 more tags.

First schema version
Please specify the initial definition of your schema below, so that it can be used in your applications or within Amazon Glue. You may change your schema definition by registering new versions at any point later.
Please enter Apache Avro schema below. [Learn more](#)

1

Version metadata - optional
No metadata key-value pairs.

[Add metadata](#)

You can add 10 more metadata key-value pairs.

[Cancel](#) [Create schema and version](#)

스키마가 생성되고 [스키마(Schemas)] 아래의 목록에 나타납니다.

Avro 데이터 포맷 작업

Avro는 데이터 직렬화 및 데이터 교환 서비스를 제공합니다. Avro는 데이터 정의를 JSON 포맷으로 저장하므로 읽고 해석하기 쉽습니다. 데이터 자체는 바이너리 포맷으로 저장됩니다.

Apache Avro 스키마 정의에 대한 자세한 내용은 [Apache Avro 사양](#)을 참조하세요.

JSON 데이터 포맷 작업

데이터는 JSON 포맷으로 직렬화 할 수 있습니다. [JSON 스키마 포맷](#)은 JSON 스키마 포맷의 표준을 정의합니다.

스키마 또는 레지스트리 업데이트

생성되면 스키마, 스키마 버전 또는 레지스트리를 편집할 수 있습니다.

레지스트리 업데이트

AWS Glue API 또는 AWS Glue 콘솔을 사용하여 레지스트리를 업데이트할 수 있습니다. 기존 레지스트리의 이름은 편집할 수 없습니다. 레지스트리에 대한 설명을 편집할 수 있습니다.

AWS Glue API

기존 레지스트리를 업데이트하려면 [UpdateRegistry 액션 \(Python: 업데이트 레지스트리\)](#) API를 사용합니다.

업데이트할 레지스트리를 나타내는 RegistryId 구조를 지정합니다. 레지스트리에 대한 설명을 변경하려면 Description을 전달합니다.

```
aws glue update-registry --description updatedDescription --registry-id
RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName1"
```

AWS Glue 콘솔

AWS Glue 콘솔을 사용하여 레지스트리를 업데이트하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [스키마 레지스트리(Schema registries)]를 선택합니다.

3. 해당 확인란을 선택하여 레지스트리 목록에서 레지스트리를 선택합니다.
4. [작업(Action)] 메뉴에서 [레지스트리 편집(Edit registry)]을 선택합니다.

스키마 업데이트

스키마에 대한 설명 또는 호환성 설정을 업데이트할 수 있습니다.

기존 스키마를 업데이트하려면 [UpdateSchema 액션 \(Python: 업데이트_스키마\)](#) API를 사용합니다.

업데이트할 스키마를 나타내는 SchemaId 구조를 지정합니다. VersionNumber 또는 Compatibility 중 하나가 제공되어야 합니다.

코드 예제 11:

```
aws glue update-schema --description testDescription --schema-id
  SchemaName="testSchema1",RegistryName="registryName1" --schema-version-number
  LatestVersion=true --compatibility NONE
```

```
aws glue update-schema --description testDescription --schema-id
  SchemaArn="arn:aws:glue:us-east-2:901234567890:schema/registryName1/testSchema1" --
  schema-version-number LatestVersion=true --compatibility NONE
```

스키마 버전 추가

스키마 버전을 추가할 때 버전을 비교하여 새 스키마가 허용되는지 확인해야 합니다.

기존 스키마에 새 버전을 추가하려면 [RegisterSchemaVersion 액션 \(Python: 레지스터_스키마_버전\)](#) API를 사용합니다.

버전을 추가하려는 스키마를 나타내려면 SchemaId 구조를 지정하고 스키마를 정의하려면 SchemaDefinition을 지정합니다.

코드 예제 12:

```
aws glue register-schema-version --schema-definition "{\"type\": \"record\", \"name\":
  \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type
  \": \"string\"} ]}" --schema-id SchemaArn="arn:aws:glue:us-east-1:901234567890:schema/
  registryName/testschema"
```

```
aws glue register-schema-version --schema-definition "{\"type\": \"record\", \"name\": \"r1\", \"fields\": [ {\"name\": \"f1\", \"type\": \"int\"}, {\"name\": \"f2\", \"type\": \"string\"} ]}" --schema-id SchemaName="testschema",RegistryName="testregistry"
```

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [스키마(Schemas)]를 선택합니다.
3. 해당 확인란을 선택하여 스키마 목록에서 스키마를 선택합니다.
4. 확인란을 선택하여 목록에서 하나 이상의 스키마를 선택합니다.
5. [작업(Action)] 메뉴에서 [새 버전 등록(Register new version)]을 선택합니다.
6. [새 버전(New version)] 상자에 새 스키마를 입력하거나 붙여 넣습니다.
7. [이전 버전과 비교(Compare with previous version)]를 선택하여 이전 스키마 버전과의 차이점을 확인합니다.
8. 선택적으로 [메타데이터 추가(Add metadata)]를 선택하여 스키마 버전에 주석을 달거나 분류할 버전 메타데이터를 추가합니다. [키(Key)]와 [값(Value)](선택 사항)을 입력합니다.
9. [버전 등록(Register version)]을 선택합니다.

AWS Glue

Data catalog

Databases

Tables

Connections

Crawlers

Classifiers

Schema registries

Schemas

Settings

ETL

AWS Glue Studio

New

Blueprints

Workflows

Jobs

ML Transforms

Triggers

Dev endpoints

Notebooks

Security

Security configurations

Tutorials

Add crawler

Explore table

Add job

Schemas > test-1 > Register version

Register a new schema version

Register version 4 to your schema.

Schema name	test-1
Data format	Apache Avro
Compatibility mode	Backward compatibility
Schema tags	No tags defined.

New Version 4

This is a copy of version 1's schema definition. A schema definition not associated with any existing schema versions must be defined in order to register a new schema version.

```

1  {
2    "type": "record",
3    "name": "r0",
4    "fields": [
5      {
6        "name": "f1",
7        "type": "int"
8      }
9    ]
10 }
```

[Compare with previous version](#)

Version metadata - optional

No metadata key-value pairs.

[Add metadata](#)

You can add 10 more metadata key-value pairs.

[Cancel](#)
[Register version](#)

스키마 버전이 버전 목록에 나타납니다. 버전이 호환 모드를 변경한 경우 버전이 체크포인트로 표시됩니다.

스키마 버전 비교의 예

[이전 버전과 비교(Compare with previous version)]를 선택하면 이전 버전과 새 버전이 함께 표시됩니다. 변경된 정보는 다음과 같이 강조 표시됩니다.

- 노란색: 변경된 정보를 나타냅니다.

- 녹색: 최신 버전에 추가된 내용을 나타냅니다.
- 빨간색: 최신 버전에서 제거된 콘텐츠를 나타냅니다.

이전 버전과 비교할 수도 있습니다.

Schema version comparison

Schema test-1 Compatibility Mode Backward compatibility

Version 1 (latest a... ▾) Version 4 (new) ▾

```

1 {
2   "type": "record",
3-  "name": "r0",
4   "fields": [
5     {
6       "name": "f1",
7       "type": "int"
8     }
9   ]
10 }

```

```

1 {
2   "type": "record",
3+  "name": "user.record",
4+  "aliases": "userInfo",
5   "fields": [
6     {
7       "name": "f1",
8       "type": "int"
9     }
10  ]
11 }

```

Registered Thu, 01 Oct 2020 17:37:19 GMT Registered -

Metadata - Metadata -

Close

스키마 또는 레지스트리 삭제

스키마, 스키마 버전 또는 레지스트리 삭제는 취소할 수 없는 영구적인 작업입니다.

스키마 삭제

AWS Management Console 또는 [DeleteSchema 액션 \(Python: 삭제_스키마\)](#) API를 사용하여 레지스트리 내에서 더 이상 사용되지 않는 스키마를 삭제할 수 있습니다.

하나 이상의 스키마 삭제는 실행 취소할 수 없는 영구 작업입니다. 스키마가 이제 필요 없는지 확인합니다.

레지스트리에서 스키마를 삭제하려면 [DeleteSchema 액션 \(Python: 삭제_스키마\)](#) API를 호출하여 스키마를 식별하는 SchemaId 구조를 지정합니다.

예:

```
aws glue delete-schema --schema-id SchemaArn="arn:aws:glue:us-east-2:901234567890:schema/registryName1/schemaname"
```

```
aws glue delete-schema --schema-id SchemaName="TestSchema6-deleteschemabynome",RegistryName="default-registry"
```

AWS Glue 콘솔

AWS Glue 콘솔에서 스키마를 삭제하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [스키마 레지스트리(Schema registries)]를 선택합니다.
3. 레지스트리 목록에서 스키마가 포함된 레지스트리를 선택합니다.
4. 확인란을 선택하여 목록에서 하나 이상의 스키마를 선택합니다.
5. [작업(Action)] 메뉴에서 [스키마 삭제>Delete schema)]를 선택합니다.
6. 필드에 **Delete** 텍스트를 입력하여 삭제를 확인합니다.
7. 삭제를 선택합니다.

지정한 스키마가 레지스트리에서 삭제됩니다.

스키마 버전 삭제

스키마가 레지스트리에 누적되면 AWS Management Console 또는 [DeleteSchemaVersions 액션](#) ([Python: 스키마_버전 삭제](#)) API를 사용하여 원치 않는 스키마 버전을 삭제할 수 있습니다. 하나 이상의 스키마 버전 삭제는 실행 취소할 수 없는 영구 작업입니다. 스키마 버전이 이제 필요 없는지 확인합니다.

스키마 버전을 삭제할 때 다음 제약 조건에 유의합니다.

- 체크포인트된 버전은 삭제할 수 없습니다.
- 연속 버전의 범위는 25개를 초과할 수 없습니다.
- 최신 스키마 버전은 보류 상태가 아니어야 합니다.

SchemaId 구조를 지정하여 스키마를 식별하고 Versions를 삭제할 버전 범위로 지정합니다. 버전 또는 버전 범위 지정에 대한 자세한 내용은 [DeleteRegistry 액션 \(Python: 레지스트리 삭제\)](#) 섹션을 참조하세요. 지정한 스키마 버전이 레지스트리에서 삭제됩니다.

이 호출 후 [ListSchemaVersions 액션 \(Python: 목록 스키마 버전\)](#) API를 호출하면 삭제된 버전의 상태가 나열됩니다.

예:

```
aws glue delete-schema-versions --schema-id
  SchemaName="TestSchema6",RegistryName="default-registry" --versions "1-1"
```

```
aws glue delete-schema-versions --schema-id SchemaArn="arn:aws:glue:us-
east-2:901234567890:schema/default-registry/TestSchema6-NON-Existent" --versions "1-1"
```

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [스키마 레지스트리(Schema registries)]를 선택합니다.
3. 레지스트리 목록에서 스키마가 포함된 레지스트리를 선택합니다.
4. 확인란을 선택하여 목록에서 하나 이상의 스키마를 선택합니다.
5. [작업(Action)] 메뉴에서 [스키마 삭제>Delete schema)]를 선택합니다.
6. 필드에 **Delete** 텍스트를 입력하여 삭제를 확인합니다.
7. 삭제를 선택합니다.

지정한 스키마 버전이 레지스트리에서 삭제됩니다.

레지스트리 삭제

포함된 스키마가 해당 레지스트리 아래에 더 이상 구성되지 않아야 하는 경우 레지스트리를 삭제할 수 있습니다. 이러한 스키마를 다른 레지스트리에 재할당해야 합니다.

하나 이상의 레지스트리 삭제는 실행 취소할 수 없는 영구 작업입니다. 레지스트리가 이제 필요 없는지 확인합니다.

AWS CLI를 사용하여 기본 레지스트리를 삭제할 수 있습니다.

AWS Glue API

스키마와 모든 해당 버전을 포함한 전체 레지스트리를 삭제하려면 [DeleteRegistry 액션 \(Python: 레지스트리 삭제\)](#) API를 호출합니다. RegistryId 구조를 지정하여 레지스트리를 식별합니다.

예:

```
aws glue delete-registry --registry-id RegistryArn="arn:aws:glue:us-east-2:901234567890:registry/registryName1"
```

```
aws glue delete-registry --registry-id RegistryName="TestRegistry-deletebyname"
```

삭제 작업의 상태를 얻으려면 비동기 호출 후 GetRegistry API를 호출할 수 있습니다.

AWS Glue 콘솔

AWS Glue 콘솔에서 레지스트리를 삭제하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [스키마 레지스트리(Schema registries)]를 선택합니다.
3. 확인란을 선택하여 목록에서 레지스트리를 선택합니다.
4. [작업(Action)] 메뉴에서 [레지스트리 삭제>Delete registry)]를 선택합니다.
5. 필드에 **Delete** 텍스트를 입력하여 삭제를 확인합니다.
6. 삭제를 선택합니다.

선택한 레지스트리가 AWS Glue에서 삭제됩니다.

Serializer에 대한 IAM 예

Note

AWS 관리형 정책은 일반 사용 사례에서 필요한 권한을 부여합니다. 관리형 정책으로 스키마 레지스트리 관리에 대한 자세한 내용은 [AWS에 대한 관리형 \(사전 정의된\) 정책 AWS Glue](#) 섹션을 참조하세요.

Serializer의 경우 지정된 스키마 정의에 대해 schemaVersionId를 찾는 기능을 제공하려면 아래와 유사한 최소 정책을 만들어야 합니다. 레지스트리의 스키마를 읽으려면 레지스트리에 대한 읽기 권한이 있어야 합니다. Resource 절을 사용하여 읽을 수 있는 레지스트리를 제한할 수 있습니다.

코드 예제 13:

```
{
  "Sid" : "GetSchemaByDefinition",
  "Effect" : "Allow",
  "Action" :
  [
    "glue:GetSchemaByDefinition"
  ],
  "Resource" : ["arn:aws:glue:us-east-2:012345678:registry/registryname-1",
                "arn:aws:glue:us-east-2:012345678:schema/registryname-1/
schemaname-1",
                "arn:aws:glue:us-east-2:012345678:schema/registryname-1/
schemaname-2"
               ]
}
```

또한 다음과 같은 추가 메서드를 포함하여 생산자가 새 스키마 및 버전을 생성하도록 허용할 수도 있습니다. 참고로 레지스트리 내부의 스키마를 추가/제거/발전시키기 위해 레지스트리를 검사할 수 있어야 합니다. Resource 절을 사용하여 검사할 수 있는 레지스트리를 제한할 수 있습니다.

코드 예제 14:

```
{
  "Sid" : "RegisterSchemaWithMetadata",
  "Effect" : "Allow",
  "Action" :
  [
    "glue:GetSchemaByDefinition",
    "glue:CreateSchema",
    "glue:RegisterSchemaVersion",
    "glue:PutSchemaVersionMetadata",
  ],
  "Resource" : ["arn:aws:glue:aws-region:123456789012:registry/registryname-1",
                "arn:aws:glue:aws-region:123456789012:schema/registryname-1/
schemaname-1",
                "arn:aws:glue:aws-region:123456789012:schema/registryname-1/
schemaname-2"
               ]
}
```


}

Deserializer에 대한 IAM 예

Deserializer(소비자 측)의 경우 deserializer가 역직렬화를 위해 Schema Registry에서 스키마를 가져올 수 있도록 아래와 유사한 정책을 생성해야 합니다. 참고로 레지스트리 내부의 스키마를 가져오기 위해 레지스트리를 검사할 수 있어야 합니다.

코드 예제 15:

```
{
  "Sid" : "GetSchemaVersion",
  "Effect" : "Allow",
  "Action" :
  [
    "glue:GetSchemaVersion"
  ],
  "Resource" : ["*"]
}
```

AWS PrivateLink를 사용한 프라이빗 연결

AWS PrivateLink를 사용하여 AWS Glue에 대한 인터페이스 VPC 엔드포인트를 정의하여 데이터 생산자의 VPC를 AWS Glue에 연결할 수 있습니다. VPC 엔드포인트를 사용하는 경우 VPC와 AWS Glue 사이의 통신은 모두 AWS 네트워크에서 수행됩니다. 자세한 내용은 [VPC 엔드포인트와 함께 AWS Glue 사용](#)을 참조하세요.

Amazon CloudWatch 지표에 액세스

Amazon CloudWatch 지표는 CloudWatch 프리 티어의 일부로 제공됩니다. CloudWatch 콘솔에서 이러한 지표에 액세스할 수 있습니다. API 수준 지표에는 CreateSchema(성공 및 대기 시간), GetSchemaByDefinition, (성공 및 대기 시간), GetSchemaVersion(성공 및 대기 시간), RegisterSchemaVersion(성공 및 대기 시간), PutSchemaVersionMetadata(성공 및 대기 시간)가 포함됩니다. 리소스 수준 지표에는 Registry.ThrottledByLimit, SchemaVersion.ThrottledByLimit, SchemaVersion.Size가 포함됩니다.

Schema Registry용 샘플 AWS CloudFormation 템플릿

다음은 AWS CloudFormation에서 Schema Registry 리소스를 생성하기 위한 샘플 템플릿입니다. 계정에 이 스택을 생성하려면 위 템플릿을 파일 SampleTemplate.yaml에 복사하고 다음 명령을 실행합니다.

```
aws cloudformation create-stack --stack-name ABCSchemaRegistryStack --template-body
  "'cat SampleTemplate.yaml'"
```

이 예에서는 `AWS::Glue::Registry`를 사용하여 레지스트리를 생성하고, `AWS::Glue::Schema`를 사용하여 스키마를 생성하고, `AWS::Glue::SchemaVersion`을 사용하여 스키마 버전을 생성하고, `AWS::Glue::SchemaVersionMetadata`를 사용하여 스키마 버전 메타데이터를 채웁니다.

Description: "A sample CloudFormation template for creating Schema Registry resources."

Resources:

ABCRegistry:

Type: "AWS::Glue::Registry"

Properties:

Name: "ABCSchemaRegistry"

Description: "ABC Corp. Schema Registry"

Tags:

- Key: "Project"

Value: "Foo"

ABCSchema:

Type: "AWS::Glue::Schema"

Properties:

Registry:

Arn: !Ref ABCRegistry

Name: "TestSchema"

Compatibility: "NONE"

DataFormat: "AVRO"

SchemaDefinition: >

```
{ "namespace": "foo.avro", "type": "record", "name": "user", "fields":
```

```
[{"name": "name", "type": "string"}, {"name": "favorite_number", "type": "int"}]}
```

Tags:

- Key: "Project"

Value: "Foo"

SecondSchemaVersion:

Type: "AWS::Glue::SchemaVersion"

Properties:

Schema:

SchemaArn: !Ref ABCSchema

SchemaDefinition: >

```
{ "namespace": "foo.avro", "type": "record", "name": "user", "fields":
```

```
[{"name": "status", "type": "string", "default": "ON"}, {"name": "name", "type": "string"},
```

```
{"name": "favorite_number", "type": "int"}]}
```

FirstSchemaVersionMetadata:

Type: "AWS::Glue::SchemaVersionMetadata"

```

Properties:
  SchemaVersionId: !GetAtt ABCSchema.InitialSchemaVersionId
  Key: "Application"
  Value: "Kinesis"
SecondSchemaVersionMetadata:
  Type: "AWS::Glue::SchemaVersionMetadata"
  Properties:
    SchemaVersionId: !Ref SecondSchemaVersion
    Key: "Application"
    Value: "Kinesis"

```

AWS Glue Schema Registry와 통합

이 섹션에서는 AWS Glue Schema Registry와의 통합에 대해 설명합니다. 이 섹션의 예에서는 AVRO 데이터 포맷의 스키마를 보여줍니다. JSON 데이터 포맷의 스키마를 포함한 더 많은 예제는 [AWS Glue Schema Registry 오픈 소스 리포지토리](#)의 통합 테스트 및 README 정보를 참조하세요.

주제

- [사용 사례: Amazon MSK 또는 Apache Kafka에 Schema Registry 연결](#)
- [사용 사례: AWS Glue Schema Registry와 Amazon Kinesis Data Streams 통합](#)
- [사용 사례: Amazon Managed Service for Apache Flink](#)
- [사용 사례: AWS Lambda와 통합](#)
- [사용 사례: AWS Glue Data Catalog](#)
- [사용 사례: AWS Glue 스트리밍](#)
- [사용 사례: Apache Kafka Streams](#)
- [사용 사례: Apache Kafka Connect](#)

사용 사례: Amazon MSK 또는 Apache Kafka에 Schema Registry 연결

Apache Kafka 주제에 데이터를 쓰고 있다고 가정하고 다음 단계에 따라 시작할 수 있습니다.

1. 하나 이상의 주제로 Amazon Managed Streaming for Apache Kafka(Amazon MSK) 또는 Apache Kafka 클러스터를 생성합니다. Amazon MSK 클러스터를 생성하는 경우 AWS Management Console을 사용할 수 있습니다. Amazon Managed Streaming for Apache Kafka Developer Guide의 [Getting Started Using Amazon MSK](#)에서 설명하는 지침을 따릅니다.
2. 위의 [SerDe 라이브러리 설치](#) 단계를 따릅니다.

3. 스키마 레지스트리, 스키마 또는 스키마 버전을 생성하려면 이 문서의 [Schema Registry 시작하기](#) 섹션에 있는 지침을 따르세요.
4. 생산자와 소비자가 Schema Registry를 사용하여 Amazon MSK 또는 Apache Kafka 주제에서 레코드를 쓰고 읽도록 시작합니다. 예제 생산자 및 소비자 코드는 Serde 라이브러리의 [ReadMe 파일](#)에서 찾을 수 있습니다. 생산자의 Schema Registry 라이브러리는 자동으로 레코드를 직렬화하고 스키마 버전 ID로 레코드를 장식합니다.
5. 이 레코드의 스키마가 입력되었거나 자동 등록이 설정되어 있으면 Schema Registry에 스키마가 등록됩니다.
6. AWS Glue Schema Registry 라이브러리를 사용하여 Amazon MSK 또는 Apache Kafka 주제에서 읽는 소비자는 Schema Registry에서 스키마를 자동으로 조회합니다.

사용 사례: AWS Glue Schema Registry와 Amazon Kinesis Data Streams 통합

이 통합을 위해서는 기존 Amazon Kinesis 데이터 스트림이 있어야 합니다. 자세한 내용은 Amazon Kinesis Data Streams Developer Guide의 [Getting Started with Amazon Kinesis Data Streams](#)를 참조하세요.

Kinesis 데이터 스트림의 데이터와 상호 작용할 수 있는 두 가지 방법이 있습니다.

- Java의 Kinesis Producer Library(KPL) 및 Kinesis Client Library(KCL) 라이브러리를 통해 다국어 지원은 제공되지 않습니다.
- AWS SDK for Java에서 사용 가능한 PutRecords, PutRecord 및 GetRecords Kinesis Data Streams API를 통해.

현재 KPL/KCL 라이브러리를 사용하는 경우 해당 방법을 계속 사용하는 것이 좋습니다. 예제와 같이 Schema Registry가 통합된 업데이트된 KCL 및 KPL 버전이 있습니다. 그렇지 않으면 AWS Glue KDS API를 직접 사용하는 경우 샘플 코드를 사용하여 Schema Registry를 활용할 수 있습니다.

Schema Registry 통합은 KPL v0.14.2 이상 및 KCL v2.3 이상에서만 사용할 수 있습니다. JSON 데이터 포맷과 Schema Registry 통합은 KPL v0.14.8 이상 및 KCL v2.3.6 이상에서 사용할 수 있습니다.

Kinesis SDK V2를 사용하여 데이터와 상호 작용

이 섹션에서는 Kinesis SDK V2를 사용한 Kinesis와의 상호 작용에 대해 설명합니다.

```
// Example JSON Record, you can construct a AVRO record also
private static final JsonDataWithSchema record =
    JsonDataWithSchema.builder(schemaString, payloadString);
```

```
private static final DataFormat dataFormat = DataFormat.JSON;

//Configurations for Schema Registry
GlueSchemaRegistryConfiguration gsrConfig = new GlueSchemaRegistryConfiguration("us-
east-1");

GlueSchemaRegistrySerializer glueSchemaRegistrySerializer =
    new GlueSchemaRegistrySerializerImpl(awsCredentialsProvider, gsrConfig);
GlueSchemaRegistryDataFormatSerializer dataFormatSerializer =
    new GlueSchemaRegistrySerializerFactory().getInstance(dataFormat, gsrConfig);

Schema gsrSchema =
    new Schema(dataFormatSerializer.getSchemaDefinition(record), dataFormat.name(),
    "MySchema");

byte[] serializedBytes = dataFormatSerializer.serialize(record);

byte[] gsrEncodedBytes = glueSchemaRegistrySerializer.encode(streamName, gsrSchema,
    serializedBytes);

PutRecordRequest putRecordRequest = PutRecordRequest.builder()
    .streamName(streamName)
    .partitionKey("partitionKey")
    .data(SdkBytes.fromByteArray(gsrEncodedBytes))
    .build();
shardId = kinesisClient.putRecord(putRecordRequest)
    .get()
    .shardId();

GlueSchemaRegistryDeserializer glueSchemaRegistryDeserializer = new
    GlueSchemaRegistryDeserializerImpl(awsCredentialsProvider, gsrConfig);

GlueSchemaRegistryDataFormatDeserializer gsrDataFormatDeserializer =
    glueSchemaRegistryDeserializerFactory.getInstance(dataFormat, gsrConfig);

GetShardIteratorRequest getShardIteratorRequest = GetShardIteratorRequest.builder()
    .streamName(streamName)
    .shardId(shardId)
    .shardIteratorType(ShardIteratorType.TRIM_HORIZON)
    .build();

String shardIterator = kinesisClient.getShardIterator(getShardIteratorRequest)
    .get()
    .shardIterator();
```

```

GetRecordsRequest getRecordRequest = GetRecordsRequest.builder()
    .shardIterator(shardIterator)
    .build();
GetRecordsResponse recordsResponse = kinesisClient.getRecords(getRecordRequest)
    .get();

List<Object> consumerRecords = new ArrayList<>();
List<Record> recordsFromKinesis = recordsResponse.records();

for (int i = 0; i < recordsFromKinesis.size(); i++) {
    byte[] consumedBytes = recordsFromKinesis.get(i)
        .data()
        .asByteArray();

    Schema gsrSchema = glueSchemaRegistryDeserializer.getSchema(consumedBytes);
    Object decodedRecord =
    gsrDataFormatDeserializer.deserialize(ByteBuffer.wrap(consumedBytes),

    gsrSchema.getSchemaDefinition());
    consumerRecords.add(decodedRecord);
}

```

KPL/KCL 라이브러리를 사용하여 데이터와 상호 작용

이 섹션에서는 KPL/KCL 라이브러리를 사용하여 Kinesis Data Streams를 Schema Registry와 통합하는 방법을 설명합니다. KPL/KCL 사용에 대한 자세한 내용은 Amazon Kinesis Data Streams Developer Guide의 [Developing Producers Using the Amazon Kinesis Producer Library](#)를 참조하세요.

KPL에서 Schema Registry 설정

1. AWS Glue Schema Registry에서 작성된 데이터, 데이터 포맷 및 스키마 이름에 대한 스키마 정의를 정의합니다.
2. 필요에 따라 GlueSchemaRegistryConfiguration 객체를 구성합니다.
3. addUserRecord API로 스키마 객체를 전달합니다.

```

private static final String SCHEMA_DEFINITION = "{\"namespace\": \"example.avro\",\\n\"
+ \" \"type\": \"record\",\\n\"
+ \" \"name\": \"User\",\\n\"
+ \" \"fields\": [\\n\"
+ \" {\"name\": \"name\", \"type\": \"string\"},\\n\"
+ \" {\"name\": \"favorite_number\", \"type\": [\"int\", \"null\"]},\\n\"

```

```
+ " {"name": "favorite_color", "type": ["string", "null"]}\n"
+ " ]\n"
+ "}";

KinesisProducerConfiguration config = new KinesisProducerConfiguration();
config.setRegion("us-west-1")

//[Optional] configuration for Schema Registry.

GlueSchemaRegistryConfiguration schemaRegistryConfig =
new GlueSchemaRegistryConfiguration("us-west-1");

schemaRegistryConfig.setCompression(true);

config.setGlueSchemaRegistryConfiguration(schemaRegistryConfig);

///Optional configuration ends.

final KinesisProducer producer =
    new KinesisProducer(config);

final ByteBuffer data = getDataToSend();

com.amazonaws.services.schemaregistry.common.Schema gsrSchema =
new Schema(SCHEMA_DEFINITION, DataFormat.AVRO.toString(), "demoSchema");

ListenableFuture<UserRecordResult> f = producer.addUserRecord(
    config.getStreamName(), TIMESTAMP, Utils.randomExplicitHashKey(), data, gsrSchema);

private static ByteBuffer getDataToSend() {
    org.apache.avro.Schema avroSchema =
        new org.apache.avro.Schema.Parser().parse(SCHEMA_DEFINITION);

    GenericRecord user = new GenericData.Record(avroSchema);
    user.put("name", "Emily");
    user.put("favorite_number", 32);
    user.put("favorite_color", "green");

    ByteArrayOutputStream outBytes = new ByteArrayOutputStream();
    Encoder encoder = EncoderFactory.get().directBinaryEncoder(outBytes, null);
    new GenericDatumWriter<>(avroSchema).write(user, encoder);
    encoder.flush();
    return ByteBuffer.wrap(outBytes.toByteArray());
}
```

}

Kinesis 클라이언트 라이브러리 설정

Java로 Kinesis Client Library 소비자를 개발합니다. 자세한 내용은 Amazon Kinesis Data Streams Developer Guide의 [Developing a Kinesis Client Library Consumer in Java](#)를 참조하세요.

1. `GlueSchemaRegistryConfiguration` 객체를 전달하여 `GlueSchemaRegistryDeserializer`의 인스턴스를 생성합니다.
2. `retrievalConfig.glueSchemaRegistryDeserializer`에 `GlueSchemaRegistryDeserializer`를 전달합니다.
3. `kinesisClientRecord.getSchema()`를 호출하여 수신 메시지의 스키마에 액세스합니다.

```

GlueSchemaRegistryConfiguration schemaRegistryConfig =
    new GlueSchemaRegistryConfiguration(this.region.toString());

GlueSchemaRegistryDeserializer glueSchemaRegistryDeserializer =
    new
GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(),
    schemaRegistryConfig);

RetrievalConfig retrievalConfig =
    configsBuilder.retrievalConfig().retrievalSpecificConfig(new
    PollingConfig(streamName, kinesisClient));
retrievalConfig.glueSchemaRegistryDeserializer(glueSchemaRegistryDeserializer);

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    retrievalConfig
);

public void processRecords(ProcessRecordsInput processRecordsInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Processing {} record(s)",
            processRecordsInput.records().size());
    }
}

```



```

        processRecordsInput.records()
            .forEach(
                r ->
                    log.info("Processed record pk: {} -- Seq: {} : data {} with
schema: {}",
                        r.partitionKey(),
r.sequenceNumber(), recordToAvroObj(r).toString(), r.getSchema());
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

private GenericRecord recordToAvroObj(KinesisClientRecord r) {
    byte[] data = new byte[r.data().remaining()];
    r.data().get(data, 0, data.length);
    org.apache.avro.Schema schema = new
org.apache.avro.Schema.Parser().parse(r.schema().getSchemaDefinition());
    DatumReader datumReader = new GenericDatumReader<>(schema);

    BinaryDecoder binaryDecoder = DecoderFactory.get().binaryDecoder(data, 0,
data.length, null);
    return (GenericRecord) datumReader.read(null, binaryDecoder);
}

```

Kinesis Data Streams API를 사용하여 데이터와 상호 작용

이 섹션에서는 Kinesis Data Streams API를 사용하여 Kinesis Data Streams를 Schema Registry와 통합하는 방법을 설명합니다.

1. 다음 Maven 종속성을 업데이트합니다.

```

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>1.11.884</version>
      <type>pom</type>
      <scope>import</scope>

```

```

        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-kinesis</artifactId>
    </dependency>

    <dependency>
        <groupId>software.amazon.glue</groupId>
        <artifactId>schema-registry-serde</artifactId>
        <version>1.1.5</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-cbor</artifactId>
        <version>2.11.3</version>
    </dependency>
</dependencies>

```

2. 생산자에서 Kinesis Data Streams의 PutRecords 또는 PutRecord API를 사용하여 스키마 헤더 정보를 추가합니다.

```

//The following lines add a Schema Header to the record
    com.amazonaws.services.schemaregistry.common.Schema awsSchema =
        new com.amazonaws.services.schemaregistry.common.Schema(schemaDefinition,
            DataFormat.AVRO.name(),
            schemaName);
    GlueSchemaRegistrySerializerImpl glueSchemaRegistrySerializer =
        new
        GlueSchemaRegistrySerializerImpl(DefaultCredentialsProvider.builder().build(), new
        GlueSchemaRegistryConfiguration(getConfigs()));
    byte[] recordWithSchemaHeader =
        glueSchemaRegistrySerializer.encode(streamName, awsSchema,
        recordAsBytes);

```

3. 생산자에서 PutRecords 또는 PutRecord API를 사용하여 레코드를 데이터 스트림에 넣습니다.
4. 소비자의 헤더에서 스키마 레코드를 제거하고 Avro 스키마 레코드를 직렬화합니다.

```

//The following lines remove Schema Header from record

```

```

        GlueSchemaRegistryDeserializerImpl glueSchemaRegistryDeserializer =
            new
GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(),
getConfigs());
        byte[] recordWithSchemaHeaderBytes = new
byte[recordWithSchemaHeader.remaining()];
        recordWithSchemaHeader.get(recordWithSchemaHeaderBytes, 0,
recordWithSchemaHeaderBytes.length);
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            glueSchemaRegistryDeserializer.getSchema(recordWithSchemaHeaderBytes);
        byte[] record =
glueSchemaRegistryDeserializer.getData(recordWithSchemaHeaderBytes);

        //The following lines serialize an AVRO schema record
        if (DataFormat.AVRO.name().equals(awsSchema.getDataFormat())) {
            Schema avroSchema = new
org.apache.avro.Schema.Parser().parse(awsSchema.getSchemaDefinition());
            Object genericRecord = convertBytesToRecord(avroSchema, record);
            System.out.println(genericRecord);
        }
    }

```

Kinesis Data Streams API를 사용하여 데이터와 상호 작용

다음은 PutRecords 및 GetRecords API 사용을 위한 예제 코드입니다.

```

//Full sample code
import
com.amazonaws.services.schemaregistry.deserializers.GlueSchemaRegistryDeserializerImpl;
import
com.amazonaws.services.schemaregistry.serializers.GlueSchemaRegistrySerializerImpl;
import com.amazonaws.services.schemaregistry.utils.AVROUtils;
import com.amazonaws.services.schemaregistry.utils.AWSSchemaRegistryConstants;
import org.apache.avro.Schema;
import org.apache.avro.generic.GenericData;
import org.apache.avro.generic.GenericDatumReader;
import org.apache.avro.generic.GenericDatumWriter;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.io.Decoder;
import org.apache.avro.io.DecoderFactory;
import org.apache.avro.io.Encoder;
import org.apache.avro.io.EncoderFactory;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.services.glue.model.DataFormat;

```

```
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class PutAndGetExampleWithEncodedData {
    static final String regionName = "us-east-2";
    static final String streamName = "testStream1";
    static final String schemaName = "User-Topic";
    static final String AVRO_USER_SCHEMA_FILE = "src/main/resources/user.avsc";
    KinesisApi kinesisApi = new KinesisApi();

    void runSampleForPutRecord() throws IOException {
        Object testRecord = getTestRecord();
        byte[] recordAsBytes = convertRecordToBytes(testRecord);
        String schemaDefinition =
AVROUtils.getInstance().getSchemaDefinition(testRecord);

        //The following lines add a Schema Header to a record
        com.amazonaws.services.schemaregistry.common.Schema awsSchema =
            new com.amazonaws.services.schemaregistry.common.Schema(schemaDefinition,
DataFormat.AVRO.name(),
                schemaName);
        GlueSchemaRegistrySerializerImpl glueSchemaRegistrySerializer =
            new
GlueSchemaRegistrySerializerImpl(DefaultCredentialsProvider.builder().build(), new
GlueSchemaRegistryConfiguration(regionName));
        byte[] recordWithSchemaHeader =
            glueSchemaRegistrySerializer.encode(streamName, awsSchema, recordAsBytes);

        //Use PutRecords api to pass a list of records
        kinesisApi.putRecords(Collections.singletonList(recordWithSchemaHeader),
streamName, regionName);

        //OR
        //Use PutRecord api to pass single record
        //kinesisApi.putRecord(recordWithSchemaHeader, streamName, regionName);
    }
}
```

```

byte[] runSampleForGetRecord() throws IOException {
    ByteBuffer recordWithSchemaHeader = kinesisApi.getRecords(streamName,
regionName);

    //The following lines remove the schema registry header
    GlueSchemaRegistryDeserializerImpl glueSchemaRegistryDeserializer =
        new
GlueSchemaRegistryDeserializerImpl(DefaultCredentialsProvider.builder().build(), new
GlueSchemaRegistryConfiguration(regionName));
    byte[] recordWithSchemaHeaderBytes = new
byte[recordWithSchemaHeader.remaining()];
    recordWithSchemaHeader.get(recordWithSchemaHeaderBytes, 0,
recordWithSchemaHeaderBytes.length);

    com.amazonaws.services.schemaregistry.common.Schema awsSchema =
        glueSchemaRegistryDeserializer.getSchema(recordWithSchemaHeaderBytes);

    byte[] record =
glueSchemaRegistryDeserializer.getData(recordWithSchemaHeaderBytes);

    //The following lines serialize an AVRO schema record
    if (DataFormat.AVRO.name().equals(awsSchema.getDataFormat())) {
        Schema avroSchema = new
org.apache.avro.Schema.Parser().parse(awsSchema.getSchemaDefinition());
        Object genericRecord = convertBytesToRecord(avroSchema, record);
        System.out.println(genericRecord);
    }

    return record;
}

private byte[] convertRecordToBytes(final Object record) throws IOException {
    ByteArrayOutputStream recordAsBytes = new ByteArrayOutputStream();
    Encoder encoder = EncoderFactory.get().directBinaryEncoder(recordAsBytes,
null);
    GenericDatumWriter datumWriter = new
GenericDatumWriter<>(AVROUtils.getInstance().getSchema(record));
    datumWriter.write(record, encoder);
    encoder.flush();
    return recordAsBytes.toByteArray();
}

private GenericRecord convertBytesToRecord(Schema avroSchema, byte[] record) throws
IOException {

```

```
        final GenericDatumReader<GenericRecord> datumReader = new
GenericDatumReader<>(avroSchema);
        Decoder decoder = DecoderFactory.get().binaryDecoder(record, null);
        GenericRecord genericRecord = datumReader.read(null, decoder);
        return genericRecord;
    }

    private Map<String, String> getMetadata() {
        Map<String, String> metadata = new HashMap<>();
        metadata.put("event-source-1", "topic1");
        metadata.put("event-source-2", "topic2");
        metadata.put("event-source-3", "topic3");
        metadata.put("event-source-4", "topic4");
        metadata.put("event-source-5", "topic5");
        return metadata;
    }

    private GlueSchemaRegistryConfiguration getConfigs() {
        GlueSchemaRegistryConfiguration configs = new
GlueSchemaRegistryConfiguration(regionName);
        configs.setSchemaName(schemaName);
        configs.setAutoRegistration(true);
        configs.setMetadata(getMetadata());
        return configs;
    }

    private Object getTestRecord() throws IOException {
        GenericRecord genericRecord;
        Schema.Parser parser = new Schema.Parser();
        Schema avroSchema = parser.parse(new File(AVRO_USER_SCHEMA_FILE));

        genericRecord = new GenericData.Record(avroSchema);
        genericRecord.put("name", "testName");
        genericRecord.put("favorite_number", 99);
        genericRecord.put("favorite_color", "red");

        return genericRecord;
    }
}
```

사용 사례: Amazon Managed Service for Apache Flink

Apache Flink는 무제한 및 제한 데이터 스트림에 대한 상태 저장 계산에 널리 사용되는 오픈 소스 프레임워크 및 분산 처리 엔진입니다. Amazon Managed Service for Apache Flink는 스트리밍 데이터를 처리하는 Apache Flink 애플리케이션을 구축 및 관리할 수 있는 완전관리형 AWS 서비스입니다.

오픈 소스 Apache Flink는 다양한 소스와 싱크를 제공합니다. 예를 들어 사전 정의된 데이터 원본에는 파일, 디렉터리 및 소켓에서 읽기와 컬렉션 및 반복기에서 데이터 수집이 포함됩니다. Apache Flink DataStream 커넥터는 Apache Flink가 소스 및/또는 싱크로 Apache Kafka 또는 Kinesis와 같은 다양한 서드 파티 시스템과 인터페이스할 수 있는 코드를 제공합니다.

자세한 내용은 [Amazon Kinesis Data Analytics Developer Guide](#)를 참조하세요.

Apache Flink Kafka 커넥터

Apache Flink는 정확히 1회 보장으로 Kafka 주제에서 데이터를 읽고 쓰기 위한 Apache Kafka 데이터 스트림 커넥터를 제공합니다. Flink의 Kafka 소비자인 FlinkKafkaConsumer는 하나 이상의 Kafka 주제에서 읽을 수 있는 액세스를 제공합니다. Apache Flink의 Kafka Producer인 FlinkKafkaProducer를 사용하면 하나 이상의 Kafka 주제에 대한 레코드 스트림을 작성할 수 있습니다. 자세한 내용은 [Apache Kafka 커넥터](#)를 참조하세요.

Apache Flink Kinesis 스트림 커넥터

Kinesis 데이터 스트림 커넥터는 Amazon Kinesis Data Streams에 대한 액세스를 제공합니다. FlinkKinesisConsumer는 동일한 AWS 서비스 리전 내에서 여러 Kinesis 스트림을 구독하는 정확히 1회 병렬 스트리밍 데이터 원본이며 작업이 실행되는 동안 스트림의 다시 샤딩을 투명하게 처리할 수 있습니다. 소비자의 각 하위 태스크는 여러 Kinesis 샤드에서 데이터 레코드를 가져오는 일을 담당합니다. 각 하위 태스크에서 가져온 샤드 수는 샤드가 다하고 Kinesis에서 생성됨에 따라 변경됩니다. FlinkKinesisProducer는 Kinesis Producer Library(KPL)를 사용하여 Apache Flink 스트림의 데이터를 Kinesis 스트림에 넣습니다. 자세한 내용은 [Amazon Kinesis Streams 커넥터](#)를 참조하세요.

자세한 내용은 [AWS Glue 스키마 Github 리포지토리](#)를 참조하세요.

Apache Flink와의 통합

Schema Registry와 함께 제공되는 SerDes 라이브러리는 Apache Flink와 통합됩니다. Apache Flink를 사용하려면 Apache Flink 커넥터에 연결할 수 있는 GlueSchemaRegistryAvroSerializationSchema 및 GlueSchemaRegistryAvroDeserializationSchema라는 [SerializationSchema](#) 및 [DeserializationSchema](#) 인터페이스를 구현해야 합니다.

Apache Flink 애플리케이션에 AWS Glue Schema Registry 종속성 추가

Apache Flink 애플리케이션에서 AWS Glue Schema Registry에 대한 통합 종속성을 설정하려면

1. 종속 프로그램을 pom.xml 파일에 추가합니다.

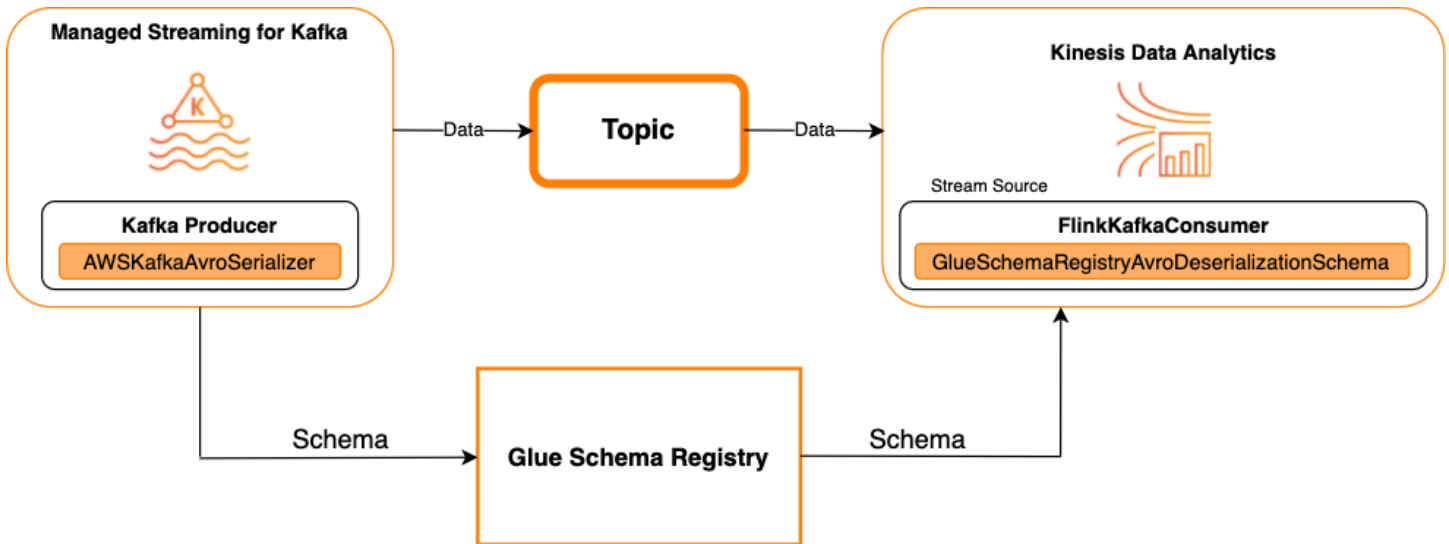
```
<dependency>
  <groupId>software.amazon.glue</groupId>
  <artifactId>schema-registry-flink-serde</artifactId>
  <version>1.0.0</version>
</dependency>
```

Apache Flink와 Kafka 또는 Amazon MSK 통합

Kafka를 소스나 싱크로 사용하여 Apache Flink용 Managed Service for Apache Flink를 사용할 수 있습니다.

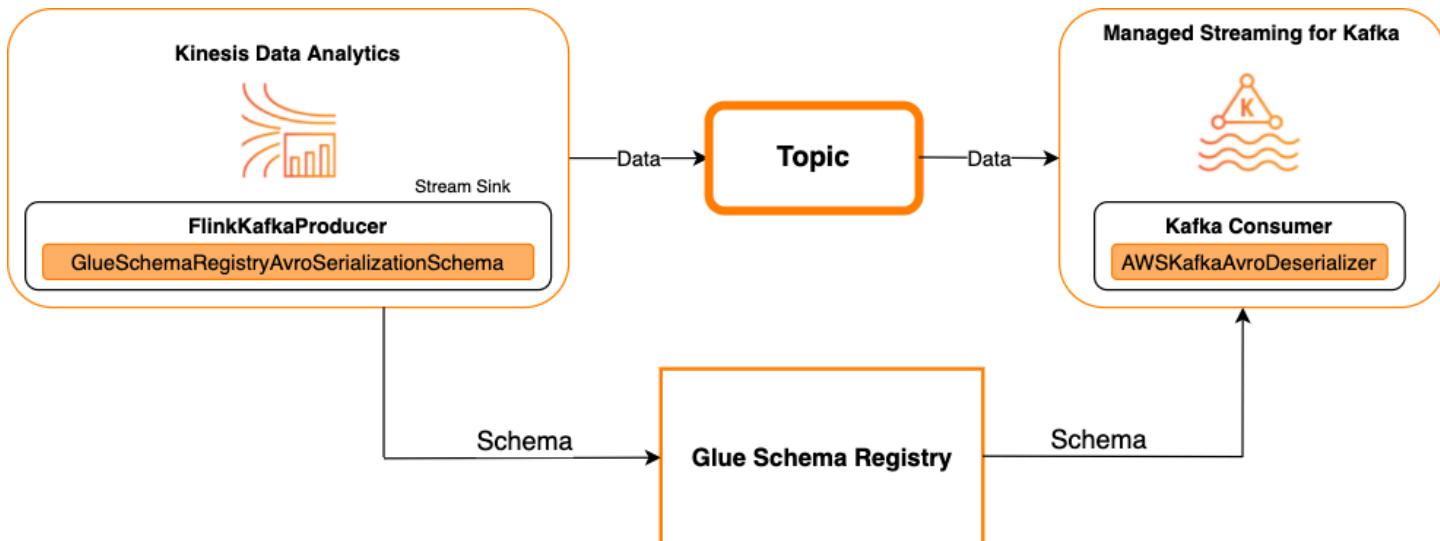
소스형 Kafka

다음 다이어그램에서는 Kafka를 소스로 사용하여 Kinesis Data Streams를 Apache Flink용 Managed Service for Apache Flink와 통합하는 방법을 보여줍니다.



싱크로 Kafka

다음 다이어그램에서는 Kafka를 싱크로 사용하여 Kinesis Data Streams를 Apache Flink용 Managed Service for Apache Flink와 통합하는 방법을 보여줍니다.



Kafka를 소스 또는 싱크로 사용하여 Kafka(또는 Amazon MSK)를 Apache Flink용 Managed Service for Apache Flink와 통합하려면 아래와 같이 코드를 변경합니다. 굵게 표시된 코드 블록을 유사한 섹션의 해당 코드에 추가합니다.

Kafka가 소스인 경우 deserializer 코드를 사용합니다(블록 2). Kafka가 싱크인 경우 serializer 코드(블록 3)를 사용합니다.

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

String topic = "topic";
Properties properties = new Properties();
properties.setProperty("bootstrap.servers", "localhost:9092");
properties.setProperty("group.id", "test");

// block 1
Map<String, Object> configs = new HashMap<>();
configs.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
configs.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
configs.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
    AvroRecordType.GENERIC_RECORD.getName());

FlinkKafkaConsumer<GenericRecord> consumer = new FlinkKafkaConsumer<>(
    topic,
    // block 2
    GlueSchemaRegistryAvroDeserializationSchema.forGeneric(schema, configs),
    properties);

FlinkKafkaProducer<GenericRecord> producer = new FlinkKafkaProducer<>(
```

```

topic,
// block 3
GlueSchemaRegistryAvroSerializationSchema.forGeneric(schema, topic, configs),
properties);

DataStream<GenericRecord> stream = env.addSource(consumer);
stream.addSink(producer);
env.execute();

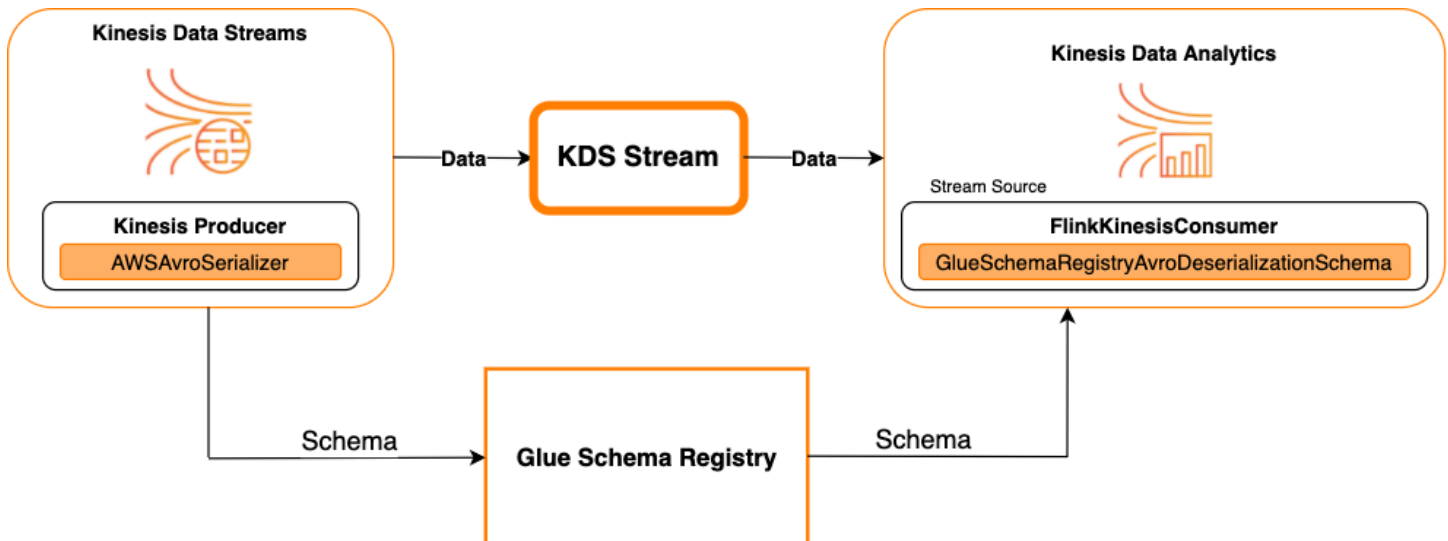
```

Apache Flink와 Kinesis Data Streams 통합

Kinesis Data Streams를 소스 또는 싱크로 사용하여 Apache Flink용 Managed Service for Apache Flink를 사용할 수 있습니다.

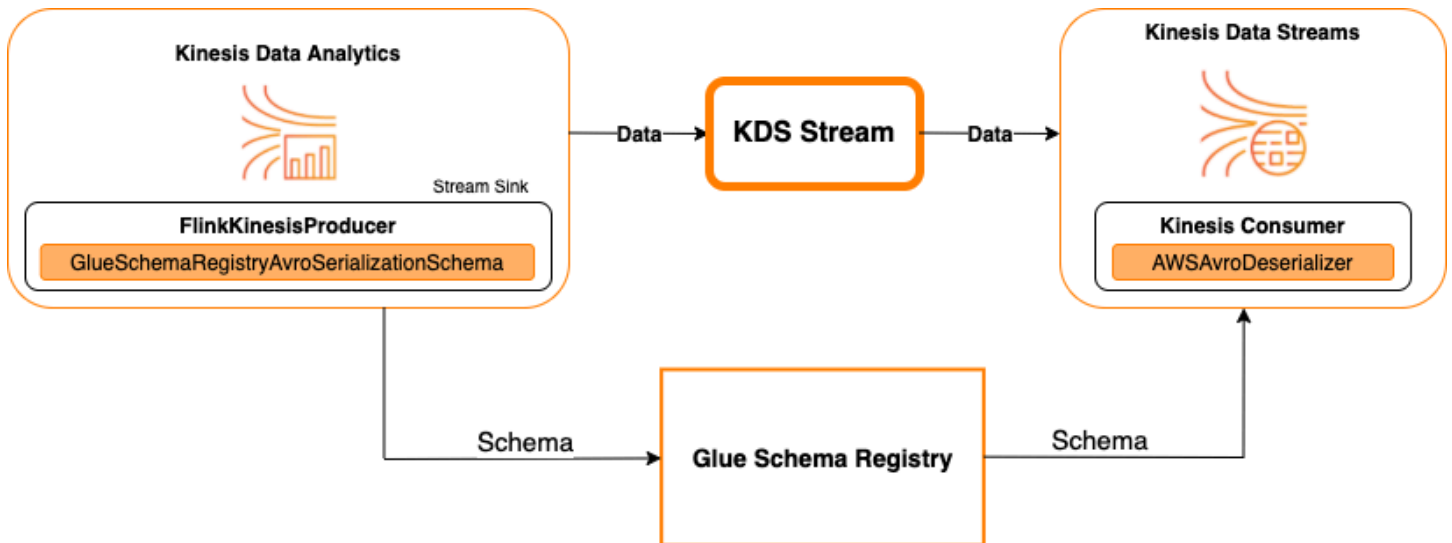
소스로 Kinesis Data Streams

다음 다이어그램에서는 Kinesis Data Streams를 소스로 사용하여 Kinesis Data Streams를 Apache Flink용 Managed Service for Apache Flink와 통합하는 방법을 보여줍니다.



싱크로 Kinesis Data Streams

다음 다이어그램에서는 Kinesis Data Streams를 싱크로 사용하여 Kinesis Data Streams를 Apache Flink용 Managed Service for Apache Flink와 통합하는 방법을 보여줍니다.



Kinesis Data Streams를 소스 또는 싱크로 사용하여 Kinesis Data Streams를 Apache Flink용 Managed Service for Apache Flink와 통합하려면 아래와 같이 코드를 변경합니다. 굵게 표시된 코드 블록을 유사한 섹션의 해당 코드에 추가합니다.

Kinesis Data Streams가 소스인 경우 deserializer 코드(블록 2)를 사용합니다. Kinesis Data Streams가 싱크인 경우 serializer 코드(블록 3)를 사용합니다.

```

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

String streamName = "stream";
Properties consumerConfig = new Properties();
consumerConfig.put(AWSConfigConstants.AWS_REGION, "aws-region");
consumerConfig.put(AWSConfigConstants.AWS_ACCESS_KEY_ID, "aws_access_key_id");
consumerConfig.put(AWSConfigConstants.AWS_SECRET_ACCESS_KEY, "aws_secret_access_key");
consumerConfig.put(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "LATEST");

// block 1
Map<String, Object> configs = new HashMap<>();
configs.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
configs.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
configs.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
AvroRecordType.GENERIC_RECORD.getName());

FlinkKinesisConsumer<GenericRecord> consumer = new FlinkKinesisConsumer<>(
    streamName,
    // block 2
GlueSchemaRegistryAvroDeserializationSchema.forGeneric(schema, configs),
    properties);
  
```

```

FlinkKinesisProducer<GenericRecord> producer = new FlinkKinesisProducer<>(
    // block 3
    GlueSchemaRegistryAvroSerializationSchema.forGeneric(schema, topic, configs),
    properties);
producer.setDefaultStream(streamName);
producer.setDefaultPartition("0");

DataStream<GenericRecord> stream = env.addSource(consumer);
stream.addSink(producer);
env.execute();

```

사용 사례: AWS Lambda와 통합

AWS Lambda 함수를 Apache Kafka/Amazon MSK 소비자로 사용하고 AWS Glue Schema Registry를 사용하여 Avro 인코딩 메시지를 역직렬화하려면 [MSK Labs 페이지](#)를 방문합니다.

사용 사례: AWS Glue Data Catalog

AWS Glue 테이블은 수동으로 지정하거나 AWS Glue Schema Registry를 참조하여 지정할 수 있는 스키마를 지원합니다. Schema Registry는 데이터 카탈로그와 통합되어 데이터 카탈로그에서 AWS Glue 테이블 또는 파티션을 생성하거나 업데이트할 때 Schema Registry에 저장된 스키마를 선택적으로 사용할 수 있습니다. Schema Registry에서 스키마 정의를 식별하려면 최소한 스키마가 속한 스키마의 ARN을 알아야 합니다. 스키마 정의를 포함하는 스키마의 스키마 버전은 해당 UUID 또는 버전 번호로 참조할 수 있습니다. 버전 번호나 UUID를 몰라도 조회할 수 있는 "최신" 버전이라는 하나의 스키마 버전이 항상 있습니다.

CreateTable 또는 UpdateTable 작업을 호출할 때 Schema Registry의 기존 스키마에 SchemaReference가 있을 수 있는 StorageDescriptor를 포함하는 TableInput 구조를 전달합니다. 마찬가지로 GetTable 또는 GetPartition API를 호출할 때 응답에 스키마와 SchemaReference가 포함될 수 있습니다. 스키마 참조를 사용하여 테이블 또는 파티션이 생성되면 데이터 카탈로그는 이 스키마 참조에 대한 스키마를 가져오려고 시도합니다. Schema Registry에서 스키마를 찾을 수 없는 경우 GetTable 응답에서 빈 스키마를 반환합니다. 그렇지 않으면 응답에 스키마와 스키마 참조가 모두 포함됩니다.

AWS Glue 콘솔에서 작업을 수행할 수도 있습니다.

이러한 작업을 수행하고 스키마 정보를 생성, 업데이트 또는 보려면 GetSchemaVersion API에 대한 권한을 제공하는 호출 사용자에게 IAM 역할을 부여해야 합니다.

테이블 추가 또는 테이블에 대한 스키마 업데이트

기존 스키마에서 새 테이블을 추가하면 테이블이 특정 스키마 버전에 바인딩됩니다. 새 스키마 버전이 등록되면 AWS Glue 콘솔의 테이블 보기 페이지에서 또는 [UpdateTable 액션 \(Python: 업데이트_테이블\)](#) API를 사용하여 이 테이블 정의를 업데이트할 수 있습니다.

기존 스키마에서 테이블 추가

AWS Glue 콘솔 또는 CreateTable API를 사용하여 레지스트리의 스키마 버전에서 AWS Glue 테이블을 생성할 수 있습니다.

AWS Glue API

CreateTable API를 호출할 때 Schema Registry의 기존 스키마에 SchemaReference가 있고 StorageDescriptor가 포함된 TableInput을 전달합니다.

AWS Glue 콘솔

AWS Glue 콘솔에서 테이블을 생성하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [테이블(Tables)]을 선택합니다.
3. [테이블 추가(Add Tables)] 메뉴에서 [기존 스키마에서 테이블 추가(Add table from existing schema)]를 선택합니다.
4. AWS Glue 개발자 안내서에 따라 테이블 속성 및 데이터 스토어를 구성합니다.
5. [Glue 스키마 선택(Choose a Glue schema)] 페이지에서 스키마가 있는 [레지스트리(Registry)]를 선택합니다.
6. [스키마 이름(Schema name)]을 선택하고 적용할 스키마의 [버전(Version)]을 선택합니다.
7. 스키마 미리 보기를 검토하고 [다음(Next)]을 선택합니다.
8. 테이블을 검토하고 생성합니다.

테이블에 적용된 스키마 및 버전은 테이블 목록의 [Glue 스키마(Glue schema)] 열에 나타납니다. 테이블을 보고 자세한 세부 정보를 확인할 수 있습니다.

테이블에 대한 스키마 업데이트

새 스키마 버전을 사용할 수 있게 되면 [UpdateTable 액션 \(Python: 업데이트_테이블\)](#) API 또는 AWS Glue 콘솔을 사용하여 테이블의 스키마를 업데이트할 수 있습니다.

⚠ Important

AWS Glue 스키마가 수동으로 지정된 기존 테이블의 스키마를 업데이트할 때 Schema Registry에서 참조하는 새 스키마가 호환되지 않을 수 있습니다. 이로 인해 작업이 실패할 수 있습니다.

AWS Glue API

UpdateTable API를 호출할 때 Schema Registry의 기존 스키마에 SchemaReference가 있고 StorageDescriptor가 포함된 TableInput을 전달합니다.

AWS Glue 콘솔

AWS Glue 콘솔에서 테이블에 대한 스키마를 업데이트하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 [데이터 카탈로그(Data catalog)]에서 [테이블(Tables)]을 선택합니다.
3. 테이블 목록에서 테이블을 봅니다.
4. 새 버전에 대해 알려주는 상자에서 [스키마 업데이트(Update schema)]를 클릭합니다.
5. 현재 스키마와 새 스키마의 차이점을 검토합니다.
6. [모든 스키마 차이점 표시(Show all schema differences)]를 선택 더 많은 세부 정보를 볼 수 있습니다.
7. [테이블 저장(Save table)]을 선택하여 새 버전을 수락합니다.

사용 사례: AWS Glue 스트리밍

AWS Glue 스트리밍은 출력 싱크에 쓰기 전에 스트리밍 소스의 데이터를 소비하고 ETL 작업을 수행합니다. 입력 스트리밍 소스는 데이터 테이블을 사용하여 지정하거나 소스 구성을 지정하여 직접 지정할 수 있습니다.

AWS Glue 스트리밍은 에 AWS Glue 스키마 레지스트리에 있는 스키마를 사용하여 생성된 스트리밍 소스에 대한 데이터 카탈로그 테이블을 지원합니다. AWS Glue 스키마 레지스트리에서 스키마를 생성하고 이 스키마를 사용하는 스트리밍 소스가 있는 AWS Glue 테이블을 생성할 수 있습니다. 이 AWS Glue 테이블을 입력 스트림의 데이터를 역직렬화하기 위한 AWS Glue 스트리밍 작업에 대한 입력으로 사용할 수 있습니다.

여기서 유의해야 할 점은 AWS Glue 스키마 레지스트리의 스키마가 변경되면 AWS Glue 스트리밍 작업을 다시 시작해야 스키마의 변경 사항이 반영된다는 것입니다.

사용 사례: Apache Kafka Streams

Apache Kafka Streams API는 Apache Kafka에 저장된 데이터를 처리하고 분석하기 위한 클라이언트 라이브러리입니다. 이 섹션에서는 데이터 스트리밍 애플리케이션에서 스키마를 관리하고 시행할 수 있는 AWS Glue Schema Registry와 Apache Kafka Streams의 통합에 대해 설명합니다. Apache Kafka Streams에 대한 자세한 내용은 [Apache Kafka Streams](#)를 참조하세요.

SerDes 라이브러리와 통합

Streams 애플리케이션을 구성할 수 있는 GlueSchemaRegistryKafkaStreamsSerde 클래스가 있습니다.

Kafka Streams 애플리케이션 예 코드

Apache Kafka Streams 애플리케이션 내에서 AWS Glue Schema Registry를 사용하려면

1. Kafka Streams 애플리케이션을 구성합니다.

```
final Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "avro-streams");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass().getName());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
AWSKafkaAvroSerDe.class.getName());
    props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

    props.put(AWSSchemaRegistryConstants.AWS_REGION, "aws-region");
    props.put(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING, true);
    props.put(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
AvroRecordType.GENERIC_RECORD.getName());
    props.put(AWSSchemaRegistryConstants.DATA_FORMAT, DataFormat.AVRO.name());
```

2. 주제 avro-input에서 스트림을 생성합니다.

```
StreamsBuilder builder = new StreamsBuilder();
final KStream<String, GenericRecord> source = builder.stream("avro-input");
```

3. 데이터 레코드를 처리합니다(이 예에서는 favorite_color 값이 pink이거나 양의 값이 15인 레코드를 필터링함).

```
final KStream<String, GenericRecord> result = source
    .filter((key, value) -
    > !"pink".equals(String.valueOf(value.get("favorite_color"))));
    .filter((key, value) -> !"15.0".equals(String.valueOf(value.get("amount"))));
```

4. avro-output 주제에 결과를 다시 씁니다.

```
result.to("avro-output");
```

5. Apache Kafka Streams 애플리케이션을 시작합니다.

```
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.start();
```

구현 결과

이 결과는 3단계에서 favorite_color "pink" 또는 값 "15.0"으로 필터링된 레코드의 필터링 프로세스를 보여줍니다.

필터링 전 레코드:

```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
{"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
{"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
{"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}
{"name": "Jay", "favorite_number": 0, "favorite_color": "pink"}

{"id": "commute_1", "amount": 3.5}
{"id": "grocery_1", "amount": 25.5}
{"id": "entertainment_1", "amount": 19.2}
{"id": "entertainment_2", "amount": 105}
{"id": "commute_1", "amount": 15}
```

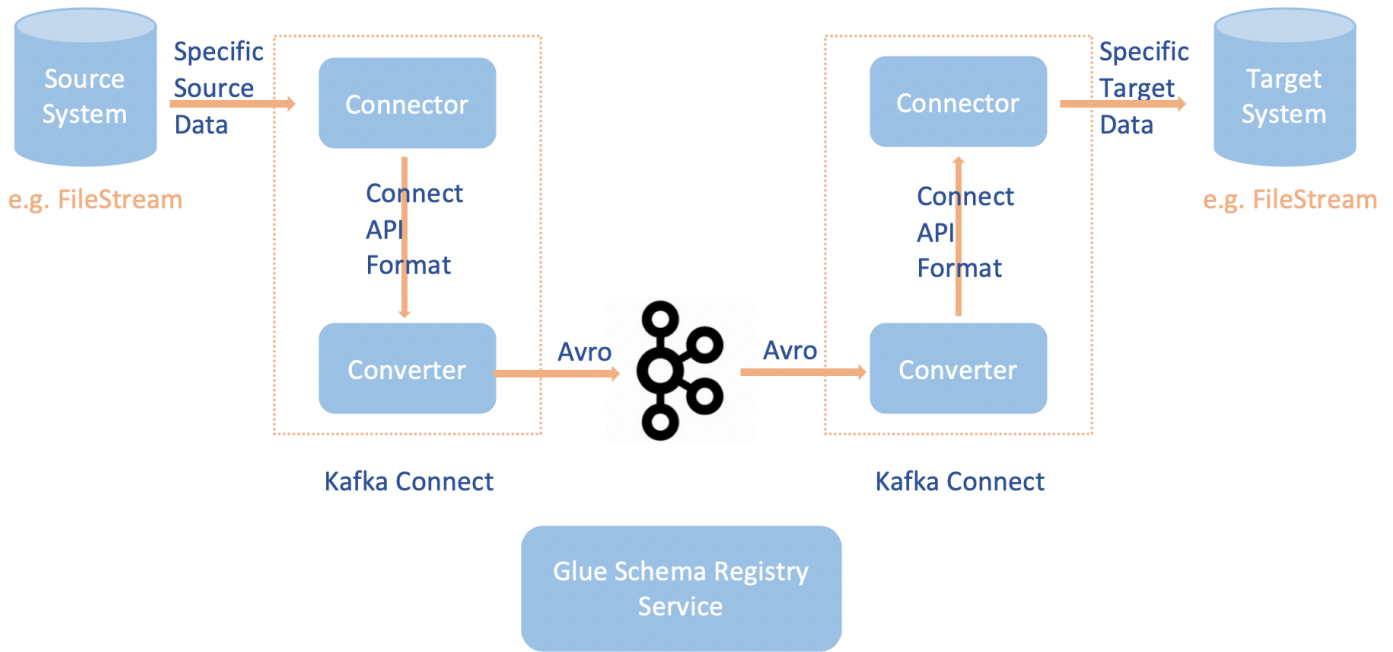

필터링 후 레코드:

```
{"name": "Sansa", "favorite_number": 99, "favorite_color": "white"}
{"name": "Harry", "favorite_number": 10, "favorite_color": "black"}
{"name": "Hermione", "favorite_number": 1, "favorite_color": "red"}
{"name": "Ron", "favorite_number": 0, "favorite_color": "pink"}

{"id": "commute_1", "amount": 3.5}
{"id": "grocery_1", "amount": 25.5}
{"id": "entertainment_1", "amount": 19.2}
{"id": "entertainment_2", "amount": 105}
```

사용 사례: Apache Kafka Connect

Apache Kafka Connect와 AWS Glue Schema Registry의 통합으로 커넥터에서 스키마 정보를 가져올 수 있습니다. Apache Kafka 변환기는 Apache Kafka 내의 데이터 포맷과 이를 Apache Kafka Connect 데이터로 변환하는 방법을 지정합니다. 모든 Apache Kafka Connect 사용자는 Apache Kafka에서 로드하거나 Apache Kafka에 저장할 때 데이터를 원하는 포맷을 기반으로 이러한 변환기를 구성해야 합니다. 이러한 방식으로 Apache Kafka Connect 데이터를 AWS Glue Schema Registry(예: Avro)에서 사용되는 유형으로 변환하고 serializer를 사용하여 스키마를 등록하고 직렬화를 수행하는 변환기를 정의할 수 있습니다. 그런 다음 변환기는 deserializer를 사용하여 Apache Kafka에서 수신한 데이터를 역직렬화하고 이를 다시 Apache Kafka Connect 데이터로 변환할 수도 있습니다. 다음은 예제 워크플로 다이어그램입니다.



1. [AWS Glue Schema Registry용 Github 리포지토리](#)를 복제하여 aws-glue-schema-registry 프로젝트를 설치합니다.

```
git clone git@github.com:aws-labs/aws-glue-schema-registry.git
cd aws-glue-schema-registry
mvn clean install
mvn dependency:copy-dependencies
```

2. 독립 실행형 모드에서 Apache Kafka Connect를 사용하려는 경우 이 단계에 대한 아래 지침을 사용하여 connect-standalone.properties를 업데이트합니다. 분산 모드에서 Apache Kafka Connect를 사용하려는 경우 동일한 지침을 사용하여 connect-avro-distributed.properties를 업데이트합니다.

- a. Apache Kafka 연결 속성 파일에도 다음 속성을 추가합니다.

```
key.converter.region=aws-region
value.converter.region=aws-region
key.converter.schemaAutoRegistrationEnabled=true
value.converter.schemaAutoRegistrationEnabled=true
key.converter.avroRecordType=GENERIC_RECORD
value.converter.avroRecordType=GENERIC_RECORD
```

- b. kafka-run-class.sh 아래의 [시작 모드(Launch mode)] 섹션에 아래 명령을 추가합니다.

```
-cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
```

3. kafka-run-class.sh 아래의 [시작 모드(Launch mode)] 섹션에 아래 명령을 추가합니다.

```
-cp $CLASSPATH:"<your AWS GlueSchema Registry base directory>/target/dependency/*"
```

형식은 다음과 같아야 합니다.

```
# Launch mode
if [ "$DAEMON_MODE" = "xtrue" ]; then
    nohup "$JAVA" $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS
    $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp $CLASSPATH:"/Users/johndoe/aws-glue-schema-
    registry/target/dependency/*" $KAFKA_OPTS "$@" > "$CONSOLE_OUTPUT_FILE" 2>&1 < /dev/
    null &
else
    exec "$JAVA" $KAFKA_HEAP_OPTS $KAFKA_JVM_PERFORMANCE_OPTS $KAFKA_GC_LOG_OPTS
    $KAFKA_JMX_OPTS $KAFKA_LOG4J_OPTS -cp $CLASSPATH:"/Users/johndoe/aws-glue-schema-
    registry/target/dependency/*" $KAFKA_OPTS "$@"
fi
```

4. bash를 사용하는 경우 아래 명령을 실행하여 bash_profile에 CLASSPATH를 설정합니다. 다른 셸의 경우 그에 따라 환경을 업데이트합니다.

```
echo 'export GSR_LIB_BASE_DIR=<>' >> ~/.bash_profile
echo 'export GSR_LIB_VERSION=1.0.0' >> ~/.bash_profile
echo 'export KAFKA_HOME=<your Apache Kafka installation directory>' >> ~/.bash_profile
echo 'export CLASSPATH=$CLASSPATH:$GSR_LIB_BASE_DIR/avro-kafkaconnect-converter/
target/schema-registry-kafkaconnect-converter-$GSR_LIB_VERSION.jar:$GSR_LIB_BASE_DIR/
common/target/schema-registry-common-$GSR_LIB_VERSION.jar:$GSR_LIB_BASE_DIR/
avro-serializer-deserializer/target/schema-registry-serde-$GSR_LIB_VERSION.jar'
>> ~/.bash_profile
source ~/.bash_profile
```

5. (선택 사항) 간단한 파일 소스로 테스트하려면 파일 소스 커넥터를 복제합니다.

```
git clone https://github.com/mmolimar/kafka-connect-fs.git
cd kafka-connect-fs/
```

- a. 소스 커넥터 구성에서 데이터 포맷을 Avro로, 파일 리더를 AvroFileReader으로 편집하고 읽고 있는 파일 경로에서 예제 Avro 객체를 업데이트합니다. 예:

```
vim config/kafka-connect-fs.properties
```

```
fs.uris=<path to a sample avro object>
policy.regex=^.*\.avro$
file_reader.class=com.github.mmolimar.kafka.connect.fs.file.reader.AvroFileReader
```

- b. 소스 커넥터를 설치합니다.

```
mvn clean package
echo "export CLASSPATH=\$CLASSPATH:\\"$(find target/ -type f -name '*.jar'| grep
'\-package' | tr '\n' ':')\\"" >> ~/.bash_profile
source ~/.bash_profile
```

- c. *<your Apache Kafka installation directory>/config/connect-file-sink.properties* 아래의 싱크 속성을 업데이트하고 주제 이름과 출력 파일 이름을 업데이트합니다.

```
file=<output file full path>
topics=<my topic>
```

6. 소스 커넥터를 시작합니다 (이 예에서는 파일 소스 커넥터임).

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-standalone.properties config/kafka-connect-fs.properties
```

7. 싱크 커넥터를 실행합니다(이 예에서는 파일 싱크 커넥터임).

```
$KAFKA_HOME/bin/connect-standalone.sh $KAFKA_HOME/config/connect-standalone.properties $KAFKA_HOME/config/connect-file-sink.properties
```

Kafka Connect 사용 예는 [AWS Glue Schema Registry용 Github 리포지토리](#)의 integration-tests 폴더에 있는 run-local-tests.sh 스크립트를 참조하세요.

서드 파티 Schema Registry에서 AWS Glue Schema Registry로 마이그레이션

서드 파티 Schema Registry에서 AWS Glue Schema Registry로의 마이그레이션은 기존의 현재 서드 파티 Schema Registry에 종속됩니다. Apache Kafka 주제에 서드 파티 스키마 레지스트리를 사용하여 전송된 레코드가 있는 경우 소비자는 해당 레코드를 역직렬화하기 위해 서드 파티 스키마 레지스트리가 필요합니다. `AWSKafkaAvroDeserializer`는 서드 파티 deserializer를 가리키고 해당 레코드를 역직렬화하는 데 사용되는 보조 deserializer 클래스를 지정하는 기능을 제공합니다.

서드 파티 스키마의 사용 중지에는 두 가지 기준이 있습니다. 첫째, 서드 파티 스키마 레지스트리를 사용하는 Apache Kafka 주제의 레코드가 소비자에게 더 이상 필요하지 않은 경우에만 사용 중지가 발생할 수 있습니다. 둘째, 해당 주제에 대해 지정된 보존 기간에 따라 Apache Kafka 주제에서 만료로 사용 중지가 발생할 수 있습니다. 영구 보존이 있는 주제가 있는 경우 AWS Glue Schema Registry로 계속 마이그레이션할 수 있지만 서드 파티 Schema Registry를 사용 중지할 수는 없습니다. 해결 방법으로 애플리케이션 또는 Mirror Maker 2를 사용하여 현재 주제에서 읽고 AWS Glue Schema Registry를 사용하여 새 주제를 생성할 수 있습니다.

서드 파티 Schema Registry에서 AWS Glue Schema Registry로 마이그레이션하려면

1. AWS Glue Schema Registry에서 레지스트리를 생성하거나 기본 레지스트리를 사용합니다.
2. 소비자를 중지합니다. AWS Glue Schema Registry를 기본 deserializer로 포함하고 서드 파티 Schema Registry를 보조로 포함하도록 수정합니다.
 - 소비자 속성을 설정합니다. 이 예에서 `secondary_deserializer`는 다른 deserializer로 설정됩니다. 동작은 다음과 같습니다. 소비자는 Amazon MSK에서 레코드를 검색하고 먼저 `AWSKafkaAvroDeserializer`를 사용하려고 시도합니다. AWS Glue Schema Registry 스키마에 대한 Avro 스키마 ID가 포함된 매직 바이트를 읽을 수 없는 경우 `AWSKafkaAvroDeserializer`는 `secondary_deserializer`에서 제공되는 deserializer 클래스를 사용하려고 시도합니다. 보조 deserializer와 관련된 속성도 아래와 같이 `schema_registry_url_config` 및 `specific_avro_reader_config`와 같은 소비자 속성에 제공해야 합니다.

```
consumerProps.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
    StringDeserializer.class.getName());
consumerProps.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
    AWSKafkaAvroDeserializer.class.getName());
consumerProps.setProperty(AWSSchemaRegistryConstants.AWS_REGION,
    KafkaClickstreamConsumer.gsrRegion);
```

```
consumerProps.setProperty(AWSSchemaRegistryConstants.SECONDARY_DESERIALIZER,
    KafkaAvroDeserializer.class.getName());
consumerProps.setProperty(KafkaAvroDeserializerConfig.SCHEMA_REGISTRY_URL_CONFIG,
    "URL for third-party schema registry");
consumerProps.setProperty(KafkaAvroDeserializerConfig.SPECIFIC_AVRO_READER_CONFIG,
    "true");
```

3. 소비자를 다시 시작합니다.
4. 생산자를 중지하고 생산자가 AWS Glue Schema Registry를 가리키도록 합니다.
 - a. 생산자 속성을 설정합니다. 이 예에서 생산자는 기본 레지스트리 및 자동 등록 스키마 버전을 사용합니다.

```
producerProps.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    StringSerializer.class.getName());
producerProps.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    AWSKafkaAvroSerializer.class.getName());
producerProps.setProperty(AWSSchemaRegistryConstants.AWS_REGION, "us-east-2");
producerProps.setProperty(AWSSchemaRegistryConstants.AVRO_RECORD_TYPE,
    AvroRecordType.SPECIFIC_RECORD.getName());
producerProps.setProperty(AWSSchemaRegistryConstants.SCHEMA_AUTO_REGISTRATION_SETTING,
    "true");
```

5. (선택 사항) 기존 스키마 및 스키마 버전을 현재 서드 파티 Schema Registry에서 AWS Glue Schema Registry로, AWS Glue Schema Registry의 기본 레지스트리 또는 AWS Glue Schema Registry의 기본이 아닌 특정 레지스트리로 수동으로 이동합니다. 이는 JSON 포맷의 서드 파티 Schema Registry에서 스키마를 내보내고 AWS Management Console 또는 AWS CLI를 사용하여 AWS Glue Schema Registry에서 새 스키마를 생성하여 수행할 수 있습니다.

이 단계는 AWS CLI 및 AWS Management Console을 사용하여 새로 생성된 스키마 버전에 대해 이전 스키마 버전과의 호환성 검사를 사용해야 하거나 생산자가 스키마 버전 자동 등록이 설정된 상태로 새 스키마로 메시지를 보낼 때 중요할 수 있습니다.

6. 생산자를 시작합니다.

데이터에 연결

AWS Glue 연결은 특정 데이터 저장소의 로그인 자격 증명, URI 문자열, 가상 사설 클라우드 (VPC) 정보 등을 저장하는 데이터 카탈로그 개체입니다. AWS Glue 크롤러, 작업 및 개발 엔드포인트는 연결을 사용하여 특정 유형의 데이터 저장소에 액세스합니다. 소스와 대상 모두에 대한 연결을 사용하고 여러 크롤러 또는 추출, 전환, 적재(ETL) 작업에서 동일한 연결을 재사용할 수 있습니다.

AWS Glue 다음과 같은 연결 유형을 지원합니다.

- Amazon DocumentDB
- Amazon OpenSearch 서비스, Spark와 함께 AWS Glue 사용할 수 있습니다.
- Amazon Redshift
- Azure Cosmos, ETL 작업과 함께 NoSQL용 Azure Cosmos DB를 사용하는 경우 AWS Glue
- Azure SQL, 스파크에 사용할 수 있습니다. AWS Glue
- 구글에서 BigQuery 스파크와 함께 AWS Glue 사용할 수 있습니다.
- JDBC
- Kafka
- MongoDB
- MongoDB Atlas
- Salesforce
- AWS Glue 스파크용으로 사용할 수 있는 SAP HANA.
- 스노우플레이크, 스파크용. AWS Glue
- 테라데이터 밴티지 (스�파크에 사용할 경우) AWS Glue
- 버티카, 스파크용. AWS Glue
- 각종 Amazon Relational Database Service(RDS) 제안.
- 네트워크(Amazon Virtual Private Cloud(Amazon VPC)에 있는 데이터 소스에 대한 연결 지정)
- Aurora(기본 JDBC 드라이버를 사용하는 경우 지원됨, 일부 드라이버 기능은 사용할 수 없음)

AWS Glue Studio를 사용하면 커넥터에 대한 연결을 생성할 수도 있습니다. 커넥터는 AWS Glue Studio의 데이터 스토어에 액세스하는 데 도움이 되는 선택적 코드 패키지입니다. 자세한 내용은 [AWS Glue Studio에서 커넥터 및 연결 사용](#)을 참조하세요.

온프레미스 데이터베이스에 연결하는 방법에 대한 자세한 내용은 [AWS 빅데이터 블로그 웹 사이트](#)를 사용하여 온프레미스 데이터 AWS Glue 저장소에 액세스하고 분석하는 방법을 참조하십시오.

이 섹션에는 AWS Glue 연결 사용에 도움이 되는 다음 주제가 포함되어 있습니다.

- [AWS Glue 연결 속성](#)
- [AWS Secrets Manager에 연결 자격 증명 저장](#)
- [AWS Glue 연결 추가](#)
- [AWS Glue 연결 테스트](#)
- [AWS 호출이 VPC를 통과하도록 구성](#)
- [VPC의 JDBC 데이터 스토어로 연결](#)
- [MongoDB 또는 MongoDB Atlas 연결 사용](#)
- [VPC 엔드포인트를 사용한 Amazon S3 데이터 스토어 크롤링](#)
- [AWS Glue 연결 문제 해결](#)
- [자습서: AWS Glue Connector for Elasticsearch 사용](#)

AWS Glue 연결 속성

이 항목에는 AWS Glue 연결 속성에 대한 정보가 포함되어 있습니다.

주제

- [필수 연결 속성](#)
- [AWS Glue JDBC 연결 속성](#)
- [AWS Glue MongoDB 및 MongoDB Atlas 연결 속성](#)
- [Salesforce 연결 속성](#)
- [Snowflake 연결](#)
- [Vertica 연결](#)
- [SAP HANA 연결](#)
- [Azure SQL 연결](#)
- [Teradata Vantage 연결](#)
- [OpenSearch 서비스 연결](#)
- [Azure Cosmos 연결](#)

- [AWS Glue SSL 연결 속성](#)
- [클라이언트 인증을 위한 Apache Kafka 연결 속성](#)
- [구글 BigQuery 커넥션](#)
- [Vertica 연결](#)

필수 연결 속성

AWS Glue 콘솔에서 연결을 정의할 때 다음 속성에 대한 값을 제공해야 합니다.

연결 이름

연결에 고유한 이름을 입력합니다.

연결 유형

JDBC 또는 특정 연결 유형 중 하나를 선택합니다.

JDBC 연결 유형에 대한 자세한 내용은 [the section called “JDBC 연결 속성”](#) 단원을 참조하십시오.

[네트워크(Network)]를 선택하여 Amazon Virtual Private Cloud 환경(Amazon VPC) 내의 데이터 원본에 연결합니다.

선택한 유형에 따라 AWS Glue 콘솔에 다른 필수 필드가 표시됩니다. 예를 들어 [Amazon RDS]를 선택한 경우 데이터베이스 엔진을 선택해야 합니다.

필요한 SSL 연결

이 옵션을 선택하면 AWS Glue에서 데이터 스토어 연결이 신뢰할 수 있는 보안 소켓 계층(SSL)을 통해 이루어졌는지 확인해야 합니다.

이 옵션을 선택할 때 사용할 수 있는 추가 옵션을 비롯한 자세한 내용은 [the section called “SSL 연결 속성”](#) 단원을 참조하십시오.

MSK 클러스터 선택(Amazon Managed Streaming for Apache Kafka(MSK)만 해당)

다른 AWS 계정의 MSK 클러스터를 지정합니다.

Kafka 부트스트랩 서버 URL(Kafka만 해당)

쉼표로 구분된 부트스트랩 서버 URL 목록을 지정합니다. 포트 번호를 포함합니다. 예: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-2.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-3.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094

AWS Glue JDBC 연결 속성

AWS Glue는 JDBC 연결을 통해 다음 데이터 스토어에 연결할 수 있습니다.

- Amazon Redshift
- Amazon Aurora
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- 스노우플레이크, 크롤러를 사용하는 경우 AWS Glue
- Aurora(기본 JDBC 드라이버를 사용하는 경우 지원됨, 일부 드라이버 기능은 사용할 수 없음)
- Amazon RDS for MariaDB

Important

현재 ETL 작업은 하나의 서브넷 내에서만 JDBC 연결을 사용할 수 있습니다. 작업에서 다양한 데이터 스토어가 있으면 모두 동일한 서브넷에 있거나 서브넷에서 액세스 가능해야 합니다. AWS Glue 크롤러에 대해 자체 JDBC 드라이버 버전을 가져오기로 선택한 경우 크롤러는 AWS Glue 작업 및 Amazon S3의 리소스를 소비하여 제공된 드라이버가 사용자 환경에서 실행 되도록 합니다. 리소스의 추가 사용량은 계정에 반영됩니다. 또한 자체 JDBC 드라이버를 제공한다고 해서 크롤러가 해당 드라이버의 모든 기능을 사용할 수 있는 것은 아닙니다. 드라이버는 [데이터 카탈로그의 연결 정의](#)에 설명된 속성으로 제한됩니다.

다음은 JDBC 연결 유형에 대한 추가 속성입니다.

JDBC URL

JDBC 데이터 스토어의 URL을 입력합니다. 대부분 데이터베이스 엔진의 경우, 이 필드는 다음과 같은 포맷입니다. 이 포맷에서 *protocol*, *host*, *port*, *db_name*을 고유한 정보로 바꿉니다.

```
jdbc:protocol://host:port/db_name
```

데이터베이스 엔진에 따라 다른 JDBC URL 포맷이 필요합니다. 이 포맷은 콜론(:), 슬래시(/), 및 기타 키워드를 다르게 사용하여 데이터베이스를 지정합니다.

JDBC를 데이터 스토어에 연결하는 데 필요한 데이터 스토어의 db_name입니다. db_name는 지원한 username 및 password를 통해 네트워크 연결을 만드는 데 사용됩니다. 연결되면 AWS Glue는 데이터 스토어의 다른 데이터베이스에 액세스할 수 있어 크롤러나 ETL 작업을 실행할 수 있습니다.

다음 JDBC URL 예제에서는 몇 가지 데이터베이스 엔진의 구문을 보여 줍니다.

- dev 데이터베이스로 Amazon Redshift 클러스터 데이터 스토어에 연결하려면

```
jdbc:redshift://xxx.us-east-1.redshift.amazonaws.com:8192/dev
```

- employee 데이터베이스로 Amazon RDS for MySQL 데이터 스토어에 연결하려면

```
jdbc:mysql://xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:3306/employee
```

- employee 데이터베이스로 Amazon RDS for PostgreSQL 데이터 스토어에 연결하려면

```
jdbc:postgresql://xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:5432/employee
```

- employee 서비스 이름으로 Amazon RDS for Oracle 데이터 스토어에 연결하려면

```
jdbc:oracle:thin://@xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:1521/employee
```

Amazon RDS for Oracle의 구문은 다음 패턴을 따를 수 있습니다. 이러한 패턴에서 *host*, *port*, *service_name*, *SID*를 고유한 정보로 바꿉니다.

- `jdbc:oracle:thin://@host:port/service_name`
- `jdbc:oracle:thin://@host:port:SID`
- employee 데이터베이스로 Amazon RDS for Microsoft SQL Server 데이터 스토어에 연결하려면

```
jdbc:sqlserver://xxx-cluster.cluster-xxx.us-east-1.rds.amazonaws.com:1433;databaseName=employee
```

Amazon RDS for SQL Server의 구문은 다음 패턴을 따를 수 있습니다. 이러한 패턴에서 *server_name*, *port* 및 *db_name*을 사용자 자신의 정보로 바꿉니다.


- `jdbc:sqlserver://server_name:port;database=db_name`
- `jdbc:sqlserver://server_name:port;databaseName=db_name`

- 데이터베이스 Amazon Aurora PostgreSQL 인스턴스에 연결하려면 employee 데이터베이스 인스턴스의 엔드포인트, 포트, 데이터베이스 이름을 지정합니다.

```
jdbc:postgresql://employee_instance_1.xxxxxxxxxxxxx.us-east-2.rds.amazonaws.com:5432/employee
```

- 데이터베이스가 있는 Amazon RDS for MariaDB 데이터 저장소에 연결하려면 employee 데이터베이스 인스턴스의 엔드포인트, 포트 및 데이터베이스 이름을 지정합니다.


```
jdbc:mysql://xxx-cluster.cluster-xxx.aws-region.rds.amazonaws.com:3306/employee
```

 Warning

Snowflake JDBC 연결은 크롤러에서만 지원됩니다. AWS Glue 작업에 스노우플레이크 커넥터를 사용할 때는 스노우플레이크 연결 유형을 사용하십시오. AWS Glue

sample 데이터베이스의 Snowflake 인스턴스에 연결하려면 Snowflake 인스턴스의 엔드포인트, 사용자, 데이터베이스 이름 및 역할 이름을 지정합니다. warehouse 파라미터를 선택적으로 추가할 수 있습니다.

```
jdbc:snowflake://account_name.snowflakecomputing.com/?user=user_name&db=sample&role=role_name&warehouse=warehouse_name
```

 Important

JDBC를 통한 Snowflake 연결의 경우 URL의 파라미터 순서가 적용되므로, user, db, role_name, warehouse의 순서로 정렬되어야 합니다.

- AWS 개인 링크를 사용하여 sample 데이터베이스의 Snowflake 인스턴스에 연결하려면 다음과 같이 눈송이 JDBC URL을 지정하십시오.

```
jdbc:snowflake://account_name.region.privatelink.snowflakecomputing.com/?user=user_name&db=sample&role=role_name&warehouse=warehouse_name
```

사용자 이름

Note

사용자 이름과 암호를 직접 제공하는 대신 AWS 암호를 사용하여 연결 자격 증명을 저장하는 것이 좋습니다. 자세한 정보는 [AWS Secrets Manager에 연결 자격 증명 저장](#)을 참조하세요.

JDBC 데이터 스토어 연결 권한을 갖는 사용자 이름을 제공합니다.

암호

JDBC 데이터 스토어에 대한 액세스 권한이 있는 사용자 이름의 비밀번호를 입력합니다.

Port

Amazon RDS Oracle 인스턴스에 연결하기 위해 JDBC URL에 사용된 포트를 입력합니다. 이 필드는 Amazon RDS Oracle 인스턴스에 대해 [SSL 연결 필요(Require SSL connection)]가 선택된 경우에만 표시됩니다.

VPC

데이터 스토어를 포함한 Virtual Private Cloud(VPC) 이름을 선택합니다. AWS Glue 콘솔은 현재 리전의 모든 VPC를 나열합니다.

Important

Snowflake의 AWS데이터와 같이 호스팅되는 JDBC 연결을 통해 작업하는 경우 VPC에 트래픽을 퍼블릭 서브넷과 프라이빗 서브넷으로 분할하는 NAT 게이트웨이가 있어야 합니다. 퍼블릭 서브넷은 외부 소스에 연결하는 데 사용되고 내부 서브넷은 에서 처리하는 데 사용됩니다. AWS Glue외부 연결을 위해 Amazon VPC를 구성하는 방법에 대한 자세한 내용은 [NAT 디바이스를 사용하여 인터넷 또는 다른 네트워크에 연결](#) 및 [AWS Glue에서 Amazon RDS 데이터 스토어에 대해 JDBC를 연결하도록 Amazon VPC 설정](#) 섹션을 참조하세요.

서브넷

데이터 스토어를 포함하는 VPC 내 서브넷을 선택합니다. AWS Glue 콘솔은 VPC에 데이터 스토어에 대한 모든 서브넷을 나열합니다.

보안 그룹

데이터 스토어와 관련된 보안 그룹을 선택합니다. AWS Glue에는 연결할 AWS Glue를 허용하는 인바운드 소스 규칙과 함께 하나 이상의 보안 그룹이 필요합니다. AWS Glue 콘솔은 VPC에 인바운드 액세스를 보장하는 보안 그룹을 나열합니다. AWS Glue는 VPC 서브넷에 연결된 탄력적 네트워크 인터페이스를 통해 보안 그룹이 연결됩니다.

JDBC 드라이버 클래스 이름 - 선택 사항

사용자 지정 JDBC 드라이버 클래스 이름을 제공합니다.

- Postgres – org.postgresql.Driver
- MySQL – com.mysql.jdbc.Driver, com.mysql.cj.jdbc.Driver
- Redshift – com.amazon.redshift.jdbc.Driver, com.amazon.redshift.jdbc42.Driver
- 오라클 — oracle.jdbc.driver. OracleDriver
- SQL 서버 — com.Microsoft.sqlServer.jdbc.sql ServerDriver

JDBC 드라이버 S3 경로 - 선택 사항

사용자 지정 JDBC 드라이버에 Amazon S3 위치를 제공합니다. 이는 .jar 파일의 절대 경로입니다. 크롤러 지원 데이터베이스에서 데이터 소스에 연결하기 위해 자체 JDBC 드라이버를 제공하려는 경우 customJdbcDriverS3Path 및 customJdbcDriverClassName 파라미터에 대한 값을 지정할 수 있습니다.

고객이 제공한 JDBC 드라이버의 사용은 필요한 [필수 연결 속성](#)으로만 제한됩니다.

AWS Glue MongoDB 및 MongoDB Atlas 연결 속성

다음은 MongoDB 또는 MongoDB Atlas 연결 유형에 대한 추가 속성입니다.

MongoDB URL

MongoDB 또는 MongoDB Atlas 데이터 스토어의 URL을 입력합니다.

- MongoDB의 경우: mongodb://host:port/database. 호스트는 호스트 이름, IP 주소 또는 UNIX 도메인 소켓일 수 있습니다. 연결 문자열이 포트를 지정하지 않는 경우 기본 MongoDB 포트인 27017을 사용합니다.

- MongoDB Atlas의 경우: mongodb+srv://server.example.com/database. 호스트는 DNS SRV 레코드에 해당하는 호스트 이름일 수 있습니다. SRV 형식에는 포트가 필요하지 않으며 기본 MongoDB 포트인 27017을 사용합니다.

사용자 이름

Note

사용자 이름과 암호를 직접 제공하는 대신 AWS 암호를 사용하여 연결 자격 증명을 저장하는 것이 좋습니다. 자세한 정보는 [AWS Secrets Manager에 연결 자격 증명 저장](#)을 참조하세요.

JDBC 데이터 스토어 연결 권한을 갖는 사용자 이름을 제공합니다.

암호

MongoDB 또는 MongoDB Atlas 데이터 스토어에 대한 액세스 권한이 있는 사용자 이름에 대한 비밀번호를 입력합니다.

Salesforce 연결 속성

다음은 Salesforce 연결 유형의 추가 속성입니다.

- ENTITY_NAME(문자열) - (필수) 읽기/쓰기에 사용됩니다. Salesforce에 있는 개체의 이름.
- API_VERSION(문자열) - (필수) 읽기/쓰기에 사용됩니다. 사용하려는 세일즈포스 레스트 API 버전입니다.
- SELECTED_FIELDS(목록<String>) - 기본값: 비어 있음 (SELECT *). 읽기에 사용됩니다. 개체에 대해 선택하려는 열
- FILTER_PREDICATE(문자열) - 기본값: 비어 있습니다. 읽기에 사용됩니다. 스파크 SQL 형식이어야 합니다.
- QUERY(문자열) - 기본값: 비어 있습니다. 읽기에 사용됩니다. 전체 스파크 SQL 쿼리
- PARTITION_FIELD(문자열) - 읽기에 사용됩니다. 쿼리를 분할하는 데 사용할 필드입니다.
- LOWER_BOUND(문자열) - 읽기에 사용됩니다. 선택한 파티션 필드의 포함된 하한 값입니다.
- UPPER_BOUND(문자열) - 읽기에 사용됩니다. 선택한 파티션 필드의 배타적 상한값입니다.
- NUM_PARTITIONS(정수) - 디폴트 값: 1. 읽기에 사용됩니다. 읽을 파티션 수

- `IMPORT_DELETED_RECORDS`(문자열) - 기본값: `FALSE`. 읽기에 사용됩니다. 쿼리하는 동안 삭제 기록을 가져오려는 경우
- `WRITE_OPERATION`(문자열) - 기본값: 삽입. 쓰기에 사용됩니다. 값은 삽입, 업데이트, 업서트, 삭제여야 합니다.
- `ID_FIELD_NAMES`(문자열) - 기본값: `null`. UPSERT에만 사용됩니다.

Snowflake 연결

다음 속성은 ETL 작업에 사용되는 Snowflake 연결을 설정하는 데 사용됩니다. AWS Glue Snowflake를 크롤링할 때는 JDBC 연결을 사용합니다.

Snowflake URL

Snowflake 엔드포인트의 URL입니다. Snowflake 엔드포인트 URL에 대한 자세한 내용은 Snowflake 설명서의 [Connecting to Your Accounts](#)를 참조하세요.

AWS 시크릿

비밀의 비밀 이름 AWS Secrets Manager. AWS Glue 시크릿의 `sfUser` 및 `sfPassword` 키를 사용하여 스노우플레이크에 연결합니다.

Snowflake 역할(선택 사항)

연결 시 스노우플레이크 보안 역할이 사용됩니다 AWS Glue .

AWS PrivateLink를 사용하여 Amazon VPC에 호스팅되는 Snowflake 엔드포인트에 대한 연결을 구성할 때 다음 속성을 사용합니다.

VPC

데이터 스토어를 포함한 Virtual Private Cloud(VPC) 이름을 선택합니다. AWS Glue 콘솔은 현재 리전의 모든 VPC를 나열합니다.

서브넷

데이터 스토어를 포함하는 VPC 내 서브넷을 선택합니다. AWS Glue 콘솔은 VPC에 데이터 스토어에 대한 모든 서브넷을 나열합니다.

보안 그룹

데이터 스토어와 관련된 보안 그룹을 선택합니다. AWS Glue에는 연결할 AWS Glue를 허용하는 인바운드 소스 규칙과 함께 하나 이상의 보안 그룹이 필요합니다. AWS Glue 콘솔은 VPC에 인바운드

액세스를 보장하는 보안 그룹을 나열합니다. AWS Glue는 VPC 서브넷에 연결된 탄력적 네트워크 인터페이스를 통해 보안 그룹이 연결됩니다.

Vertica 연결

다음 속성을 사용하여 ETL 작업을 위한 AWS Glue Vertica 연결을 설정합니다.

Vertica 호스트

Vertica 설치의 호스트 이름.

Vertica 포트

Vertica 설치를 사용할 수 있는 포트입니다.

AWS 시크릿

비밀의 비밀 이름 AWS Secrets Manager. AWS Glue 시크릿 키를 사용하여 Vertica에 연결합니다.

Amazon VPC에 호스팅되는 Vertica 엔드포인트에 대한 연결을 구성할 때 다음 속성을 사용합니다.

VPC

데이터 스토어를 포함한 Virtual Private Cloud(VPC) 이름을 선택합니다. AWS Glue 콘솔은 현재 리전의 모든 VPC를 나열합니다.

서브넷

데이터 스토어를 포함하는 VPC 내 서브넷을 선택합니다. AWS Glue 콘솔은 VPC에 데이터 스토어에 대한 모든 서브넷을 나열합니다.

보안 그룹

데이터 스토어와 관련된 보안 그룹을 선택합니다. AWS Glue에는 연결할 AWS Glue를 허용하는 인바운드 소스 규칙과 함께 하나 이상의 보안 그룹이 필요합니다. AWS Glue 콘솔은 VPC에 인바운드 액세스를 보장하는 보안 그룹을 나열합니다. AWS Glue는 VPC 서브넷에 연결된 탄력적 네트워크 인터페이스를 통해 보안 그룹이 연결됩니다.

SAP HANA 연결

다음 속성을 사용하여 AWS Glue ETL 작업을 위한 SAP HANA 연결을 설정합니다.

SAP HANA URL

SAP JDBC URL.

SAP HANA JDBC URL은

`jdbc:sap://saphanaHostname:saphanaPort/?databaseName=saphanaDBname,ParameterName`
형식입니다

AWS Glue 다음과 같은 JDBC URL 매개 변수가 필요합니다.

- `databaseName` - 연결할 SAP HANA의 기본 데이터베이스입니다.

AWS 시크릿

비밀의 비밀 이름 AWS Secrets Manager. AWS Glue 시크릿 키를 사용하여 SAP HANA에 연결합니다.

를 사용하여 Amazon VPC에 호스팅되는 SAP HANA 엔드포인트에 대한 연결을 구성할 때 다음 속성을 사용합니다.

VPC

데이터 스토어를 포함한 Virtual Private Cloud(VPC) 이름을 선택합니다. AWS Glue 콘솔은 현재 리전의 모든 VPC를 나열합니다.

서브넷

데이터 스토어를 포함하는 VPC 내 서브넷을 선택합니다. AWS Glue 콘솔은 VPC에 데이터 스토어에 대한 모든 서브넷을 나열합니다.

보안 그룹

데이터 스토어와 관련된 보안 그룹을 선택합니다. AWS Glue에는 연결할 AWS Glue를 허용하는 인바운드 소스 규칙과 함께 하나 이상의 보안 그룹이 필요합니다. AWS Glue 콘솔은 VPC에 인바운드 액세스를 보장하는 보안 그룹을 나열합니다. AWS Glue는 VPC 서브넷에 연결된 탄력적 네트워크 인터페이스를 통해 보안 그룹이 연결됩니다.

Azure SQL 연결

다음 속성을 사용하여 AWS Glue ETL 작업에 대한 Azure SQL 연결을 설정합니다.

Azure SQL URL

Azure SQL 엔드포인트의 JDBC URL입니다.

목록은

```
jdbc:sqlserver://databaseServerName:databasePort;databaseName=azuresqlDBname;
```

형식이어야 합니다.

AWS Glue 다음과 같은 URL 속성이 필요합니다.

- `databaseName` - 연결할 Azure SQL의 기본 데이터베이스입니다.

Azure SQL 관리형 인스턴스용 JDBC URL에 대한 자세한 내용은 [Microsoft 설명서](#)를 참조하십시오.

AWS 시크릿

비밀의 비밀 이름 AWS Secrets Manager. AWS Glue 시크릿 키를 사용하여 Azure SQL에 연결합니다.

Teradata Vantage 연결

다음 속성을 사용하여 ETL 작업에 대한 테라데이타 Vantage 연결을 설정합니다. AWS Glue

Teradata URL

Teradata 인스턴스에 연결하려면 데이터베이스 인스턴스의 호스트 이름과 관련 Teradata 매개변수를 지정합니다.

```
jdbc:teradata://teradataHostname/ParameterName=ParameterValue,ParameterName=Pa
```

AWS Glue 다음과 같은 JDBC URL 매개변수를 지원합니다.

- `DATABASE_NAME` - 연결할 Teradata의 기본 데이터베이스입니다.
- `DBS_PORT` - Teradata 포트(비표준인 경우)를 지정합니다.

AWS 시크릿

비밀의 비밀 이름 AWS Secrets Manager. AWS Glue 시크릿 키를 사용하여 테라데이타 밴티지에 연결합니다.

Amazon VPC에 호스팅되는 Teradata Vantage 엔드포인트에 대한 연결을 구성할 때 다음 속성을 사용합니다.

VPC

데이터 스토어를 포함한 Virtual Private Cloud(VPC) 이름을 선택합니다. AWS Glue 콘솔은 현재 리전의 모든 VPC를 나열합니다.

서브넷

데이터 스토어를 포함하는 VPC 내 서브넷을 선택합니다. AWS Glue 콘솔은 VPC에 데이터 스토어에 대한 모든 서브넷을 나열합니다.

보안 그룹

데이터 스토어와 관련된 보안 그룹을 선택합니다. AWS Glue에는 연결할 AWS Glue를 허용하는 인바운드 소스 규칙과 함께 하나 이상의 보안 그룹이 필요합니다. AWS Glue 콘솔은 VPC에 인바운드 액세스를 보장하는 보안 그룹을 나열합니다. AWS Glue는 VPC 서브넷에 연결된 탄력적 네트워크 인터페이스를 통해 보안 그룹이 연결됩니다.

OpenSearch 서비스 연결

다음 속성을 사용하여 AWS Glue ETL 작업에 대한 OpenSearch 서비스 연결을 설정합니다.

도메인 엔드포인트

Amazon OpenSearch 서비스 도메인 엔드포인트의 기본 형식은 `https://search-#####-unstructuredIdContent##.es.amazonaws.com`입니다. 도메인 엔드포인트 식별에 대한 자세한 내용은 Amazon 서비스 설명서의 [Amazon OpenSearch Service 도메인 생성 및 관리](#)를 참조하십시오. OpenSearch

Port

엔드포인트에서 포트가 열립니다.

AWS 비밀

비밀의 비밀 이름 AWS Secrets Manager. AWS Glue 시크릿 키를 사용하여 OpenSearch 서비스에 연결합니다.

Amazon VPC에 호스팅된 OpenSearch 서비스 엔드포인트에 대한 연결을 구성할 때 다음 속성을 사용하십시오.

VPC

데이터 스토어를 포함한 Virtual Private Cloud(VPC) 이름을 선택합니다. AWS Glue 콘솔은 현재 리전의 모든 VPC를 나열합니다.

서브넷

데이터 스토어를 포함하는 VPC 내 서브넷을 선택합니다. AWS Glue 콘솔은 VPC에 데이터 스토어에 대한 모든 서브넷을 나열합니다.

보안 그룹

데이터 스토어와 관련된 보안 그룹을 선택합니다. AWS Glue에는 연결할 AWS Glue를 허용하는 인바운드 소스 규칙과 함께 하나 이상의 보안 그룹이 필요합니다. AWS Glue 콘솔은 VPC에 인바운드 액세스를 보장하는 보안 그룹을 나열합니다. AWS Glue는 VPC 서브넷에 연결된 탄력적 네트워크 인터페이스를 통해 보안 그룹이 연결됩니다.

Azure Cosmos 연결

다음 속성을 사용하여 ETL 작업을 위한 Azure Cosmos 연결을 설정합니다. AWS Glue

Azure Cosmos DB 계정 엔드포인트 URI

Amazon Cosmos에 연결하는 데 사용된 엔드포인트. 자세한 내용은 [Azure 설명서](#)를 참조하십시오.

AWS 시크릿

비밀의 비밀 이름 AWS Secrets Manager. AWS Glue 시크릿 키를 사용하여 Azure Cosmos에 연결합니다.

AWS Glue SSL 연결 속성

다음은 Require SSL connection(SSL 연결 필요) 속성에 대한 세부 정보입니다.

SSL 연결이 필요하지 않은 경우 AWS Glue에서 SSL을 사용하여 데이터 스토어에 대한 연결을 암호화할 때 실패를 무시합니다. 구성 지침은 데이터 스토어의 문서를 참조하세요. 이 옵션을 선택하면 AWS Glue에서 연결할 수 없을 때 개발 엔드포인트의 작업 실행, 크롤러 또는 ETL 문이 실패합니다.

Note

Snowflake는 기본적으로 SSL 연결을 지원하므로 이 속성은 Snowflake에 적용되지 않습니다.

이 옵션은 AWS Glue 클라이언트 측에서 검증됩니다. JDBC 연결의 경우 AWS Glue는 인증서 및 호스트 이름 검증을 사용하여 SSL을 통해서만 연결합니다. SSL 연결 지원은 다음에 사용할 수 있습니다.

- Oracle Database
- Microsoft SQL Server
- PostgreSQL
- Amazon Redshift
- MySQL(Amazon RDS 인스턴스만 해당)
- Amazon Aurora MySQL(Amazon RDS 인스턴스만 해당)
- Amazon Aurora PostgreSQL (Amazon RDS 인스턴스만 해당)
- 카프카, 여기에는 다음이 포함됩니다. Amazon Managed Streaming for Apache Kafka
- MongoDB

Note

Amazon RDS Oracle 데이터 스토어에서 [SSL 연결 필요(Require SSL connection)]를 사용할 수 있도록 하려면 옵션 그룹을 생성하여 Oracle 인스턴스에 연결해야 합니다.

1. AWS Management Console 로그인하고 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. [옵션 그룹(Option group)]을 Amazon RDS Oracle 인스턴스에 추가합니다. Amazon RDS 콘솔에서 옵션 그룹을 추가하는 방법에 대한 자세한 내용은 [옵션 그룹 생성](#)을 참조하세요.
3. 옵션을 SSL에 대한 옵션 그룹에 추가합니다. SSL에 대해 지정하는 [포트(Port)]는 나중에 Amazon RDS Oracle 인스턴스에 대한 AWS Glue JDBC 연결 URL을 생성할 때 사용됩니다. Amazon RDS 콘솔에서 옵션을 추가하는 방법에 대한 자세한 내용은 Amazon RDS User Guide의 [Adding an Option to an Option Group](#)을 참조하세요. Oracle SSL 옵션에 대한 자세한 내용은 Amazon RDS User Guide의 [Oracle SSL](#)을 참조하세요.
4. AWS Glue 콘솔에서 Amazon RDS Oracle 인스턴스에 대한 연결을 생성합니다. 연결 정의에서 SSL 연결 필요를 선택합니다. 요청 시 Amazon RDS Oracle SSL 옵션에서 사용한 [포트(Port)]를 입력합니다.

연결에 대해 [SSL 연결 필요(Require SSL connection)]를 선택한 경우 다음과 같은 추가 선택적 속성을 사용할 수 있습니다.

S3의 사용자 지정 JDBC 인증서

현재 온프레미스 또는 클라우드 데이터베이스와의 SSL 통신에 사용 중인 인증서가 있는 경우 AWS Glue 데이터 원본 또는 대상에 대한 SSL 연결에 해당 인증서를 사용할 수 있습니다. 사용자 정의 루트 인증서가 포함된 Amazon Simple Storage Service(Amazon S3) 위치를 입력합니다. AWS Glue는 이 인증서를 사용하여 데이터베이스에 대한 SSL 연결을 설정합니다. AWS Glue는 X.509 인증서만 처리합니다. 인증서는 DER로 인코딩되고 base64 인코딩 PEM 형식으로 제공되어야 합니다.

이 필드를 비워두면 기본 인증서가 사용됩니다.

사용자 지정 JDBC 인증서 문자열

JDBC 데이터베이스에 특정한 인증서 정보를 입력합니다. 이 문자열은 도메인 일치 또는 고유 이름(DN) 일치에 사용됩니다. Oracle Database의 경우 이 문자열은 tnsnames.ora 파일의 보안 섹션에 있는 SSL_SERVER_CERT_DN 파라미터에 매핑됩니다. Microsoft SQL Server의 경우 이 문자열은 hostNameInCertificate로 사용됩니다.

다음은 Oracle Database SSL_SERVER_CERT_DN 파라미터의 예입니다.

```
cn=sales,cn=OracleContext,dc=us,dc=example,dc=com
```

Kafka 프라이빗 CA 인증서 위치

Kafka 데이터 스토어와의 SSL 통신에 현재 사용 중인 인증서가 있는 경우 해당 인증서를 AWS Glue 연결과 함께 사용할 수 있습니다. 이 옵션은 Kafka 데이터 스토어에는 필수이고 데이터 스토어에는 선택 사항입니다. Amazon Managed Streaming for Apache Kafka 사용자 정의 루트 인증서가 포함된 Amazon Simple Storage Service(Amazon S3) 위치를 입력합니다. AWS Glue는 이 인증서를 사용하여 Kafka 데이터 스토어에 대한 SSL 연결을 설정합니다. AWS Glue는 X.509 인증서만 처리합니다. 인증서는 DER로 인코딩되고 base64 인코딩 PEM 형식으로 제공되어야 합니다.

인증서 검증 건너뛰기

AWS Glue의 사용자 정의 인증서 검증을 건너뛰려면 [인증서 검증 건너뛰기(Skip certificate validation)] 확인란을 선택합니다. 검증하기로 한 경우, AWS Glue가 인증서의 서명 알고리즘 및 주제 퍼블릭 키 알고리즘을 검증합니다. 인증서가 검증에 실패하면, 연결을 사용하는 모든 ETL 작업 또는 크롤러가 실패합니다.

허용되는 유일한 서명 알고리즘은 SHA256withRSA, SHA384withRSA 또는 SHA512withRSA. 주제 퍼블릭 키 알고리즘의 경우, 키 길이는 2048 이상이어야 합니다.

Kafka 클라이언트 키 스토어 위치

Kafka 클라이언트 측 인증을 위한 클라이언트 키 스토어 파일의 Amazon S3 위치입니다. 경로는 `s3://bucket/prefix/filename.jks` 형식이어야 합니다. 파일 이름과 `.jks` 확장자로 끝나야 합니다.

Kafka 클라이언트 키 스토어 암호(선택 사항)

제공된 키 스토어에 액세스하기 위한 암호입니다.

Kafka 클라이언트 키 암호(선택 사항)

키 스토어는 여러 키로 구성 될 수 있으므로 Kafka 서버 측 키와 함께 사용할 클라이언트 키에 액세스하기 위한 암호입니다.

클라이언트 인증을 위한 Apache Kafka 연결 속성

AWS Glue는 Apache Kafka 연결을 생성할 때 인증을 위한 SASL(Simple Authentication and Security Layer) 프레임워크를 지원합니다. SASL 프레임워크는 다양한 인증 메커니즘을 지원하며, AWS Glue는 SCRAM(사용자 이름 및 암호), GSSAPI(Kerberos 프로토콜), PLAIN 프로토콜을 제공합니다.

다음 클라이언트 인증 AWS Glue Studio 방법 중 하나를 구성하는 데 사용합니다. 자세한 내용은 [내용은 사용 AWS Glue Studio 설명서의 커넥터 연결 만들기를](#) 참조하십시오.

- 없음(None) - 인증이 없습니다. 테스트 목적으로 연결을 생성하는 경우에 유용합니다.
- SASL/SCRAM-SHA-512 - 이 인증 방법을 선택하면 인증 자격 증명을 지정할 수 있습니다. 두 가지 옵션을 사용할 수 있습니다.
 - AWS Secrets Manager 사용 (권장) - 이 옵션을 선택하면 사용자 이름과 암호를 AWS Secrets Manager에 저장하고 필요할 때 AWS Glue 액세스할 수 있습니다. SSL 또는 SASL 인증 자격 증명을 저장하는 보안 암호를 지정합니다. 자세한 정보는 [AWS Secrets Manager에 연결 자격 증명 저장](#)을 참조하세요.
 - 사용자 이름과 암호를 직접 제공합니다.
- SASL/GSSAPI (Kerberos)(SASL/GSSAPI(Kerberos)) - 이 옵션을 선택하면 keytab 파일, krb5.conf 파일의 위치를 선택하고 Kerberos 보안 주체 이름과 Kerberos 서비스 이름을 입력할 수 있습니다. keytab 파일과 krb5.conf 파일의 위치는 Amazon S3 위치에 있어야 합니다. MSK는 아직 SASL/GSSAPI를 지원하지 않으므로 이 옵션은 고객 관리형 Apache Kafka 클러스터에만 사용할 수 있습니다. 자세한 내용은 [MIT Kerberos Documentation: Keytab](#)(MIT Kerberos 설명서: Keytab)을 참조하세요.
- SASL/PLAIN - 인증 자격 증명을 지정하려면 이 인증 방법을 선택합니다. 두 가지 옵션을 사용할 수 있습니다.

- AWS Secrets Manager 사용 (권장) - 이 옵션을 선택하면 자격 증명을 AWS Secrets Manager에 저장하고 필요할 때 정보에 AWS Glue 액세스할 수 있습니다. SSL 또는 SASL 인증 자격 증명을 저장하는 보안 암호를 지정합니다.
- 사용자 이름 및 암호를 직접 제공합니다.
- SSL 클라이언트 인증(SSL Client Authentication) - 이 옵션을 선택하면 Amazon S3를 검색하여 Kafka 클라이언트 키 스토어의 위치를 선택할 수 있습니다. 선택 사항으로 Kafka 클라이언트 키 스토어 암호와 Kafka 클라이언트 키 암호를 입력할 수 있습니다.

구글 BigQuery 커넥션

AWS Glue ETL 작업에 사용되는 Google BigQuery 연결을 설정하는 데 사용되는 속성은 다음과 같습니다. 자세한 정보는 [the section called “BigQuery 연결”](#)을 참조하세요.

AWS 시크릿

비밀의 비밀 이름 AWS Secrets Manager. AWS Glue ETL 작업은 시크릿 credentials 키를 BigQuery 사용하여 Google에 연결됩니다.

Vertica 연결

다음 속성은 AWS Glue ETL 작업에 사용되는 Vertica 연결을 설정하는 데 사용됩니다. 자세한 내용은 [the section called “수직 연결”](#)을(를) 참조하세요.

AWS Secrets Manager에 연결 자격 증명 저장

AWS Secrets Manager를 사용하여 데이터 스토어에 대한 연결 자격 증명을 제공하는 것이 좋습니다. 이런 식으로 Secrets Manager를 사용하면 AWS Glue가 런타임에 ETL 작업 및 크롤러 실행을 위해 암호에 액세스할 수 있고 자격 증명을 안전하게 유지하는 데 도움이 됩니다.

사전 조건

Secrets Manager를 AWS Glue와 함께 사용하려면 [AWS Glue에 대한 IAM 역할](#)에 암호 값을 검색할 수 있는 권한을 부여해야 합니다. AWS 관리형 정책 AWSGlueServiceRole에는 AWS Secrets Manager 권한이 포함되어 있지 않습니다. 예제 IAM 정책을 알아보려면 AWS Secrets Manager 사용 설명서의 [예: 보안 암호 값을 검색할 수 있는 권한](#)을 참조하세요.

네트워크 설정에 따라 VPC와 Secrets Manager 간의 프라이빗 연결을 설정하기 위해 VPC 엔드포인트를 생성해야 할 수도 있습니다. 자세한 내용을 알아보려면 [AWS Secrets Manager VPC 엔드포인트 사용](#)을 참조하세요.

AWS Glue에 대한 보안 암호를 생성하려면

1. AWS Secrets Manager 사용 설명서의 [보안 암호 생성 및 관리](#)의 지침을 따르세요. 다음 예제 JSON은 AWS Glue용 암호를 생성할 때 Plaintext 탭에서 자격 증명을 지정하는 방법을 보여 줍니다.

```
{
  "username": "EXAMPLE-USERNAME",
  "password": "EXAMPLE-PASSWORD"
}
```

2. AWS Glue Studio 인터페이스를 사용하여 보안 암호를 연결과 연결합니다. 자세한 지침은 AWS Glue Studio 사용 설명서의 [커넥터에 대한 연결 생성](#)을 참조하세요.

AWS Glue 연결 추가

Spark에 대한 AWS Glue를 프로그래밍 방식으로 데이터 소스에 연결할 수 있습니다. 자세한 내용은 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#) 단원을 참조하세요.

AWS Glue 콘솔을 사용하여 연결을 추가, 편집, 삭제 및 테스트할 수도 있습니다. AWS Glue 연결에 대한 자세한 내용은 [데이터에 연결](#) 단원을 참조하십시오.

AWS Glue 연결을 추가하려면

1. 에 AWS Management Console 로그인하고 <https://console.aws.amazon.com/glue/> 에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 데이터 카탈로그에서 연결을 선택합니다.
3. 연결 추가를 선택한 다음 [the section called “AWS Glue 연결 속성”](#)에 설명된 대로 연결 속성을 입력하여 마법사를 완료합니다.

아마존 Redshift에 연결 AWS Glue Studio

Note

AWS GlueSpark를 사용하여 외부 Amazon Redshift 데이터베이스의 테이블에서 읽고 쓸 수 있습니다. AWS Glue Studio 프로그래밍 방식으로 AWS Glue 작업을 구성하려면 Amazon Redshift 을 참조하십시오. [Redshift 연결](#)

AWS Glue에 대한 Amazon Redshift 기본 지원을 제공합니다. AWS Glue Studio AWS Glue Studio 서버리스 Spark 런타임에서 연결하고 Amazon Redshift, 데이터 통합 작업을 작성하고, 실행할 수 있는 시각적 인터페이스를 제공합니다.

주제

- [Amazon Redshift 연결 만들기](#)
- [Amazon Redshift 소스 노드 생성](#)
- [Amazon Redshift 대상 노드 생성](#)
- [고급 옵션](#)

Amazon Redshift 연결 만들기

필요한 권한

Amazon Redshift 클러스터와 Amazon Redshift 서버리스 환경을 사용하려면 추가 권한이 필요합니다. ETL 작업에 권한을 추가하는 방법에 대한 자세한 내용은 [ETL 작업에 필요한 IAM 권한 검토](#)를 참조하십시오.

- 레드시프트: DescribeClusters
- 레드시프트-서버리스: ListWorkgroups
- 레드시프트-서버리스: ListNamespaces

개요

Amazon Redshift 연결을 추가할 때 데이터 소스 - Redshift 노드를 추가할 때 기존 Amazon Redshift 연결을 선택하거나 새 연결을 만들 수 있습니다. AWS Glue Studio

AWS Glue Amazon Redshift 클러스터와 Amazon Redshift 서버리스 환경을 모두 지원합니다. 연결을 만들면 Amazon Redshift 서버리스 환경은 연결 옵션 옆에 서버리스 레이블을 표시합니다.

Amazon Redshift 연결을 만드는 방법에 대한 자세한 내용은 [데이터 간 이동을](#) 참조하십시오. Amazon Redshift

Amazon Redshift 소스 노드 생성

필요한 권한

Amazon Redshift 데이터 소스를 사용하는 AWS Glue Studio 작업에는 추가 권한이 필요합니다. ETL 작업에 권한을 추가하는 방법에 대한 자세한 내용은 [ETL 작업에 필요한 IAM 권한 검토](#)를 참조하세요.

Amazon Redshift 연결을 사용하려면 다음 권한이 필요합니다.

- redshift-data:ListSchemas
- redshift-data:ListTables
- redshift-data:DescribeTable
- redshift-data:ExecuteStatement
- redshift-data:DescribeStatement
- redshift-data:GetStatementResult

Amazon Redshift 데이터 소스 추가

데이터 소스 - Amazon Redshift 노드를 추가하려면:

1. Amazon Redshift 액세스 유형을 선택합니다.
 - 직접 데이터 연결(권장) - Amazon Redshift 데이터에 직접 액세스하려면 이 옵션을 선택합니다. 이 옵션은 권장 옵션이며 기본값이기도 합니다.
 - Data Catalog tables - 사용하려는 데이터 카탈로그 테이블이 있는 경우 이 옵션을 선택합니다.
2. 직접 데이터 연결을 선택하는 경우 Amazon Redshift 데이터 소스의 연결을 선택합니다. 이 경우 연결이 이미 존재하며 기존 연결 중에서 선택할 수 있다고 가정합니다. 연결을 생성해야 하는 경우 Redshift 연결 생성을 선택합니다. 자세한 내용은 [커넥터 및 연결 사용 개요](#)를 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다. URL, 보안 그룹, 서브넷, 가용 영역, 설명, 생성 날짜(UTC) 및 최종 업데이트(UTC) 타임스탬프를 비롯한 연결 정보가 표시 됩니다.

3. 다음과 같은 Amazon Redshift 소스 옵션을 선택합니다.
 - 단일 테이블 선택 - 단일 Amazon Redshift 테이블에서 액세스하려는 데이터가 들어 있는 테이블입니다.
 - 사용자 지정 쿼리 입력 - 사용자 지정 쿼리를 기반으로 여러 Amazon Redshift 테이블의 데이터 세트에 액세스할 수 있습니다.
4. 단일 테이블을 선택한 경우 Amazon Redshift 스키마를 선택합니다. 선택할 수 있는 스키마 목록은 선택한 테이블에 따라 결정됩니다.

또는 사용자 지정 쿼리 입력을 선택합니다. 여러 Amazon Redshift 테이블에서 사용자 지정 데이터 세트에 액세스하려면 이 옵션을 선택합니다. 이 옵션을 선택하는 경우 Amazon Redshift 쿼리를 입력합니다.

Amazon Redshift 서버리스 환경에 연결할 때 사용자 지정 쿼리에 다음 권한을 추가합니다.

```
GRANT SELECT ON ALL TABLES IN <schema> TO PUBLIC
```

스키마 추론을 선택하여 입력한 쿼리를 기반으로 스키마를 읽을 수 있습니다. Redshift 쿼리 편집기 열기를 선택하여 Amazon Redshift 쿼리를 입력할 수도 있습니다. 자세한 내용은 [쿼리 편집기를 사용하여 데이터베이스 쿼리](#)를 참조하세요.

5. 성능 및 보안에서 Amazon S3 스테이징 디렉터리와 IAM 역할을 선택합니다.
 - Amazon S3 스테이징 디렉터리 - 데이터를 임시로 스테이징할 Amazon S3 위치를 선택합니다.
 - IAM 역할 - 선택한 Amazon S3 위치에 기록할 수 있는 IAM 역할을 선택합니다.
6. 사용자 지정 Redshift 파라미터 - 선택 사항에서 파라미터와 값을 입력합니다.

Amazon Redshift 대상 노드 생성

필요한 권한

AWS Glue Studio Amazon Redshift 데이터 대상을 사용하는 작업에는 추가 권한이 필요합니다. ETL 작업에 권한을 추가하는 방법에 대한 자세한 내용은 [ETL 작업에 필요한 IAM 권한 검토](#)를 참조하세요.

Amazon Redshift 연결을 사용하려면 다음 권한이 필요합니다.

- 레드시프트 데이터: ListSchemas

- 적색편이 데이터: ListTables

대상 노드 추가 Amazon Redshift

Amazon Redshift 대상 노드를 만들려면:

1. 기존 Amazon Redshift 테이블을 대상으로 선택하거나 새 테이블 이름을 입력합니다.
2. 데이터 대상 - Redshift 대상 노드를 사용하는 경우 다음 옵션 중에서 선택할 수 있습니다.
 - 추가 - 테이블이 이미 있는 경우 모든 새 데이터를 테이블에 삽입으로 덤프합니다. 테이블이 없으면 새로 생성한 후 새 데이터를 모두 삽입합니다.

또한 대상 테이블의 기존 레코드를 업데이트(업서트)하려면 상자를 선택합니다. 테이블이 먼저 있어야 합니다. 그렇지 않으면 작업에 실패합니다.

 - 병합 - AWS Glue는 사용자가 지정한 조건에 따라 대상 테이블에 데이터를 추가하거나 업데이트합니다.

Note

에서 AWS Glue 병합 작업을 사용하려면 Amazon Redshift 병합 기능을 활성화해야 합니다. Amazon Redshift 인스턴스의 병합을 활성화하는 방법에 대한 지침은 [MERGE \(미리 보기\)](#) 를 참조하십시오.

다음과 같은 옵션을 선택합니다.

- 키 및 간단한 작업 선택 - 소스 데이터와 대상 데이터 세트 사이에서 일치하는 키로 사용할 열을 선택합니다.

일치하는 경우 다음 옵션을 지정합니다.

- 대상 데이터 세트의 레코드를 소스의 데이터로 업데이트합니다.
- 대상 데이터 세트에서 레코드를 삭제합니다.

일치하지 않는 경우 다음 옵션을 지정합니다.

- 소스 데이터를 대상 데이터 세트에 새 행으로 삽입합니다.
- 아무 작업 안 함.
- 사용자 지정 MERGE 명령문 입력 - 그런 다음 병합 명령문 검증을 선택하여 명령문이 유효한지 여부를 검증할 수 있습니다.

- **잘라내기** - 테이블이 이미 있는 경우 먼저 대상 테이블의 콘텐츠를 지워서 테이블 데이터를 잘라냅니다. 잘라내기에 성공하면 모든 데이터를 삽입합니다. 테이블이 없는 경우 테이블을 생성하고 모든 데이터를 삽입합니다. 잘라내기에 실패하면 작업에 실패합니다.
- **삭제** - 테이블이 이미 있는 경우 테이블 메타데이터와 데이터를 삭제합니다. 삭제에 성공하면 모든 데이터를 삽입합니다. 테이블이 없는 경우 테이블을 생성하고 모든 데이터를 삽입합니다. 삭제에 실패하면 작업에 실패합니다.
- **생성** - 기본 이름을 사용하여 새 테이블을 생성합니다. 테이블 이름이 이미 있는 경우 고유성을 유지하기 위해 이름에 `job_datetime`의 이름 접미사를 사용하는 새 테이블을 생성합니다. 그러면 모든 데이터가 새 테이블에 삽입됩니다. 테이블이 있는 경우 최종 테이블 이름에는 접미사가 추가됩니다. 테이블이 없는 경우 테이블이 생성됩니다. 어느 경우든 새 테이블이 생성됩니다.

고급 옵션

[Using the Amazon Redshift Spark connector on AWS Glue](#)를 참조하세요.

AWS Glue Studio에서 Azure Cosmos DB에 연결

AWS Glue에서는 Azure Cosmos DB를 기본으로 지원합니다. AWS Glue Studio에서는 NoSQL의 Azure Cosmos DB에 연결하고, 데이터 통합 작업을 작성하며, AWS Glue Studio 서버리스 Spark 런타임에서 실행할 수 있는 시각적 인터페이스를 제공합니다.

주제

- [Azure Cosmos DB 연결 생성](#)
- [Azure Cosmos DB 소스 노드 생성](#)
- [Azure Cosmos DB 대상 노드 생성](#)
- [고급 옵션](#)

Azure Cosmos DB 연결 생성

사전 조건:

- Azure에서는, AWS Glue, `cosmosKey`에서 사용할 Azure Cosmos DB 키를 식별하거나 생성해야 합니다. 자세한 내용은 Azure 설명서의 [Azure Cosmos DB의 데이터에 대한 보안 액세스](#)를 참조하십시오.

Azure Cosmos DB에 대한 연결 구성 방법:

1. AWS Secrets Manager에서 Azure Cosmos DB 키를 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *cosmosKey*가 포함된 키 `spark.cosmos.accountKey`에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때는 Azure Cosmos DB를 선택합니다.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.

Azure Cosmos DB 소스 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “Azure Cosmos DB 연결 생성”](#)에서 설명한 대로 AWS Secrets Manager 암호로 구성된 AWS Glue Azure Cosmos DB 연결입니다
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 읽으려는 NoSQL 컨테이너용 Azure Cosmos 데이터베이스. 컨테이너의 식별 정보가 필요합니다.

NoSQL용 Azure Cosmos 컨테이너는 해당 데이터베이스 및 컨테이너로 식별됩니다.

NoSQL API용 Azure Cosmos에 연결할 때 데이터베이스, *cosmosDBName*, 및 컨테이너, *cosmosContainerName*, 이름을 제공해야 합니다.

Azure Cosmos DB 데이터 소스 추가

데이터 소스 - Azure Cosmos DB 노드 추가하는 방법:

1. Azure Cosmos DB 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 Azure Cosmos DB 연결 생성을 선택합니다. 자세한 내용은 [이전the section called “Azure Cosmos DB 연결 생성”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.
2. Cosmos DB 데이터베이스 이름 선택 - 읽으려는 ##### *cosmosDBName*을 입력합니다.

3. Azure Cosmos DB 컨테이너 선택 - 읽으려는 컨테이너의 이름인 *cosmosContainerName*을 입력합니다.
4. 선택적으로 Azure Cosmos DB 사용자 지정 쿼리를 선택합니다. Azure Cosmos DB에서 특정 정보를 검색할 수 있는 SQL SELECT 쿼리를 제공합니다.
5. 사용자 지정 Azure Cosmos 속성(선택 사항)에서 필요한 경우 파라미터와 값을 입력합니다.

Azure Cosmos DB 대상 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “Azure Cosmos DB 연결 생성”](#)에서 설명한 대로 AWS Secrets Manager 암호로 구성된 AWS Glue Azure Cosmos DB 연결입니다
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 쓰려는 Azure Cosmos DB 테이블. 컨테이너의 식별 정보가 필요합니다. 연결 방법을 호출하기 전에 컨테이너를 만들어야 합니다.

NoSQL용 Azure Cosmos 컨테이너는 해당 데이터베이스 및 컨테이너로 식별됩니다.

NoSQL API용 Azure Cosmos에 연결할 때 데이터베이스, *cosmosDBName*, 및 컨테이너, *cosmosContainerName*, 이름을 제공해야 합니다.

Azure Cosmos DB 데이터 대상 추가

데이터 대상 - Azure Cosmos DB 노드 추가 방법:

1. Azure Cosmos DB 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 Azure Cosmos DB 연결 생성을 선택합니다. 자세한 내용은 [이전the section called “Azure Cosmos DB 연결 생성”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. Cosmos DB 데이터베이스 이름 선택 - 읽으려는 ##### *cosmosDBName*을 입력합니다.
3. Azure Cosmos DB 컨테이너 선택 - 읽으려는 컨테이너의 이름인 *cosmosContainerName*을 입력합니다.
4. 사용자 지정 Azure Cosmos 속성(선택 사항)에서 필요한 경우 파라미터와 값을 입력합니다.

고급 옵션

Azure Cosmos DB 노드를 생성할 때 고급 옵션을 제공할 수 있습니다. 이 옵션은 Spark 스크립트에 대한 AWS Glue를 프로그래밍할 때 사용할 수 있는 옵션과 동일합니다.

[the section called “Azure Cosmos DB 연결”](#)를 참조하세요.

AWS Glue Studio에서 Azure SQL에 연결

AWS Glue에서는 Azure SQL을 기본으로 지원합니다. AWS Glue Studio에서는 Azure SQL에 연결하고, 데이터 통합 작업을 작성하며, AWS Glue Studio 서버리스 Spark 런타임에서 실행할 수 있는 시각적 인터페이스를 제공합니다.

주제

- [Azure SQL 연결 만들기](#)
- [Azure SQL 소스 노드 생성](#)
- [Azure SQL 대상 노드 생성](#)
- [고급 옵션](#)

Azure SQL 연결 만들기

AWS Glue에서 Azure SQL에 연결하려면 Azure SQL 보안 인증 정보를 만들어 AWS Secrets Manager 암호에 저장한 다음 해당 암호를 Azure SQL AWS Glue 연결에 연결해야 합니다.

Azure SQL에 대한 연결을 구성하는 방법:

1. AWS Secrets Manager에서 Azure SQL 보안 인증을 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *azuresqlUsername*이 포함된 키 user에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *azuresqlPassword*가 포함된 키 password에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 Azure SQL를 선택합니다.

- Azure SQL URL을 제공할 때는 JDBC 엔드포인트 URL을 제공하십시오.

목록은

`jdbc:sqlserver://databaseServerName:databasePort;databaseName=azuresqlDBName`
형식이어야 합니다.

AWS Glue는 다음 URL 속성이 필요합니다.

- `databaseName` - 연결할 Azure SQL의 기본 데이터베이스입니다.

Azure SQL 관리형 인스턴스용 JDBC URL에 대한 자세한 내용은 [Microsoft 설명서](#)를 참조하십시오.

- AWS 보안 암호를 선택할 때 `secretName`을 입력합니다.

Azure SQL 소스 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “Azure SQL 연결 만들기”](#)에서 설명한 대로 AWS Secrets Manager 암호로 구성된 AWS Glue Azure SQL 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 읽으려는 Azure SQL 테이블. `tableName`.

Azure SQL 테이블은 데이터베이스, 스키마 및 테이블 이름으로 식별됩니다. Azure SQL에 연결할 때 데이터베이스 이름과 테이블 이름을 제공해야 합니다. 스키마가 기본값인 "public"이 아닌 경우에도 스키마를 제공해야 합니다. 데이터베이스는 `connectionName`의 URL 속성을, `dbtable`을 통해 스키마 및 테이블 이름을 제공 받습니다.

Azure SQL 데이터 소스 추가

데이터 소스 - Azure SQL 노드 추가 방법:

1. Azure SQL 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 Azure SQL 연결 생성을 선택합니다. 자세한 내용은 [이전 the section called “Azure SQL 연결 만들기”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. Azure SQL 소스 옵션을 선택합니다.

- 단일 테이블 선택 - 단일 테이블에서 모든 데이터에 액세스할 수 있습니다.
 - 사용자 지정 쿼리 입력 - 사용자 지정 쿼리를 기반으로 여러 테이블의 데이터 세트에 액세스할 수 있습니다.
3. 단일 테이블을 선택한 경우 *tableName*을 입력합니다.

사용자 지정 쿼리 입력을 선택한 경우 TransactSQL SELECT 쿼리를 입력합니다.
 4. 사용자 지정 Azure SQL 속성에서 필요한 경우 파라미터와 값을 입력합니다.

Azure SQL 대상 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “Azure SQL 연결 만들기”](#)에서 설명한 대로 AWS Secrets Manager 암호로 구성된 AWS Glue Azure SQL 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 쓰려는 Azure SQL 테이블, *tableName*.

Azure SQL 테이블은 데이터베이스, 스키마 및 테이블 이름으로 식별됩니다. Azure SQL에 연결할 때 데이터베이스 이름과 테이블 이름을 제공해야 합니다. 스키마가 기본값인 "public"이 아닌 경우에도 스키마를 제공해야 합니다. 데이터베이스는 *connectionName*의 URL 속성을, *dbtable*을 통해 스키마 및 테이블 이름을 제공 받습니다.

Azure SQL 데이터 대상 추가

데이터 대상 - Azure SQL 노드 추가 방법:

1. Azure SQL 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 Azure SQL 연결 생성을 선택합니다. 자세한 내용은 [이전the section called “Azure SQL 연결 만들기”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. *tableName*을 제공하여 테이블 이름을 구성합니다.
3. 사용자 지정 Azure SQL 속성에서 필요한 경우 파라미터와 값을 입력합니다.

고급 옵션

Azure SQL 노드를 생성할 때 고급 옵션을 제공할 수 있습니다. 이 옵션은 Spark 스크립트에 대한 AWS Glue를 프로그래밍할 때 사용할 수 있는 옵션과 동일합니다.

[the section called “Azure SQL 연결”](#)를 참조하세요.

BigQuery 구글에 접속하기 AWS Glue Studio

Note

Google AWS Glue 4.0 이상 BigQuery 버전에서는 AWS Glue Spark를 사용하여 테이블을 읽고 쓸 수 있습니다. 프로그래밍 방식으로 Google에서 AWS Glue 작업을 구성하려면 BigQuery를 참조하십시오. [BigQuery 연결](#).

AWS Glue Studio에서 서버리스 Spark 런타임에서 연결하고 BigQuery, 데이터 통합 작업을 작성하고, 실행할 수 있는 시각적 인터페이스를 제공합니다.

주제

- [BigQuery 연결 생성](#)
- [BigQuery 소스 노드 생성](#)
- [BigQuery 대상 노드 생성](#)
- [고급 옵션](#)

BigQuery 연결 생성

AWS Glue에서 Google BigQuery에 연결하려면 AWS Secrets Manager 보안 암호에서 Google Cloud Platform 보안 인증 정보를 생성하고 저장한 다음 해당 보안 암호를 Google AWS Glue BigQuery 연결에 연결해야 합니다.

BigQuery에 대한 연결 구성하기:

1. Google Cloud Platform에서 관련 리소스를 생성하고 식별합니다.
 - 연결하려는 BigQuery 테이블이 포함된 GCP 프로젝트를 생성하거나 식별합니다.

- BigQuery API를 활성화합니다. 자세한 내용은 [BigQuery Storage Read API를 사용하여 테이블 데이터 읽기](#)를 참조하세요.
2. Google Cloud Platform에서 서비스 계정 보안 인증 정보를 생성하고 내보냅니다.

BigQuery 보안 인증 마법사를 사용하여 [보안 인증 정보 생성하기](#) 단계를 신속하게 처리할 수 있습니다.

GCP에서 서비스 계정을 생성하려면 [서비스 계정 생성하기](#)에서 제공되는 튜토리얼을 따르세요.

- 프로젝트를 선택할 때 BigQuery 테이블이 포함된 프로젝트를 선택합니다.
- 서비스 계정의 GCP IAM 역할을 선택할 때 BigQuery 작업을 실행하여 BigQuery 테이블을 읽고, 쓰고, 생성할 수 있는 적절한 권한을 부여하는 역할을 추가하거나 생성하세요.

서비스 계정의 보안 인증 정보를 생성하려면 [서비스 계정 키 생성하기](#)에서 제공되는 튜토리얼을 따르세요.

- 키 유형을 선택할 때 JSON을 선택합니다.

이제 서비스 계정의 보안 인증 정보가 포함된 JSON 파일을 다운로드했어야 합니다. 예를 들면 다음과 같아야 합니다.

```
{
  "type": "service_account",
  "project_id": "*****",
  "private_key_id": "*****",
  "private_key": "*****",
  "client_email": "*****",
  "client_id": "*****",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "*****",
  "universe_domain": "googleapis.com"
}
```

3. base64는 다운로드한 보안 인증 파일을 인코딩합니다. AWS CloudShell 세션 또는 유사한 사항인 경우 명령줄에서 `cat credentialsFile.json | base64 -w 0`(를) 실행하여 이 작업을 수행할 수 있습니다. 이 명령의 출력인 *credentialString*을 유지합니다.

4. AWS Secrets Manager에서 Google Cloud Platform 보안 인증 정보를 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *credentialString*이 포함된 키 credentials에 대한 페어를 생성합니다.
5. AWS Glue 데이터 카탈로그에서 <https://docs.aws.amazon.com/glue/latest/dg/console-connections.html>의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 Google BigQuery를 선택합니다.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.
6. AWS Glue 작업 권한과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여합니다.
7. AWS Glue 작업 구성에서 추가 네트워크 연결로 *connectionName*을 제공합니다.

BigQuery 소스 노드 생성

필수 전제 조건

- BigQuery 유형 AWS Glue 데이터 카탈로그 연결
- 연결에 사용되는 Google BigQuery 자격 증명의 AWS Secrets Manager 보안 암호.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 읽으려는 테이블 및 해당 Google Cloud 프로젝트의 이름 및 데이터세트.

BigQuery 데이터 소스 추가

데이터 소스 - BigQuery 노드 추가하기:

1. BigQuery 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 BigQuery 연결 생성을 선택합니다. 자세한 내용은 [커넥터 및 연결 사용 개요](#)를 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. 읽으려는 BigQuery 데이터를 식별한 다음 BigQuery 소스 옵션을 선택합니다
 - 단일 테이블 선택 — 테이블에서 모든 데이터를 가져올 수 있습니다.

- 사용자 지정 쿼리 입력 — 쿼리를 제공하여 검색할 데이터를 사용자 지정할 수 있습니다.

3. 읽고 싶은 데이터를 설명하세요

(필수 사항) 상위 프로젝트를 테이블이 포함된 프로젝트로 설정하거나 해당하는 경우 청구 대상 상위 프로젝트로 설정합니다.

단일 테이블을 선택한 경우 테이블을 [dataset].[table] 형식의 Google BigQuery 테이블 이름으로 설정합니다

쿼리를 선택한 경우 쿼리에 제공합니다. 쿼리에서 [project].[dataset].[tableName] 형식으로 정규화된 테이블 이름이 있는 테이블을 참조하세요.

4. BigQuery 속성을 제공합니다

단일 테이블을 선택한 경우 추가 속성을 제공하지 않아도 됩니다.

쿼리를 선택한 경우 다음 사용자 지정 Google BigQuery 속성을 제공해야 합니다.

- viewsEnabled를 true로 설정합니다.
- materializationDataset을(를) 데이터세트에 설정합니다. AWS Glue 연결을 통해 제공된 보안 인증 정보로 인증된 GCP 주체는 이 데이터 세트에 테이블을 생성할 수 있어야 합니다.

BigQuery 대상 노드 생성

필수 전제 조건

- BigQuery 유형 AWS Glue 데이터 카탈로그 연결
- 연결에 사용되는 Google BigQuery 자격 증명의 AWS Secrets Manager 보안 암호.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 쓰려는 테이블 및 해당 Google Cloud 프로젝트의 이름 및 데이터세트.

BigQuery 데이터 대상 추가

데이터 대상 - BigQuery 노드 추가하기:

1. BigQuery 데이터 대상의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 BigQuery 연결 생성을 선택합니다. 자세한 내용은 [커넥터 및 연결 사용 개요](#)를 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. 쓰려는 BigQuery 테이블을 식별한 다음 쓰기 방법을 선택합니다.

- 직접 — BigQuery Storage 쓰기 API를 사용하여 BigQuery에 직접 글을 씁니다.
- 간접 — Google Cloud Storage에 쓴 다음 BigQuery에 복사합니다.

간접적으로 쓰려는 경우 임시 GCS 버킷과 함께 대상 GCS 위치를 제공하세요. AWS Glue 연결에 추가 구성을 제공해야 합니다. 자세한 내용은 [Google BigQuery를 통한 간접 쓰기 사용](#)을 참조하세요.

3. 읽고 싶은 데이터를 설명하세요

(필수 사항) 상위 프로젝트를 테이블이 포함된 프로젝트로 설정하거나 해당하는 경우 청구 대상 상위 프로젝트로 설정합니다.

단일 테이블을 선택한 경우 테이블을 [dataset].[table] 형식의 Google BigQuery 테이블 이름으로 설정합니다

고급 옵션

노드를 생성할 때 고급 옵션을 제공할 수 있습니다. BigQuery 이러한 옵션은 Spark 스크립트를 AWS Glue 프로그래밍할 때 사용할 수 있는 옵션과 동일합니다.

AWS Glue 개발자 안내서의 [BigQuery 연결 옵션 참조](#)를 참조하십시오.

AWS Glue Studio에서 MongoDB에 연결

AWS Glue에서는 MongoDB를 기본으로 지원합니다. AWS Glue Studio에서는 MongoDB에 연결하고, 데이터 통합 작업을 작성하며, AWS Glue Studio 서버리스 Spark 런타임에서 실행할 수 있는 시각적 인터페이스를 제공합니다.

주제

- [MongoDB 연결 생성](#)
- [MongoDB 소스 노드 생성](#)
- [MongoDB 대상 노드 생성](#)
- [고급 옵션](#)

MongoDB 연결 생성

사전 조건:

- MongoDB 인스턴스가 Amazon VPC에 있는 경우, 퍼블릭 인터넷을 통과하는 트래픽 없이 AWS Glue 작업이 MongoDB 인스턴스와 통신할 수 있도록 Amazon VPC를 구성하십시오.

Amazon VPC에서 작업을 실행하는 동안 AWS Glue가 사용할 VPC, 서브넷 및 보안 그룹을 식별하거나 생성합니다. 또한 MongoDB 인스턴스와 이 위치 간의 네트워크 트래픽을 허용하도록 Amazon VPC를 구성해야 합니다. 네트워크 레이어아웃에 따라 보안 그룹 규칙, 네트워크 ACL, NAT 게이트웨이 및 피어링 연결을 변경해야 할 수도 있습니다.

MongoDB 연결 구성 방법:

1. 대안으로 AWS Secrets Manager에서 MongoDB 보안 인증을 사용하여 보안 암호를 생성할 수 있습니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.

- 키/값 페어를 선택하면 값 *mongodbUser*이 포함된 키 username에 대한 페어를 생성합니다.

키/값 페어를 선택하면 값 *mongodbPass*가 포함된 키 password에 대한 페어를 생성합니다.

2. AWS Glue 콘솔에서 [the section called "AWS Glue 연결 추가"](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.

- 연결 유형을 선택할 때에는 MongoDB 또는 MongoDB Atlas를 선택합니다.
- MongoDB URL 또는 MongoDB Atlas URL을 선택할 때에는 MongoDB 인스턴스의 호스트 이름을 제공하십시오.

MongoDB URL은 `mongodb://mongoHost:mongoPort/mongoDBName` 형식으로 제공됩니다.

MongoDB Atlas URL은 `mongodb+srv://mongoHost:mongoPort/mongoDBName` 형식으로 제공됩니다.

연결을 위한 기본 데이터베이스를 제공하는 *mongoDBName*은 선택 사항입니다.

- Secrets Manager 암호를 생성하기로 선택한 경우 AWS Secrets Manager 보안 인증 정보 유형을 선택합니다.

그런 다음 AWS 암호에 *secretName*을 입력합니다.

- 사용자 이름과 비밀번호를 제공하기로 선택한 경우 *mongodbUser*와 *mongodbPass*를 제공하십시오.

3. 다음과 같은 상황에서는 추가 구성이 필요할 수도 있습니다.

- Amazon VPC에서 AWS에 호스팅된 MongoDB 인스턴스의 경우
 - MongoDB 보안 보안 인증 정보를 정의하는 AWS Glue 연결에 Amazon VPC 연결 정보를 제공해야 합니다. 연결을 만들거나 업데이트할 때 네트워크 옵션에서 VPC, 서브넷 및 보안 그룹을 설정합니다.

AWS Glue MongoDB 연결을 생성한 후에는 AWS Glue 작업을 실행하기 전에 다음 단계를 수행해야 합니다.

- 비주얼 에디터에서 작업을 수행할 때는 AWS Glue 작업에 대한 Amazon VPC 연결 정보를 제공하여 MongoDB에 연결할 수 있습니다. Amazon VPC에서 적합한 위치를 식별하여 MongoDB AWS Glue 연결에 제공하십시오.
- Secrets Manager 암호를 생성하기로 선택한 경우, AWS Glue 작업과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여하십시오.

MongoDB 소스 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “MongoDB 연결 생성”](#)에서 설명한 AWS Glue MongoDB 연결입니다.
- Secrets Manager 암호를 생성하기로 선택한 경우 연결에 사용되는 암호를 읽을 수 있는 적절한 작업 권한이 있어야 합니다.
- 읽으려는 MongoDB 컬렉션. 컬렉션에 대한 식별 정보가 필요합니다.

MongoDB 컬렉션은 데이터베이스 이름 *mongodbName* 및 컬렉션 이름 *mongodbCollection*으로 식별됩니다.

MongoDB 데이터 소스 추가

데이터 소스 - MongoDB 노드 추가 방법:

1. MongoDB 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 MongoDB 연결 생성을 선택합니다. 자세한 내용은 [이전 the section called “MongoDB 연결 생성”](#) 섹션을 참조하세요.
연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.
2. 데이터베이스를 선택합니다. *mongodbName*을 입력합니다.
3. 컬렉션을 선택하십시오. *mongodbCollection*을 입력하십시오.
4. 파티셔너, 파티션 크기(MB), 파티션 키를 선택합니다. 파티션 파라미터에 대한 자세한 내용은 [the section called “소스로서의 “connectionType”: “mongodb”](#) 섹션을 참조하십시오.
5. 사용자 지정 MongoDB 속성에서 필요한 경우 파라미터와 값을 입력합니다.

MongoDB 대상 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “MongoDB 연결 생성”](#)에서 설명한 대로 AWS Secrets Manager 암호로 구성된 AWS Glue MongoDB 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 쓰고 싶은 MongoDB 테이블, *tableName*입니다.

MongoDB 데이터 대상 추가

데이터 대상 - MongoDB 노드 추가 방법:

1. MongoDB 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 MongoDB 연결 생성을 선택합니다. 자세한 내용은 [이전 the section called “MongoDB 연결 생성”](#) 섹션을 참조하세요.
연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.
2. 데이터베이스를 선택합니다. *mongodbName*을 입력합니다.
3. 컬렉션을 선택하십시오. *mongodbCollection*을 입력하십시오.
4. 파티셔너, 파티션 크기(MB), 파티션 키를 선택합니다. 파티션 파라미터에 대한 자세한 내용은 [the section called “소스로서의 “connectionType”: “mongodb”](#) 섹션을 참조하십시오.

5. 원하는 경우 쓰기 재시도를 선택합니다.
6. 사용자 지정 MongoDB 속성에서 필요한 경우 파라미터와 값을 입력합니다.

고급 옵션

MongoDB 노드를 생성할 때 고급 옵션을 제공할 수 있습니다. 이 옵션은 Spark 스크립트에 대한 AWS Glue를 프로그래밍할 때 사용할 수 있는 옵션과 동일합니다.

[the section called “MongoDB 연결”](#)를 참조하세요.

AWS Glue Studio에서 OpenSearch Service에 연결

AWS Glue에서는 Amazon OpenSearch Service를 기본으로 지원합니다. AWS Glue Studio에서는 Amazon OpenSearch Service에 연결하고, 데이터 통합 작업을 작성하며, AWS Glue Studio 서버리스 Spark 런타임에서 실행할 수 있는 시각적 인터페이스를 제공합니다. 이 기능은 OpenSearch Service 서버리스와 호환되지 않습니다.

주제

- [OpenSearch Service 연결 생성](#)
- [OpenSearch Service 소스 노드 생성](#)
- [OpenSearch Service 대상 노드 생성](#)
- [고급 옵션](#)

OpenSearch Service 연결 생성

사전 조건:

- Amazon OpenSearch Service 설명서의 지침에 따라 읽으려는 도메인 엔드포인트, *aosEndpoint* 및 포트, *aosPort*를 식별하거나 리소스를 생성하십시오. 도메인 생성에 관해 자세한 내용을 알아보려면 Amazon OpenSearch Service 설명서의 [Amazon OpenSearch Service 도메인 생성 및 관리](#)를 참조하십시오.

Amazon OpenSearch Service 도메인 엔드포인트는 기본 형식이 `https://search-domainName-unstructuredIdContent.region.es.amazonaws.com`입니다. 도메인 엔드포인트에 관해 자세한 내용을 알아보려면 Amazon OpenSearch Service 설명서의 [Amazon OpenSearch Service 도메인 생성 및 관리](#)를 참조하십시오.

도메인의 HTTP 기본 보안 인증 정보, *aosUser*와 *aosPassword*를 확인하거나 생성하십시오.

OpenSearch Service에 대한 연결을 구성하는 방법:

1. AWS Secrets Manager에서 OpenSearch 보안 인증을 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *aosUser*이 포함된 키 `opensearch.net.http.auth.user`에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *aosPassword*이 포함된 키 `opensearch.net.http.auth.pass`에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때는 OpenSearch Service를 선택합니다.
 - 도메인 엔드포인트를 선택할 때는 *aosEndpoint*를 제공하십시오.
 - 포트를 선택할 때는 *aosPort*를 제공하십시오.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.

OpenSearch Service 소스 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “OpenSearch Service 연결 생성”](#)에서 설명한 대로 AWS Secrets Manager 암호로 구성된 AWS Glue OpenSearch Service 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 읽으려는 OpenSearch Service 인덱스, *aosIndex*.

OpenSearch Service 데이터 소스 추가

데이터 소스 - OpenSearch Service 노드를 추가하는 방법:

1. OpenSearch Service 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 OpenSearch Service 연결 생성을 선택합니다. 자세한 내용은 이전 [the section called “OpenSearch Service 연결 생성”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. 읽고 싶은 인덱스인 인덱스를 입력합니다.
3. 더 구체적인 결과를 제공하기 위해 OpenSearch 쿼리인 Query를 제공할 수도 있습니다. OpenSearch 쿼리 작성에 대한 자세한 내용은 [the section called “OpenSearch Service에서 읽기”](#)를 참조하십시오.
4. 사용자 지정 OpenSearch Service 속성에서 필요한 경우 파라미터와 값을 입력합니다.

OpenSearch Service 대상 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “OpenSearch Service 연결 생성”](#)에서 설명한 대로 AWS Secrets Manager 암호로 구성된 AWS Glue OpenSearch Service 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 쓰려는 OpenSearch Service 인덱스, *aosIndex*.

OpenSearch Service 데이터 대상 추가

데이터 대상을 추가하는 방법 - OpenSearch Service 노드:

1. OpenSearch Service 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 OpenSearch Service 연결 생성을 선택합니다. 자세한 내용은 이전 [the section called “OpenSearch Service 연결 생성”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. 읽고 싶은 인덱스인 인덱스를 입력합니다.
3. 사용자 지정 OpenSearch Service 속성에서 필요한 경우 파라미터와 값을 입력합니다.

고급 옵션

OpenSearch Service 노드를 생성할 때 고급 옵션을 제공할 수 있습니다. 이 옵션은 Spark 스크립트에 대한 AWS Glue를 프로그래밍할 때 사용할 수 있는 옵션과 동일합니다.

[the section called “OpenSearch Service 연결”](#)를 참조하세요.

에서 세일즈포스에 연결 AWS Glue Studio

Salesforce는 영업, 고객 서비스, 전자 상거래 등에 도움이 되는 고객 관계 관리 (CRM) 소프트웨어를 제공합니다. Salesforce 사용자인 경우 Salesforce 계정에 연결할 AWS Glue 수 있습니다. 그러면 Salesforce를 ETL 작업의 데이터 소스 또는 대상으로 사용할 수 있습니다. 이러한 작업을 실행하여 Salesforce와 AWS 서비스 또는 기타 지원되는 애플리케이션 간에 데이터를 전송하세요.

주제

- [AWS Glue 세일즈포스 지원](#)
- [연결 생성 및 사용을 위한 API 작업을 포함하는 정책](#)
- [세일즈포스 구성](#)
- [세일즈포스 연결 구성](#)
- [Salesforce 엔티티에서 읽기](#)
- [세일즈포스에 글쓰기](#)
- [세일즈포스 연결 옵션](#)
- [세일즈포스 커넥터에 대한 제한](#)
- [세일즈포스의 JWT 베어러 OAuth 플로우 설정](#)

AWS Glue 세일즈포스 지원

AWS Glue 다음과 같이 세일즈포스를 지원합니다.

소스로 지원되나요?

예. AWS Glue ETL 작업을 사용하여 Salesforce에서 데이터를 쿼리할 수 있습니다.

대상으로 지원되나요?

예. AWS Glue ETL을 사용하여 Salesforce에 레코드를 쓸 수 있습니다.

지원되는 세일즈포스 API 버전

지원되는 세일즈포스 API 버전은 다음과 같습니다.

- v5.8.0
- v59.0
- v60.0

연결 생성 및 사용을 위한 API 작업을 포함하는 정책

다음 샘플 정책은 연결 생성 및 사용에 필요한 AWS IAM 권한을 설명합니다. 새 역할을 생성하는 경우 다음을 포함하는 정책을 생성합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:ListConnectionTypes",
        "glue:DescribeConnectionType",
        "glue:RefreshOAuth2Tokens",
        "glue:ListEntities",
        "glue:DescribeEntity"
      ],
      "Resource": "*"
    }
  ]
}
```

다음 IAM 정책을 사용하여 액세스를 허용할 수도 있습니다.

- [AWSGlueServiceRole](#)— 사용자를 대신하여 다양한 AWS Glue 프로세스를 실행하는 데 필요한 리소스에 대한 액세스 권한을 부여합니다. 이러한 리소스에는 Amazon S3, IAM, CloudWatch 로그 및 Amazon EC2가 포함됩니다 AWS Glue. 이 정책에 지정된 리소스의 이름 지정 규칙을 따르는 경우 AWS Glue 프로세스에 필요한 권한이 부여됩니다. 이 정책은 크롤러, 작업 및 개발 엔드포인트를 정의할 때 지정된 역할에 일반적으로 추가됩니다.
- [AWSGlueConsoleFullAccess](#)— 정책이 연결된 ID가 AWS Management Console을 사용하는 경우 AWS Glue 리소스에 대한 전체 액세스 권한을 부여합니다. 이 정책에 지정된 리소스의 이름 변환을 따르면 사용자는 콘솔 전체 용량을 소유합니다. 이 정책은 일반적으로 AWS Glue 콘솔 사용자에게 연결됩니다.

세일즈포스 구성

를 사용하여 AWS Glue Salesforce와 데이터를 주고받을 수 있으려면 먼저 다음 요구 사항을 충족해야 합니다.

최소 요구 사항

최소 요구 사항은 다음과 같습니다.

- 세일즈포스 계정이 있습니다.
- Salesforce 계정에는 API 액세스가 활성화되어 있습니다. 엔터프라이즈, 언리미티드, 개발자 및 퍼포먼스 에디션에서는 API 액세스가 기본적으로 활성화됩니다.
- Salesforce 계정을 사용하여 연결된 앱을 설치할 수 있습니다. 이 기능에 액세스할 수 없는 경우 Salesforce 관리자에게 문의하세요. 자세한 내용은 Salesforce 도움말의 [연결된 앱을](#) 참조하십시오.

이러한 요구 사항을 충족하면 Salesforce 계정에 AWS Glue 연결할 준비가 된 것입니다. AWS Glue AWS 관리형 연결 앱으로 나머지 요구 사항을 처리합니다.

Salesforce용 AWS 관리형 연결 앱

AWS 관리형 연결 앱을 사용하면 더 적은 단계로 Salesforce 연결을 만들 수 있습니다. Salesforce에서 연결된 앱은 외부 애플리케이션 (예 AWS Glue: Salesforce 데이터 액세스) 을 승인하는 프레임워크입니다.

- 콘솔을 사용하여 Salesforce 연결을 생성하세요. AWS Glue
- 연결을 구성할 때 OAuth 권한 부여 유형을 인증 코드로 설정합니다.

세일즈포스 연결 구성

Salesforce 연결을 구성하려면:

1. AWS Secrets Manager에서 다음과 같은 세부 정보가 포함된 시크릿을 생성합니다.
 - a. JWT_TOKEN 권한 부여 유형의 경우 암호에는 해당 값과 함께 JWT_TOKEN 키가 포함되어야 합니다.
 - b. AuthorizationCode 권한 부여 유형의 경우: 고객 관리형 연결 앱의 경우 암호에는 키가 있는 연결된 앱 소비자 비밀번호가 포함되어야 합니다. USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET AWS 관리형 연결 앱의 경우 빈 암호 또는 임시 값이 있는 암호.
 - c. 참고: 연결마다 암호를 만들어야 AWS Glue합니다.
2. AWS Glue 데이터 카탈로그에서 아래 단계에 따라 연결을 생성합니다.
 - a. 연결 유형을 선택할 때는 Salesforce를 선택합니다.
 - b. 연결하려는 세일즈포스의 인스턴스_URL을 입력합니다.

- c. 세일즈포스 환경을 제공하십시오.
- d. 다음 작업을 맡을 수 있고 권한을 가질 AWS Glue 수 있는 AWS IAM 역할을 선택합니다.
- e. 연결에 사용할 OAuth2 권한 유형을 선택합니다. 권한 유형에 따라 Salesforce와 AWS Glue 통신하여 데이터에 대한 액세스를 요청하는 방법이 결정됩니다. 선택에 따라 연결을 생성하기 전에 충족해야 하는 요구 사항이 달라집니다. 다음 유형 중 하나를 선택할 수 있습니다.

- JWT_BEARER 부여 유형: 이 권한 부여 유형은 Salesforce 인스턴스에서 특정 사용자의 권한으로 JSON 웹 토큰 (JWT) 을 미리 생성할 수 있으므로 자동화 시나리오에 적합합니다. 생성자는 JWT의 유효 기간을 제어할 수 있습니다. AWS Glue JWT를 사용하여 Salesforce API를 호출하는 데 사용되는 액세스 토큰을 얻을 수 있습니다.

이 흐름을 사용하려면 사용자가 Salesforce 인스턴스에 연결된 앱을 만들어 사용자에게 JWT 기반 액세스 토큰을 발급할 수 있도록 해야 합니다.

[JWT 베어러 OAuth 흐름을 위한 연결된 앱을 만드는 방법에 대한 자세한 내용은 통합을 위한 OAuth 2.0 JWT 베어러 플로우를 참조하십시오. server-to-server Salesforce 연결 앱을 사용하여 JWT 베어러 플로우를 설정하려면 을 참조하십시오. 세일즈포스의 JWT 베어러 OAuth 플로우 설정](#)

- AUTHORIZATION_CODE 권한 부여 유형: 이 권한 부여 유형은 사용자를 타사 권한 부여 서버로 리디렉션하여 사용자를 인증해야 하므로 “세 가지 제한이 있는” OAuth로 간주됩니다. 콘솔을 통해 연결을 생성할 때 사용됩니다 AWS Glue . 연결을 생성하는 사용자는 기본적으로 Salesforce 인스턴스 URL을 제외한 OAuth 관련 정보를 제공할 필요가 없는 AWS Glue 연결된 앱 (AWS Glue 관리되는 클라이언트 애플리케이션) 을 사용할 수 있습니다. AWS Glue 콘솔은 사용자를 Salesforce로 리디렉션합니다. Salesforce에서는 사용자가 로그인하고 요청된 권한으로 Salesforce 인스턴스에 액세스할 수 있도록 AWS Glue 허용해야 합니다.

사용자는 여전히 Salesforce에서 직접 연결된 앱을 만들고 콘솔을 통해 연결을 생성할 때 고유한 클라이언트 ID와 클라이언트 암호를 제공하도록 선택할 수 있습니다. AWS Glue 이 시나리오에서도 사용자는 여전히 Salesforce로 리디렉션되어 로그인하고 리소스에 액세스할 수 있는 권한을 AWS Glue 부여받게 됩니다.

이 권한 부여 유형을 사용하면 새로 고침 토큰과 액세스 토큰이 생성됩니다. 액세스 토큰은 수명이 짧으며 새로 고침 토큰을 사용하여 사용자 상호 작용 없이 자동으로 새로 고쳐질 수 있습니다.

인증 코드 OAuth 흐름을 위한 연결된 앱을 만드는 방법에 대한 자세한 내용은 [인증 코드 및 자격 증명 흐름을 위한 연결된 앱 구성을 참조하십시오.](#)

- f. 이 연결에 사용할 항목을 선택하여 AWS Glue 토큰을 입력합니다. secretName

- g. 네트워크를 사용하려면 네트워크 옵션을 선택하세요. AWS Glue 작업과 관련된 IAM 역할에 읽기 `secretName` 권한을 부여하십시오.
3. AWS Glue 작업과 관련된 IAM 역할에 읽기 권한을 부여하십시오. `secretName`
 4. AWS Glue 작업 구성에서 추가 네트워크 **connectionName** 연결로 제공하십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetSecretValue",
        "secretsmanager:PutSecretValue",
        "ec2:CreateNetworkInterface",
        "ec2:DescribeNetworkInterface",
        "ec2>DeleteNetworkInterface",
      ],
      "Resource": "*"
    }
  ]
}
```

Salesforce 엔티티에서 읽기

사전 조건

읽고 싶은 세일즈포스 Subject입니다. 또는 또는 와 같은 개체 이름이 필요합니다. Account Case Opportunity

예:

```
salesforce_read = glueContext.create_dynamic_frame.from_options(
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v60.0"
    }
)
```

파티셔닝 쿼리

Spark에서 동시성을 활용하려는 NUM_PARTITIONS 경우 추가 Spark 옵션 PARTITION_FIELD LOWER_BOUND UPPER_BOUND, 및 을 제공할 수 있습니다. 이러한 매개변수를 사용하면 원본 쿼리가 Spark 작업에서 동시에 실행할 수 있는 NUM_PARTITIONS 여러 개의 하위 쿼리로 분할됩니다.

- PARTITION_FIELD: 쿼리를 분할하는 데 사용할 필드의 이름.
- LOWER_BOUND: 선택한 파티션 필드의 포괄적인 하한 값입니다.

타임스탬프 필드의 경우 Spark SQL 쿼리에 사용되는 Spark 타임스탬프 형식을 사용할 수 있습니다.

유효한 값의 예:

```
"TIMESTAMP \"1707256978123\""  
"TIMESTAMP '2024-02-06 22:02:58.123 UTC'"  
"TIMESTAMP \"2018-08-08 00:00:00 Pacific/Tahiti\""  
"TIMESTAMP \"2018-08-08 00:00:00\""  
"TIMESTAMP \"-123456789\" Pacific/Tahiti"  
"TIMESTAMP \"1702600882\""
```

- UPPER_BOUND: 선택한 파티션 필드의 배타적 상한값입니다.
- NUM_PARTITIONS: 파티션 수.

예제

```
salesforce_read = glueContext.create_dynamic_frame.from_options(  
    connection_type="salesforce",  
    connection_options={  
        "connectionName": "connectionName",  
        "ENTITY_NAME": "Account",  
        "API_VERSION": "v60.0",  
        "PARTITION_FIELD": "SystemModstamp"  
        "LOWER_BOUND": "TIMESTAMP '2021-01-01 00:00:00 Pacific/Tahiti'"  
        "UPPER_BOUND": "TIMESTAMP '2023-01-10 00:00:00 Pacific/Tahiti'"  
        "NUM_PARTITIONS": "10"  
    }  
)
```

세일즈포스에 글쓰기

사전 조건

글을 쓰고 싶은 세일즈포스 SoBject입니다. 또는 또는 와 같은 개체 이름이 필요합니다. Account Case Opportunity

Salesforce 커넥터는 네 가지 쓰기 작업을 지원합니다.

- INSERT
- UPSERT
- UPDATE
- DELETE

UPSERT 쓰기 작업을 사용하는 경우 레코드의 외부 ID 필드를 지정하려면 ID_FIELD_NAMES 제공해야 합니다.

예

```
salesforce_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="salesforce",
    connection_options={
        "connectionName": "connectionName",
        "ENTITY_NAME": "Account",
        "API_VERSION": "v60.0",
        "WRITE_OPERATION": "INSERT"
    }
}
```

세일즈포스 연결 옵션

Salesforce의 연결 옵션은 다음과 같습니다.

- ENTITY_NAME(문자열) - (필수) 읽기/쓰기에 사용됩니다. Salesforce에 있는 개체의 이름.
- API_VERSION(문자열) - (필수) 읽기/쓰기에 사용됩니다. 사용하려는 세일즈포스 레스트 API 버전.
- SELECTED_FIELDS(목록<String>) - 기본값: 비어 있음 (SELECT *). 읽기에 사용됩니다. 개체에 대해 선택하려는 열.
- FILTER_PREDICATE(문자열) - 기본값: 비어 있습니다. 읽기에 사용됩니다. 스파크 SQL 형식이어야 합니다.
- QUERY(문자열) - 기본값: 비어 있습니다. 읽기에 사용됩니다. 전체 스파크 SQL 쿼리.
- PARTITION_FIELD(문자열) - 읽기에 사용됩니다. 쿼리를 분할하는 데 사용할 필드입니다.

- LOWER_BOUND(문자열) - 읽기에 사용됩니다. 선택한 파티션 필드의 포함된 하한 값입니다.
- UPPER_BOUND(문자열) - 읽기에 사용됩니다. 선택한 파티션 필드의 배타적 상한값입니다.
- NUM_PARTITIONS(정수) - 기본값: 1. 읽기에 사용됩니다. 읽을 파티션 수.
- IMPORT_DELETED_RECORDS(문자열) - 기본값: FALSE. 읽기에 사용됩니다. 쿼리하는 동안 삭제 기록을 가져오려는 경우
- WRITE_OPERATION(문자열) - 기본값: 삽입. 쓰기에 사용됩니다. 값은 삽입, 업데이트, 업서트, 삭제여야 합니다.
- ID_FIELD_NAMES(문자열) - 기본값: null. UPSERT에만 사용됩니다.

세일즈포스 커넥터에 대한 제한

Salesforce 커넥터에 대한 제한 사항은 다음과 같습니다.

- Spark SQL만 지원하며 세일즈포스 SOQL은 지원되지 않습니다.
- 작업 북마크는 지원되지 않습니다.

세일즈포스의 JWT 베어러 OAuth 플로우 설정

[OAuth 2.0 JSON 웹 토큰과의 통합을 server-to-server 활성화하는 방법은 세일즈포스 공개 설명서를 참조하십시오.](#)

PEM 파일의 인증서/키 쌍 생성

PEM 파일의 인증서/키 쌍 생성

```
openssl req -newkey rsa:4096 -new -nodes -x509 -days 3650 -keyout key.pem -out cert.pem
```

JWT로 세일즈포스 커넥티드 앱 만들기

1. [Salesforce](#)에 로그인하고 오른쪽 상단의 설정 톱니바퀴를 클릭하고 설정을 선택합니다.
2. 왼쪽에서 앱 관리자로 이동합니다. (플랫폼 도구 > 앱 > 앱 관리자)
3. 새 연결 앱을 선택합니다.
4. 앱 이름을 제공하고 나머지는 자동으로 입력되도록 합니다.
5. OAuth 설정 활성화 체크박스를 선택합니다.
6. 콜백 URL을 설정합니다. JWT에는 사용되지 않으므로 `https://localhost` 을 사용할 수 있습니다.

7. 디지털 서명 사용 체크박스를 체크하세요.
8. 이전에 만든 cert.pem 파일을 업로드합니다.
9. 필수 권한 추가:
 - a. API (api) 를 통해 사용자 데이터를 관리합니다.
 - b. 사용자 지정 권한 (custom_permissions) 에 액세스할 수 있습니다.
 - c. ID URL 서비스 (ID, 프로필, 이메일, 주소, 전화) 에 액세스합니다.
 - d. 고유 사용자 식별자 (openid) 에 액세스할 수 있습니다.
 - e. 언제든지 요청을 수행할 수 있습니다 (리프레시_토큰, 오프라인_액세스).
10. 지정된 사용자에게 JSON 웹 토큰 (JWT) 기반 액세스 토큰 발급 체크박스를 선택합니다.
11. 저장을 선택합니다.
12. 계속을 선택합니다.
13. 소비자 세부 정보 관리(Manage Consumer Details)를 선택합니다.
14. 소비자 키 (클라이언트 ID) 를 복사합니다.
15. 소비자 암호 (클라이언트 암호) 를 복사합니다.
16. 취소를 클릭합니다.

JSON 웹 토큰 (JWT) 생성

1. 키 쌍을 pkcs12로 변환합니다 (메시지가 표시되면 내보내기 암호 설정).

```
openssl pkcs12 -export -in cert.pem -inkey key.pem -name jwtcert > jwtcert.p12
```

2. pkcs12에서 Java 키스토어를 생성합니다. 메시지가 표시되면 대상 키스토어 비밀번호를 설정하고 소스 키스토어 비밀번호에는 이전 익스포트 비밀번호를 입력합니다.

```
keytool -importkeystore -srckeystore jwtcert.p12 -destkeystore keystore.jks -srcstoretype pkcs12 -alias jwtcert
```

3. keystore.jks에 jwtcert 별칭이 포함되어 있는지 확인합니다 (메시지가 표시되면 이전 대상 키스토어 비밀번호 입력).

```
keytool -keystore keystore.jks -list
```

4. Salesforce 설명서에 제공된 Java 클래스 JWTExample을 사용하여 서명된 토큰을 생성하십시오.
 - a. 필요에 따라 ClaimArray의 값을 편집합니다.

- 클레임 어레이 [0] = 클라이언트 ID
 - 클레임 어레이 [1] = 세일즈포스 사용자 ID
 - 클레임 어레이 [2] = 세일즈포스 로그인 URL
 - ClaimArray [4] = 에포크 이후의 만료 날짜 (단위: 밀리초). 366062400000000은 2085-12-31입니다.
- b. 경로/수신자/키스토어를 keystore.jks의 올바른 경로로 바꾸십시오.
 - c. 키스토어 비밀번호를 입력한 대상 키스토어 비밀번호로 바꿉니다.
 - d. 개인 키 암호를 입력한 소스 키 저장소 암호로 바꾸십시오.
 - e. 샘플 코드를 컴파일합니다. 코드는 base64 인코딩용 [아파치 커먼즈](#) 코덱에 따라 다릅니다.

```
javac -classpath " ../commons-codec-1.16.1.jar" JWTExample.java
```

- f. 코드를 실행합니다.

```
java -classpath " ../commons-codec-1.16.1.jar" JWTExample
```

5. 연결된 앱과 JWT를 만든 후에도 사용자는 앱에 대한 승인을 받아야 합니다. 두 가지 접근 방식에 대해서는 <https://mannharleen.github.io/2020-03-03-salesforce-jwt/> 의 3단계를 참조하십시오.

위 단계를 완료하면 Salesforce에서 액세스 토큰을 가져오는 데 사용할 수 있는 JSON 웹 토큰 (JWT) 이 출력됩니다.

입력 예:

```
export password for pkcs12: awsglue
destination keystore password for jks: awsglue
source keystore password for jks: awsglue

claimArray[0] = "client-id";
claimArray[1] = "my@email.com";
claimArray[2] = "https://login.salesforce.com";
claimArray[3] = "366062400000000";

path to keystore: ./keystore.jks
keystore password: awsglue
privatekey password: awsglue
```

출력 예:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG91IiwiaWF0Ij
```

유용한 링크:

- <https://www.base64encode.org/>
- <https://jwt.io/>
- https://help.salesforce.com/s/articleView?id=sf.remoteaccess_oauth_jwt_flow.htm

JWTExample.java:

```
import org.apache.commons.codec.binary.Base64;
import java.io.*;
import java.security.*;
import java.text.MessageFormat;

public class JWTExample {

    public static void main(String[] args) {

        String header = "{\"alg\":\"RS256\"}";
        String claimTemplate = "'{'iss\": \"{0}\", \"sub\": \"{1}\", \"aud\": \"{2}\",
        \"exp\": \"{3}\"}'";

        try {
            StringBuffer token = new StringBuffer();

            //Encode the JWT Header and add it to our string to sign
            token.append(Base64.encodeBase64URLSafeString(header.getBytes("UTF-8")));

            //Separate with a period
            token.append(".");

            //Create the JWT Claims Object
            String[] claimArray = new String[5];
            claimArray[0] = "value";
            claimArray[1] = "my@email.com";
            claimArray[2] = "https://login.salesforce.com";
            claimArray[3] = Long.toString( ( System.currentTimeMillis()/1000 ) + 300);
            MessageFormat claims;
            claims = new MessageFormat(claimTemplate);
            String payload = claims.format(claimArray);
```

```
//Add the encoded claims object
token.append(Base64.encodeBase64URLSafeString(payload.getBytes("UTF-8")));

//Load the private key from a keystore
KeyStore keystore = KeyStore.getInstance("JKS");
keystore.load(new FileInputStream("./keystore.jks"), "awsglue".toCharArray());
PrivateKey privateKey = (PrivateKey) keystore.getKey("jwtcert",
"awsglue".toCharArray());

//Sign the JWT Header + "." + JWT Claims Object
Signature signature = Signature.getInstance("SHA256withRSA");
signature.initSign(privateKey);
signature.update(token.toString().getBytes("UTF-8"));
String signedPayload = Base64.encodeBase64URLSafeString(signature.sign());

//Separate with a period
token.append(".");

//Add the encoded signature
token.append(signedPayload);

System.out.println(token.toString());

} catch (Exception e) {
    e.printStackTrace();
}
}
```

AWS Glue Studio에서 SAP HANA에 연결

AWS Glue에서는 SAP HANA를 기본으로 지원합니다. AWS Glue Studio에서는 SAP HANA에 연결하고, 데이터 통합 작업을 작성하며, AWS Glue Studio 서버리스 Spark 런타임에서 실행할 수 있는 시각적 인터페이스를 제공합니다.

주제

- [SAP HANA 커넥션 생성](#)
- [SAP HANA 소스 노드 생성](#)
- [SAP HANA 대상 노드 생성](#)
- [고급 옵션](#)

SAP HANA 커넥션 생성

AWS Glue에서 SAP HANA에 연결하려면 AWS Secrets Manager 보안 암호에서 SAP HANA 보안 인증 정보를 생성하고 저장한 다음 해당 보안 암호를 SAP HANA AWS Glue 연결에 연결해야 합니다. SAP HANA 서비스와 AWS Glue 간에 네트워크 연결을 구성해야 합니다.

사전 조건:

- SAP HANA 서비스가 Amazon VPC에 있는 경우, 퍼블릭 인터넷을 통과하는 트래픽 없이 AWS Glue 작업이 SAP HANA 서비스와 통신할 수 있도록 Amazon VPC를 구성하십시오.

Amazon VPC에서 작업을 실행하는 동안 AWS Glue가 사용할 VPC, 서브넷 및 보안 그룹을 식별하거나 생성합니다. 또한 Amazon VPC가 SAP HANA 엔드포인트와 이 위치 간의 네트워크 트래픽을 허용하도록 구성되어 있는지 확인해야 합니다. 작업을 수행하려면 SAP HANA JDBC 포트와의 TCP 연결을 설정해야 합니다. SAP HANA 포트에 대한 자세한 내용은 [SAP HANA 설명서](#)를 참조하십시오. 네트워크 레이아웃에 따라 보안 그룹 규칙, 네트워크 ACL, NAT 게이트웨이 및 피어링 연결을 변경해야 할 수도 있습니다.

SAP HANA에 대한 연결 구성하는 방법:

1. AWS Secrets Manager에서 SAP HANA 보안 인증을 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *saphanaUsername*이 포함된 키 *user*에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *saphanaPassword*가 포함된 키 *password*에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 SAP HANA를 선택합니다.
 - SAP HANA URL을 제공할 때는 인스턴스의 URL을 제공하십시오.

SAP HANA JDBC URL은

```
jdbc:sap://saphanaHostname:saphanaPort?databaseName=saphanaDBname,Parameter
```

형식입니다

AWS Glue는 JDBC URL 매개 변수가 필요합니다.

- `databaseName` - 연결할 SAP HANA의 기본 데이터베이스입니다.
- AWS 보안 암호를 선택할 때 `secretName`을 입력합니다.

AWS Glue SAP HANA 연결을 생성한 후에는 AWS Glue 작업을 실행하기 전에 다음 단계를 수행해야 합니다.

- AWS Glue 작업 권한과 연결된 IAM 역할에 `secretName`을 읽을 수 있는 권한을 부여합니다.

SAP HANA 소스 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “SAP HANA 커넥션 생성”](#)에서 설명한 것처럼 AWS Secrets Manager 암호로 구성된 AWS Glue SAP HANA 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 읽으려는 SAP HANA 테이블, `tableName` 또는 쿼리 `targetQuery`.

SAP HANA 테이블 이름 및 스키마 이름을 `schemaName.tableName` 형식으로 테이블을 지정할 수 있습니다. 테이블이 기본 스키마 "public"에 있는 경우 스키마 이름과 "." 구분 기호는 필요하지 않습니다. 이것을 `tableIdentifier`라고 합니다. 데이터베이스는 `connectionName`에서 JDBC URL 매개변수로 제공된다는 점에 유의하십시오.

SAP HANA 데이터 소스 추가

데이터 소스 - SAP HANA 노드를 추가하는 방법:

1. SAP HANA 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 SAP HANA 연결 생성을 선택합니다. 자세한 내용은 이전 [the section called “SAP HANA 커넥션 생성”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. SAP HANA 소스 옵션을 선택하십시오.
 - 단일 테이블 선택 - 단일 테이블에서 모든 데이터에 액세스할 수 있습니다.
 - 사용자 지정 쿼리 입력 - 사용자 지정 쿼리를 기반으로 여러 테이블의 데이터 세트에 액세스할 수 있습니다.
3. 단일 테이블을 선택한 경우 `tableName`을 입력합니다.

사용자 지정 쿼리 입력을 선택한 경우 SQL SELECT 쿼리를 입력합니다.

4. 사용자 지정 SAP HANA 속성에서 필요한 경우 파라미터와 값을 입력합니다.

SAP HANA 대상 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “SAP HANA 커넥션 생성”](#)에서 설명한 것처럼 AWS Secrets Manager 암호로 구성된 AWS Glue SAP HANA 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 쓰고 싶은 SAP HANA 테이블, *tableName*

SAP HANA 테이블 이름 및 스키마 이름을 *schemaName.tableName* 형식으로 테이블을 지정할 수 있습니다. 테이블이 기본 스키마 "public"에 있는 경우 스키마 이름과 "." 구분 기호는 필요하지 않습니다. 이것을 *tableIdentifier*라고 합니다. 데이터베이스는 *connectionName*에서 JDBC URL 매개변수로 제공된다는 점에 유의하십시오.

SAP HANA 데이터 대상 추가

데이터 대상 - SAP HANA 노드 추가 방법:

1. SAP HANA 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 SAP HANA 연결 생성을 선택합니다. 자세한 내용은 [이전 the section called “SAP HANA 커넥션 생성”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. *tableName*을 제공하여 테이블 이름을 구성합니다.
3. 사용자 지정 Teradata 속성에서 필요한 경우 파라미터와 값을 입력합니다.

고급 옵션

SAP HANA 노드를 생성할 때 고급 옵션을 제공할 수 있습니다. 이 옵션은 Spark 스크립트에 대한 AWS Glue를 프로그래밍할 때 사용할 수 있는 옵션과 동일합니다.

[the section called “SAP HANA 커넥션”](#)를 참조하세요.

AWS Glue Studio에서 Snowflake에 연결

Note

AWS Glue for Spark를 사용하여 AWS Glue 4.0 이상 버전에서 Snowflake의 테이블에서 읽고 쓸 수 있습니다. 프로그래밍 방식으로 AWS Glue 작업을 통해 Snowflake를 구성하려면 [Redshift 연결](#)을(를) 참조하세요.

AWS Glue에서는 Snowflake에 대한 기본 제공 지원을 제공합니다. AWS Glue Studio에서는 Snowflake에 연결하고, 데이터 통합 작업을 작성하며, AWS Glue Studio 서버리스 Spark 런타임에서 실행할 수 있는 시각적 인터페이스를 제공합니다.

주제

- [Snowflake 연결 생성](#)
- [Snowflake 소스 노드 생성](#)
- [Snowflake 대상 노드 생성](#)
- [고급 옵션](#)

Snowflake 연결 생성

AWS Glue Studio에서 데이터 소스 - Snowflake 노드를 추가할 때 기존 AWS Glue Snowflake 연결을 선택하거나 새 연결을 생성할 수 있습니다. Snowflake에 연결하도록 구성된 JDBC 유형 연결이 아니라 SNOWFLAKE 유형 연결을 선택해야 합니다. 다음 절차에 따라 AWS Glue Snowflake 연결을 생성합니다.

Snowflake 연결을 생성하려면

1. Snowflake에서 사용자, *snowflakeUser* 및 암호, *snowflakePassword*를 생성합니다.
2. 이 사용자가 상호 작용하는 Snowflake 웨어하우스(*snowflakeWarehouse*)를 결정합니다. Snowflake에서 *snowflakeUser*에 대해 DEFAULT_WAREHOUSE로 설정하거나 다음 단계에서 사용하도록 기억합니다.
3. AWS Secrets Manager에서 Snowflake 보안 인증을 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.

- 키/값 페어를 선택하면 키가 sfUser인 *snowflakeUser*에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 키가 sfPassword인 *snowflakePassword*에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 키가 sfWarehouse인 *snowflakeWarehouse*에 대한 페어를 생성합니다. Snowflake에 기본값이 설정된 경우에는 필요하지 않습니다.
4. AWS Glue 데이터 카탈로그에서 [AWS Glue 연결 추가](#) 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
- 연결 유형을 선택할 때 Snowflake를 선택합니다.
 - Snowflake URL을 선택할 때 Snowflake 인스턴스의 호스트 이름을 제공합니다. URL은 *account_identifier.snowflakecomputing.com* 양식의 호스트 이름을 사용합니다.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.

Snowflake 소스 노드 생성

필요한 권한

Snowflake 데이터 소스를 사용하는 AWS Glue Studio 작업에는 추가 권한이 필요합니다. ETL 작업에 권한을 추가하는 방법에 대한 자세한 내용은 [ETL 작업에 필요한 IAM 권한 검토](#)를 참조하세요.

SNOWFLAKE AWS Glue 연결은 AWS Secrets Manager 보안 암호를 사용하여 보안 인증 정보를 제공합니다. AWS Glue Studio에서 작업 및 데이터 미리 보기 역할에는 이 보안 암호를 읽을 수 있는 권한이 있어야 합니다.

Snowflake 데이터 소스 추가

사전 조건:

- Snowflake 보안 인증의 AWS Secrets Manager 보안 암호
- Snowflake 유형 AWS Glue 데이터 카탈로그 연결

데이터 소스 - Snowflake 노드를 추가하려면:

1. Snowflake 데이터 소스의 연결을 선택합니다. 이 경우 연결이 이미 존재하며 기존 연결 중에서 선택할 수 있다고 가정합니다. 연결을 생성해야 하는 경우 Snowflake 연결 생성을 선택합니다. 자세한 내용은 [커넥터 및 연결 사용 개요](#)를 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다. URL, 보안 그룹, 서브넷, 가용 영역, 설명, 생성 날짜(UTC) 및 최종 업데이트(UTC) 타임스탬프를 비롯한 연결 정보가 표시됩니다.

2. 다음과 같은 Snowflake 소스 옵션을 선택합니다.

- 단일 테이블 선택 - 단일 Snowflake 테이블에서 액세스하려는 데이터가 들어 있는 테이블입니다.
- 사용자 지정 쿼리 입력 - 사용자 지정 쿼리를 기반으로 여러 Snowflake 테이블의 데이터 세트에 액세스할 수 있습니다.

3. 단일 테이블을 선택한 경우 Snowflake 스키마의 이름을 입력합니다.

또는 사용자 지정 쿼리 입력을 선택합니다. 여러 Snowflake 테이블에서 사용자 지정 데이터 세트에 액세스하려면 이 옵션을 선택합니다. 이 옵션을 선택하는 경우 Snowflake 쿼리를 입력합니다.

4. 성능 및 보안 옵션(선택 사항)에서,

- 쿼리 푸시다운 활성화 - Snowflake 인스턴스로 작업을 오프로드할지 여부를 선택합니다.

5. 사용자 지정 Snowflake 속성(선택 사항)에서 필요한 경우 파라미터와 값을 입력합니다.

Snowflake 대상 노드 생성

필요한 권한

Snowflake 데이터 소스를 사용하는 AWS Glue Studio 작업에는 추가 권한이 필요합니다. ETL 작업에 권한을 추가하는 방법에 대한 자세한 내용은 [ETL 작업에 필요한 IAM 권한 검토](#)를 참조하세요.

SNOWFLAKE AWS Glue 연결은 AWS Secrets Manager 보안 암호를 사용하여 보안 인증 정보를 제공합니다. AWS Glue Studio에서 작업 및 데이터 미리 보기 역할에는 이 보안 암호를 읽을 수 있는 권한이 있어야 합니다.

Snowflake 데이터 대상 추가

Snowflake 대상 노드를 생성하려면:

1. 기존 Snowflake 테이블을 대상으로 선택하거나 새 테이블 이름을 입력합니다.
2. 데이터 대상 - Snowflake 대상 노드를 사용하는 경우 다음 옵션 중에서 선택할 수 있습니다.
 - 추가 - 테이블이 이미 있는 경우 모든 새 데이터를 테이블에 삽입으로 덤프합니다. 테이블이 없으면 새로 생성한 후 새 데이터를 모두 삽입합니다.

- 병합 - AWS Glue는 사용자가 지정한 조건에 따라 대상 테이블에 데이터를 추가하거나 업데이트합니다.

다음과 같은 옵션을 선택합니다.

- 키 및 간단한 작업 선택 - 소스 데이터와 대상 데이터 세트 사이에서 일치하는 키로 사용할 열을 선택합니다.

일치하는 경우 다음 옵션을 지정합니다.

- 대상 데이터 세트의 레코드를 소스의 데이터로 업데이트합니다.
- 대상 데이터 세트에서 레코드를 삭제합니다.

일치하지 않는 경우 다음 옵션을 지정합니다.

- 소스 데이터를 대상 데이터 세트에 새 행으로 삽입합니다.
- 아무 작업 안 함.
- 사용자 지정 MERGE 명령문 입력 - 그런 다음 병합 명령문 검증을 선택하여 명령문이 유효한지 여부를 검증할 수 있습니다.
- 잘라내기 - 테이블이 이미 있는 경우 먼저 대상 테이블의 콘텐츠를 지워서 테이블 데이터를 잘라냅니다. 잘라내기에 성공하면 모든 데이터를 삽입합니다. 테이블이 없는 경우 테이블을 생성하고 모든 데이터를 삽입합니다. 잘라내기에 실패하면 작업에 실패합니다.
- 삭제 - 테이블이 이미 있는 경우 테이블 메타데이터와 데이터를 삭제합니다. 삭제에 성공하면 모든 데이터를 삽입합니다. 테이블이 없는 경우 테이블을 생성하고 모든 데이터를 삽입합니다. 삭제에 실패하면 작업에 실패합니다.

고급 옵션

AWS Glue 개발자 안내서의 [Snowflake 연결](#)을 참조하세요.

AWS Glue Studio의 Teradata 밴티지에 연결

AWS Glue에서는 Teradata Vantage를 기본으로 지원합니다. AWS Glue Studio에는 Teradata에 연결하고, 데이터 통합 작업을 작성하며, AWS Glue Studio 서버리스 Spark 런타임에서 실행할 수 있는 시각적 인터페이스가 있습니다.

주제

- [Teradata Vantage 연결 생성](#)
- [Teradata 소스 노드 생성](#)

- [Teradata 대상 노드 생성](#)
- [고급 옵션](#)

Teradata Vantage 연결 생성

AWS Glue에서 Teradata Vantage에 연결하려면 Teradata 보안 인증 정보를 생성하여 AWS Secrets Manager 암호에 저장한 다음 해당 암호를 AWS Glue Teradata 연결에 연결해야 합니다.

사전 조건:

- Amazon VPC를 통해 Teradata 환경에 액세스하는 경우, AWS Glue 작업이 Teradata 환경과 통신할 수 있도록 Amazon VPC를 구성하십시오. 퍼블릭 인터넷을 통해 Teradata 환경에 액세스하는 것은 권장하지 않습니다.

Amazon VPC에서 작업을 실행하는 동안 AWS Glue가 사용할 VPC, 서브넷 및 보안 그룹을 식별하거나 생성합니다. 또한 Amazon VPC가 Teradata 인스턴스와 이 위치 간의 네트워크 트래픽을 허용하도록 구성되어 있는지 확인해야 합니다. 작업을 수행하려면 Teradata 클라이언트 포트와 TCP 연결을 설정해야 합니다. Teradata 포트에 대한 자세한 내용은 [Teradata 설명서](#)를 참조하십시오.

네트워크 레이아웃에 따라 보안 VPC 연결에는 Amazon VPC 및 기타 네트워킹 서비스를 변경해야 할 수 있습니다. AWS 연결에 대한 자세한 내용은 Teradata 설명서의 [AWS 연결 옵션](#)을 참조하십시오.

AWS Glue Teradata 연결을 구성하는 방법:

1. Teradata 구성에서 AWS Glue가 *teradataUser* 및 *teradataPassword*와 연결할 사용자 및 암호를 식별하거나 생성합니다. 자세한 내용은 Teradata 설명서의 [Vantage 보안 개요](#)를 참조하십시오.
2. AWS Secrets Manager에서 Teradata 보안 인증 정보를 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *teradataUsername*이 포함된 키 user에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *teradataPassword*가 포함된 키 password에 대한 페어를 생성합니다.

3. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 Teradata를 선택합니다.
 - JDBC URL을 제공할 때는 인스턴스의 URL을 제공하십시오. 또한 JDBC URL에 쉼표로 구분된 특정 연결 매개변수를 하드코딩할 수 있습니다. URL의 형식: `jdbc:teradata://teradataHostname/ParameterName=ParameterValue,ParameterName` 지원되는 URL 파라미터는 다음과 같습니다.
 - DATABASE - 기본으로 액세스하는 호스트의 데이터베이스 이름입니다.
 - DBS_PORT - 비표준 포트에서 실행할 때 사용되는 데이터베이스 포트입니다.
 - 보안 인증 정보 유형을 선택할 때에는 AWS Secrets Manager을 선택한 다음 AWS보안 암호를 *secretName*으로 설정합니다.
4. 다음과 같은 상황에서는 추가 구성이 필요할 수도 있습니다.
 - Amazon VPC에서 AWS에 호스팅된 Teradata 인스턴스의 경우
 - Teradata 보안 인증 정보를 정의하는 AWS Glue 연결에 Amazon VPC 연결 정보를 제공해야 합니다. 연결을 만들거나 업데이트할 때 네트워크 옵션에서 VPC, 서브넷 및 보안 그룹을 설정합니다.

Teradata 소스 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “Teradata Vantage 연결 생성”](#)에서 설명한 것처럼 AWS Secrets Manager 암호로 구성된 AWS Glue Teradata Vantage 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 읽으려는 Teradata 테이블, *tableName* 또는 쿼리 *targetQuery*.

Teradata 데이터 소스 추가

데이터 소스 - Teradata 노드를 추가하는 방법:

1. Teradata 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 새 연결 생성을 선택합니다. 자세한 내용은 이전 [the section called “Teradata Vantage 연결 생성”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. Teradata 소스 옵션 선택:

- 단일 테이블 선택 - 단일 테이블에서 모든 데이터에 액세스할 수 있습니다.
- 사용자 지정 쿼리 입력 - 사용자 지정 쿼리를 기반으로 여러 테이블의 데이터 세트에 액세스할 수 있습니다.

3. 단일 테이블을 선택한 경우 *tableName*을 입력합니다.

사용자 지정 쿼리 입력을 선택한 경우 SQL SELECT 쿼리를 입력합니다.

4. 사용자 지정 Teradata 속성에서 필요한 경우 파라미터와 값을 입력합니다.

Teradata 대상 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “Teradata Vantage 연결 생성”](#)에서 설명한 것처럼 AWS Secrets Manager 암호로 구성된 AWS Glue Teradata Vantage 연결입니다.
- 연결에 사용되는 보안 암호를 읽을 작업에 대한 적절한 권한.
- 쓰고 싶은 Teradata 테이블, *tableName*

Teradata 데이터 대상 추가

데이터 대상 - Teradata 노드를 추가하는 방법:

1. Teradata 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 Teradata 연결 생성을 선택합니다. 자세한 내용은 [커넥터 및 연결 사용 개요](#)를 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. *tableName*을 제공하여 테이블 이름을 구성합니다.
3. 사용자 지정 Teradata 속성에서 필요한 경우 파라미터와 값을 입력합니다.

고급 옵션

Teradata 노드를 생성할 때 고급 옵션을 제공할 수 있습니다. 이 옵션은 Spark 스크립트에 대한 AWS Glue를 프로그래밍할 때 사용할 수 있는 옵션과 동일합니다.

[the section called “Teradata Vantage 연결”](#)를 참조하세요.

AWS Glue Studio의 Vertica 입력에 연결

AWS Glue에서는 Vertica를 기본으로 지원합니다. AWS Glue Studio에는 Vertica에 연결하고, 데이터 통합 작업을 작성하며, AWS Glue Studio 서버리스 Spark 런타임에서 실행할 수 있는 시각적 인터페이스가 있습니다.

주제

- [Vertica 연결 생성](#)
- [Vertica 소스 노드 생성](#)
- [Vertica 대상 노드 생성](#)
- [고급 옵션](#)

Vertica 연결 생성

사전 조건:

- *tempS3Path*에서 참조하여 데이터베이스에서 읽고 쓸 때 임시 스토리지로 사용하는 Amazon S3 버킷 또는 폴더.

Note

AWS Glue 작업 데이터 미리 보기에서 Vertica를 사용하는 경우 임시 파일이 *tempS3Path*에서 자동으로 제거되지 않을 수 있습니다. 임시 파일을 제거하려면 데이터 미리 보기 창에서 세션 종료를 선택하여 데이터 미리 보기 세션을 바로 종료합니다. 데이터 미리 보기 세션이 바로 종료되도록 보장할 수 없는 경우 이전 데이터를 제거하도록 Amazon S3 수명 주기 구성을 설정하는 것이 좋습니다. 최대 작업 런타임에 여백을 더한 기준으로 49시간이 지난 데이터는 제거하는 것이 좋습니다. Amazon S3 수명 주기 구성에 대한 자세한 내용은 Amazon S3 설명서에서 [스토리지 수명 주기 관리](#)를 참조하십시오.

- Amazon S3 경로에 대한 적절한 권한이 있는 IAM 정책을 AWS Glue 작업 역할과 연결할 수 있습니다.
- Vertica 인스턴스가 Amazon VPC에 있는 경우, 퍼블릭 인터넷을 통과하는 트래픽 없이 AWS Glue 작업이 Vertica 인스턴스와 통신할 수 있도록 Amazon VPC를 구성하십시오.

Amazon VPC에서 작업을 실행하는 동안 AWS Glue가 사용할 VPC, 서브넷 및 보안 그룹을 식별하거나 생성합니다. 또한 Vertica 인스턴스와 이 위치 간의 네트워크 트래픽을 허용하도록 Amazon VPC를 구성해야 합니다. 작업을 수행하려면 Vertica 클라이언트 포트(기본값 5433)와의 TCP 연결을 설정해야 합니다. 네트워크 레이아웃에 따라 보안 그룹 규칙, 네트워크 ACL, NAT 게이트웨이 및 피어링 연결을 변경해야 할 수도 있습니다.

Vertica에 대한 연결 구성하는 방법:

1. AWS Secrets Manager에서 Vertica 보안 인증 정보, *verticaUsername*, *verticaPassword*을 사용하여 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *verticaUsername*이 포함된 키 *user*에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *verticaPassword*이 포함된 키 *password*에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 Vertica를 선택합니다.
 - Vertica 호스트를 선택할 때 Vertica 설치의 호스트 이름을 제공합니다.
 - Vertica 포트를 선택하면 해당 포트를 통해 Vertica 설치를 사용할 수 있습니다.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.
3. 다음과 같은 상황에서는 추가 구성이 필요할 수도 있습니다.
 - Amazon VPC에서 AWS에 호스팅된 Vertica 인스턴스의 경우
 - Vertica 보안 인증 정보를 정의하는 Amazon VPC 연결 정보를 AWS Glue 연결에 제공하십시오. 연결을 만들거나 업데이트할 때 네트워크 옵션에서 VPC, 서브넷 및 보안 그룹을 설정합니다.

AWS Glue 작업을 실행하기 전에 다음 단계를 수행해야 합니다.

- AWS Glue 작업과 권한과 연결된 IAM 역할을 *tempS3Path*에 부여합니다.
- AWS Glue 작업 권한과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여합니다.

Vertica 소스 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “Vertica 연결 생성”](#)에서 설명한 대로 Vertica 유형 AWS Glue 데이터 카탈로그 연결인 *connectionName*과 임시 Amazon S3 위치인 *tempS3Path*를 사용합니다.
- 읽으려는 Vertica 테이블, *tableName* 또는 쿼리 *targetQuery*.

Vertica 데이터 소스 추가

데이터 소스 - Vertica 노드 추가하는 방법:

1. Vertica 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 Vertica 연결 생성을 선택합니다. 자세한 내용은 이전 [the section called “Vertica 연결 생성”](#) 섹션을 참조하세요.

연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.

2. 테이블이 포함된 데이터베이스를 선택합니다.
3. Amazon S3에서 스테이징 영역을 선택하고 *tempS3Path*에 S3A URI를 입력합니다.
4. Vertica 소스를 선택합니다.
 - 단일 테이블 선택 - 단일 테이블에서 모든 데이터에 액세스할 수 있습니다.
 - 사용자 지정 쿼리 입력 - 사용자 지정 쿼리를 기반으로 여러 테이블의 데이터 세트에 액세스할 수 있습니다.
5. 단일 테이블을 선택한 경우 *tableName*을 입력하고 선택적으로 스키마를 선택합니다.

사용자 지정 쿼리를 선택한 경우 SQL SELECT 쿼리를 입력하고 선택적으로 스키마를 선택합니다.

6. 사용자 지정 Vertica 속성에서 필요한 경우 파라미터와 값을 입력합니다.

Vertica 대상 노드 생성

필수 전제 조건

- 이전 섹션 [the section called “Vertica 연결 생성”](#)에서 설명한 대로 Vertica 유형 AWS Glue 데이터 카탈로그 연결인 *connectionName*과 임시 Amazon S3 위치인 *tempS3Path*를 사용합니다.

Vertica 데이터 대상 추가

데이터 대상 - Vertica 노드를 추가하는 방법:

1. Vertica 데이터 소스의 연결을 선택합니다. 생성했으므로 드롭다운에서 사용할 수 있을 것입니다. 연결을 생성해야 하는 경우 Vertica 연결 생성을 선택합니다. 자세한 내용은 이전 [the section called “Vertica 연결 생성”](#) 섹션을 참조하세요.
연결을 선택한 후에는 속성 보기를 클릭하여 연결 속성을 볼 수 있습니다.
2. 테이블이 포함된 데이터베이스를 선택합니다.
3. Amazon S3에서 스테이징 영역을 선택하고 *tempS3Path*에 S3A URI를 입력합니다.
4. *tableName*을 입력하고 선택적으로 스키마를 선택합니다.
5. 사용자 지정 Vertica 속성에서 필요한 경우 파라미터와 값을 입력합니다.

고급 옵션

Vertica 노드를 생성할 때 고급 옵션을 제공할 수 있습니다. 이 옵션은 Spark 스크립트에 대한 AWS Glue를 프로그래밍할 때 사용할 수 있는 옵션과 동일합니다.

[the section called “수직 연결”](#)를 참조하세요.

AWS Glue Studio에서 커넥터 및 연결 사용

AWS Glue는 JDBC 연결을 사용하여 가장 일반적으로 사용되는 데이터 스토어(예: Amazon Redshift, Amazon Aurora, Microsoft SQL Server, MySQL, MongoDB 및 PostgreSQL)에 대한 기본 제공 지원을 제공합니다. AWS Glue를 사용하면 추출, 변환, 로드 작업에서 사용자 정의 JDBC 드라이버를 사용할 수도 있습니다. SaaS 애플리케이션과 같이 기본적으로 지원되지 않는 데이터 스토어의 경우 커넥터를 사용할 수 있습니다.

커넥터는 AWS Glue Studio의 데이터 스토어에 액세스하는 데 도움이 되는 선택적 코드 패키지입니다. AWS Marketplace에서 제공하는 여러 커넥터를 구독할 수 있습니다.

ETL 작업을 만들 때 기본적으로 지원되는 데이터 저장소, 의 커넥터 또는 사용자 지정 커넥터를 사용할 수 있습니다. AWS Marketplace 커넥터를 사용하는 경우 먼저 커넥터에 대한 연결을 생성해야 합니다. 특정 데이터 스토어에 연결하는 데 필요한 속성을 포함하는 연결입니다. ETL 작업에서 데이터 원본 및 데이터 대상과의 연결을 사용합니다. 커넥터와 연결은 함께 작동하여 데이터 스토어에 쉽게 액세스할 수 있습니다.

주제

- [커넥터 및 연결 사용 개요](#)
- [AWS Glue Studio에 커넥터 추가](#)
- [사용 가능한 연결](#)
- [커넥터에 대한 연결 생성](#)
- [사용자 정의 커넥터로 작업 작성](#)
- [커넥터 및 연결 관리](#)
- [사용자 정의 커넥터 개발](#)
- [AWS Glue Studio에서 커넥터와 연결을 사용하는 경우의 제한 사항](#)

커넥터 및 연결 사용 개요

특정 데이터 스토어에 연결하는 데 필요한 속성을 포함하는 연결입니다. 연결을 생성하면 AWS Glue Data Catalog에 저장됩니다. 커넥터를 선택한 다음 해당 커넥터를 기반으로 연결을 생성합니다.

에서 AWS Marketplace 기본적으로 지원되지 않는 데이터 저장소의 커넥터를 구독한 다음 연결을 만들 때 해당 커넥터를 사용할 수 있습니다. 개발자는 자체 커넥터를 생성하여 연결 생성 시 사용할 수 있습니다.

Note

사용자 지정 또는 AWS Marketplace 커넥터를 사용하여 만든 연결은 유형이 로 설정된 AWS Glue 상태로 콘솔에 AWS Glue Studio 표시됩니다. UNKNOWN

다음 단계는 AWS Glue Studio에서 커넥터를 사용하는 전체 프로세스에 대해 설명합니다.

1. 에서 AWS Marketplace 커넥터를 구독하거나 자체 커넥터를 개발하여 업로드하십시오 AWS Glue Studio. 자세한 정보는 [AWS Glue Studio에 커넥터 추가](#)을 참조하세요.
2. 커넥터 사용 정보를 검토합니다. 이 정보는 커넥터 제품 페이지의 [사용(Usage)] 탭에서 찾을 수 있습니다. 예를 들어 이 제품 페이지인 [AWS Glue Connector for BigQuery Google에서](#) 사용 탭을 클릭하면 추가 리소스 섹션에서 이 커넥터 사용에 대한 블로그 링크를 볼 수 있습니다. [Cloudwatch Logs connector for AWS Glue\(Cloudwatch Logs 커넥터\)](#)의 커넥터 제품 페이지에 표시된 것처럼 다른 커넥터에는 Overview(개요) 섹션의 지침에 대한 링크가 포함될 수 있습니다.
3. 연결을 생성합니다. 사용할 커넥터를 선택하고 로그인 자격 증명, URI 문자열 및 Virtual Private Cloud(VPC) 정보와 같은 연결에 대한 추가 정보를 제공합니다. 자세한 정보는 [커넥터에 대한 연결 생성](#)을 참조하세요.

4. 작업에 대한 IAM 역할을 생성합니다. 작업은 생성할 때 지정한 [IAM role(IAM 역할)]의 권한을 가집니다. 이 IAM 역할에는 데이터 스토어를 인증하고, 데이터 스토어에서 데이터를 추출하고, 데이터 스토어에 데이터를 쓰는 데 필요한 권한이 있어야 합니다.
5. ETL 작업을 생성하고 ETL 작업에 대한 데이터 원본 속성을 구성합니다. 사용자 정의 커넥터 공급자의 지시에 따라 연결 옵션 및 인증 정보를 제공합니다. 자세한 정보는 [사용자 정의 커넥터로 작업 작성](#)을 참조하세요.
6. [AWS Glue Studio을\(를\) 사용한 시각적 ETL](#)에 설명된 대로 변환 또는 추가 데이터 스토어를 추가하여 ETL 작업을 사용자 지정합니다.
7. 데이터 대상에 커넥터를 사용하는 경우 ETL 작업에 대한 데이터 대상 속성을 구성합니다. 사용자 정의 커넥터 공급자의 지시에 따라 연결 옵션 및 인증 정보를 제공합니다. 자세한 정보는 [the section called “사용자 정의 커넥터로 작업 작성”](#)을 참조하세요.
8. [작업 속성 수정](#)에 설명된 대로 작업 속성을 구성하여 작업 실행 환경을 사용자 지정합니다.
9. 작업을 실행합니다.

AWS Glue Studio에 커넥터 추가

커넥터는 데이터 스토어와 AWS Glue 간의 통신을 용이하게 하는 코드입니다. 에서 AWS Marketplace 제공하는 커넥터를 구독하거나 사용자 지정 커넥터를 직접 만들 수 있습니다.

주제

- [커넥터 구독 AWS Marketplace](#)
- [사용자 정의 커넥터 생성](#)

커넥터 구독 AWS Marketplace

AWS Glue Studio에서 AWS Marketplace 커넥터를 쉽게 추가할 수 있습니다.

에서 에 커넥터 AWS Marketplace 추가하기 AWS Glue Studio

1. AWS Glue Studio 콘솔의 콘솔 탐색 창에서 커넥터(Connectors)를 선택합니다.
2. Connectors(커넥터) 페이지에서 Go to AWS Marketplace(이동)을 선택합니다.
3. 에서 AWS Marketplace 추천 제품에서 사용하려는 커넥터를 선택합니다. 추천 커넥터 중 하나를 선택하거나 검색을 사용할 수 있습니다. 커넥터의 이름이나 유형을 검색하고 옵션을 사용하여 검색 결과를 구체화할 수 있습니다.

주요 커넥터 중 하나를 사용하려면 [제품 보기(View product)]를 선택합니다. 검색을 사용하여 커넥터를 찾은 경우 커넥터 이름을 선택합니다.

4. 커넥터의 제품 페이지에서 탭을 사용하여 커넥터에 대한 정보를 봅니다. 이 커넥터를 구매하기로 결정했다면 [계속 구독(Continue to Subscribe)]을 선택합니다.
5. 결제 정보를 제공한 다음 [계속 구성(Continue to Configure)]을 선택합니다.
6. [이 소프트웨어 구성(Configure this software)] 페이지에서 배포 방법과 사용할 커넥터 버전을 선택합니다. 그런 뒤 [계속 시작(Continue to Launch)]을 선택합니다.
7. [이 소프트웨어 시작(Launch this software)] 페이지에서 커넥터 공급자가 제공한 [사용 지침(Usage Instructions)]을 검토할 수 있습니다. 계속할 준비가 되면 Activate connection in AWS Glue Studio(연결 활성화)를 선택합니다.

잠시 후 콘솔에 AWS Glue Studio의 마켓플레이스 연결 생성(Create marketplace connection) 페이지가 표시됩니다.

8. [커넥터에 대한 연결 생성](#)에 설명된 대로 이 커넥터를 사용하는 연결을 생성합니다.

또는 [커넥터만 활성화(Activate connector only)]를 선택하여 지금 연결 생성을 건너뛸 수 있습니다. 커넥터를 사용하려면 나중에 연결을 생성해야 합니다.

사용자 정의 커넥터 생성

고유한 커넥터를 구축하고 커넥터 코드를 AWS Glue Studio에 업로드할 수도 있습니다.

사용자 지정 커넥터는 AWS Glue Spark 런타임 API를 통해 AWS Glue Studio에 통합됩니다. AWS Glue Spark 런타임을 사용하면 Spark, Athena 또는 JDBC 인터페이스와 호환되는 모든 커넥터를 연결할 수 있습니다. 사용자 정의 커넥터에서 사용할 수 있는 연결 옵션을 전달할 수 있습니다.

[AWS Glue 연결](#)로 모든 연결 속성을 캡슐화하고 ETL 작업에 연결 이름을 제공할 수 있습니다. Data Catalog 연결과의 통합을 통해 단일 Spark 애플리케이션 또는 다른 애플리케이션의 여러 호출에서 동일한 연결 속성을 사용할 수 있습니다.

연결에 대한 추가 옵션을 지정할 수 있습니다. AWS Glue Studio가 생성하는 작업 스크립트에는 연결을 사용하여 지정된 연결 옵션으로 커넥터를 플러그 인하는 Datasource 항목이 포함되어 있습니다. 예:

```
Datasource = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"dbTable":"Account", "connectionName":"my-custom-jdbc-
```

```
connection"}}, transformation_ctx = "DataSource0")
```

AWS Glue Studio에 사용자 지정 커넥터를 추가하려면

1. 사용자 정의 커넥터에 대한 코드를 생성합니다. 자세한 정보는 [사용자 정의 커넥터 개발](#)을 참조하세요.
2. 커넥터에 AWS Glue 기능 지원을 추가합니다. 다음은 이러한 기능의 몇 가지 예와 AWS Glue Studio에서 생성한 작업 스크립트 내에서 기능이 사용되는 방식입니다.

- 데이터 유형 매핑 - 커넥터가 기본 데이터 스토어에서 열을 읽는 동안 열을 유형 변환할 수 있습니다. 예를 들어, {"INTEGER":"STRING"}의 dataTypeMapping은 레코드를 구문 분석하고 DynamicFrame을 구성할 때 Integer 유형의 모든 열을 String 유형의 열로 변환합니다. 이는 사용자가 선택한 유형으로 열을 캐스팅하는 데 도움이 됩니다.

```
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type = "custom.jdbc", connection_options = {"dataTypeMapping":{"INTEGER":"STRING"}}, connectionName:"test-connection-jdbc", transformation_ctx = "DataSource0")
```

- 병렬 읽기를 위한 분할 - AWS Glue는 열에서 데이터를 분할하여 데이터 스토어에서 병렬 데이터 읽기를 허용합니다. 파티션 열, 하위 파티션 경계, 상위 파티션 경계 및 파티션 수를 지정해야 합니다. 이 기능을 사용하면 데이터 병렬 처리와 Spark 애플리케이션에 할당된 여러 Spark 실행기를 사용할 수 있습니다.

```
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type = "custom.jdbc", connection_options = {"upperBound":"200","numPartitions":"4","partitionColumn":"id","lowerBound":"0"}, connectionName:"test-connection-jdbc", transformation_ctx = "DataSource0")
```

- 자격 증명을 저장하는 데 사용 AWS Secrets Manager - Data Catalog 연결에는 저장된 secretId 암호에 대한 입력도 포함될 수 있습니다. AWS Secrets Manager AWS 암호는 인증 및 자격 증명 정보를 안전하게 저장하고 런타임 AWS Glue 시 제공할 수 있습니다. 또는 다음과 같이 Spark 스크립트에서 secretId를 지정할 수 있습니다.

```
DataSource = glueContext.create_dynamic_frame.from_options(connection_type = "custom.jdbc", connection_options = {"connectionName":"test-connection-jdbc", "secretId"-> "my-secret-id"}, transformation_ctx = "DataSource0")
```

- 행 조건자 및 열 프로젝션으로 소스 데이터 필터링 - AWS Glue Spark 런타임을 사용하면 SQL 쿼리를 푸시다운하여 행 조건자 및 열 프로젝션으로 소스에서 데이터를 필터링할 수도 있습니다. 이를 통해 ETL 작업은 푸시다운을 지원하는 데이터 스토어에서 필터링된 데이터를 더 빠

르게 로드할 수 있습니다. JDBC 데이터 원본으로 푸시다운된 SQL 쿼리의 예는 `SELECT id, name, department FROM department WHERE id < 200`입니다.

```
DataSource = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"query":"SELECT id, name, department FROM
department
WHERE id < 200"}, "connectionName":"test-connection-jdbc"}, transformation_ctx =
"DataSource0")
```

- 작업 북마크 – AWS Glue는 JDBC 소스에서 데이터의 증분 로드를 지원합니다. AWS Glue는 데이터 스토어에서 마지막으로 처리된 레코드를 추적하고 후속 ETL 작업 실행에서 새 데이터 레코드를 처리합니다. 작업 북마크는 이 열이 순차적으로 증가하거나 감소하는 경우 기본 키를 북마크 키의 기본 열로 사용합니다. 작업 북마크에 대한 자세한 내용은 AWS Glue Developer Guide의 [Job Bookmarks](#)를 참조하세요.

```
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
"custom.jdbc", connection_options = {"jobBookmarkKeys":["empno"],
"jobBookmarkKeysSortOrder"
:"asc", "connectionName":"test-connection-jdbc"}, transformation_ctx =
"DataSource0")
```

3. 사용자 정의 커넥터를 JAR 파일로 패키징하고 파일을 Amazon S3에 업로드합니다.
4. 사용자 정의 커넥터를 테스트합니다. 자세한 내용은 [Glue 사용자 지정 커넥터: 로컬 유효성 검사 테스트 가이드](#)의 지침을 참조하십시오. GitHub
5. AWS Glue Studio 콘솔의 콘솔 탐색 창에서 커넥터(Connectors)를 선택합니다.
6. [커넥터(Connectors)] 페이지에서 [사용자 정의 커넥터 생성(Create custom connector)]을 선택합니다.
7. [사용자 정의 커넥터 생성(Create custom connector)] 페이지에서 다음 정보를 입력합니다.
 - Amazon S3에서 사용자 정의 코드 JAR 파일의 위치에 대한 경로입니다.
 - AWS Glue Studio에서 사용할 커넥터의 이름입니다.
 - [JDBC], [Spark] [또는 Athena] 중 하나일 수 있는 커넥터 유형입니다.
 - AWS Glue Studio가 커넥터를 사용하기 위해 호출하는 사용자 지정 코드 내의 진입점 이름입니다.
 - JDBC 커넥터의 경우 이 필드는 JDBC 드라이버의 클래스 이름이어야 합니다.
 - Spark 커넥터의 경우 이 필드는 format 연산자로 Spark 데이터 원본을 로드할 때 사용하는 정규화된 데이터 원본 클래스 이름 또는 해당 별칭이어야 합니다.

- (JDBC만 해당) 데이터 스토어에 대한 JDBC 연결에서 사용하는 기본 URL입니다.
 - (선택 사항) 사용자 정의 커넥터에 대한 설명입니다.
8. [커넥터 생성(Create connector)]을 선택합니다.
 9. [커넥터에 대한 연결 생성](#)에 설명된 대로 [커넥터(Connectors)] 페이지에서 이 커넥터를 사용하는 연결을 생성합니다.

사용 가능한 연결

커넥터 연결을 생성할 때 사용할 수 있는 연결은 다음과 같습니다.

- Amazon Aurora - 내장된 보안, 백업 및 복원, 인메모리 가속화 기능을 갖춘 확장 가능한 고성능 관계형 데이터베이스 엔진.
- Amazon DocumentDB - MongoDB 및 SQL API를 지원하는 확장 가능하고 가용성이 뛰어난 완전 관리형 도큐먼트 데이터베이스 서비스입니다.
- Amazon Redshift - MongoDB 및 SQL API를 지원하는 확장 가능하고 가용성이 뛰어난 완전 관리형 도큐먼트 데이터베이스입니다.
- Azure SQL - 확장 가능하고 안정적이며 안전한 데이터 스토리지 및 관리 기능을 제공하는 Microsoft Azure의 클라우드 기반 관계형 데이터베이스 서비스입니다.
- Cosmos DB - 확장 가능한 고성능 데이터 스토리지 및 쿼리 기능을 제공하는 Microsoft Azure의 전 세계에 분산된 클라우드 데이터베이스 서비스입니다.
- Google BigQuery — 대규모 데이터 세트에서 빠른 SQL 쿼리를 실행하기 위한 서버리스 클라우드 데이터 웨어하우스입니다.
- JDBC - 데이터 연결 및 상호 작용에 Java API를 사용하는 관계형 데이터베이스 관리 시스템 (RDBMS)입니다.
- Kafka - 실시간 데이터 스트리밍 및 메시징에 사용되는 오픈 소스 스트림 처리 플랫폼입니다.
- MariaDB - 커뮤니티에서 개발한 MySQL 포크로 향상된 성능, 확장성 및 기능을 제공합니다.
- MongoDB - 높은 확장성, 유연성 및 성능을 제공하는 크로스 플랫폼 문서 지향 데이터베이스입니다.
- MongoDB Atlas - MongoDB 배포의 관리 및 확장을 간소화하는 MongoDB에서 제공하는 클라우드 기반 서비스형 데이터베이스(DBaaS)입니다.
- Microsoft SQL Server - 강력한 데이터 스토리지, 분석 및 보고 기능을 제공하는 Microsoft의 관계형 데이터베이스 관리 시스템(RDBMS)입니다.
- MySQL - 웹 애플리케이션에서 널리 사용되고 신뢰성과 확장성으로 잘 알려진 오픈 소스 관계형 데이터베이스 관리 시스템(RDBMS)입니다.

- 네트워크 - 네트워크 데이터 소스는 데이터 통합 플랫폼에서 액세스할 수 있는 네트워크 액세스 가능 리소스 또는 서비스를 나타냅니다.
- OpenSearch— OpenSearch 데이터 소스는 데이터에 연결하고 데이터를 OpenSearch 수집할 수 있는 애플리케이션입니다.
- Oracle - 오라클의 관계형 데이터베이스 관리 시스템(RDBMS)으로 강력한 데이터 스토리지, 분석 및 보고 기능을 제공합니다.
- PostgreSQL — 강력한 데이터 스토리지, 분석 및 보고 기능을 제공하는 오픈 소스 관계형 데이터베이스 관리 시스템(RDBMS)입니다.
- Salesforce — Salesforce는 영업, 고객 서비스, 전자 상거래 등에 도움이 되는 고객 관계 관리 (CRM) 소프트웨어를 제공합니다. Salesforce 사용자인 경우 Salesforce 계정에 연결할 수 있습니다. AWS Glue 그러면 Salesforce를 ETL 작업의 데이터 소스 또는 대상으로 사용할 수 있습니다. 이러한 작업을 실행하여 Salesforce와 AWS 서비스 또는 기타 지원되는 애플리케이션 간에 데이터를 전송하세요.
- SAP HANA - 빠른 데이터 처리, 고급 분석, 실시간 데이터 통합을 제공하는 인메모리 데이터베이스 및 분석 플랫폼입니다.
- Snowflake - 확장 가능한 고성능 데이터 스토리지 및 분석 서비스를 제공하는 클라우드 기반 데이터 웨어하우스입니다.
- Teradata - 고성능 데이터 저장, 분석 및 보고 기능을 제공하는 관계형 데이터베이스 관리 시스템 (RDBMS)입니다.
- Vertica - 빅데이터 분석을 위해 설계된 컬럼 중심의 분석 데이터 웨어하우스로, 빠른 쿼리 성능, 고급 분석 및 확장성을 제공합니다.

커넥터에 대한 연결 생성

AWS Glue 연결은 특정 데이터 스토어에 대한 연결 정보를 저장하는 Data Catalog 객체입니다. 연결은 로그인 자격 증명, URI 문자열, Virtual Private Cloud(VPC) 정보 등을 저장합니다. Data Catalog에서 연결을 생성하면 작업을 생성할 때마다 모든 연결 세부 정보를 지정해야 하는 수고를 덜 수 있습니다.

커넥터에 대한 연결을 생성하려면

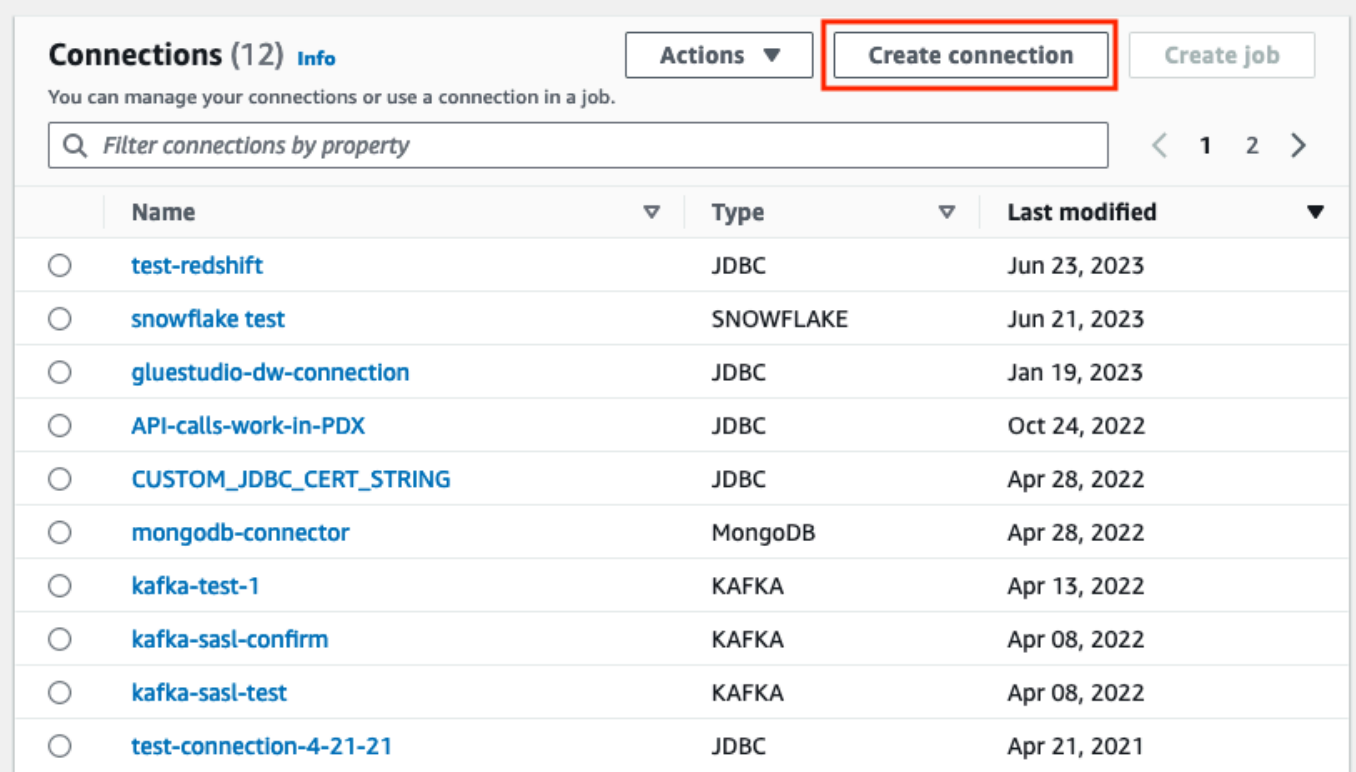
1. AWS Glue Studio 콘솔의 콘솔 탐색 창에서 커넥터(Connectors)를 선택합니다. 아웃바운드 연결 섹션에서 연결 생성을 선택합니다.
2. 데이터 연결 만들기 마법사의 1단계에서 연결을 만들려는 데이터 소스를 선택합니다. 사용 가능한 데이터 소스를 보는 방법에는 다음과 같이 여러 가지가 있습니다.

- 탭을 선택하여 사용 가능한 데이터 소스를 필터링합니다. 기본적으로 모든 커넥터가 선택됩니다.
- 목록을 토글하여 데이터 소스를 목록으로 보거나 그리드로 다시 전환하여 그리드 레이아웃에서 사용 가능한 커넥터를 볼 수 있습니다.
- 검색 창을 사용하여 데이터 소스의 목록의 범위를 좁힙니다. 입력할 때 일치하는 검색이 표시되고 일치하지 않는 소스는 보기에서 제거됩니다.

데이터 소스를 선택한 후 다음을 선택합니다.

3. 마법사의 2단계에서 연결을 구성합니다.

연결 세부 정보를 입력합니다. 선택한 커넥터 유형에 따라 추가 정보를 입력하라는 메시지가 나타납니다.



Connections (12) [Info](#) Actions ▾ Create connection Create job

You can manage your connections or use a connection in a job.

🔍 Filter connections by property < 1 2 >

	Name ▾	Type ▾	Last modified ▾
<input type="radio"/>	test-redshift	JDBC	Jun 23, 2023
<input type="radio"/>	snowflake test	SNOWFLAKE	Jun 21, 2023
<input type="radio"/>	gluestudio-dw-connection	JDBC	Jan 19, 2023
<input type="radio"/>	API-calls-work-in-PDX	JDBC	Oct 24, 2022
<input type="radio"/>	CUSTOM_JDBC_CERT_STRING	JDBC	Apr 28, 2022
<input type="radio"/>	mongodb-connector	MongoDB	Apr 28, 2022
<input type="radio"/>	kafka-test-1	KAFKA	Apr 13, 2022
<input type="radio"/>	kafka-sasl-confirm	KAFKA	Apr 08, 2022
<input type="radio"/>	kafka-sasl-test	KAFKA	Apr 08, 2022
<input type="radio"/>	test-connection-4-21-21	JDBC	Apr 21, 2021

4. 데이터 연결 만들기 마법사의 1단계에서 연결을 만들려는 데이터 소스를 선택합니다. 사용 가능한 데이터 소스는 여러 가지 방법으로 확인할 수 있습니다. 기본적으로 사용 가능한 모든 데이터 소스가 그리드 레이아웃으로 표시됩니다. 다른 방법:

- 목록을 토글하여 데이터 소스를 목록으로 보거나 그리드로 다시 전환하여 그리드 레이아웃에서 사용 가능한 커넥터를 볼 수 있습니다.

- 검색 창을 사용하여 데이터 소스의 목록의 범위를 좁힙니다. 입력할 때 일치하는 검색이 표시되고 일치하지 않는 소스는 보기에서 제거됩니다.

Choose data source

데이터 소스를 선택한 후 다음을 선택합니다.

5. 마법사의 2단계에서 연결을 구성합니다.

연결 세부 정보를 입력합니다. 선택한 커넥터 유형에 따라 추가 연결 정보를 입력해야 할 수도 있습니다. 여기에는 다음이 포함됩니다.

- 연결 세부 정보 - 이 필드는 연결 중인 데이터 소스에 따라 달라집니다. 예를 들어, Amazon DocumentDB 데이터베이스에 연결하는 경우 Amazon DocumentDB URL을 입력합니다. Amazon Aurora에 연결하는 경우 데이터베이스 인스턴스를 선택하고 데이터베이스 이름을 입력합니다. 필요한 연결 세부 정보는 다음과 Amazon Aurora 같습니다.

- 보안 인증 정보 유형 - 사용자 이름과 암호 또는 AWS Secrets Manager를 선택합니다. 요청된 인증 정보를 입력합니다.
- JDBC를 사용하는 커넥터의 경우 데이터 스토어에 대한 JDBC URL을 생성하는 데 필요한 정보를 입력합니다.

- Virtual Private Cloud(VPC)를 사용하는 경우 VPC에 대한 네트워크 정보를 입력합니다.
6. 마법사의 3단계에서 연결 속성을 설정합니다. 이 단계의 선택적 부분으로 설명과 태그를 추가할 수 있습니다. 이름은 필수이며 기본값으로 미리 채워집니다. 다음을 선택합니다.
 7. 연결 소스, 세부 정보 및 속성을 검토하십시오. 변경을 하려면 마법사의 편집 단계를 선택합니다. 준비되면 연결 생성을 선택합니다.

연결 생성을 선택합니다.

[커넥터(Connectors)] 페이지로 돌아가고 정보 배너가 생성된 연결을 나타냅니다. 이제 AWS Glue Studio 작업에서 연결을 사용할 수 있습니다.

Kafka 연결 생성

Kafka 연결을 생성할 때 드롭다운 메뉴에서 Kafka를 선택하면 구성할 추가 설정이 표시됩니다.

- Kafka 클러스터 세부 정보
- 인증
- 암호화(Encryption)
- 네트워크 옵션

Kafka 클러스터 세부 정보 구성(Configure Kafka cluster details)

1. 클러스터 위치를 선택합니다. Amazon Managed Streaming for Apache Kafka(MSK) (Amazon managed streaming for Apache Kafka (MSK)) 클러스터 또는 고객 관리형 Apache Kafka(Customer managed Apache Kafka) 클러스터 중에서 선택할 수 있습니다. Amazon Managed Streaming for Apache Kafka에 대한 자세한 내용은 [Amazon managed streaming for Apache Kafka \(MSK\)](#)(Amazon Managed Streaming for Apache Kafka(MSK))를 참조하세요.

Note

Amazon Managed Streaming for Apache Kafka는 TLS 및 SASL/SCRAM-SHA-512 인증 방법만 지원합니다.

Kafka cluster details [Info](#)

Cluster location

Amazon managed streaming for Apache Kafka (MSK)
 Customer managed Apache Kafka

Kafka bootstrap server URLs [Info](#)

A comma-separated list of bootstrap server URLs. Include the port number.

Enter list of URLs, separated by commas

Example: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-2.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094, b-3.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094

- Kafka 부트스트랩 서버의 URL을 입력합니다. 각 서버를 쉼표로 구분하여 둘 이상 입력할 수 있습니다. :<port number>를 추가하여 URL 끝에 포트 번호를 포함합니다.

예: b-1.vpc-test-2.034a88o.kafka-us-east-1.amazonaws.com:9094

인증 방법 선택(Select authentication method)

Authentication [Info](#)

Authentication method

Choose authentication method ▼

AWS Glue에서는 인증을 위해 SASL(Simple Authentication and Security Layer) 프레임워크를 지원합니다. SASL 프레임워크는 다양한 인증 메커니즘을 지원하며, AWS Glue는 SCRAM(사용자 이름 및 암호), GSSAPI(Kerberos 프로토콜), PLAIN(사용자 이름 및 암호) 프로토콜을 제공합니다.

드롭다운 메뉴에서 인증 방법을 선택할 때 다음 클라이언트 인증 방법을 선택할 수 있습니다.

- 없음(None) - 인증이 없습니다. 이는 테스트 목적으로 연결을 생성하는 경우에 유용합니다.
- SASL/SCRAM-SHA-512 - 인증 자격 증명을 지정하려면 이 인증 방법을 선택합니다. 두 가지 옵션을 사용할 수 있습니다.

- **AWS Secrets Manager 사용(권장)** - 이 옵션을 선택하면 AWS Secrets Manager에 보안 인증 정보를 저장하여 필요할 때 AWS Glue에서 정보에 액세스하도록 할 수 있습니다. SSL 또는 SASL 인증 자격 증명을 저장하는 보안 암호를 지정합니다.

Authentication Info

Authentication method
 SASL/SCRAM-SHA-512

Authentication credentials

Use AWS Secrets Manager (recommended)
 Store your token in AWS Secrets Manager, and let AWS Glue access it when needed.

Provide username and password directly
 Provide your username and password directly to AWS Glue.

Secret from [AWS Secrets Manager](#)

Search secret by name or type ARN

- 사용자 이름 및 암호를 직접 제공합니다.
- **SASL/GSSAPI (Kerberos)(SASL/GSSAPI(Kerberos))** - 이 옵션을 선택하면 keytab 파일, krb5.conf 파일의 위치를 선택하고 Kerberos 보안 주체 이름과 Kerberos 서비스 이름을 입력할 수 있습니다. keytab 파일과 krb5.conf 파일의 위치는 Amazon S3 위치에 있어야 합니다. MSK는 아직 SASL/GSSAPI를 지원하지 않으므로 이 옵션은 고객 관리형 Apache Kafka 클러스터에만 사용할 수 있습니다. 자세한 내용은 [MIT Kerberos Documentation: Keytab](#)(MIT Kerberos 설명서: Keytab)을 참조하세요.
- **SASL/PLAIN** - 인증 자격 증명을 지정하려면 이 인증 방법을 선택합니다. 두 가지 옵션을 사용할 수 있습니다.
 - **AWS Secrets Manager 사용(권장)** - 이 옵션을 선택하면 AWS Secrets Manager에 보안 인증 정보를 저장하여 필요할 때 AWS Glue에서 정보에 액세스하도록 할 수 있습니다. SSL 또는 SASL 인증 자격 증명을 저장하는 보안 암호를 지정합니다.
 - 사용자 이름 및 암호를 직접 제공합니다.
- **SSL 클라이언트 인증(SSL Client Authentication)** - 이 옵션을 선택하면 Amazon S3를 검색하여 Kafka 클라이언트 키 스토어의 위치를 선택할 수 있습니다. 선택 사항으로 Kafka 클라이언트 키 스토어 암호와 Kafka 클라이언트 키 암호를 입력할 수 있습니다.

Authentication Info

Authentication method

SSL client authentication ▼

Kafka client keystore location

🔍 View ↗ Browse S3

Path must be in the form s3://bucket/prefix/path/. It must end with the file name and .jks extension.

Kafka client keystore password - *optional*

Kafka client key password - *optional*

암호화 설정 구성(Configure encryption settings)

1. Kafka 연결에 SSL 연결이 필요한 경우 SSL 연결 필요(Require SSL connection) 확인란을 선택합니다. SSL을 통해 연결할 수 없으면 연결이 실패합니다. 암호화용 SSL은 모든 인증 방법(SASL/SCRAM-SHA-512, SASL/GSSAPI, SASL/PLAIN 또는 SSL 클라이언트 인증)과 함께 사용할 수 있으며 선택 사항입니다.

인증 방법이 SSL 클라이언트 인증(SSL client authentication)으로 설정된 경우 이 옵션은 자동으로 선택되며 변경을 방지하기 위해 비활성화됩니다.

2. (선택 사항). 인증 기관(CA)의 사설 인증서 위치를 선택합니다. 인증 위치는 S3 위치에 있어야 합니다. 연결된 S3 버킷에서 파일을 선택하려면 찾아보기(Browse)를 선택합니다. 경로는 s3://bucket/prefix/filename.pem 형식이어야 합니다. 파일 이름과 .pem 확장자로 끝나야 합니다.
3. 인증 기관(CA)의 인증서 검증을 건너뛰도록 선택할 수 있습니다. 인증 기관(CA)의 인증서 검증 건너뛰기(Skip validation of certificate from certificate authority (CA)) 확인란을 선택합니다. 이 확인란을 선택하지 않으면 AWS Glue는 다음 세 가지 알고리즘에 대해 인증서를 검증합니다.
 - SHA256withRSA
 - SHA384withRSA
 - SHA512withRSA

Encryption Info

Require SSL connection
The connection will fail if it's unable to connect over SSL.

Location of private certificate from certificate authority (CA) - *optional*

Path must be in the form s3://bucket/prefix/path/. It must end with the file name and .pem extension.

Skip validation of certificate from certificate authority (CA)
AWS Glue validates for three algorithms: SHA256withRSA, SHA384withRSA and SHA512withRSA.

(선택 사항) 네트워크 옵션((Optional) Network options)

다음은 VPC, 서브넷 및 보안 그룹을 구성하는 선택적 단계입니다. AWS Glue 작업이 VPC(Virtual Private Cloud) 서브넷의 Amazon EC2 인스턴스에서 실행되어야 하는 경우 추가 VPC별 구성 정보를 제공해야 합니다.

1. 데이터 원본이 포함된 VPC(Virtual Private Cloud)를 선택합니다.
2. VPC가 있는 서브넷을 선택합니다.
3. 하나 이상의 보안 그룹을 선택하여 VPC 서브넷의 데이터 스토어에 대한 액세스를 허용합니다. 보안 그룹은 서브넷에 연결된 ENI에 연결됩니다. 모든 TCP 포트에 대해 자체 참조 인바운드 규칙이 있는 보안 그룹을 하나 이상 선택해야 합니다.

▼ Network options - *optional*

If your AWS Glue job needs to run on [Amazon Elastic Compute Cloud](#) (EC2) instances in a virtual private cloud (VPC) subnet, you must provide additional VPC-specific configuration information.

VPC [Info](#)

Choose the virtual private cloud that contains your data source.

Subnet [Info](#)

Choose the subnet within your VPC.

Security groups [Info](#)

Choose one or more security groups to allow access to the data store in your VPC subnet. Security groups are associated to the ENI attached to your subnet. You must choose at least one security group with a self-referencing inbound rule for all TCP ports.

사용자 정의 커넥터로 작업 작성

AWS Glue Studio에서 데이터 원본 노드와 데이터 대상 노드 둘 다에 커넥터와 연결을 사용할 수 있습니다.

주제

- [데이터 원본에 커넥터를 사용하는 작업 생성](#)
- [커넥터를 사용하는 노드의 소스 속성 구성](#)
- [커넥터를 사용하는 노드의 대상 속성 구성](#)

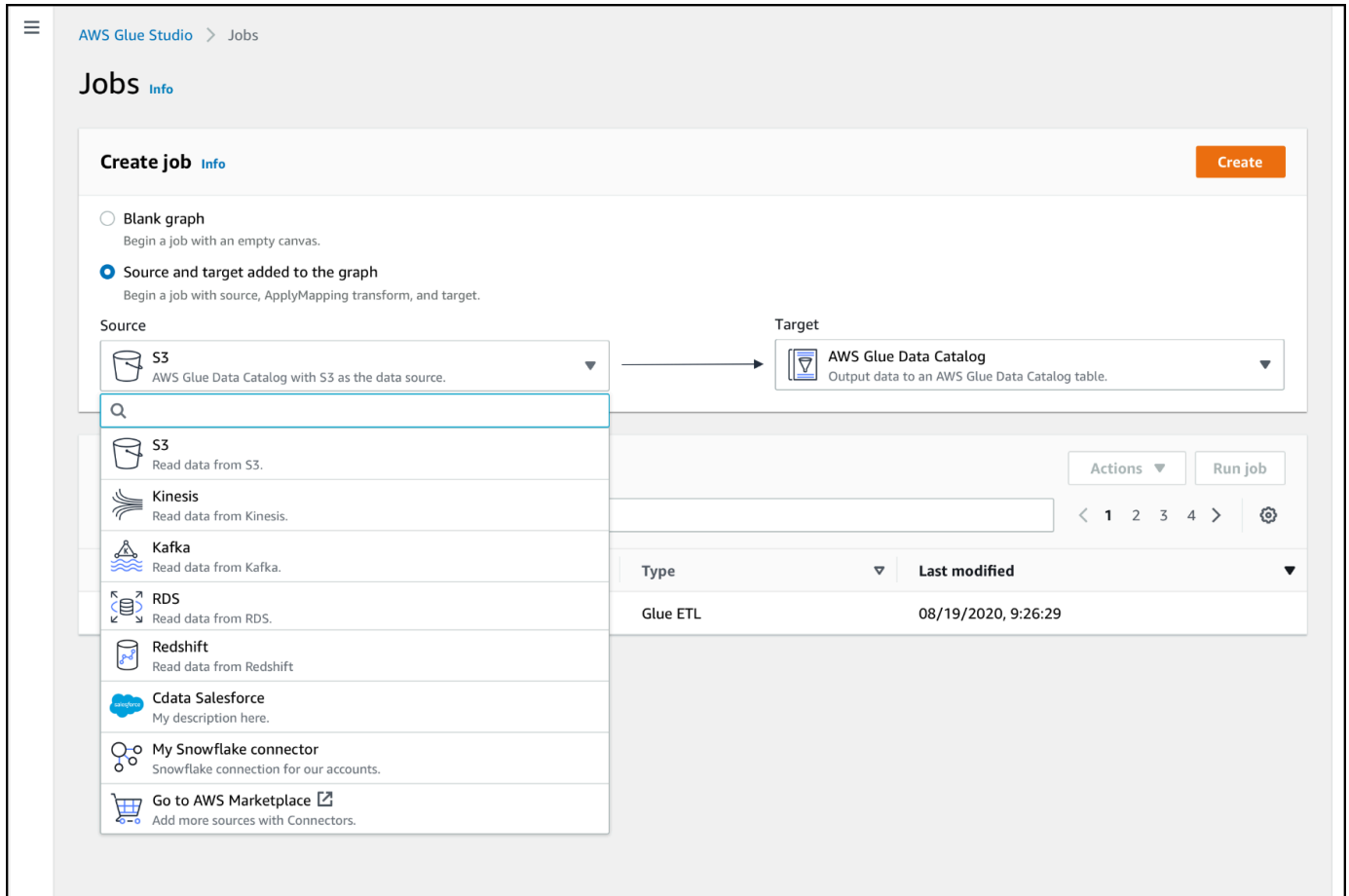
데이터 원본에 커넥터를 사용하는 작업 생성

새 작업을 생성할 때 데이터 원본 및 데이터 대상에 대한 커넥터를 선택할 수 있습니다.

데이터 원본 또는 데이터 대상에 대한 커넥터를 사용하는 작업을 생성하려면

1. <https://console.aws.amazon.com/gluestudio/> 에서 AWS Management Console 로그인하고 AWS Glue Studio 콘솔을 여십시오.
2. [커넥터(Connectors)] 페이지의 [연결(Your connections)] 리소스 목록에서 작업에 사용할 연결을 선택한 다음 [작업 생성(Create job)]을 선택합니다.

또는 AWS Glue Studio 작업(Jobs) 페이지의 작업 생성(Create job)에서 그래프에 추가된 소스 및 대상(Source and target added to the graph)을 선택합니다. [소스(Source)] 드롭다운 목록에서 작업에 사용하려는 사용자 정의 커넥터를 선택합니다. [대상(Target)]에 대한 커넥터를 선택할 수도 있습니다.



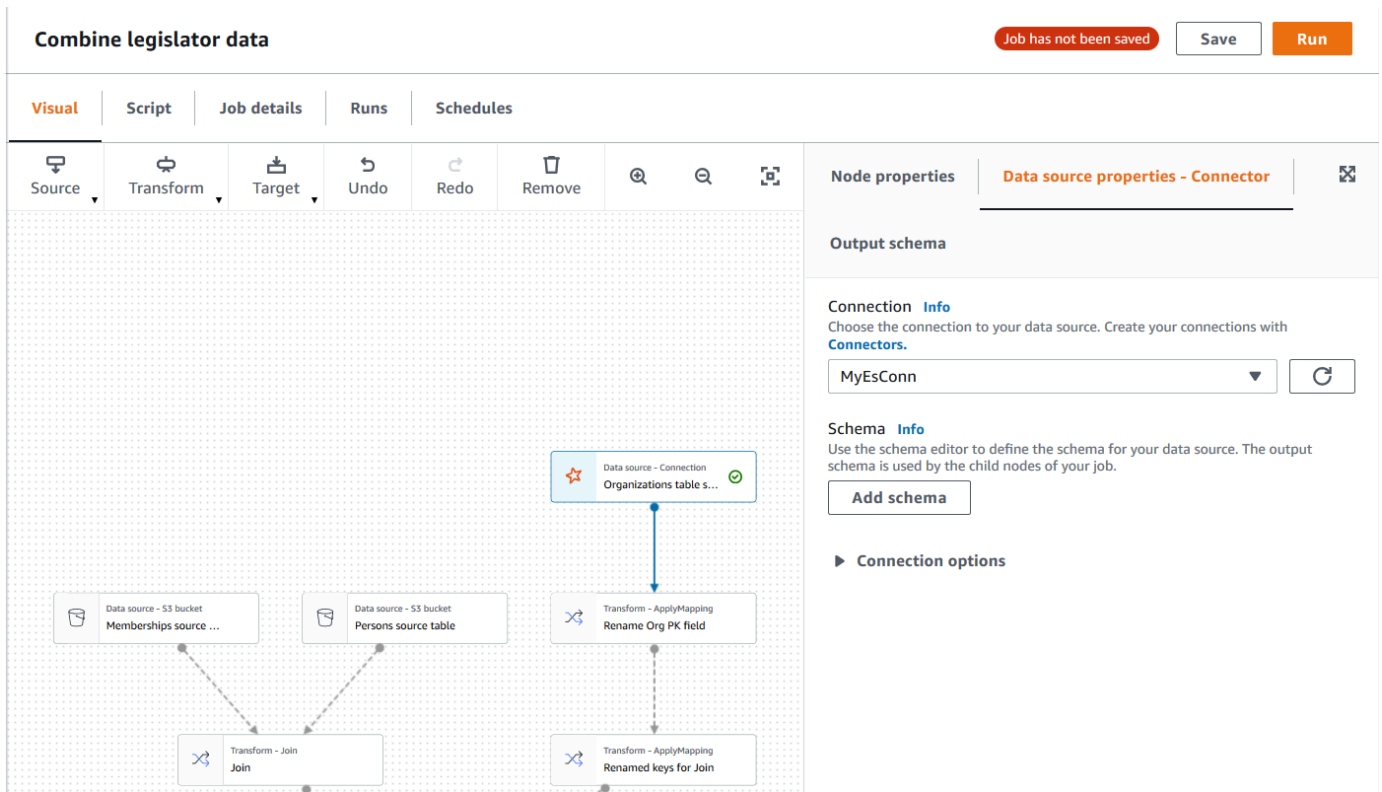
3. [생성(Create)]을 선택하여 시각적 작업 편집기를 엽니다.
4. 커넥터를 사용하는 노드의 소스 속성 구성에 설명된 대로 데이터 원본 노드를 구성합니다.
5. AWS Glue Studio을(를) 사용한 시각적 ETL에 설명된 대로 변환, 추가 데이터 스토어 및 데이터 대상 추가하여 ETL 작업을 계속 생성합니다.
6. 작업 속성 수정에 설명된 대로 작업 속성을 구성하여 작업 실행 환경을 사용자 지정합니다.
7. 작업을 저장하고 실행합니다.

커넥터를 사용하는 노드의 소스 속성 구성

데이터 원본에 대해 커넥터를 사용하는 작업을 생성한 후 시각적 작업 편집기는 커넥터에 대해 구성된 데이터 원본 노드가 있는 작업 그래프를 표시합니다. 해당 노드에 대한 데이터 원본 속성을 구성해야 합니다.

커넥터를 사용하는 데이터 원본 노드의 속성을 구성하려면

1. 작업 그래프에서 커넥터 데이터 원본 노드를 선택하거나 새 노드를 추가하고 [노드 유형(Node type)]에 대한 커넥터를 선택하십시오. 그런 다음 오른쪽의 노드 세부 정보 패널에서 [데이터 원본 속성(Data source properties)] 탭을 선택합니다(아직 선택하지 않은 경우).



2. [데이터 원본 속성(Data source properties)] 탭에서 이 작업에 사용할 연결을 선택합니다.

각 연결 유형에 필요한 추가 정보를 입력합니다.

JDBC

- [데이터 원본 입력 유형(Data source input type)]: 테이블 이름 또는 SQL 쿼리를 데이터 원본으로 제공하도록 선택합니다. 선택 항목에 따라 다음 추가 정보를 제공해야 합니다.

- [테이블 이름(Table name)]: 데이터 원본에 있는 테이블의 이름입니다. 데이터 소스에서 테이블이라는 용어를 사용하지 않는 경우 사용자 지정 커넥터 사용 정보 (에서 사용 가능 AWS Marketplace) 에 표시된 대로 적절한 데이터 구조의 이름을 제공하십시오.
- 필터 조건자(Filter predicate): 데이터의 하위 집합을 검색하는 데 사용되는 WHERE 절과 유사하게 데이터 원본을 읽을 때 사용할 조건 절입니다.
- [쿼리 코드(Query code)]: 데이터 원본에서 특정 데이터 집합을 검색하는 데 사용할 SQL 쿼리를 입력합니다. 기본 SQL 쿼리의 예는 다음과 같습니다.

```
SELECT column_list FROM
           table_name WHERE where_clause
```

- 스키마(Schema): AWS Glue Studio는 Data Catalog 테이블에서 메타데이터 정보를 검색하는 대신 연결에 저장된 정보를 사용하여 데이터 원본에 액세스하므로 데이터 원본에 대한 스키마 메타데이터를 제공해야 합니다. [스키마 추가(Add schema)]를 선택하여 스키마 편집기를 엽니다.

스키마 편집기를 사용하는 방법에 대한 지침은 [사용자 정의 변환 노드에서 스키마 편집](#) 섹션을 참조하세요.

- [파티션 열(Partition column)]: (선택 사항) [파티션 열(Partition column)], [하한(Lower bound)], [상한(Upper bound)] 및 [파티션 수(Number of partitions)]에 대한 값을 제공하여 데이터 읽기를 분할하도록 선택할 수 있습니다.

lowerBound 및 upperBound 값은 테이블의 행을 필터링하는 것이 아니라 파티션 스트라이드를 결정하는 데 사용됩니다. 테이블의 모든 행이 분할되어 반환됩니다.

Note

열 분할은 데이터를 읽는 데 사용되는 쿼리에 추가 분할 조건을 추가합니다. 테이블 이름 대신 쿼리를 사용하는 경우 쿼리가 지정된 분할 조건에서 작동하는지 확인해야 합니다. 예:

- 쿼리 포맷이 "SELECT col1 FROM table1"이면 파티션 열을 사용하는 쿼리 끝에 WHERE 절을 추가하여 쿼리를 테스트합니다.
- 쿼리 포맷이 "SELECT col1 FROM table1 WHERE col2=val"이면 AND와 파티션 열을 사용하는 표현식으로 WHERE 절을 확장하여 쿼리를 테스트합니다.

- [데이터 유형 캐스팅(Data type casting)]: 데이터 원본에서 JDBC에서 사용할 수 없는 데이터 유형을 사용하는 경우 이 섹션을 사용하여 데이터 원본의 데이터 유형을 JDBC 데이터 유형으로 변환하는 방법을 지정합니다. 최대 50개의 데이터 유형 변환을 지정할 수 있습니다. 동일한 데이터 유형을 사용하는 데이터 원본의 모든 열을 동일한 방식으로 변환합니다.

예를 들어 Float 데이터 유형을 사용하는 데이터 원본에 3개의 열이 있고 Float 데이터 유형을 JDBC String 데이터 유형으로 변환해야 한다고 지정하면 Float 데이터 유형을 사용하는 3개의 열이 모두 String 데이터 유형으로 변환됩니다.

- [작업 북마크 키(Job bookmark keys)]: AWS Glue는 작업 북마크로 상태 정보를 유지하고 이전 데이터의 재처리를 방지합니다. 하나 이상의 열을 북마크 키로 지정합니다. AWS Glue Studio는 북마크 키를 사용하여 이전에 ETL 작업을 실행할 때 이미 처리된 데이터를 추적합니다. 사용자 정의 북마크 키에 사용하는 모든 열은 엄격하게 단조롭게 증가하거나 감소해야 하지만 간격은 허용됩니다.

여러 북마크 키를 입력하면 결합되어 단일 복합 키를 형성합니다. 복합 작업 북마크 키에 중복 열이 포함되면 안 됩니다. 북마크 키를 지정하지 않으면 AWS Glue Studio는 기본값으로 기본 키를 북마크 키로 사용합니다. 단, 기본 키가 간격 없이 순차적으로 증가하거나 감소해야 합니다. 테이블에 기본 키가 없지만 작업 북마크 속성이 활성화된 경우 사용자 정의 작업 북마크 키를 제공해야 합니다. 그렇지 않으면 기본값으로 사용할 기본 키 검색이 실패하고 작업 실행이 실패합니다.

- [작업 북마크 키 정렬 순서(Job bookmark keys sorting order)]: 키 값이 순차적으로 증가할지 또는 감소할지를 선택합니다.

Spark

- 스키마(Schema): AWS Glue Studio는 Data Catalog 테이블에서 메타데이터 정보를 검색하는 대신 연결에 저장된 정보를 사용하여 데이터 원본에 액세스하므로 데이터 원본에 대한 스키마 메타데이터를 제공해야 합니다. [스키마 추가(Add schema)]를 선택하여 스키마 편집기를 엽니다.

스키마 편집기를 사용하는 방법에 대한 지침은 [사용자 정의 변환 노드에서 스키마 편집](#) 섹션을 참조하세요.

- [연결 옵션(Connection options)]: 추가 연결 정보 또는 옵션을 제공하기 위해 필요에 따라 추가 키-값 페어를 입력합니다. 예를 들어 데이터베이스 이름, 테이블 이름, 사용자 이름 및 암호를 입력할 수 있습니다.

예를 들어, 의 경우 에 OpenSearch 설명된 대로 다음과 같은 키-값 쌍을 입력합니다. [the section called “자습서: AWS Glue Connector for Elasticsearch 사용”](#)

- es.net.http.auth.user : *username*
- es.net.http.auth.pass : *password*
- es.nodes : https://<*Elasticsearch endpoint*>
- es.port : 443
- path: <*Elasticsearch resource*>
- es.nodes.wan.only : true

사용할 최소 연결 옵션의 예는 연결에서 일반적으로 제공하는 연결 옵션을 보여주는 샘플 테스트 스크립트 [MinimalSparkConnectorTest.scala](#) GitHub on을 참조하십시오.

Athena

- [테이블 이름(Table name)]: 데이터 원본에 있는 테이블의 이름입니다. CloudWatch Athena-logs에서 읽는 데 커넥터를 사용하는 경우 테이블 이름을 입력합니다. all_log_streams
- [Athena 스키마 이름(Athena schema name)]: Athena 데이터 원본에서 테이블을 포함하는 데이터베이스에 해당하는 스키마를 선택합니다. CloudWatch Athena-logs에서 읽는 데 커넥터를 사용하는 경우 다음과 비슷한 스키마 이름을 입력합니다. /aws/glue/*name*
- 스키마(Schema): AWS Glue Studio는 Data Catalog 테이블에서 메타데이터 정보를 검색하는 대신 연결에 저장된 정보를 사용하여 데이터 원본에 액세스하므로 데이터 원본에 대한 스키마 메타데이터를 제공해야 합니다. [스키마 추가(Add schema)]를 선택하여 스키마 편집기를 엽니다.

스키마 편집기를 사용하는 방법에 대한 지침은 [사용자 정의 변환 노드에서 스키마 편집](#) 섹션을 참조하세요.

- [추가 연결 옵션(Additional connection options)]: 추가 연결 정보 또는 옵션을 제공하기 위해 필요에 따라 추가 키-값 페어를 입력합니다.

예를 보려면 <https://github.com/aws-samples/aws-glue-samples/tree/master/GlueCustomConnectors/Development/Athena>의 README.md 파일을 참조하십시오. 이 문서의 단계에서 샘플 코드는 최소한의 필수 연결 옵션인 tableName, schemaName 및 className을 보여줍니다. 코드 예제에서는 이러한 옵션을 optionsMap 변수의 일부로 지정하지만 연결에 대해 지정한 다음 연결을 사용할 수 있습니다.

3. (선택 사항) 필수 정보를 제공한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터 원본에 대한 결과 데이터 스키마를 볼 수 있습니다. 이 탭에 표시된 스키마는 작업 그래프에 추가하는 모든 하위 노드에서 사용됩니다.
4. (선택 사항) 노드 속성과 데이터 원본 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 데이터 원본에서 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

커넥터를 사용하는 노드의 대상 속성 구성

데이터 대상 유형에 커넥터를 사용하는 경우 데이터 대상 노드의 속성을 구성해야 합니다.

커넥터를 사용하는 데이터 대상 노드의 속성을 구성하려면

1. 작업 그래프에서 커넥터 데이터 대상 노드를 선택합니다. 그런 다음 오른쪽의 노드 세부 정보 패널에서 [데이터 대상 속성(Data target properties)] 탭을 선택합니다(아직 선택하지 않은 경우).
2. [데이터 대상 속성(Data target properties)] 탭에서 대상에 쓰는 데 사용할 연결을 선택합니다.

각 연결 유형에 필요한 추가 정보를 입력합니다.

JDBC

- [연결(Connection)]: 커넥터에 사용할 연결을 선택합니다. 연결을 생성하는 방법에 대한 자세한 내용은 [커넥터에 대한 연결 생성](#) 섹션을 참조하세요.
- [테이블 이름(Table name)]: 데이터 대상에 있는 테이블의 이름입니다. 데이터 대상이 테이블이라는 용어를 사용하지 않는 경우 사용자 지정 커넥터 사용 정보 (에서 사용 가능) 에 표시된 대로 적절한 데이터 구조의 이름을 제공하십시오. AWS Marketplace
- [배치 크기(Batch size)](선택 사항): 단일 작업으로 대상 테이블에 삽입할 행 또는 레코드 수를 입력합니다. 기본값은 1000행입니다.

Spark

- [연결(Connection)]: 커넥터에 사용할 연결을 선택합니다. 이전에 연결을 생성하지 않은 경우 [연결 생성(Create connection)]을 선택하여 생성합니다. 연결을 생성하는 방법에 대한 자세한 내용은 [커넥터에 대한 연결 생성](#) 섹션을 참조하세요.

- [연결 옵션(Connection options)]: 추가 연결 정보 또는 옵션을 제공하기 위해 필요에 따라 추가 키-값 페어를 입력합니다. 데이터베이스 이름, 테이블 이름, 사용자 이름 및 암호를 입력할 수 있습니다.

예를 들어, 의 경우 에 OpenSearch 설명된 대로 다음과 같은 키-값 쌍을 입력합니다. [the section called “자습서: AWS Glue Connector for Elasticsearch 사용”](#)

- es.net.http.auth.user : *username*
- es.net.http.auth.pass : *password*
- es.nodes : https://<*Elasticsearch endpoint*>
- es.port : 443
- path: <*Elasticsearch resource*>
- es.nodes.wan.only : true

사용할 최소 연결 옵션의 예는 연결에서 일반적으로 제공하는 연결 옵션을 보여주는 샘플 테스트 스크립트 [MinimalSparkConnectorTest.scala](#) GitHub on을 참조하십시오.

3. 필수 정보를 제공한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터 원본에 대한 결과 데이터 스키마를 볼 수 있습니다.

커넥터 및 연결 관리

AWS Glue의 커넥터 페이지를 사용하여 커넥터와 연결을 관리합니다.

주제

- [커넥터 및 연결 세부 정보 보기](#)
- [커넥터 및 연결 편집](#)
- [커넥터 및 연결 삭제](#)
- [커넥터에 대한 구독 취소](#)

커넥터 및 연결 세부 정보 보기

[커넥터(Connectors)] 페이지의 [커넥터(Your connectors)] 및 [연결(Your connections)] 리소스 테이블에서 커넥터 및 연결에 대한 요약 정보를 볼 수 있습니다. 세부 정보를 보려면 다음 단계를 수행합니다.

커넥터 또는 연결 세부 정보를 보려면

1. AWS Glue Studio 콘솔의 콘솔 탐색 창에서 커넥터(Connectors)를 선택합니다.
2. 자세한 정보를 보려는 커넥터 또는 연결을 선택합니다.
3. [작업(Actions)]을 선택하고 [세부 정보 보기(View details)]를 선택하여 해당 커넥터 또는 연결에 대한 세부 정보 페이지를 엽니다.
4. 세부 정보 페이지에서 커넥터 또는 연결을 Edit(편집)하거나 Delete(삭제)하도록 선택할 수 있습니다.
 - 커넥터의 경우 [연결 생성(Create connection)]을 선택하여 커넥터를 사용하는 새 연결을 생성할 수 있습니다.
 - 연결의 경우 [작업 생성(Create job)]을 클릭하여 연결을 사용하는 작업을 생성할 수 있습니다.

커넥터 및 연결 편집

[커넥터(Connectors)] 페이지를 사용하여 커넥터와 연결에 저장된 정보를 변경합니다.

커넥터 또는 연결을 수정하려면

1. AWS Glue Studio 콘솔의 콘솔 탐색 창에서 커넥터(Connectors)를 선택합니다.
2. 변경하려는 커넥터나 연결을 선택합니다.
3. 작업을 선택한 후 편집을 선택합니다.

[세부 정보 보기(View details)]를 선택하고 커넥터 또는 연결 세부 정보 페이지에서 [편집(Edit)]을 선택할 수도 있습니다.

4. [커넥터 편집(Edit connector)] 또는 [연결 편집(Edit connection)] 페이지에서 정보를 업데이트한 다음 [저장(Save)]을 선택합니다.

커넥터 및 연결 삭제

[커넥터(Connectors)] 페이지를 사용하여 커넥터와 연결을 삭제합니다. 커넥터를 삭제하면 해당 커넥터에 대해 생성된 연결도 모두 삭제되어야 합니다.

AWS Glue Studio에서 커넥터를 제거하려면

1. AWS Glue Studio 콘솔의 콘솔 탐색 창에서 커넥터(Connectors)를 선택합니다.
2. 삭제하려는 커넥터나 연결을 선택합니다.

3. 작업을 선택한 후 삭제를 선택합니다.

[세부 정보 보기(View details)]를 선택하고 커넥터 또는 연결 세부 정보 페이지에서 [삭제(Delete)]를 선택할 수도 있습니다.

4. Delete를 입력하여 커넥터 또는 연결 제거를 확인한 다음 [삭제(Delete)]를 선택합니다.

커넥터를 삭제하면 해당 커넥터에 대해 생성된 연결도 모두 삭제됩니다.

삭제된 연결을 사용하는 모든 작업은 더 이상 작동하지 않습니다. 작업을 편집하여 다른 데이터 스토어를 사용하거나 작업을 제거할 수 있습니다. 작업을 삭제하는 방법에 대한 자세한 정보는 [작업 삭제](#) 섹션을 참조하세요.

커넥터를 삭제해도 AWS Marketplace에서 커넥터에 대한 구독이 취소되지는 않습니다. 삭제된 커넥터에 대한 구독을 제거하려면 [커넥터에 대한 구독 취소](#)의 지침을 따릅니다.

커넥터에 대한 구독 취소

에서 AWS Glue Studio 연결 및 커넥터를 삭제한 후 커넥터가 더 이상 필요하지 AWS Marketplace 않은 경우 구독을 취소할 수 있습니다.

Note

커넥터에 대한 구독을 취소해도 계정에서 커넥터나 연결이 제거되지는 않습니다. 커넥터 및 관련 연결을 사용하는 모든 작업은 더 이상 커넥터를 사용할 수 없으며 실패합니다.

에서 커넥터를 구독 취소하거나 재가입하려면 먼저 해당 AWS Marketplace AWS Marketplace 제품과 관련된 기존 연결 및 커넥터를 삭제해야 합니다.

의 커넥터 구독을 취소하려면 AWS Marketplace

1. <https://console.aws.amazon.com/marketplace> 에서 AWS Marketplace 콘솔에 로그인합니다.
2. [구독 관리(Manage subscriptions)]를 선택합니다.
3. [구독 관리(Manage subscriptions)] 페이지에서 취소하려는 커넥터 구독 옆에 있는 [관리(Manage)]를 선택합니다.
4. 그런 다음 [작업(Actions)]을 선택하고 [구독 취소(Cancel Subscription)]를 선택합니다.
5. 실행 중인 인스턴스가 계정에 청구됨을 확인하는 확인란을 선택한 다음 [예, 구독을 취소합니다(Yes, cancel subscription)]를 선택합니다.

사용자 정의 커넥터 개발

데이터 스토어에서 데이터를 읽거나 데이터를 쓰고 AWS Glue Studio 작업에 사용할 데이터를 포맷하는 코드를 작성할 수 있습니다. Spark, Athena 및 JDBC 데이터 스토어용 커넥터를 생성할 수 있습니다. 예 게시된 샘플 코드는 구현에 필요한 기본 인터페이스의 개요를 GitHub 제공합니다.

커넥터 코드를 생성하려면 로컬 개발 환경이 필요합니다. 모든 IDE 또는 명령줄 편집기를 사용하여 커넥터를 작성할 수 있습니다. 개발 환경의 예는 다음과 같습니다.

- AWS Glue Developer Guide의 [Developing Locally with Scala](#)에 설명된 대로 로컬 AWS Glue ETL Maven 라이브러리가 있는 로컬 Scala 환경.
- IntelliJ IDE(<https://www.jetbrains.com/idea/>)에서 다운로드).

주제

- [Spark 커넥터 개발](#)
- [Athena 커넥터 개발](#)
- [JDBC 커넥터 개발](#)
- [AWS Glue Studio에서 사용자 지정 커넥터를 사용하는 예제](#)
- [를 위한 커넥터 개발 AWS GlueAWS Marketplace](#)

Spark 커넥터 개발

Spark DataSource API V2 (Spark 2.4) 를 사용하여 Spark 커넥터를 생성하여 데이터를 읽을 수 있습니다.

사용자 정의 Spark 커넥터 생성

[스파크 커넥터 개발을 위한 AWS Glue GitHub 샘플 라이브러리의 단계를 따르십시오. 이 라이브러리는 https://github.com/aws-samples/ /tree/master/ /Development/Spark/README.md에 있습니다. aws-glue-samples GlueCustomConnectors](https://github.com/aws-samples/ /tree/master/ /Development/Spark/README.md)

Athena 커넥터 개발

AWS Glue 및 AWS Glue Studio에서 사용자 지정 데이터 원본을 쿼리하는 데 사용할 Athena 커넥터를 생성할 수 있습니다.

사용자 정의 Athena 커넥터 생성

Athena 커넥터 개발을 위한 AWS Glue GitHub 샘플 라이브러리의 단계를 따르십시오. 이 라이브러리는 <https://github.com/aws-samples/aws-glue-samples/tree/master/Develoment/Athena>에 있습니다.

JDBC 커넥터 개발

JDBC를 사용하여 데이터 스토어에 액세스하는 커넥터를 생성할 수 있습니다.

사용자 정의 JDBC 커넥터를 생성하려면

1. 로컬 개발 환경에 AWS Glue Spark 런타임 라이브러리를 설치합니다. <https://github.com/aws-samples/aws-glue-samples/tree/master/development/README.md>의 샘플 라이브러리에 있는 지침을 참조하십시오. [AWS Glue GitHub aws-glue-samples GlueCustomConnectors GlueSparkRuntime](#)
2. 데이터 원본에서 데이터 검색을 담당하는 JDBC 드라이버를 구현합니다. Java SE 8에 대한 [Java 설명서](#)를 참조하세요.

AWS Glue Studio가 커넥터를 찾는 데 사용하는 코드 내의 진입점을 생성합니다. [클래스 이름 (Class name)] 필드는 JDBC 드라이버의 전체 경로여야 합니다.

3. GlueContext API를 사용하여 커넥터로 데이터를 읽습니다. 사용자는 필요한 경우 AWS Glue Studio 콘솔에서 더 많은 입력 옵션을 추가하여 데이터 원본에 대한 연결을 구성할 수 있습니다. 사용자 정의 JDBC 커넥터를 사용하여 JDBC 데이터베이스를 읽고 쓰는 방법을 보여주는 코드 예제는 사용자 정의 [및](#) ConnectionType 값을 참조하십시오. AWS Marketplace

AWS Glue Studio에서 사용자 지정 커넥터를 사용하는 예제

사용자 정의 커넥터 사용 예는 다음 블로그를 참조하세요.

- [AWS Glue를 사용하여 데이터 저장소용 사용자 지정 커넥터 개발, 테스트 및 배포](#)
- Apache Hudi: [AWS Glue 사용자 정의 커넥터를 사용하여 Apache Hudi 테이블에 쓰기](#)
- Google BigQuery: [AWS Glue 사용자 지정 커넥터를 BigQuery 사용하여 Google에서 Amazon S3로 데이터 마이그레이션](#)
- Snowflake(JDBC): [Snowflake 및 AWS Glue를 사용하여 데이터 변환 수행](#)
- SingleStore: [및 를 사용하여 빠른 ETL 구축 SingleStore AWS Glue](#)
- Salesforce: [AWS Glue와 함께 CData JDBC 사용자 정의 커넥터를 사용하여 Salesforce 데이터를 Amazon S3에 수집 -](#)
- MongoDB: [Amazon DocumentDB\(MongoDB 호환성 포함\) 및 MongoDB를 사용하여 AWS Glue Spark ETL 작업 구축](#)

- Amazon Relational Database Service (아마존 RDS): 아마존 RDS용 [자체 JDBC 드라이버를 가져와 AWS Glue 스파크 ETL 작업을 구축하세요](#)
- MySQL (JDBC): <https://github.com/aws-samples/ /blob/master/ /Development/Spark/ sql.Scala aws-glue-samples GlueCustomConnectors SparkConnectorMy>

를 위한 커넥터 개발 AWS GlueAWS Marketplace

AWS 파트너는 사용자 지정 커넥터를 만들고 업로드하여 AWS Glue 고객에게 AWS Marketplace 판매할 수 있습니다.

커넥터 코드를 개발하는 프로세스는 사용자 정의 커넥터와 동일하지만 커넥터 코드를 업로드하고 확인하는 프로세스가 더 세부적입니다. GitHub 웹 사이트의 [커넥터 AWS Marketplace생성에](#) 있는 지침을 참조하십시오.

AWS Glue Studio에서 커넥터와 연결을 사용하는 경우의 제한 사항

에서 사용자 정의 커넥터 또는 커넥터를 사용하는 AWS Marketplace경우 다음 제한 사항에 유의하십시오.

- testConnection API는 사용자 정의 커넥터에 대해 생성된 연결에서 지원되지 않습니다.
- Data Catalog 연결 암호 암호화는 사용자 정의 커넥터에서 지원되지 않습니다.
- JDBC 커넥터를 사용하는 데이터 원본 노드에 대한 필터 조건자를 지정하는 경우 작업 북마크를 사용할 수 없습니다.
- Marketplace 연결 생성은 AWS Glue Studio 사용자 인터페이스 외부에서 지원되지 않습니다.

시각적 ETL 작업을 사용하여 데이터 소스에 연결

새 작업을 생성하는 동안 AWS Glue에서 시각적 ETL 작업을 편집할 때 연결을 사용하여 데이터에 연결할 수 있습니다. 커넥터를 사용하여 데이터를 읽는 소스 노드와 데이터 쓰기 위치를 지정하는 대상 노드를 추가하여 이 작업을 수행할 수 있습니다.

주제

- [데이터 소스 노드의 속성 수정하기](#)
- [데이터 원본에 데이터 카탈로그 테이블 사용](#)
- [데이터 원본에 커넥터 사용](#)
- [데이터 원본에 Amazon S3의 파일 사용](#)

- [스트리밍 데이터 원본 사용](#)
- [참조](#)

데이터 소스 노드의 속성 수정하기

데이터 원본 속성을 지정하려면 먼저 작업 다이어그램에서 데이터 원본 노드를 선택합니다. 그런 다음 노드 세부 정보 패널의 오른쪽에서 노드 속성을 구성합니다.

데이터 원본 노드의 속성을 수정하려면

1. 새 작업 또는 저장된 작업의 시각적 편집기로 이동합니다.
2. 작업 다이어그램에서 데이터 원본 노드를 선택합니다.
3. 노드 세부 정보 패널에서 [노드 속성(Node properties)] 탭을 선택하고 다음 정보를 입력합니다.
 - [이름(Name)]: (선택 사항) 작업 다이어그램의 노드와 연결할 이름을 입력합니다. 이 이름은 이 작업에 대한 모든 노드에서 고유해야 합니다.
 - [노드 유형(Node type)]: 노드 유형에 따라 노드에서 수행되는 작업이 결정됩니다. [노드 유형(Node type)]에 대한 옵션 목록에서 [데이터 원본(Data source)] 제목 아래에 나열된 값 중 하나를 선택합니다.
4. [데이터 원본 속성(Data source properties)] 정보를 구성합니다. 자세한 내용은 다음 단원을 참조하세요.
 - [데이터 원본에 데이터 카탈로그 테이블 사용](#)
 - [데이터 원본에 커넥터 사용](#)
 - [데이터 원본에 Amazon S3의 파일 사용](#)
 - [스트리밍 데이터 원본 사용](#)
5. (선택 사항) 노드 속성과 데이터 원본 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터 원본에 대한 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.
6. (선택 사항) 노드 속성과 데이터 원본 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 데이터 원본에서 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

데이터 원본에 데이터 카탈로그 테이블 사용

Amazon S3와 커넥터를 제외한 모든 데이터 원본의 경우 선택한 원본 유형의 테이블이 AWS Glue Data Catalog에 있어야 합니다. AWS Glue는 데이터 카탈로그 테이블을 생성하지 않습니다.

데이터 카탈로그 테이블을 기반으로 데이터 원본 노드를 구성하려면

1. 새 작업 또는 저장된 작업의 시각적 편집기로 이동합니다.
2. 작업 다이어그램에서 데이터 원본 노드를 선택합니다.
3. [데이터 원본 속성(Data source properties)] 탭을 선택한 후 다음 정보를 입력합니다.
 - [S3 소스 유형(S3 source type)]: (Amazon S3 데이터 원본만 해당) 기존 AWS Glue Data Catalog 테이블을 사용하려면 [카탈로그 테이블 선택(Select a Catalog table)] 옵션을 선택합니다.
 - [데이터베이스(Database)]: 데이터 카탈로그에서 이 작업에 사용할 원본 테이블이 포함된 데이터베이스를 선택합니다. 검색 필드를 사용하여 이름으로 데이터베이스를 검색할 수 있습니다.
 - [테이블(Table)]: 목록에서 원본 데이터와 연결된 테이블을 선택합니다. 이 테이블이 AWS Glue Data Catalog에 이미 존재해야 합니다. 검색 필드를 사용하여 이름으로 테이블을 검색할 수 있습니다.
 - 파티션 조건자: (Amazon S3 데이터 원본만 해당) 분할 열만 포함하는 Spark SQL 기반 부울 표현식을 입력합니다. 예: "(year=='2020' and month=='04')"
 - [임시 디렉터리(Temporary directory)]: (Amazon Redshift 데이터 원본만 해당) ETL 작업에서 임시 중간 결과를 작성할 수 있는 Amazon S3 작업 디렉터리 위치에 대한 경로를 입력합니다.
 - [클러스터와 연결된 역할(Role associated with the cluster)]: (Amazon Redshift 데이터 원본만 해당) Amazon Redshift 클러스터에 대한 권한이 포함된 ETL 작업에 사용할 역할을 입력합니다. 자세한 내용은 [the section called “데이터 원본 및 데이터 대상 권한”](#) 단원을 참조하십시오.

데이터 원본에 커넥터 사용

[노드 유형(Node type)]에 대한 커넥터를 선택하는 경우 [사용자 정의 커넥터로 작업 작성](#)의 지침에 따라 데이터 원본 속성 구성을 완료합니다.

데이터 원본에 Amazon S3의 파일 사용

Amazon S3를 데이터 원본으로 선택한 경우 다음 중 하나를 선택할 수 있습니다.

- 데이터 카탈로그 데이터베이스 및 테이블.

- Amazon S3의 버킷, 폴더 또는 파일.

Amazon S3 버킷을 데이터 원본으로 사용하는 경우 AWS Glue는 파일 중 하나에서 또는 샘플 파일로 지정한 파일을 사용하여 지정된 위치에 있는 데이터의 스키마를 탐지합니다. [스키마 추론(Infer schema)] 버튼을 사용할 때 스키마 감지가 발생합니다. Amazon S3 위치 또는 샘플 파일을 변경하는 경우 [스키마 추론(Infer schema)]을 다시 선택하여 새 정보로 스키마 감지를 수행해야 합니다.

Amazon S3의 파일에서 직접 읽는 데이터 원본 노드를 구성하려면

1. 새 작업 또는 저장된 작업의 시각적 편집기로 이동합니다.
2. Amazon S3 소스에 대한 작업 다이어그램에서 데이터 원본 노드를 선택합니다.
3. [데이터 원본 속성(Data source properties)] 탭을 선택한 후 다음 정보를 입력합니다.
 - [S3 소스 유형(S3 source type)]: (Amazon S3 데이터 원본만 해당) [S3 위치(S3 location)] 옵션을 선택합니다.
 - [S3 URL]: 작업에 대한 데이터가 포함된 Amazon S3 버킷, 폴더 또는 파일의 경로를 입력합니다. [S3 찾아보기(Browse S3)]를 선택하여 계정에서 사용할 수 있는 위치에서 경로를 선택할 수 있습니다.
 - 재귀(Recursive): AWS Glue가 S3 위치에 있는 하위 폴더의 파일에서 데이터를 읽도록 하려면 이 옵션을 선택합니다.

하위 폴더에 분할된 데이터가 포함된 경우 AWS Glue는 폴더 이름에 지정된 파티션 정보를 데이터 카탈로그에 추가하지 않습니다. 예를 들어 Amazon S3의 다음 폴더를 고려합니다.

```
S3://sales/year=2019/month=Jan/day=1
S3://sales/year=2019/month=Jan/day=2
```

재귀(Recursive)를 선택하고 sales 폴더를 S3 위치로 선택하면 AWS Glue는 모든 하위 폴더의 데이터를 읽지만 연도, 월 또는 일의 파티션을 생성하지 않습니다.

- [데이터 포맷(Data format)]: 데이터가 저장되는 포맷을 선택합니다. JSON, CSV 또는 Parquet를 선택할 수 있습니다. 선택한 값은 소스 파일에서 데이터를 읽는 방법을 AWS Glue 작업에 알려줍니다.

Note

올바른 데이터 포맷을 선택하지 않을 경우 AWS Glue는 스키마를 올바르게 추론할 수 있지만 작업에서 소스 파일의 데이터를 올바르게 구문 분석할 수 없습니다.

선택한 포맷에 따라 추가 구성 옵션을 입력할 수 있습니다.

- [JSON](JavaScript Object Notation)

- [JsonPath]: 테이블 스키마 정의에 사용하는 객체를 가리키는 JSON 경로입니다. JSON 경로 표현식은 XPath 표현식이 XML 문서와 함께 사용되는 것과 같은 방식으로 항상 JSON 구조를 참조합니다. JSON 경로의 "루트 멤버 객체"는 객체 또는 배열인 경우에도 항상 \$라고 합니다. JSON 경로 작성 시 점이나 대괄호를 사용할 수 없습니다.

JSON 경로에 대한 자세한 내용은 GitHub 웹 사이트의 [JsonPath](#)를 참조하세요.

- [소스 파일의 레코드가 여러 줄에 걸쳐 있을 수 있음(Records in source files can span multiple lines)]: 단일 레코드가 CSV 파일의 여러 줄에 걸쳐 있을 수 있는 경우 이 옵션을 선택합니다.
- [CSV](쉼표로 분리된 값)
 - 구분 기호: 행의 각 열 항목을 구분하는 문자(예: ; 또는 ,)를 입력합니다.
 - [이스케이프 문자(Escape character)]: 이스케이프 문자로 사용되는 문자를 입력합니다. 이 문자는 이스케이프 문자 바로 뒤에 오는 문자를 문자 그대로 사용해야 하며 구분 기호로 해석되어서는 안 됨을 나타냅니다.
 - [인용 문자(Quote character)]: 개별 문자열을 단일 값으로 그룹화하는 데 사용되는 문자를 입력합니다. 예를 들어 CSV 파일에 "This is a single value"와 같은 값이 있는 경우 [큰따옴표(")](Double quote ("))를 선택합니다.
 - [소스 파일의 레코드가 여러 줄에 걸쳐 있을 수 있음(Records in source files can span multiple lines)]: 단일 레코드가 CSV 파일의 여러 줄에 걸쳐 있을 수 있는 경우 이 옵션을 선택합니다.
 - [소스 파일의 첫 번째 줄에 열 머리글 포함(First line of source file contains column headers)]: CSV 파일의 첫 번째 행에 데이터 대신 열 머리글이 포함된 경우 이 옵션을 선택합니다.
- [Parquet](Apache Parquet 컬럼 방식 스토리지)

Parquet 포맷으로 저장된 데이터에 대해 구성할 추가 설정이 없습니다.

- [분할 조건자(Partition predicate)]: 데이터 원본에서 읽은 데이터를 분할하려면 분할 열만 포함하는 Spark SQL 기반의 부울 식을 입력합니다. 예: "(year=='2020' and month=='04')"
- 고급 옵션(Advanced options): AWS Glue가 특정 파일을 기준으로 데이터 스키마를 탐지하도록하려면 이 섹션을 확장합니다.
- 스키마 추론(Schema inference): AWS Glue가 파일을 선택하도록 하는 대신 특정 파일을 사용하려면 S3에서 샘플 파일 선택(Choose a sample file from S3) 옵션을 선택합니다.
- [자동 샘플링된 파일(Auto-sampled file)]: 스키마를 추론하는 데 사용할 Amazon S3의 파일 경로를 입력합니다.

데이터 원본 노드를 편집하고 선택한 샘플 파일을 변경하는 경우 [스키마 다시 로드(Reload schema)]를 사용하여 새 샘플 파일로 스키마를 감지합니다.

4. [스키마 추론(Infer schema)] 버튼을 선택하여 Amazon S3의 소스 파일에서 스키마를 감지합니다. Amazon S3 위치 또는 샘플 파일을 변경하는 경우 [스키마 추론(Infer schema)]을 다시 선택하여 새 정보로 스키마를 추론해야 합니다.

스트리밍 데이터 원본 사용

지속적으로 실행되고 Amazon Kinesis Data Streams, Apache Kafka 및 Amazon Managed Streaming for Apache Kafka(Amazon MSK)의 스트리밍 소스의 데이터 사용하는 스트리밍 추출, 변환, 로드 작업을 생성할 수 있습니다.

스트리밍 데이터 원본의 속성을 구성하려면

1. 새 작업 또는 저장된 작업의 시각적 그래프 편집기로 이동합니다.
2. Kafka 또는 Kinesis Data Streams에 대한 그래프에서 데이터 원본 노드를 선택합니다.
3. [데이터 원본 속성(Data source properties)] 탭을 선택한 후 다음 정보를 입력합니다.

Kinesis

- Kinesis 소스 유형(Kinesis source type): 스트림 세부 정보(Stream details) 옵션을 선택하여 스트리밍 소스에 직접 액세스하거나 데이터 카탈로그 테이블(Data Catalog table)을 선택하여 테이블에 저장된 정보를 대신 사용합니다.

스트림 세부 정보(Stream details)를 선택한 경우 다음 추가 정보를 지정합니다.

- 데이터 스트림 위치: 스트림이 현재 사용자와 연결되어 있는지 또는 다른 사용자와 연결되어 있는지 여부를 선택합니다.

- 리전(Region): 스트림이 있는 AWS 리전을 선택합니다. 이 정보는 데이터 스트림에 액세스하기 위한 ARN을 구성하는 데 사용됩니다.
- 스트림 ARN(Stream ARN): Kinesis 데이터 스트림의 Amazon 리소스 이름(ARN)을 입력합니다. 스트림이 현재 계정 내에 있는 경우 드롭다운 목록에서 스트림 이름을 선택할 수 있습니다. 검색 필드를 사용하여 이름이나 ARN으로 데이터 스트림을 검색할 수 있습니다.
- 데이터 포맷(Data format): 목록에서 데이터 스트림에 사용할 포맷을 선택합니다.

AWS Glue는 스트리밍 데이터에서 스키마를 자동으로 탐지합니다.

데이터 카탈로그 테이블(Data Catalog table)을 선택한 경우 다음 추가 정보를 지정합니다.

- 데이터베이스(Database): (선택 사항) AWS Glue 데이터 카탈로그에서 스트리밍 데이터 원본과 연결된 테이블이 포함된 데이터베이스를 선택합니다. 검색 필드를 사용하여 이름으로 데이터베이스를 검색할 수 있습니다.
- [테이블(Table)]: (선택 사항) 목록에서 원본 데이터와 연결된 테이블을 선택합니다. 이 테이블이 AWS Glue 데이터 카탈로그에 이미 존재해야 합니다. 검색 필드를 사용하여 이름으로 테이블을 검색할 수 있습니다.
- 스키마 탐지(Detect schema): AWS Glue가 데이터 카탈로그 테이블의 스키마 정보를 사용하는 대신 스트리밍 데이터에서 스키마를 탐지하도록 하려면 이 옵션을 선택합니다. 스트림 세부 정보(Stream details) 옵션을 선택한 경우 이 옵션은 자동으로 사용됩니다.
- 시작 위치(Starting position): 기본적으로 ETL 작업은 가장 오래된 항목(Earliest) 옵션을 사용하므로 스트림에서 사용 가능한 가장 오래된 레코드부터 데이터를 읽습니다. 최신 항목(Latest)을 대신 선택할 수도 있습니다. 이 경우 ETL 작업은 스트림에서 가장 최근 레코드 직후부터 읽기를 시작해야 합니다.
- [기간 크기(Window size)]: 기본적으로 ETL 작업은 100초 기간에 데이터를 처리하고 작성합니다. 이를 통해 데이터를 효율적으로 처리할 수 있으며 예상보다 늦게 도착하는 데이터에 대해 집계를 수행할 수 있습니다. 이 기간 크기를 수정하여 적시성 또는 집계 정확도를 높일 수 있습니다.

AWS Glue 스트리밍 작업은 작업 북마크 대신 체크포인트를 사용하여 읽은 데이터를 추적합니다.

- 연결 옵션(Connection options): 키-값 페어를 추가하여 추가 연결 옵션을 지정하려면 이 섹션을 확장합니다. 여기에서 지정할 수 있는 옵션에 대한 자세한 내용은 AWS Glue 개발자 가이드의 ["connectionType": "kinesis"](#)를 참조하세요.

Kafka

- Apache Kafka 소스(Apache Kafka source): 스트림 세부 정보(Stream details) 옵션을 선택하여 스트리밍 소스에 직접 액세스하거나 데이터 카탈로그 테이블(Data Catalog table)을 선택하여 테이블에 저장된 정보를 대신 사용합니다.

데이터 카탈로그 테이블(Data Catalog table)을 선택한 경우 다음 추가 정보를 지정합니다.

- 데이터베이스(Database): (선택 사항) AWS Glue 데이터 카탈로그에서 스트리밍 데이터 원본과 연결된 테이블이 포함된 데이터베이스를 선택합니다. 검색 필드를 사용하여 이름으로 데이터베이스를 검색할 수 있습니다.
- [테이블(Table)]: (선택 사항) 목록에서 원본 데이터와 연결된 테이블을 선택합니다. 이 테이블이 AWS Glue 데이터 카탈로그에 이미 존재해야 합니다. 검색 필드를 사용하여 이름으로 테이블을 검색할 수 있습니다.
- 스키마 탐지(Detect schema): AWS Glue가 데이터 카탈로그 테이블에 스키마 정보를 저장하는 대신 스트리밍 데이터에서 스키마를 탐지하도록 하려면 이 옵션을 선택합니다. 스트림 세부 정보(Stream details) 옵션을 선택한 경우 이 옵션은 자동으로 사용됩니다.

스트림 세부 정보(Stream details)를 선택한 경우 다음 추가 정보를 지정합니다.

- 연결 이름(Connection name): Kafka 데이터 스트림에 대한 액세스 및 인증 정보가 포함된 AWS Glue 연결을 선택합니다. Kafka 스트리밍 데이터 원본과의 연결을 사용해야 합니다. 연결이 없는 경우 AWS Glue 콘솔을 사용하여 Kafka 데이터 스트림에 대한 연결을 생성할 수 있습니다.
- 주제 이름(Topic name): 읽을 주제의 이름을 입력합니다.
- 데이터 포맷(Data format): Kafka 이벤트 스트림에서 데이터를 읽을 때 사용할 포맷을 선택합니다.
- 시작 위치(Starting position): 기본적으로 ETL 작업은 가장 오래된 항목(Earliest) 옵션을 사용하므로 스트림에서 사용 가능한 가장 오래된 레코드부터 데이터를 읽습니다. 최신 항목(Latest)을 대신 선택할 수도 있습니다. 이 경우 ETL 작업은 스트림에서 가장 최근 레코드 직후부터 읽기를 시작해야 합니다.
- [기간 크기(Window size)]: 기본적으로 ETL 작업은 100초 기간에 데이터를 처리하고 작성합니다. 이를 통해 데이터를 효율적으로 처리할 수 있으며 예상보다 늦게 도착하는 데이터에 대해 집계를 수행할 수 있습니다. 이 기간 크기를 수정하여 적시성 또는 집계 정확도를 높일 수 있습니다.

AWS Glue 스트리밍 작업은 작업 북마크 대신 체크포인트를 사용하여 읽은 데이터를 추적합니다.

- 연결 옵션(Connection options): 키-값 페어를 추가하여 추가 연결 옵션을 지정하려면 이 섹션을 확장합니다. 여기에서 지정할 수 있는 옵션에 대한 자세한 내용은 AWS Glue 개발자 가이드의 "[connectionType": "kafka"](#)를 참조하세요.

Note

데이터 미리 보기는 현재 스트리밍 데이터 원본에 대해 지원되지 않습니다.

참조

모범 사례

- [Build an ETL service pipeline to load data incrementally from Amazon S3 to Amazon Redshift using AWS Glue](#)

ETL 프로그래밍

- [AWS Glue에서 ETL에 대한 관련 연결 유형 및 옵션](#)
- [JDBC connectionType 값](#)
- [Amazon Redshift 간 데이터 이동을 위한 고급 옵션](#)

자체 JDBC 드라이버를 사용하여 JDBC 연결 추가

JDBC 연결을 사용할 때 자체 JDBC 드라이버를 사용할 수 있습니다. AWS Glue 크롤러가 사용하는 기본 드라이버에서 데이터베이스에 연결할 수 없는 경우 자체 JDBC 드라이버를 사용할 수 있습니다. 예를 들어 Postgres 데이터베이스에서 SHA-256을 사용하려고 하지만 이전 Postgres 드라이버에서 이를 지원하지 않는 경우 자체 JDBC 드라이버를 사용할 수 있습니다.

지원되는 데이터 소스

지원되는 데이터 소스	지원되지 않는 데이터 소스
MySQL	Snowflake

지원되는 데이터 소스	지원되지 않는 데이터 소스
Postgres	
Oracle	
Redshift	
SQL Server	
Aurora*	

*기본 JDBC 드라이버를 사용하는 경우 지원됩니다. 일부 드라이버 기능은 사용할 수 없습니다.

JDBC 연결에 JDBC 드라이버 추가

Note

자체 JDBC 드라이버 버전을 가져오기로 선택한 경우 AWS Glue 크롤러는 AWS Glue작업 및 Amazon S3 버킷의 리소스를 소비하여 제공된 드라이버가 사용자 환경에서 실행되도록 합니다. 리소스의 추가 사용량은 계정에 반영됩니다. AWS Glue 크롤러 및 작업 비용은 청구에서 AWS Glue 카테고리에 속합니다. 또한 자체 JDBC 드라이버를 제공한다고 해서 크롤러가 해당 드라이버의 모든 기능을 사용할 수 있는 것은 아닙니다.

JDBC 연결에 자체 JDBC 드라이버를 추가하려면:

1. JDBC 드라이버 파일을 Amazon S3 위치에 추가합니다. 버킷 및/또는 폴더를 생성하거나 기존 버킷 및/또는 폴더를 사용할 수 있습니다.
2. AWS Glue 콘솔의 왼쪽 메뉴에서 데이터 카탈로그 아래 연결을 선택한 다음 새 연결을 생성합니다.
3. 연결 속성 필드를 작성하고 연결 유형으로 JDBC를 선택합니다.
4. 연결 액세스에서 JDBC URL 및 JDBC 드라이버 클래스 - 선택 사항을 입력합니다. 드라이버 클래스 이름은 AWS Glue 크롤러에서 지원하는 데이터 소스의 이름이어야 합니다.

Connection access

JDBC URL
Use the JDBC protocol to access Amazon Redshift, Amazon RDS, and publicly accessible databases.

JDBC syntax for most database engines is jdbc:protocol://host:port/databasename.

JDBC Driver Class name - optional

Type a custom JDBC driver class name for the crawler to connect to the data source.

JDBC Driver S3 Path - optional

Browse for or enter an existing S3 path to a .jar file.

Please note that if you choose to bring in your own JDBC driver versions to be used with Glue Crawlers, the Glue Crawlers will consume resources in Glue Jobs and S3 to ensure your provided driver are run in your environment. The additional usage of resources will be reflected in your account.

Credential type

Username and password
 Secret

Username

Password

- JDBC 드라이버 Amazon S3 경로 - 선택 사항 필드에서 JDBC 드라이버가 있는 Amazon S3 경로를 선택합니다.
- 사용자 이름과 암호 또는 보안 암호를 입력하는 경우 보안 인증 유형 필드를 작성합니다. 완료되면 연결 생성을 선택합니다.

Note

현재 테스트 연결은 지원되지 않습니다. 사용자가 제공한 JDBC 드라이버로 데이터 소스를 크롤링하는 경우 크롤러는 이 단계를 건너뛰습니다.

- 새로 생성된 연결을 크롤러에 추가합니다. AWS Glue 콘솔 왼쪽 메뉴에서 데이터 카탈로그 아래 크롤러를 선택한 다음 새 크롤러를 생성합니다.

8. 크롤러 추가 마법사의 2단계에서 데이터 소스 추가를 선택합니다.

Add data source ✕

Data source
Choose the source of data to be crawled.

JDBC ▼

Connection
Select a connection to access the data sources below.

mysql-connection068fd134-c2f1-4234-ad6b-345968e73be8 ▼ ↻

Clear selection

Add new connection ↗

Include path

public/%

You can substitute the percent (%) character for a schema or table. For databases that support schemas, enter MyDatabase/MySchema/% to match all tables in MySchema within MyDatabase. Oracle Database and MySQL don't support schema in the path; instead, enter MyDatabase/%. For Oracle database without SSL, MyDatabase can be either the system identifier (SID) or the service name (SERVICE_NAME). For Oracle database with SSL, MyDatabase must be the service name (SERVICE_NAME).

Additional metadata - optional

▼

Select additional metadata properties for the crawler to crawl.

Exclude tables matching pattern

Cancel

Add a JDBC data source

9. JDBC를 데이터 소스로 선택하고 이전 단계에서 생성한 연결을 선택합니다. 완료

10. 크롤러에서 자체 JDBC 드라이버를 사용하려면 AWS Glue 크롤러가 사용하는 역할에 다음 권한을 추가하십시오.

- CreateJob, DeleteJob, GetJob, GetJobRun, StartJobRun 작업에 대한 권한을 부여합니다.
- IAM 작업, iam:PassRole에 대한 권한 부여

- Amazon S3 작업, s3:DeleteObjects, s3:GetObject, s3:ListBucket, s3:PutObject에 대한 권한을 부여합니다.
- 서비스 보안 주체에 IAM 정책의 버킷 및 폴더에 대한 액세스 권한을 부여합니다.

IAM 정책 예제:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-name/driver-parent-folder/driver.jar",
        "arn:aws:s3:::bucket-name"
      ]
    }
  ]
}
```

11. VPC를 사용하는 경우 인터페이스 엔드포인트를 생성하여 AWS Glue 엔드포인트에 대한 액세스를 허용하고 이를 라우팅 테이블에 추가해야 합니다. 자세한 내용은 [AWS Glue에 대한 인터페이스 VPC 엔드포인트 생성](#)을 참조하세요.
12. 데이터 카탈로그에서 암호화를 사용하는 경우 AWS KMS 인터페이스 엔드포인트를 생성하여 라우팅 테이블에 추가하십시오. 자세한 내용은 [AWS KMS용 VPC 엔드포인트 생성](#)을 참조하세요.

AWS Glue 연결 테스트

가장 좋은 방법은 ETL 작업에서 AWS Glue 연결을 사용하기 전에 AWS Glue 콘솔을 사용하여 연결을 테스트하는 것입니다. AWS Glue에서는 연결할 때 파라미터를 사용하여 데이터 스토어에 액세스할 수

있는지 확인하고 오류를 보고합니다. AWS Glue 연결에 대한 자세한 내용은 [데이터에 연결](#) 단원을 참조하십시오.

AWS Glue 연결을 테스트하려면

1. 에 AWS Management Console 로그인하고 <https://console.aws.amazon.com/glue/> 에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 데이터 카탈로그에서 연결을 선택합니다. 탐색 창의 데이터 카탈로그 위의 데이터 연결을 선택할 수도 있습니다.
3. 연결에서 원하는 연결 옆의 확인란을 선택한 다음 작업을 선택합니다. 드롭다운 메뉴에서 연결 테스트를 선택합니다.
4. 연결 테스트 대화 상자에서 역할을 선택하거나 IAM 역할 생성을 선택하여 AWS Identity and Access Management (IAM) 콘솔로 이동하여 새 역할을 생성합니다. 역할에는 데이터 스토어에 대한 권한이 있어야 합니다.
5. 확인을 선택합니다.

테스트가 시작되고 완료하는 데 몇 분이 걸릴 수 있습니다. 테스트에 실패하는 경우 문제 해결을 선택하여 문제 해결 단계를 확인합니다.

6. 로그인을 보려면 [Logs] 를 선택합니다. CloudWatch 로그를 보려면 필요한 IAM 권한이 있어야 합니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의AWS 관리형 (사전 정의된) CloudWatch 로그 [정책](#)을 참조하십시오.

AWS 호출이 VPC를 통과하도록 구성

특수 작업 파라미터 `disable-proxy-v2`를 사용하면 VPC를 통해 호출을 Amazon S3, CloudWatch 및 AWS Glue와 같은 서비스로 라우팅할 수 있습니다. 기본적으로 AWS Glue는 로컬 프록시를 사용하여 AWS Glue VPC를 통해 Amazon S3의 스크립트와 라이브러리를 다운로드하고, 로그와 메트릭을 게시하기 위해 CloudWatch에 요청을 전송하고, 데이터 카탈로그에 액세스하기 위해 AWS Glue로 요청을 보냅니다. 이 프록시를 사용하면 VPC가 Amazon S3, CloudWatch 및 AWS Glue와 같은 기타 AWS 서비스에 대한 적절한 경로를 구성하지 않더라도 작업이 정상적으로 작동하도록 합니다. 이제 AWS Glue는 이 동작을 비활성화할 수 있는 파라미터를 제공합니다. 자세한 내용은 [AWS Glue에서 사용하는 작업 파라미터](#)를 참조하십시오. AWS Glue는 AWS Glue 작업의 CloudWatch 로그 게시에 로컬 프록시를 계속 사용합니다.

Note

- 이 기능은 AWS Glue 버전 2.0 이상을 사용하는 AWS Glue 작업에 대해 지원됩니다. 이 기능을 사용하는 경우 VPC가 NAT 또는 서비스 VPC 엔드포인트를 통해 Amazon S3에 대한 경로를 구성했는지 확인해야 합니다.
- 더 이상 사용되지 않는 작업 파라미터 `disable-proxy`는 VPC를 통해 스크립트와 라이브러리를 다운로드할 수 있도록 호출을 Amazon S3로 라우팅합니다. 새 파라미터 `disable-proxy-v2`를 대신 사용하는 것이 좋습니다.

사용 예

`disable-proxy-v2`를 사용하여 AWS Glue 작업 생성:

```
aws glue create-job \
  --name no-proxy-job \
  --role GlueDefaultRole \
  --command "Name=glueetl,ScriptLocation=s3://my-bucket/glue-script.py" \
  --connections Connections="traffic-monitored-connection" \
  --default-arguments '{"--disable-proxy-v2" : "true"}'
```

VPC의 JDBC 데이터 스토어로 연결

일반적으로 리소스는 퍼블릭 인터넷을 통해 액세스할 수 없도록 Amazon Virtual Private Cloud(Amazon VPC) 내에 생성합니다. 기본적으로 AWS Glue는 VPC 내의 리소스에 액세스할 수 없습니다. VPC 안의 리소스에 액세스하도록 AWS Glue를 설정하려면 VPC 서브넷 ID와 보안 그룹 ID가 포함된 VPC 구성 정보를 추가로 제공해야 합니다. AWS Glue는 이 정보를 사용하여 개인 VPC의 다른 리소스에 안전하게 연결하는 기능을 활성화하는 [탄력적 네트워크 인터페이스](#)를 설정합니다.

VPC 엔드포인트를 사용하는 경우 이를 라우팅 테이블에 추가합니다. 자세한 내용은 [AWS Glue에 대한 인터페이스 VPC 엔드포인트 생성 및 필수 조건](#) 섹션을 참조하세요.

데이터 카탈로그에서 암호화를 사용하는 경우 KMS 인터페이스 엔드포인트를 생성하고 이를 라우팅 테이블에 추가합니다. 자세한 내용은 [AWS KMS용 VPC 엔드포인트 생성](#)을 참조하세요.

탄력적 네트워크 인터페이스를 사용하여 VPC 데이터에 액세스

AWS Glue가 JDBC 데이터 스토어를 VPC에 연결하면 AWS Glue는 탄력적 네트워크 인터페이스를 (접두사 `Glue_`와 함께) 계정에 생성하여 VPC 데이터로 액세스할 수 있습니다. AWS Glue에 연결되

어 이 네트워크 인터페이스를 삭제할 수 없습니다. 탄력적 네트워크 인터페이스를 생성하는 일환으로 AWS Glue는 하나 이상의 보안 그룹을 인터페이스로 연결합니다. AWS Glue를 활성화하여 네트워크 인터페이스를 생성하기 위해서는 리소스와 연결된 보안 그룹은 소스 규칙으로 인바운드 액세스를 허용해야 합니다. 이 규칙은 리소스와 관련된 보안 그룹입니다. 따라서 탄력적 네트워크 인터페이스는 동일한 보안 그룹을 통해 데이터 스토어에 액세스할 수 있습니다.

AWS Glue가 구성 요소와 통신하려면 보안 그룹이 모든 TCP 포트에 자기 참조 인바운드 규칙을 지정해야 합니다. 자기 참조 규칙을 생성하여 VPC에서 동일한 보안 그룹으로 소스를 제한하고 외부로 공개되는 것을 예방합니다. VPC의 기본 보안 그룹은 ALL Traffic의 자기 참조 인바운드 액세스 규칙이 있습니다.

Amazon VPC 콘솔에 규칙을 생성할 수 있습니다. AWS Management Console을 통해 규칙 설정을 업데이트하려면 VPC 콘솔(<https://console.aws.amazon.com/vpc/>)을 탐색하고 적절한 보안 그룹을 선택합니다. ALL TCP의 인바운드 규칙을 지정하여 원본이 동일한 보안 그룹 이름을 갖도록 합니다. 보안 그룹 규칙에 대한 자세한 내용은 [VPC를 위한 보안 그룹](#)을 참조하십시오.

지정한 서브넷의 IP 주소 범위에 속하는 프라이빗 IP 주소가 이 탄력적 네트워크 인터페이스에 할당됩니다. 네트워크 인터페이스에 퍼블릭 IP 주소가 할당되지 않습니다. AWS Glue는 인터넷이 연결되어야 합니다(예: VPC 엔드포인트가 없는 AWS 서비스에 액세스할 때). 네트워크 주소 변환(NAT) 인스턴스를 VPC에 구성하거나 Amazon VPC NAT 게이트웨이를 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [NAT 게이트웨이](#)를 참조하세요. VPC에 연결된 인터넷 게이트웨이를 서브넷 라우팅 테이블에서 라우트로 직접 사용할 수 없습니다. 그 이유는 네트워크 인터페이스가 퍼블릭 IP 주소를 갖도록 요구하기 때문입니다.

VPC 네트워크 속성 enableDnsHostnames 및 enableDnsSupport는 true로 설정되어 있어야 합니다. 자세한 내용은 [VPC에서 DNS 사용하기](#) 단원을 참조하세요.

Important

인터넷 액세스가 없는 퍼블릭 서브넷 또는 프라이빗 서브넷에 데이터 스토어를 추가하지 마십시오. 대신 NAT 인스턴스 또는 Amazon VPC NAT 게이트웨이를 통해 인터넷 액세스가 가능한 프라이빗 서브넷에만 연결합니다.

탄력적 네트워크 인터페이스 속성

다음 속성을 제공하여 탄력적 네트워크 인터페이스를 생성합니다.

VPC

데이터 스토어를 포함하는 VPC 이름.

서브넷

데이터 스토어를 포함하는 VPC 내 서브넷.

보안 그룹

데이터 스토어와 연결된 보안 그룹입니다. AWS Glue은 VPC 서브넷과 연결된 탄력적 네트워크 인터페이스에 이러한 보안 그룹을 연결합니다. AWS Glue 구성 요소가 통신할 수 있도록 허용하면서도 다른 네트워크로부터의 액세스를 차단하려면 하나 이상의 선택된 보안 그룹이 모든 TCP 포트에 자체 참조 인바운드 규칙을 지정해야 합니다.

Amazon Redshift를 사용하여 VPC 관리에 대한 자세한 내용은 [Amazon Virtual Private Cloud\(VPC\)의 클러스터 관리](#)를 참조하세요.

Amazon Relational Database Service(Amazon RDS)로 VPC 관리에 대한 자세한 내용은 [VPC에서 Amazon RDS DB 인스턴스 작업](#)을 참조하세요.

MongoDB 또는 MongoDB Atlas 연결 사용

MongoDB 또는 MongoDB Atlas에 대한 연결을 생성한 후 ETL 작업에서 해당 연결을 사용할 수 있습니다. AWS Glue Data Catalog에 테이블을 생성하고 테이블의 connection 속성에 대해 MongoDB 또는 MongoDB Atlas 연결을 지정합니다.

AWS Glue는 연결 url과 자격 증명을 MongoDB 연결에 저장합니다. 연결 URI 형식은 다음과 같습니다.

- MongoDB의 경우: mongodb://host:port/database. 호스트는 호스트 이름, IP 주소 또는 UNIX 도메인 소켓일 수 있습니다. 연결 문자열이 포트를 지정하지 않는 경우 기본 MongoDB 포트인 27017을 사용합니다.
- MongoDB Atlas의 경우: mongodb+srv://server.example.com/database. 호스트는 DNS SRV 레코드에 해당하는 호스트 이름일 수 있습니다. SRV 형식에는 포트가 필요하지 않으며 기본 MongoDB 포트인 27017을 사용합니다.

또한 작업 스크립트에서 옵션을 지정할 수 있습니다. 자세한 내용은 [the section called “MongoDB 연결”](#) 단원을 참조하십시오.

VPC 엔드포인트를 사용한 Amazon S3 데이터 스토어 크롤링

Amazon S3 데이터 스토어 또는 Amazon S3 기반 데이터 카탈로그 테이블이 보안, 감사 또는 제어를 위해 Amazon Virtual Private Cloud 환경(Amazon VPC)에 의해서만 액세스되도록 구성할 수 있습니다. 이 주제에서는 Network 연결 유형을 사용하여 VPC 엔드포인트에서 Amazon S3 데이터 스토어 또는 Amazon S3 기반 데이터 카탈로그 테이블에 대한 연결을 생성하고 테스트하는 방법을 설명합니다.

데이터 스토어에서 크롤러를 실행하려면 다음 태스크를 수행합니다.

- [the section called “필수 조건”](#)
- [the section called “Amazon S3에 대한 연결 생성”](#)
- [the section called “Amazon S3에 대한 연결 테스트”](#)
- [the section called “Amazon S3 데이터 스토어의 크롤러 생성”](#)
- [the section called “크롤러 실행”](#)

필수 조건

Amazon S3 데이터 스토어 또는 Amazon S3 기반 데이터 카탈로그 테이블이 Amazon Virtual Private Cloud 환경(Amazon VPC)을 통해 액세스되도록 설정하기 위한 다음과 같은 필수 조건을 충족했는지 확인합니다.

- 구성된 VPC. 예: vpc-01685961063b0d84b. 자세한 내용은 Amazon VPC 사용 설명서의 [Amazon VPC 시작하기](#)를 참조하세요.
- VPC에 연결된 Amazon S3 엔드포인트. 예: vpc-01685961063b0d84b. 자세한 내용은 Amazon VPC 사용 설명서의 [Amazon S3용 엔드포인트 정책](#)을 참조하세요.

Name	VPC ID	State	IPv4 CIDR	IPv6	DHCP options set	Main Route table	Main Network ACL	Tenancy	Default VPC
privateVPC	vpc-01685961063b0d84b	available	192.168.1.0/24	-	dopt-a79e5acc	rtb-0750198567d5...	acl-02d197f2c9f46...	default	No

VPC: vpc-01685961063b0d84b

Description		CIDR Blocks		Flow Logs		Tags	
VPC ID	vpc-01685961063b0d84b	Tenancy	default				
State	available	Default VPC	No				
IPv4 CIDR	192.168.1.0/24	IPv6 CIDR	-				
IPv6 Pool	-	DNS resolution	Enabled				
Network ACL	acl-02d197f2c9f46be	DNS hostnames	Disabled				
DHCP options set	dopt-a79e5acc	Route table	rtb-0750198567d5b5202				
Owner	261353713322						

- VPC 엔드포인트를 가리키는 경로 항목. 예: VPC 엔드포인트(vpce-0ec5da4d265227786)에서 사용하는 라우팅 테이블의 vpce-0ec5da4d265227786.

Route Table ID: rtb-0750198567d5b5202

Summary Routes Subnet Associations Edge Associations Route Propagation Tags

Edit routes

View All routes

Destination	Target	Status	Propagate
192.168.1.0/24	local	active	No
pl-7ba54012 (com.amazonaws.us-east-2.s3, 52.219.80.0/20, 3.5.128.0/22, 3.5.132.0/23, 52.219.96.0/20, 52.92.76.0/22)	vpce-0ec5da4d265227786	active	No

- VPC에 연결된 네트워크 ACL은 트래픽을 허용합니다.
- VPC에 연결된 보안 그룹은 트래픽을 허용합니다.

Amazon S3에 대한 연결 생성

일반적으로 리소스는 퍼블릭 인터넷을 통해 액세스할 수 없도록 Amazon Virtual Private Cloud(Amazon VPC) 내에 생성합니다. 기본적으로 AWS Glue는 VPC 내의 리소스에 액세스할 수 없습니다. AWS Glue가 VPC 내부의 리소스에 액세스할 수 있도록 하려면 VPC 서브넷 ID 및 보안 그룹 ID를 포함하는 추가 VPC별 구성 정보를 제공해야 합니다. Network 연결을 생성하려면 다음 정보를 지정해야 합니다.

- VPC ID
- VPC 내의 서브넷
- 보안 그룹

Network 연결을 설정하려면

1. AWS Glue 콘솔의 탐색 창에서 [연결 추가(Add connection)]를 선택합니다.
2. 연결 이름을 입력하고 연결 유형으로 [네트워크(Network)]를 선택합니다. Next(다음)를 선택합니다.

Add connection



Connection properties

Connection access

Review all steps

Set up your connection's properties.

For more information, see [Working with Connections](#).

Connection name

Connection type

Description (optional)

3. VPC, 서브넷 및 보안 그룹 정보를 구성합니다.

- VPC: 데이터 스토어를 포함하는 VPC 이름을 선택합니다.
- 서브넷: VPC 내의 서브넷을 선택합니다.
- 보안 그룹: VPC의 데이터 스토어에 대한 액세스를 허용하는 하나 이상의 보안 그룹을 선택합니다.

Add connection
✕

- ✔ **Connection properties**
- TestNetworkConnecti
on
Type: Network
- **Connection access**
- Review all steps

Set up access to your data store.

For more information, see [Working with Connections](#).

VPC

Choose the VPC name that contains your data store.

vpc-01685961063b0d84b | privateVPC
▼

Subnet

Choose the subnet within your VPC.

subnet-0b350d86953aa6d60 | Range192
▼

Security groups

Choose one or more security groups that allow access to the data store in your VPC. AWS Glue associates these security groups to the ENI attached to your subnet. To allow AWS Glue components to communicate and also prevent access from other networks, at least one chosen security group must specify a self-referencing inbound rule for all TCP ports.

<input checked="" type="checkbox"/> Group ID	Group name
<input checked="" type="checkbox"/> sg-0ce8b36fb6206c56e	default

Back
Next

4. Next(다음)를 선택합니다.

5. 연결 정보를 확인하고 [마침(Finish)]을 선택합니다.

Add connection
✕

Connection properties

TestNetworkConnection
Type: Network

Connection access

VPC Id:
vpc-01685961063b0d84b

Review all steps

Connection properties

Name	TestNetworkConnection
Type	Network
Description (optional)	This is a demo Network Connection

Connection access

VPC Id	vpc-01685961063b0d84b
Subnet	subnet-0b350d86953aa6d60
Security groups	sg-0ce8b36fb6206c56e

Back
Finish

Amazon S3에 대한 연결 테스트

Network 연결을 생성한 후에는 VPC 엔드포인트에서 Amazon S3 데이터 스토어에 대한 연결을 테스트할 수 있습니다.

연결을 테스트할 때 다음 오류가 발생할 수 있습니다.

- [인터넷 연결 오류(INTERNET CONNECTION ERROR)]: 인터넷 연결 문제를 나타냅니다.
- [잘못된 버킷 오류(INVALID BUCKET ERROR)]: Amazon S3 버킷에 문제가 있음을 나타냅니다.
- [S3 연결 오류(S3 CONNECTION ERROR)]: Amazon S3 연결에 실패했음을 나타냅니다.
- [잘못된 연결 유형(INVALID CONNECTION TYPE)]: 연결 유형에 예상 값 NETWORK가 없음을 나타냅니다.
- [잘못된 연결 테스트 유형(INVALID CONNECTION TEST TYPE)]: 네트워크 연결 테스트의 유형에 문제가 있음을 나타냅니다.
- [잘못된 대상(INVALID TARGET)]: Amazon S3 버킷이 제대로 지정되지 않았음을 나타냅니다.

Network 연결을 테스트하려면

1. AWS Glue 콘솔에서 [네트워크(Network)] 연결을 선택합니다.
2. 연결 테스트를 선택합니다.

- 이전 단계에서 생성한 IAM 역할을 선택하고 Amazon S3 버킷을 지정합니다.
- [연결 테스트(Test connection)]를 선택하여 테스트를 시작합니다. 결과를 표시하는 데 몇 분 정도 걸릴 수 있습니다.

AWS Glue

Data catalog

Databases

Tables

Connections

Crawlers

Classifiers

Settings

ETL

Workflows

Jobs

ML Transforms

Triggers

Dev endpoints

Notebooks

Security

Security configurations

Test connection

Test connection from your VPC and subnet to data stores and Amazon S3.

IAM role ⓘ

AWSGlueServiceRole-glue

Ensure that this role has permission to access your data store.
[Create IAM role.](#)

Include path

s3://crawlertestfiles

S3

- athenaoutputprdept
- aws-glue-large-test-file
- aws-glue-scripts-261353713322-us-east-1
- aws-glue-temporary-261353713322-us-east-1
- cloudtrail-awslogs-261353713322-epvpwx6d-isengard-do-not-delete
- crawlertestfiles
- crawlertestfiles1
- dataforrunningcrawler
- do-not-delete-gatedgarden-audit-261353713322
- lf-kms-bucket
- lifecycleconfiguration
- mys3accesslogsprdept

Test connection

오류가 발생하면 다음을 확인합니다.

- 선택한 역할에 올바른 권한이 제공됩니다.
- 올바른 Amazon S3 버킷이 제공됩니다.
- 보안 그룹과 네트워크 ACL은 필요한 수신 및 발신 트래픽을 허용합니다.
- 지정한 VPC가 Amazon S3 VPC 엔드포인트에 연결되어 있습니다.

연결 테스트에 성공하면 크롤러를 생성할 수 있습니다.

Amazon S3 데이터 스토어의 크롤러 생성

이제 생성한 Network 연결을 지정하는 크롤러를 생성할 수 있습니다. 크롤러 생성에 대한 자세한 내용은 [크롤러 구성](#) 섹션을 참조하세요.

1. AWS Glue 콘솔의 탐색 창에서 [크롤러(Crawlers)]를 선택하여 시작합니다.
2. 크롤러 추가를 선택합니다.
3. 크롤러 이름을 지정하고 [다음(Next)]을 선택합니다.
4. 데이터 원본을 묻는 메시지가 표시되면 [S3]를 선택하고 Amazon S3 버킷 접두사와 이전에 생성한 연결을 지정합니다.

Add crawler [X]

Add a data store

Chosen data stores
S3: [X]

Crawler info
TestNetworkConnection

Crawler source type
Data stores

Data store
S3:

IAM Role

Schedule

Output

Review all steps

Choose a data store
S3 [v]

Connection
AddNetworkConnection [v]

Optionally include a Network connection to use with this S3 target. Note that each crawler is limited to one Network connection so any future S3 targets will also use the same connection (or none, if left blank).

[Add connection](#)

Crawl data in
 Specified path

Include path
s3://crawlerstestfiles [v]

All folders and files contained in the include path are crawled. For example, type s3://MyBucket/MyFolder/ to crawl all objects in MyFolder within MyBucket.

▶ Exclude patterns (optional)

[Back](#) [Next](#)

5. 필요한 경우 동일한 네트워크 연결에 다른 데이터 스토어를 추가합니다.
6. IAM 역할을 선택합니다. IAM 역할은 AWS Glue 서비스 및 Amazon S3 버킷에 대한 액세스를 허용해야 합니다. 자세한 내용은 [the section called “크롤러 구성”](#) 단원을 참조하십시오.

Add crawler

- ✔ **Crawler info**
TestNetworkConnecti
on
- ✔ **Crawler source type**
Data stores
- ✔ **Data store**
S3: s3://crawlertestf...
- **IAM Role**
- **Schedule**
- **Output**
- **Review all steps**

Choose an IAM role

The IAM role allows the crawler to run and access your Amazon S3 data stores. [Learn more](#)

- Update a policy in an IAM role
- Choose an existing IAM role
- Create an IAM role

IAM role ⓘ

AWSGlueServiceRole-glue



This role must provide permissions similar to the AWS managed policy, **AWSGlueServiceRole**, plus access to your data stores.

- s3://crawlertestfiles

You can also create an IAM role on the [IAM console](#).

Back

Next

7. 크롤러의 일정을 정의합니다.

8. 데이터 카탈로그에서 기존 데이터베이스를 선택하거나 새 데이터베이스 항목을 생성합니다.

Add crawler



- ✔ **Crawler info**
TestNetworkConnecti
on
- ✔ **Crawler source type**
Data stores
- ✔ **Data store**
S3: s3://crawlertestf...
- ✔ **IAM Role**
arn:aws:iam::2613537
13322:role/service-
role/AWSGlueService
Role-glue
- ✔ **Schedule**
Run on demand
- **Output**
- **Review all steps**

Configure the crawler's output

Database ⓘ

testnetworkconnectiondb

Add database

Prefix added to tables (optional) ⓘ

Type a prefix added to table names

- ▶ Grouping behavior for S3 data (optional)
- ▶ Configuration options (optional)

Back

Next

9. 나머지 설정을 완료합니다.

Amazon S3 기반 데이터 카탈로그 테이블의 크롤러 생성

이제 앞에서 생성한 Network 연결과 카탈로그 소스 유형을 지정하는 크롤러를 생성할 수 있습니다. 크롤러 생성에 대한 자세한 내용은 [크롤러 구성](#) 섹션을 참조하세요.

1. AWS Glue 콘솔의 탐색 창에서 [크롤러(Crawlers)]를 선택하여 시작합니다.
2. 크롤러 추가를 선택합니다.
3. 크롤러 이름을 지정하고 [다음(Next)]을 선택합니다.
4. 크롤러 소스 유형을 묻는 메시지가 표시되면 Existing catalog tables(기존 카탈로그 테이블)를 선택한 후 Available Tables(사용 가능한 테이블) 목록에서 크롤링할 기존 카탈로그 테이블을 지정합니다.

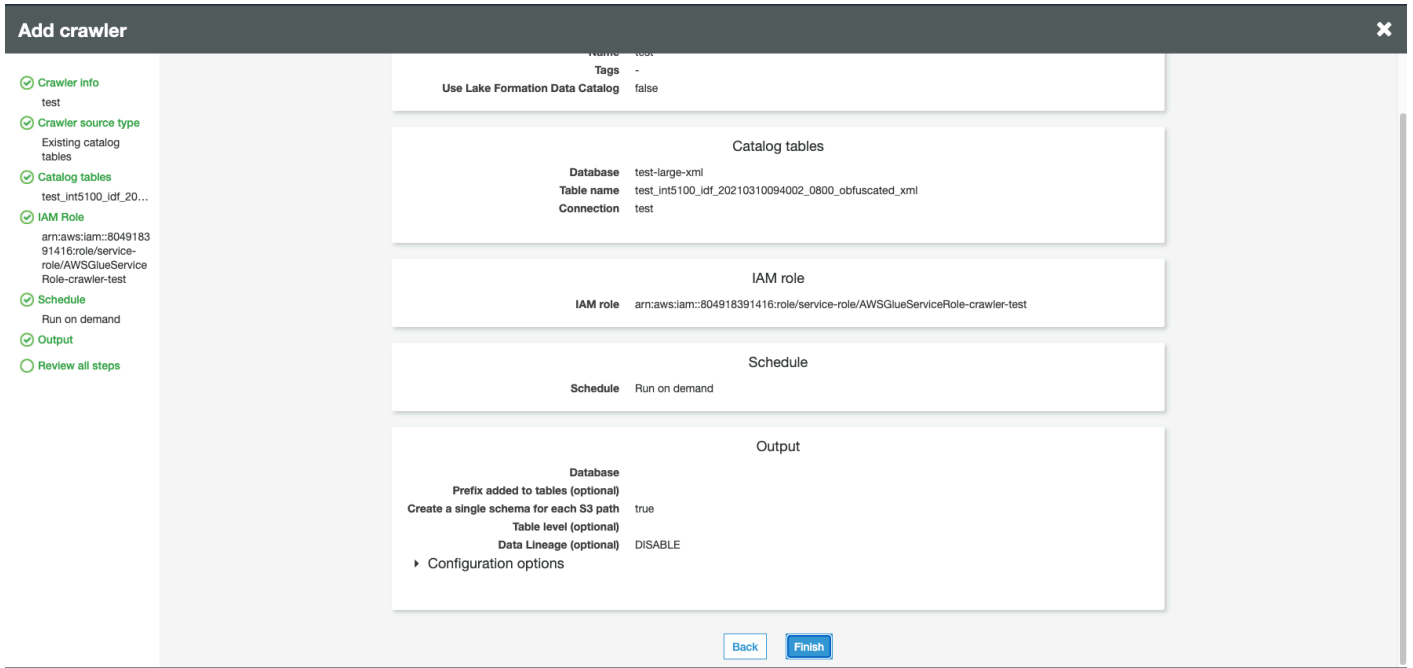
The screenshot shows the 'Add crawler' wizard in the AWS Glue console. The current step is 'Choose catalog tables'. On the left, there is a sidebar with navigation options: 'Crawler info' (selected), 'Crawler source type' (with sub-options 'Existing catalog tables' and 'Catalog tables'), 'IAM Role', 'Schedule', 'Output', and 'Review all steps'. The main content area is titled 'Choose catalog tables' and contains two tables. The 'Selected tables' table is empty, showing 'No items selected'. The 'Available tables' table has 3 rows, each with an 'Add' button and columns for Name, Database, Location, and Classification. Below the tables, there is a 'Connection' dropdown menu with the text 'Select a connection' and an 'Add connection' button.

Selected tables			
Name	Database	Location	Classification
No items selected			

Available tables			
Name	Database	Location	Classification
Add s3_event_crawl_demo	test-sampling-db	s3://s3-event-crawl-demo/	json
Add test_int5100_idf_20210310094002_0800_obfusca...	test-large-xml	s3://crawltickets/TEST_INT5100_IDF_20210310...	Unknown
Add test_int5100_idf_20210310094002_0800_obfusca...	test-cx-whitelist	s3://crawltickets/TEST_INT5100_IDF_20210310...	xml

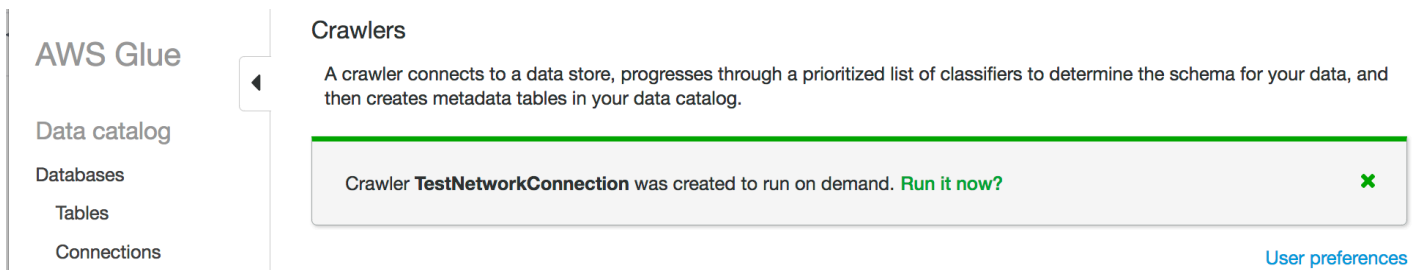
Connection: [Add connection](#)

5. IAM 역할을 선택합니다. IAM 역할은 AWS Glue 서비스 및 Amazon S3 버킷에 대한 액세스를 허용해야 합니다. 자세한 내용은 [the section called “크롤러 구성”](#) 단원을 참조하십시오.
6. 크롤러의 일정을 정의합니다.
7. 데이터 카탈로그에서 기존 데이터베이스를 선택하거나 새 데이터베이스 항목을 생성합니다.
8. 나머지 설정을 완료하고 단계를 검토합니다.



크롤러 실행

크롤러를 실행합니다.



문제 해결

VPC 게이트웨이를 사용하는 Amazon S3 버킷과 관련된 문제 해결은 [왜 게이트웨이 VPC 엔드포인트를 사용하여 S3 버킷에 연결할 수 없나요?](#)를 참조하세요.

AWS Glue 연결 문제 해결

AWS Glue 크롤러 혹은 작업을 데이터 스토어에 연결하기 위해 연결 속성을 사용할 경우, 연결을 시도할 때 오류가 발생할 수 있습니다. 탄력적 네트워크 인터페이스를 지정 Virtual Private Cloud(VPC)와 서브넷에 생성할 경우 AWS Glue는 서브넷의 프라이빗 IP 주소를 사용합니다. 연결에 지정된 보안 그룹은 탄력적 네트워크 인터페이스 각각에 적용됩니다. 보안 그룹이 발신 연결을 허용하는지 데이터베이스 클러스터 연결을 허용하는지 확인합니다.

또한 Apache Spark는 드라이버 및 관리자 노드의 양방향 연결을 요구합니다. 보안 그룹 중 하나는 모든 TCP 포트에서 수신을 허용해야 합니다. 자기 참조 보안 그룹을 통해 보안 그룹의 소스를 자체 내에서 제한하여 외부로 공개되는 것을 예방합니다.

다음은 연결 문제를 해결하기 위해 취해야 할 일반적인 행동입니다.

- 연결의 포트 주소를 확인합니다.
- 연결 또는 보안 암호에서 사용자 이름과 암호 문자열을 확인합니다.
- JDBC 데이터 스토어의 경우, 수신 연결을 허용하는지 확인합니다.
- VPC 내에서 스토어가 연결되는지 확인합니다.
- AWS Secrets Manager를 사용하여 연결 자격 증명을 저장하는 경우 AWS Glue에 대한 IAM 역할에 보안 암호에 액세스할 수 있는 권한이 있는지 확인합니다. 자세한 내용을 알아보려면 AWS Secrets Manager 사용 설명서의 [예: 보안 암호 값을 검색할 수 있는 권한](#)을 참조하세요. 네트워크 설정에 따라 VPC와 Secrets Manager 간의 프라이빗 연결을 설정하기 위해 VPC 엔드포인트를 생성해야 할 수도 있습니다. 자세한 내용을 알아보려면 [AWS Secrets Manager VPC 엔드포인트 사용](#)을 참조하세요.

자습서: AWS Glue Connector for Elasticsearch 사용

Elasticsearch는 로그 분석, 실시간 애플리케이션 모니터링, 클릭 스트림 분석 같은 사용 사례를 위한 인기 오픈 소스 검색 및 분석 엔진입니다. AWS Glue Studio에서 AWS Glue Connector for Elasticsearch를 구성하여 OpenSearch를 추출, 변환 및 로드(ETL) 작업의 데이터 저장소로 사용할 수 있습니다. 이 커넥터는 [AWS Marketplace](#)에서 무료로 사용할 수 있습니다.

Note

[AWS Marketplace Elasticsearch Spark 커넥터](#)는 더 이상 사용되지 않습니다. 대신 [AWS Glue Connector for Elasticsearch](#)를 사용하세요.

이 자습서에서는 최소한의 단계로 Amazon OpenSearch Service 노드에 연결하는 방법을 보여줍니다.

주제

- [필수 조건](#)
- [1단계: \(선택 사항\) OpenSearch 클러스터 정보에 대한 AWS 보안 암호 생성](#)
- [2단계: 커넥터 구독](#)

- [3단계: AWS Glue Studio에서 커넥터 활성화 및 연결 생성](#)
- [4단계: ETL 작업에 대한 IAM 역할 구성](#)
- [5단계: OpenSearch 연결을 사용하는 작업 생성](#)
- [6단계: 작업 실행](#)

필수 조건

이 튜토리얼을 사용하려면 다음이 필요합니다.

- AWS Glue Studio에 액세스
- AWS 클라우드의 OpenSearch 클러스터에 대한 액세스 권한
- (선택 사항) AWS Secrets Manager에 대한 액세스 권한.

1단계: (선택 사항) OpenSearch 클러스터 정보에 대한 AWS 보안 암호 생성

연결 자격 증명을 안전하게 보관하고 사용하려면 AWS Secrets Manager에 자격 증명을 저장합니다. 생성한 보안 암호는 튜토리얼의 뒷부분에서 연결에 사용됩니다. 자격 증명 키-값 페어는 일반 연결 옵션으로 AWS Glue Connector for Elasticsearch에 공급됩니다.

보안 암호 생성에 대한 자세한 내용은 AWS Secrets Manager User Guide의 [Creating and Managing Secrets with AWS Secrets Manager](#)를 참조하세요.

AWS 보안 암호를 생성하려면

1. [AWS Secrets Manager 콘솔](#)에 로그인합니다.
2. 서비스 소개 페이지 또는 비밀 목록 페이지에서 새 비밀 저장을 선택합니다.
3. 새 보안 암호 저장(Store a new secret) 페이지에서 다른 유형의 보안 암호(Other type of secret)를 선택합니다. 이 옵션은 보안 암호의 구조 및 세부 정보를 제공해야 함을 의미합니다.
4. OpenSearch 클러스터 사용자 이름에 대한 [키(Key)] 및 [값(Value)] 페어를 추가합니다. 예:


```
es.net.http.auth.user:username
```
5. [+ 행 추가(+ Add row)]를 선택하고 암호에 대해 다른 키-값 페어를 입력합니다. 예:


```
es.net.http.auth.pass: password
```
6. 다음을 선택합니다.
7. 보안 암호 이름을 입력합니다. 예: my-es-secret. 선택적으로 설명을 포함할 수 있습니다.

이 튜토리얼의 뒷부분에서 사용되는 보안 암호를 기록하고 [다음(Next)]을 선택합니다.

8. [다음(Next)]을 다시 선택하고 [저장(Store)]을 선택하여 보안 암호를 생성합니다.

다음 단계

[2단계: 커넥터 구독](#)

2단계: 커넥터 구독

AWS Glue Connector for Elasticsearch는 [AWS Marketplace](#)에서 무료로 제공됩니다.

AWS Marketplace에서 AWS Glue Connector for Elasticsearch를 구독하려면

1. License Manager를 사용하도록 AWS 계정을 아직 구성하지 않은 경우 다음을 수행합니다.
 - a. AWS License Manager 콘솔<https://console.aws.amazon.com/license-manager>을 엽니다.
 - b. [고객 관리형 라이선스 생성(Create customer managed license)]을 선택합니다.
 - c. IAM permissions(IAM 권한)(1회 설정) 창에서 I grant AWS License Manager the required permissions(필요한 권한 부여)를 선택한 다음 Grant permissions(권한 부여)를 선택합니다.

이 창이 표시되지 않으면 필요한 권한을 이미 구성한 것입니다.

2. <https://console.aws.amazon.com/gluestudio/>에서 AWS Glue Studio 콘솔을 엽니다.
3. AWS Glue Studio 콘솔에서 메뉴 아이콘
(☰)을 확장한 다음, 탐색 창에서 커넥터(Connectors)를 선택합니다.
4. Connectors(커넥터) 페이지에서 Go to AWS Marketplace(이동)을 선택합니다.
5. AWS Marketplace에서 Search AWS Glue Studio products(Glue Studio 제품 검색) 섹션의 검색 필드에 AWS Glue Connector for Elasticsearch를 입력하고 Enter 키를 누릅니다.
6. 커넥터 이름인 AWS Glue Connector for Elasticsearch를 선택합니다.
7. 커넥터의 제품 페이지에서 탭을 사용하여 커넥터에 대한 정보를 봅니다. 계속할 준비가 되면 [계속 구독(Continue to Subscribe)]을 선택합니다.
8. 소프트웨어의 사용 약관을 검토합니다. 약관 수락을 클릭하세요.
9. 구독 프로세스가 완료되면 다음과 같은 알림이 표시됩니다. "이 제품을 구독해 주셔서 감사합니다! 이제 소프트웨어를 구성할 수 있습니다." 배너 위에는 구성 계속 버튼이 있습니다. Continue to Configuration(구성 계속)을 선택합니다.

10. 소프트웨어 구성 페이지에서 이행(Fulfillment) 옵션을 선택합니다. AWS Glue 1.0/2.0 또는 AWS Glue 3.0 중에서 선택할 수 있습니다. 그런 뒤 [계속 시작(Continue to Launch)]을 선택합니다.

다음 단계

[3단계: AWS Glue Studio에서 커넥터 활성화 및 연결 생성](#)

3단계: AWS Glue Studio에서 커넥터 활성화 및 연결 생성

[계속 시작(Continue to Launch)]을 선택하면 AWS Marketplace에 [이 소프트웨어 시작(Launch this software)] 페이지가 표시됩니다. 링크를 사용하여 AWS Glue Studio에서 커넥터를 활성화한 후 연결을 생성합니다.

AWS Glue Studio에서 커넥터를 배포하고 연결을 생성하려면

1. AWS Marketplace 콘솔의 [이 소프트웨어 시작(Launch this software)] 페이지에서 [사용 지침(Usage Instructions)]을 선택한 다음 나타나는 창에서 링크를 선택합니다.
브라우저가 AWS Glue Studio 콘솔의 Marketplace 연결 생성(Create Marketplace connection) 페이지로 리디렉션됩니다.
2. 연결의 이름을 입력합니다. 예: my-es-connection.
3. [연결 액세스(Connection access)] 섹션에서 [연결 자격 증명 유형(Connection credential type)]에 대해 [사용자 이름 및 암호(User name and password)]를 선택합니다.
4. AWS secret(보안 암호)에 대해 보안 암호 이름을 입력합니다. 예: my-es-secret.
5. [네트워크 옵션(Network options)] 섹션에서 OpenSearch 클러스터에 연결할 VPC 정보를 입력합니다.
6. [연결 생성 및 커넥터 활성화(Create connection and activate connector)]를 선택합니다.

다음 단계

[4단계: ETL 작업에 대한 IAM 역할 구성](#)

4단계: ETL 작업에 대한 IAM 역할 구성

AWS Glue ETL 작업을 생성할 때 사용할 작업에 대한 AWS Identity and Access Management(IAM) 역할을 지정합니다. 역할은 Amazon S3(모든 소스, 대상, 스크립트, 드라이버 파일 및 임시 디렉터리용)와 AWS Glue Data Catalog 객체를 포함하여 작업에서 사용하는 모든 리소스에 대한 액세스 권한을 부여해야 합니다.

AWS Glue ETL 작업에 대해 수입된 IAM 역할은 이전 섹션에서 생성된 보안 암호에도 액세스할 수 있어야 합니다. 기본적으로 AWS 관리형 역할 `AWSGlueServiceRole`은 보안 암호에 액세스할 수 없습니다. 보안 암호에 대한 액세스 제어를 설정하려면 [AWS Secrets Manager에 대한 인증 및 액세스 제어와 특정 보안 암호에 대한 액세스 제한](#)을 참조하세요.

ETL 작업에 대한 IAM 역할을 구성하려면

1. [the section called “ETL 작업에 필요한 IAM 권한 검토”](#)에 설명된 권한을 구성합니다.
2. [the section called “커넥터 사용에 필요한 권한”](#)에 설명된 것처럼, AWS Glue Studio에서 커넥터를 사용할 때 필요한 추가 권한을 구성합니다.

다음 단계

[5단계: OpenSearch 연결을 사용하는 작업 생성](#)

5단계: OpenSearch 연결을 사용하는 작업 생성

ETL 작업에 대한 역할을 생성한 후 AWS Glue Studio에서 Open Spark Elasticsearch용 커넥터와 연결을 사용하는 작업을 생성할 수 있습니다.

작업이 Amazon Virtual Private Cloud(Amazon VPC) 내에서 실행되는 경우 VPC가 올바르게 구성되었는지 확인합니다. 자세한 내용은 [the section called “ETL 작업에 사용할 VPC 구성”](#) 단원을 참조하십시오.

Elasticsearch Spark 커넥터를 사용하는 작업을 생성하려면

1. AWS Glue Studio에서 커넥터(Connectors)를 선택합니다.
2. [연결(Your connections)] 목록에서 방금 생성한 연결을 선택하고 [작업 생성(Create job)]을 선택합니다.
3. 시각적 작업 편집기에서 데이터 원본 노드를 선택합니다. 오른쪽의 [데이터 원본 속성 - 커넥터 (Data source properties - Connector)] 탭에서 커넥터에 대한 추가 정보를 구성합니다.
 - a. [스키마 추가(Add schema)]를 선택하고 데이터 원본에 있는 데이터 집합의 스키마를 입력합니다. 연결은 데이터 카탈로그에 저장된 테이블을 사용하지 않습니다. 즉, AWS Glue Studio는 데이터의 스키마를 알지 못합니다. 이 스키마 정보를 수동으로 제공해야 합니다. 스키마 편집기를 사용하는 방법에 대한 지침은 [the section called “사용자 정의 변환 노드에서 스키마 편집”](#) 섹션을 참조하세요.
 - b. [연결 옵션(Connection options)]을 확장합니다.

- c. [새 옵션 추가(Add new option)]를 선택하고 AWS 보안 암호에 입력되지 않은 커넥터에 필요한 정보를 입력합니다.
- es.nodes: `https://<OpenSearch domain endpoint>`
 - es.port: 443
 - path: 테스트
 - es.nodes.wan.only.: true

이러한 연결 옵션에 대한 설명은 <https://www.elastic.co/guide/en/elasticsearch/hadoop/current/configuration.html> 섹션을 참조하세요.

4. 그래프에 대상 노드를 추가합니다.

데이터 대상은 Amazon S3이거나 AWS Glue Data Catalog 또는 커넥터의 정보를 사용하여 다른 위치에 데이터를 쓸 수 있습니다. 예를 들어 데이터 카탈로그 테이블을 사용하여 Amazon RDS의 데이터베이스에 쓰거나 커넥터를 데이터 대상으로 사용하여 AWS Glue에서 기본적으로 지원되지 않는 데이터 스토어에 쓸 수 있습니다.

데이터 대상에 대한 커넥터를 선택하는 경우 해당 커넥터에 대해 생성된 연결을 선택해야 합니다. 또한 커넥터 공급자가 요구하는 경우 커넥터에 추가 정보를 제공하는 옵션을 추가해야 합니다. AWS 보안 암호에 대한 정보가 포함된 연결을 사용하는 경우 연결 옵션에서 사용자 이름과 암호 인증을 제공할 필요가 없습니다.

5. 필요에 따라 [the section called “AWS Glue 관리형 데이터 변환 노드 편집”](#)에 설명된 대로 추가 데이터 원본과 하나 이상의 변환 노드를 추가합니다.
6. [the section called “작업 속성 수정”](#)에 설명된 대로 3단계부터 작업 속성을 구성하고 작업을 저장합니다.

다음 단계

[6단계: 작업 실행](#)

6단계: 작업 실행

작업을 저장한 후 작업을 실행하여 ETL 작업을 수행할 수 있습니다.

AWS Glue Connector for Elasticsearch에 대해 생성한 작업을 실행하려면

1. AWS Glue Studio 콘솔을 사용해 시각적 편집기 페이지에서 실행(Run)을 선택합니다.

2. 성공 배너에서 [실행 세부 정보(Run Details)]를 선택하거나 시각적 편집기의 [실행(Runs)] 탭을 선택하여 작업 실행에 대한 정보를 볼 수 있습니다.

대화형 세션을 사용하여 AWS Glue 작업 구축

데이터 엔지니어는 AWS Glue의 대화형 세션을 사용하기 전보다 더 쉽고 빠르게 AWS Glue 작업을 작성할 수 있습니다.

주제

- [AWS Glue 대화형 세션 개요](#)
- [AWS Glue 대화형 세션 시작하기](#)
- [Jupyter 및 AWS Glue Studio 노트북용 AWS Glue 대화형 세션 구성](#)
- [Ray 대화형 세션 시작하기 \(미리 보기\) AWS Glue](#)
- [IAM을 이용한 대화형 세션](#)
- [스크립트 또는 노트북을 AWS Glue 작업으로 변환](#)
- [스트리밍을 위한 AWS Glue 대화형 세션](#)
- [로컬로 AWS Glue 작업 스크립트 개발 및 테스트](#)
- [개발 엔드포인트](#)

AWS Glue 대화형 세션 개요

AWS Glue 대화형 세션을 통해 데이터 준비 및 분석 애플리케이션을 빠르게 구축, 테스트 및 실행할 수 있습니다. 대화형 세션은 데이터 준비를 위한 추출, 변환, 적재(ETL) 스크립트를 구축하고 테스트하기 위한 프로그래밍 방식의 시각적 인터페이스를 제공합니다. 대화형 세션은 Apache Spark 분석 애플리케이션을 실행하고 원격 Spark 런타임 환경에 대한 온디맨드 액세스를 제공합니다. AWS Glue는 이러한 대화형 세션에 대해 서버리스 Spark를 투명하게 관리합니다.

대화형 세션은 유연하기 때문에 선택한 환경에서 애플리케이션을 구축하고 테스트할 수 있습니다. AWS Command Line Interface 및 API를 통해 대화형 세션을 만들고 작업할 수 있습니다. Jupyter 호환 노트북을 사용하여 노트북 스크립트를 시각적으로 작성하고 테스트할 수 있습니다. 대화형 세션은 PyCharm, IntelliJ 및 VS Code와 같은 IDE와의 통합을 포함하여 Jupyter가 수행하는 거의 모든 곳에서 통합되는 오픈 소스 Jupyter 커널을 제공합니다. 이러한 특징 덕분에 로컬 환경에서 코드를 작성하고 대화형 세션 백엔드에서 원활하게 실행할 수 있습니다.

대화형 세션 API를 사용하여 고객은 Spark 인프라를 관리할 필요 없이 Apache Spark 분석을 사용하는 애플리케이션을 프로그래밍 방식으로 실행할 수 있습니다. 단일 대화형 세션 내에서 하나 이상의 Spark 문을 실행할 수 있습니다.

따라서 대화형 세션은 데이터 준비 및 분석 애플리케이션을 구축하고 실행하는 더 빠르고 경제적이며 유연한 방법을 제공합니다. 대화형 세션 사용 방법에 대한 자세한 내용은 이 섹션의 설명서를 참조하세요. [AWS Glue에서 지원하는 매직](#)

제한 사항

- 대화형 세션에서는 작업 북마크가 지원되지 않습니다.
- AWS Command Line Interface를 사용한 노트북 작업 생성은 지원되지 않습니다.

AWS Glue 대화형 세션 시작하기

이 섹션에서는 로컬에서 AWS Glue 대화형 세션을 실행하는 방법을 설명합니다.

대화형 세션을 로컬에서 설정하기 위한 사전 조건

다음은 대화형 세션을 설치하기 위한 사전 조건입니다.

- 지원되는 Python 버전은 3.6부터 3.10 이상입니다.
- MacOS/리눅스 및 Windows 지침은 아래 섹션을 참조하세요.

주피터 및 AWS Glue 인터랙티브 세션 Jupyter 커널 설치

다음을 사용하여 커널을 로컬에 설치합니다.

`install-glue-kernels` 명령은 pyspark 커널과 spark 커널 모두에 대한 jupyter kernelspec을 설치하고 올바른 디렉터리에 로고도 설치합니다.

```
pip3 install --upgrade jupyter boto3 aws-glue-sessions
```

```
install-glue-kernels
```

Jupyter 실행

Jupyter Notebook을 실행하려면 다음 단계를 완료합니다.

1. 다음 명령을 실행하여 Jupyter Notebook을 시작합니다.

jupyter notebook

2. 새로 생성(New)을 선택한 다음 AWS Glue 커널 중 하나를 선택하여 AWS Glue에 대한 코딩을 시작합니다.

세션 자격 증명 및 리전 구성

MacOS/Linux 지침

AWS Glue 대화형 세션에는 AWS Glue 작업 및 개발 엔드포인트와 동일한 IAM 권한이 필요합니다. 다음 두 가지 방법 중 하나로 대화형 세션에 사용되는 역할을 지정합니다.

1. `%iam_role` 및 `%region` 매직 사용
2. `~/.aws/config`에 추가 줄 사용

매직을 사용하여 세션 역할 구성

첫 번째 셀에서, 실행된 첫 번째 셀에 `%iam_role <YourGlueServiceRole>`을 입력합니다.

`~/.aws/config`를 사용하여 세션 역할 구성

AWS Glue대화형 세션의 서비스 역할은 노트북 자체에서 지정하거나 구성과 함께 저장할 수 있습니다. AWS CLI 일반적으로 AWS Glue 작업에서 사용하는 역할이 있다면 바로 그 역할일 것입니다. AWS Glue 작업에 사용하는 역할이 없는 경우 이 안내서의 [AWS Glue에 대한 IAM 권한 구성](#)에 따라 구성합니다.

이 역할을 대화형 세션의 기본 역할로 설정하려면 다음을 수행하세요.

1. 텍스트 편집기로 `~/.aws/config`를 엽니다.
2. AWS Glue에서 사용하는 프로파일을 찾습니다. 프로파일을 사용하지 않는 경우 [Default] 프로파일을 사용합니다.
3. 프로파일에서, 사용하려는 역할에 `glue_role_arn=<AWSGlueServiceRole>`과 같은 줄을 추가합니다.
4. [선택 사항]: 프로파일에 기본 리전 세트가 없는 경우, `region=us-east-1`을 추가하고 원하는 리전으로 `us-east-1`을 대체하는 것이 좋습니다.
5. 구성을 저장합니다.

자세한 정보는 [IAM을 이용한 대화형 세션](#)을 참조하세요.

Windows 지침

AWS Glue 대화형 세션에는 AWS Glue 작업 및 개발 엔드포인트와 동일한 IAM 권한이 필요합니다. 다음 두 가지 방법 중 하나로 대화형 세션에 사용되는 역할을 지정합니다.

1. %iam_role 및 %region 매직 사용
2. ~/.aws/config에 추가 줄 사용

매직을 사용하여 세션 역할 구성

첫 번째 셀에서, 실행된 첫 번째 셀에 %iam_role <YourGlueServiceRole>을 입력합니다.

~/.aws/config를 사용하여 세션 구성

AWS Glue대화형 세션의 서비스 역할은 노트북 자체에서 지정하거나 AWS CLI 구성과 함께 저장할 수 있습니다. 일반적으로 AWS Glue 작업에서 사용하는 역할이 있다면 바로 그 역할일 것입니다. AWS Glue 작업에 사용하는 역할이 없는 경우, [AWS Glue에 대한 IAM 권한 설정](#) 가이드를 따라 역할을 설정하세요.

이 역할을 대화형 세션의 기본 역할로 설정하려면 다음을 수행하세요.

1. 텍스트 편집기로 ~/.aws/config를 엽니다.
2. AWS Glue에서 사용하는 프로파일을 찾습니다. 프로파일을 사용하지 않는 경우 [Default] 프로파일을 사용합니다.
3. 프로파일에서, 사용하려는 역할에 glue_role_arn=<AWSGlueServiceRole>과 같은 줄을 추가합니다.
4. [선택 사항]: 프로파일에 기본 리전 세트가 없는 경우, region=us-east-1을 추가하고 원하는 리전으로 us-east-1을 대체하는 것이 좋습니다.
5. 구성을 저장합니다.

자세한 정보는 [IAM을 이용한 대화형 세션](#)을 참조하세요.

대화형 세션 평가판 업그레이드

커널은 버전 0.27과 함께 릴리스되었을 때 새 이름으로 업그레이드되었습니다. 미리 보기 버전의 커널을 정리하려면 터미널에서 다음을 실행하거나 실행하십시오. PowerShell

Note

사용자 지정 서비스 모델이 필요한 다른 AWS Glue 평가판의 일부인 경우 커널을 제거하면 사용자 지정 서비스 모델이 제거됩니다.

```
# Remove Old Glue Kernels
jupyter kernelspec remove glue_python_kernel
jupyter kernelspec remove glue_scala_kernel

# Remove Custom Model
cd ~/.aws/models
rm -rf glue/
```

SageMaker Studio에서 대화형 세션 사용

AWS Glue 대화형 세션은 데이터 과학자와 엔지니어가 데이터 준비 및 분석 애플리케이션을 빠르게 구축, 테스트 및 실행하는 데 사용할 수 있는 온디맨드 서버리스 Apache Spark 런타임 환경입니다. Amazon SageMaker Studio Classic 노트북을 시작하여 AWS Glue 대화형 세션을 시작할 수 있습니다.

자세한 내용은 [AWS Glue 대화형 세션을 사용하여 데이터 준비](#)를 참조하세요.

Microsoft Visual Studio Code에서 대화형 세션 사용

사전 조건

- AWS Glue 대화형 세션을 설치하고 Jupyter Notebook에서 작동하는지 확인합니다.
- Visual Studio Code with Jupyter를 다운로드하고 설치합니다. 자세한 내용은 [VS Code의 Jupyter Notebook](#)을 참조하세요.

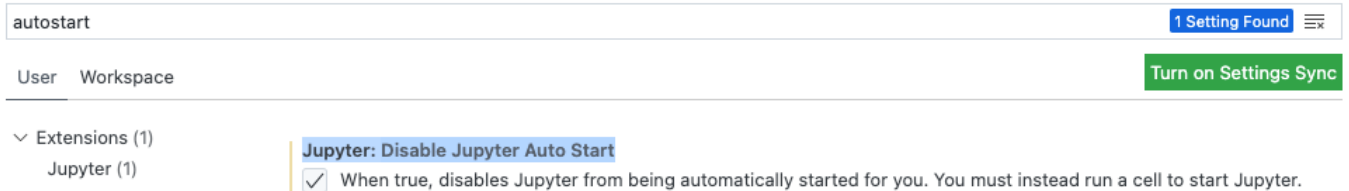
VSCode를 사용하여 대화형 세션을 시작하려면

1. VSCode에서 Jupyter 자동 시작을 비활성화합니다.

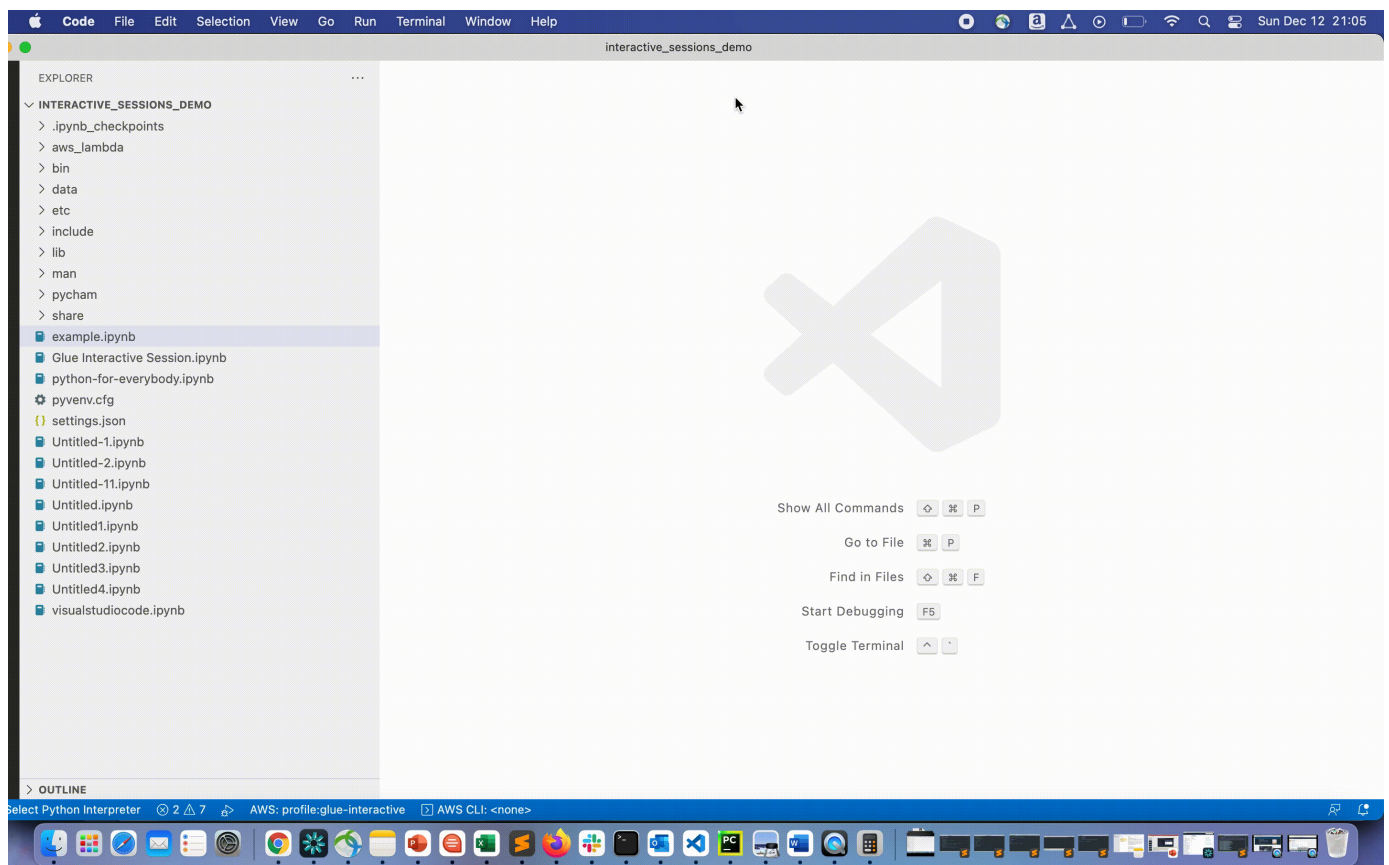
Visual Studio Code에서 Jupyter 커널이 자동으로 시작되어 세션이 이미 시작될 때 매직이 적용되지 않습니다. Windows에서 자동 시작을 비활성화하려면 파일 > 기본 설정 > 확장 프로그램 > Jupyter로 이동하고 Jupyter에서 마우스 오른쪽 버튼을 클릭한 다음 확장 프로그램 설정을 선택합니다.

MacOS의 경우 코드 > 설정 > 확장 프로그램 > Jupyter로 이동하고 Jupyter에서 마우스 오른쪽 버튼을 클릭한 다음 확장 프로그램 설정을 선택합니다.

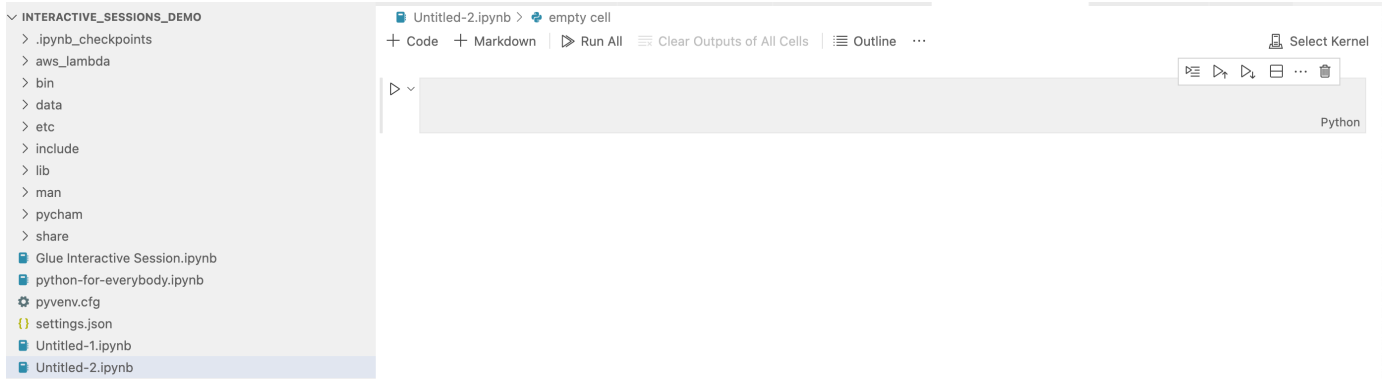
Jupyter: Jupyter 자동 시작 비활성화가 보일 때까지 아래로 스크롤합니다. 'True이면 Jupyter가 자동으로 시작되지 않도록 설정합니다. 대신 Jupyter를 시작하려면 셸을 실행해야 합니다.'("When true, disables Jupyter from being automatically started for you. You must instead run a cell to start Jupyter.")라는 레이블이 지정된 확인란을 선택합니다.



2. 파일(File) > 새 파일(New File) > 저장(Save)으로 이동하여 원하는 이름의 .ipynb 확장자로 저장하거나 언어 선택(Select a language)에서 Jupyter를 선택하여 파일을 저장합니다.



3. 파일을 두 번 클릭합니다. Jupyter 셸이 표시되고 노트북이 열립니다.



4. Windows에서 파일을 처음 만들 때 기본적으로 선택되어 있는 커널은 없습니다. 커널 선택(Select Kernel)을 클릭하면 사용 가능한 커널 목록이 표시됩니다. Glue PySpark를 선택합니다.

MacOS에서 Glue PySpark 커널이 보이지 않으면 다음 단계를 시도하세요.

1. 로컬 Jupyter 세션을 실행하여 URL을 확보합니다.

예를 들어 다음 명령을 실행하여 Jupyter Notebook을 시작합니다.

```
jupyter notebook
```

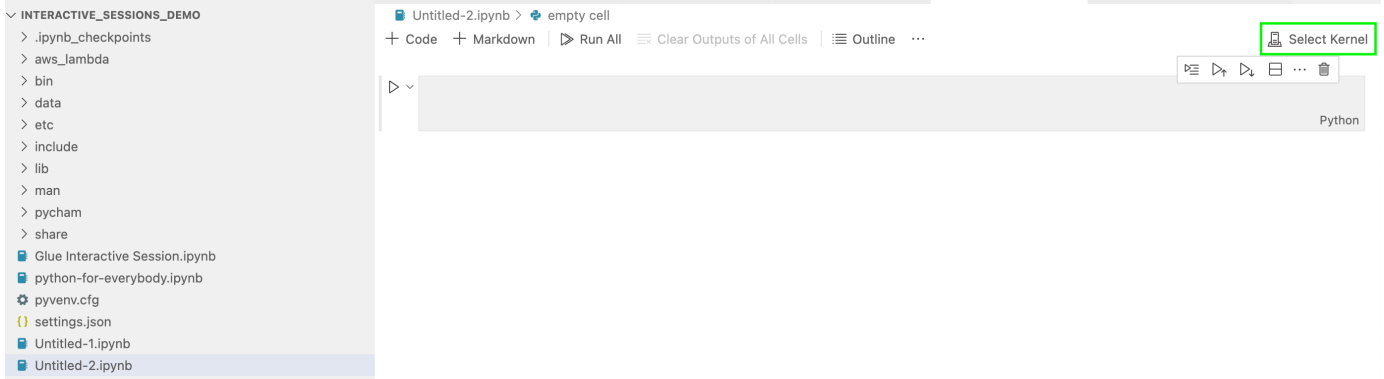
노트북을 처음 실행하면 `http://localhost:8888/?token=3398XXXXXXXXXXXXXXXXXX`와 같은 URL이 표시됩니다.

URL을 복사합니다.

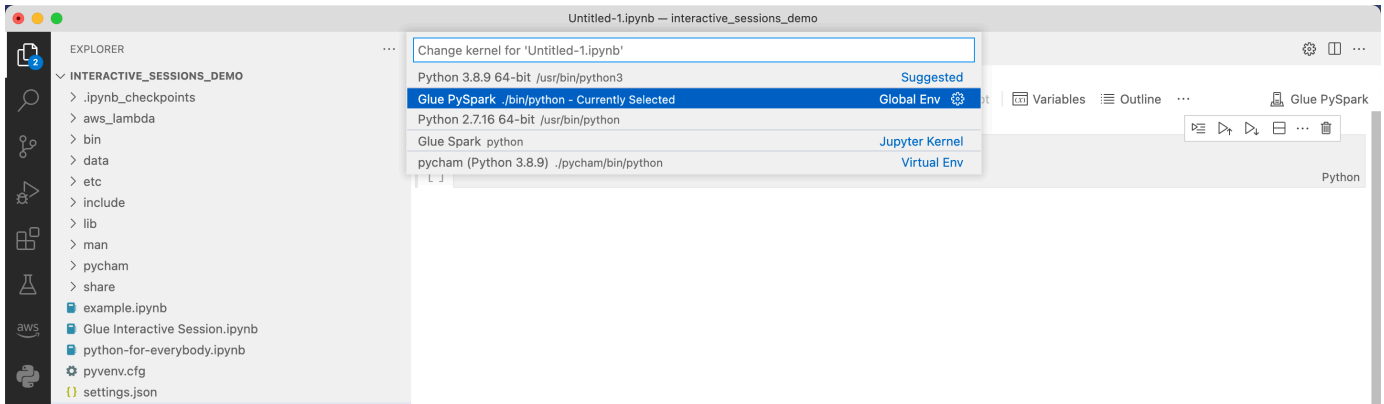
2. VS Code에서 현재 커널을 클릭한 다음 다른 커널 선택..., 기존 Jupyter 서버...를 차례로 선택합니다. 위 단계에서 복사한 URL을 붙여넣습니다.

오류 메시지가 표시된다면, [VS Code Jupyter wiki](#)를 참조하세요.

3. 성공하면 커널이 Glue PySpark로 설정됩니다.



Glue PySpark 또는 Glue Spark 커널(각각 Python 및 Scala용)을 선택합니다.



드롭다운 목록에 AWS Glue PySpark 및 AWS Glue Spark 커널이 없다면 전 단계에서 AWS Glue 커널을 설치했는지 또는 Visual Studio Code에서 `python.defaultInterpreterPath` 설정이 올바른지 확인하세요. 자세한 내용은 [python.defaultInterpreterPath 설정 설명](#)을 참조하세요.

5. AWS Glue 대화형 세션을 생성합니다. Jupyter Notebook에서와 동일한 방식으로 세션을 생성합니다. 첫 번째 셀 위에 매직을 지정하고 코드 문을 실행합니다.

Jupyter 및 AWS Glue Studio 노트북용 AWS Glue 대화형 세션 구성

Jupyter 매직 소개

Jupyter 매직은 셀의 시작 부분이나 셀 전체에 실행할 수 있는 명령입니다. %로 시작하는 매직은 라인 매직이고 %%로 시작하면 셀 매직입니다. 라인 매직(예:%region 및 %connections)은 하나의 셀 안에서 여러 매직으로 실행될 수 있으며, 또는 다음 예제와 같이 셀 본문에 포함된 코드로 실행할 수 있습니다.

```
%region us-east-2
%connections my_rds_connection
dy_f = glue_context.create_dynamic_frame.from_catalog(database='rds_tables',
table_name='sales_table')
```

셀 매직은 전체 셀을 사용해야 하며 명령이 여러 줄에 걸쳐 포함될 수 있습니다. %%sql의 예는 아래와 같습니다.

```
%%sql
select * from rds_tables.sales_table
```

Jupyter용 AWS Glue 대화형 세션에서 지원되는 매직

다음은 Jupyter Notebook용 AWS Glue 대화형 세션에서 사용할 수 있는 매직입니다.

세션 매직

명칭	유형	설명
%help	해당 사항 없음	모든 매직 명령에 대한 설명 및 입력 유형 목록을 반환합니다.
%profile	String	AWS 구성에서 자격 증명 공급자로 사용할 프로필을 지정합니다.
%region	String	세션을 초기화할 AWS 리전; 를 지정합니다. 기본값은 ~/.aws/config 에 있습니다. 예제: %region us-west-1
%idle_timeout	정수	셀이 실행된 후 세션이 시간 초과되기까지의 비활성 시간(분)입니다. Spark ETL 세션의 기본 유휴 제한 시간 값은 기본 제한 시간 값인 2,880분(48시간)입니다. 다른 세션 유형에 대해서는 해당 세션 유형에 대한 설명서를 참조하세요.

명칭	유형	설명
		예제: <code>%idle_timeout 3000</code>
<code>%session_id</code>	해당 사항 없음	실행 중인 세션의 세션 ID를 반환합니다.
<code>%session_id_prefix</code>	String	형식이 <code>[session_id_prefix]-[session_id]</code> 인 모든 세션 ID 앞에 오는 문자열을 정의합니다. 세션 ID가 제공되지 않으면 임의의 UUID가 생성됩니다. AWS Glue Studio에서 Jupyter Notebook을 실행할 때는 이 매직이 지원되지 않습니다. 예제: <code>%session_id_prefix 001</code>
<code>%status</code>		기간, 구성, 실행 사용자/역할을 포함한 현재 AWS Glue 세션의 상태를 반환합니다.
<code>%stop_session</code>		현재 세션을 중지합니다.
<code>%list_sessions</code>		현재 실행 중인 모든 세션을 이름과 ID로 나열합니다.
<code>%session_type</code>	String	세션 유형을 스트리밍, ETL 또는 Ray 중 하나로 설정합니다. 예제: <code>%session_type Streaming</code>
<code>%glue_version</code>	String	이 세션에서 사용할 AWS Glue 버전입니다. 예제: <code>%glue_version 3.0</code>

작업 유형 선택을 위한 매직

명칭	유형	설명
<code>%streaming</code>	String	세션 유형을 AWS Glue 스트리밍으로 변경합니다.

명칭	유형	설명
<code>%etl</code>	String	세션 유형을 AWS Glue ETL로 변경합니다.
<code>%glue_ray</code>	String	세션 유형을 AWS Glue for Ray로 변경합니다. AWS Glue Ray 대화형 세션에서 지원하는 Magics 를 참조하십시오.

AWS Glue for Sparta 구성 매직

`%%configure` 매직은 세션에 대한 모든 구성 파라미터로 구성된 JSON 형식의 사전입니다. 각 파라미터는 여기서 지정하거나 개별 매직을 통해 지정할 수 있습니다.

명칭	유형	설명
<code>%%configure</code>	딕셔너리	세션에 대한 모든 구성 파라미터로 구성된 JSON 포맷 딕셔너리를 지정합니다. 각 파라미터는 여기서 지정하거나 개별 매직을 통해 지정할 수 있습니다. <code>%%configure</code> 사용 방법에 관한 예제와 파라미터 목록은 아래 테이블, <code>%%configure</code> 사용을 참조하세요.
<code>%iam_role</code>	String	세션을 실행하는 데 사용할 IAM 역할 ARN을 지정합니다. 기본값은 <code>~/.aws/config</code> 에 있습니다. 예제: <code>%iam_role AWSGlueServiceRole</code>
<code>%number_of_workers</code>	정수	작업이 실행될 때 할당되는 정의된 <code>worker_type</code> 의 작업자 수입니다. <code>worker_type</code> 도 설정해야 합니다. <code>number_of_workers</code> 의 기본값은 5입니다. 예제: <code>%number_of_workers 2</code>

명칭	유형	설명
<code>%additional_python_modules</code>	나열	<p>클러스터에 포함할 추가 Python 모듈의 심플로 구분된 목록입니다(PyPI 또는 S3에서 가져올 수 있음).</p> <p>예: <code>%additional_python_modules pandas, numpy.</code></p>

명칭	유형	설명
%%tags	String	<p>세션에 태그를 추가합니다. 태그를 중괄호 {}로 묶어 지정합니다. 각 태그 이름 페어는 괄호(" ")로 묶고 쉼표(,)로 구분합니다.</p> <pre data-bbox="894 394 1507 594">%%tags {"billing":"Data-Platform", "team":"analytics"}</pre> <p>%%status 매직을 사용하여 세션과 관련된 태그를 확인합니다.</p> <pre data-bbox="894 751 1507 831">%%status</pre> <pre data-bbox="894 863 1507 1776">Session ID: <sessionId> Status: READY Role: <example-role> CreatedOn: 2023-05-26 11:12:17.056000-07:00 GlueVersion: 3.0 Job Type: glueetl Tags: {'owner':'example-owner', 'team':'analytics', 'billing':'Data-Platform'} Worker Type: G.4X Number of Workers: 5 Region: us-west-2 Applying the following default arguments: --glue_kernel_version 0.38.0 --enable-glue-datacatalog true Arguments Passed: ['--glue_kernel_version: 0.38.0', '--enable-glue-datacatalog: true']</pre>

명칭	유형	설명
%%assume_role	사전	<p>json 형식의 사전 또는 IAM 역할 ARN 문자열을 지정하여 크로스 계정 액세스를 위한 세션을 생성합니다.</p> <p>ARN 관련 예제:</p> <pre>%%assume_role { 'arn:aws:iam::XXXXXXXXXXXX: role/AWSGlueServiceRole' }</pre> <p>보안 인증 관련 예제:</p> <pre>%%assume_role {{ "aws_access_key_id" = "XXXXXXXXXXXX", "aws_secret_access_key" = "XXXXXXXXXXXX", "aws_session_token" = "XXXXXXXXXXXX" }}</pre>

%%configure 셀 매직 인수

%%configure 매직은 세션에 대한 모든 구성 파라미터로 구성된 JSON 형식의 사전입니다. 각 파라미터는 여기서 지정하거나 개별 매직을 통해 지정할 수 있습니다. %%configure 셀 매직이 지원하는 인수 예제는 아래를 참조하세요. 작업에 지정된 실행 인수에 -- 접두사를 사용합니다. 예제

```
%%configure
{
  "--user-jars-first": "true",
  "--enable-glue-datacatalog": "false"
}
```

작업 파라미터에 대한 자세한 내용은 [작업 파라미터](#) 섹션을 참조하세요.

세션 구성

파라미터	유형	설명
max_retries	정수	실패한 경우 이 작업을 다시 시도할 수 있는 최대 횟수입니다. <pre>%%configure { "max_retries": "0" }</pre>
max_concurrent_runs	정수	작업에 허용된 최대 동시 실행 수입니다. 예제 <pre>%%configure { "max_concurrent_runs": "3" }</pre>

세션 파라미터

파라미터	유형	설명
--enable-spark-ui	불	Spark UI를 활성화하여 AWS Glue ETL 작업을 모니터링하고 디버깅합니다. <pre>%%configure { "--enable-spark-ui": "true" }</pre>
--spark-event-logs-path	String	Amazon S3 경로를 지정합니다. Spark UI 모니터링 기능을 사용하는 경우.

파라미터	유형	설명
		<p>예제</p> <pre>%%configure { "--spark-event-logs-path": "s3://path/to/event/logs/" }</pre>
--script_location	String	<p>작업을 실행하는 스크립트의 S3 경로를 지정합니다.</p> <p>예제</p> <pre>%%configure { "script_location": "s3://new- folder-here" }</pre>
--SECURITY_CONFIGURATION	String	<p>AWS Glue 보안 구성의 이름</p> <p>예제</p> <pre>%%configure { "--SECURITY_CONFIGURATION": <i>security-configuration-name</i> , }</pre>

파라미터	유형	설명
<code>--job-language</code>	String	<p>스크립트 프로그래밍 언어. 'scala' 또는 'python'의 값이 허용됩니다. 기본값은 'python'입니다.</p> <p>예제</p> <pre>%%configure { "--job-language": "scala" }</pre>
<code>--class</code>	String	<p>Scala 스크립트 진입점으로써 Scala 클래스. 기본값은 null입니다.</p> <p>예제</p> <pre>%%configure { "--class": "className" }</pre>
<code>--user-jars-first</code>	불	<p>클래스 경로에서 고객의 추가 JAR 파일의 우선순위를 지정합니다. 기본값은 null입니다.</p> <p>예제</p> <pre>%%configure { "--user-jars-first": "true" }</pre>

파라미터	유형	설명
<code>--use-postgres-driver</code>	불	<p>JDBC 드라이버와의 충돌을 방지하기 위해 클래스 경로에서 Postgres JDBC 드라이버의 우선 순위를 지정합니다. Amazon Redshift 기본값은 null입니다.</p> <p>예제</p> <pre>%%configure { "--use-postgres-driver": "true" }</pre>
<code>--extra-files</code>	List(string)	<p>스크립트를 실행하기 전에 AWS Glue에서 스크립트의 작업 디렉터리에 복사하는 구성 파일과 같은 추가 파일에 대한 Amazon S3 경로입니다.</p> <p>예제</p> <pre>%%configure { "--extra-files": "s3://path/to/additional/files/" }</pre>

파라미터	유형	설명
<code>--job-bookmark-option</code>	String	<p>작업 북마크 동작을 제어합니다. ", '또는' 'job-bookmark-enable' 값을 허용합니다. job-bookmark-disable job-bookmark-pause 기본값은 job-bookmark-disable "입니다.</p> <p>예제</p> <pre>%%configure { "--job-bookmark-option": "job-bookmark-enable" }</pre>
<code>--TempDir</code>	String	<p>작업의 임시 디렉터리로 사용될 수 있는 버킷에 대한 Amazon S3 경로를 지정합니다. 기본값은 null입니다.</p> <p>예제</p> <pre>%%configure { "--TempDir": "s3://path/to/temp/dir" }</pre>

파라미터	유형	설명
<code>--enable-s3-parquet-optimized-committer</code>	불	<p>Amazon S3에 Parquet 데이터를 쓸 수 있도록 EMRFS Amazon S3 최적화 커미터를 활성화합니다. 기본값은 'true'입니다.</p> <p>예제</p> <pre>%%configure { "--enable-s3-parquet-optimized-committer": "false" }</pre>
<code>--enable-rename-algorithm-v2</code>	불	<p>EMRFS 이름 바꾸기 알고리즘 버전을 버전 2로 설정합니다. 기본값은 'true'입니다.</p> <p>예제</p> <pre>%%configure { "--enable-rename-algorithm-v2": "true" }</pre>
<code>--enable-glue-data-catalog</code>	불	<p>AWS Glue 데이터 카탈로그를 Apache Spark Hive 메타스토어로 사용할 수 있습니다.</p> <p>예제</p> <pre>%%configure { "--enable-glue-datacatalog": "true" }</pre>

파라미터	유형	설명
<code>--enable-metrics</code>	불	<p>작업 실행을 위해 작업 프로파일링용 지표 수집을 활성화합니다. 기본값은 'false'입니다.</p> <p>예제</p> <pre>%%configure { "--enable-metrics": "true" }</pre>
<code>--enable-continuous-cloudwatch-log</code>	불	<p>AWS Glue 작업에 대한 실시간 연속 로깅을 활성화합니다. 기본값은 'false'입니다.</p> <p>예제</p> <pre>%%configure { "--enable-continuous-cloudwatch-log": "true" }</pre>
<code>--enable-continuous-log-filter</code>	불	<p>연속 로깅을 위해 활성화된 작업을 편집하거나 해당 작업을 생성할 때 표준 필터 또는 필터 없음을 지정합니다. 기본값은 'true'입니다.</p> <p>예제</p> <pre>%%configure { "--enable-continuous-log-filter": "true" }</pre>

파라미터	유형	설명
<code>--continuous-log-stream-prefix</code>	String	<p>연속 로깅이 활성화된 작업의 사용자 지정 Amazon CloudWatch 로그 스트림 접두사를 지정합니다. 기본값은 null입니다.</p> <p>예제</p> <pre>%%configure { "--continuous-log-stream-prefix": "prefix" }</pre>
<code>--continuous-log-conversionPattern</code>	String	<p>연속 로깅에 대해 활성화된 작업의 사용자 지정 변환 로그 패턴을 지정합니다. 기본값은 null입니다.</p> <p>예제</p> <pre>%%configure { "--continuous-log-conversionPattern": "pattern" }</pre>

파라미터	유형	설명
--conf	String	<p>Spark 구성 파라미터를 제어합니다. 고급 사용 사례에 해당됩니다. 각 파라미터 앞에 --conf를 사용하세요. 예제</p> <pre> %%configure { "--conf": "spark.hadoop.hive .metastore.glue.catalogid=1 23456789012 --conf hive.meta store.client.factory.class= com.amazonaws.glue.catalog. metastore.AWSGlueDataCatalo gHiveClientFactory --conf hive.metastore.schema.verif ication=false" } </pre>

Spark 작업(ETL 및 스트리밍) 매직

명칭	유형	설명
%worker_type	String	<p>표준, G.1X 또는 G.2X가 있습니다. number_of_workers 도 설정해야 합니다. 기본 worker_type은 G.1X입니다.</p>
%connections	나열	<p>세션에서 사용할 연결을 심표로 구분된 목록으로 지정합니다.</p> <p>예제</p> <pre> %connections my_rds_connection dy_f = glue_context.create_dynamic _frame.from_catalog(databas </pre>

명칭	유형	설명
		<code>e='rds_tables', table_name='sales_table')</code>
<code>%extra_py_files</code>	나열	Amazon S3에 있는 추가 Python 파일의 심포로 구분된 목록입니다.
<code>%extra_jars</code>	나열	클러스터에 포함할 추가 jar의 심포로 구분된 목록입니다.
<code>%spark_conf</code>	String	세션의 사용자 지정 Spark 구성을 지정합니다. 예를 들어 <code>%spark_conf spark.serializer=org.apache.spark.serializer.KryoSerializer</code> 입니다.

Ray 작업을 위한 매직

명칭	유형	설명
<code>%min_workers</code>	정수	Ray 작업에 할당되는 작업자의 최소 수입니다. 기본값: 1. 예제: <code>%min_workers 2</code>
<code>%object_memory_head</code>	정수	웜 스타트 이후 인스턴스 헤드 노드에서 사용 가능한 메모리의 백분율입니다. 최소값: 0 최대값: 100 예제: <code>%object_memory_head 100</code>
<code>%object_memory_worker</code>	정수	웜 스타트 이후 인스턴스 워커 노드에서 사용 가능한 메모리의 백분율입니다. 최소값: 0 최대값: 100 예제: <code>%object_memory_worker 100</code>

작업 매직

명칭	유형	설명
<code>%%sql</code>	String	<p>SQL 코드를 실행합니다. 처음 <code>%%sql</code> 매직 이후의 모든 줄이 SQL 코드의 일부로 전달됩니다.</p> <p>예제: <code>%%sql select * from rds_tables.sales_table</code></p>
<code>%matplotlib</code>	Matplotlib 수치	<p>matplotlib 라이브러리를 사용하여 데이터를 시각화합니다.</p> <p>예제</p> <pre>import matplotlib.pyplot as plt # Set X-axis and Y-axis values x = [5, 2, 8, 4, 9] y = [10, 4, 8, 5, 2] # Create a bar chart plt.bar(x, y) # Show the plot %matplotlib plt</pre>
<code>%plotly</code>	Plotly 수치	<p>plotly 라이브러리를 사용하여 데이터를 시각화합니다.</p> <p>예제</p> <pre>import plotly.express as px #Create a graphical figure fig = px.line(x=["a","b","c"], y=[1,3,2], title="sample figure")</pre>

명칭	유형	설명
		#Show the figure %plotly fig

세션 이름 지정

AWS Glue대화형 세션은 AWS 리소스이므로 이름이 필요합니다. 이름은 각 세션마다 고유해야 하며 IAM 관리자가 제한할 수 있습니다. 자세한 정보는 [IAM을 이용한 대화형 세션](#)을 참조하세요. Jupyter 커널은 자동으로 고유한 세션 이름을 생성합니다. 그러나 세션의 이름은 두 가지 방법으로 직접 지정할 수 있습니다.

1. 에 있는 AWS Command Line Interface 구성 파일 사용 `~.aws/config` 를 [사용하여 AWS Config 설정](#)을 참조하십시오. AWS Command Line Interface
2. `%session_id_prefix` 매직을 사용합니다. [Jupyter용 AWS Glue 대화형 세션에서 지원되는 매직 섹션](#)을 참조하십시오.

세션 이름은 다음과 같이 생성됩니다.

- 접두사와 `session_id`가 제공된 경우 세션 이름은 {접두사}-{UUID}가 됩니다.
- 제공된 항목이 없을 경우 세션 이름은 {UUID}가 됩니다.

세션 이름을 접두어로 붙이면 AWS CLI 또는 콘솔에 세션을 나열할 때 세션을 인식할 수 있습니다.

대화형 세션에 대한 IAM 역할 지정

대화형 세션에서 실행하는 AWS Glue ETL 코드와 함께 사용할 AWS ID 및 액세스 관리 (IAM) 역할을 지정해야 합니다.

역할에는 AWS Glue 작업을 실행하는 데 필요한 IAM 권한과 동일한 IAM 권한이 필요합니다. AWS Glue 작업 및 대화형 세션을 위한 역할 생성에 대한 자세한 내용은 [AWS Glue에 대한 IAM 역할 생성](#)을 참조하세요.

IAM 역할은 두 가지 방법으로 지정할 수 있습니다.

- `~.aws/config`(권장) 에 있는 AWS Command Line Interface 구성 파일 사용. 자세한 내용은 [~.aws/config를 사용하여 세션 구성](#)을 참조하세요.

Note

%profile 매직이 사용될 때 해당 프로파일의 glue_iam_role에 대한 구성이 적용됩니다.

- %iam_role 매직을 사용합니다. 자세한 정보는 [Jupyter용 AWS Glue 대화형 세션에서 지원되는 매직](#)을 참조하세요.

명명된 프로파일을 사용하여 세션 구성

AWS Glue대화형 세션은 boto3와 동일한 자격 증명을 사용하며 대화형 세션은 (Linux 및 macOS) AWS Command Line Interface 또는 ~/.aws/config (Windows) 에 있는 것과 같은 이름이 지정된 프로파일을 사용하고 작동합니다. AWS CLI %USERPROFILE%\aws\config 자세한 내용은 [명명된 프로파일 사용](#)을 참조하세요.

대화형 세션은 AWS Glue 서비스 역할 및 세션 ID 접두사가 프로파일에서 지정될 수 있도록 하여 명명된 프로파일을 활용합니다. 프로파일 역할을 구성하려면 아래와 같이 iam_role 키 및/또는 session_id_prefix 에 대한 줄을 명명된 프로파일에 추가합니다. session_id_prefix에는 따옴표가 필요하지 않습니다. 예를 들어 session_id_prefix를 추가하려면 session_id_prefix=myprefix의 값을 입력합니다.

```
[default]
region=us-east-1
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
glue_iam_role=arn:aws:iam::<AccountID>:role/<GlueServiceRole>
session_id_prefix=<prefix_for_session_names>

[user1]
region=eu-west-1
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
glue_iam_role=arn:aws:iam::<AccountID>:role/<GlueServiceRoleUser1>
session_id_prefix=<prefix_for_session_names_for_user1>
```

자격 증명을 생성하는 사용자 지정 방법이 있는 경우, ~/.aws/config 파일에서 credential_process 파라미터를 사용하도록 프로파일을 구성할 수도 있습니다 예:

```
[profile developer]
```



```
region=us-east-1
credential_process = "/Users/Dave/generate_my_credentials.sh" --username helen
```

`credential_process` 파라미터를 통한 자격 증명 소싱에 대한 자세한 내용은 [외부 프로세스를 통해 자격 증명 소싱](#)을 참조하세요.

사용 중인 프로파일에 리전 또는 `iam_role`이 설정되어 있지 않으면 실행하는 첫 번째 셀에서 `%region` 및 `%iam_role` 매직을 사용하여 지정해야 합니다.

Ray 대화형 세션 시작하기 (미리 보기) AWS Glue

Warning

AWS Glue for Ray 인터랙티브 세션의 미리보기는 2024년 4월 30일에 종료되었습니다. 더 이상 AWS Glue for Ray에서 새 대화형 세션을 생성할 수 없습니다.

Note

AWS Glue for Ray는 미국 동부 (버지니아 북부), 미국 동부 (오하이오), 미국 서부 (오레곤), 아시아 태평양 (도쿄), 유럽 (아일랜드) 에서 사용할 수 있습니다.

콘솔의 Ray 인터랙티브 세션 AWS Glue Studio

AWS Glue Studio 콘솔의 작업 페이지에서 기존 Jupyter Notebook 옵션을 선택합니다. 그러면 Kernel(커널)을 선택할 수 있는 Notebook setup(노트북 설정) 페이지가 열립니다. Ray 커널을 선택하여 Ray 대화형 세션을 시작합니다. 대화형 세션에 대한 자세한 내용과 대화형 세션을 사용하는 방법은 [the section called “AWS Glue 대화형 세션 시작하기”](#) 섹션을 참조하세요.

Notebook setup [Info](#)

Initial configuration

Job name

Enter a name for the job. This name will be used for the script and the notebook file.

IAM Role

Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job.

Kernel

The kernel with which the notebook will be created.

Jupyter 커널을 사용하는 Ray 대화형 세션

AWS Glue Studio 콘솔 외부에서 Ray Kernel을 사용하려면 PyPI에 게시된 `aws-glue-sessions` 패키지를 설치해야 합니다. 커널 패키지 사용에 대한 자세한 내용은 [the section called “AWS Glue 대화형 세션 시작하기”](#) 설명서를 참조하세요.

커널을 업데이트하거나 설치하려면 `pip install --upgrade aws-glue-sessions`을 실행합니다. Ray 커널을 사용하려면 .37 이상 버전이 필요합니다.

Ray 대화형 세션에서는 Ray as Ray 작업과 동일한 라이브러리 및 버전에 액세스할 수 있습니다. 평가판에서 모든 Ray 대화형 세션은 Ray 2.4.0을 사용합니다.

Ray 대화형 세션 제한 시간 기본값

- 세션의 제한 시간 기본값은 8시간입니다.
- 유희 제한 시간 기본값은 1시간입니다.

AWS Glue Ray 대화형 세션에서 지원되는 매직

Ray 대화형 세션을 구동하는 AWS Glue Jupyter 커널의 매직은 Spark 세션의 매직과 비슷합니다. 자세한 내용은 [the section called “Jupyter 및 AWS Glue Studio 노트북용 AWS Glue 대화형 세션 구성”](#) 섹션을 참조하세요.

세션 매직

세션 매직은 AWS Glue for Ray 평가판 이전 버전과 거의 동일합니다. 이 평가판 이외의 세션 매직에 대한 자세한 내용은 [the section called “Jupyter용 AWS Glue 대화형 세션에서 지원되는 매직”](#) 섹션을 참조하세요. 세션 유형을 AWS Glue for Ray로 설정하는 새로운 매직을 소개합니다.

명칭	유형	설명
<code>%glue_ray</code>	String	세션 유형을 AWS Glue for Ray로 변경합니다.

AWS Glue 구성 마법

대화형 세션에서 AWS Glue를 구성하는 매직은 세션 유형에 따라 다를 수 있습니다. 현재는 AWS Glue for Ray를 사용할 때 기존 매직 중 이 하위 세트만 지원됩니다.

Warning

알려진 문제: **additional_python_modules**

Ray 대화형 세션(평가판)에서 `additional_python_modules` 매직을 사용하면 노트북을 저장할 때 문제가 발생할 수 있습니다. Ray 세션에 대한 Python 모듈을 구성하려면 `%%configure` 매직을 사용하여 [the section called “Ray 작업 파라미터”](#)에 정의된 `pip-install` 파라미터를 설정합니다.

명칭	유형	설명
<code>%%configure</code>	딕셔너리	세션에 대한 모든 구성 파라미터로 구성된 JSON 포맷 딕셔너리를 지정합니다. 각 파라미터는 여기서 지정하거나 개별 매직을 통해 지정할 수 있습니다.

명칭	유형	설명
<code>%iam_role</code>	String	세션을 실행하는 데 사용할 IAM 역할 ARN을 지정합니다. 기본값은 <code>~/.aws/configure</code> 에 있습니다.
<code>%number_of_workers</code>	int	작업이 실행될 때 할당되는 정의된 <code>worker_type</code> 의 작업자 수입니다. <code>worker_type</code> 도 설정해야 합니다.
<code>%worker_type</code>	String	AWS Glue for Ray(평가판)에서 지원되는 유일한 작업자 유형은 Z.2X입니다.
<code>%additional_python_modules</code>	나열	클러스터에 포함할 추가 Python 모듈의 심프로 구분된 목록입니다(Pypi 또는 S3에서 가져올 수 있음).

작업 매직

AWS Glue Ray 세션은 작업 매직을 지원하지 않습니다.

IAM을 이용한 대화형 세션

이 섹션에서는 AWS Glue 대화형 세션의 보안 고려 사항에 대해 설명합니다.

주제

- [대화형 세션에 사용되는 IAM 보안 주체](#)
- [클라이언트 보안 주체 설정](#)
- [런타임 역할 설정](#)
- [세션을 비공개로 설정하세요. TagOnCreate](#)
- [IAM 정책 고려 사항](#)

대화형 세션에 사용되는 IAM 보안 주체

AWS Glue 대화형 세션에 사용되는 두 가지 IAM 보안 주체를 사용합니다.

- 클라이언트 보안 주체: 클라이언트 보안 주체(사용자 또는 역할)는 보안 주체의 ID 기반 자격 증명으로 구성된 AWS Glue 클라이언트의 대화형 세션에 대한 API 작업에 권한을 부여합니다. 예를 들어, 이는 일반적으로 AWS Glue Console에 액세스하는 데 사용하는 IAM 역할일 수도 있습니다. 이는 자격 증명을 사용하는 IAM 사용자 또는 대화형 세션 Jupyter 커널에서 사용하는 AWS Glue 클라이언트에게 부여되는 역할일 수도 있습니다. AWS Command Line Interface
- 런타임 역할: 런타임 역할은 클라이언트 보안 주체가 대화형 세션 API 작업에 전달하는 IAM 역할입니다. AWS Glue는 이 역할을 사용하여 세션에서 문을 실행합니다. 예를 들어, 이 역할은 AWS Glue ETL 작업 실행에 사용되는 역할일 수 있습니다.

자세한 정보는 [런타임 역할 설정](#)을 참조하세요.

클라이언트 보안 주체 설정

대화형 세션 API를 호출할 수 있도록 하려면 클라이언트 보안 주체에 자격 증명 정책을 연결해야 합니다. 이 역할에는 CreateSession과 같은 대화형 세션 API로 전달할 실행 역할에 대한 iam:PassRole 액세스 권한이 있어야 합니다. 예를 들어 AWSGlueConsoleFullAccess관리형 정책을 IAM 역할에 연결하여 정책이 연결된 계정의 사용자가 계정에서 생성된 모든 세션 (예: 런타임 명령문 또는 취소 문)에 액세스할 수 있도록 할 수 있습니다.

세션을 보호하고 특정 IAM 역할 (예: 세션을 생성한 사용자와 관련된 역할)에만 비공개로 설정하려면 AWS Glue Interactive Session의 태그 기반 권한 부여 제어라는 것을 사용할 수 있습니다. TagOnCreate 자세한 내용은 소유자 태그 기반 범위 축소 관리형 정책을 사용하여 세션을 비공개로 설정하는 방법을 참조하십시오 [세션을 비공개로 설정하세요. TagOnCreate](#). TagOnCreate ID 기반 정책에 대한 자세한 내용은 [AWS Glue에 대한 ID 기반 정책](#)을 참조하세요.

런타임 역할 설정

대화형 세션에서 명령문을 수임하고 실행할 수 AWS Glue 있으려면 CreateSession API 작업에 IAM 역할을 전달해야 합니다. 역할에는 일반적인 AWS Glue 작업을 실행하는 데 필요한 IAM 권한과 동일한 IAM 권한이 있어야 합니다. 예를 들어 사용자 대신 서비스를 AWS Glue 호출하도록 허용하는 AWSGlueServiceRole정책을 사용하여 AWS 서비스 역할을 생성할 수 있습니다. AWS Glue 콘솔을 사용하는 경우 자동으로 사용자를 대신하여 서비스 역할을 생성하거나 기존 역할을 사용합니다. 고유한 IAM 역할을 생성하고 고유한 IAM 정책을 연결하여 유사한 권한을 허용할 수도 있습니다.

세션을 보호하고 세션을 만든 사용자에게만 비공개로 설정하려면 AWS Glue Interactive Session의 태그 기반 권한 부여 제어라는 것을 사용할 수 TagOnCreate 있습니다. 자세한 내용은 소유자 태그 기반 범위 축소 관리 정책을 사용하여 세션을 비공개로 설정하는 방법을 참조하십시오 [세션을 비공개로 설정하세요. TagOnCreate](#). TagOnCreate 자격 증명 기반 정책에 대한 자세한 내용은 [Glue에 대한 ID 기](#)

반 정책 AWS 섹션을 참조하세요. IAM 콘솔에서 직접 실행 역할을 생성하고 TagOnCreate 기능을 사용하여 서비스를 비공개로 설정하려는 경우 아래 단계를 따르세요.

1. 역할 유형이 Glue로 설정된 IAM 역할을 생성합니다.
2. 이 AWS Glue 관리형 정책을 첨부하세요. `AwsGlueSessionUserRestrictedServiceRole`
3. 역할 이름 앞에 정책 이름을 `AwsGlueSessionUserRestrictedServiceRole` 붙입니다. 예를 들어 이름이 `AwsGlueSessionUserRestrictedServiceRole-myrole`인 역할을 만들고 관리형 정책을 연결할 AWS Glue 수 있습니다. `AwsGlueSessionUserRestrictedServiceRole`
4. AWS Glue이(가) 역할을 수임할 수 있도록 다음과 같이 신뢰 정책을 연결합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "glue.amazonaws.com"
        ]
      },
      "Action": [
        "sts:AssumeRole"
      ]
    }
  ]
}
```

대화형 세션 Jupyter 커널의 경우 프로필에 `iam_role` 키를 지정할 수 있습니다. AWS Command Line Interface 자세한 내용은 [~/.aws/config를 사용하여 세션 구성](#)을 참조하세요. AWS Glue 노트북을 사용하여 대화형 세션과 상호 작용하는 경우 실행하는 첫 번째 셀에서 `%iam_role` 매직의 실행 역할을 전달할 수 있습니다.

세션을 비공개로 설정하세요. TagOnCreate

AWS Glue 대화형 세션은 대화형 세션에 대한 태그 지정 및 TBAC(태그 기반 권한 부여)를 명명된 리소스로 지원합니다. TBAC 사용 `TagResource` 및 `UntagResource` API 외에도 AWS Glue 대화형 세션은 작업을 통해 세션을 생성하는 동안에만 지정된 태그로 세션에 '태그'를 지정하는 `TagOnCreate` 기능을 지원합니다. `CreateSession` 이는 일명 해당 태그가 제거된다는 의미이기도 합니다. `DeleteSession` `UntagOnDelete`

TagOnCreate 세션을 만든 사람만 볼 수 있도록 하는 강력한 보안 메커니즘을 제공합니다. 예를 들어 요청 시 호출자의 UserID 값과 일치하는 "owner" RequestTag 태그가 UserID 태그로 제공된 경우에만 세션 생성을 허용하기 위해 클라이언트 보안 주체 (예: 사용자) 에 "owner"와 값이 \$ {aws:UserID} 인 IAM 정책을 연결할 수 있습니다. CreateSession 이 정책은 AWS Glue 대화형 세션이 세션 리소스를 생성하도록 하고 세션 생성 시간 동안에만 userId 태그가 있는 세션에 태그를 지정하도록 허용합니다. 또한 실행 중에 전달한 실행 역할에 "owner"가 ResourceTag 포함된 IAM 정책을 연결하여 세션 생성자 (예: \$ {aws:userId} 값을 가진 소유자 태그) 에게만 세션 액세스 범위를 좁힐 수 CreateSession 있습니다.

세션을 세션 생성자만 볼 수 있는 비공개로 설정하는 TagOnCreate 기능을 보다 쉽게 사용할 수 있도록 특수 관리형 정책 및 서비스 역할을 AWS Glue 제공합니다.

IAM AssumeRole 보안 주체를 사용하여 (즉, IAM 역할을 위임하여 제공되는 자격 증명을 사용하여) AWS Glue 대화형 세션을 생성하고 세션을 생성자만 볼 수 있도록 하려면 각각 및 와 유사한 정책을 사용하십시오.

AWSGlueSessionUserRestrictedNotebookPolicyAWSGlueSessionUserRestrictedNotebookServiceRole 이러한 정책을 통해 \$ {aws:PrincipalTag} AWS Glue 를 사용하여 소유자 태그 값을 추출할 수 있습니다. 이를 위해서는 역할 수입 자격 증명에서와 같이 값이 \$ {aws:UserID} 인 UserID 태그를 전달해야 합니다. SessionTag [ID 세션 태그](#)를 참조하세요. 자격 증명을 판매하는 인스턴스 프로필과 함께 Amazon EC2 인스턴스를 사용하고 있고 Amazon EC2 인스턴스 내에서 세션을 생성하거나 세션과 상호 작용하려는 경우 역할 수입 자격 증명과 같이 \$ {AWS:userId} 값을 가진 UserID 태그를 전달해야 합니다.

SessionTag

예를 들어, IAM AssumeRole 주체 자격 증명을 사용하여 세션을 생성하고 TagOnCreate 기능을 사용하여 서비스를 비공개로 설정하려는 경우 아래 단계를 따르세요.

1. IAM 콘솔에서 직접 런타임 역할을 생성합니다. 이 AWS Glue 관리형 정책을 AwsGlueSessionUserRestrictedNotebookServiceRole 연결하고 역할 이름 앞에 정책 이름을 붙이세요. AwsGlueSessionUserRestrictedNotebookServiceRole 예를 들어 이름이 AwsGlueSessionUserRestrictedNotebookServiceRole-myrole인 역할을 만들고 관리형 정책을 연결할 AWS Glue 수 있습니다. AwsGlueSessionUserRestrictedNotebookServiceRole
2. AWS Glue가 위의 역할을 수입할 수 있도록 허용하려면 아래와 같이 신뢰 정책을 연결하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```

    "Service": [
      "glue.amazonaws.com"
    ],
  },
  "Action": [
    "sts:AssumeRole"
  ]
}
]
}

```

3. 접두사를 `AwsGlueSessionUserRestrictedNotebookPolicy` 사용하여 이름을 지정한 다른 역할을 만들고 AWS Glue 관리형 정책을 추가하여 세션을 `AwsGlueSessionUserRestrictedNotebookPolicy` 비공개로 설정합니다. 관리형 정책 외에도 1단계에서 생성한 역할에 `PassRole iam:`를 허용하는 다음 인라인 정책을 추가하세요.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/AwsGlueSessionUserRestrictedNotebookServiceRole*"
      ],
      "Condition": {
        "StringLike": {
          "iam:PassedToService": [
            "glue.amazonaws.com"
          ]
        }
      }
    }
  ]
}

```

4. 다음과 같이 위의 IAM AWS Glue에 신뢰 정책을 연결하여 역할을 수임합니다.


```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": [
        "glue.amazonaws.com"
      ]
    },
    "Action": [
      "sts:AssumeRole",
      "sts:TagSession"
    ]
  }]
}
```

Note

선택적으로 단일 역할 (예: 노트북 역할) 을 사용하고 위의 관리형 정책 `AwsGlueSessionUserRestrictedNotebookServiceRole` 및 을 모두 연결할 수 있습니다. `AwsGlueSessionUserRestrictedNotebookPolicy` 또한 추가 인라인 정책을 연결하여 `iam:passrole`을 AWS Glue에 허용하세요. 마지막으로 위의 신뢰 정책을 연결하여 `sts:AssumeRole` 및 `sts:TagSession`을 허용하세요.

AWSGlueSessionUserRestrictedNotebookPolicy

는 태그 키 “소유자”와 값이 주체 (사용자 또는 역할) 의 AWS 사용자 ID와 일치하는 경우에만 노트북에서 AWS Glue 대화형 세션을 생성할 수 있는 액세스를 `AWSGlueSessionUserRestrictedNotebookPolicy` 제공합니다. 자세한 내용은 [정책 변수를 사용할 수 있는 경우](#)를 참조하세요. 이 정책은 AWS Glue Studio에서 AWS Glue 대화형 세션 노트북을 생성하는 보안 주체(사용자 또는 역할)에 연결됩니다. 또한 이 정책은 AWS Glue Studio 노트북에 대한 충분한 액세스를 허용하여 보안 주체의 AWS 사용자 ID와 일치하는 “소유자” 태그 값으로 생성된 AWS Glue Studio Interactive Session 리소스와 상호 작용할 수 있도록 합니다. 이 정책은 세션이 생성된 후 AWS Glue 세션 리소스에서 ‘owner’ 태그를 변경하거나 제거할 수 있는 권한을 거부합니다.

AWSGlueSessionUserRestrictedNotebookServiceRole

는 노트북에 대한 충분한 액세스를 AWSGlueSessionUserRestrictedNotebookServiceRole 제공하여 AWS Glue Studio 노트북 작성자의 주체 (사용자 또는 역할) 의 AWS 사용자 ID와 일치하는 “소유자” 태그 값으로 생성된 AWS Glue Interactive Session 리소스와 상호 작용할 수 있도록 합니다. 자세한 내용은 [정책 변수를 사용할 수 있는 경우](#)를 참조하세요. 이 서비스 역할 정책은 노트북에 마법처럼 전달되거나 API에 실행 역할로 전달되는 역할에 연결됩니다. CreateSession 또한 이 정책은 태그 키 “소유자”와 값이 보안 주체의 AWS 사용자 ID와 일치하는 경우에만 노트북에서 AWS Glue 대화형 세션을 생성할 수 있도록 허용합니다. 이 정책은 세션이 생성된 후 AWS Glue 세션 리소스에서 ‘owner’ 태그를 변경하거나 제거할 수 있는 권한을 거부합니다. 이 정책에는 Amazon S3 버킷 쓰기 및 읽기, CloudWatch 로그 작성, 에서 사용하는 Amazon EC2 리소스의 태그 생성 및 삭제에 대한 권한도 포함됩니다. AWS Glue

사용자 정책에서 세션을 프라이빗으로 설정

계정의 각 사용자에게 연결된 대상 IAM 역할을 연결하여 사용자가 자신의 `{aws:UserId}` 와 일치하는 값을 가진 소유자 태그만 사용하여 세션을 생성하지 못하도록 제한할 수 있습니다. AWSGlueSessionUserRestrictedPolicy AWSGlueSessionUserRestrictedNotebookPolicy 및 를 사용하는 대신 각각 및 와 AWSGlueSessionUserRestrictedNotebookServiceRole 유사한 정책을 사용해야 합니다. AWSGlueSessionUserRestrictedPolicy AWSGlueSessionUserRestrictedServiceRole 자세한 내용은 [자격 증명 기반 정책 사용](#)을 참조하세요. 이 정책은 자체 `{aws:userId}`를 포함한 소유자 태그로 세션을 생성한 사용자의 `{aws:userId}`인 생성자에게만 세션에 대한 액세스 범위를 좁힙니다. 위의 단계에 따라 IAM 콘솔을 사용하여 실행 역할을 직접 생성한 경우 AwsGlueSessionUserRestrictedPolicy 관리형 정책을 연결하는 것 외에도 계정의 각 사용자에게 다음 인라인 정책을 연결하여 이전에 만든 실행 역할을 iam:PassRole 허용하십시오. [런타임 역할 설정](#)

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": [
      "arn:aws:iam::*:role/AwsGlueSessionUserRestrictedServiceRole*"
    ],
    "Condition": {
      "StringLike": {
        "iam:PassedToService": [
          "glue.amazonaws.com"
        ]
      }
    }
  ]
}
```

```

    ]
  }
}
}]
}

```

AWSGlueSessionUserRestrictedPolicy

는 “owner”라는 태그 키와 AWS 사용자 ID와 일치하는 값이 제공된 경우에만 CreateSession API를 사용하여 AWS Glue 대화형 세션을 생성할 수 있는 액세스를 AWSGlueSessionUserRestrictedPolicy에 제공합니다. 이 ID 정책은 CreateSession API를 호출하는 사용자에게 연결됩니다. 또한 이 정책은 AWS 사용자 ID와 일치하는 “소유자” 태그 및 값으로 생성된 AWS Glue Interactive Session 리소스와 상호 작용할 수 있도록 허용합니다. 이 정책은 세션이 생성된 후 AWS Glue 세션 리소스에서 ‘owner’ 태그를 변경하거나 제거할 수 있는 권한을 거부합니다.

AWSGlueSessionUserRestrictedServiceRole

에서는 세션을 제외한 모든 AWS Glue 리소스에 대한 전체 액세스 권한을 AWSGlueSessionUserRestrictedServiceRole에 제공하며 사용자는 해당 사용자와 연결된 대화형 세션만 만들고 사용할 수 있습니다. 이 정책에는 다른 AWS 서비스의 Glue 리소스를 관리하는 AWS Glue 데 필요한 기타 권한도 포함됩니다. 이 정책에서는 다른 AWS 서비스의 AWS Glue 리소스에 태그를 추가할 수도 있습니다.

IAM 정책 고려 사항

대화형 세션은 AWS Glue의 IAM 리소스입니다. IAM 리소스이기 때문에 세션에 대한 액세스 및 상호 작용은 IAM 정책에 의해 관리됩니다. 클라이언트 보안 주체 또는 관리자가 구성한 실행 역할에 연결된 IAM 정책을 기반으로 클라이언트 보안 주체(사용자 또는 역할)는 새 세션을 생성하고 자신의 세션 및 다른 세션과 상호 작용할 수 있습니다.

관리자가 해당 계정의 모든 AWS Glue 리소스에 대한 액세스를 AWSGlueConsoleFullAccess 허용하거나 AWSGlueServiceRole 허용하는 IAM 정책을 추가한 경우 클라이언트 보안 주체는 서로 협업할 수 있습니다. 예를 들어 정책에서 허용하는 경우 한 사용자는 다른 사용자가 생성한 세션과 상호 작용할 수 있습니다.

특정 요구 사항에 맞게 정책을 구성하려면 [정책의 리소스 구성에 대한 IAM 설명서](#)를 참조하세요. 예를 들어, 사용자에게 속한 세션을 분리하려면 AWS Glue 대화형 세션에서 지원하는 TagOnCreate 기능을 사용할 수 있습니다. [세션을 비공개로 설정하세요. TagOnCreate](#) 섹션을 참조하십시오.

대화형 세션에서는 특정 VPC 조건에 따라 세션 생성을 제한할 수 있습니다. [조건 키를 사용하여 설정을 제어하는 정책 제어](#)를 참조하세요.

스크립트 또는 노트북을 AWS Glue 작업으로 변환

스크립트나 노트북을 AWS Glue 작업으로 변환할 수 있는 방법은 다음과 같이 두 가지가 있습니다.

- nbconvert를 사용하여 Jupyter .ipynb Notebook 문서 파일을 .py 파일로 변환합니다. 자세한 내용은 [nbconvert를 이용하여 노트북을 다른 형식으로 변환](#)을 참조하세요.
- AWS Glue Studio 노트북에 파일을 업로드합니다.
 - AWS Glue Studio 콘솔의 탐색 메뉴에서 작업(Jobs)을 선택합니다.
 - 작업 생성(Create job) 섹션에서 Jupyter Notebook을 선택합니다.
 - 옵션(Options) 섹션에서 기존 노트북 업로드 및 편집(Upload and edit an existing notebook)을 선택합니다.
 - 파일 선택(Choose file)을 선택하여 .ipynb 파일을 업로드합니다.

스트리밍을 위한 AWS Glue 대화형 세션

스트리밍 세션 유형 전환

AWS Glue 대화형 세션 구성 매직(%streaming)을 사용하여 실행 중인 작업을 정의하고 스트리밍 대화형 세션을 초기화합니다.

대화형 개발을 위한 샘플링 입력 스트림

인터랙티브 세션에서 인터랙티브 경험을 향상시키기 위해 개발한 도구 중 하나는 정적 스트림의 스냅샷을 얻을 수 있는 새로운 방법을 추가한 DynamicFrame 것입니다. AWS Glue GlueContext GlueContext워크플로를 검사, 상호 작용 및 구현할 수 있습니다.

GlueContext 클래스 인스턴스를 사용하면 getSampleStreamingDynamicFrame 메서드를 찾을 수 있습니다. 이 메서드의 필수 인수는 다음과 같습니다.

- dataFrame: 스파크 스트리밍 DataFrame
- options: 아래 사용 가능한 옵션 참조

사용 가능한 옵션은 다음과 같습니다.

- windowSize: 마이크로 배치 기간이라고도 합니다. 이 파라미터는 이전 배치가 트리거된 후 스트리밍 쿼리가 대기하는 시간을 결정합니다. 이 파라미터 값은 pollingTimeInMs보다 작아야 합니다.

- `pollingTimeInMs`: 메서드가 실행되는 총 시간입니다. 입력 스트림에서 샘플 레코드를 얻기 위해 적어도 하나의 마이크로 배치를 실행합니다.
- `recordPollingLimit`: 이 매개변수를 사용하면 스트림에서 폴링할 총 레코드 수를 제한할 수 있습니다.
- (선택 사항) `writeStreamFunction`을 사용하여 모든 레코드 샘플링 함수에 이 사용자 지정 함수를 적용할 수도 있습니다. Scala 및 Python 예제는 아래를 참조하세요.

Scala

```
val sampleBatchFunction = (batchDF: DataFrame, batchId: Long) => { //Optional but
  you can replace your own forEachBatch function here}
val jsonString: String = s""""{"pollingTimeInMs": "10000", "windowSize": "5
  seconds"}""""
val dynFrame = glueContext.getSampleStreamingDynamicFrame(YOUR_STREAMING_DF,
  JsonOptions(jsonString), sampleBatchFunction)
dynFrame.show()
```

Python

```
def sample_batch_function(batch_df, batch_id):
    //Optional but you can replace your own forEachBatch function here
    options = {
        "pollingTimeInMs": "10000",
        "windowSize": "5 seconds",
    }
    glue_context.getSampleStreamingDynamicFrame(YOUR_STREAMING_DF, options,
    sample_batch_function)
```

Note

몇 가지 이유로 인해 샘플링된 DynFrame이 비어 있는 경우가 발생할 수 있습니다.

- 스트리밍 소스가 '최신'으로 설정되어 있으며 샘플링 기간 동안 새 데이터가 수집되지 않았습니다.
- 폴링 시간이 충분하지 않아 수집된 레코드를 처리할 수 없습니다. 전체 배치가 처리되지 않으면 데이터가 표시되지 않습니다.

대화형 세션에서 스트리밍 애플리케이션 실행

AWS Glue 대화형 세션에서는 AWS Glue 콘솔에서 스트리밍 애플리케이션을 생성하는 것처럼 AWS Glue 스트리밍 애플리케이션을 실행할 수 있습니다. 대화형 세션은 세션 기반이므로 런타임에 예외가 발생해도 세션이 중지되지 않습니다. 이제 배치 함수를 반복적으로 개발할 수 있다는 추가 이점이 있습니다. 예:

```
def batch_function(data_frame, batch_id):
    log.info(data_frame.count())
    invalid_method_call()
glueContext.forEachBatch(frame=streaming_df, batch_function = batch_function, options =
{**})
```

위의 예에서는 잘못된 메서드 사용을 포함했고, 전체 애플리케이션을 종료하는 일반 AWS Glue 작업과는 달리 사용자의 코딩 컨텍스트 및 정의가 완전히 보존되며 세션이 여전히 작동 중입니다. 새 클러스터를 부트스트랩하고 모든 이전 변환을 다시 실행할 필요가 없습니다. 이를 통해 배치 함수 구현을 신속하게 반복하여 바람직한 결과를 얻을 수 있습니다.

대화형 세션은 세션이 한 번에 하나의 문만 실행하도록 각 문을 차단 방식으로 평가한다는 점에 유의해야 합니다. 스트리밍 쿼리는 지속적이고 끝나지 않으므로 활성 스트리밍 쿼리가 포함된 세션은 중단되지 않는 한 어떤 후속 문도 처리할 수 없습니다. Jupyter Notebook에서 직접 중단 명령을 실행할 수 있으며 커널이 취소를 처리할 것입니다.

실행 대기 중인 다음 일련의 문을 예로 들어 보겠습니다.

```
Statement 1:
    val number = df.count()
    #Spark Action with deterministic result
    Result: 5

Statement 2:
    streamingQuery.start().awaitTermination()
    #Spark Streaming Query that will be executing continuously
    Result: Constantly updated with each microbatch

Statement 3:
    val number2 = df.count()
    #This will not be executed as previous statement will be running indefinitely
```

로컬로 AWS Glue 작업 스크립트 개발 및 테스트

Spark용 AWS Glue 작업 스크립트를 개발하고 테스트할 때 사용할 수 있는 옵션은 여러 가지가 있습니다.

- AWS Glue Studio 콘솔
 - Visual editor(시각적 편집기)
 - 스크립트 에디터
 - AWS Glue Studio 노트북
- 대화형 세션
 - Jupyter Notebook
- 도커 이미지
 - 로컬 개발
 - 원격 개발
- AWS Glue Studio ETL 라이브러리
 - 로컬 개발

요구 사항에 따라 위의 옵션 중 하나를 선택할 수 있습니다.

코드가 없거나 코드 경험이 적은 경우 AWS Glue Studio 시각적 편집기를 선택하는 것이 좋습니다.

대화형 노트북 환경을 선호하는 경우 AWS Glue Studio 노트북을 선택하는 것이 좋습니다. 자세한 내용은 [AWS Glue Studio 및 AWS Glue을\(를\) 사용하여 노트북 사용](#)을 참조하세요. 로컬 환경을 직접 사용하려는 경우 대화형 세션을 선택하는 것이 좋습니다. 자세한 내용은 [AWS Glue를 사용하여 대화형 세션 사용](#)을 참조하세요.

로컬/원격 개발 환경을 선호하는 경우 Docker 이미지를 사용하는 것이 좋습니다. 이를 통해 AWS Glue 비용을 들이지 않고도 원하는 곳에서 AWS Glue for Spark 작업 스크립트를 개발하고 테스트할 수 있습니다.

Docker를 사용하지 않고 로컬 개발을 선호하는 경우 AWS Glue ETL 라이브러리 디렉토리를 사용하는 것이 좋습니다.

AWS Glue Studio를 사용하여 개발

AWS Glue Studio 시각적 편집기는 AWS Glue에서 추출, 변환, 로드 작업을 쉽게 생성, 실행, 모니터링할 수 있게 해주는 그래픽 인터페이스입니다. 데이터 변환 워크플로를 시각적으로 구성하고 AWS Glue

의 Apache Spark 기반 서버리스 ETL 엔진에서 원활하게 실행할 수 있습니다. 작업의 각 단계에서 스키마 및 데이터 결과를 검사할 수 있습니다. 자세한 내용은 [AWS Glue Studio 사용 설명서](#)를 참조하십시오.

대화형 세션을 사용하여 개발

대화형 세션을 사용하면 선택한 환경에서 애플리케이션을 구축하고 테스트할 수 있습니다. 자세한 내용은 [AWS Glue를 사용하여 대화형 세션 사용](#)을 참조하세요.

Docker 이미지를 사용하여 개발

Note

이 섹션의 지침은 Microsoft Windows 운영 체제에서 테스트되지 않았습니다.

Windows 플랫폼에서의 로컬 개발 및 테스트에 대한 자세한 내용은 블로그([Building an AWS Glue ETL pipeline locally without an AWS account](#))를 참조하세요.

프로덕션 준비 데이터 플랫폼의 경우 AWS Glue에 대한 개발 프로세스 및 CI/CD 파이프라인 작업이 핵심 주제입니다. 도커 컨테이너에서 AWS Glue 작업을 유연하게 개발하고 테스트할 수 있습니다. AWS Glue는 Docker Hub에서 도커 이미지를 호스팅하여 추가 유틸리티로 개발 환경을 설정합니다. AWS Glue ETL 라이브러리를 사용하여 선호하는 IDE, 노트북 또는 REPL을 사용할 수 있습니다. 이 주제에서는 도커 이미지를 사용하여 도커 컨테이너에서 AWS Glue 버전 4.0 작업을 개발하고 테스트하는 방법을 설명합니다.

Docker Hub에서 AWS Glue에 대해 사용할 수 있는 도커 이미지는 다음과 같습니다.

- AWS Glue 버전 4.0의 경우: `amazon/aws-glue-libs:glue_libs_4.0.0_image_01`
- AWS Glue 버전 3.0의 경우: `amazon/aws-glue-libs:glue_libs_3.0.0_image_01`
- AWS Glue 버전 2.0의 경우: `amazon/aws-glue-libs:glue_libs_2.0.0_image_01`

이러한 이미지는 x86_64용입니다. 이 아키텍처에서 테스트하는 것이 좋습니다. 하지만 지원되지 않는 기본 이미지에서 로컬 개발 솔루션을 재작업하는 것은 가능할 수도 있습니다.

이 예제는 로컬 시스템에서 `amazon/aws-glue-libs:glue_libs_4.0.0_image_01` 사용 및 컨테이너 실행을 설명합니다. 이 컨테이너 이미지는 AWS Glue 버전 3.3 Spark 작업에 대해 테스트되었습니다. 이 이미지는 다음이 포함되어 있습니다.

- Amazon Linux
- AWS Glue ETL 라이브러리([aws-glue-libs](#))
- Apache Spark 3.3.0
- Spark 기록 서버
- Jupyter Lab
- Livy
- 다른 라이브러리 종속성(AWS Glue 작업 시스템 중 하나와 동일한 집합)

요구 사항에 따라 다음 섹션 중 하나를 완료합니다.

- spark-submit을 사용하도록 컨테이너 설정
- REPL 셸(PySpark)을 사용하도록 컨테이너 설정
- Pytest를 사용하도록 컨테이너 설정
- Pytest를 사용하도록 컨테이너 설정
- Visual Studio Code를 사용하도록 컨테이너 설정

필수 조건

시작하기 전에 Docker가 설치되어 있고 Docker 데몬이 실행 중인지 확인하세요. 설치 지침은 [Mac](#) 또는 [Linux](#)용 도커 설명서를 참조하세요. Docker를 실행하는 시스템은 AWS Glue 컨테이너를 호스트합니다. 또한 Docker를 실행하는 호스트의 이미지를 위해 7GB 이상의 디스크 공간이 있는지 확인하세요.

로컬로 AWS Glue 코드 개발 시 제한에 대한 자세한 정보는 [로컬 개발 제한 사항](#)을 참조하세요.

AWS 구성

컨테이너에서 AWS API 호출을 활성화하려면 다음 단계에 따라 AWS 자격 증명을 설정합니다. 다음 섹션에서는 AWS(이)라는 이름의 프로필을 사용합니다.

1. AWS CLI를 설정하고 명명된 프로필을 구성합니다. AWS CLI 구성에 대한 자세한 내용은 AWS CLI 사용 설명서의 [구성 및 자격 증명 파일 설정](#)을 참조하세요.
2. 터미널에서 다음 명령을 실행합니다.

```
PROFILE_NAME="<your_profile_name>"
```

요청을 보낼 AWS 리전을 지정하려면 AWS_REGION 환경 변수를 설정해야 할 수도 있습니다.

컨테이너 설정 및 실행

spark-submit 명령을 통해 PySpark 코드를 실행하도록 컨테이너를 설정하는 데는 다음 수준의 단계를 포함합니다.

1. Docker Hub에서 이미지를 가져옵니다.
2. 컨테이너를 실행합니다.

Docker Hub에서 이미지 가져오기

Docker Hub로부터 이미지를 로컬 시스템으로 가져오려면 다음 명령을 실행합니다.

```
docker pull amazon/aws-glue-libs:glue_libs_4.0.0_image_01
```

컨테이너 실행

이제 이 이미지를 사용하여 컨테이너를 실행할 수 있습니다. 요구 사항에 따라 다음 중 하나를 선택할 수 있습니다.

spark-submit

컨테이너에서 spark-submit 명령을 실행하여 AWS Glue 작업 스크립트를 실행할 수 있습니다.

1. 스크립트를 작성하고 /local_path_to_workspace 디렉터리에 sample1.py(으)로 저장합니다. 샘플 코드는 이 주제의 부록으로 포함되어 있습니다.

```
$ WORKSPACE_LOCATION=/local_path_to_workspace
$ SCRIPT_FILE_NAME=sample.py
$ mkdir -p ${WORKSPACE_LOCATION}/src
$ vim ${WORKSPACE_LOCATION}/src/${SCRIPT_FILE_NAME}
```

2. 다음 명령을 실행하여 컨테이너에서 spark-submit 명령을 실행하여 새 Spark 애플리케이션을 제출합니다.

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 --name glue_spark_submit amazon/aws-glue-libs:glue_libs_4.0.0_image_01 spark-submit /home/glue_user/workspace/src/${SCRIPT_FILE_NAME}
```

```

...22/01/26 09:08:55 INFO DAGScheduler: Job 0 finished: fromRDD at
  DynamicFrame.scala:305, took 3.639886 s
root
|-- family_name: string
|-- name: string
|-- links: array
| |-- element: struct
| | |-- note: string
| | |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
| |-- element: struct
| | |-- scheme: string
| | |-- identifier: string
|-- other_names: array
| |-- element: struct
| | |-- lang: string
| | |-- note: string
| | |-- name: string
|-- sort_name: string
|-- images: array
| |-- element: struct
| | |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
| |-- element: struct
| | |-- type: string
| | |-- value: string
|-- death_date: string

...

```

3. (선택 사항) 환경에 맞게 `spark-submit`을 구성합니다. 예를 들어 `--jars` 구성을 사용하여 종속성을 전달할 수 있습니다. 자세한 내용은 Spark 설명서에서 [Launching Applications with spark-submit](#)을 참조하세요.

REPL 셸(PySpark)

대화형 개발을 위해 REPL(read-eval-print loops) 셸을 실행할 수 있습니다.

다음 명령을 실행하여 컨테이너에서 PySpark 명령을 실행하여 REPL 셸을 시작합니다.

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -e AWS_PROFILE=$PROFILE_NAME -e
  DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 --name glue_pyspark amazon/aws-glue-
  libs:glue_libs_4.0.0_image_01 pyspark
...
  ____  _
 /  _/  _/  _/  _/  _/
 _\  \  _  \  \  \  \
 /_  /  .  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \
 /_  /
 /_  /

Using Python version 3.7.10 (default, Jun 3 2021 00:02:01)
Spark context Web UI available at http://56e99d000c99:4040
Spark context available as 'sc' (master = local[*], app id = local-1643011860812).
SparkSession available as 'spark'.
>>>
```

Pytest

단위 테스트의 경우 AWS Glue Spark 작업 스크립트를 위해 pytest를 사용할 수 있습니다.

준비를 위해 다음 명령을 실행합니다.

```
$ WORKSPACE_LOCATION=/local_path_to_workspace
$ SCRIPT_FILE_NAME=sample.py
$ UNIT_TEST_FILE_NAME=test_sample.py
$ mkdir -p ${WORKSPACE_LOCATION}/tests
$ vim ${WORKSPACE_LOCATION}/tests/${UNIT_TEST_FILE_NAME}
```

다음 명령을 실행하여 테스트 스위트에서 pytest을(를) 실행합니다.

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/home/glue_user/
  workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true --rm -p 4040:4040 -p
  18080:18080 --name glue_pytest amazon/aws-glue-libraries:glue_libs_4.0.0_image_01 -c
  "python3 -m pytest"
starting org.apache.spark.deploy.history.HistoryServer,
  logging to /home/glue_user/spark/logs/spark-glue_user-
  org.apache.spark.deploy.history.HistoryServer-1-5168f209bd78.out
*===== test session starts
  =====
*platform linux -- Python 3.7.10, pytest-6.2.3, py-1.11.0, pluggy-0.13.1
```

```

rootdir: /home/glue_user/workspace
plugins: anyio-3.4.0
*collected 1 item *

tests/test_sample.py . [100%]

===== warnings summary =====
tests/test_sample.py::test_counts
/home/glue_user/spark/python/pyspark/sql/context.py:79: DeprecationWarning: Deprecated
in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
DeprecationWarning)

-- Docs: https://docs.pytest.org/en/stable/warnings.html
===== 1 passed, *1 warning* in
21.07s =====

```

Jupyter Lab

노트북에서 대화형 개발 및 임시 쿼리를 위해 Jupyter를 시작할 수 있습니다.

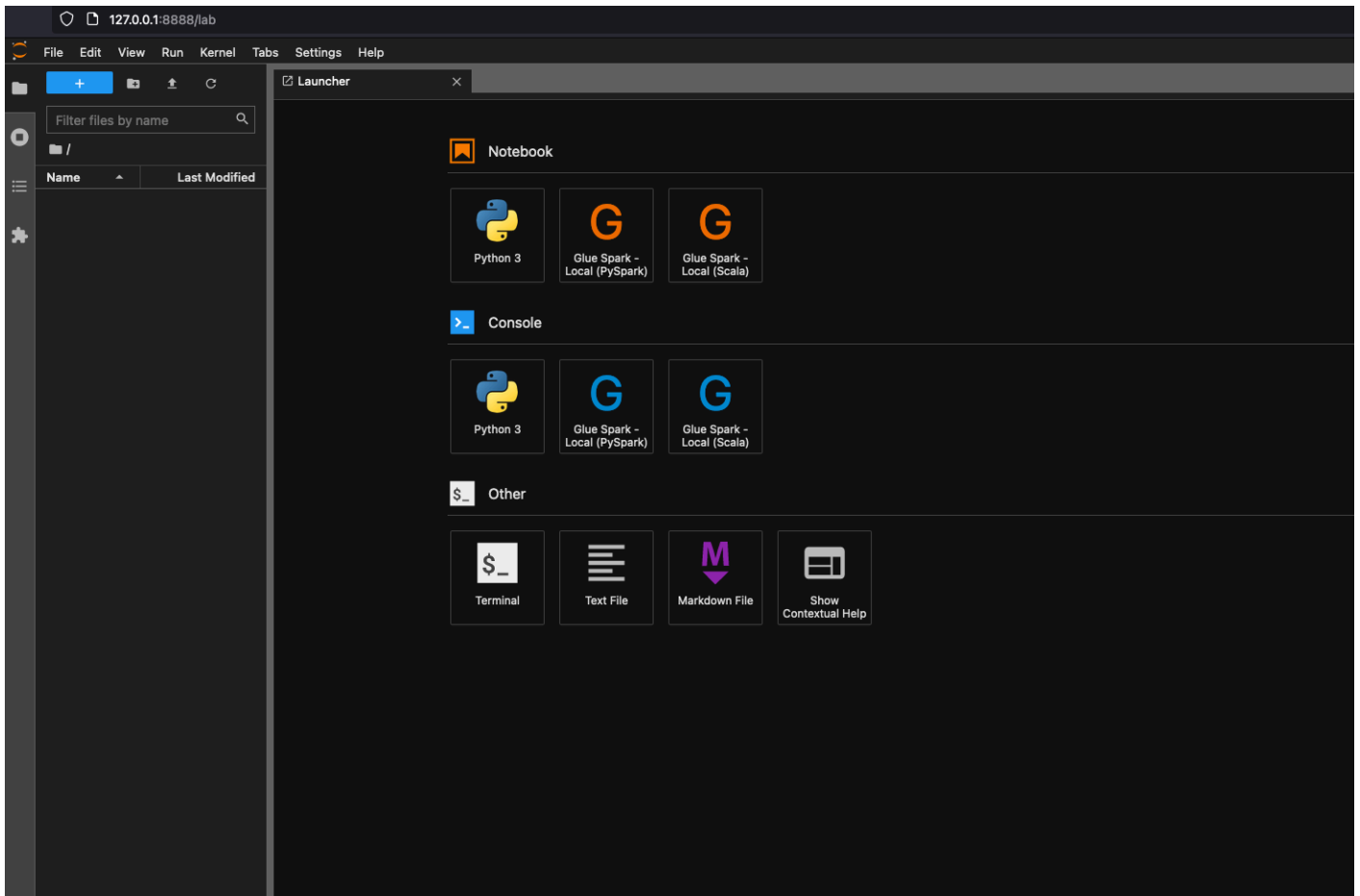
1. Jupyter Lab을 시작하려면 다음 명령을 실행합니다.

```

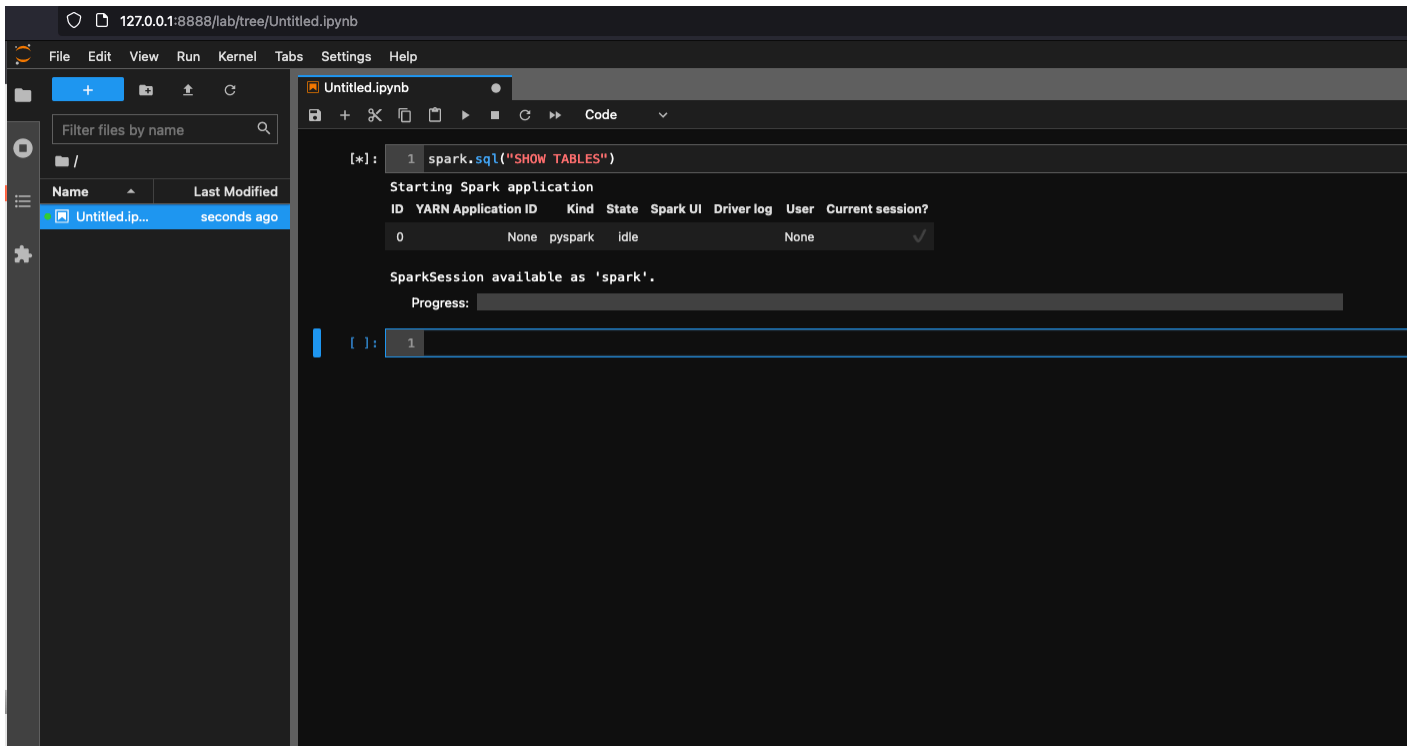
$ JUPYTER_WORKSPACE_LOCATION=/local_path_to_workspace/jupyter_workspace/
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $JUPYTER_WORKSPACE_LOCATION:/
home/glue_user/workspace/jupyter_workspace/ -e AWS_PROFILE=$PROFILE_NAME -e
DISABLE_SSL=true --rm -p 4040:4040 -p 18080:18080 -p 8998:8998 -p 8888:8888 --name
glue_jupyter_lab amazon/aws-glue-libs:glue_libs_4.0.0_image_01 /home/glue_user/
jupyter/jupyter_start.sh
...
[I 2022-01-24 08:19:21.368 ServerApp] Serving notebooks from local directory: /home/
glue_user/workspace/jupyter_workspace
[I 2022-01-24 08:19:21.368 ServerApp] Jupyter Server 1.13.1 is running at:
[I 2022-01-24 08:19:21.368 ServerApp] http://faa541f8f99f:8888/lab
[I 2022-01-24 08:19:21.368 ServerApp] or http://127.0.0.1:8888/lab
[I 2022-01-24 08:19:21.368 ServerApp] Use Control-C to stop this server and shut down
all kernels (twice to skip confirmation).

```

2. Jupyter 랩 UI를 보기 위해, 로컬 시스템의 웹 브라우저에서 <http://127.0.0.1:8888/lab>을 엽니다.



3. 노트북에서 Glue Spark Local(PySpark)을 선택합니다. 대화형 Jupyter Notebook UI에서 코드 개발을 시작할 수 있습니다.



Visual Studio Code를 사용하도록 컨테이너 설정

사전 조건:

1. Visual Studio Code를 설치합니다.
2. [Python](#)을 설치합니다.
3. [Visual Studio Code Remote - Containers](#)를 설치합니다.
4. Visual Studio Code에서 작업 영역 폴더를 엽니다.
5. 설정(Settings)을 선택합니다.
6. Workspace를 선택합니다.
7. Open Settings(설정 열기)(JSON)를 선택합니다.
8. 다음 JSON을 붙여 넣고 저장합니다.

```
{
  "python.defaultInterpreterPath": "/usr/bin/python3",
  "python.analysis.extraPaths": [
    "/home/glue_user/aws-glue-libs/PyGlue.zip:/home/glue_user/spark/python/lib/
py4j-0.10.9-src.zip:/home/glue_user/spark/python/",
  ]
}
```

}

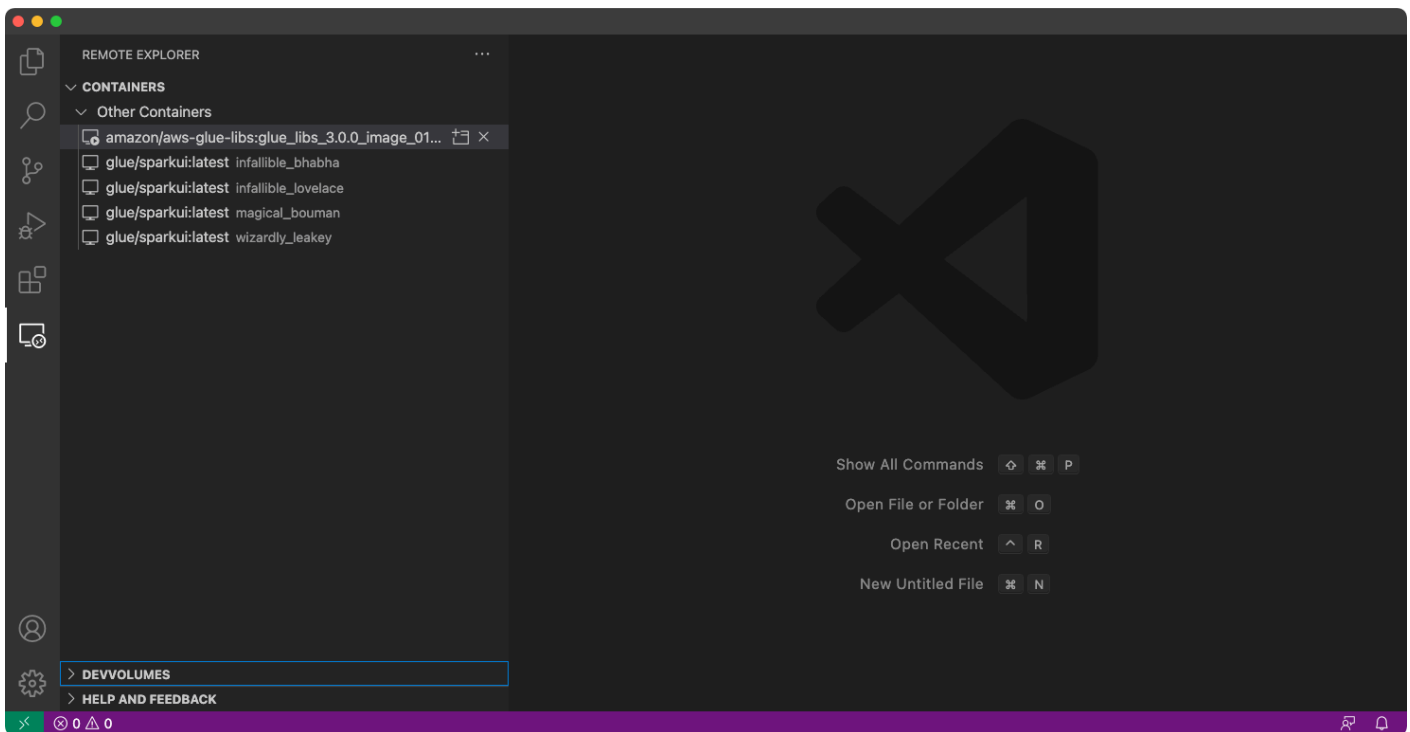
단계:

1. Docker 컨테이너를 실행합니다.

```
$ docker run -it -v ~/.aws:/home/glue_user/.aws -v $WORKSPACE_LOCATION:/
home/glue_user/workspace/ -e AWS_PROFILE=$PROFILE_NAME -e DISABLE_SSL=true
--rm -p 4040:4040 -p 18080:18080 --name glue_pyspark amazon/aws-glue-
libs:glue_libs_4.0.0_image_01 pyspark
```

2. Visual Studio Code를 시작합니다.

3. 왼쪽 메뉴에서 Remote Explorer를 선택하고 amazon/aws-glue-libs:glue_libs_4.0.0_image_01을(를) 선택합니다.



4. 마우스 오른쪽 버튼을 클릭하고 Attach to Container(컨테이너에 첨부)을 선택합니다. 대화 상자가 표시되면 Got it(선택)을 선택합니다.

5. /home/glue_user/workspace/를 엽니다.

6. Glue PySpark 스크립트를 만들고 Run(실행)을 선택합니다.

스크립트가 성공적으로 실행됨을 확인할 수 있습니다.


```

sample.py — workspace [Container amazon/aws-glue-libs:glue_libs_3.0.0_image_01 (glue_pyspark)]
EXPLORER
WORKSPACE [CONTAINER AMAZON/AWS...]
  .vscode
  settings.json
  src
    sample.py
    tests
      test_sample.py
sample.py
7
8 class GluePythonSampleTest:
9     def __init__(self):
10        params = []
11        if '--JOB_NAME' in sys.argv:
12            params.append('JOB_NAME')
13        args = getResolvedOptions(sys.argv, params)
14
15        self.context = GlueContext(SparkContext.getOrCreate())
16        self.job = Job(self.context)
17
18        if 'JOB_NAME' in args:
19            jobname = args['JOB_NAME']
20        else:
21            jobname = "test"
22        self.job.init(jobname, args)
23
24    def run(self):
25        dyf = read_json(self.context, "s3://awsglue-datasets/examples/us-legislators/all/persons.json")
26        dyf.printSchema()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
| | | |— name: string
|— sort_name: string
|— images: array
| | | |— element: struct
| | | | |— url: string
|— given_name: string
|— birth_date: string
|— id: string
|— contact_details: array
| | | |— element: struct
| | | | |— type: string
| | | | |— value: string
|— death_date: string
[glue_user@6d963c62aae0 workspace]$
Python 3.8.9 64-bit
Ln 8, Col 21 Spaces: 4 UTF-8 LF Python

```

부록: 테스트를 위한 AWS Glue 작업 샘플 코드

이 부록에서는 테스트 목적의 AWS Glue 작업 샘플 코드와 같은 스크립트를 제공합니다.

sample.py: Amazon S3 API 호출을 사용하여 AWS Glue ETL 라이브러리를 활용하기 위한 샘플 코드

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

class GluePythonSampleTest:
    def __init__(self):
        params = []
        if '--JOB_NAME' in sys.argv:
            params.append('JOB_NAME')
        args = getResolvedOptions(sys.argv, params)

```

```

self.context = GlueContext(SparkContext.getOrCreate())
self.job = Job(self.context)

if 'JOB_NAME' in args:
    jobname = args['JOB_NAME']
else:
    jobname = "test"
self.job.init(jobname, args)

def run(self):
    dyf = read_json(self.context, "s3://awsglue-datasets/examples/us-legislators/
all/persons.json")
    dyf.printSchema()

    self.job.commit()

def read_json(glue_context, path):
    dynamicframe = glue_context.create_dynamic_frame.from_options(
        connection_type='s3',
        connection_options={
            'paths': [path],
            'recurse': True
        },
        format='json'
    )
    return dynamicframe

if __name__ == '__main__':
    GluePythonSampleTest().run()

```

위 코드는 AWS IAM에 Amazon S3 권한이 필요합니다. Amazon S3 경로에 대해 ListBucket 및 GetObject 호출을 허용하는 IAM 관리형 정책 `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess` 또는 IAM 사용자 지정 정책을 부여해야 합니다.

`test_sample.py`: `sample.py` 단위 테스트를 위한 샘플 코드입니다.

```

import pytest
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

```

```

import sys
from src import sample

@pytest.fixture(scope="module", autouse=True)
def glue_context():
    sys.argv.append('--JOB_NAME')
    sys.argv.append('test_count')

    args = getResolvedOptions(sys.argv, ['JOB_NAME'])
    context = GlueContext(SparkContext.getOrCreate())
    job = Job(context)
    job.init(args['JOB_NAME'], args)

    yield(context)

    job.commit()

def test_counts(glue_context):
    dyf = sample.read_json(glue_context, "s3://awsglue-datasets/examples/us-
legislators/all/persons.json")
    assert dyf.toDF().count() == 1961

```

AWS Glue ETL 라이브러리를 통한 개발

AWS Glue ETL 라이브러리는 퍼블릭 Amazon S3 버킷에서 제공되며, Apache Maven 구축 시스템에서 사용될 수 있습니다. 따라서 네트워크 연결 없이 로컬에서도 Python 및 Scala ETL 스크립트를 개발하여 테스트할 수 있습니다. 이 라이브러리 사용을 위해 적절하게 구성된 환경을 제공하므로 도커 이미지를 사용한 로컬 개발을 권장합니다.

로컬 개발은 AWS Glue 버전 0.9, 1.0, 2.0 및 그 이상을 포함한 모든 AWS Glue 버전에서 사용할 수 있습니다. AWS Glue에서 사용할 수 있는 Python 및 Apache Spark 버전에 대한 자세한 내용은 [Glue version job property](#) 단원을 참조하십시오.

라이브러리는 Amazon 소프트웨어 라이선스(<https://aws.amazon.com/asi>)와 함께 제공됩니다.

로컬 개발 제한 사항

AWS Glue Scala 라이브러리를 사용해 로컬에서 개발할 때는 다음 제한 사항을 주의하십시오.

- AWS Glue 라이브러리를 사용해 어셈블리 jar("fat jar" 또는 "uber jar")을 생성하지 마십시오. 그러면 다음 기능이 비활성화될 수 있습니다.

- [작업 북마크](#)
- AWS Glue Parquet 쓰기([AWS Glue에서 Parquet 형식 사용](#))
- FillMissingValues 변환([Scala](#) 또는 [Python](#))

위 기능은 AWS Glue 작업 시스템에서만 사용할 수 있습니다.

- [FindMatches 변환](#)은 로컬 개발에서 지원되지 않습니다.
- [벡터화된 SIMD CSV 리더](#)는 로컬 개발에서 지원되지 않습니다.
- 로컬 개발에서는 S3 경로에서 JDBC 드라이버를 로드하기 위한 [customJdbcDriverS3Path](#) 속성이 지원되지 않습니다. 또는 로컬에 JDBC 드라이버를 다운로드하고 로컬에서 로드할 수도 있습니다.
- [Glue Data Quality](#)는 로컬 개발에서 지원되지 않습니다.

Python을 사용한 로컬 개발

몇 가지 선행 단계를 마친 후 AWS Glue 유틸리티를 사용해 Python ETL 스크립트를 테스트하고 제출합니다.

로컬 Python 개발을 위한 선행 작업

다음 단계에 따라 로컬 Python 개발을 준비합니다.

1. GitHub(<https://github.com/aws-labs/aws-glue-libs>)에서 AWS Glue Python 리포지토리를 복제합니다.
2. 다음 중 하나를 수행하십시오.
 - AWS Glue 버전 0.9의 경우 glue-0.9 브랜치를 확인합니다.
 - AWS Glue 버전 1.0의 경우 glue-1.0 브랜치를 확인합니다. AWS Glue 0.9 이후의 최신 버전에서는 Python 3을 지원합니다.
 - AWS Glue 버전 2.0의 경우 glue-2.0 브랜치를 확인합니다.
 - AWS Glue 버전 3.0의 경우 glue-3.0 브랜치를 확인합니다.
 - AWS Glue 버전 4.0의 경우 master 브랜치를 확인합니다.
3. Apache Maven을 다음 위치에서 설치합니다. <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-common/apache-maven-3.6.0-bin.tar.gz>.
4. Apache Spark 배포판을 다음 위치에 설치합니다.

- AWS Glue 버전 0.9의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz>
 - AWS Glue 버전 1.0의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz>
 - AWS Glue 버전 2.0의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-2.0/spark-2.4.3-bin-hadoop2.8.tgz>
 - AWS Glue 버전 3.0의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-3.0/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3.tgz>
 - AWS Glue 버전 4.0의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-4.0/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0.tgz>
5. Spark 아카이브에서 추출된 루트 위치로 설정하여 SPARK_HOME 환경 변수를 내보냅니다. 예:
- AWS Glue 버전 0.9의 경우: `export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`
 - AWS Glue 버전 1.0 및 2.0의 경우: `export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`
 - AWS Glue 버전 3.0의 경우: `export SPARK_HOME=/home/$USER/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3`
 - AWS Glue 버전 4.0의 경우: `export SPARK_HOME=/home/$USER/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0`

Python ETL 스크립트 실행

AWS Glue jar 파일을 로컬 개발에 사용할 경우 AWS Glue Python 패키지를 로컬에서 실행할 수 있습니다.

Python 스크립트를 테스트 및 실행하는 데 사용되는 유틸리티와 프레임워크는 다음과 같습니다. 다음 표에 나열된 명령은 [AWS Glue Python 패키지](#)의 루트 디렉터리에서 실행됩니다.

유틸리티	Command	설명
AWS Glue 셸	<code>./bin/gluepyspark</code>	AWS Glue ETL 라이브러리와 통합되는 셸에 Python 스크립트를 입력하여 실행합니다.
AWS Glue 제출	<code>./bin/gluesparksubmit</code>	실행할 전체 Python 스크립트를 제출합니다.

유틸리티	Command	설명
Pytest	<code>./bin/gluepytest</code>	Python 코드를 작성하여 유닛 테스트를 실행합니다. 이때 <code>pytest</code> 모듈을 설치하여 PATH에서 사용해야 합니다. 자세한 내용은 pytest 설명서 를 참조하십시오.

Scala를 사용한 로컬 개발

몇 가지 선행 단계를 마친 후 Maven 명령을 사용해 로컬에서 Scala ETL 스크립트를 실행합니다.

로컬 Scala 개발을 위한 선행 작업

다음 단계를 완료하여 로컬 Scala 개발을 준비합니다.

1단계: 소프트웨어 설치

이 단계에서는 소프트웨어를 설치하고 필요한 환경 변수를 설정합니다.

1. Apache Maven을 다음 위치에서 설치합니다. <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-common/apache-maven-3.6.0-bin.tar.gz>.
2. Apache Spark 배포판을 다음 위치에 설치합니다.
 - AWS Glue 버전 0.9의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-0.9/spark-2.2.1-bin-hadoop2.7.tgz>
 - AWS Glue 버전 1.0의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-1.0/spark-2.4.3-bin-hadoop2.8.tgz>
 - AWS Glue 버전 2.0의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-2.0/spark-2.4.3-bin-hadoop2.8.tgz>
 - AWS Glue 버전 3.0의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-3.0/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3.tgz>
 - AWS Glue 버전 4.0의 경우: <https://aws-glue-etl-artifacts.s3.amazonaws.com/glue-4.0/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0.tgz>
3. Spark 아카이브에서 추출된 루트 위치로 설정하여 `SPARK_HOME` 환경 변수를 내보냅니다. 예:
 - AWS Glue 버전 0.9의 경우: `export SPARK_HOME=/home/$USER/spark-2.2.1-bin-hadoop2.7`

- AWS Glue 버전 1.0 및 2.0의 경우: `export SPARK_HOME=/home/$USER/spark-2.4.3-bin-spark-2.4.3-bin-hadoop2.8`
- AWS Glue 버전 3.0의 경우: `export SPARK_HOME=/home/$USER/spark-3.1.1-amzn-0-bin-3.2.1-amzn-3`
- AWS Glue 버전 4.0의 경우: `export SPARK_HOME=/home/$USER/spark-3.3.0-amzn-1-bin-3.3.3-amzn-0`

2단계: Maven 프로젝트 구성

다음 pom.xml 파일을 AWS Glue Scala 애플리케이션 템플릿으로 사용합니다. 여기에는 필요한 dependencies, repositories 및 plugins 요소가 포함되어 있습니다. Glue version 문자열을 다음 중 하나로 바꿉니다.

- AWS Glue 버전 4.0용 4.0.0
- AWS Glue 버전 3.0용 3.0.0
- AWS Glue 버전 1.0 또는 2.0용 1.0.0
- AWS Glue 버전 0.9용 0.9.0

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.amazonaws</groupId>
  <artifactId>AWSGlueApp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>${project.artifactId}</name>
  <description>AWS ETL application</description>

  <properties>
    <scala.version>2.11.1 for AWS Glue 2.0 or below, 2.12.7 for AWS Glue 3.0
and 4.0</scala.version>
    <glue.version>Glue version with three numbers (as mentioned earlier)</
glue.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.scala-lang</groupId>
      <artifactId>scala-library</artifactId>
```

```

        <version>${scala.version}</version>
    <!-- A "provided" dependency, this will be ignored when you package your application
-->
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>AWSGlueETL</artifactId>
    <version>${glue.version}</version>
    <!-- A "provided" dependency, this will be ignored when you package your
application -->
        <scope>provided</scope>
    </dependency>
</dependencies>

<repositories>
    <repository>
        <id>aws-glue-etl-artifacts</id>
        <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/</url>
    </repository>
</repositories>
<build>
    <sourceDirectory>src/main/scala</sourceDirectory>
    <plugins>
        <plugin>
            <!-- see http://davidb.github.com/scala-maven-plugin -->
            <groupId>net.alchim31.maven</groupId>
            <artifactId>scala-maven-plugin</artifactId>
            <version>3.4.0</version>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>testCompile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>1.6.0</version>
            <executions>
                <execution>

```



```
        <goals>
          <goal>java</goal>
        </goals>
      </execution>
    </executions>
  </configuration>
  <systemProperties>
    <systemProperty>
      <key>spark.master</key>
      <value>local[*]</value>
    </systemProperty>
    <systemProperty>
      <key>spark.app.name</key>
      <value>localrun</value>
    </systemProperty>
    <systemProperty>
      <key>org.xerial.snappy.lib.name</key>
      <value>libsnappyjava.jnilib</value>
    </systemProperty>
  </systemProperties>
</configuration>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-enforcer-plugin</artifactId>
  <version>3.0.0-M2</version>
  <executions>
    <execution>
      <id>enforce-maven</id>
      <goals>
        <goal>enforce</goal>
      </goals>
      <configuration>
        <rules>
          <requireMavenVersion>
            <version>3.5.3</version>
          </requireMavenVersion>
        </rules>
      </configuration>
    </execution>
  </executions>
</plugin>
```

<!-- The shade plugin will be helpful in building a uberjar or fatjar.

You can use this jar in the AWS Glue runtime environment. For more information, see <https://maven.apache.org/plugins/maven-shade-plugin/> -->

```

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>3.2.4</version>
      <configuration>
        <!-- any other shade configurations -->
      </configuration>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
</project>

```

Scala ETL 스크립트 실행

Maven 프로젝트 루트 디렉터리에서 다음 명령을 사용해 Scala ETL 스크립트를 실행합니다.

```
mvn exec:java -Dexec.mainClass="mainClass" -Dexec.args="--JOB-NAME jobName"
```

위에서 *mainClass*는 스크립트 메인 클래스의 정규화 클래스 이름으로 변경하십시오. *jobName*은 원하는 작업 이름으로 변경하십시오.

테스트 환경 구성

로컬 테스트 환경 구성의 예는 다음 블로그 도움말을 참조하세요.

- [Building an AWS Glue ETL pipeline locally without an AWS account](#)
- [Developing AWS Glue ETL jobs locally using a container](#)

ETL 스크립트를 테스트하기 위해 개발 엔드포인트 또는 노트북을 사용하려면 [개발 엔드포인트를 사용하여 스크립트 개발](#)를 참조하세요.

Note

개발 엔드포인트는 AWS Glue 버전 2.0 작업과 함께 사용하도록 지원되지 않습니다. 자세한 내용은 [단축된 시작 시간으로 Spark ETL 작업 실행](#)을 참조하세요.

개발 엔드포인트

Note

개발 엔드포인트의 콘솔 경험은 2023년 3월 31일을 기점으로 제거됩니다. 개발 엔드포인트의 생성, 업데이트 및 모니터링은 [개발 엔드포인트 API](#) 및 [AWS Glue CLI](#)를 통해 계속 사용 가능합니다.

아래 나열된 이유로 개발 엔드포인트에서 대화형 세션으로 마이그레이션하는 것이 좋습니다. 개발 엔드포인트에서 대화형 세션으로 마이그레이션하는 방법에 대한 필수 작업은 [개발 엔드포인트에서 대화형 세션으로 마이그레이션](#)을 참조하세요.

설명	dev 엔드포인트	대화형 세션
Glue 버전 지원	AWS Glue 버전 0.9 및 1.0을 지원합니다	AWS Glue 버전 2.0 이상을 지원합니다.
개발 엔드포인트는 향후 아시아 태평양(자카르타)(ap-southeast-3), 중동(UAE)(me-central-1), 유럽(스페인)(eu-south-2), 유럽(취리히)(eu-central-2) 또는 기타 신규 리전에서 사용할 수 없습니다.	대화형 세션은 현재 중동(UAE)(me-central-1) 리전에서는 사용할 수 없지만 나중에 사용할 수 있습니다.	
Spark 클러스터에 대한 액세스 방법	SSH, REPL 셸, Jupyter Notebook, IDE(예: PyCharm) 지원	AWS Glue Studio notebook, Jupyter notebook, 다양한 IDE(예: Visual Studio Code,

설명	dev 엔드포인트	대화형 세션
		PyCharm) 및 SageMaker notebook 지원
첫 번째 쿼리 시간	Spark 클러스터를 설정하는 데 10~15분 소요	임시 Spark 클러스터를 설정하는 데 최대 1분이 소요될 수 있음
가격 모델	AWS에서는 엔드포인트가 프로비저닝된 시간과 DPU 수를 기준으로 개발 엔드포인트 요금이 청구됩니다. 개발 엔드포인트는 제한 시간이 없습니다. 프로비저닝된 각 개발 엔드포인트에는 최소 10분의 청구 기간이 있습니다. 또한 AWS에서는 Amazon EC2 인스턴스의 Jupyter Notebook과 개발 엔드포인트로 구성된 SageMaker 노트북에 대한 요금을 부과합니다.	AWS는 세션이 활성화된 시간과 DPU 수를 기준으로 대화형 세션 요금을 청구합니다. 대화형 세션에는 유휴 시간 제한을 구성할 수 있습니다. AWS Glue Studio 노트북에는 대화형 세션을 위한 내장 인터페이스가 추가 비용 없이 제공됩니다. 각 대화형 세션에는 최소 1분의 청구 기간이 있습니다. AWS Glue Studio 노트북은 대화형 세션을 위한 기본 제공 인터페이스를 추가 비용 없이 제공합니다.
콘솔 경험	CLI 및 API를 통해서만 사용 가능	AWS Glue 콘솔, CLI 및 API를 통해 사용 가능

개발 엔드포인트에서 대화형 세션으로 마이그레이션

다음 체크리스트를 사용하여 개발 엔드포인트에서 대화형 세션으로 마이그레이션하는 적절한 방법을 결정합니다.

스크립트가 AWS Glue 0.9 또는 1.0 특정 기능(예: HDFS, YARN 등)에 종속됩니까?

대답이 '예'인 경우 Glue 0.9 또는 1.0에서 Glue 3.0 이상으로 마이그레이션하는 방법을 알아보려면 [AWS Glue 버전 3.0으로 AWS Glue 작업 마이그레이션](#)을 참조하세요.

개발 엔드포인트에 액세스하는 데 사용하는 방법은 무엇입니까?

사용되는 방법	수행할 작업
SageMaker 노트북, Jupyter notebook 또는 Jupyter Lab	Jupyter에서 .ipynb 파일을 다운로드하여 AWS Glue Studio 노트북 으로 마이그레이션하고 .ipynb 파일을 업로드하여 새 AWS Glue Studio 노트북 작업을 생성합니다. 또는 SageMaker Studio 를 사용하고 AWS Glue 커널을 선택할 수도 있습니다.
Zeppelin 노트북	코드를 복사하여 붙여 넣거나 ze2nb와 같은 타사 변환기를 자동으로 사용하여 노트북을 Jupyter Notebook으로 수동으로 변환할 수 있습니다. 그런 다음 AWS Glue Studio 노트북 또는 SageMaker Studio에서 노트북을 사용합니다.
IDE	PyCharm에서 AWS Glue 대화형 세션을 사용하여 AWS Glue 작업 작성 또는 Microsoft Visual Studio 코드로 대화형 세션 사용 을 참조하세요.
REPL	aws-glue-session package 를 로컬로 설치한 후 다음 명령을 실행합니다. <ul style="list-style-type: none"> Python의 경우: <code>jupyter console --kernel glue_pyspark</code> Scala의 경우: <code>jupyter console --kernel glue_spark</code>
SSH	대화형 세션에는 해당 옵션이 없습니다. 또는 도커 이미지를 사용할 수 있습니다. 자세한 내용은 도커 이미지를 사용하여 개발 을 참조하세요.

다음 단원에서는 dev 엔드포인트를 사용하여 AWS Glue 버전 1.0에서 작업을 개발하는 방법에 대한 정보를 제공합니다.

주제

- [개발 엔드포인트를 사용하여 스크립트 개발](#)
- [노트북 관리](#)

개발 엔드포인트를 사용하여 스크립트 개발

Note

개발 엔드포인트는 AWS Glue의 2.0 이전 버전에서만 지원됩니다. ETL 스크립트를 작성하고 테스트할 수 있는 대화형 환경에서는 [AWS Glue Studio에서 노트북](#)을 사용합니다.

AWS Glue는 추출, 변환 및 로드(ETL) 스크립트를 반복적으로 개발하고 테스트할 수 있는 개발 엔드포인트라는 환경을 생성할 수 있습니다. AWS Glue 콘솔 또는 API를 사용하여 개발 엔드포인트를 만들고, 편집하고, 삭제할 수 있습니다.

개발 환경 관리

개발 엔드포인트를 만들 때 구성 값을 제공하여 개발 환경을 제공합니다. 이 값은 AWS Glue가 어떻게 네트워크를 설치하는지 알려줘 안전하게 엔드포인트로 액세스할 수 있고 엔드포인트는 데이터 스토어에 액세스할 수 있습니다.

그런 다음, 엔드포인트로 연결되는 노트북을 만들고 이를 사용하여 ETL 스크립트를 작성 및 테스트할 수 있습니다. 개발 절차에 따른 결과를 만족하면 스크립트를 실행하는 ETL 작업을 생성합니다. 이 과정을 통해 함수를 추가하고 상호적으로 스크립트를 디버깅할 수 있습니다.

이 섹션의 자습서에 따라 노트북에서 개발 엔드포인트를 사용하는 방법을 배우십시오.

주제

- [개발 엔드포인트 워크플로](#)
- [AWS Glue 개발 엔드포인트가 SageMaker 노트북과 함께 작동하는 방식](#)
- [개발 엔드포인트 추가](#)
- [개발 엔드포인트 액세스](#)
- [자습서: JupyterLab에 Jupyter Notebook을 설정하여 ETL 스크립트 테스트 및 디버깅](#)
- [자습서: 개발 엔드포인트와 SageMaker 노트북 함께 사용](#)
- [자습서: 개발 엔드포인트를 통해 REPL 셸 사용하기](#)
- [자습서: 개발 엔드포인트로 PyCharm Professional 설치하기](#)
- [고급 구성: 여러 사용자 간에 개발 엔드포인트 공유](#)

개발 엔드포인트 워크플로

다음 절차에 따라 AWS Glue 개발 엔드포인트를 사용합니다.

1. API를 사용하여 개발 엔드포인트를 생성합니다. 이 엔드포인트를 정의한 보안 그룹으로 Virtual Private Cloud(VPC)에서 시작합니다.
2. 개발 엔드포인트가 프로비저닝되어 작업을 시작할 준비가 된 상태가 될 때까지 API를 사용하여 개발 엔드포인트를 폴링합니다. 준비가 되었다면, 다음 방법 중 하나를 사용해 개발 엔드포인트로 연결하여 AWS Glue 스크립트를 작성 및 테스트할 수 있습니다.
 - 계정에서 SageMaker 노트북을 생성합니다. 노트북을 생성하는 방법에 대한 자세한 내용은 [the section called “AWS Glue Studio 노트북을 사용한 코드 작성”](#) 단원을 참조하십시오.
 - 터미널 창을 열어 바로 개발 엔드포인트로 연결합니다.
 - JetBrains [PyCharm Python IDE](#)의 전문가 버전이 있다면 개발 엔드포인트로 연결하고 이것을 사용하여 상호적으로 개발합니다. 스크립트에 pydevd 문을 입력하면 PyCharm은 원격 중단점을 지원할 수 있습니다.
3. 개발 엔드포인트의 디버깅과 테스트가 완료된다면 이것을 삭제할 수 있습니다.

AWS Glue 개발 엔드포인트가 SageMaker 노트북과 함께 작동하는 방식

개발 엔드포인트에 액세스하는 보편적인 방법 중 하나로 SageMaker에서 [Jupyter](#)를 사용하는 방법이 있습니다. Jupyter Notebook은 오픈 소스 웹 애플리케이션의 일종으로 시각화, 분석, 기계 학습 등에 폭넓게 사용됩니다. AWS Glue SageMaker 노트북이 AWS Glue 개발 엔드포인트를 포함한 Jupyter Notebook 환경을 제공합니다. AWS Glue SageMaker 노트북에서 Jupyter Notebook 환경은 [SparkMagic](#)으로 사전 구성되어 있습니다. 이것은 오픈 소스 Jupyter 플러그인으로 Spark 작업을 원격 Spark 클러스터에 제출하는 역할을 합니다. [Apache Livy](#)는 REST API를 통해 원격 Spark 클러스터와 상호작용을 주고받게 해주는 서비스입니다. AWS Glue SageMaker 노트북에 SparkMagic이 구성되어 있어 AWS Glue 개발 엔드포인트에서 실행되는 Livy 서버에 대하여 REST API를 호출합니다.

다음 텍스트 흐름은 각각의 구성 요소의 작동 방식을 설명한 것입니다.

AWS Glue SageMaker 노트북: (Jupyter → SparkMagic) → (네트워크) → AWS Glue 개발 엔드포인트: (Apache Livy → Apache Spark)

Jupyter Notebook의 각 단락에 쓰인 Spark 스크립트를 실행하면 해당 Spark 코드를 SparkMagic을 통해 Livy 서버로 제출하고, 그런 다음 "livy-session-N"이라는 이름의 Spark 작업이 Spark 클러스터에서 실행됩니다. 이 작업을 Livy 세션이라고 합니다. 이 Spark 작업은 노트북 세션이 실행 중인 동안 실행됨

니다. 노트북에서 Jupyter 커널을 종료하거나 세션이 시간 초과되면 해당 Spark 작업이 종료됩니다. 노트북(.ipnyb) 파일당 Spark 작업이 하나씩 실행됩니다.

여러 개의 SageMaker 노트북 인스턴스와 단 한 개의 AWS Glue 개발 엔드포인트만 사용할 수 있습니다. 각각의 SageMaker 노트북 인스턴스에 노트북 파일을 여러 개 생성할 수 있습니다. 각각의 노트북 파일을 열고 단락을 실행하면 SparkMagic을 통해 Spark 클러스터의 노트북 파일당 Livy 세션이 하나씩 실행됩니다. Livy 세션은 각각 Spark 작업 한 개씩에 상응합니다.

AWS Glue 개발 엔드포인트 및 SageMaker 노트북의 기본 동작

Spark 작업은 [Spark 구성](#)에 따라 실행됩니다. Spark 구성은 여러 가지 방식으로 설정할 수 있습니다 (예: Spark 클러스터 구성, SparkMagic 구성 등).

기본적으로 Spark는 Spark 클러스터 구성에 따라 클러스터 리소스를 Livy 세션에 할당합니다. AWS Glue 개발 엔드포인트의 경우 클러스터 구성은 작업자 유형에 좌우됩니다. 다음은 작업자 유형에 따른 보편적인 구성을 설명한 테이블입니다.

	표준	G.1X	G.2X
<code>spark.driver.memory</code>	5G	10G	20G
<code>spark.executor.memory</code>	5G	10G	20G
<code>spark.executor.cores</code>	4	8	16
<code>spark.dynamicAllocation.enabled</code>	TRUE	TRUE	TRUE

Spark 실행기 최대 수는 DPU(또는 NumberOfWorkers)와 작업자 유형을 조합하여 자동으로 계산됩니다.

	표준	G.1X	G.2X
Spark 실행기 최대 수	$(\text{DPU} - 1) * 2 - 1$	$(\text{NumberOfWorkers} - 1)$	$(\text{NumberOfWorkers} - 1)$

예를 들어 개발 엔드포인트에 작업자가 10개이고 작업자 유형이 G.1X인 경우, Spark 실행기는 9개이고 클러스터 전체에는 실행기 메모리가 90G가 됩니다. 실행기마다 메모리가 10G씩이 때문입니다.

지정된 작업자 유형과 관계없이 Spark 동적 리소스 할당이 활성화됩니다. 데이터 집합이 충분히 크면 Spark가 모든 실행기를 하나의 Livy 세션에 할당할 가능성이 있습니다. spark.dynamicAllocation.maxExecutors가 기본적으로 설정되어 있지 않기 때문입니다. 이는 같은 개발 엔드포인트상의 다른 Livy 세션은 새 실행기 실행을 대기하게 된다는 뜻입니다. 데이터 집합 크기가 작은 경우 Spark가 실행기를 동시에 여러 Livy 세션에 할당할 수 있습니다.

Note

다양한 사용 사례에서 리소스 할당 방식을 자세히 알아보고 동작을 수정하기 위해 구성을 변경하는 법을 알아보려면 [고급 구성: 여러 사용자 간에 개발 엔드포인트 공유](#) 섹션을 참조하세요.

개발 엔드포인트 추가

AWS Glue에서 개발 엔드포인트를 사용하여 추출, 변환 및 로드(ETL) 스크립트를 반복적으로 개발하고 테스트합니다. 개발 엔드포인트 작업은 AWS Command Line Interface에서만 가능합니다.

1. 명령줄 창에 다음과 유사한 명령을 입력합니다.

```
aws glue create-dev-endpoint --endpoint-name "endpoint1" --role-arn
"arn:aws:iam::account-id:role/role-name" --number-of-nodes "3" --glue-version
"1.0" --arguments '{"GLUE_PYTHON_VERSION": "3"}' --region "region-name"
```

이 명령은 AWS Glue 버전 1.0을 지정합니다. 이 버전은 Python 2와 Python 3를 모두 지원하므로 arguments 파라미터를 사용하여 원하는 Python 버전을 표시할 수 있습니다. glue-version 파라미터가 생략된 경우, AWS Glue 버전 0.9로 간주됩니다. AWS Glue 버전에 대한 자세한 내용은 [Glue version job property](#) 단원을 참조하십시오.

추가적인 명령줄 파라미터에 대한 자세한 내용은 AWS CLI Command Reference의 [create-dev-endpoint](#)를 참조하세요.

- (선택 사항) 개발 엔드포인트 상태를 확인하려면 다음 명령을 입력하십시오. 상태가 READY로 변경되면 개발 엔드포인트를 사용할 준비가 된 것입니다.

```
aws glue get-dev-endpoint --endpoint-name "endpoint1"
```

개발 엔드포인트 액세스

Virtual Private Cloud(VPC)에서 개발 엔드포인트를 생성하면, AWS Glue는 프라이빗 IP 주소만 반환합니다. 퍼블릭 IP 주소 필드는 채워지지 않습니다. 비 VPC 개발 엔드포인트를 생성할 때 AWS Glue은 (는) 퍼블릭 IP 주소만 반환합니다.

개발 엔드포인트가 퍼블릭 주소가 있다면 다음 예와 같이 개발 엔드포인트의 SSH 프라이빗 키를 통해 접근 가능하다는 것을 확인합니다.

```
ssh -i dev-endpoint-private-key.pem glue@public-address
```

개발 엔드포인트에 프라이빗 주소가 있고 VPC 서브넷이 퍼블릭 인터넷에서 라우팅 가능하며 보안 그룹이 클라이언트의 인바운드 액세스를 허용한다고 가정하십시오. 이 경우, 다음 단계에 따라 탄력적 IP 주소를 개발 엔드포인트에 연결하여 인터넷 액세스를 허용하십시오.

Note

탄력적 IP 주소를 사용하려면 사용 중인 서브넷은 라우팅 테이블을 통해 연결되는 인터넷 게이트웨이가 있어야 합니다.

탄력적 IP 주소를 연결하여 개발 엔드포인트에 액세스하려면

- <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
- 탐색 창에서 개발 엔드포인트를 선택하고 개발 엔드포인트 세부 정보 페이지로 이동하십시오. 다음 단계에서 사용할 수 있도록 [Private address(프라이빗 주소)]를 기록합니다.
- <https://console.aws.amazon.com/ec2/>에서 Amazon EC2 콘솔을 엽니다.
- 탐색 창의 Network & Security(네트워크 및 보안) 아래에서 Network Interfaces(네트워크 인터페이스)를 선택합니다.
- AWS Glue 콘솔 개발 엔드포인트 세부 정보 페이지의 [프라이빗 주소(Private address)]에 상응하는 [프라이빗 DNS(IPv4)(Private DNS (IPv4))]를 검색합니다.

Amazon EC2 콘솔에 표시할 열을 수정해야 할 수 있습니다. 네트워크 인터페이스 ID(ENI)(예: eni-12345678)는 이 주소를 위한 것이라고 알아야 합니다.

6. Amazon EC2 콘솔의 [네트워크 및 보안(Network & Security)] 아래에서 [탄력적 IP(Elastic IPs)]를 선택합니다.
7. 새 주소 할당과 할당 순으로 선택하여 새로운 탄력적 IP 주소를 할당합니다.
8. [Elastic IPs(탄력적 IP)] 페이지에서 새롭게 할당된 [Elastic IP(탄력적 IP)]를 선택합니다. 그런 다음 [Actions(작업)]을 선택한 후 Associate address(관련 주소)를 선택합니다.
9. 주소 연결 페이지에서 다음과 같이 진행합니다.
 - [Resource type(리소스 유형)]에서 Network interface(네트워크 인터페이스)을 선택합니다.
 - 네트워크 인터페이스 상자에 프라이빗 주소를 위한 Network interface ID(네트워크 인터페이스 ID)(ENI)를 입력합니다.
 - Associate(연결)를 선택합니다.
10. 다음 예와 같이, 새롭게 연결된 탄력적 IP 주소가 개발 엔드포인트의 SSH 프라이빗 키를 통해 접근 가능하다면 인증합니다.

```
ssh -i dev-endpoint-private-key.pem glue@elastic-ip
```

Bastion 호스트를 이용해 개발 엔드포인트의 프라이빗 주소에 대한 SSH 액세스를 가져오는 데 대한 자세한 내용은 AWS 보안 블로그 게시물 [Securely Connect to Linux Instances Running in a Private Amazon VPC](#)를 참조하세요.

자습서: JupyterLab에 Jupyter Notebook을 설정하여 ETL 스크립트 테스트 및 디버깅

이 튜토리얼에서는 로컬 시스템에서 실행 중인 JupyterLab의 Jupyter Notebook을 개발 엔드포인트에 연결합니다. 이렇게 하면 AWS Glue 추출, 변환, 로드(ETL) 스크립트를 배포하기에 앞서 대화식으로 이를 실행, 디버깅 및 테스트할 수 있습니다. 이 자습서에서는 Secure Shell(SSH) 전송을 사용하여 로컬 시스템을 AWS Glue 개발 엔드포인트에 연결합니다. 자세한 내용은 Wikipedia의 [Port forwarding](#)을 참조하세요.

1단계: JupyterLab과 Sparkmagic 설치

JupyterLab을 설치하려면 conda 또는 pip를 사용하면 됩니다. conda는 오픈 소스 패키지 관리 시스템 겸 환경 관리 시스템으로 Windows, macOS 및 Linux에서 실행됩니다. pip는 Python용 패키지 설치 프로그램입니다.

macOS에 설치하는 경우, Xcode가 설치되어 있어야 Sparkmagic을 설치할 수 있습니다.

1. JupyterLab, Sparkmagic과 관련 확장 프로그램을 설치합니다.

```
$ conda install -c conda-forge jupyterlab
$ pip install sparkmagic
$ jupyter nbextension enable --py --sys-prefix widgetsnbextension
$ jupyter labextension install @jupyter-widgets/jupyterlab-manager
```

2. Location에서 sparkmagic 디렉터리를 확인합니다.

```
$ pip show sparkmagic | grep Location
Location: /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-packages
```

3. 디렉터리를 Location에 대하여 반환된 것으로 변경하고, Scala와 PySpark의 커널을 설치합니다.

```
$ cd /Users/username/.pyenv/versions/anaconda3-5.3.1/lib/python3.7/site-packages
$ jupyter-kernelspec install sparkmagic/kernels/sparkkernel
$ jupyter-kernelspec install sparkmagic/kernels/pysparkkernel
```

4. 샘플 config 파일을 다운로드합니다.

```
$ curl -o ~/.sparkmagic/config.json https://raw.githubusercontent.com/jupyter-incubator/sparkmagic/master/sparkmagic/example_config.json
```

이 구성 파일에서 driverMemory 및 executorCores와 같은 Spark 관련 파라미터를 구성할 수 있습니다.

2단계: JupyterLab 시작

JupyterLab을 시작하면 기본 웹 브라우저가 자동으로 열리고 URL `http://localhost:8888/lab/workspaces/{workspace_name}`이 표시됩니다.

```
$ jupyter lab
```

3단계: SSH 포트 전송을 시작하여 개발 엔드포인트에 연결

다음으로, SSH 로컬 포트 전송을 사용하여 로컬 포트(여기서는 8998)를 AWS Glue(169.254.76.1:8998)에 의해 정의된 원격 대상으로 전송합니다.

1. SSH에 액세스를 부여하는 별도의 터미널 창을 엽니다. Microsoft Windows에서라면 [Git for Windows](#)에서 제공하는 BASH 셸을 사용해도 되고, 아니면 [Cygwin](#)을 설치해도 됩니다.
2. 다음과 같이 수정된 다음 SSH 명령을 실행합니다.
 - *private-key-file-path* 대신 개발 엔드포인트를 생성하는 데 쓴 퍼블릭 키에 상응하는 프라이빗 키를 포함한 .pem 파일에 대한 경로를 사용합니다.
 - 8998 외의 다른 포트를 전송하는 경우, 8998 대신 로컬에서 실제로 사용 중인 포트 번호를 사용합니다. 주소 169.254.76.1:8998이 원격 포트이며 이는 사용자가 변경하는 것이 아닙니다.
 - *dev-endpoint-public-dns*를 개발 엔드포인트의 퍼블릭 DNS 주소로 바꿉니다. 이 주소를 찾으려면 AWS Glue 콘솔에서 개발 엔드포인트로 이동하여 이름을 선택하고 [엔드포인트 세부 정보(Endpoint details)] 페이지에 목록으로 기재된 [퍼블릭 주소(Public address)]를 복사합니다.

```
ssh -i private-key-file-path -NTL 8998:169.254.76.1:8998 glue@dev-endpoint-public-dns
```

다음과 같은 경고 메시지가 표시됩니다.

```
The authenticity of host 'ec2-xx-xxx-xxx-xx.us-west-2.compute.amazonaws.com
(xx.xxx.xxx.xx)'
can't be established. ECDSA key fingerprint is SHA256:4e97875Brt+1wKzRko
+Jf1Snp21X7aTP3BcFnHYLEts.
Are you sure you want to continue connecting (yes/no)?
```

yes를 입력하고 JupyterLab을 사용하는 동안 터미널 창을 열어둡니다.

3. SSH 포트 전송이 개발 엔드포인트와 올바르게 함께 작동하는지 확인합니다.

```
$ curl localhost:8998/sessions
{"from":0,"total":0,"sessions":[]}
```

4단계: Notebook Paragraph의 Simple Script Fragment 실행

이제 JupyterLab의 노트북이 개발 엔드포인트와 함께 작동하는 것이 정상입니다. 다음과 같은 스크립트 조각을 노트북에 입력하여 실행합니다.

1. Spark가 실행 중인지 확인합니다. 다음 명령을 사용하면 Spark에 1을 계산하여 값을 인쇄하도록 지시합니다.

```
spark.sql("select 1").show()
```

2. AWS Glue Data Catalog 통합이 작동하는지 확인합니다. 다음 명령을 사용하면 테이블을 데이터 카탈로그에 목록으로 나열합니다.

```
spark.sql("show tables").show()
```

3. AWS Glue 라이브러리를 사용하는 단순한 스크립트 조각이 작동하는지 확인합니다.

다음 스크립트는 AWS Glue Data Catalog의 `persons_json` 테이블 메타데이터를 사용하여 샘플 데이터로부터 `DynamicFrame`을 생성합니다. 그런 다음 항목 수와 이 데이터 스키마를 출력합니다.

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create a Glue context
glueContext = GlueContext(SparkContext.getOrCreate())

# Create a DynamicFrame using the 'persons_json' table
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
    table_name="persons_json")

# Print out information about *this* data
print("Count: ", persons_DyF.count())
persons_DyF.printSchema()
```

스크립트의 출력은 다음과 같습니다.

```
Count: 1961
root
```

```
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
```

문제 해결

- JupyterLab 설치 중에 컴퓨터가 회사 프록시나 방화벽 뒤에 위치한 경우, 회사 IT 부서에서 관리하는 사용자 정의 보안 프로파일로 인해 HTTP 및 SSL 오류가 발생할 수 있습니다.

다음은 conda가 자체 리포지토리에 연결할 수 없을 때 발생하는 일반적인 오류의 예시입니다.

```
CondaHTTPError: HTTP 000 CONNECTION FAILED for url <https://repo.anaconda.com/pkg/main/win-64/current_repodata.json>
```

이것은 회사에서 Python 및 JavaScript 커뮤니티에서 광범위하게 사용되는 리포지토리로의 연결을 차단할 수 있기 때문에 발생할 가능성이 있습니다. 자세한 내용은 JupyterLab 웹 사이트의 [Installation Problems](#)를 참조하세요.

- 개발 엔드포인트에 연결을 시도할 때 [연결 거부됨(connection refused)] 오류가 발생하는 경우 날짜가 지난 개발 엔드포인트를 사용하고 있는 것일 수 있습니다. 새로운 개발 엔드포인트를 생성해보고 다시 연결하십시오.

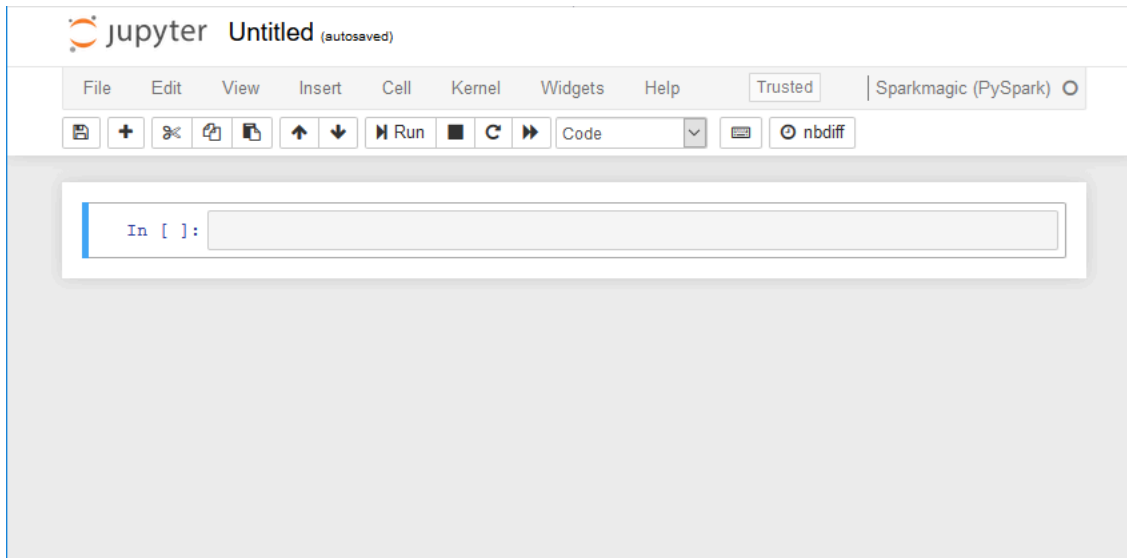
자습서: 개발 엔드포인트와 SageMaker 노트북 함께 사용

AWS Glue에서는 개발 엔드포인트를 생성하고 SageMaker 노트북을 생성하여 ETL과 기계 학습 스크립트 개발에 도움을 얻을 수 있습니다. SageMaker 노트북은 Jupyter Notebook 애플리케이션을 실행하는 완전 관리형 기계 학습 컴퓨팅 인스턴스입니다.

1. AWS Glue 콘솔에서 Dev endpoints(개발 엔드포인트)를 선택하여 개발 엔드포인트 목록을 탐색합니다.
2. 사용할 개발 엔드포인트의 이름 옆에 있는 확인란을 선택하고 작업 메뉴에서 SageMaker 노트북 생성을 선택합니다.
3. 다음과 같이 노트북 생성 및 구성 페이지를 작성합니다.
 - a. 노트북 이름을 입력합니다.
 - b. 개발 엔드포인트에 연결에서 개발 엔드포인트를 확인합니다.
 - c. AWS Identity and Access Management(IAM) 역할을 생성하거나 선택합니다.

역할을 생성하는 것이 좋습니다. 기존 역할을 사용하는 경우 필요한 권한이 있는지 확인합니다. 자세한 내용은 [the section called “6단계: SageMaker 노트북용 IAM 정책 생성”](#) 단원을 참조하십시오.
 - d. (선택 사항) VPC, 서브넷 및 하나 이상의 보안 그룹을 선택합니다.
 - e. (선택 사항) AWS Key Management Service 암호화 키를 선택합니다.
 - f. (선택 사항) 노트북 인스턴스에 대한 태그를 추가합니다.
4. 노트북 생성을 선택합니다. 노트북 페이지의 오른쪽 상단에 있는 새로 고침 아이콘을 선택하고 상태가 Ready로 표시될 때까지 계속합니다.
5. 새 노트북 이름 옆의 확인란을 선택한 다음 노트북 열기를 선택합니다.
6. 새 노트북 생성: jupyter 페이지에서 신규를 선택한 다음 Sparkmagic (PySpark)을 선택합니다.

이제 화면이 다음과 같이 보여야 합니다.



7. (선택 사항) 페이지 상단에서 Untitled를 선택하고 노트북에 이름을 지정합니다.
8. Spark 애플리케이션을 시작하려면 노트북에 다음 명령을 입력한 다음 도구 모음에서 실행을 선택합니다.

```
spark
```

잠시 후 다음과 같은 응답이 표시됩니다.

```
In [1]: spark
Starting Spark application



| ID | YARN Application ID            | Kind    | State | Spark UI             | Driver log           | Current session? |
|----|--------------------------------|---------|-------|----------------------|----------------------|------------------|
| 0  | application_1576209965005_0001 | pyspark | idle  | <a href="#">Link</a> | <a href="#">Link</a> | ✓                |



SparkSession available as 'spark'.
<pyspark.sql.session.SparkSession object at 0x7f3d54913550>
```

9. 동적 프레임을 생성하고 쿼리를 실행합니다. 복사, 붙여넣기 및 persons_json 테이블의 개수 및 스키마를 출력하는 다음 코드를 실행합니다.

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
glueContext = GlueContext(SparkContext.getOrCreate())
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
    table_name="persons_json")
print ("Count: ", persons_DyF.count())
```

```
persons_DyF.printSchema()
```

자습서: 개발 엔드포인트를 통해 REPL 셸 사용하기

AWS Glue를 통해 개발 엔드포인트를 만들고 REPL(Read-Evaluate-Print Loop) 셸을 호출하여 점차 늘리면서 PySpark 코드를 실행합니다. 그러면 ETL 스크립트를 디버깅하기 전에 상호적으로 디버깅할 수 있습니다.

개발 엔드포인트에서 REPL을 사용하려면 엔드포인트에 대한 SSH에 권한을 부여해야 합니다.

1. 로컬 컴퓨터에서 SSH 명령어를 실행할 수 있는 터미널창을 열고 편집한 SSH 명령어에 붙여 넣습니다. 명령을 실행합니다.

개발 엔드포인트에 대해 Python 3을 사용하는 AWS Glue 버전 1.0을 수락했다고 가정하면 출력은 다음과 같습니다.

```
Python 3.6.8 (default, Aug  2 2019, 17:42:44)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux
Type "help", "copyright", "credits" or "license" for more information.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/share/aws/glue/etl/jars/glue-assembly.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/spark/jars/slf4j-log4j12-1.7.16.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
2019-09-23 22:12:23,071 WARN [Thread-5] yarn.Client (Logging.scala:logWarning(66))
- Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading
libraries under SPARK_HOME.
2019-09-23 22:12:26,562 WARN [Thread-5] yarn.Client (Logging.scala:logWarning(66))
- Same name resource file:/usr/lib/spark/python/lib/pyspark.zip added multiple
times to distributed cache
2019-09-23 22:12:26,580 WARN [Thread-5] yarn.Client (Logging.scala:logWarning(66))
- Same path resource file:///usr/share/aws/glue/etl/python/PyGlue.zip added
multiple times to distributed cache.
2019-09-23 22:12:26,581 WARN [Thread-5] yarn.Client (Logging.scala:logWarning(66))
- Same path resource file:///usr/lib/spark/python/lib/py4j-src.zip added multiple
times to distributed cache.
```

```
2019-09-23 22:12:26,581 WARN [Thread-5] yarn.Client (Logging.scala:logWarning(66))
- Same path resource file:///usr/share/aws/glue/libs/pyspark.zip added multiple
times to distributed cache.
```

Welcome to

```

  ____
 /  __ \   _   _   _   _   _   _
-\  \ \ /  \ /   \ /   \ /   \ /
/_  /  .  \ /  /  /  /  /  /  /
   /  \
                                     version 2.4.3
   /  \

```

Using Python version 3.6.8 (default, Aug 2 2019 17:42:44)

SparkSession available as 'spark'.

>>>

2. `print(spark.version)`을 입력하여 REPL 셸이 올바르게 작동하는지 테스트합니다. Spark 버전을 나타내는 한 REPL은 사용할 준비되어 있습니다.
3. 셸에서 다음 샘플 스크립트를 행마다 실행하고자 합니다.

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import *
glueContext = GlueContext(SparkContext.getOrCreate())
persons_DyF = glueContext.create_dynamic_frame.from_catalog(database="legislators",
table_name="persons_json")
print ("Count: ", persons_DyF.count())
persons_DyF.printSchema()
```

자습서: 개발 엔드포인트로 PyCharm Professional 설치하기

이 자습서를 통해 로컬 시스템에서 실행되는 [PyCharm Professional](#) Python IDE를 개발 엔드포인트로 연결하여 AWS Glue ETL(추출, 변환 및 로드) 스크립트를 배치하기 전에 상호적으로 실행, 디버깅, 및 테스트할 수 있습니다. 자습서의 지침 및 화면 캡처는 PyCharm Professional 버전 2019.3을 기반으로 합니다.

상호적으로 개발 엔드포인트를 연결하기 위해서 PyCharm Professional이 설치되어 있어야 합니다. 무료 에디션을 사용하면 아무것도 할 수 없습니다.

Note

이 튜토리얼에서는 Amazon S3를 데이터 원본으로 사용합니다. 대신 JDBC 데이터 원본을 사용하려면 Virtual Private Cloud(VPC)에서 개발 엔드포인트를 실행해야 합니다. SSH를 사용하여 VPC의 개발 엔드포인트에 연결하려면 SSH 터널을 생성해야 합니다. 이 자습서에는 SSH 터널을 생성하는 지침이 포함되어 있지 않습니다. SSH를 사용하여 VPC의 개발 엔드포인트에 연결하는 방법에 대한 자세한 내용은 AWS 보안 블로그의 [Securely Connect to Linux Instances Running in a Private Amazon VPC](#)를 참조하세요.

주제

- [PyCharm Professional을 개발 엔드포인트에 연결](#)
- [스크린을 개발 엔드포인트에 배포](#)
- [원격 인터프리터 구성](#)
- [스크립트를 개발 엔드포인트에서 실행](#)

PyCharm Professional을 개발 엔드포인트에 연결

1. legislators라는 PyCharm에서 새로운 순수 Python 프로젝트를 생성합니다.
2. get_person_schema.py라는 파일을 다음 내용으로 프로젝트에 생성합니다.

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

def main():
    # Create a Glue context
    glueContext = GlueContext(SparkContext.getOrCreate())

    # Create a DynamicFrame using the 'persons_json' table
    persons_DyF =
glueContext.create_dynamic_frame.from_catalog(database="legislators",
table_name="persons_json")

    # Print out information about this data
    print("Count: ", persons_DyF.count())
    persons_DyF.printSchema()
```

```
if __name__ == "__main__":
    main()
```

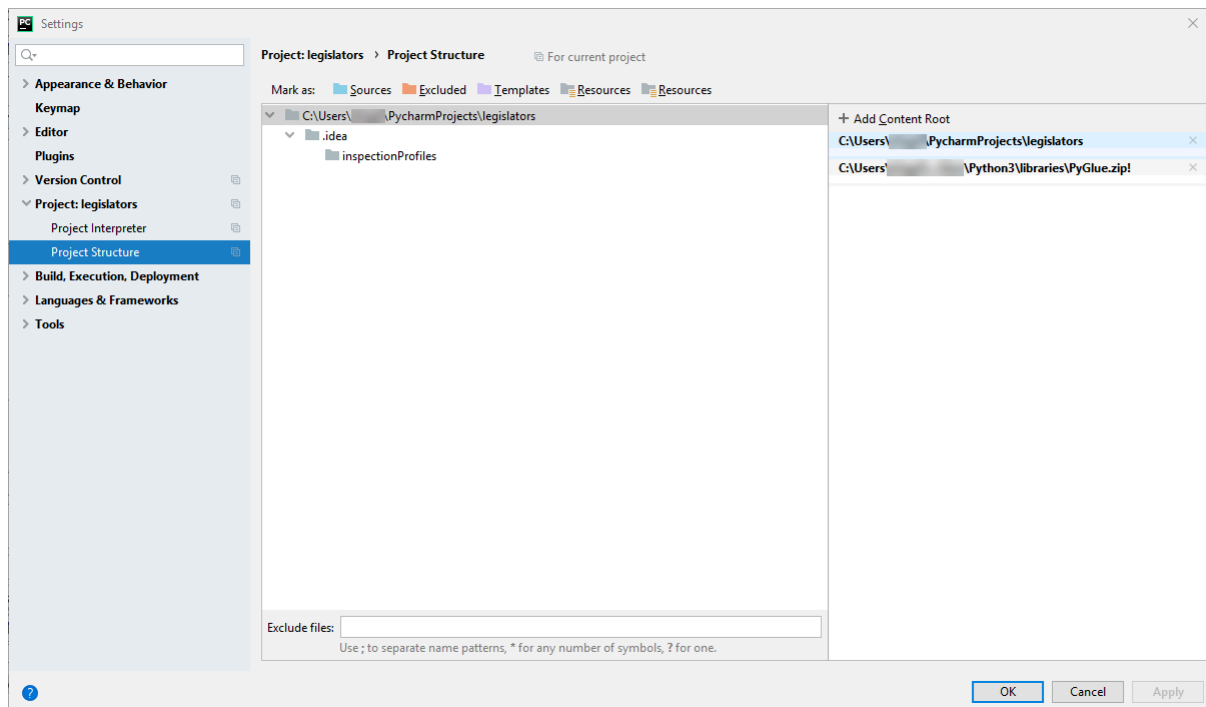
3. 다음 중 하나를 수행하십시오.

- AWS Glue 버전 0.9의 경우, AWS Glue Python 라이브러리 파일 PyGlue.zip을 <https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl/python/PyGlue.zip>에서 로컬 시스템의 편리한 위치로 다운로드합니다.
- AWS Glue 버전 1.0 이상의 경우, AWS Glue Python 라이브러리 파일 PyGlue.zip을 <https://s3.amazonaws.com/aws-glue-jes-prod-us-east-1-assets/etl-1.0/python/PyGlue.zip>에서 로컬 시스템의 편리한 위치로 다운로드합니다.

4. PyGlue.zip을 PyCharm의 프로젝트 내용 루트로써 추가합니다.

- PyCharm에서 [Settings(설정)] 대화 상자를 열기 위해서 [File(파일)]과 [Settings(설정)]를 선택합니다. (Ctrl+Alt+S를 눌러도 됩니다.)
- legislators 프로젝트를 확장하고 Project Structure(프로젝트 구조)를 선택합니다. 오른쪽 창에서 [+ Add Content Root(내용 루트 추가)]를 선택합니다.
- PyGlue.zip을 저장한 위치로 이동하여 선택한 다음 [Apply(적용)]를 선택합니다.

[Settings(설정)] 스크린은 다음과 비슷해야 합니다.

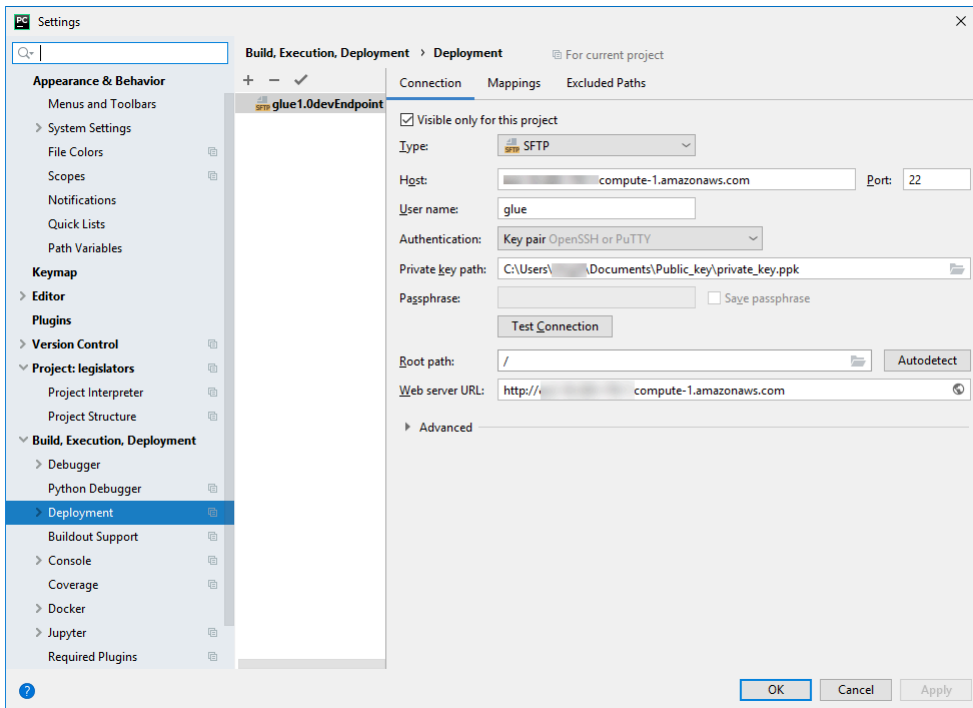


[적용(Apply)]을 선택한 다음 [설정(Settings)] 대화 상자를 열어 둡니다.

5. 개발 옵션을 구성하여 로컬 스크립트를 SFTP(이 기능은 PyCharm Professional에서만 사용 가능합니다)를 사용하여 개발 엔드포인트로 업로드합니다.
 - [Settings(설정)] 대화 상자에서 [Build, Execution, Deployment(설계, 실행, 배포)] 섹션을 확장합니다. Deployment(배포) 부 섹션을 선택합니다.
 - 중간 창의 맨 윗쪽에서 [+] 아이콘을 선택하여 새로운 서버를 추가합니다. 유형을 SFTP로 설정하고 이름을 지정합니다.
 - SFTP 호스트를 세부 정보 페이지에 나열된 대로 개발 엔드포인트의 퍼블릭 주소로 설정합니다. (AWS Glue 콘솔에서 개발 엔드포인트의 이름을 선택하여 세부 정보 페이지를 표시합니다.) VPC에서 실행 중인 개발 엔드포인트의 경우 SFTP 호스트를 호스트 주소로 설정하고 SSH 터널의 로컬 포트를 개발 엔드포인트로 설정합니다.
 - User name(사용자 이름)을 glue에 설치합니다.
 - Auth type(Auth 유형)을 Key pair (OpenSSH or Putty)(키 페어(OpenSSH 또는 Putty))에 설치합니다. 개발 엔드포인트 프라이빗 키 파일이 위치한 곳을 검색하여 [Private key file(프라이빗 키 파일)]을 설정합니다. PyCharm은 DSA, RSA 및 ECDSA OpenSSH 키 유형만 지원하므로, Putty의 프라이빗 형식의 키는 허용하지 않습니다. ssh-keygen의 업데이트 버전을 사용하여 PyCharm이 허용한 키 페어 유형을 다음과 같은 구문을 사용하여 생성합니다.


```
ssh-keygen -t rsa -f <key_file_name> -C "<your_email_address>"
```
 - [Test connection(연결 테스트)]을 선택하고 연결을 테스트하도록 허용합니다. 연결이 성공하면 [Apply(적용)]를 선택합니다.

[Settings(설정)] 스크린은 이제 다음과 비슷해야 합니다.

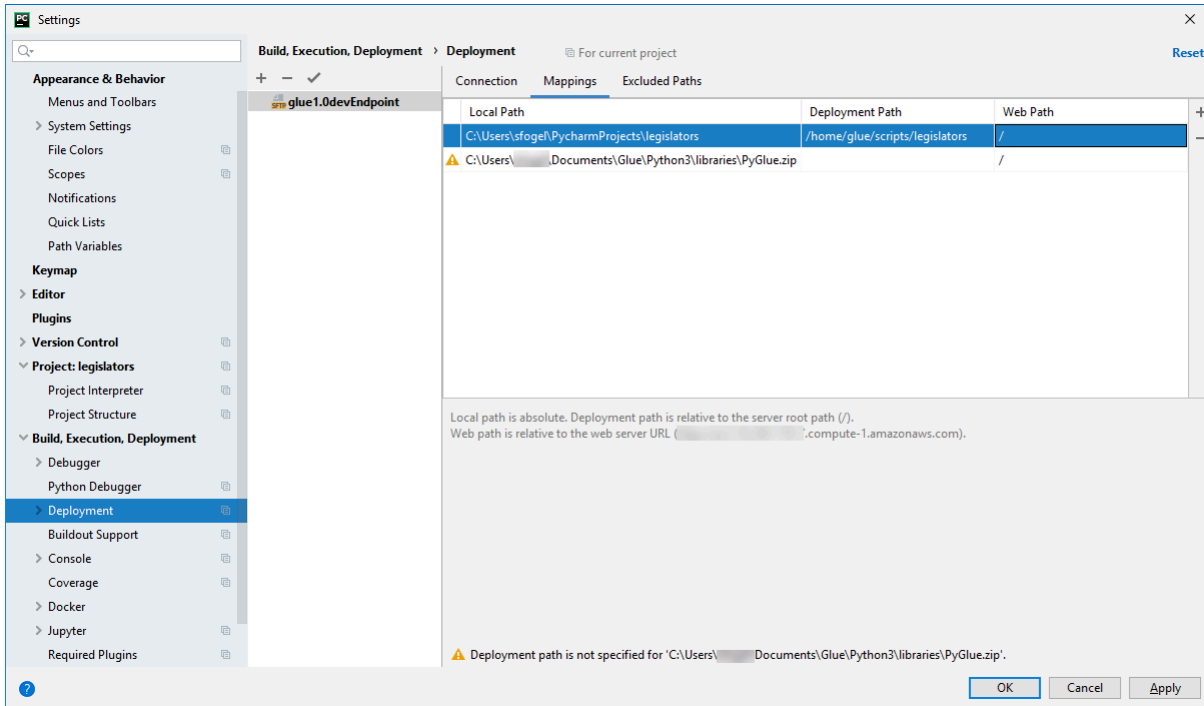


다시 한 번, [적용(Apply)]을 선택한 다음 [설정(Settings)] 대화 상자를 열어 둡니다.

6. 로컬 디렉터리를 원격 디렉터리로 매핑하여 개발합니다.

- [Deployment(배치)] 페이지의 오른쪽 화면에서 [Mappings(매핑)]이라는 윗쪽 중간 탭을 선택합니다.
- [Deployment Path(배치 경로)] 열에서 `/home/glue/scripts/`의 경로를 입력하여 프로젝트 경로를 배치합니다. 예: `/home/glue/scripts/legislators`.
- 적용을 선택합니다.

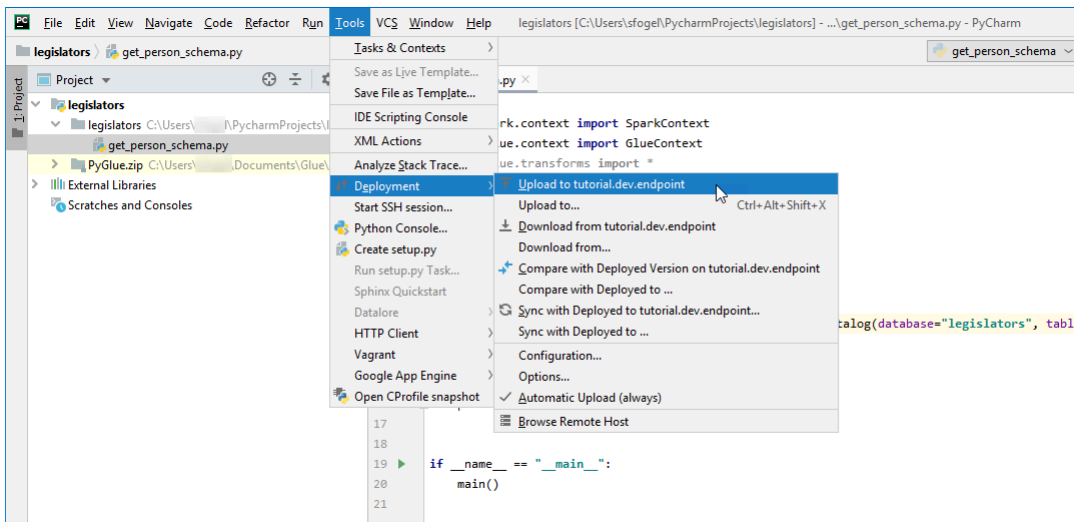
[Settings(설정)] 스크린은 이제 다음과 비슷해야 합니다.



[OK(확인)]를 선택하여 [Settings(설정)] 대화 상자를 닫습니다.

스크린을 개발 엔드포인트에 배포

1. 다음 그림과 같이 [Tools(도구)], [Deployment(배포)]를 선택한 다음 개발 엔드포인트를 설정할 이름을 선택합니다.



스크린이 배치된 후 스크린 아래는 다음과 유사하게 보입니다.


```

File Transfer: tutorial.dev.endpoint x
[12/11/2019 5:16 PM] Upload to tutorial.dev.endpoint
[12/11/2019 5:16 PM] Upload file 'C:\Users\... \PycharmProjects\legislators\get_person_schema.py' to '/home/glue/scripts/legislators/get_person_schema.py'
[12/11/2019 5:16 PM] Upload to tutorial.dev.endpoint completed in 782 ms: 1 file transferred (1.1 kbit/s)

Upload to tutorial.dev.endpoint completed: 1 file transferred
  
```

2. 메뉴 모음에서 [Tools(도구)], [Deployment(배포)], [Automatic Upload (always)(자동 업데이트(항상))]를 선택합니다. [Automatic Upload (always)(자동 업데이트(항상))] 옆에 확인 표시가 나타나는지 확인합니다.

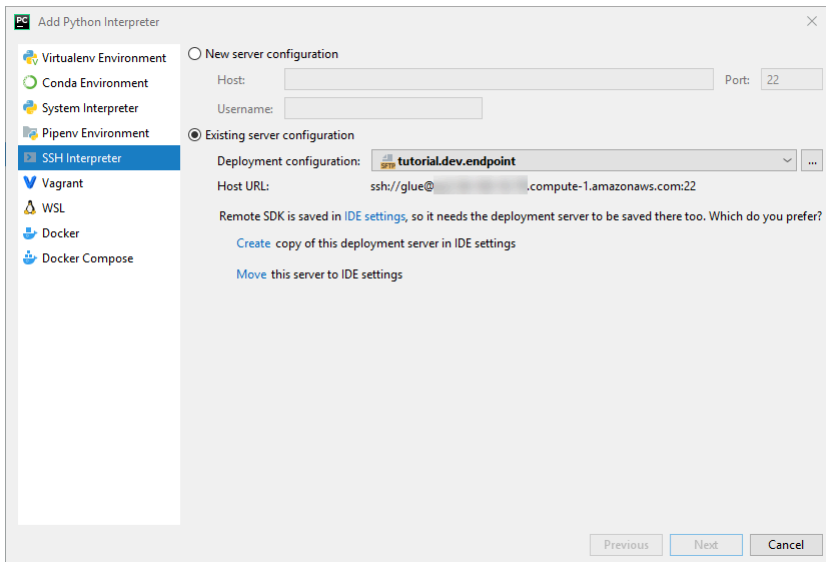
이 옵션을 활성화하면 PyCharm은 변경된 파일을 개발 엔드포인트에 자동으로 업로드합니다.

원격 인터프리터 구성

개발 엔드포인트에서 Python 인터프리터를 사용하도록 PyCharm을 구성합니다.

1. [File(파일)] 메뉴에서 [Settings(설정)]를 선택합니다.
2. 프로젝트 [legislators(제정자)]를 확장하고 [Project Interpreter(프로젝트 인터프리터)]를 선택합니다.
3. [Project Interpreter(프로젝트 인터프리터)] 목록 옆에 있는 톱니바퀴 아이콘을 선택한 다음 [Add(추가)]를 선택합니다.
4. [Add Python Interpreter(Python 인터프리터 추가)] 대화 상자의 왼쪽 창에서 [SSH Interpreter(SSH 인터프리터)]를 선택합니다.
5. [Existing server configuration(기존 서버 구성)]을 선택하고 [Deployment configuration(배포 구성)] 목록에서 구성을 선택합니다.

스크린은 다음과 비슷해야 합니다.



6. [Move this server to IDE settings(이 서버를 IDE 설정으로 이동)]를 선택하고 [Next(다음)]를 선택합니다.
7. [Interpreter(인터프리터)] 필드에서 Python 2를 사용하는 경우 경로를 `/usr/bin/gluepython`으로 변경하고 Python 3을 사용하는 경우 `/usr/bin/gluepython3`로 변경합니다. 그런 다음 Finish(완료)를 선택합니다.

스크립트를 개발 엔드포인트에서 실행

다음과 같이 스크립트를 실행합니다.

- 왼쪽 창에서 파일 이름을 마우스 오른쪽 버튼으로 클릭하고 [Run '**<filename>**'(<filename> 실행)]을 선택합니다.

일련의 메시지 후에 최종 출력에는 개수와 스키마가 표시되어야 합니다.

```
Count: 1961
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
```

```

|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string

```

Process finished with exit code 0

그런 다음 원격으로 개발 엔드포인트에 스크립트를 디버깅하도록 설정합니다.

고급 구성: 여러 사용자 간에 개발 엔드포인트 공유

이 섹션에서는 일반적인 사용 사례에서 SageMaker 노트북을 사용해 개발 엔드포인트를 유리하게 활용하여 여러 사용자 간에 개발 엔드포인트를 공유하는 법을 설명합니다.

단일 테넌시 구성

단일 테넌트 사용 사례의 경우, 개발자 환경을 간소화하고 리소스 경합을 피하려면 각각의 개발자가 자신이 작업하는 프로젝트에 맞춰 크기가 조정된 자기만의 개발 엔드포인트를 사용하도록 하는 것이 좋습니다. 이렇게 하면 작업자 유형 및 DPU 수와 관련된 결정도 간소화해줍니다. 개발자의 재량 및 개발자가 작업 중인 프로젝트에 따라 달라지도록 두기 때문입니다.

여러 노트북 파일을 동시에 실행하지 않는 한, 리소스 할당을 직접 처리하지 않아도 됩니다. 동시에 여러 노트북 파일에서 코드를 실행하는 경우, 여러 개의 Livy 세션을 동시에 실행해야 합니다. 여러 개의 Livy 세션을 동시에 실행하기 위해 Spark 클러스터 구성을 분리하려면 다중 테넌트 사용 사례에 소개된 단계를 따르면 됩니다.

예를 들어 개발 엔드포인트에 작업자가 10개이고 작업자 유형이 G.1X인 경우, Spark 실행기는 9개이고 클러스터 전체에는 실행기 메모리가 90G가 됩니다. 실행기마다 메모리가 10G씩이 때문입니다.

지정된 작업자 유형과 관계없이 Spark 동적 리소스 할당이 활성화됩니다. 데이터 집합이 충분히 크면 Spark가 모든 실행기를 하나의 Livy 세션에 할당할 가능성이 있습니다. spark.dynamicAllocation.maxExecutors가 기본적으로 설정되어 있지 않기 때문입니다. 이는 같은 개발 엔드포인트상의 다른 Livy 세션은 새 실행기 실행을 대기하게 된다는 뜻입니다. 데이터 집합 크기가 작은 경우 Spark가 실행기를 동시에 여러 Livy 세션에 할당할 수 있습니다.

Note

다양한 사용 사례에서 리소스 할당 방식을 자세히 알아보고 동작을 수정하기 위해 구성을 변경하는 법을 알아보려면 [고급 구성: 여러 사용자 간에 개발 엔드포인트 공유](#) 섹션을 참조하세요.

다중 테넌시 구성

Note

개발 엔드포인트는 AWS Glue ETL 환경에 단일 테넌트 환경으로 에뮬레이션되는 경향이 있다는 점을 알아두시기 바랍니다. 다중 테넌트 사용도 가능하기는 하지만, 이것은 고급 사용 사례이며 대부분의 사용자는 각각의 개발 엔드포인트에 단일 테넌시 패턴을 유지하는 것이 좋습니다.

다중 테넌트 사용 사례에서는 사용자가 리소스 할당을 직접 처리해야 할 가능성이 있습니다. 핵심 요인은 Jupyter Notebook을 동시에 사용하는 동시 사용자 수입니다. 'FTS(follow-the-sun)' 워크플로를 따르는 팀이고 각 시간대에 Jupyter 사용자가 한 명뿐인 경우, 동시 사용자 수가 하나뿐이므로 리소스 할당에 신경 쓰지 않아도 됩니다. 다만 노트북을 여러 사용자가 공유하고 각각의 사용자가 애드혹 방식으로 코드를 제출하는 경우라면, 아래와 같은 요점을 고려해야 할 수 있습니다.

Spark 클러스터 리소스를 여러 사용자 사이에서 파티셔닝하려면 SparkMagic 구성을 사용하면 됩니다. SparkMagic을 구성하는 방법은 두 가지입니다.

(A) %%configure -f 지시어 사용

노트북의 Livy 세션당 구성을 수정하고자 하는 경우, 노트북 단락에서 %%configure -f 지시어를 실행하면 됩니다.

예를 들어 5개의 실행기에서 Spark 애플리케이션을 실행하고자 하는 경우, 노트북 단락에서 다음과 같은 명령을 실행하면 됩니다.

```
%%configure -f
{"numExecutors":5}
```

그러면 Spark UI에 해당 작업에 대하여 실행기 5개만 실행되는 것을 확인할 수 있습니다.

동적인 리소스 할당을 위해서는 실행기 최대 수를 제한하는 것이 좋습니다.

```
%%configure -f
{"conf":{"spark.dynamicAllocation.maxExecutors":"5"}}
```

(B) SparkMagic Config 파일 수정

SparkMagic은 [Livy API](#)를 기반으로 작동합니다. SparkMagic은 driverMemory, driverCores, executorMemory, executorCores, numExecutors, conf 등의 구성으로 Livy 세션을 생성합니다. 이런 것이 Spark 클러스터 전체에서 리소스가 얼마나 사용되는지 판단하는 핵심 요인입니다. SparkMagic을 사용하면 Livy로 보내지는 그러한 파라미터를 지정할 Config 파일을 제공할 수 있습니다. 샘플 Config 파일은 이 [Github 리포지토리](#)에서 확인할 수 있습니다.

한 노트북의 모든 Livy 세션에 걸쳐 구성을 수정하고자 하는 경우, /home/ec2-user/.sparkmagic/config.json을 수정하여 session_config를 추가하도록 하면 됩니다.

SageMaker 노트북 인스턴스에서 Config 파일을 수정하려면 다음과 같은 단계를 따르면 됩니다.

1. SageMaker 노트북을 엽니다.
2. 터미널 커널을 엽니다.
3. 다음 명령을 실행합니다.

```
sh-4.2$ cd .sparkmagic
sh-4.2$ ls
config.json logs
sh-4.2$ sudo vim config.json
```

예를 들어 이러한 줄을 /home/ec2-user/.sparkmagic/config.json에 추가한 다음 노트북에서 Jupyter 커널을 다시 시작하면 됩니다.

```
"session_configs": {
```

```
"conf": {
  "spark.dynamicAllocation.maxExecutors": "5"
},
```

지침 및 모범 사례

이런 종류의 리소스 충돌을 피하려면 다음과 같은 몇 가지 기본적인 접근법을 쓰면 됩니다.

- NumberOfWorkers(가로 방향으로 확장)를 늘리고 workerType(세로 방향으로 확장)을 업그레이드하여 Spark 클러스터 크기 키우기
- 사용자당 할당하는 리소스 수 줄이기(Livy 세션당 리소스 수 줄이기)

접근법은 사용 사례에 따라 다릅니다. 개발 엔드포인트 크기가 크고 데이터 양이 많지 않은 경우, Spark가 동적 할당 전략을 기반으로 리소스를 할당할 수 있기 때문에 리소스 충돌이 발생할 가능성이 대폭 감소합니다.

위에서 설명한 것과 같이, Spark 실행기 수는 DPU(또는 NumberOfWorkers)와 작업자 유형을 조합하여 자동으로 계산됩니다. 각각의 Spark 애플리케이션이 드라이버 하나와 실행기 여러 개를 실행합니다. 계산하려면 $\text{NumberOfWorkers} = \text{NumberOfExecutors} + 1$ 이어야 합니다. 아래 매트릭스에서는 동시 사용자 수를 바탕으로 개발 엔드포인트에 필요한 용량을 설명합니다.

동시 노트북 사용자 수	사용자당 할당하고자 하는 Spark 실행기 수	개발 엔드포인트의 NumberOfWorkers 총합
3	5	18
10	5	60
50	5	300

사용자당 적은 리소스를 할당하고자 하는 경우, Livy 세션 파라미터로 구성할 가장 간편한 파라미터는 `spark.dynamicAllocation.maxExecutors`(또는 `numExecutors`)일 것입니다. `/home/ec2-user/.sparkmagic/config.json`에서 아래 구성을 설정하는 경우, SparkMagic이 Livy 세션당 최대 5개의 실행기를 할당하게 됩니다. 이렇게 하면 Livy 세션당 리소스 분리에 도움이 됩니다.

```
"session_configs": {
```

```
"conf": {
  "spark.dynamicAllocation.maxExecutors": "5"
},
```

작업자가 18명인 개발 엔드포인트가 하나 있고(G.1X) 동시에 3명의 동시 노트북 사용자가 있다고 가정합니다. 세션 구성에 `spark.dynamicAllocation.maxExecutors=5`가 포함된 경우, 각 사용자가 드라이버 1개, 실행기 5개를 사용할 수 있습니다. 동시에 여러 노트북 단락을 실행한다 해도 리소스 충돌이 발생하지 않습니다.

절충

이 세션 구성 "`spark.dynamicAllocation.maxExecutors": "5"`를 사용하면 리소스 충돌 오류를 피할 수 있으며 동시 사용자 액세스가 있는 경우 리소스 할당을 기다리지 않아도 됩니다. 다만 여유 리소스가 많이 있다 하더라도(예를 들어 다른 동시 사용자가 없는 경우) Spark는 Livy 세션에 실행기를 5개 이상 할당할 수 없습니다.

기타 참고 사항

노트북 사용을 중지할 때 Jupyter 커널을 중지하는 것이 좋습니다. 이렇게 하면 여유 리소스가 확보되어 다른 노트북 사용자가 커널 만료(자동 종료)를 기다릴 필요 없이 즉시 해당 리소스를 사용할 수 있습니다.

일반적인 문제

지침을 따라도 몇 가지 문제가 발생할 수 있습니다.

세션을 찾을 수 없음

Livy 세션이 이미 종료되었는데도 노트북 단락을 실행하려 시도하면 아래와 같은 메시지가 표시됩니다. Livy 세션을 활성화하려면 Jupyter 메뉴에서 [커널(Kernel)] > [다시 시작(Restart)]을 선택하여 Jupyter 커널을 다시 시작해야 합니다.

```
An error was encountered:
Invalid status code '404' from http://localhost:8998/sessions/13 with error payload:
"Session '13' not found."
```

YARN 리소스 부족

Spark 클러스터에 새 Livy 세션을 시작하기에 충분한 리소스가 없는데도 노트북 단락을 실행하려 시도하는 경우, 아래와 같은 메시지가 표시됩니다. 이 문제는 지침을 따르면 피할 수 있을 때가 많지만, 그래

도 이 문제에 직면할 가능성이 있습니다. 이 문제를 피하려면 불필요한 활성 Livy 세션이 있는지 확인합니다. 불필요한 Livy 세션이 있는 경우, 이를 종료하면 여유 클러스터 리소스를 확보할 수 있습니다. 자세한 내용은 다음 단원을 참조하십시오.

```
Warning: The Spark session does not have enough YARN resources to start.
The code failed because of a fatal error:
    Session 16 did not start up in 60 seconds..
```

Some things to try:

- a) Make sure Spark has enough available resources for Jupyter to create a Spark context.
- b) Contact your Jupyter administrator to make sure the Spark magics library is configured correctly.
- c) Restart the kernel.

모니터링 및 디버깅

이 섹션에서는 리소스와 세션을 모니터링하는 기법을 설명합니다.

클러스터 리소스 할당 모니터링 및 디버깅

Spark UI를 보면 Livy 세션당 할당된 리소스 양을 모니터링하여 해당 작업에서 효과적인 Spark 구성이 무엇인지 알아볼 수 있습니다. Spark UI를 활성화하려면 [개발 엔드포인트에 Apache Spark 웹 UI 사용을 참조하세요](#).

(선택 사항) Spark UI를 실시간으로 보려면 Spark 클러스터에서 실행 중인 Spark 기록 서버에 대하여 SSH 터널을 구성하면 됩니다.

```
ssh -i <private-key.pem> -N -L 8157:<development endpoint public address>:18080
glue@<development endpoint public address>
```

그러면 브라우저에서 <http://localhost:8157>을 열어 Spark UI를 확인할 수 있습니다.

사용 가능한 불필요한 Livy 세션

노트북이나 Spark 클러스터에서 불필요한 Livy 세션을 모두 종료하는 다음과 같은 절차를 검토하세요.

(a). 노트북에서 Livy 세션 종료

Jupyter Notebook에서 커널을 종료하여 불필요한 Livy 세션을 종료할 수 있습니다.

(b). Spark 클러스터에서 Livy 세션 종료

아직 실행 중인 불필요한 Livy 세션이 있는 경우, Spark 클러스터에서 해당 Livy 세션을 종료하면 됩니다.

이 절차를 수행하기 위한 전제 조건으로, 개발 엔드포인트에 대하여 SSH 퍼블릭 키를 구성해야 합니다.

Spark 클러스터에 로그인하려면 다음과 같은 명령을 실행하면 됩니다.

```
$ ssh -i <private-key.pem> glue@<development endpoint public address>
```

다음 명령을 실행하여 활성 Livy 세션을 확인합니다.

```
$ yarn application -list
20/09/25 06:22:21 INFO client.RMPProxy: Connecting to ResourceManager at
ip-255-1-106-206.ec2.internal/172.38.106.206:8032
Total number of applications (application-types: [] and states: [SUBMITTED, ACCEPTED,
RUNNING]):2
Application-Id Application-Name Application-Type User Queue State Final-State Progress
Tracking-URL
application_1601003432160_0005 livy-session-4 SPARK livy default RUNNING UNDEFINED 10%
http://ip-255-1-4-130.ec2.internal:41867
application_1601003432160_0004 livy-session-3 SPARK livy default RUNNING UNDEFINED 10%
http://ip-255-1-179-185.ec2.internal:33727
```

그리고 다음 명령으로 Livy 세션을 종료하면 됩니다.

```
$ yarn application -kill application_1601003432160_0005
20/09/25 06:23:38 INFO client.RMPProxy: Connecting to ResourceManager at
ip-255-1-106-206.ec2.internal/255.1.106.206:8032
Killing application application_1601003432160_0005
20/09/25 06:23:39 INFO impl.YarnClientImpl: Killed application
application_1601003432160_0005
```

노트북 관리

Note

개발 엔드포인트는 AWS Glue의 2.0 이전 버전에서만 지원됩니다. ETL 스크립트를 작성하고 테스트할 수 있는 대화형 환경에서는 [AWS Glue Studio에서 노트북](#)을 사용합니다.

노트북을 사용하면 개발 엔드포인트에서 추출, 전환 및 적재(ETL) 스크립트를 대화식으로 개발 및 테스트할 수 있습니다. AWS Glue는 SageMaker Jupyter 노트북에 대한 인터페이스를 제공합니다. AWS Glue를 사용하여 SageMaker 노트북을 생성하고 관리합니다. AWS Glue 콘솔에서도 SageMaker 노트북을 열 수 있습니다.

또한 Apache Spark를 SageMaker를 지원하는(AWS Glue ETL 작업은 아니고) AWS Glue 개발 엔드포인트에서 SageMaker와 함께 사용해도 됩니다. SageMaker Spark는 SageMaker용 오픈 소스 Apache Spark 라이브러리입니다. 자세한 내용을 알아보려면 [Using Apache Spark with Amazon SageMaker](#)(Amazon SageMaker에서 Apache Spark 사용)를 참조하세요.

⚠ Important

AWS Glue 개발 엔드포인트로 SageMaker 노트북 관리는 다음 AWS 리전에서 사용할 수 있습니다.

지역	코드
미국 동부(오하이오)	us-east-2
미국 동부(버지니아 북부)	us-east-1
미국 서부(캘리포니아 북부)	us-west-1
미국 서부(오레곤)	us-west-2
아시아 태평양(도쿄)	ap-northeast-1
아시아 태평양(서울)	ap-northeast-2
아시아 태평양(뭄바이)	ap-south-1
아시아 태평양(싱가포르)	ap-southeast-1
아시아 태평양(시드니)	ap-southeast-2
캐나다(중부)	ca-central-1
유럽(프랑크푸르트)	eu-central-1
유럽(아일랜드)	eu-west-1

지역	코드
유럽(런던)	eu-west-2

AWS Glue Studio를 사용하여 시각적 ETL 작업 구축

AWS Glue 작업은 소스 데이터에 연결하여 처리한 다음 데이터 대상에 작성하는 스크립트를 캡슐화합니다. 일반적으로 작업은 추출, 변환 및 로드(ETL) 스크립트를 실행합니다. 작업은 Apache Spark 및 Ray 런타임 환경용으로 설계된 스크립트를 실행할 수 있습니다. 작업은 범용 Python 스크립트(Python 셸 작업)를 실행할 수도 있습니다. AWS Glue 트리거는 일정 또는 이벤트에 따라 또는 필요에 따라 작업을 시작할 수 있습니다. 작업 실행을 모니터링하여 완료 상태, 지속 시간, 시작 시간 같은 실행 시간 지표를 이해할 수 있습니다.

AWS Glue에서 생성하는 스크립트를 사용하거나 직접 제공할 수 있습니다. 소스 스키마와 대상 위치 또는 스키마가 있을 경우 AWS Glue Studio 코드 생성기는 Apache Spark API(PySpark) 스크립트를 자동적으로 생성할 수 있습니다. 이 스크립트를 시작 포인트로 사용할 수 있고 목적에 부합하기 위해 편집할 수도 있습니다.

AWS Glue에서는 여러 데이터 형식으로 출력 파일을 작성할 수 있습니다. 작업 유형마다 지원하는 출력 형식이 다를 수 있습니다. 몇 가지 데이터 포맷의 경우, 일반 압축 포맷이 작성될 수 있습니다.

AWS Glue 콘솔로 로그인합니다

AWS Glue 작업은 추출, 변환 및 로드(ETL) 작업을 수행하는 데 필요한 비즈니스 로직으로 구성됩니다. AWS Glue 콘솔의 [ETL] 섹션에서 작업을 생성할 수 있습니다.

기존 작업을 보려면 AWS Management Console에 로그인하고 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다. 그런 다음 AWS Glue에서 [작업(Jobs)] 탭을 선택합니다. 작업이 마지막으로 수정되고 현재 작업이 옵션을 표시할 경우, [Jobs(작업)] 목록은 각 작업과 관련된 스크립트 위치를 보여줍니다.

새 작업을 생성하는 동안 또는 작업을 저장한 후 AWS Glue Studio를 사용하여 ETL 작업을 수정할 수 있습니다. 시각적 편집기에서 노드를 편집하거나 개발자 모드에서 작업 스크립트를 편집하여 이를 수행할 수 있습니다. 시각적 편집기에서 노드를 추가하고 제거하여 더 복잡한 ETL 작업을 생성할 수도 있습니다.

AWS Glue Studio에서 작업을 생성하기 위한 다음 단계

시각적 작업 편집기를 사용하여 작업에 대한 노드를 구성합니다. 각 노드는 소스 위치에서 데이터 읽기 또는 데이터에 변환 적용과 같은 작업을 나타냅니다. 작업에 추가하는 각 노드에는 데이터 위치 또는 변환에 대한 정보를 제공하는 속성이 있습니다.

다음은 작업을 생성하고 관리하는 단계입니다.

- [AWS Glue Studio을\(를\) 사용한 시각적 ETL](#)
- [작업 스크립트 보기](#)
- [작업 속성 수정](#)
- [작업 저장](#)
- [작업 실행 시작](#)
- [최근 작업 실행에 대한 정보 보기](#)
- [작업 모니터링 대시보드에 액세스](#)

AWS Glue Studio을(를) 사용한 시각적 ETL

AWS Glue Studio의 간단한 시각적 인터페이스를 사용하여 ETL 작업을 생성할 수 있습니다. [작업 (Jobs)] 페이지를 사용하여 새 작업을 생성합니다. 스크립트 편집기 또는 노트북을 사용하여 AWS Glue Studio ETL 작업 스크립트의 코드로 직접 작업할 수도 있습니다.

작업(Jobs) 페이지에서 AWS Glue Studio 또는 AWS Glue로 생성한 모든 작업을 볼 수 있습니다. 이 페이지에서 작업을 보고, 관리하고, 실행할 수 있습니다.

AWS Glue Studio에서 ETL 작업을 생성하는 방법에 대한 또 다른 예제는 [블로그 자습서](#)도 참조하세요.

AWS Glue Studio에서 작업 시작

AWS Glue에서는 시각적 인터페이스, 대화형 코드 노트북 또는 스크립트 편집기를 통해 작업을 생성할 수 있습니다. 원하는 옵션을 클릭하여 작업을 시작하거나 샘플 작업을 기반으로 새 작업을 생성할 수 있습니다.

선택한 도구를 사용하여 샘플 작업에서 작업을 생성합니다. 예를 들어 샘플 작업을 사용하면 CSV 파일을 카탈로그 테이블에 결합하는 시각적 ETL 작업을 생성하거나 pandas에 대한 작업을 수행할 때 AWS Glue for Ray 또는 AWS Glue for Spark에서 대화형 코드 노트북에서 작업을 생성하거나 SparkSQL을 사용하여 대화형 코드 노트북에서 작업을 생성할 수 있습니다.

처음부터 AWS Glue Studio에서 작업 생성

1. [AWS Management Console](#) 로그인하고 <https://console.aws.amazon.com/gluestudio/> 에서 AWS Glue Studio 콘솔을 엽니다.
2. 탐색 창에서 ETL 작업을 선택합니다.

3. 작업 생성 섹션에서 작업에 대한 구성 옵션을 선택합니다.

The screenshot shows the 'Create job' interface in AWS Glue Studio. It is divided into two main sections: 'Create job' and 'Example jobs'.

Create job section:

- Visual ETL:** Author in a visual interface focused on data flow. (Highlighted with an orange background)
- Notebook:** Author using an interactive code notebook.
- Script editor:** Author code with a script editor.

Example jobs section (with a 'Create sample job' button):

- Visual ETL job with a join:** Read three CSV files, combine the data, change the data types, then write the data to Amazon S3 and catalog it for querying later.
- Ray sample notebook - New:** Use the Ray framework for parallel processing in Python. Read many parquet files from S3, explore and filter the data, then save it to a CSV.
- Spark Pandas sample notebook:** Explore and visualize data using the popular Pandas framework combined with Spark.
- Spark SQL sample notebook:** Use SQL to get started quickly with Apache Spark. Access data via the AWS Glue Data Catalog and transform it using familiar commands.

처음부터 새로 작업을 생성하는 옵션:

- 시각적 ETL - 데이터 흐름에 초점을 맞춘 시각적 인터페이스에서 작성
- 대화형 코드 노트북을 사용하여 작성 - Jupyter Notebook 기반의 노트북 인터페이스에서 대화형으로 작업 작성

이 옵션을 선택하는 경우 노트북 작성 세션을 생성하기 전에 추가 정보를 제공해야 합니다. 이 정보를 지정하는 방법에 대한 자세한 내용은 [AWS Glue Studio에서 노트북 시작하기](#) 섹션을 참조하세요.

- 스크립트 편집기에서 코드 작성 - ETL 스크립트 작성 및 프로그래밍에 익숙한 사용자의 경우 이 옵션을 선택하여 새 Spark ETL 작업을 생성합니다. 엔진(Python 셸, Ray, Spark(Python) 또는 Spark(Scala))를 선택합니다. 그런 다음 새로 시작 또는 스크립트 업로드를 선택합니다. 그러면 로컬 파일에서 기존 스크립트를 업로드합니다. 스크립트 편집기를 사용하는 옵션을 선택한 경우 시각적 작업 편집기를 사용하여 작업을 설계하거나 편집할 수 없습니다.

Spark 작업은 AWS Glue에서 관리하는 Apache Spark 환경에서 실행됩니다. 기본적으로 새 스크립트는 Python으로 코딩됩니다. 새 Scala 스크립트를 작성하려면 [AWS Glue Studio에서 Scala 스크립트 생성 및 편집](#) 섹션을 참조하세요.

예제 작업으로부터 AWS Glue Studio에서 작업 생성

예제 작업에서 작업을 생성하도록 선택할 수 있습니다. 예제 작업 섹션에서 샘플 작업을 선택한 다음 샘플 작업 생성을 선택합니다. 옵션 중 하나에서 샘플 작업을 생성하면 작업할 수 있는 빠른 템플릿이 제공됩니다.

1. <https://console.aws.amazon.com/gluestudio/> 에서 AWS Management Console 로그인하고 AWS Glue Studio 콘솔을 엽니다.
2. 탐색 창에서 ETL 작업을 선택합니다.
3. 샘플 작업에서 작업을 생성하는 옵션을 선택합니다.
 - 여러 소스를 결합하는 시각적 ETL 작업 - 세 개의 CSV 파일을 읽고, 데이터를 결합한 후 데이터 형식을 변경하고 Amazon S3에 데이터를 쓴 후에 나중에 쿼리할 수 있도록 카탈로그화합니다.
 - Pandas를 사용하는 Spark 노트북 - Spark와 결합된 인기 있는 Pandas 프레임워크를 사용하여 데이터를 탐색하고 시각화합니다.
 - SQL을 사용하는 Spark 노트북 - SQL을 사용하여 Apache Spark를 빠르게 시작할 수 있습니다. AWS Glue 데이터 카탈로그를 통해 데이터에 액세스하고 친숙한 명령을 사용하여 데이터를 변환합니다.
4. 샘플 작업 생성을 선택합니다.

작업 편집기 기능

작업 편집기는 작업 생성 및 편집을 위해 다음과 같은 기능을 제공합니다.

- 각 작업 태스크에 대한 노드가 있는 작업의 시각적 다이어그램: 데이터를 읽기 위한 데이터 원본 노드, 데이터를 수정하기 위한 변환 노드, 데이터를 쓰기 위한 데이터 대상 노드.
- 작업 다이어그램에서 각 노드의 속성을 보고 구성할 수 있습니다. 작업 다이어그램에서 각 노드에 대한 스키마 및 샘플 데이터를 볼 수도 있습니다. 이러한 기능을 사용하면 작업을 실행할 필요 없이 작업이 올바른 방식으로 데이터를 수정하고 변환하는지 확인할 수 있습니다.
- 작업에 대해 생성된 코드를 수정할 수 있는 스크립트 보기 및 편집 탭.
 - AWS Glue ETL 작업이 실행되는 환경을 사용자 정의하기 위해 다양한 설정을 구성할 수 있는 작업 세부 정보 탭.
 - 작업의 현재 및 이전 실행을 보고, 작업 실행 상태를 보고, 작업 실행에 대한 로그에 액세스할 수 있는 실행 탭.
 - 작업에 데이터 품질 규칙을 적용할 수 있는 데이터 품질 탭.
 - 작업 시작 시간을 구성하거나 반복 작업 실행을 설정할 수 있는 일정 탭.
 - 작업에 사용할 Git 서비스를 구성할 수 있는 버전 제어 탭.

시각적 작업 편집기에서 스키마 미리 보기 사용

작업을 생성하거나 편집하는 동안 [출력 스키마(Output schema)] 탭을 사용하여 데이터에 대한 스키마를 볼 수 있습니다.

스키마를 보려면 먼저 작업 편집기에 데이터 원본에 액세스할 수 있는 권한이 필요합니다. 편집기의 작업 세부 정보 탭이나 노드의 [출력 스키마(Output schema)] 탭에서 IAM 역할을 지정할 수 있습니다. IAM 역할에 데이터 원본에 액세스하는 데 필요한 모든 권한이 있는 경우 노드의 [출력 스키마(Output schema)] 탭에서 스키마를 볼 수 있습니다.

시각적 작업 편집기에서 데이터 미리 보기 사용

데이터 미리 보기를 사용하면 작업을 반복적으로 실행할 필요 없이 데이터 샘플을 이용하여 작업을 생성하고 테스트할 수 있습니다. 데이터 미리 보기를 사용하여 다음을 수행할 수 있습니다.

- IAM 역할을 테스트하여 데이터 소스 또는 데이터 대상에 대한 액세스 권한이 있는지 확인할 수 있습니다.
- 변환이 의도한 방식으로 데이터를 수정하고 있는지 확인할 수 있습니다. 예를 들어 필터 변환을 사용하는 경우 필터가 올바른 데이터 하위 집합을 선택하는지 확인할 수 있습니다.
- 데이터를 확인하십시오. 데이터 집합에 여러 유형의 값이 있는 열이 포함된 경우 데이터 미리 보기에 이러한 열에 대한 튜플 목록이 표시됩니다. 각 튜플에는 데이터 유형과 해당 값이 포함됩니다.

작업을 생성하거나 편집하는 동안 작업 캔버스 아래에 있는 데이터 미리 보기 탭을 사용하여 데이터 샘플을 볼 수 있습니다. 작업에 역할이 이미 구성되어 있거나 계정에 기본 IAM 역할이 설정된 경우 새 데이터 미리 보기 세션이 자동으로 시작됩니다. 역할이 이전에 구성되지 않은 경우 역할을 선택하여 세션을 시작할 수 있습니다.

Data preview | **Output schema** ☰ ☰

Start a data preview session

IAM role
To start a data preview session, choose an IAM role for this job. Changing the role will end an existing data preview session.

Admin ▼
No description available.

[Create IAM role.](#)

▶ **Additional Settings**

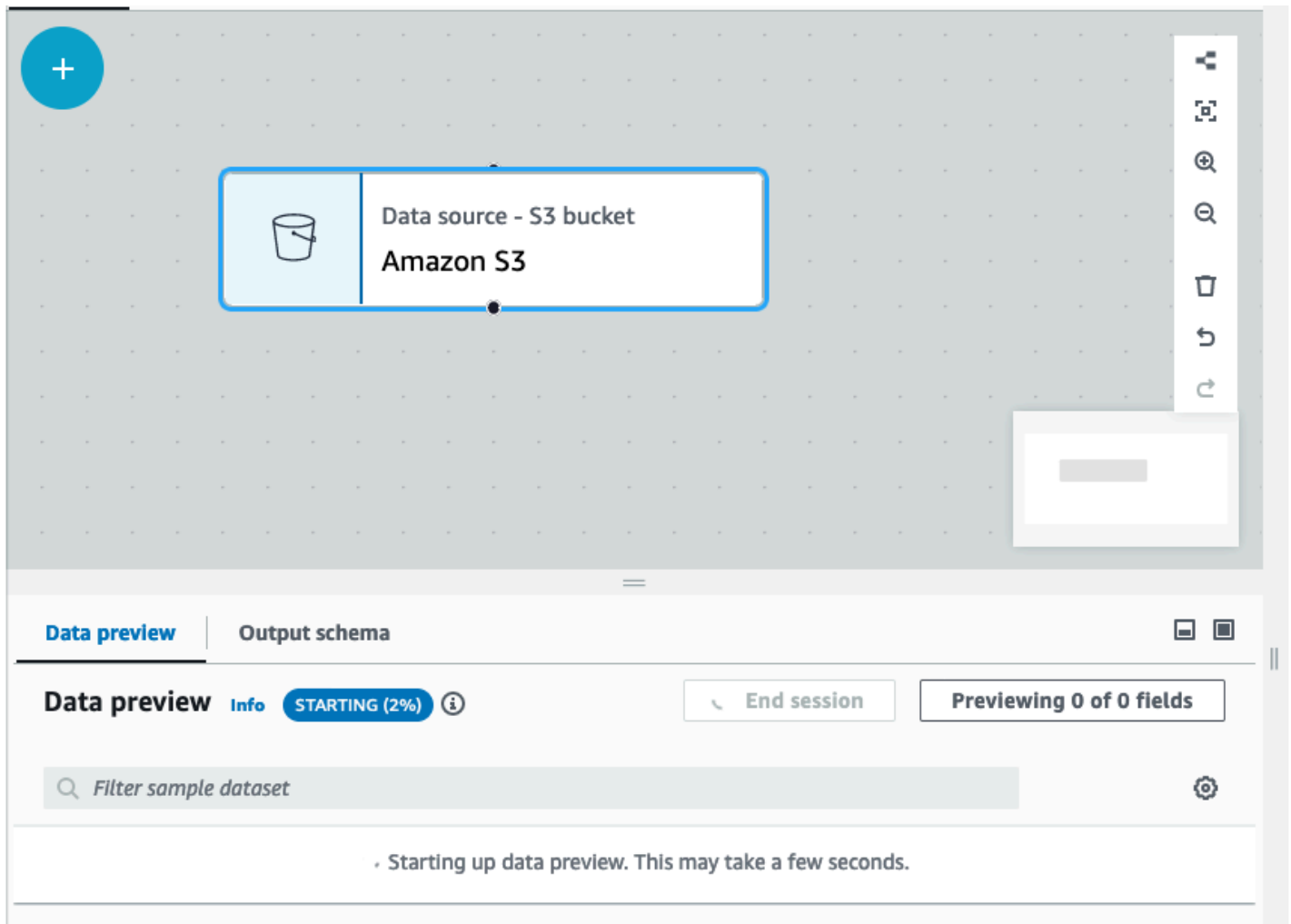
Start session

Note

데이터 미리 보기 세션에서 선택한 역할이 작업에도 사용됩니다.

정보 아이콘을 클릭하면 세션 상태와 진행 상황, 세션 세부 정보를 볼 수 있습니다.

세션이 준비되면 AWS Glue Studio가 선택한 노드에 대한 데이터를 로드합니다. 진행 상황에 따라 완료율 %을 확인할 수 있습니다.



시각적 작업을 작성할 때 출력 스키마 탭에서 세션에서 스키마 추론을 전환하면 선택한 노드의 스키마를 AWS Glue Studio가 자동으로 업데이트합니다.

The screenshot shows the AWS Glue console interface for configuring a 'Transform - SQL Query' node. The 'Input sources' section is set to 'Amazon S3' with an alias 'myDataSource'. The 'SQL query' section contains the query: 'select firstname, lastname, title from myDataSource'. Below the query, the 'Output schema' is displayed as a table with columns 'firstname', 'lastname', and 'title', all of type 'string'.

Key	Data type
firstname	string
lastname	string
title	string

데이터 미리 보기 기본 설정을 구성하는 방법:

설정 아이콘(기어 기호)을 선택하여 데이터 미리 보기에 대한 기본 설정을 구성합니다. 이러한 설정은 작업 다이어그램의 모든 노드에 적용됩니다. 다음을 할 수 있습니다.

- 한 줄에서 다음 줄로 텍스트를 줄 바꿈하도록 선택합니다. 이 옵션은 기본적으로 활성화되어 있습니다
- 행 수 변경(기본값 200개)
- 필요한 경우 IAM 역할을 선택하거나 IAM 역할을 생성합니다
- 작업을 작성할 때 새 세션을 자동으로 시작하도록 선택합니다. 이렇게 하면 작업을 작성할 때 새 대화형 세션이 프로비저닝됩니다. 이 설정은 계정 수준에서 적용됩니다. 일단 설정하면 작업을 편집할 때 계정의 모든 사용자에게 적용됩니다.
- 스키마를 자동으로 유추하도록 선택합니다. 선택한 노드에 대해 출력 스키마가 자동으로 추론됩니다
- AWS Glue 라이브러리를 자동으로 가져오도록 선택합니다. 이는 세션을 다시 시작해야 하는 새 변환을 추가할 때 데이터 미리 보기에서 새 세션이 다시 시작되지 않도록 하므로 유용합니다.

Preferences ✕

Wrap lines
Enable to wrap lines of table cell content, disable to truncate text.

Number of rows
Enter the amount of entries to sample from the dataset.

200

IAM role
To start a data preview session, choose an IAM role for this job. Changing the role will end an existing data preview session.

Admin
No description available.
▼

[Create IAM role.](#)

Automatically start data preview sessions
Data preview will automatically start new interactive sessions when entering the visual job editor enabling you to preview data more efficiently.
⚠ This setting applies to all users in your account.

Infer schema from session
Output schemas will be automatically inferred based on the result of the datapreview execution

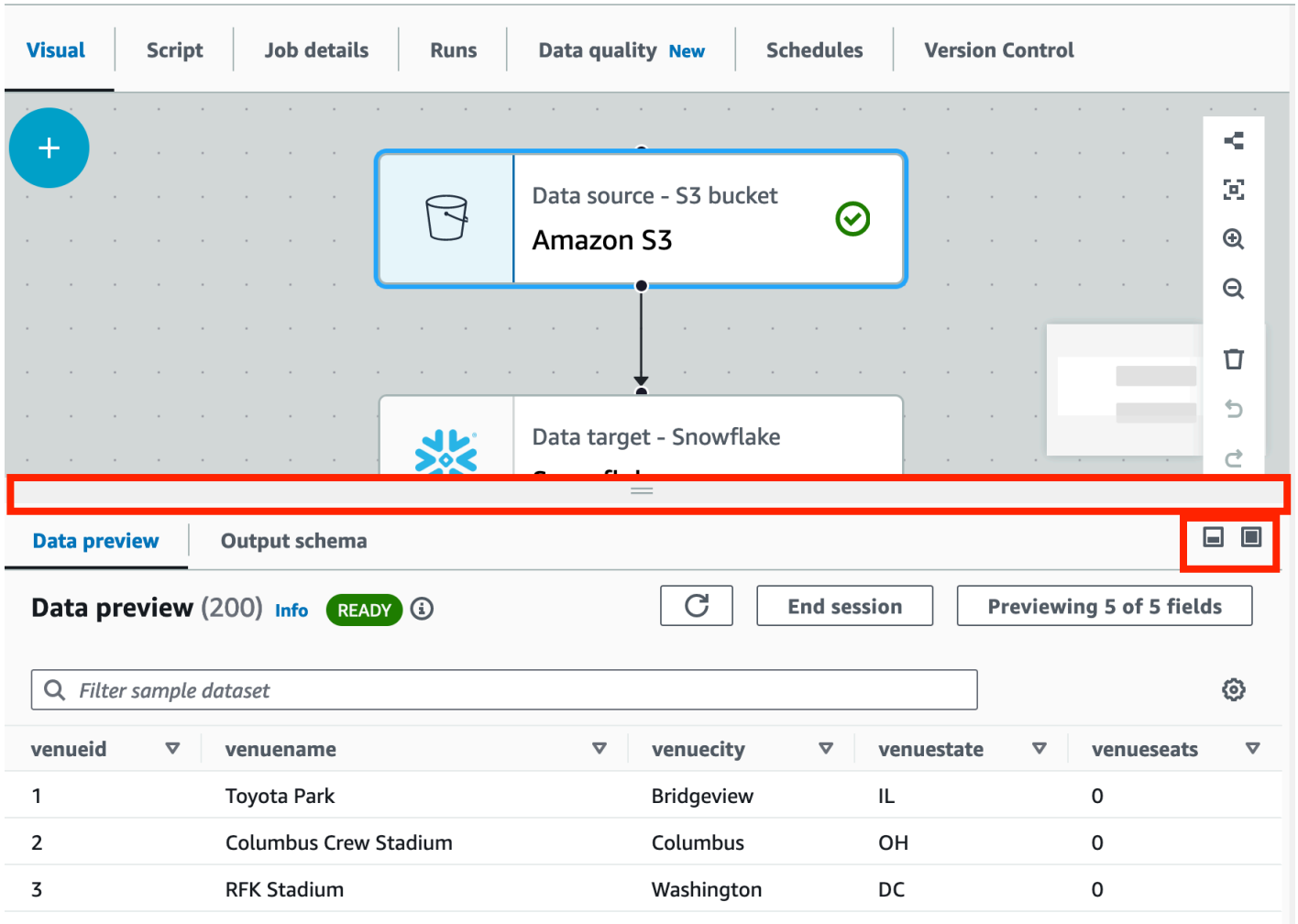
Automatically import glue libraries
Some ETL transform require extra libraries to be imported in the datapreview session, enabling this option will automatically import them to your sessions in order to prevent session from restarting during your job authoring. Note: the IAM role require read permission to Glue S3 bucket to prevent failures.

Cancel
Confirm

추가 기능에는 다음과 같은 기능이 포함됩니다.

- [필드 y개 중 x개 미리 보기(Previewing x of y fields)] 버튼을 선택하여 미리 보려는 열(필드)을 선택합니다. 기본 설정을 사용하여 데이터를 미리 보면 작업 편집기에 데이터 집합의 처음 5개 열이 표시됩니다. 모두 표시하거나 표시하지 않도록 변경할 수 있습니다(권장하지 않음).

- 데이터 미리 보기 창을 가로 및 세로로 스크롤할 수 있습니다.
- 최대화 버튼을 사용하여 데이터 미리보기 탭을 오버레이 작업 그래프로 확장하면 데이터 및 데이터 구조를 더 잘 볼 수 있습니다. 이와 유사하게 최소화 버튼을 사용하여 데이터 미리 보기 탭을 최소화합니다. 핸들 창을 잡고 위로 드래그하여 데이터 미리 보기 탭을 확장할 수도 있습니다.



- 세션 종료를 사용하여 데이터 미리 보기를 중지합니다. 세션을 중지할 때 새 IAM 역할을 선택하고, 새 세션을 자동으로 시작하거나, 스키마를 유추하거나, AWS Glue 라이브러리를 가져와서 세션을 다시 시작하도록 추가 설정(예: 설정 켜기 또는 끄기)을 설정할 수 있습니다.

데이터 미리 보기 사용 시 제한 사항

데이터 미리 보기를 사용할 때 다음과 같은 제한 사항이 있을 수 있습니다.

- [데이터 미리 보기(Data preview)] 탭을 처음 선택할 때 IAM 역할을 선택해야 합니다. 이 역할에는 데이터 미리 보기를 만드는 데 필요한 데이터 및 기타 리소스에 액세스하는 데 필요한 권한이 있어야 합니다.

- IAM 역할을 제공한 후 데이터를 볼 수 있을 때까지 시간이 걸립니다. 데이터가 1GB 미만인 데이터 집합의 경우 최대 1분이 소요될 수 있습니다. 큰 데이터 집합이 있는 경우 파티션을 사용하여 로드 시간을 개선해야 합니다. Amazon S3에서 직접 데이터를 로드하는 것이 성능이 가장 좋습니다.
- 매우 큰 데이터 집합이 있고 데이터 미리 보기를 위해 데이터를 쿼리하는 데 15분 이상 걸리는 경우 요청 시간이 초과됩니다. 데이터 미리 보기의 유효 제한 시간은 30분입니다. 이를 완화하려면 데이터 미리 보기 사용에 데이터 세트 크기를 줄이세요.
- 기본으로 처음 50 열이 데이터 미리 보기 탭에 표시됩니다. 열에 데이터 값이 없는 경우 표시 할 데이터가 없다는 메시지가 나타납니다. 샘플링된 행 수를 늘리거나 다른 열을 선택하여 데이터 값을 볼 수 있습니다.
- 데이터 미리 보기는 현재 스트리밍 데이터 원본 또는 사용자 정의 커넥터를 사용하는 데이터 원본에 대해 지원되지 않습니다.
- 한 노드의 오류는 전체 작업에 영향을 줍니다. 데이터 미리 보기에서 한 노드에 오류가 있는 경우 이를 수정할 때까지 모든 노드에 오류가 표시됩니다.
- 작업의 데이터 원본을 변경하는 경우 해당 데이터 원본의 하위 노드를 새 스키마와 일치하도록 업데이트해야 할 수 있습니다. 예를 들어 열을 수정하는 ApplyMapping 노드가 있고 해당 열이 대체 데이터 원본에 없는 경우 ApplyMapping 변환 노드를 업데이트해야 합니다.
- SQL 쿼리 변환 노드에 대한 데이터 미리 보기 탭을 볼 때 SQL 쿼리에서 잘못된 필드 이름을 사용하면 데이터 미리 보기 탭에 오류가 표시됩니다.

스크립트 코드 생성

시각적 편집기를 사용하여 작업을 생성하는 경우 자동으로 ETL 코드가 생성됩니다. AWS Glue Studio는 기능적이고 완전한 작업 스크립트를 생성하여 Amazon S3 위치에 저장합니다.

AWS Glue Studio에서는 원본 또는 클래식 버전과 간소화된 최신 버전인 두 가지 형식의 코드를 생성합니다. 기본적으로 새 코드 생성기가 작업 스크립트를 생성하는 데 사용됩니다. 스크립트(Script) 탭에서 클래식 스크립트 생성(Generate classic script) 토크 버튼을 선택하여 클래식 코드 생성기로 작업 스크립트를 생성할 수 있습니다.

새 버전의 생성된 코드에서 유의할 몇 가지 차이점은 다음과 같습니다.

- 더 이상 스크립트에 큰 주석 블록이 추가되지 않습니다.
- 시각적 편집기에서 지정한 노드 이름이 코드의 출력 구조에 사용됩니다. 클래식 스크립트에서 출력 구조의 이름이 간단히 DataSource0, DataSource1, Transform0, Transform1, DataSink0, DataSink1 등으로 지정됩니다.
- 긴 명령이 여러 줄로 분할되므로 전체 명령을 보기 위해 페이지를 스크롤할 필요가 없습니다.

AWS Glue Studio의 새로운 기능을 사용하려면 새 버전의 코드를 생성해야 하며 클래식 코드 스크립트에서는 작동하지 않습니다. 이러한 작업을 실행하려고 하면 업데이트하라는 메시지가 표시됩니다.

AWS Glue 관리형 데이터 변환 노드 편집

AWS Glue Studio에서는 두 가지 유형의 변환을 제공합니다.

- **AWS Glue 네이티브 변환** - 모든 사용자가 사용할 수 있으며 AWS Glue에서 관리합니다.
- **사용자 지정 시각적 변환** - 자체 변환을 업로드하여 AWS Glue Studio에서 사용할 수 있습니다.

AWS Glue 관리형 데이터 변환 노드

AWS Glue Studio는 기본 설정 변환 세트를 제공하여 데이터를 사용할 수 있습니다. 데이터는 작업 다이어그램의 한 노드에서 Apache Spark SQL DataFrame의 확장인 DynamicFrame이라는 데이터 구조의 다른 노드로 전달됩니다.

작업에 대해 미리 채워진 다이어그램에서 데이터 소스와 데이터 대상 노드 사이에는 스키마 변경 변환 노드가 있습니다. 이 변환 노드를 구성하여 데이터를 수정하거나 추가 변환을 사용할 수 있습니다.

AWS Glue Studio에서 사용 가능한 기본 제공 변환은 다음과 같습니다.

- **[ChangeSchema](#)**: 데이터 소스의 데이터 속성 키를 데이터 대상의 데이터 속성 키에 매핑합니다. 키의 이름을 바꾸고 키의 데이터 유형을 수정하고 데이터 집합에서 삭제할 키를 선택할 수 있습니다.
- **[SelectFields](#)**: 유지할 데이터 속성 키를 선택합니다.
- **[DropFields](#)**: 삭제할 데이터 속성 키를 선택합니다.
- **[RenameField](#)**: 단일 데이터 속성 키의 이름을 바꿉니다.
- **[Spigot](#)**: Amazon S3 버킷에 데이터 샘플을 씁니다.
- **[Join](#)**: 지정된 데이터 속성 키의 비교 구문을 사용하여 두 데이터 집합을 하나의 데이터 집합으로 조인합니다. 내부, 외부, 왼쪽, 오른쪽, 왼쪽 반 및 왼쪽 안티 조인을 사용할 수 있습니다.
- **[Union](#)**: 스키마가 동일한 둘 이상의 데이터 소스에서 행을 결합합니다.
- **[SplitFields](#)**: 데이터 속성 키를 두 개의 DynamicFrames로 분할합니다. 출력은 DynamicFrames의 컬렉션입니다. 하나는 선택한 데이터 속성 키가 있고 다른 하나는 나머지 데이터 속성 키가 있습니다.
- **[SelectFromCollection](#)**: DynamicFrames 컬렉션에서 DynamicFrame을 하나 선택합니다. 출력은 선택된 DynamicFrame입니다.

- **FillMissingValues**: 데이터 집합에서 누락 값이 있는 레코드를 찾고 대체를 통해 결정된 제안 값으로 새 필드를 추가합니다.
- **필터(Filter)**: 필터 조건에 따라 하나의 데이터 집합을 두 개로 분할합니다.
- **Null 필드 삭제**: 열의 모든 값이 'null'인 경우 데이터 집합에서 열을 제거합니다.
- **중복 삭제**: 전체 행을 일치시키거나 키를 지정하도록 선택하여 데이터 소스에서 행을 제거합니다.
- **SQL**: SQL 쿼리를 사용하여 데이터를 변환하려면 텍스트 입력 필드에 SparkSQL 코드를 입력합니다. 출력은 단일 DynamicFrame입니다.
- **집계**: 선택한 필드와 행에서 계산(예: 평균, 합계, 최소, 최대)을 수행하고 새로 계산된 값으로 새 필드를 생성합니다.
- **Flatten**: 구조체 내부의 필드를 최상위 필드로 추출합니다.
- **UUID**: 각 행에 범용 고유 식별자가 있는 열을 추가합니다.
- **식별자**: 각 행에 숫자 식별자가 있는 열을 추가합니다.
- **타임스탬프로 변환**: 열을 타임스탬프 유형으로 변환합니다.
- **타임스탬프 형식 지정**: 타임스탬프 열을 형식이 지정된 문자열로 변환합니다.
- **조건부 라우터 변환**: 수신 데이터에 여러 조건을 적용합니다. 수신 데이터의 각 행은 그룹 필터 조건을 기준으로 평가되고 해당 그룹으로 처리됩니다.
- **열 연결 변환**: 선택적 스페이서가 있는 다른 열의 값을 사용하여 새 문자열 열을 구축합니다.
- **문자열 분할 변환**: 정규식을 사용하여 문자열을 토큰 배열로 분할해 분할 방식을 정의합니다.
- **배열을 열로 변환**: 배열 유형의 열에 있는 일부 또는 모든 요소를 새 열로 추출합니다.
- **현재 타임스탬프 추가 변환**: 데이터가 처리된 시간으로 행을 표시합니다. 이는 감사 목적이거나 데이터 파이프라인에서 지연 시간을 추적하는 데 유용합니다.
- **행을 열로 피벗 변환**: 선택한 열에서 고유 값을 교체하여 숫자 열을 집계합니다. 이 열은 새 열이 됩니다. 열을 여러 개 선택하면 값이 연결되어 새 열의 이름이 지정됩니다.
- **열을 행으로 피벗 취소 변환**: 열을 새 열의 값으로 변환하여 각 고유 값에 대한 행을 생성합니다.
- **처리 균형 자동 조절 변환**: 더 나은 성능을 위해 작업자 사이에서 데이터를 재배포합니다. 이는 데이터가 불균형하거나 소스에서 가져온 데이터로 인해 충분한 병렬 처리가 불가능한 경우에 유용합니다.
- **파생 열 변환**: 상수 및 리터럴뿐만 아니라 데이터의 다른 열을 사용할 수 있는 수학 공식 또는 SQL 표현식을 기반으로 새 열을 정의합니다.
- **조회 변환**: 키가 데이터에 정의된 조회 열과 일치하는 경우 정의된 카탈로그 테이블의 열을 추가합니다.
- **배열 또는 맵을 행으로 분해 변환**: 중첩된 구조에서 조작하기 쉬운 개별 행으로 값을 추출합니다.

- [레코드 일치 변환](#): 기존 레코드 일치 기계 학습 데이터 분류 변환을 간접 호출합니다.
- [null 행 제거 변환](#) 제거: 모든 열이 null이거나 비어 있는 행을 데이터 세트에서 제거합니다.
- [JSON 열 구문 분석 변환](#): JSON 데이터를 포함하는 문자열 열을 구문 분석하고 JSON이 객체인지 또는 배열인지에 따라 각각 해당 문자열 열을 구문 또는 배열 열로 변환합니다.
- [JSON 경로 추출 변환](#): JSON 문자열 열에서 새 열을 추출합니다.
- [정규식에서 문자열 조각 추출](#): 정규식을 사용하여 문자열 조각을 추출하고 문자열 조각에서 새 열을 생성하거나 정규식 그룹을 사용하는 경우 여러 열을 생성합니다.
- [사용자 지정 변환\(Custom transform\)](#): 사용자 지정 변환을 사용하려면 텍스트 입력 필드에 코드를 입력합니다. 출력은 DynamicFrames의 컬렉션입니다.

AWS Glue Studio에서 데이터 준비 레시피 사용

AWS Glue Studio를 사용하면 시각적 워크플로에서 AWS Glue DataBrew 레시피를 사용할 수 있습니다. 이를 통해 고객의 AWS Glue DataBrew 레시피를 다른 AWS Glue Studio 노드와 함께 AWS Glue 작업에서 실행할 수 있습니다.

DataBrew에서 레시피는 데이터 변환 단계 세트입니다. DataBrew 레시피에서는 이미 읽은 데이터를 변환하는 방법을 규정하지만, 데이터를 읽는 위치와 방법, 그리고 데이터를 쓰는 방법과 위치에 대해서는 설명하지 않습니다. 레시피는 AWS Glue Studio의 소스 및 대상 노드에서 구성됩니다. 레시피에 대한 자세한 내용은 [Creating and using AWS Glue DataBrew recipes](#)를 참조하세요.

데이터 준비 레시피 노드는 리소스 패널에서 사용할 수 있습니다. 데이터 준비 레시피 노드를 시각적 워크플로의 다른 노드(데이터 소스 노드이거나 다른 변환 노드)에 연결할 수 있습니다. AWS Glue DataBrew 레시피와 버전을 선택하면 레시피에 적용된 단계가 노드 속성 탭에 표시됩니다.

필수 조건

- AWS Glue DataBrew에서 AWS Glue DataBrew 레시피가 생성되었습니다.
- 아래 섹션에 나와 있는 필수 IAM 권한이 있습니다.

AWS Glue DataBrew에 대한 IAM 권한

이 주제에서는 IAM 관리자가 데이터 준비 레시피 변환에 대한 AWS Identity and Access Management(IAM) 정책에서 사용할 수 있는 작업과 리소스를 이해하는 데 도움이 되는 정보를 제공합니다.

AWS Glue에서 보안에 대한 자세한 내용은 [액세스 관리](#)를 참조하세요.

다음 테이블에는 데이터 준비 레시피 변환을 사용하는 데 필요한 특정 작업을 수행하기 위해 사용자에게 필요한 권한이 나와 있습니다.

데이터 준비 레시피 변환 작업

작업	설명
<code>databrew:ListRecipes</code>	AWS Glue DataBrew 레시피를 검색할 수 있는 권한을 부여합니다.
<code>databrew:ListRecipeVersions</code>	AWS Glue DataBrew 레시피 버전을 검색할 수 있는 권한을 부여합니다.
<code>databrew:DescribeRecipe</code>	AWS Glue DataBrew 레시피 설명을 검색할 수 있는 권한을 부여합니다.

이 기능에 액세스하는 데 사용하는 역할에는 여러 AWS Glue DataBrew를 허용하는 정책이 있어야 합니다. 필요한 작업을 포함하는 `AWSGlueConsoleFullAccess` 정책을 사용하거나 역할에 다음과 같은 인라인 정책을 추가하면 됩니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "databrew:ListRecipes",
        "databrew:ListRecipeVersions",
        "databrew:DescribeRecipe"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

데이터 준비 레시피 변환을 사용하려면 권한 정책에 `IAM:PassRole` 작업을 추가해야 합니다.

필요한 추가 권한

작업	설명
iam:PassRole	사용자가 승인된 역할을 전달할 수 있도록 IAM에 권한을 부여합니다.

이 권한이 없으면 다음과 같은 오류가 발생합니다.

```
"errorCode": "AccessDenied"
"errorMessage": "User: arn:aws:sts::account_id:assumed-role/AWSGlueServiceRole is not
authorized to perform: iam:PassRole on resource: arn:aws:iam::account_id:role/service-
role/AWSGlueServiceRole
because no identity-based policy allows the iam:PassRole action"
```

제한 사항

- AWS Glue에서 모든 AWS Glue DataBrew 레시피가 지원되는 것은 아닙니다. 일부 레시피는 AWS Glue Studio에서 실행할 수 없습니다.
 - 집합 및 조인 변환이 포함된 레시피는 지원되지 않습니다. 대신, AWS Glue Studio에는 데이터 준비 레시피 노드 이전 또는 이후에 사용할 수 있는 '조인' 및 '집합' 변환 노드가 이미 있습니다.
- 데이터 준비 레시피 노드는 AWS Glue 버전 4.0 이상 작업에 대해 지원됩니다. 이 버전은 데이터 준비 레시피 노드가 작업에 추가된 후에 자동으로 선택됩니다.
- 데이터 준비 레시피 노드에는 Python이 필요합니다. 데이터 준비 레시피 노드가 작업에 추가될 때 자동으로 설정됩니다.
- 데이터 미리 보기를 사용하는 경우 작업에 데이터 준비 레시피 노드를 추가한 후 데이터 미리 보기 세션을 다시 시작해야 합니다.

AWS Glue Studio에서 AWS Glue DataBrew 레시피를 사용하는 방법

AWS Glue Studio에서 AWS Glue DataBrew 레시피를 사용하려면 먼저 AWS Glue DataBrew에서 레시피를 생성합니다. 사용할 레시피가 이미 있으면 이 단계를 건너뛸 수 있습니다.

AWS Glue DataBrew에서 AWS Glue DataBrew 레시피를 생성하려면:

1. AWS Glue DataBrew에서 레시피를 작성합니다. 자세한 내용은 [Getting started with AWS Glue DataBrew](#)를 참조하세요.

2. 레시피를 저장합니다.
3. 레시피를 게시합니다. 그러면 레시피가 버전 1.0으로 게시됩니다.

AWS Glue Studio에서 데이터 준비 레시피 노드를 사용하려면:

시각적 ETL 작업에서 둘 이상의 데이터 준비 레시피 노드를 사용할 수 있습니다. 이렇게 하려면 아래 단계에 따라 데이터 준비 레시피 노드를 추가하고 작업에 다른 데이터 준비 레시피 노드를 추가합니다. 예를 들어 워크플로는 다음과 같은 패턴을 따를 수 있습니다.

- 데이터 소스 1 > 레시피 1 > 출력 1
- 데이터 소스 2 > 레시피 2 > 출력 2
- 출력 1, 출력 2 > 조인

1. AWS Glue Studio에서 데이터 소스와 함께 AWS Glue 작업을 시작합니다.
2. 데이터 준비 레시피 노드를 데이터소스에 추가합니다.
3. 검색 필드에 레시피 이름을 입력하여 이름을 기준으로 레시피를 필터링합니다.
4. 게시된 버전을 선택합니다. 게시된 버전만 사용할 수 있습니다.
5. 필요한 경우 다른 변환 노드를 추가하여 작업 작성을 완료하고 데이터 대상 노드를 추가하여 작업 출력을 저장합니다.
6. 필요한 경우 작업 세부 정보 탭에서 작업 이름 지정 및 할당된 용량 조정과 같은 필요한 구성 변경을 수행하고 작업을 저장합니다.
7. 작업 드롭다운 메뉴에서 실행을 선택하여 작업을 실행합니다.

데이터 소스가 Amazon S3이고 데이터 형식이 CSV인 경우 스키마를 변경하려면:

AWS Glue Studio에서 CSV 파일의 모든 열이 처음에 문자열 데이터 형식으로 로드되는 경우 열 데이터 형식이 AWS Glue DataBrew 레시피의 나머지 단계와 호환되는지 확인해야 합니다.

AWS Glue DataBrew 레시피는 이미 읽은 데이터를 변환하는 방법만 규정합니다. 데이터를 읽는 위치와 방법은 기술하지 않습니다.

1. 다단계 레시피 노드 앞에 스키마 변경 노드를 추가합니다.
2. 스키마 변경 노드를 클릭하고 필요한 경우 열에 대한 변환에서 새 데이터 형식을 선택하여 AWS Glue DataBrew의 열 데이터 형식과 동일하도록 스키마를 변경합니다.

Transform
✕

Name

Node parents
Choose which nodes will provide inputs for this one.

Choose one or more parent node
▼

S3 bucket
✕

S3 - DataSource

Change Schema (Apply mapping) ⌵

Source key	Target key	Data type	Drop
col0	<input style="width: 80%;" type="text" value="col0"/>	string ▼	<input type="checkbox"/>
col1	<input style="width: 80%;" type="text" value="col1"/>	string ▼	<input type="checkbox"/>
col2	<input style="width: 80%;" type="text" value="col2"/>	string ▼	<input type="checkbox"/>
col3	<input style="width: 80%;" type="text" value="col3"/>	string ▼	<input type="checkbox"/>
col4	<input style="width: 80%;" type="text" value="col4"/>	string ▼	<input type="checkbox"/>
col5	<input style="width: 80%;" type="text" value="col5"/>	string ▼	<input type="checkbox"/>
col6	<input style="width: 80%;" type="text" value="col6"/>	string ▼	<input type="checkbox"/>
col7	<input style="width: 80%;" type="text" value="col7"/>	string ▼	<input type="checkbox"/>
col8	<input style="width: 80%;" type="text" value="col8"/>	string ▼	<input type="checkbox"/>

데이터 소스에 헤더가 없는 경우 스키마를 변경하려면:

AWS Glue DataBrew 레시피는 이미 읽은 데이터를 변환하는 방법만 규정합니다. 데이터를 읽는 위치와 방법은 기술하지 않습니다.

AWS Glue Studio에서 헤더가 없는 데이터 세트를 로드할 때 기본 헤더 이름은 AWS Glue DataBrew에서 로드된 헤더 이름과 다릅니다.

1. ETL 작업에서 데이터 준비 레시피 앞에 스키마 변경 노드를 추가합니다.
2. 스키마 변경 노드를 선택하고 AWS Glue DataBrew 레시피에 사용된 것과 동일한 이름으로 열 이름을 변경합니다.

스키마 변경을 사용하여 데이터 속성 키 다시 매핑

스키마 변경 변환은 소스 데이터 속성 키를 대상 데이터에 대해 구성된 원하는 항목으로 다시 매핑합니다. 스키마 변경 변환 노드에서는 다음을 수행할 수 있습니다.

- 여러 데이터 속성 키의 이름을 변경합니다.
- 새 데이터 유형이 지원되고 두 데이터 유형 사이에 변환 경로가 있는 경우 데이터 속성 키의 데이터 유형을 변경합니다.
- 삭제할 데이터 속성 키를 표시하여 데이터 속성 키의 하위 집합을 선택합니다.

필요에 따라 작업 다이어그램에 변경 스키마 노드를 추가할 수도 있습니다 (예: 추가 데이터 소스를 수정하거나 조인 변환 수행).

10진수 데이터 유형의 변경 스키마 사용

변경 스키마 변환을 십진 데이터 유형과 함께 사용하는 경우 변경 스키마 변환은 정밀도를 디폴트 값인 (10,2) 로 수정합니다. 이를 수정하고 사용 사례의 정밀도를 설정하려면 SQL 쿼리 변환을 사용하고 특정 정밀도로 열을 캐스팅할 수 있습니다.

예를 들어 Decimal 유형의 입력 열이 DecimalCol 이고 특정 정밀도가 (18,6) 인 입력 열을 OutputDecimalCol 라는 출력 열에 다시 매핑하려면 다음과 같이 하십시오.

1. 스키마 변경 변환 이후에 후속 SQL 쿼리 변환을 추가합니다.
2. SQL 쿼리 변환에서 SQL 쿼리를 사용하여 재매핑된 열을 원하는 정밀도로 캐스팅합니다. SQL 쿼리는 다음과 같습니다.

```
SELECT col1, col2, CAST(DecimalCol AS DECIMAL(18,6)) AS OutputDecimalCol
FROM __THIS__
```

위의 SQL 쿼리에서:

- `col1`과 `col2`는 수정 없이 전달하려는 데이터의 다른 열입니다.
- `DecimalCol`는 입력 데이터의 원래 열 이름입니다.

- `CAST (DecimalCol AS DECIMAL (18,6))`는 ``를 소수점 18자리, 소수점 6자리의 정밀도를 가진 십진수 유형으로 캐스팅합니다. DecimalCol
- `AS`는 캐스팅된 열의 이름을 ``로 바꿉니다. OutputDecimalCol OutputDecimalCol

SQL 쿼리 변환을 사용하면 변경 스키마 변환으로 설정된 기본 정밀도를 재정의하고 십진수 열을 원하는 정밀도로 명시적으로 변환할 수 있습니다. 이 접근 방식을 사용하면 변경 스키마 변환을 활용하여 데이터 이름을 바꾸고 구조화하는 동시에 후속 SQL 쿼리 변환을 통해 십진수 열에 대한 정밀도 요구 사항을 처리할 수 있습니다.

변경 스키마 변환을 작업에 추가

Note

스키마 변경 변환은 대소문자를 구분하지 않습니다.

작업 다이어그램에 스키마 변경 변환 노드를 추가하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 스키마 변경을 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. 노드 속성 패널에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 노드 속성 패널에서 변환 탭을 선택합니다.
4. 입력 스키마를 수정합니다.
 - 데이터 속성 키의 이름을 바꾸려면 [대상 키(Target key)] 필드에 키의 새 이름을 입력합니다.
 - 데이터 속성 키의 데이터 유형을 변경하려면 [데이터 유형(Data type)] 목록에서 키의 새 데이터 유형을 선택합니다.
 - 대상 스키마에서 데이터 속성 키를 제거하려면 해당 키의 [삭제(Drop)] 확인란을 선택합니다.
5. (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.

6. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

중복 삭제 사용

중복 삭제 변환은 두 가지 옵션을 제공하여 데이터 소스에서 행을 제거합니다. 완전히 동일한 중복 행을 제거하거나 일치시킬 필드를 선택하고 선택한 필드를 기반으로 해당 행만 제거하도록 선택할 수 있습니다.

예를 들어 이 데이터 세트에는 일부 행에서는 모든 값이 다른 행과 완전히 동일하고, 또 다른 행에서는 값이 같거나 다른 중복 행이 있습니다.

열	명칭	이메일	Age	State	참고
1	Joy	joy@gmail	33	NY	
2	Tim	tim@gmail	45	OH	
3	Rose	rose@gmail	23	NJ	
4	Tim	tim@gmail	42	OH	
5	Rose	rose@gmail	23	NJ	
6	Tim	tim@gmail	42	OH	이것은 중복된 행이며 4번째 행의 모든 값과 완전히 일치합니다.
7	Rose	rose@gmail	23	NJ	이것은 중복된 행이며 5번째 행의 모든 값과 완전히 일치합니다.

전체 행을 일치시키도록 선택하면 6번째 행과 7번째 행이 데이터 세트에서 제거됩니다. 이제 데이터 세트는 다음과 같습니다.

열	명칭	이메일	Age	State
1	Joy	joy@gmail	33	NY
2	Tim	tim@gmail	45	OH
3	Rose	rose@gmail	23	NJ
4	Tim	tim@gmail	42	OH
5	Rose	rose@gmail	23	NJ

키를 지정하기로 선택한 경우 'name' 및 'email'에서 일치하는 행을 제거하도록 선택할 수 있습니다. 이렇게 하면 데이터 세트의 '중복 행'을 더 세밀하게 제어할 수 있습니다. 'name' 및 'email'을 지정하면 이제 데이터 세트는 다음과 같습니다.

열	명칭	이메일	Age	State
1	Joy	joy@gmail	33	NY
2	Tim	tim@gmail	45	OH
3	Rose	rose@gmail	23	NJ

다음 사항에 유의하세요.

- 중복 행으로 인식하기 위해 값은 대소문자를 구분합니다. 행의 모든 값은 동일한 대소문자를 사용해야 합니다. 이는 선택한 옵션(전체 행 일치 또는 키 지정)에 적용됩니다.
- 모든 값은 문자열로 읽습니다.
- 중복 삭제 변환은 Spark dropDuplicates 명령을 활용합니다.
- 중복 삭제 변환을 사용하는 경우 첫 번째 행은 유지되고 다른 행은 삭제됩니다.
- 중복 삭제 변환은 데이터 프레임의 스키마를 변경하지 않습니다. 키를 지정하도록 선택한 경우 모든 필드가 그 결과로 나타나는 데이터 프레임에서 유지됩니다.

SelectFields를 사용하여 대부분의 데이터 속성 키 제거

SelectFields 변환을 사용하여 데이터 집합에서 데이터 속성 키의 하위 집합을 생성할 수 있습니다. 유지하려는 데이터 속성 키를 지정하면 나머지는 데이터 집합에서 제거됩니다.

Note

SelectFields는 대/소문자를 구분합니다. 필드를 선택하기 위해 대/소문자를 구분하지 않는 방법이 필요한 경우 ApplyMapping을 사용합니다.

작업 다이어그램에 SelectFields 변환 노드를 추가하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 SelectFields를 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 노드 세부 정보 패널에서 [변환(Transform)] 탭을 선택합니다.
4. 머리글 [SelectFields] 아래에서 유지하려는 데이터 집합의 데이터 속성 키를 선택합니다. 선택하지 않은 데이터 속성 키는 데이터 집합에서 모두 삭제됩니다.

열 머리글 [필드(Field)] 옆에 있는 확인란을 선택하여 데이터 집합의 모든 데이터 속성 키를 자동으로 선택할 수도 있습니다. 그런 다음 개별 데이터 속성 키를 선택 취소하여 데이터 집합에서 제거할 수 있습니다.

5. (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.
6. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

DropFields를 사용하여 대부분의 데이터 속성 키 유지

DropFields 변환을 사용하여 데이터 집합에서 데이터 속성 키의 하위 집합을 생성할 수 있습니다. 데이터 집합에서 제거하려는 데이터 속성 키를 지정하면 나머지 키는 유지됩니다.

Note

DropFields 변환은 대/소문자를 구분합니다. 필드를 선택할 때 대소문자를 구분하지 않는 방법이 필요한 경우 스키마 변경을 사용합니다.

작업 다이어그램에 DropFields 변환 노드를 추가하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 DropFields를 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 노드 세부 정보 패널에서 [변환(Transform)] 탭을 선택합니다.
4. 머리글 [DropFields] 아래에서 데이터 원본에서 삭제할 데이터 속성 키를 선택합니다.

열 머리글 [필드(Field)] 옆에 있는 확인란을 선택하여 데이터 집합의 모든 데이터 속성 키를 자동으로 선택할 수도 있습니다. 그런 다음 개별 데이터 속성 키를 선택 취소하여 데이터 집합에 보존할 수 있습니다.

5. (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.
6. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

데이터 집합에서 필드 이름 바꾸기

RenameField 변환을 사용하여 데이터 집합의 개별 속성 키 이름을 변경할 수 있습니다.

Note

RenameField는 대/소문자를 구분합니다. 대/소문자를 구분하지 않는 변환이 필요한 경우 ApplyMapping을 사용합니다.

Tip

스키마 변경 변환을 사용하는 경우 단일 변환으로 데이터 세트에 있는 여러 데이터 속성 키의 이름을 바꿀 수 있습니다.

작업 다이어그램에 RenameField 변환 노드를 추가하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 RenameField를 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. [변환(Transform)] 탭을 선택합니다.
4. 머리글 [데이터 필드(Data field)] 아래에서 소스 데이터의 속성 키를 선택한 다음 [새 필드 이름(New field name)] 필드에 새 이름을 입력합니다.
5. (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.
6. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

Spigot을 사용하여 데이터 집합 샘플링

작업에서 수행한 변환을 테스트하기 위해 데이터 샘플을 가져와 변환이 의도한 대로 작동하는지 확인할 수 있습니다. Spigot 변환은 데이터 집합의 레코드 하위 집합을 Amazon S3 버킷의 JSON 파일에 기

록합니다. 데이터 샘플링 방법은 파일 시작 부분의 특정 레코드 수 또는 레코드 선택에 사용되는 확률 요소일 수 있습니다.

작업 다이어그램에 Spigot 변환 노드를 추가하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 Spigot을 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 노드 세부 정보 패널에서 [변환(Transform)] 탭을 선택합니다.
4. Amazon S3 경로를 입력하거나 [S3 찾아보기(Browse S3)]를 선택하여 Amazon S3에서 위치를 선택합니다. 작업이 데이터 샘플이 포함된 JSON 파일을 작성하는 위치입니다.
5. 샘플링 방법에 대한 정보를 입력합니다. 데이터 집합의 시작부터 쓸 [레코드 수(Number of records)]에 대한 값과 지정된 레코드를 선택하는 [확률 임계값(Probability threshold)](최대값이 1인 십진수 값으로 입력됨)을 지정할 수 있습니다.

예를 들어 데이터 집합에서 처음 50개의 레코드를 쓰려면 [레코드 수(Number of records)]를 50으로 설정하고 [확률 임계값(Probability threshold)]을 1(100%)로 설정합니다.

데이터 집합 조인

Join 변환을 사용하면 2개의 데이터 집합을 하나로 결합할 수 있습니다. 비교할 각 데이터 집합의 스키마에 키 이름을 지정합니다. 출력 DynamicFrame에는 키가 조인 조건을 충족하는 행이 포함됩니다. 조인 조건을 충족하는 각 데이터 집합의 행은 두 데이터 집합에서 찾은 모든 열을 포함하는 출력 DynamicFrame의 단일 행으로 결합됩니다.

작업 다이어그램에 Join 변환 노드를 추가하려면

1. 사용 가능한 데이터 원본이 하나만 있는 경우 작업 다이어그램에 새 데이터 원본 노드를 추가해야 합니다.
2. 조인을 위한 소스 노드 중 하나를 선택합니다. 리소스 패널을 열고 Join을 선택하여 작업 다이어그램에 새 변환을 추가합니다.
3. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다.
4. 조인에 대한 입력을 제공하는 2개의 데이터 집합이 있도록 [노드 속성(Node properties)] 탭의 머리글 [노드 상위 항목(Node parents)] 아래에서 상위 노드를 추가합니다. 상위 항목은 데이터 원본 노드 또는 변환 노드일 수 있습니다.

Note

조인은 2개의 상위 노드만 가질 수 있습니다.

5. [변환(Transform)] 탭을 선택합니다.

충돌하는 키 이름이 있다는 메시지가 나타나면 다음 중 하나를 수행할 수 있습니다.

- 작업 다이어그램에 ApplyMapping 변환 노드를 자동으로 추가하려면 [해결(Resolve it)]을 선택합니다. ApplyMapping 노드는 다른 데이터 집합의 키와 이름이 같은 데이터 집합의 모든 키에 접두사를 추가합니다. 예를 들어 기본값인 **right**를 사용하면 왼쪽 데이터 집합의 키와 이름이 같은 오른쪽 데이터 집합의 모든 키 이름이 (right)key name으로 바뀝니다.
- 충돌하는 키를 제거하거나 이름을 바꾸려면 작업 다이어그램 앞부분에 변환 노드를 수동으로 추가합니다.

6. [조인 유형(Join type)] 목록에서 조인 유형을 선택합니다.

- [내부 조인(Inner join)]: 조인 조건에 따라 모든 일치 항목에 대해 두 데이터 집합의 열이 있는 행을 반환합니다. 조인 조건을 만족하지 않는 행은 반환되지 않습니다.
- [왼쪽 조인(Left join)]: 왼쪽 데이터 집합의 모든 행과 조인 조건을 충족하는 오른쪽 데이터 집합의 행만.
- [오른쪽 조인(Right join)]: 오른쪽 데이터 집합의 모든 행과 조인 조건을 충족하는 왼쪽 데이터 집합의 행만.
- [외부 조인(Outer join)]: 두 데이터 집합의 모든 행.
- [왼쪽 세미 조인(Left semi join)]: 조인 조건에 따라 오른쪽 데이터 집합과 일치하는 왼쪽 데이터 집합의 모든 행.
- [왼쪽 안티 조인(Left semi join)]: 조인 조건에 따라 오른쪽 데이터 집합과 일치하지 않는 왼쪽 데이터 집합의 모든 행.

7. [변환(Transform)] 탭의 [조인 조건(Join conditions)] 아래에서 [조건 추가(Add condition)]를 선택합니다. 각 데이터 집합에서 비교할 속성 키를 선택합니다. 비교 연산자의 왼쪽에 있는 속성 키를 왼쪽 데이터 집합이라고 하고 오른쪽에 있는 속성 키를 오른쪽 데이터 집합이라고 합니다.

더 복잡한 조인 조건의 경우 [조건 추가(Add condition)]를 두 번 이상 선택하여 일치하는 키를 추가할 수 있습니다. 실수로 조건을 추가한 경우 삭제 아이콘

()

을 클릭하여 제거할 수 있습니다.

8. (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.
9. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

조인 출력 스키마의 예를 들어 다음 속성 키를 사용하여 두 데이터 집합 간의 조인을 고려합니다.

```
Left: {id, dept, hire_date, salary, employment_status}
Right: {id, first_name, last_name, hire_date, title}
```

조인은 = 비교 연산자를 사용하여 id 및 hire_date 키에서 일치하도록 구성됩니다.

두 데이터 집합 모두 id 및 hire_date 키를 포함하므로 [해결(Resolve it)]을 선택하여 올바른 데이터 집합의 키에 접두사 **right**를 자동으로 추가합니다.

출력 스키마의 키는 다음과 같습니다.

```
{id, dept, hire_date, salary, employment_status,
(right)id, first_name, last_name, (right)hire_date, title}
```

집합을 사용하여 행 결합

스키마가 같은 둘 이상의 데이터 소스에서 행을 결합하려는 경우 집합(Union) 변환 노드를 사용합니다.

집합 변환에는 두 가지 유형이 있습니다.

1. 모두 - 모두를 적용하면 그 결과로 나타나는 집합에서 중복된 행을 제거하지 않습니다.
2. 고유 - 고유를 적용하면 그 결과로 나타나는 집합에서 중복된 행을 제거합니다.

집합과 조인 비교

집합을 사용하여 행을 결합합니다. 조인을 사용하여 열을 결합합니다.

시각적 ETL 캔버스에서 집합 변환 사용

1. 하나 이상의 데이터 소스를 추가하여 집합 변환을 수행합니다. 데이터 소스를 추가하려면 리소스 패널을 열고 소스 탭에서 데이터 소스를 선택합니다. 집합 변환을 사용하기 전에 집합과 관련된 모든 데이터 소스의 스키마와 구조가 동일한지 확인해야 합니다.
2. 집합 변환을 사용하여 결합하려는 데이터 소스가 둘 이상 있는 경우 집합 변환을 캔버스에 추가하여 집합 변환을 생성합니다. 캔버스에서 리소스 패널을 열고 '집합'을 검색합니다. 리소스 패널에서 변환 탭을 선택하고 집합 변환을 찾을 때까지 아래로 스크롤한 후 집합을 선택할 수도 있습니다.
3. 작업 캔버스에서 집합 노드를 선택합니다. 노드 속성 창에서 집합 변환에 연결할 상위 노드를 선택합니다.
4. AWS Glue에서는 집합 변환을 모든 데이터 소스에 적용할 수 있는지 확인하기 위해 호환성을 검사합니다. 데이터 소스의 스키마가 동일하면 작업이 허용됩니다. 데이터 소스의 스키마가 같지 않으면 유효하지 않음을 나타내는 오류 메시지가 표시됩니다. 'The input schemas of this union are not the same Consider using ApplyMapping to match the schemas.' 이 문제를 해결하려면 ApplyMapping 사용을 선택합니다.
5. 집합 유형을 선택합니다.
 1. 모두 - 기본적으로 모두 집합 유형이 선택됩니다. 이 경우 데이터 조합에 중복된 행이 있을 경우 행이 중복됩니다.
 2. 고유 - 그 결과로 나타나는 데이터 조합에서 중복된 행을 제거하려면 고유를 선택합니다.

SplitFields를 사용하여 데이터 집합을 2개로 분할

SplitFields 변환을 사용하면 입력 데이터 집합에서 일부 데이터 속성 키를 선택하여 하나의 데이터 집합에 넣고 선택되지 않은 키를 별도의 데이터 집합에 넣을 수 있습니다. 이 변환의 출력은 DynamicFrames의 컬렉션입니다.

Note

출력을 대상 위치로 보내기 전에 SelectFromCollection 변환을 사용하여 DynamicFrames 컬렉션을 단일 DynamicFrame으로 변환해야 합니다.

SplitFields는 대/소문자를 구분합니다. 대/소문자를 구분하지 않는 속성 키 이름이 필요한 경우 ApplyMapping 변환을 상위 노드로 추가합니다.

작업 다이어그램에 SplitFields 변환 노드를 추가하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 SplitFields를 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. [변환(Transform)] 탭을 선택합니다.
4. 첫 번째 데이터 집합에 넣을 속성 키를 선택합니다. 선택하지 않는 키는 두 번째 데이터 집합에 배치됩니다.
5. (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.
6. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.
7. 결과 데이터 집합을 처리하도록 SelectFromCollection 변환 노드를 구성합니다.

SelectFromCollection 변환의 개요

특정 변환에는 단일 데이터 집합 대신 여러 데이터 집합이 출력으로 포함됩니다(예: SplitFields). SelectFromCollection 변환은 데이터 집합(DynamicFrames의 배열)에서 하나의 데이터 집합(DynamicFrame)을 선택합니다. 변환에 대한 출력은 선택된 DynamicFrame입니다.

다음과 같이 DynamicFrames의 컬렉션을 생성하는 변환을 사용한 후에는 이 변환을 사용해야 합니다.

- 사용자 정의 코드 변환
- SplitFields

이러한 변환 후 작업 다이어그램에 SelectFromCollection 변환 노드를 추가하지 않으면 작업에 대한 오류가 발생합니다.

이 변환의 상위 노드는 DynamicFrames의 컬렉션을 반환하는 노드여야 합니다. Join 변환과 같이 단일 DynamicFrame을 반환하는 이 변환 노드의 상위 항목을 선택하면 작업에서 오류를 반환합니다.

마찬가지로, 단일 DynamicFrame을 입력으로 예상하는 변환의 상위 항목으로 작업 다이어그램에서 SelectFromCollection 노드를 사용하는 경우 작업에서 오류를 반환합니다.

Node parents

Select which node(s) will provide inputs for this one

Split Fields ✕
 SplitFields - Transform

⚠ Parent node Split Fields outputs a collection, but node Drop Fields does not accept a collection.

SelectFromCollection을 사용하여 유지할 데이터 집합 선택

SelectFromCollection 변환을 사용하여 DynamicFrames의 컬렉션을 단일 DynamicFrame으로 변환합니다.

작업 다이어그램에 SelectFromCollection 변환 노드를 추가하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 SelectFromCollection을 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. [변환(Transform)] 탭을 선택합니다.
4. [프레임 인덱스(Frame index)] 머리글 아래에서 DynamicFrames의 컬렉션에서 선택하려는 DynamicFrame에 해당하는 배열 인덱스 번호를 선택합니다.

예를 들어 이 변환의 상위 노드가 SplitFields 변환인 경우 해당 노드의 [출력 스키마(Output schema)] 탭에서 각 DynamicFrame에 대한 스키마를 볼 수 있습니다. [출력 2(Output 2)]의 스키마와 연결된 DynamicFrame을 유지하려면 목록의 두 번째 값인 [프레임 인덱스(Frame index)] 값으로 **1**을 선택합니다.

선택하는 DynamicFrame만 출력에 포함됩니다.

5. (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보

(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.

6. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

데이터 집합에서 누락된 값 찾기 및 채우기

FillMissingValues 변환을 사용하여 데이터 집합에서 누락된 값이 있는 레코드를 찾고 대체에 의해 결정된 값으로 새 필드를 추가할 수 있습니다. 입력 데이터 집합은 누락 값을 결정하는 기계 학습 모델을 훈련하는 데 사용됩니다. 증분 데이터 집합을 사용하는 경우 각 증분 집합은 기계 학습 모델의 훈련 데이터로 사용되므로 결과가 정확하지 않을 수 있습니다.

작업 다이어그램에서 FillMissingValues 변환 노드를 사용하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 FillMissingValues를 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. [변환(Transform)] 탭을 선택합니다.
4. [데이터 필드(Data field)]에서 누락된 값을 분석할 소스 데이터의 열 또는 필드 이름을 선택합니다.
5. (선택 사항) [새 필드 이름(New field name)] 필드에 분석된 필드의 예상 대체 값을 보유할 각 레코드에 추가된 필드의 이름을 입력합니다. 분석된 필드에 누락된 값이 없으면 분석된 필드의 값이 새 필드에 복사됩니다.

새 필드의 이름을 지정하지 않으면 기본 이름은 `_filled`가 추가된 분석된 열의 이름입니다. 예를 들어 [데이터 필드(Data field)]에 **Age**를 입력하고 [새 필드 이름(New field name)]에 값을 지정하지 않으면 **Age_filled**라는 새 필드가 각 레코드에 추가됩니다.

6. (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.

7. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

데이터 집합 내의 키 필터링

Filter 변환을 사용하여 정규식을 기반으로 입력 데이터 집합의 레코드를 필터링하여 새 데이터 집합을 생성합니다. 필터 조건을 만족하지 않는 행은 출력에서 제거됩니다.

- 문자열 데이터 유형의 경우 키 값이 지정된 문자열과 일치하는 행을 필터링할 수 있습니다.
- 숫자 데이터 유형의 경우 비교 연산자 <, >, =, !=, <= 및 >=로 키 값을 지정된 값과 비교하여 행을 필터링할 수 있습니다.

여러 필터 조건을 지정하면 기본적으로 AND 연산자를 사용하여 결과가 결합되지만 대신 OR를 선택할 수 있습니다.

Filter 변환은 대/소문자를 구분합니다. 대/소문자를 구분하지 않는 속성 키 이름이 필요한 경우 ApplyMapping 변환을 상위 노드로 추가합니다.

작업 다이어그램에 Filter 변환 노드를 추가하려면

1. (선택 사항) 필요한 경우 리소스 패널을 열고 필터를 선택하여 작업 다이어그램에 새 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. [변환(Transform)] 탭을 선택합니다.
4. [전역 AND(Global AND)] 또는 [전역 OR(Global OR)]를 선택합니다. 이에 따라 여러 필터 조건이 결합되는 방법이 결정됩니다. 모든 조건은 AND 또는 OR 연산을 사용하여 결합됩니다. 필터 조건이 하나만 있는 경우 둘 중 하나를 선택할 수 있습니다.
5. [필터 조건(Filter condition)] 섹션에서 [조건 추가(Add condition)] 버튼을 선택하여 필터 조건을 추가합니다.

[키(Key)] 필드에서 데이터 집합의 속성 키 이름을 선택합니다. [연산(Operation)] 필드에서 비교 연산자를 선택합니다. [값(Value)] 필드에 비교 값을 입력합니다. 다음은 필터 조건의 몇 가지 예입니다.

- `year >= 2018`
- `State matches 'CA*'`

문자열 값을 필터링할 때 비교 값이 작업 속성(Python 또는 Scala)에서 선택한 스크립트 언어와 일치하는 정규식 포맷을 사용하는지 확인합니다.

- 필요에 따라 필터 조건을 추가합니다.
- (선택 사항) 변환 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 수정된 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.
- (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

DropNullField를 사용하여 Null 값이 포함된 필드 제거

필드의 모든 값이 'null'인 경우 데이터 집합에서 필드를 제거하려면 DropNullFields 변환을 사용합니다. 기본값으로 AWS Glue Studio는 Null 객체를 인식하지만 빈 문자열, 'null'인 문자열, -1 정수 또는 0과 같은 다른 자리 표시자 등의 일부 값은 자동으로 Null로 인식되지 않습니다.

DropNullFields를 사용하려면

- 작업 다이어그램에 DropNullFields 노드를 추가합니다.
- 노드 속성(Node properties) 탭에서 Null 값을 나타내는 추가 값을 선택합니다. 값을 선택하지 않거나 모두 선택할 수 있습니다.

Node properties
Transform
Output schema
Data preview
⌵

DropNullFields Info

Remove fields or columns where all the values are the null objects.

Choose additional values that represent a null value below.

Empty String (" or ")

"null" String

-1 Integer

Add custom null values

Specify custom null values by entering the value and choosing the datatype.

▼

Add new value

- 빈 문자열(" 또는 ") - 빈 문자열이 포함된 필드가 제거됩니다.
 - 'Null 문자열' - 'null'이라는 단어를 포함하는 문자열이 포함된 필드가 제거됩니다.
 - -1 정수 - -1(음수 1) 정수가 포함된 필드가 제거됩니다.
3. 필요한 경우 사용자 지정 Null 값을 지정할 수도 있습니다. 이러한 Null 값은 해당 데이터 집합에 고유할 수 있습니다. 사용자 지정 Null 값을 추가하려면 새 값 추가(Add new value)를 선택합니다.
 4. 사용자 지정 Null 값을 입력합니다. 예를 들어 0이거나 데이터 집합에서 Null을 나타내는 데 사용되는 임의 값일 수 있습니다.
 5. 드롭다운 필드에서 데이터 유형을 선택합니다. 데이터 유형은 문자열 또는 정수일 수 있습니다.

i Note

필드가 Null 값으로 인식되고 필드가 제거되려면 사용자 지정 Null 값과 해당 데이터 유형이 정확히 일치해야 합니다. 사용자 지정 null 값만 일치하고 데이터 형식은 일치하지 않는 부분 일치의 경우 필드가 제거되지 않습니다.

SQL 쿼리를 사용하여 데이터 변환

SQL 변환을 사용하여 SQL 쿼리 형식으로 고유한 변환을 작성할 수 있습니다.

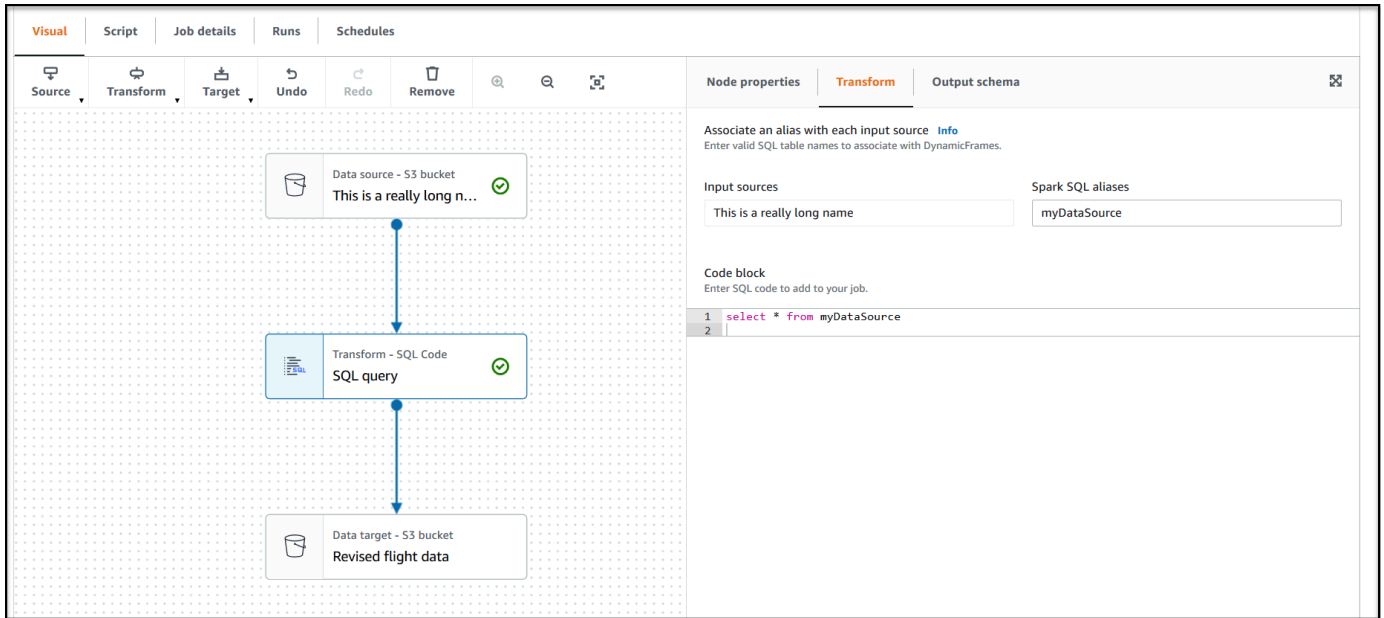
SQL 변환 노드는 여러 데이터 집합을 입력으로 가질 수 있지만 출력으로 단일 데이터 집합만 생성합니다. Apache SparkSQL 쿼리를 입력하는 텍스트 필드가 있습니다. 단순히 SQL 쿼리를 옮기 위해 입력으로 사용되는 각 데이터 집합에 별칭을 할당할 수 있습니다. SQL 구문에 대한 자세한 내용은 [Spark SQL 설명서](#)를 참조하세요.

Note

VPC에 위치한 데이터 원본으로 Spark SQL 변환을 사용하는 경우 데이터 원본이 포함된 VPC에 AWS Glue VPC 엔드포인트를 추가합니다. 개발 엔드포인트 구성에 대한 자세한 내용은 AWS Glue Developer Guide의 [Adding a Development Endpoint](#), [Setting Up Your Environment for Development Endpoints](#), and [Accessing Your Development Endpoint](#)를 참조하세요.

작업 다이어그램에서 SQL 변환 노드를 사용하려면

1. (선택 사항) 필요한 경우 작업 다이어그램에 변환 노드를 추가합니다. 노드 유형으로 [Spark SQL]을 선택합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않았거나 SQL 변환에 대해 여러 입력을 원하는 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다. 필요에 따라 상위 노드를 추가합니다.
3. 노드 세부 정보 패널에서 [변환(Transform)] 탭을 선택합니다.
4. SQL 쿼리의 소스 데이터 집합은 각 노드의 [이름(Name)] 필드에 지정한 이름으로 식별됩니다. 이러한 이름을 사용하지 않으려는 경우나 이름이 SQL 쿼리에 적합하지 않은 경우 각 데이터 집합에 이름을 연결할 수 있습니다. 콘솔은 MyDataSource와 같은 기본 별칭을 제공합니다.



예를 들어, SQL 변환 노드의 상위 노드 이름이 Rename Org PK field인 경우 org_table이라는 이름을 이 데이터 집합과 연결할 수 있습니다. 그런 다음 이 별칭을 노드 이름 대신 SQL 쿼리에서 사용할 수 있습니다.

5. 머리글 [코드 블록(Code block)] 아래의 텍스트 입력 필드에 SQL 쿼리를 붙여넣거나 입력합니다. 텍스트 필드에는 SQL 구문 강조 표시 및 키워드 제안 사항이 표시됩니다.
6. SQL 변환 노드를 선택한 상태에서 [출력 스키마(Output schema)] 탭을 선택한 다음 [편집(Edit)]을 선택합니다. SQL 쿼리의 출력 필드를 설명하는 열과 데이터 유형을 제공합니다.

페이지의 [출력 스키마(Output schema)] 섹션에서 다음 작업을 사용하여 스키마를 지정합니다.

- 열 이름을 바꾸려면 열의 [키(Key)] 텍스트 상자(필드 또는 속성 키라고도 함)에 커서를 놓고 새 이름을 입력합니다.
- 열의 데이터 유형을 변경하려면 드롭다운 목록에서 열의 새 데이터 유형을 선택합니다.
- 스키마에 새 최상위 열을 추가하려면 오버플로 (...) 버튼을 선택한 다음 [루트 키 추가(Add root key)]를 선택합니다. 새 열이 스키마 맨 위에 추가됩니다.
- 스키마에서 열을 제거하려면 키 이름의 맨 오른쪽에 있는 삭제 아이콘 (X) 을 선택합니다.

7. 출력 스키마 지정을 마치면 [적용(Apply)]을 선택하여 변경 사항을 저장하고 스키마 편집기를 종료합니다. 변경 사항을 저장하지 않으려면 [취소(Cancel)]를 선택하여 스키마 편집기를 편집합니다.


8. (선택 사항) 노드 속성과 변환 속성을 구성한 후 노드 세부 정보 패널에서 [데이터 미리 보기(Data preview)] 탭을 선택하여 수정된 데이터 집합을 미리 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. 이 기능 사용과 관련된 비용이 있으며 IAM 역할을 제공하는 즉시 결제가 시작됩니다.

집계를 사용하여 선택한 필드에서 요약 계산 수행

집계(Aggregate) 변환을 사용하려면

1. 작업 다이어그램에 집계(Aggregate) 노드를 추가합니다.
2. 노드 속성(Node properties) 탭에서 드롭다운 필드를 선택하여 그룹화할 필드를 선택합니다(선택 사항). 한 번에 둘 이상의 필드를 선택하거나 검색 창에 입력하여 필드 이름을 검색할 수 있습니다.

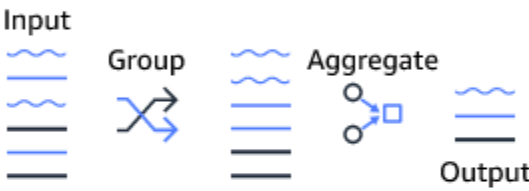
필드를 선택하면 이름과 데이터 유형이 표시됩니다. 필드를 제거하려면 필드에서 'X'를 선택합니다.

Node properties | **Transform** 1 | Output schema | 

Data preview

▼ Aggregate [Info](#)

This transform first groups your rows by fields you choose, and then computes the aggregated value for fields you choose by specific function (e.g., sum, average, max).



Fields to group by - optional

Select the fields you would like to group your rows by, so the aggregation would be done for each unique group.


Choose one or more fields ▼

Aggregate another column

 **Add an aggregation.**

3. 다른 열 집계(Aggregate another column)를 선택합니다. 하나 이상의 필드를 선택해야 합니다.

Field to aggregate Aggregation function [Info](#)

Choose a field ▼ *Choose a function* ▼ 

Aggregate another column

4. 집계할 필드(Field to aggregate) 드롭다운에서 필드를 선택합니다.
5. 선택한 필드에 적용할 집계 함수를 선택합니다.

- avg - 평균을 계산합니다.
- countDistinct - Null이 아닌 고유 값 수를 계산합니다.
- count - Null이 아닌 값 수를 계산합니다.
- first - 'group by' 기준을 충족하는 첫 번째 값을 반환합니다.
- last - 'group by' 기준을 충족하는 마지막 값을 반환합니다.
- kurtosis - 빈도 분포 곡선의 정점 첨도를 계산합니다.
- max - 'group by' 기준을 충족하는 최대값을 반환합니다.
- min - 'group by' 기준을 충족하는 최소값을 반환합니다.
- skewness - 정규 분포의 확률 분포 비대칭 측정값입니다.
- stddev_pop - 모집단 표준 편차를 계산하고 모집단 분산의 제곱근을 반환합니다.
- sum - 그룹에 있는 모든 값의 합계입니다.
- sumDistinct - 그룹에 있는 고유 값의 합계입니다.
- var_samp - 그룹의 표본 분산입니다(Null 무시).
- var_pop - 그룹의 모집단 분산입니다(Null 무시).

중첩된 구조체 평면화

데이터에 있는 중첩 구조체의 필드를 평면화하여 최상위 필드가 되도록 합니다. 새 필드의 이름은 필드 이름 앞에 해당 필드에 전달될 구조체 필드 이름을 넣고 점으로 구분하여 지정합니다.

예를 들어, 데이터에 이름이 'phone_number'인 Struct 유형의 필드가 있고, 다른 필드에는 'country_code'와 'number'라는 두 개의 필드가 있는 'home_phone'이라는 'Struct' 유형의 필드가 있는 경우를 예로 들어 보겠습니다. 일단 평면화되면 이 두 필드는 각각 'phone_numbers.home_phone.country_code' 및 'phone_numbers.home_phone.number'라는 이름의 최상위 필드가 됩니다.

작업 다이어그램에 Flatten 변환 노드를 추가하려면

1. 리소스 패널을 열고 변환 탭을 선택한 후 평면화를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 검색 표시줄을 사용하여 'Flatten'을 입력하고 평면화 노드를 클릭할 수도 있습니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.

2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. (선택 사항) 변환 탭에서 최대 중첩 수준을 평면화하도록 제한할 수 있습니다. 예를 들어 이 값을 1로 설정하면 최상위 구조체만 평면화됩니다. 최대값을 2로 설정하면 최상위 레벨과 그 바로 아래에 있는 구조체가 평면화됩니다.

UUID 열 추가

UUID (범용 고유 식별) 열을 추가하면 각 행에 고유한 36자 문자열이 할당됩니다.

작업 다이어그램에 UUID 변환 노드를 추가하려면

1. 리소스 패널을 열고 UUID를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.

2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. (선택 사항) 변환 탭에서 새 열의 이름을 사용자 지정할 수 있습니다. 기본적으로 이름은 'uuid'로 지정됩니다.

식별자 열 추가

데이터 세트의 각 행에 숫자 식별자를 할당합니다.

작업 다이어그램에서 식별자 변환 노드를 추가하려면

1. 리소스 패널을 열고 식별자를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. (선택 사항) 변환 탭에서 새 열의 이름을 사용자 지정할 수 있습니다. 기본적으로 이름은 'id'로 지정됩니다.
4. (선택 사항) 작업에서 데이터를 증분 방식으로 처리하고 저장하는 경우 작업 실행 간에 동일한 ID가 재사용되지 않도록 해야 합니다.

변환 탭에서 고유 확인란 옵션을 선택합니다. 식별자에 작업 타임스탬프가 포함되므로 여러 번 실행할 때 고유하게 표시됩니다. 더 큰 숫자를 입력하도록 허용하려면 long을 입력하는 대신 열을 십진수로 입력합니다.

열을 타임스탬프 형식으로 변환

변환 대상인 타임스탬프로 변환을 사용하여 숫자 또는 문자열 열의 데이터 유형을 타임스탬프로 변경하여 해당 데이터 유형과 함께 저장하거나 타임스탬프가 필요한 다른 변환에 적용할 수 있습니다.

작업 다이어그램에 타임스탬프로 변환 변환 노드를 추가하려면

1. 리소스 패널을 열고 타임스탬프로 변환을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.

2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 변환할 열의 이름을 입력합니다.
4. 변환 탭에서 유형을 선택하여 선택한 열의 구문을 분석하는 방법을 정의합니다.

값이 숫자인 경우 초(Unix/Python 타임스탬프), 밀리초 또는 마이크로초로 표현할 수 있습니다. 해당 옵션을 선택하세요.

값이 형식이 지정된 문자열인 경우 'iso' 유형을 선택합니다. 문자열은 ISO 형식의 변형 중 하나를 준수해야 합니다(예: '2022-11-02T 14:40:59 .915Z').

이 시점에서 유형을 모르거나 행마다 다른 유형을 사용하는 경우 'autodetect'를 선택하면 시스템이 성능 저하를 줄이면서 최선의 추정을 할 수 있습니다.

5. (선택 사항) 변환 탭에서 선택한 열을 변환하는 대신 새 열을 만들고 새 열의 이름을 입력하여 원본을 유지할 수 있습니다.

타임스탬프 열을 형식이 지정된 문자열로 변환

타임스탬프 열을 패턴에 따라 문자열로 형식을 지정합니다. 타임스탬프 형식 지정을 사용하여 날짜 및 시간을 원하는 형식의 문자열로 가져올 수 있습니다. [Spark 날짜 구문](#)과 대부분의 [Python 날짜 코드](#)를 사용하여 형식을 정의할 수 있습니다.

예를 들어, 날짜 문자열을 '2023-01-01 00:00'과 같은 형식으로 지정하려면 Spark 구문을 사용하여 'YYYYY-MM-DD HH:mm' 같은 형식으로 정의하거나 이에 상응하는 Python 날짜 코드를 '%Y-%m-%d %H: %M' 같은 형식으로 정의하면 됩니다.

작업 다이어그램에서 타임스탬프 형식 지정 변환 노드를 추가하려면

1. 리소스 패널을 열고 타임스탬프 형식 지정을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 변환할 열의 이름을 입력합니다.
4. 변환 탭에서 [Spark 날짜 구문](#) 또는 [Python 날짜 코드](#)를 사용하여 표현된 사용하려는 타임스탬프 형식 패턴을 입력합니다.

5. (선택 사항) 변환 탭에서 선택한 열을 변환하는 대신 새 열을 만들고 새 열의 이름을 입력하여 원본을 유지할 수 있습니다.

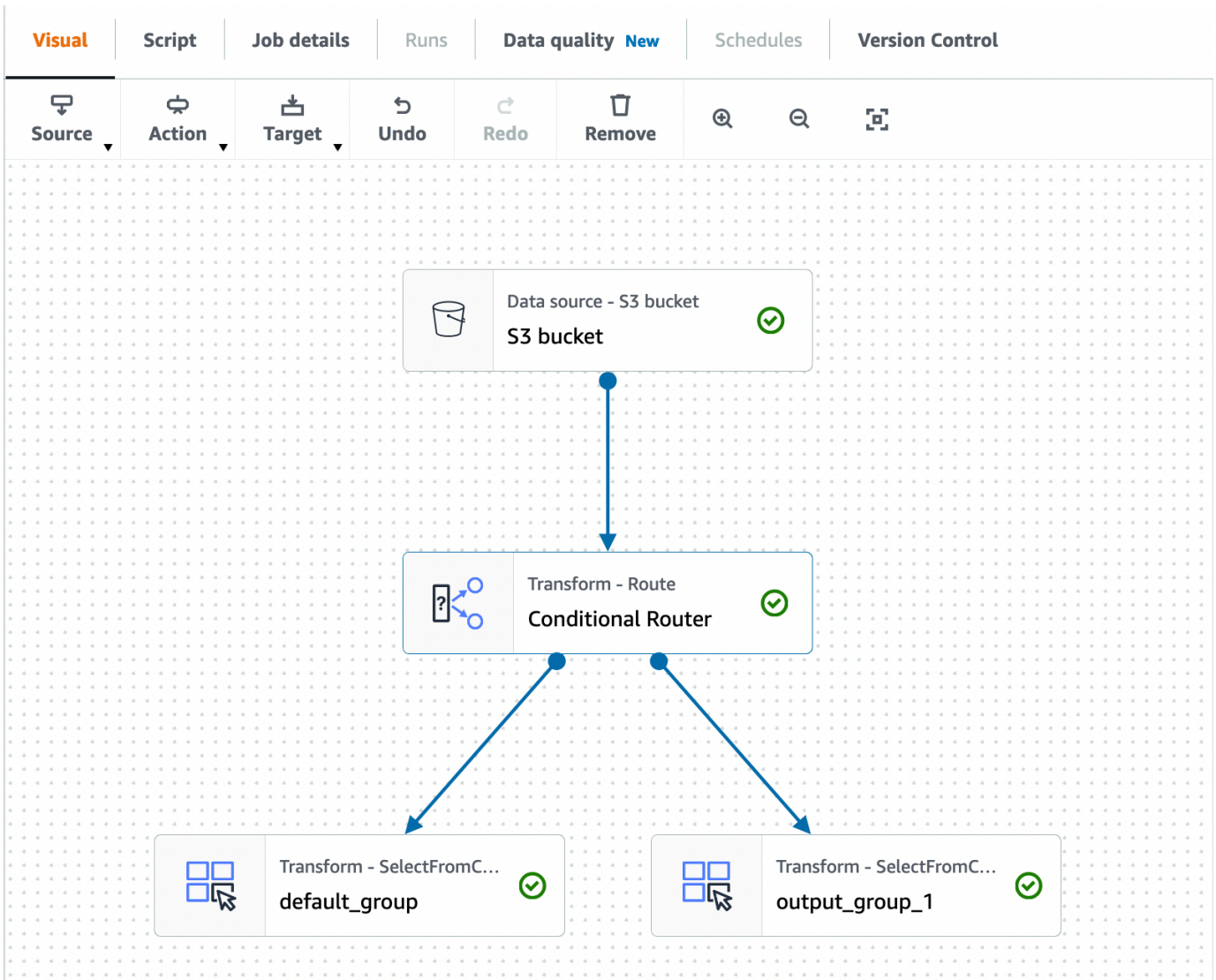
조건부 라우터 변환 생성

조건부 라우터 변환을 사용하면 수신 데이터에 여러 조건을 적용할 수 있습니다. 수신 데이터의 각 행은 그룹 필터 조건을 기준으로 평가되고 해당 그룹으로 처리됩니다. 행이 둘 이상의 그룹 필터 조건을 충족하는 경우 변환은 행을 여러 그룹에 전달합니다. 조건을 충족하지 않는 행은 삭제되거나 기본 출력 그룹으로 라우팅될 수 있습니다.

이 변환은 필터 변환과 유사하지만 여러 조건에서 동일한 입력 데이터를 테스트하려는 사용자에게 유용합니다.

조건부 라우터 변환을 추가하려면:

1. 조건부 라우터 변환을 수행할 노드를 선택합니다. 소스 노드일 수도 있고 다른 변환일 수도 있습니다.
2. 작업을 선택한 다음 검색 창을 사용하여 '조건부 라우터'를 찾아 선택합니다. 조건부 라우터 변환이 두 개의 출력 노드와 함께 추가됩니다. 한 출력 노드인 '기본 그룹'에는 다른 출력 노드에 정의된 조건을 전혀 충족하지 않는 레코드가 포함되어 있습니다. 기본 그룹은 편집할 수 없습니다.



그룹 추가를 선택하여 출력 그룹을 더 추가할 수 있습니다. 각 출력 그룹에 대해 그룹 이름을 지정하고 필터 조건과 논리 연산자를 추가할 수 있습니다.

Node properties
Transform
Output schema
Data preview
✕

Add group

output_group_1

Define a set of conditions a record has to meet in order to be routed to the output group.

Remove group

Group name
The name of this output group, as it would appear in your job. Letters, numbers, _ and - are allowed.

output_group_1

Logical operator

AND
Trigger only when ALL conditions are met.

OR
Trigger when at least one of the conditions is met.

Filter condition [Info](#)
Specify your filter condition by choosing the key, operator, and entering a value.

Start by adding a filter condition.

Add condition


Default group
Records which do not meet any of the conditions defined above will be routed here.

3. 그룹의 새 이름을 입력하여 출력 그룹 이름을 바꿉니다. AWS Glue Studio는 자동으로 사용자에게 대한 그룹 이름을 지정합니다(예: 'output_group_1').
4. 논리 연산자(AND, OR)를 선택하고 키, 연산 및 값을 지정하여 필터 조건을 추가합니다. 논리 연산자를 사용하면 둘 이상의 필터 조건을 구현하고 지정한 각 필터 조건에 대해 논리 연산자를 수행할 수 있습니다.

키를 지정할 때 스키마에서 사용 가능한 키를 선택할 수 있습니다. 그런 다음 선택한 키 유형에 따라 사용할 수 있는 연산을 선택할 수 있습니다. 예를 들어 키 유형이 '문자열'인 경우 선택할 수 있는 연산은 '일치'입니다.

Filter condition **Info**

Specify your filter condition by choosing the key, operator, and entering a value.

Key	Operation	Value	
year ▼	= ▼	2023	
Add condition			

- 값 필드에 값을 입력합니다. 조건 추가를 선택하여 추가 필터 조건을 추가할 수 있습니다. 필터 조건을 제거하려면 휴지통 아이콘을 선택합니다.


열 연결 변환을 사용하여 열 추가

연결 변환을 사용하면 선택적 스페이서가 있는 다른 열의 값을 사용하여 새 문자열 열을 구축할 수 있습니다. 예를 들어 '-'를 스페이서로 사용해 'year', 'month', 'day'가 순서대로 연결된 항목으로 연결된 열 'date'를 정의하면 다음과 같은 결과가 나옵니다.

day	개월	년	date
01	01	2020	2020-01-01
02	01	2020	2020-01-02
03	01	2020	2020-01-03
04	01	2020	2020-01-04

연결 변환을 추가하려면:

- 리소스 패널을 엽니다. 그런 다음 열 연결을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
- (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
- 변환 탭에서 연결된 문자열 및 연결할 열을 보관할 열의 이름을 입력합니다. 드롭다운에서 열을 선택하는 순서대로 사용됩니다.

Node properties	Transform	Output schema	Data preview	
<p>Name of the concatenated column Name of the string column that will be generated</p> <input type="text"/>				
<p>List of column named separated by comma or spaces The fields listed will be concatenated on that order</p> <input type="text"/>				
<p>Array new column Name - optional String to place between the concatenated fields, by default there is no spacer.</p> <input type="text"/>				
<p>Null value - optional The string to use when a column value is null, for example: 'NULL' or 'NA', by default an empty string will be used</p> <input type="text"/>				

- 스페이스 - 선택 사항 - 연결된 필드 사이에 배치할 문자열을 입력합니다. 기본적으로 스페이스는 없습니다.
- Null 값 - 선택 사항 - 열 값이 null일 때 사용할 문자열을 입력합니다. 기본적으로 열의 값이 'NULL' 또는 'NA'인 경우에는 빈 문자열이 사용됩니다.

문자열 분할 변환을 사용하여 문자열 열 구분


문자열 분할 변환을 사용하면 정규식을 사용하여 문자열을 토큰 배열로 분할해 분할 방식을 정의할 수 있습니다. 그런 다음 각 토큰의 의미를 미리 알고 있다고 가정하고 열을 배열 유형으로 유지하거나 배열에서 열로 변환을 이 다음에 적용하여 배열 값을 상위 수준의 필드로 추출할 수 있습니다. 또한 토큰 순서가 무관한 경우(예: 카테고리 세트) 분해 변환을 사용하여 각 값에 대해 별도의 행을 생성할 수 있습니다.

예를 들어 쉼표를 패턴으로 사용하여 'categories' 열을 분할해 'categories_arr' 열을 추가할 수 있습니다.

product_id	categories	categories_arr
1	sports,winter	[sports, winter]
2	garden,tools	[garden, tools]
3	videogames	[videogames]
4	game,boardgame,social	[game, boardgame, social]

문자열 분할 변환을 추가하려면:

1. 리소스 패널을 열고 문자열 분할을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 분할할 열을 선택하고 문자열을 분할하는 데 사용할 패턴을 입력합니다. 일반 표현식 처럼 특별한 의미가 있어서 이스케이프 처리해야 하는 경우가 아니라면 대부분의 경우 문자만 입력하면 됩니다. 문자 앞에 백슬래시를 추가하여 이스케이프 처리해야 하는 문자는 \. [] {} () <> * + -= ! ? ^ \$ | 입니다. 예를 들어 점('.')으로 구분하려면 \. 를 입력해야 합니다. 하지만 쉼표는 특별한 의미가 없으므로 , 그대로 지정할 수 있습니다.

Node properties | **Transform** | Output schema | Data preview 

Column to split

Column whose string will be split into an string array

Splitting regular expression

Regex defining the separator token, examples: ',', '\|' (pipe needs to be escaped) or '\s+' (whitespace split)

Array column Name - optional

Name to use for the column with the extracted array resulting of the split. If not specified, instead of a new column the existing one is replaced

- 4. (선택 사항) 원래 문자열 열을 유지하려는 경우 새 배열 열의 이름을 입력할 수 있습니다. 이렇게 하면 원래 문자열 열과 토큰화된 새 배열 열을 모두 유지할 수 있습니다.

배열을 열로 변환을 사용하여 배열 요소를 상위 수준 열로 추출

배열을 열로 변환을 사용하면 배열 유형의 열에 있는 일부 또는 모든 요소를 새 열로 추출할 수 있습니다. 배열에 추출하기에 충분한 값이 있는 경우 변환은 새 열을 최대한 많이 채우며, 선택적으로 지정된 위치의 요소를 가져올 수도 있습니다.

예를 들어 ip v4 서브넷에서 '문자열 분할' 변환을 적용한 결과에 해당하는 배열 열 'subnet'이 있는 경우 첫 번째 위치와 네 번째 위치를 새 열 'first_octect' 및 'forth_octect'로 추출할 수 있습니다. 이 예제에서 변환 출력은 다음과 같습니다(마지막 두 행의 배열이 예상보다 짧음).

서브넷	first_octect	fourth_octect
[54, 240, 197, 238]	54	238
[192, 168, 0, 1]	192	1
[192, 168]	192	
[]		

배열을 열로 변환을 추가하려면:

1. 리소스 패널을 열고 배열을 열로 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 추출할 배열 열을 선택하고 추출된 토큰의 새 열 목록을 입력합니다.

Node properties
Transform
Output schema
Data preview
✕

Array type column

Column of type array from which the new columns are extracted

Output columns

The names (separated by commas) of the columns to create out of the array fields. The data type will be the same as the array. For each row, the transform will try to fill them as much as possible using the array elements, the rest will be NULL

Array indexes to use - optional

List of array positions (starting from 1 and separated by commas), indicating which columns to take to fill the columns. Only need to set this if you want to skip some positions of the array

4. (선택 사항) 열에 할당하기 위해 배열 토큰을 사용하지 않으려면 가져올 인덱스를 지정할 수 있습니다. 이 인덱스는 지정된 동일한 순서대로 열 목록에 할당됩니다. 예를 들어 출력 열이 'column1, column2, column3'이고 인덱스가 4, 1, 3'인 경우 배열의 네 번째 요소는 column1로, 첫 번째 요소는 column2로, 세 번째 요소는 column3으로 이동합니다(배열이 인덱스 수보다 적으면 NULL 값이 설정됨).

현재 타임스탬프 추가 변환 사용

현재 타임스탬프 추가 변환을 사용하면 데이터가 처리된 시간으로 행을 표시할 수 있습니다. 이는 감사 목적이나 데이터 파이프라인에서 지연 시간을 추적하는 데 유용합니다. 이 새 열을 타임스탬프 데이터 형식이나 형식이 지정된 문자열로 추가할 수 있습니다.

현재 타임스탬프 추가 변환을 추가하려면:

1. 리소스 패널을 열고 현재 타임스탬프 추가를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.

Node properties
Transform
Output schema
Data preview
✕

Timestamp column - optional
 Name to use for the new column, by default: timestamp. With type "string" if a dataFormat is specified, otherwise "timestamp"

Timestamp format - optional
 Optional pattern to format as a string, accepts most Python date format codes, such as '%Y-%m-%d %H:%M:%S'; as well as Spark patterns such as 'yyyy-MM-dd'T'HH:mm:ss.SSSZ'

3. (선택 사항) 변환 탭에서 새 열의 사용자 지정 이름을 입력하고 형식이 지정된 날짜 문자열로 열을 구성하려면 형식을 입력합니다.

행을 열로 피벗 변환 사용

행을 열로 피벗 변환을 사용하면 선택한 열에서 고유 값을 교체하여 숫자 열을 집계할 수 있으며, 이 열은 새 열이 됩니다(여러 개의 열을 선택한 경우 값이 연결되어 새 열의 이름으로 지정됨). 이렇게 하면 행이 통합되지만 각 고유 값에 대한 부분 집계를 통해 열이 더 많아집니다. 예를 들어 다음과 같이 month 및 country별 매출 데이터 세트가 있습니다(쉽게 설명하기 위해 정렬됨).

년	개월	country	amount
2020	Jan	uk	32
2020	Jan	de	42
2020	Jan	us	64

년	개월	country	amount
2020	Feb	uk	67
2020	Feb	de	4
2020	Feb	de	7
2020	Feb	us	6
2020	Feb	us	12
2020	Jan	us	90

금액 및 국가를 집계 열로 사용하여 피벗하면 원래 국가 열에서 새 열이 생성됩니다. 아래 표에는 국가 열 대신 de, uk 및 us에 대한 새 열이 있습니다.

년	개월	de	uk	us
2020	Jan	42	32	64
2020	Jan	11	67	18
2021	Jan			90

대신 month와 country를 모두 피벗하려는 경우 해당 열 값의 모든 조합에 대한 열이 생성됩니다.

년	Jan_de	Jan_uk	Jan_us	Feb_de	Feb_uk	Feb_us
2020	42	32	64	11	67	18
2021			90			

행을 열로 피벗 변환을 추가하려면:

1. 리소스 패널을 열고 행을 열로 피벗을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.

2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 새 열의 값을 생성하기 위해 집계될 숫자 열, 적용할 집계 함수, 고유한 값을 새 열로 변환할 열을 선택합니다.

Node properties
Transform
Output schema
Data preview

Aggregation column

Numeric column on which the aggregation function is applied

Aggregation

The Spark function to apply to the aggregation column.

Columns to convert

List of columns whose values will become new columns. If multiple columns are specified, the values are concatenated using underscore.

Choose options

열을 행으로 피벗 취소 변환 사용

피벗 취소 변환을 사용하면 열을 새 열의 값으로 변환하여 각 고유 값에 대한 행을 생성할 수 있습니다. 피벗과 반대이지만 원래 열로 결합을 분할하거나 집계된 동일한 값의 행을 구분할 수 없다는 점에서 차이가 있습니다(나중에 분할 변환을 사용하여 분할 가능). 예를 들어 다음과 같은 테이블이 있습니다.

년	개월	de	uk	us
2020	Jan	42	32	64
2020	Feb	11	67	18
2021	Jan			90

'amount' 값을 사용해 'de', 'uk', 'us' 열을 'country' 열로 피벗 취소하고 다음과 같은 결과를 얻을 수 있습니다(설명을 위해 여기에서는 정렬됨).

년	개월	country	amount
2020	Jan	uk	32
2020	Jan	de	42
2020	Jan	us	64
2020	Feb	uk	67
2020	Feb	de	11
2020	Feb	us	18
2021	Jan	us	90

값이 NULL인 열(2021년 1월의 'de' 및 'uk')은 기본적으로 생성되지 않습니다. 이 옵션을 활성화하여 다음을 얻을 수 있습니다.

년	개월	country	amount
2020	Jan	uk	32
2020	Jan	de	42
2020	Jan	us	64
2020	Feb	uk	67
2020	Feb	de	11
2020	Feb	us	18
2021	Jan	us	90
2021	Jan	de	

년	개월	country	amount
2021	Jan	uk	

열을 행으로 피벗 취소 변환을 추가하려면:

1. 리소스 패널을 열고 열을 행으로 피벗 취소를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 피벗 취소할 열의 이름과 값을 보관하기 위해 생성할 새 열을 입력합니다.

Node properties
Transform
Output schema
Data preview

Unpivot names column

Column to create out of the source columns names

Unpivot values column

Column to create out of values of the old columns

Columns to unpivot into the new value column

List of columns whose name will become values of the new column


Choose options
▼

처리 균형 자동 조절 변환을 사용하여 런타임 최적화

처리 균형 자동 조절 변환은 더 나은 성능을 위해 작업자 사이에서 데이터를 재배포합니다. 이는 데이터가 불균형하거나 소스에서 가져온 데이터로 인해 충분한 병렬 처리가 불가능한 경우에 유용합니다. 소스가 gzip으로 압축되었거나 JDBC인 경우에 이러한 상황이 흔히 나타납니다. 데이터 재배포에도 어느 정도의 성능 비용이 발생하므로, 데이터의 균형이 이미 조절된 상태인 경우 최적화해도 보상 효과를 얻지 못할 수도 있습니다. 이 변환은 Apache Spark 재파티셔닝을 사용하여 클러스터 용량에 최적화된 여러 파티션 사이에서 데이터를 임의로 재할당합니다. 고급 사용자의 경우 여러 파티션을 수동으로 입

력할 수 있습니다. 또한 지정된 열을 기준으로 데이터를 재구성하여 파티셔닝된 테이블의 쓰기를 최적화하는 데 사용할 수 있습니다. 그러면 출력 파일의 통합 기능이 더욱 강화됩니다.

1. 리소스 패널을 열고 처리 균형 자동 조절을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. (선택 사항) 변환 탭에서 파티션 수를 입력할 수 있습니다. 일반적으로 이 값은 시스템에서 결정하도록 하는 것이 좋지만, 이 값을 제어해야 하는 경우 승수를 조정하거나 특정 값을 입력할 수 있습니다. 열을 기준으로 파티셔닝된 데이터를 저장하려는 경우 리파티션 열과 동일한 열을 선택할 수 있습니다. 이렇게 하면 각 파티션에서 파일 수를 최소화하고 파티션당 파일 수가 많아지지 않도록 방지할 수 있습니다. 파일 수가 많아지면 해당 데이터를 쿼리하는 도구의 성능이 저하될 수 있습니다.

Node properties	Transform	Output schema	Data preview	
<p>Number of partitions - optional</p> <p>Number of partitions on which to randomly distribute the data. If the number ends with the x letter then it means it's a multiple of the number of cores in the cluster. By default: 2x</p> <input type="text"/>				
<p>Repartition columns - optional</p> <p>Instead of randomly reassign the data to partitions, assign data with the same values of the columns specified to the same partition.</p> <input type="text" value="Choose options"/>				

파생 열 변환을 사용하여 다른 열 결합

파생 열 변환을 사용하면 상수 및 리터럴뿐만 아니라 데이터의 다른 열을 사용할 수 있는 수학적 공식 또는 SQL 표현식을 기반으로 새 열을 정의할 수 있습니다. 예를 들어 'success' 및 'count' 열에서 'percentage' 열을 파생하려면 SQL 표현식 'success * 100 / count || "%'"를 입력하면 됩니다.

결과 예제:

success	count	percentage
14	100	14%
6	20	3%
3	40	7.5%

파생 열 변환을 추가하려면:

1. 리소스 패널을 열고 파생 열을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 열 이름 및 해당 콘텐츠에 대한 표현식을 입력합니다.

Node properties
Transform
Output schema
Data preview
✕

Name of the derived column
Name to use for the new column or replace an existing one

SQL Expression
A SQL expression that defines the column, which can be derived from other existing columns and use operators to modify or combine them. For instance, to derive a percentage from the columns "success" and "count", you can enter: "success * 100 / count"

조회 변환을 사용하여 카탈로그 테이블에서 일치하는 데이터 추가

조회 변환을 사용하면 키가 데이터에 정의된 조회 열과 일치하는 경우 정의된 카탈로그 테이블의 열을 추가할 수 있습니다. 이는 조건 일치 열을 사용하여 데이터와 조회 테이블 사이에서 왼쪽 외부 조인을 수행하는 방법과 같습니다.

조회 변환을 추가하려면:

1. 리소스 패널을 열고 조회를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 조회를 수행하는 데 사용할 완전히 정규화된 카탈로그 테이블 이름을 입력합니다. 예를 들어 데이터베이스가 'mydb'이고 테이블이 'mytable'인 경우 'mydb.mytable'을 입력합니다. 그런 다음 조회 키가 구성된 경우 조회 테이블에서 일치 항목을 찾을 기준을 입력합니다. 쉼표로 구분된 키 열 목록을 입력합니다. 키 열 중 하나 이상의 이름에서 같은 이름이 없는 경우 일치 매핑을 정의해야 합니다.

예를 들어 데이터 열이 'user_id' 및 'region'이고 사용자 테이블에서 해당 열의 이름이 'id' 및 'region'인 경우 일치시킬 열 필드에 'user_id=id, region'을 입력합니다. region=region을 입력할 수도 있지만 동일하므로 이 작업은 필요하지 않습니다.

4. 마지막으로, 조회 테이블에서 일치하는 행에서 가져올 열을 입력하여 데이터에 통합합니다. 일치하는 항목이 없으면 해당 열은 NULL로 설정됩니다.

Note

조회 변환 아래에서 효율성을 높이기 위해 왼쪽 조인을 사용합니다. 조회 테이블에 복합 키가 있는 경우 하나만 일치하도록 일치하는 열을 모든 키 열과 매칭하도록 설정합니다. 그렇지 않으면 여러 조회 행이 매칭되어 각 일치 항목에 대해 추가 행이 추가됩니다.

Node properties

Transform

Output schema

Data preview

**AWS Glue Data Catalog table**

Qualified name of the catalog table to use for the lookup, specifying the database and table name separated by a dot

Lookup key columns to match

Columns in the lookup table to match separated by commas; if the column names don't match, you can specify the mapping between the data and the lookup table separating the names with an equals sign =

Lookup columns to take

Columns in the lookup table to add to the data when a match is found in the lookup table

배열 또는 맵을 행으로 분해 변환 사용

분해 변환을 사용하면 중첩된 구조에서 조작하기 쉬운 개별 행으로 값을 추출할 수 있습니다. 배열의 경우 이 변환은 행의 다른 열에 대한 값을 복제하여 배열의 각 값에 대해 행을 생성합니다. 맵의 경우 이 변환은 열 및 행의 다른 열을 키와 값으로 사용해 각 항목에 대한 행을 생성합니다.

예를 들어 다음 데이터 세트에는 값이 여러 개인 'category' 배열 열이 있습니다.


product_id	category
1	[sports, winter]
2	[garden, tools]
3	[videogames]
4	[game, boardgame, social]
5	[]

'category' 열을 같은 이름의 열로 분해하면 해당 열이 재정의됩니다. 다음 결과를 얻기 위해 NULL을 포함하도록 선택할 수 있습니다(설명을 위해 정렬됨).

product_id	category
1	sports
1	winter
2	garden
2	tool
3	videogames
4	게임
4	boardgame
4	social
5	

배열 또는 맵을 행으로 분해 변환을 추가하려면:

1. 리소스 패널을 열고 분해 또는 맵을 행으로 분해를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. (선택 사항) 노드 속성 탭에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 분해할 열을 선택합니다(배열 또는 맵 유형이어야 함). 그런 다음 배열 항목에 대한 열 이름 또는 맵을 분해하는 경우 키와 값에 대한 열 이름을 입력합니다.
4. (선택 사항) 변환 탭에서 분해할 열이 NULL이거나 구조가 비어 있는 경우 기본적으로 분해된 데이터 세트에서 해당 열은 생략됩니다. 행을 유지하고 싶으면(새 열을 NULL로 표시) 'NULL 포함'을 선택합니다.

Node properties	Transform	Output schema	Data preview	
-----------------	------------------	---------------	--------------	---

Column to explode
A column of type array or map

New column name
The name of the column to put the array values or the dictionary keys

Values column - optional
If exploding a dictionary, you can specify a name for a column to contain the values. Default name: "value"

Include NULLs - optional
If selected, NULL values will also generate a new rows, otherwise the row with a NULL value is omitted

레코드 일치 변환을 사용하여 기존 데이터 분류 변환을 간접적으로 호출

이 변환은 기존 레코드 일치 기계 학습 데이터 분류 변환을 간접적으로 호출합니다.

변환은 레이블을 기반으로 훈련된 모델을 기준으로 현재 데이터를 평가합니다. 알고리즘 훈련에 따라 동등한 것으로 간주되는 항목 그룹에 각 행을 할당하기 위해 'match_id' 열이 추가됩니다. 자세한 내용은 [Lake Formation FindMatches를 사용하는 레코드 일치](#)를 참조하세요.

Note

시각적 작업에서 사용하는 AWS Glue의 버전은 레코드 일치 변환을 생성하기 위해 AWS Glue에서 사용하는 버전과 일치해야 합니다.

Transform		Output schema		Data preview		
Data preview (20) Info				Previewing 6 of 7 fields		
<input type="text" value="Filter sample dataset"/>						
id	title	venue	year	source	match_id	
journals_sigmod_Liu02	Editor's Notes	SIGMOD Record	2002	DBLP	25769803776	
journals_sigmod_Hammer02	Report on the ACM Fourth International Workshop on Data Warehousing and OLAP (DOLAP 2001)	null	2002	DBLP	25769803777	
journals_sigmod_Konig-RiesMMPPRSVW02	Report on the NSF Workshop on Building an Infrastructure for Mobile and Wireless Systems	null	2002	DBLP	68719476736	

작업 다이어그램에 레코드 일치 변환 노드를 추가하려면

1. 리소스 패널을 열고 정규식 일치를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. 노드 속성 패널에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 기계 학습 변환 페이지로부터 가져온 ID를 입력합니다.

AWS Glue > ML transforms

Machine learning transforms (1) [Info](#)
Clean all your data using machine learning transforms.

Transform name	ID	Status	Label count
○ Test	tfm-3d291b652cec092a79aeda5062f2c96e7c528474	✔ Ready for use	352

4. (선택 사항) 변환 탭에서 신뢰도 점수를 추가하는 옵션을 확인할 수 있습니다. 추가 컴퓨팅 용량을 소모하여 모델은 각 일치에 대한 신뢰도 점수를 추가 열로 추정합니다.

null 행 제거

이 변환은 데이터 세트에서 모든 열이 null인 행을 제거합니다. 또한 빈 필드를 포함하도록 이 기준을 확장하여 하나 이상의 열이 비어 있지 않은 행을 유지할 수 있습니다.

작업 다이어그램에 null 행 제거 변환 노드를 추가하려면

1. 리소스 패널을 열고 null 행 제거를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. 노드 속성 패널에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. (선택 사항) 행이 null이 아닐 뿐만 아니라 비어 있지 않도록 하려면 변환 탭에서 확장 옵션을 선택합니다. 이렇게 하면 이 변환의 목적을 위해 빈 문자열, 배열 또는 맵이 null로 간주됩니다.

JSON 데이터를 포함하는 문자열 열 구문 분석

이 변환은 JSON 데이터를 포함하는 문자열 열을 구문 분석하고 JSON이 객체인지 또는 배열인지에 따라 각각 해당 문자열 열을 구문 또는 배열 열로 변환합니다. 선택적으로 구문 분석된 열과 원래 열을 모두 유지할 수 있습니다.

선택적 샘플링을 통해 JSON 스키마를 제공하거나 추론할 수 있습니다(JSON 객체의 경우).

작업 다이어그램에 JSON 열 구문 분석 변환 노드를 추가하려면

1. 리소스 패널을 열고 JSON 열 구문 분석을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. 노드 속성 패널에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 JSON 문자열이 포함된 열을 선택합니다.
4. (선택 사항) 변환 탭에서 SQL 구문을 사용하여 JSON 데이터가 따르는 스키마를 입력합니다(예: 객체의 경우 'field1 STRING, field2 INT', 배열의 경우 'ARRAY<STRING>').

배열의 경우 스키마가 필요하지만 객체의 경우 스키마가 지정되지 않았다면 데이터를 사용하여 추론됩니다. 스키마 추론에 따른 영향(특히 대규모 데이터 세트의 경우)을 줄이려면 스키마를 추론하는 데 사용할 샘플 비율을 입력하여 전체 데이터를 두 번 읽지 않도록 합니다. 값이 1보다 작으면

해당 비율의 무작위 샘플이 스키마를 추론하는 데 사용됩니다. 데이터를 신뢰할 수 있고 객체가 행 사이에서 일관된 경우 0.1과 같은 작은 비율을 사용하면 성능을 개선할 수 있습니다.

5. (선택 사항) 원래 문자열 열과 구문 분석된 열을 모두 유지하려는 경우 변환 탭에 새 열 이름을 입력할 수 있습니다.

JSON 경로 추출

이 변환은 JSON 문자열 열에서 새 열을 추출합니다. 이 변환은 몇 개의 데이터 요소만 필요하고 전체 JSON 콘텐츠를 테이블 스키마로 가져오지 않으려는 경우에 유용합니다.

작업 다이어그램에 JSON 경로 추출 변환 노드를 추가하려면

1. 리소스 패널을 열고 JSON 경로 추출을 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.
2. 노드 속성 패널에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 JSON 문자열이 포함된 열을 선택합니다. 쉼표로 구분된 하나 이상의 JSON 경로 식을 입력합니다. 각 표현식은 JSON 배열 또는 객체에서 값을 추출하는 방법을 참조합니다. 예를 들어 JSON 열에 속성이 'prop_1' 및 'prop2'인 객체가 포함된 경우 이름을 'prop_1, prop_2'로 지정하여 모두 추출할 수 있습니다.

예를 들어 JSON 필드에 JSON {"a. a": 1}에서 속성을 추출하기 위한 특수 문자가 있는 경우 \$['a. a'] 경로를 사용할 수 있습니다. 단, 쉼표는 경로를 구분하는 기호로 예약되었으므로, 예외입니다. 그런 다음 각 경로에 대해 해당 열 이름을 쉼표로 구분하여 입력합니다.

4. (선택 사항) 추출 후에 변환 탭에서 JSON 열을 삭제하도록 선택할 수 있습니다. 필요한 부분을 추출한 후 나머지 JSON 데이터가 필요하지 않은 경우에 유용합니다.

정규식을 사용하여 문자열 조각 추출

이 변환은 정규식을 사용하여 문자열 조각을 추출하고 문자열 조각에서 새 열을 생성하거나 정규식 그룹을 사용하는 경우 여러 열을 생성할 수 있습니다.

작업 다이어그램에 정규식 추출기 변환 노드를 추가하려면

1. 리소스 패널을 열고 정규식 추출기를 선택하여 작업 다이어그램에 새 변환을 추가합니다. 노드를 추가할 때 선택한 노드가 상위 노드가 됩니다.

2. 노드 속성 패널에서 작업 다이어그램에 노드 이름을 입력할 수 있습니다. 노드 상위 항목이 아직 선택되지 않은 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.
3. 변환 탭에서 정규식과 정규식을 적용해야 하는 열을 입력합니다. 그런 다음 일치하는 문자열을 저장할 새 열의 이름을 입력합니다. 소스 열이 null인 경우에만 새 열이 null이 되고, 정규식이 일치하지 않으면 열이 비어 있게 됩니다.

정규식이 그룹을 사용하는 경우 쉼표로 구분된 해당 열 이름이 있지만 열 이름을 비워 두면 그룹을 건너뛸 수 있습니다.

예를 들어 ISO의 긴 날짜 형식과 ISO의 짧은 날짜 형식을 모두 사용하는 문자열이 포함된 'purchase_date' 열이 있는 경우 가능하면 연도, 월, 일, 시간을 추출하려고 합니다. 시간 그룹은 선택 사항이지만, 시간 그룹을 사용할 수 없는 행에서는 정규식이 일치하지 않으므로 추출된 모든 그룹이 빈 문자열이 됩니다. 이 경우 그룹에서 시간을 선택 사항으로 지정하지 않고 내부 항목을 사용하려고 하므로 이름을 비우고 추출되지 않도록 합니다(이 그룹에는 T 문자가 포함됨).

Transform
Output schema
Data preview
✕

Name

Regex extractor

Node parents
Choose which nodes will provide inputs for this one.

Choose one or more parent node ▼

S3 bucket
✕

S3 - DataSource

Column to extract from
String column on which to apply the regex.

purchase_date
▼

string

Regular expression
Regex to apply on the column, if multiple columns need to be extracted then the expression needs an equal number of groups.


(\d\d\d\d)-(\d\d)-(\d\d)(T(\d\d))?

Extracted column
The name of the column where to extract the matched regex. Multiple column names can be specified separated by commas, if the name is empty it means that group is skipped. If the source column is null, the new column will be null as well, otherwise an empty string means there was no match.

year, month, day, ,hour

데이터 미리 보기의 결과:

Data preview (5) [Info](#) Previewing 5 of 5 fields



<code>purchase_date</code>	<code>year</code>	<code>month</code>	<code>day</code>	<code>hour</code>
2023-03-04T12:23:31	2023	03	04	12
2021-06-09T02:21:01	2021	06	09	02
2022-02-04	2022	02	04	
2020-09-05T23:07:02	2020	09	05	23
2020-09-08	2020	09	08	

사용자 정의 변환 생성

데이터에 대해 더 복잡한 변환을 수행해야 하거나 데이터 속성 키를 데이터 집합에 추가하려는 경우 작업 다이어그램에 [사용자 정의 코드(Custom code)] 변환을 추가할 수 있습니다. 사용자 정의 코드 노드를 사용하면 변환을 수행하는 스크립트를 입력할 수 있습니다.

사용자 정의 코드를 사용하는 경우 스키마 편집기를 사용하여 사용자 정의 코드를 통해 출력에 대한 변경 사항을 표시해야 합니다. 스키마를 편집할 때 다음 작업을 수행할 수 있습니다.

- 데이터 속성 키 추가 또는 제거
- 데이터 속성 키의 데이터 유형 변경
- 데이터 속성 키의 이름 변경
- 중첩 속성 키 재구성

출력을 대상 위치로 보내기 전에 사용자 정의 변환 노드의 결과에서 단일 `DynamicFrame`을 선택하려면 `SelectFromCollection` 변환을 사용해야 합니다.

다음 태스크를 사용하여 작업 다이어그램에 사용자 정의 변환 노드를 추가합니다.

작업 다이어그램에 사용자 정의 코드 변환 노드 추가

작업 다이어그램에 사용자 정의 변환 노드를 추가하려면

1. (선택 사항) 리소스 패널을 열고 사용자 지정 변환을 선택하여 작업 다이어그램에 사용자 지정 변환을 추가합니다.
2. [노드 속성(Node properties)] 탭에서 작업 다이어그램에 노드 이름을 입력합니다. 노드 상위 항목이 아직 선택되지 않았거나 사용자 정의 변환에 대해 여러 입력을 원하는 경우 [노드 상위 항목(Node parents)] 목록에서 변환의 입력 소스로 사용할 노드를 선택합니다.

사용자 정의 변환 노드에 대한 코드 입력

입력 필드에 코드를 입력하거나 복사할 수 있습니다. 작업에서는 이 코드를 사용하여 데이터 변환을 수행합니다. Python 또는 Scala에서 코드 조각을 제공할 수 있습니다. 코드는 입력으로 하나 이상의 DynamicFrames를 취해야 하며 DynamicFrames 컬렉션을 반환합니다.

사용자 정의 변환 노드에 대한 스크립트를 입력하려면

1. 작업 다이어그램에서 사용자 정의 변환 노드를 선택한 상태에서 [변환(Transform)] 탭을 선택합니다.
2. 머리글 [코드 블록(Code block)] 아래의 텍스트 입력 필드에 변환 코드를 붙여넣거나 입력합니다. 사용하는 코드는 [작업 세부 정보(Job details)] 탭에서 작업에 대해 지정된 언어와 일치해야 합니다.

코드에서 입력 노드를 참조할 때 AWS Glue Studio는 작업 다이어그램 노드에서 반환된 DynamicFrames의 이름을 생성 순서에 따라 순차적으로 지정합니다. 코드에 다음 이름 지정 방법 중 하나를 선택합니다.

- 클래식 코드 생성 - 함수 이름을 사용하여 작업 다이어그램에서 노드를 참조합니다.
 - 데이터 원본 노드: DataSource0, DataSource1, DataSource2 등.
 - 변환 노드: Transform0, Transform1, Transform2 등.
- 새 코드 생성 - 노드의 노드 속성(Node properties) 탭에서 지정된 이름에 '_node1', '_node2' 등을 추가하여 사용합니다. 예: S3bucket_node1, ApplyMapping_node2, S3bucket_node2, MyCustomNodeName_node1.

새 코드 생성기에 대한 자세한 내용은 [스크립트 코드 생성](#) 섹션을 참조하세요.

다음 예는 코드 상자에 입력할 코드의 포맷을 보여줍니다.

Python

다음 예제에서는 처음 수신한 `DynamicFrame`을 가져와 `DataFrame`으로 변환하여 기본 필터 방식을 적용한 다음(1,000개 이상의 투표가 있는 레코드만 유지) 반환하기 전에 다시 `DynamicFrame`으로 변환합니다.

```
def FilterHighVoteCounts (glueContext, dfc) -> DynamicFrameCollection:
    df = dfc.select(list(dfc.keys())[0]).toDF()
    df_filtered = df.filter(df["vote_count"] > 1000)
    dyf_filtered = DynamicFrame.fromDF(df_filtered, glueContext, "filter_votes")
    return(DynamicFrameCollection({"CustomTransform0": dyf_filtered}, glueContext))
```

Scala

다음 예제에서는 처음 수신한 `DynamicFrame`을 가져와 `DataFrame`으로 변환하여 기본 필터 방식을 적용한 다음(1,000개 이상의 투표가 있는 레코드만 유지) 반환하기 전에 다시 `DynamicFrame`으로 변환합니다.

```
object FilterHighVoteCounts {
    def execute(glueContext : GlueContext, input : Seq[DynamicFrame]) :
    Seq[DynamicFrame] = {
        val frame = input(0).toDF()
        val filtered = DynamicFrame(frame.filter(frame("vote_count") > 1000),
        glueContext)
        Seq(filtered)
    }
}
```

사용자 정의 변환 노드에서 스키마 편집

사용자 지정 변환 노드를 사용하는 경우 AWS Glue Studio는 변환을 통해 생성되는 출력 스키마를 자동으로 유추할 수 없습니다. 스키마 편집기를 사용하여 사용자 정의 변환 코드에 의해 구현된 스키마 변경 사항을 설명합니다.

사용자 정의 코드 노드에는 사용자 정의 코드에 대한 입력으로 `DynamicFrame`을 제공하는 상위 노드가 여러 개 있을 수 있습니다. 사용자 정의 코드 노드는 `DynamicFrames`의 컬렉션을 반환합니다. 입력으로 사용되는 각 `DynamicFrame`에는 연결된 스키마가 있습니다. 사용자 정의 코드 노드에서 반환된 각 `DynamicFrame`을 설명하는 스키마를 추가해야 합니다.



Note



사용자 지정 변환에서 사용자 고유의 스키마를 설정하면 AWS Glue Studio에서는 이전 노드의 스키마를 상속하지 않습니다. 스키마를 업데이트하려면 사용자 지정(Custom transform) 변환 노드를 선택한 다음 데이터 미리 보기(Data preview) 탭을 선택합니다. 미리 보기가 생성되면 '미리 보기 스키마 사용(Use Preview Schema)'을 선택합니다. 그러면 스키마가 미리 보기 데이터를 사용하는 스키마로 대체됩니다.





사용자 정의 변환 노드에 대한 출력 스키마를 편집하려면

1. 작업 다이어그램에서 사용자 정의 변환 노드를 선택한 상태에서 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택합니다.
2. [편집(Edit)]을 선택하여 스키마를 변경합니다.

배열 또는 객체와 같은 중첩 데이터 속성 키가 있는 경우 각 스키마 패널의 오른쪽 상단에 있는 [행 확장(Expand-Rows)] 아이콘

() 을 선택하여 하위 데이터 속성 키 목록을 확장할 수 있습니다. 이 아이콘을 선택하면 하위 속성 키 목록을 축소하도록 선택할 수 있는 [행 축소(Collapse-Rows)] 아이콘 () 으로 변경됩니다.

3. 페이지 오른쪽 섹션에서 다음 작업을 사용하여 스키마를 수정합니다.
 - 속성 키의 이름을 바꾸려면 속성 키의 [키(Key)] 텍스트 상자에 커서를 놓고 새 이름을 입력합니다.
 - 속성 키의 데이터 유형을 변경하려면 목록을 사용하여 속성 키의 새 데이터 유형을 선택합니다.
 - 스키마에 새 최상위 속성 키를 추가하려면 [취소(Cancel)] 버튼 왼쪽에 있는 [오버플로(Overflow)] () 아이콘을 선택한 다음 [루트 키 추가(Add root key)]를 선택합니다.
 - 스키마에 하위 속성 키를 추가하려면 상위 키와 연결된 [키 추가(Add-Key)] 아이콘 () 을 선택합니다. 하위 키의 이름을 입력하고 데이터 유형을 선택합니다.

- 스키마에서 속성 키를 제거하려면 키 이름의 맨 오른쪽에 있는 [제거(Remove)] 아이콘 () 을 선택합니다.
4. 사용자 정의 변환 코드가 여러 DynamicFrames를 사용하는 경우 출력 스키마를 더 추가할 수 있습니다.
- 비어 있는 새 스키마를 추가하려면 [오버플로(Overflow)] () 아이콘을 선택한 다음 [출력 스키마 추가(Add output schema)]를 선택합니다.
 - 기존 스키마를 새 출력 스키마로 복사하려면 복사하려는 스키마가 스키마 선택기에 표시되는지 확인합니다. [오버플로(Overflow)] () 아이콘을 선택한 다음 [복제(Duplicate)]를 선택합니다.
- 출력 스키마를 제거하려면 복사하려는 스키마가 스키마 선택기에 표시되는지 확인합니다. [오버플로(Overflow)] () 아이콘을 선택한 다음 [삭제(Delete)]를 선택합니다.
5. 새 스키마에 새 루트 키를 추가하거나 복제된 키를 편집합니다.
6. 출력 스키마를 수정할 때 [적용(Apply)] 버튼을 선택하여 변경 사항을 저장하고 스키마 편집기를 종료합니다.

변경 사항을 저장하지 않으려면 [취소(Cancel)] 버튼을 선택합니다.

사용자 정의 변환 출력 구성

사용자 정의 코드 변환은 결과 집합에 DynamicFrame이 하나만 있더라도 DynamicFrames의 컬렉션을 반환합니다.

사용자 정의 변환 노드에서 출력을 처리하려면

1. 사용자 정의 변환 노드가 상위 노드로 있는 SelectFromCollection 변환 노드를 추가합니다. 이 변환을 업데이트하여 사용하려는 데이터 집합을 나타냅니다. 자세한 정보는 [SelectFromCollection을 사용하여 유지할 데이터 집합 선택](#)을 참조하세요.
2. 사용자 정의 변환 노드에서 생성된 추가 DynamicFrames를 사용하려면 작업 다이어그램에 SelectFromCollection 변환을 더 추가합니다.

항공 데이터 집합을 여러 데이터 집합으로 분할하기 위해 사용자 정의 변환 노드를 추가하지 만 비행 날짜 또는 항공편 번호와 같은 각 출력 스키마에서 일부 식별 속성 키를 복제하는 시 나리오를 고려하세요. 사용자 정의 변환 노드를 상위 항목로 사용하여 각 출력 스키마에 대해 `SelectFromCollection` 변환 노드를 추가합니다.

3. (선택 사항) 그런 다음 각 `SelectFromCollection` 변환 노드를 작업의 다른 노드에 대한 입력으로 사 용하거나 데이터 대상 노드의 부모로 사용할 수 있습니다.

AWS Glue 사용자 지정 시각적 변환

사용자 지정 시각적 변환을 사용하면 변환을 생성하여 AWS Glue Studio 작업에 사용할 수 있습니다. 사용자 지정 시각적 변환을 사용하면 코딩에 익숙하지 않은 ETL 개발자도 AWS Glue Studio 인터페이 스를 사용하여 점점 늘어나는 변환 라이브러리를 검색하고 사용할 수 있습니다.

사용자 지정 시각적 변환을 생성한 다음 Amazon S3에 업로드하여 AWS Glue Studio의 시각적 편집기 를 통해 이러한 작업을 수행하는 데 사용할 수 있습니다.

주제

- [사용자 지정 시각적 변환 시작하기](#)
- [단계 1. JSON 구성 파일 생성](#)
- [단계 2. 변환 로직 구현](#)
- [단계 3. AWS Glue Studio에서 사용자 지정 시각적 변환을 검증하고 문제를 해결합니다.](#)
- [4단계. 필요에 따라 사용자 지정 시각적 변환 업데이트](#)
- [5단계. AWS Glue Studio에서 사용자 지정 시각적 변환 사용](#)
- [사용 예제:](#)
- [사용자 지정 시각적 스크립트의 예](#)
- [비디오](#)

사용자 지정 시각적 변환 시작하기

사용자 지정 시각적 변환을 생성하려면 다음 단계를 따라야 합니다.

- 단계 1. JSON 구성 파일 생성
- 단계 2. 변환 로직 구현
- 단계 3. 사용자 지정 시각적 변환 검증

- 4단계. 필요에 따라 사용자 지정 시각적 변환 업데이트
- 5단계. AWS Glue Studio에서 사용자 지정 시각적 변환 사용

Amazon S3 버킷을 설정하여 시작하고 1단계로 진행합니다. JSON 구성 파일을 생성합니다.

필수 조건

고객이 제공한 변환은 고객 AWS 계정 내에 있습니다. 해당 계정은 변환을 소유하므로 변환 내용을 보거나(검색 및 사용) 편집하거나 삭제할 수 있는 모든 권한을 가집니다.

AWS Glue Studio에서 사용자 지정 변환을 사용하려면 다음 두 파일을 생성하여 해당 AWS 계정의 Amazon S3 자산 버킷에 업로드해야 합니다.

- Python 파일 - 변환 함수를 포함합니다.
- JSON 파일 - 변환을 설명합니다. 이 파일은 변환을 정의하는 데 필요한 구성 파일이라고도 합니다.

파일을 함께 연결하려면 두 파일에 동일한 이름을 사용하세요. 예:

- myTransform.json
- myTransform.py

선택적으로 아이콘이 포함된 SVG 파일을 제공하여 사용자 지정 시각적 변환에 사용자 지정 아이콘을 제공할 수 있습니다. 파일을 함께 페어링하려면 아이콘에서 동일한 이름을 사용합니다.

- myTransform.svg

AWS Glue Studio에서는 각 파일 이름을 사용하여 자동으로 일치시킵니다. 기존 모듈의 파일 이름은 같을 수 없습니다.

변환 파일 이름에 대한 권장 규칙

AWS Glue Studio에서는 파일을 작업 스크립트에 모듈(예: `import myTransform`)로 가져옵니다. 따라서 파일 이름은 Python 변수 이름(식별자)에 설정된 것과 동일한 이름 지정 규칙을 따라야 합니다. 특히 문자 또는 밑줄로 시작하고 그 뒤에는 문자, 숫자 및/또는 밑줄로만 구성되어야 합니다.

Note

예기치 못한 런타임 문제를 방지하려면 변환 파일 이름이 기존에 로드된 Python 모듈(예: `sys`, `array`, `copy` 등)과 충돌하지 않는지 확인하세요.

Amazon S3 버킷 설정

생성된 변환은 Amazon S3에 저장되며 사용자 AWS 계정에서 소유합니다. 모든 작업 스크립트가 현재 저장되어 있는 Amazon S3 자산 폴더(예: `s3://aws-glue-assets-<accountid>-<region>/transforms`)에 파일(json 및 py)을 업로드하면 새로운 사용자 지정 시각적 변환이 자동으로 생성됩니다. 사용자 지정 아이콘을 사용하는 경우 아이콘도 업로드합니다. 기본적으로 AWS Glue Studio에서는 동일한 S3 버킷의 `/transforms` 폴더에서 모든 `.json` 파일을 읽습니다.

단계 1. JSON 구성 파일 생성

JSON 구성 파일은 사용자 지정 시각적 변환을 정의하고 설명하는 데 필요합니다. 구성 파일의 스키마는 다음과 같습니다.

JSON 파일 구조

필드

- `name: string` - (필수) 변환을 식별하는 데 사용되는 변환 시스템 이름입니다. Python 변수 이름(식별자)에 설정된 것과 동일한 이름 지정 규칙을 따릅니다. 특히 문자 또는 밑줄로 시작하고 그 뒤에는 문자, 숫자 및/또는 밑줄로만 구성되어야 합니다.
- `displayName: string` - (선택 사항) AWS Glue Studio 시각적 작업 편집기에 표시되는 변환의 이름입니다. `displayName`을 지정하지 않은 경우 `name`이 AWS Glue Studio에서 변환의 이름으로 사용됩니다.
- `description: string` - (선택 사항) 변환 설명이 AWS Glue Studio에 표시되고 검색할 수 있습니다.
- `functionName: string` - (필수) Python 함수 이름은 Python 스크립트에서 호출할 함수를 식별하는 데 사용됩니다.
- `path: string` - (선택 사항) Python 소스 파일의 전체 Amazon S3 경로입니다. 지정하지 않은 경우 AWS Glue에서는 파일 이름 일치를 사용하여 `.json` 파일과 `.py` 파일을 함께 연결합니다. 예를 들어 `myTransform.json` JSON 파일은 동일한 Amazon S3 위치에 있는 Python 파일 `myTransform.py`와 연결됩니다.

- **parameters:** Array of TransformParameter object - (선택 사항) AWS Glue Studio 시각적 편집기에서 파라미터를 구성할 때 표시할 파라미터 목록입니다.

TransformParameter fields

- **name:** string - (필수) Python 함수에 작업 스크립트에서 명명된 인수로 전달될 파라미터 이름입니다. Python 변수 이름(식별자)에 설정된 것과 동일한 이름 지정 규칙을 따릅니다. 특히 문자 또는 밑줄로 시작하고 그 뒤에는 문자, 숫자 및/또는 밑줄로만 구성되어야 합니다.
- **displayName:** string - (선택 사항) AWS Glue Studio 시각적 작업 편집기에 표시되는 변환의 이름입니다. displayName을 지정하지 않은 경우 name이 AWS Glue Studio에서 변환의 이름으로 사용됩니다.
- **type:** string - (필수) 일반적인 Python 데이터 유형을 허용하는 파라미터 유형입니다. 유효한 값: 'str' | 'int' | 'float' | 'list' | 'bool'.
- **isOptional:** boolean - (선택 사항) 파라미터가 선택 사항인지 여부를 결정합니다. 기본적으로 모든 파라미터는 필수입니다.
- **description:** string - (선택 사항) 사용자가 변환 파라미터를 구성하는 데 도움이 되는 설명이 AWS Glue Studio에 표시됩니다.
- **validationType:** string - (선택 사항) 이 파라미터의 유효성을 검사하는 방법을 정의합니다. 현재는 정규 표현식만 지원합니다. 기본적으로 검증 유형은 RegularExpression으로 설정됩니다.
- **validationRule:** string - (선택 사항) validationType을 RegularExpression으로 설정한 경우에 제출하기 전에 양식 입력을 검증하는 데 사용되는 정규 표현식입니다. 정규 표현식 구문은 [RegExp EcmaScript 사양](#)과 호환되어야 합니다.
- **validationMessage:** string - (선택 사항) 검증에 실패한 경우에 표시할 메시지입니다.
- **listOptions:** An array of TransformParameterListOption object, string 또는 문자열 값 'column' - (선택 사항) 선택 또는 다중 선택 UI 제어에 표시할 옵션입니다. 심표로 구분된 값 목록 또는 강력한 형식의 TransformParameterListOption JSON 객체 유형을 허용합니다. 또한 문자열 값 'column'을 지정하여 상위 노드 스키마의 열 목록을 동적으로 채울 수 있습니다.
- **listType:** string - (선택 사항) type = 'list'에 대한 옵션 유형을 정의합니다. 유효한 값: 'str' | 'int' | 'float' | 'list' | 'bool'. 일반적인 Python 데이터 유형을 허용하는 파라미터 유형입니다.

TransformParameterListOption 필드

- **value:** string | int | float | bool - (필수) 옵션 값입니다.
- **label:** string - (선택 사항) 선택 드롭다운에 표시되는 옵션 레이블입니다.

AWS Glue Studio의 파라미터 변환

.json 파일에 `isOptional`로 표시되지 않는 한 기본적으로 파라미터가 필요합니다. AWS Glue Studio에서는 파라미터가 Transform(변환) 탭에 표시됩니다. 이 예에서는 이메일 주소, 전화번호, 나이, 성별, 출신 국가와 같은 사용자 정의 파라미터를 보여줍니다.

The screenshot displays the AWS Glue Studio interface. On the left, a canvas shows a 'Data source - S3 bucket Amazon S3' connected to a 'Transform - Dynamic Trans...' node named 'My Transform'. On the right, the 'Transform' tab is active, showing a form with the following fields:

- Email Address:** Enter your work email address below. (Text input field)
- Phone Number:** Enter your mobile phone number below. (Text input field)
- Your age:** (Text input field)
- Your gender:** (Dropdown menu)
- Your origin country? - optional:** What country were you born in? (Dropdown menu with 'Choose options' selected)
- Do you want to receive promotional newsletter from us? - optional:** (Checkbox, currently unchecked)

`validationMessage`에서 `validationRule` 파라미터를 지정하고 검증 메시지를 지정하여 json 파일에서 정규 표현식을 통해 AWS Glue Studio에서 일부 검증을 적용할 수 있습니다.

```
"validationRule": "^\\((?\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
"validationMessage": "Please enter a valid US number"
```

Note

검증은 브라우저에서 수행되므로 정규 표현식 구문은 [RegExp EcmaScript 사양](#)과 호환되어야 합니다. Python 구문은 이러한 정규 표현식에 지원되지 않습니다.

검증을 추가하면 사용자가 잘못된 사용자 입력으로 작업을 저장하는 것을 방지할 수 있습니다. AWS Glue Studio에서는 예제에 표시된 대로 검증 메시지를 표시합니다.

Node properties | **Transform** 1 | Output schema | Data preview

Email Address
Enter your work email address below

wrongEmail.com

⚠ Please enter a valid email address

파라미터는 파라미터 구성에 따라 AWS Glue Studio에 표시됩니다.

- type이 str, int 또는 float일 때 텍스트 입력 필드가 표시됩니다. 예를 들어 스크린샷은 'Email Address' 및 'Your age' 파라미터의 입력 필드를 보여줍니다.

Email Address
Enter your work email address below

|

Your age

|

- type이 bool이면 확인란이 표시됩니다.

Do you want to receive promotional newsletter from us?

- type이 str이고 listOptions가 제공되면 단일 선택 목록이 표시됩니다.

Your gender

Male	▲
Male	✓
Female	
Other	

- type이 list 및 listOptions이고 listType이 제공되면 다중 선택 목록이 표시됩니다.

Country recently visited - optional

What countries did you visit in the past 2 years?

Choose options ▲

- Iceland
- India India
- Indor India
- Iran
- Iraq
- Ireland
- Israel
- Italy
- Jamaica
- Japan

열 선택기를 파라미터로 표시

구성 시 사용자가 스키마에서 열을 선택해야 하는 경우 열 선택기를 표시할 수 있습니다. 이 경우 사용자가 열 이름을 입력하지 않아도 됩니다. `listOptions` 필드를 'column'으로 설정하면 AWS Glue Studio에서는 상위 노드 출력 스키마를 기반으로 열 선택기를 표시합니다. AWS Glue Studio에서는 단일 또는 다중 열 선택기를 표시할 수 있습니다.

다음 예제에서는 스키마를 사용합니다.

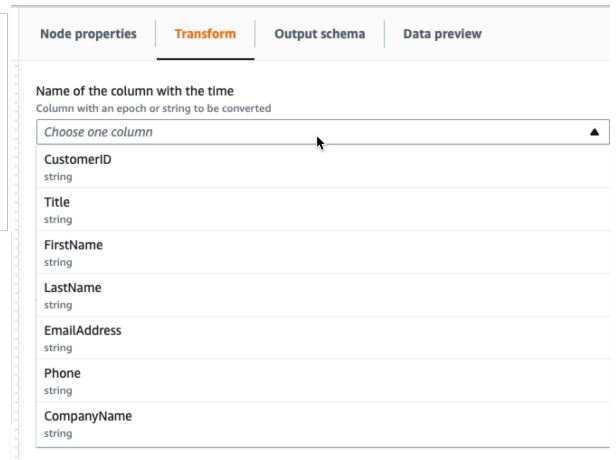
Key	Data type	Partition
CustomerID	string	-
Title	string	-
FirstName	string	-
LastName	string	-
EmailAddress	string	-
Phone	string	-
CompanyName	string	-

단일 열을 표시하도록 사용자 지정 시각적 변환 파라미터를 정의하려면:

1. JSON 파일에서 parameters 객체의 listOptions 값을 'column'으로 설정합니다. 이렇게 하면 사용자가 AWS Glue Studio의 선택 목록에서 열을 선택할 수 있습니다.

```

{
  "name": "mb_to_timestamp",
  "displayName": "MB Convert column to timestamp",
  "description": "Convert a timestamp string or a system epoch column into a new timestamp type column.",
  "functionName": "mb_to_timestamp",
  "parameters": [
    {
      "name": "colName",
      "displayName": "Name of the column with the time",
      "type": "str",
      "listOptions": "column",
      "description": "Column with an epoch or string to be converted"
    }
  ]
}
    
```



2. 파라미터를 다음과 같이 정의하여 여러 열을 선택할 수도 있습니다.

- listOptions: "column"
- type: "list"

```
{
  "name": "mb_to_timestamp",
  "displayName": "MB Convert column to timestamp",
  "description": "Convert a timestamp string or a system epoch column into a new timestamp type column.",
  "functionName": "mb_to_timestamp",
  "parameters": [
    {
      "name": "colNames",
      "displayName": "Name of the column with the time",
      "type": "list",
      "listOptions": "column",
      "listType": "str",
      "description": "Column with an epoch or string to be converted"
    }
  ]
}
```

Node properties | **Transform** | Output schema | Data preview

Name of the column with the time
Column with an epoch or string to be converted

Choose options

- CustomerID
string
- Title
string
- FirstName
string
- LastName
string
- EmailAddress
string
- Phone
string
- CompanyName
string

단계 2. 변환 로직 구현

Note

사용자 지정 시각적 변환은 Python 스크립트만 지원합니다. Scala는 지원되지 않습니다.

.json 구성 파일에 정의된 함수를 구현하는 코드를 추가하려면 Python 파일을 .json 파일과 동일한 위치에 이름은 동일하지만 “.py” 확장자로 저장하는 것이 좋습니다. AWS Glue Studio에서는 .json 파일과 .py 파일을 자동으로 연결하므로 구성 파일에 Python 파일의 경로를 지정할 필요가 없습니다.

Python 파일에서 명명된 파라미터를 구성하여 선언된 함수를 추가하고 DynamicFrame에서 사용하기 위해 함수를 등록합니다. 다음은 Python 파일의 예제입니다.

```
from awsglue import DynamicFrame

# self refers to the DynamicFrame to transform,
# the parameter names must match the ones defined in the config
# if it's optional, need to provide a default value
def myTransform(self, email, phone, age=None, gender="",
                country="", promotion=False):
    resulting_dynf = # do some transformation on self
    return resulting_dynf

DynamicFrame.myTransform = myTransform
```

Python 코드를 가장 빠르게 개발하고 테스트하려면 AWS Glue 노트북을 사용하는 것이 좋습니다.

[AWS Glue Studio에서 노트북 시작하기](#)를 참조하세요.

변환 로직을 구현하는 방법을 설명하기 위해 아래 예제의 사용자 지정 시각적 변환은 들어오는 데이터를 필터링하여 특정 미국 주와 관련된 데이터만 유지하는 변환입니다. json 파일에는 `functionName`에 대한 파라미터(`custom_filter_state`)와 두 개의 인수(유형이 "str"인 "state" 및 "colName")가 포함되어 있습니다.

예제 구성 json 파일은 다음과 같습니다.

```
{
  "name": "custom_filter_state",
  "displayName": "Filter State",
  "description": "A simple example to filter the data to keep only the state indicated.",
  "functionName": "custom_filter_state",
  "parameters": [
    {
      "name": "colName",
      "displayName": "Column name",
      "type": "str",
      "description": "Name of the column in the data that holds the state postal code"
    },
    {
      "name": "state",
      "displayName": "State postal code",
      "type": "str",
      "description": "The postal code of the state whole rows to keep"
    }
  ]
}
```

Python에서 컴패니언 스크립트를 구현하려면

1. AWS Glue 노트북을 시작하고 시작할 세션에 제공된 초기 셀을 실행합니다. 초기 셀을 실행하면 필요한 기본 구성 요소가 생성됩니다.
2. 예제에 설명된 대로 필터링을 수행하는 함수를 생성하고 `DynamicFrame`에 등록합니다. 아래 코드를 복사하여 AWS Glue 노트북의 셀에 붙여 넣습니다.

```
from awsglue import DynamicFrame
```

```
def custom_filter_state(self, colName, state):
    return self.filter(lambda row: row[colName] == state)

DynamicFrame.custom_filter_state = custom_filter_state
```

3. 샘플 데이터를 생성하거나 로드하여 동일한 셀 또는 새 셀에서 코드를 테스트합니다. 새 셀에 샘플 데이터를 추가하는 경우 셀을 실행하는 것을 잊지 마세요. 예:

```
# A few of rows of sample data to test
data_sample = [
    {"state": "CA", "count": 4},
    {"state": "NY", "count": 2},
    {"state": "WA", "count": 3}
]
df1 = glueContext.sparkSession.sparkContext.parallelize(data_sample).toDF()
dynf1 = DynamicFrame.fromDF(df1, glueContext, None)
```

4. 다른 인수를 사용하여 “custom_filter_state”를 검증하려면 테스트하세요.

```
[14]: dynf1.custom_filter_state("state", "NY").show()
{"count": 2, "state": "NY"}
```

5. 여러 테스트를 실행한 후 확장명이 .py인 코드를 저장하고 .json 파일 이름을 미러링하는 이름으로 .py 파일의 이름을 지정합니다. .py 파일과 .json 파일은 동일한 변환 폴더에 있어야 합니다.

다음 코드를 복사하여 파일에 붙여넣고 .py 파일 확장명으로 이름을 바꿉니다.

```
from awsglue import DynamicFrame

def custom_filter_state(self, colName, state):
    return self.filter(lambda row: row[colName] == state)

DynamicFrame.custom_filter_state = custom_filter_state
```

6. AWS Glue Studio에서 시각적 작업을 열고 사용 가능한 Transforms(변환) 목록에서 선택하여 작업에 변환을 추가합니다.

Python 스크립트 코드에서 이 변환을 다시 사용하려면 “참조된 파일 경로” 아래의 작업에 있는 .py 파일에 Amazon S3 경로를 추가하고 스크립트에서 Python 파일 이름(확장자 제외)을 파일 상단에 추가하여 가져옵니다. 예를 들면 다음과 같습니다. `import <name of the file (without the extension)>`

단계 3. AWS Glue Studio에서 사용자 지정 시각적 변환을 검증하고 문제를 해결합니다.

AWS Glue Studio에서는 사용자 지정 시각적 변환이 AWS Glue Studio에 로드되기 전에 JSON 구성 파일을 검증합니다. 검증에는 다음이 포함됩니다.

- 필수 필드의 존재 여부
- JSON 형식 검증
- 올바르지 않거나 유효하지 않은 파라미터
- .py 파일과 .json 파일이 동일한 Amazon S3 경로에 모두 존재하는지 여부
- .py와 .json의 파일 이름 일치

검증이 성공하면 변환이 시각적 편집기의 사용 가능한 Actions(작업) 목록에 나열됩니다. 사용자 지정 아이콘이 제공된 경우 작업 옆에 표시되어야 합니다.

검증에 실패할 경우 AWS Glue Studio에서는 사용자 지정 시각적 변환을 로드하지 않습니다.

4단계. 필요에 따라 사용자 지정 시각적 변환 업데이트

변환이 해당 json 정의를 따르는 한 변환 스크립트를 생성하고 사용한 이후에 업데이트할 수 있습니다.

- DynamicFrame에 할당할 때 사용되는 이름은 json functionName과 일치해야 합니다.
- 함수 인수는 [단계 1. JSON 구성 파일 생성](#)에 설명된 대로 json 파일에 정의되어야 합니다.
- 작업이 Python 파일에 직접 의존하므로 Python 파일의 Amazon S3 경로는 변경할 수 없습니다.

Note

업데이트가 필요한 경우 스크립트와 .json 파일이 지속적으로 업데이트되고 모든 시각적 작업이 새 변환과 함께 올바르게 다시 저장되었는지 확인하세요. 업데이트 후 시각적 작업을 저장하지 않으면 업데이트가 적용 및 검증되지 않습니다. Python 스크립트 파일의 이름이 바뀌거나 .json 파일 옆에 배치되지 않은 경우 .json 파일에 전체 경로를 지정해야 합니다.

사용자 지정 아이콘

워크플로의 일부에서 작업의 기본 아이콘이 시각적으로 구분되지 않는다고 판단되면 [the section called “ 사용자 지정 시각적 변환 시작하기 ”](#)에 설명된 대로 사용자 지정 아이콘을 제공할 수 있습니다. Amazon S3에 호스팅된 해당 SVG를 업데이트하여 아이콘을 업데이트할 수 있습니다.

최상의 결과를 얻으려면 Cloudscape Design System의 지침에 따라 32x32px로 표시되도록 이미지를 설계합니다. Cloudscape 지침에 대한 자세한 내용은 [Cloudscape 설명서](#)를 참조하세요.

5단계. AWS Glue Studio에서 사용자 지정 시각적 변환 사용

AWS Glue Studio에서 사용자 지정 시각적 변환을 사용하려면 구성 및 소스 파일을 업로드한 다음 Action(작업) 메뉴에서 변환을 선택합니다. 값 또는 입력이 필요한 파라미터는 Transform(변환) 탭에서 사용할 수 있습니다.

1. 두 파일(Python 소스 파일 및 JSON 구성 파일)을 작업 스크립트가 저장되는 Amazon S3 자산 폴더에 업로드합니다. 기본적으로 AWS Glue에서는 동일한 Amazon S3 버킷의 /transforms 폴더에서 모든 .json 파일을 가져옵니다.
2. Action(작업) 메뉴에서 사용자 지정 시각적 변환을 선택합니다. .json 구성 파일에 지정된 변환 displayName 또는 이름으로 이름이 지정됩니다.
3. 구성 파일에 구성된 모든 파라미터의 값을 입력합니다.

The screenshot displays the AWS Glue Studio interface. On the left, a workflow diagram shows a 'Data source - S3 bucket Amazon S3' node connected to a 'Transform - Dynamic Trans... My Transform' node. The right panel is titled 'Transform' and contains the following configuration fields:

- Email Address:** Enter your work email address below. (Text input field)
- Phone Number:** Enter your mobile phone number below. (Text input field)
- Your age:** (Text input field)
- Your gender:** (Dropdown menu)
- Your origin country? - optional:** What country were you born in? (Dropdown menu with 'Choose options' selected)
- Do you want to receive promotional newsletter from us? - optional:** (Checkbox, currently unchecked)

사용 예제:

다음은 .json 구성 파일에 있는 모든 가능한 파라미터의 예입니다.

```
{
  "name": "MyTransform",
  "displayName": "My Transform",
  "description": "This transform description will be displayed in UI",
  "functionName": "myTransform",
  "parameters": [
    {
      "name": "email",
      "displayName": "Email Address",
      "type": "str",
      "description": "Enter your work email address below",
      "validationType": "RegularExpression",
      "validationRule": "^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$",
      "validationMessage": "Please enter a valid email address"
    },
    {
      "name": "phone",
      "displayName": "Phone Number",
      "type": "str",
      "description": "Enter your mobile phone number below",
      "validationRule": "^\\((?\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
      "validationMessage": "Please enter a valid US number"
    },
    {
      "name": "age",
      "displayName": "Your age",
      "type": "int",
      "isOptional": true
    },
    {
      "name": "gender",
      "displayName": "Your gender",
      "type": "str",
      "listOptions": [
        {"label": "Male", "value": "male"},
        {"label": "Female", "value": "female"},
        {"label": "Other", "value": "other"}
      ],
      "isOptional": true
    },
  ],
}
```

```

{
  "name": "country",
  "displayName": "Your origin country ?",
  "type": "list",
  "listOptions": "Afghanistan,Albania,Algeria,American Samoa,Andorra,Angola,Anguilla,Antarctica,Antigua and Barbuda,Argentina,Armenia,Aruba,Australia,Austria,Azerbaijan,Bahamas,Bahrain,Bangladesh,Barbados and Herzegovina,Botswana,Bouvet Island,Brazil,British Indian Ocean Territory,Brunei Darussalam,Bulgaria,Burkina Faso,Burundi,Cambodia,Cameroon,Canada,Cape Verde,Cayman Islands,Central African Republic,Chad,Chile,China,Christmas Island,Cocos (Keeling Islands),Colombia,Comoros,Congo,Cook Islands,Costa Rica,Cote D'Ivoire (Ivory Coast),Croatia (Hrvatska,Cuba,Cyprus,Czech Republic,Denmark,Djibouti,Dominica,Dominican Republic,East Timor,Ecuador,Egypt,El Salvador,Equatorial Guinea,Eritrea,Estonia,Ethiopia,Falkland Islands (Malvinas),Faroe Islands,Fiji,Finland,France,France,Metropolitan,French Guiana,French Polynesia,French Southern Territories,Gabon,Gambia,Georgia,Germany,Ghana,Gibraltar,Greece,Greenland,Grenada,Guadeloupe,Guinea-Bissau,Guyana,Haiti,Heard and McDonald Islands,Honduras,Hong Kong,Hungary,Iceland,India,Indonesia,Iran,Iraq,Ireland,Israel,Italy,Jamaica,Japan,Jordan,Kazakhstan (North),Korea (South),Kuwait,Kyrgyzstan,Laos,Latvia,Lebanon,Lesotho,Liberia,Libya,Liechtenstein,Lithuania,Luxembourg,Macau,Macao Islands,Martinique,Mauritania,Mauritius,Mayotte,Mexico,Micronesia,Moldova,Monaco,Mongolia,Montserrat,Antilles,New Caledonia,New Zealand,Nicaragua,Niger,Nigeria,Niue,Norfolk Island,Northern Mariana Islands,Norway,Oman,Pakistan,Palau,Panama,Papua New Guinea,Paraguay,Peru,Philippines,Pitcairn,Poland,Portugal,Puerto Rico,Qatar,Reunion,Romania,Russian Federation,Rwanda,Saint Kitts and Nevis,Saint Lucia,Saint Vincent and The Grenadines,Samoa,San Marino,Sao Tome and Principe,Saudi Arabia,Senegal,Seychelles,Sierra Leone,Singapore,Slovak Republic,Slovenia,Solomon Islands,Somalia,South Africa,S. Georgia and S. Sandwich Isls.,Spain,Sri Lanka,St. Helena,St. Pierre and Miquelon,Sudan,Suriname,Svalbard and Jan Mayen Islands,Swaziland,Sweden,Switzerland,Syria,Tajikistan,Tanzania,Thailand,Togo,Tokelau,Tonga,Trieste and Tobago,Tunisia,Turkey,Turkmenistan,Turks and Caicos Islands,Tuvalu,Uganda,Ukraine,United Arab Emirates,United Kingdom (Britain / UK),United States of America (USA),US Minor Outlying Islands,Uruguay,Uzbekistan,Vanuatu,Vatican City State (Holy See),Venezuela,Viet Nam,Virgin Islands (British),Virgin Islands (US),Wallis and Futuna Islands,Western Sahara,Yemen,Yugoslavia,Zaire,Zambia,Zimbabwe",
  "description": "What country were you born in?",
  "listType": "str",
  "isOptional": true
},
{
  "name": "promotion",
  "displayName": "Do you want to receive promotional newsletter from us?",

```

```

    "type": "bool",
    "isOptional": true
  }
]
}

```

사용자 지정 시각적 스크립트의 예

다음 예에서는 동일한 변환을 수행합니다. 하지만 두 번째 예제(SparkSQL)가 가장 깔끔하고 가장 효율적이며, 그 다음은 Pandas UDF이며, 첫 번째 예제의 하위 수준 매핑이 마지막입니다. 다음 예제는 두 열을 합산하는 간단한 변환의 전체 예입니다.

```

from awsglue import DynamicFrame

# You can have other auxiliary variables, functions or classes on this file, it won't
# affect the runtime
def record_sum(rec, col1, col2, resultCol):
    rec[resultCol] = rec[col1] + rec[col2]
    return rec

# The number and name of arguments must match the definition on json config file
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
# (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    # The mapping will alter the columns order, which could be important
    fields = [field.name for field in self.schema()]
    if resultCol not in fields:
        # If it's a new column put it at the end
        fields.append(resultCol)
    return self.map(lambda record: record_sum(record, col1, col2,
resultCol)).select_fields(paths=fields)

# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns

```

다음 예제는 SparkSQL API를 활용하는 동일한 변환입니다.

```

from awsglue import DynamicFrame

# The number and name of arguments must match the definition on json config file
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
# (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    df = self.toDF()
    return DynamicFrame.fromDF(
        df.withColumn(resultCol, df[col1] + df[col2]) # This is the conversion logic
        , self.glue_ctx, self.name)

# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns

```

다음 예제에서는 동일한 변환을 사용하지만 pandas UDF를 사용하므로 일반 UDF를 사용하는 것보다 더 효율적입니다. pandas UDF 작성에 대한 자세한 내용은 [Apache Spark SQL 설명서](#)를 참조하세요.

```

from awsglue import DynamicFrame
import pandas as pd
from pyspark.sql.functions import pandas_udf

# The number and name of arguments must match the definition on json config file
# (expect self which is the current DynamicFrame to transform
# If an argument is optional, you need to define a default value here
# (resultCol in this example is an optional argument)
def custom_add_columns(self, col1, col2, resultCol="result"):
    @pandas_udf("integer") # We need to declare the type of the result column
    def add_columns(value1: pd.Series, value2: pd.Series) # pd.Series:
        return value1 + value2

    df = self.toDF()
    return DynamicFrame.fromDF(
        df.withColumn(resultCol, add_columns(col1, col2)) # This is the conversion
        logic
        , self.glue_ctx, self.name)

# The name we assign on DynamicFrame must match the configured "functionName"
DynamicFrame.custom_add_columns = custom_add_columns

```

비디오

다음 동영상은 시각적 사용자 지정 변환을 소개하고 사용 방법을 보여줍니다.

AWS Glue Studio에서 데이터 레이크 프레임워크 사용

개요

오픈 소스 데이터 레이크 프레임워크는 Amazon S3에 빌드된 데이터 레이크에 저장된 파일의 증분 데이터 처리를 간소화합니다. AWS Glue 3.0 이상에서는 다음과 같은 오픈 소스 데이터 레이크 스토리지 프레임워크를 지원합니다.

- Apache Hudi
- Linux Foundation Delta Lake
- Apache Iceberg

AWS Glue 4.0을 기준으로 AWS Glue에서는 Amazon S3에 저장된 데이터를 트랜잭션 형태로 일관성 있게 읽고 쓸 수 있도록 이러한 프레임워크에 대한 기본 지원을 제공합니다. AWS Glue 작업에 이러한 프레임워크를 사용하기 위해 별도의 커넥터를 설치하거나 추가 구성 단계를 완료할 필요가 없습니다.

데이터 레이크 프레임워크는 Spark 스크립트 편집기 작업을 통해 AWS Glue Studio 내에서 소스 또는 대상으로 사용될 수 있습니다. Apache Hudi, Apache Iceberg, Delta Lake를 사용하는 방법에 대한 자세한 내용은 [AWS Glue ETL 작업에 데이터 레이크 프레임워크 사용](#)을 참조하세요.

AWS Glue 스트리밍 소스에서 오픈 테이블 형식 생성

AWS Glue 스트리밍 ETL 작업은 스트리밍 소스의 데이터를 지속적으로 소비하고, 전송 중인 데이터를 정리 및 변환하여 몇 초 만에 분석에 사용할 수 있도록 합니다.

AWS 요구 사항을 지원하는 다양한 서비스를 제공합니다. Database Migration Service와 같은 AWS 데이터베이스 복제 서비스는 일반적으로 데이터 레이크의 스토리지 계층을 호스팅하는 Amazon S3로 원본 시스템의 데이터를 복제할 수 있습니다. 온라인 소스 애플리케이션을 지원하는 관계형 데이터베이스 관리 시스템(RDBMS)에서 업데이트를 적용하는 것은 간단하지만 데이터 레이크에 이 CDC 프로세스를 적용하기는 어렵습니다. 오픈 소스 데이터 관리 프레임워크는 증분 데이터 처리 및 데이터 파이프라인 개발을 간소화하는 데 유용한 옵션입니다.

자세한 내용은 다음을 참조하세요.

- [AWS Glue 스트리밍을 사용하여 Apache Hudi 기반의 실시간에 가까운 트랜잭션 데이터 레이크를 만들 수 있습니다](#)

- [Build a real-time GDPR-aligned Apache Iceberg data lake](#)

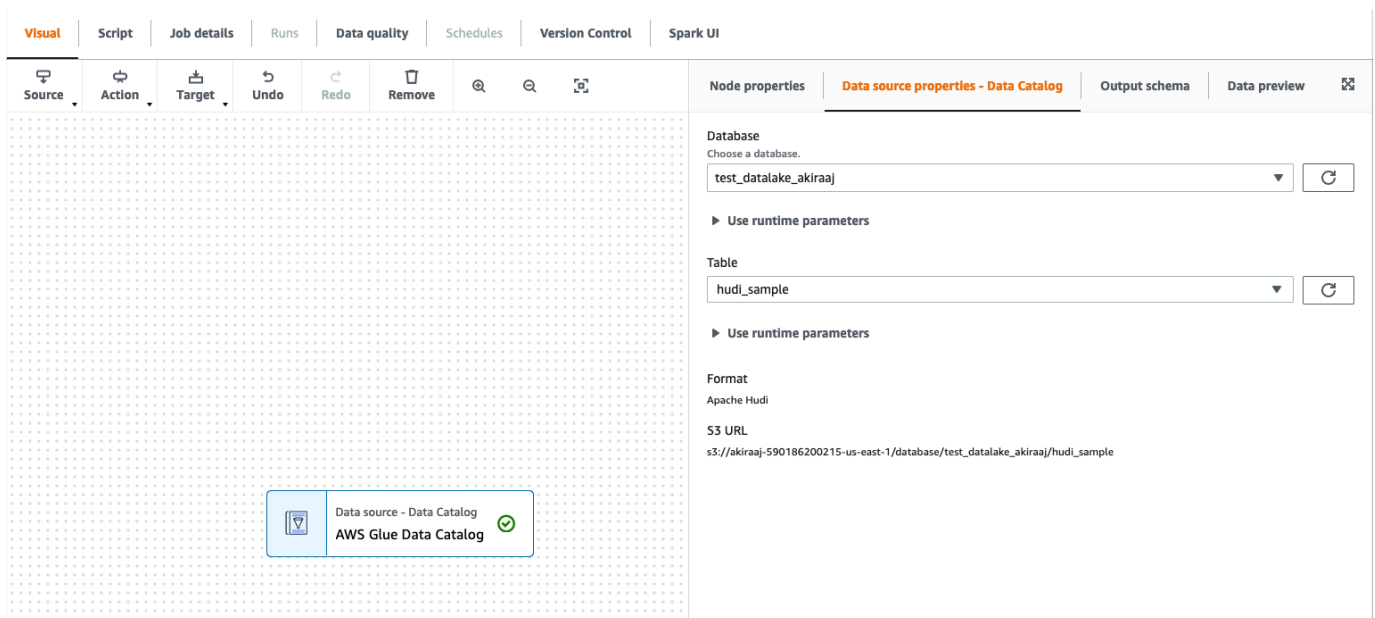
AWS Glue Studio에서 Hudi 프레임워크 사용

작업을 생성하거나 편집할 때 AWS Glue Studio는 사용 중인 AWS Glue의 버전에 따라 해당하는 Hudi 라이브러리를 자동으로 추가합니다. 자세한 내용은 [AWS Glue에서 Hudi 프레임워크 사용](#)을 참조하세요.

데이터 카탈로그 데이터 소스에서 Apache Hudi 프레임워크 사용

작업에 Hudi 데이터 소스 형식을 추가하려면


1. 소스 메뉴에서 AWS Glue Studio 데이터 카탈로그를 선택합니다.
2. 데이터 소스 속성 탭에서 데이터베이스와 테이블을 선택합니다.
3. AWS Glue Studio는 형식 유형을 Apache Hudi와 Amazon S3 URL로 표시합니다.



Amazon S3 데이터 소스에서 Hudi 프레임워크 사용

1. 소스 메뉴에서 Amazon S3를 선택합니다.
2. Amazon S3 소스 유형으로 데이터 카탈로그 테이블을 선택한 경우 데이터베이스와 테이블을 선택합니다.
3. AWS Glue Studio는 형식을 Apache Hudi와 Amazon S3 URL로 표시합니다.

4. Amazon S3 소스 유형으로 Amazon S3 위치를 선택한 경우 Amazon S3 찾아보기를 클릭하여 Amazon S3 URL을 선택합니다.
5. 데이터 형식에서 Apache Hudi를 선택합니다.

 Note

AWS Glue Studio가 선택한 Amazon S3 폴더 또는 파일에서 스키마를 유추할 수 없는 경우 추가 옵션을 선택하여 새 폴더 또는 파일을 선택합니다.
추가 옵션의 스키마 추론 아래에서 다음 옵션을 선택합니다.

- AWS Glue Studio에서 통해 샘플 파일을 자동으로 선택하도록 하겠습니까. AWS Glue Studio는 스키마를 추론할 수 있도록 Amazon S3 위치에서 샘플 파일을 선택합니다. 자동 샘플링된 파일 필드에서 자동으로 선택된 파일을 볼 수 있습니다.
- Amazon S3에서 샘플 파일을 선택합니다. Amazon S3 찾아보기를 클릭하여 사용할 Amazon S3 파일을 선택합니다.

6. 스키마 추론을 클릭합니다. 그런 다음 출력 스키마 탭을 클릭하여 출력 스키마를 볼 수 있습니다.
7. 키-값 페어를 입력하려면 추가 옵션을 선택합니다.

Additional options [Info](#)

Enter additional key-value pairs for your data source connection.

Key

Value



Add new option

데이터 대상에서 Apache Hudi 프레임워크 사용

데이터 카탈로그 데이터 대상에서 Apache Hudi 프레임워크 사용

1. 대상 메뉴에서 AWS Glue Studio 데이터 카탈로그를 선택합니다.
2. 데이터 소스 속성 탭에서 데이터베이스와 테이블을 선택합니다.
3. AWS Glue Studio는 형식 유형을 Apache Hudi와 Amazon S3 URL로 표시합니다.

Amazon S3 데이터 대상에서 Apache Hudi 프레임워크 사용

값을 입력하거나 사용 가능한 옵션 중에서 선택하여 Apache Hudi 형식을 구성합니다. Apache Hudi에 대한 자세한 내용은 [Apache Hudi 설명서](#)를 참조하세요.

Node properties
Data target properties - S3 2
Output schema
Data preview ✕

Format

Apache Hudi
▼

Hudi Table Name

Hudi Storage Type

Copy on write
Recommended for optimizing read performance

Merge on read
Recommended for minimizing write latency

Hudi Write Operation

Upsert
▼

Hudi Record Key Fields
Set your primary key(s)

Select a source location to view schema
▼

Hudi Deduplicate Records by Field Value
Set your field to choose the largest value when inserting records with duplicate key(s)

Select a source location to view schema
▼

Compression Type

GZIP
▼

S3 Target Location
Choose an S3 location in the format s3://bucket/prefix/object/ with a trailing slash (/).

🔍

View
🔗

Browse S3

Data Catalog update options [Info](#)

Choose how you want to update the Data Catalog table's schema and partitions. These options will only apply if the Data Catalog table is an S3 backed source.

Do not update the Data Catalog

Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions

Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions

Partition keys - optional
Add partition keys.

- Hudi 테이블 이름 - HUDI 테이블의 이름입니다.
- Hudi 스토리지 유형 - 다음 두 가지 옵션 중에서 선택합니다.
 - 쓸 때 복사 - 읽기 성능을 최적화하기 위해 권장됩니다. Hudi 스토리지의 기본 유형입니다. 업데이트할 때마다 쓰기 중에 새 버전의 파일이 생성됩니다.
 - 읽을 때 병합 - 쓰기 지연 시간을 최소화하기 위해 권장됩니다. 업데이트는 행 기반 delta 파일에 기록되며 새 버전의 열 형식 파일을 작성할 때 필요에 따라 압축됩니다.
- Hudi 쓰기 작업 - 다음 옵션 중에서 선택합니다.
 - Upsert - 인덱스를 조회하여 입력 레코드를 먼저 삽입 또는 업데이트로 태깅하는 기본 작업입니다. 기존 데이터를 업데이트하는 경우에 권장됩니다.
 - 삽입 - 레코드를 삽입하지만 기존 레코드를 확인하지 않으므로 중복이 발생할 수 있습니다.
 - 대량 삽입 - 레코드를 삽입하며 대량의 데이터에 권장됩니다.
- Hudi 레코드 키 필드 - 검색 창을 사용하여 기본 레코드 키를 검색하고 선택합니다. Hudi의 레코드는 한 쌍의 레코드 키와 해당 레코드가 속한 파티션 경로인 프라이머리 키로 식별됩니다.
- Hudi 사전 결합 필드 - 실제 쓰기 전에 사전 결합에 사용되는 필드입니다. 두 레코드에 동일한 키 값이 있는 경우 AWS Glue Studio는 사전 결합 필드에 대해 가장 큰 값을 가진 레코드를 선택합니다. 증분 값(예: updated_at)이 속한 필드를 설정합니다.
- 압축 유형 - 압축 유형 옵션(비압축, GZIP, LZO 또는 Snappy) 중 하나를 선택합니다.
- Amazon S3 대상 위치 - S3 찾아보기를 클릭하여 Amazon S3 대상 위치를 선택합니다.
- 데이터 카탈로그 업데이트 옵션 - 다음 옵션 중에 선택합니다.
 - [데이터 카탈로그 업데이트 안 함(Do not update the Data Catalog)]: (기본값) 스키마가 변경되거나 새 파티션이 추가된 경우에도 작업에서 데이터 카탈로그를 업데이트하지 않으려면 이 옵션을 선택합니다.
 - 데이터 카탈로그에 테이블 생성, 후속 실행 시 스키마 업데이트 및 새 파티션 추가: 이 옵션을 선택하면 작업이 처음 실행될 때 데이터 카탈로그에 테이블이 생성됩니다. 후속 작업 실행 시 스키마가 변경되거나 새 파티션이 추가되면 작업이 데이터 카탈로그 테이블을 업데이트합니다.

또한 데이터 카탈로그에서 데이터베이스를 선택하고 테이블 이름을 입력해야 합니다.
- [데이터 카탈로그에 테이블 생성, 기존 스키마 유지 및 새 파티션 추가(Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions)]: 이 옵션을 선택하면 작업이 처음 실행될 때 데이터 카탈로그에 테이블이 생성됩니다. 후속 작업 실행 시 작업은 새 파티션을 추가하기 위해서만 데이터 카탈로그 테이블을 업데이트합니다.

또한 데이터 카탈로그에서 데이터베이스를 선택하고 테이블 이름을 입력해야 합니다.

- [파티션 키(Partition keys)]: 출력에서 파티션 키로 사용할 열을 선택합니다. 파티션 키를 더 추가하려면 [파티션 키 추가(Add a partition key)]를 선택합니다.
- 추가 옵션 - 필요에 따라 키-값 페어를 입력합니다.

AWS Glue Studio를 통해 코드 생성

작업이 저장될 때 Hudi 소스 또는 대상이 감지되면 다음 작업 파라미터가 작업에 추가됩니다.

- `--datalake-formats` - 시각적 작업에서 감지된 데이터 레이크 형식의 개별 목록('형식'을 선택하여 직접적으로 또는 데이터 레이크가 지원하는 카탈로그 테이블을 선택하여 간접적으로)
- `--conf` - `---datalake-formats`의 값을 기반으로 생성됩니다. 예를 들어, `---datalake-formats`의 값이 'hudi'인 경우 AWS Glue에 이 파라미터에 대한 `spark.serializer=org.apache.spark.serializer.KryoSerializer --conf spark.sql.hive.convertMetastoreParquet=false` 값이 생성됩니다.

AWS Glue 제공 라이브러리 재정의

AWS Glue에서 지원하지 않는 Hudi 버전을 사용하려는 경우 자체 Hudi 라이브러리 JAR 파일을 지정하면 됩니다. 자체 JAR 파일을 사용하려면:

- `--extra-jars` 작업 파라미터를 사용합니다. 예를 들면 '`--extra-jars`': `'s3pathtojarfile.jar'`입니다. 자세한 내용을 알아보려면 [AWS Glue 작업 파라미터](#)를 참조하세요.
- `--datalake-formats` 작업 파라미터의 값으로 hudi를 포함하지 마세요. 빈 문자열을 값으로 입력하면 AWS Glue에서 데이터 레이크 라이브러리를 자동으로 제공하지 않습니다. 자세한 내용은 [AWS Glue에서 Hudi 프레임워크 사용](#)을 참조하세요.

AWS Glue Studio에서 Delta Lake 프레임워크 사용

데이터 소스에서 Delta Lake 프레임워크 사용

Amazon S3 데이터 소스에서 Delta Lake 프레임워크 사용

1. 소스 메뉴에서 Amazon S3를 선택합니다.
2. Amazon S3 소스 유형으로 데이터 카탈로그 테이블을 선택한 경우 데이터베이스와 테이블을 선택합니다.
3. AWS Glue Studio는 형식을 Delta Lake 및 Amazon S3 URL로 표시합니다.

- 키-값 페어를 입력하려면 추가 옵션을 선택합니다. 예를 들어 키-값 페어는 키: timestampAsOf와 값: 2023-02-24 14:16:18일 수 있습니다.

Additional options [Info](#)

Enter additional key-value pairs for your data source connection.

Key	Value	
<input type="text"/>	<input type="text"/>	
<input type="button" value="Add new option"/>		

- Amazon S3 소스 유형으로 Amazon S3 위치를 선택한 경우 Amazon S3 찾아보기를 클릭하여 Amazon S3 URL을 선택합니다.
- 데이터 형식에서 델타 레이크를 선택합니다.

Note

AWS Glue Studio가 선택한 Amazon S3 폴더 또는 파일에서 스키마를 유추할 수 없는 경우 추가 옵션을 선택하여 새 폴더 또는 파일을 선택합니다.
추가 옵션의 스키마 추론 아래에서 다음 옵션을 선택합니다.

- AWS Glue Studio에서 통해 샘플 파일을 자동으로 선택하도록 하겠습니까. AWS Glue Studio는 스키마를 추론할 수 있도록 Amazon S3 위치에서 샘플 파일을 선택합니다. 자동 샘플링된 파일 필드에서 자동으로 선택된 파일을 볼 수 있습니다.
- Amazon S3에서 샘플 파일을 선택합니다. Amazon S3 찾아보기를 클릭하여 사용할 Amazon S3 파일을 선택합니다.

- 스키마 추론을 클릭합니다. 그런 다음 출력 스키마 탭을 클릭하여 출력 스키마를 볼 수 있습니다.

데이터 카탈로그 데이터 소스에서 Delta Lake 프레임워크 사용

- 소스 메뉴에서 AWS Glue Studio 데이터 카탈로그를 선택합니다.
- 데이터 소스 속성 탭에서 데이터베이스와 테이블을 선택합니다.
- AWS Glue Studio는 형식 유형을 Delta Lake 및 Amazon S3 URL로 표시합니다.

Note

Delta Lake 소스가 AWS Glue 데이터 카탈로그 테이블로 등록되지 않은 경우 다음 두 가지 옵션을 사용할 수 있습니다.

1. Delta Lake 데이터 스토어에 대한 AWS Glue 크롤러를 생성합니다. 자세한 내용은 [Delta Lake 데이터 스토어에 대한 구성 옵션을 지정하는 방법을 참조하세요](#).
2. Amazon S3 데이터 소스를 사용하여 Delta Lake 데이터 소스를 선택합니다. [Amazon S3 데이터 소스에서 Delta Lake 프레임워크 사용](#) 섹션을 참조하세요.

데이터 대상에서 Delta Lake 형식 사용

데이터 카탈로그 데이터 대상에서 Delta Lake 형식 사용

1. 대상 메뉴에서 AWS Glue Studio 데이터 카탈로그를 선택합니다.
2. 데이터 소스 속성 탭에서 데이터베이스와 테이블을 선택합니다.
3. AWS Glue Studio는 형식 유형을 Delta Lake 및 Amazon S3 URL로 표시합니다.

Amazon S3 데이터 소스에서 Delta Lake 형식 사용

값을 입력하거나 사용 가능한 옵션 중에서 선택하여 Delta Lake 형식을 구성합니다.

- 압축 유형 - 압축 유형 옵션(비압축 또는 Snappy) 중 하나를 선택합니다.
- Amazon S3 대상 위치 - S3 찾아보기를 클릭하여 Amazon S3 대상 위치를 선택합니다.
- 데이터 카탈로그 업데이트 옵션 - Glue Studio 시각적 편집기에서 이 형식에 대한 데이터 카탈로그 업데이트는 지원되지 않습니다.
- [데이터 카탈로그 업데이트 안 함(Do not update the Data Catalog)]: (기본값) 스키마가 변경되거나 새 파티션이 추가된 경우에도 작업에서 데이터 카탈로그를 업데이트하지 않으려면 이 옵션을 선택합니다.
- AWS Glue 작업 실행 후 데이터 카탈로그를 업데이트하려면 AWS Glue 크롤러를 실행하거나 일정을 예약합니다. 자세한 내용은 [Delta Lake 데이터 스토어에 대한 구성 옵션을 지정하는 방법을 참조하세요](#).
- 파티션 키 - 출력에서 파티션 키로 사용할 열을 선택합니다. 파티션 키를 더 추가하려면 [파티션 키 추가(Add a partition key)]를 선택합니다.

- 선택 사항으로 추가 옵션을 선택하여 키-값 페어를 입력합니다. 예를 들어 키-값 페어는 키: timestampAsOf와 값: 2023-02-24 14:16:18일 수 있습니다.

AWS Glue Studio에서 Apache Iceberg 프레임워크 사용

데이터 대상에서 Apache Iceberg 프레임워크 사용

데이터 카탈로그 데이터 대상에서 Apache Iceberg 프레임워크 사용

1. 대상 메뉴에서 AWS Glue Studio 데이터 카탈로그를 선택합니다.
2. 데이터 소스 속성 탭에서 데이터베이스와 테이블을 선택합니다.
3. AWS Glue Studio는 형식 유형을 Apache Iceberg 및 Amazon S3 URL로 표시합니다.

Amazon S3 데이터 대상에서 Apache Iceberg 프레임워크 사용

값을 입력하거나 사용 가능한 옵션 중에서 선택하여 Apache Iceberg 형식을 구성합니다.

- 형식 - 드롭다운 메뉴에서 Apache Iceberg를 선택합니다.
- Amazon S3 대상 위치 - S3 찾아보기를 클릭하여 Amazon S3 대상 위치를 선택합니다.
- 데이터 카탈로그 업데이트 옵션 - 계속 진행하려면 데이터 카탈로그에 테이블 생성, 후속 실행 시 기존 스키마 유지 및 새 파티션 추가를 선택해야 합니다. AWS Glue를 사용하여 새 Iceberg 테이블을 작성하려면 Data Catalog를 Iceberg 테이블의 카탈로그로 구성해야 합니다. Data Catalog에 등록된 기존 Iceberg 테이블을 업데이트하려면 Data Catalog를 대상으로 선택합니다.
- 데이터베이스 - Data Catalog에서 데이터베이스를 선택합니다.
- 테이블 이름 - 사용자의 테이블 이름에 대한 값을 입력합니다. Apache Iceberg 테이블 이름은 모두 소문자여야 합니다. 공백은 허용되지 않으므로 필요한 경우 밑줄을 사용합니다. 예를 들어 'data_lake_format_tables'와 같습니다.

Node properties	Data target properties - S3	Output schema	Data preview
-----------------	-----------------------------	---------------	--------------

Format

Apache Iceberg

Compression Type

GZIP

S3 Target Location

Choose an S3 location in the format `s3://bucket/prefix/object/` with a trailing slash (/).

s3://data-lake-format-data/output/ View Browse S3

Data Catalog update options

Choose how you want to update the Data Catalog table's schema and partitions. These options will only apply if the Data Catalog table is an S3 backed source.

- Do not update the Data Catalog
- Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions
- Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions

Database

Choose the database from the AWS Glue Data Catalog.

data_lake_format_tables Refresh

▶ Use runtime parameters

Table name

Enter a table name for the AWS Glue Data Catalog.

my_new_table

Amazon S3 데이터 소스에서 Apache Iceberg 프레임워크 사용

데이터 카탈로그 데이터 소스에서 Apache Iceberg 프레임워크 사용

1. 소스 메뉴에서 AWS Glue Studio 데이터 카탈로그를 선택합니다.
2. 데이터 소스 속성 탭에서 데이터베이스와 테이블을 선택합니다.
3. AWS Glue Studio는 형식 유형을 Apache Iceberg 및 Amazon S3 URL로 표시합니다.

Node properties	Data source properties - S3	Output schema	Data preview
<p>S3 source type</p> <p><input type="radio"/> S3 location Choose a file or folder in an S3 bucket.</p> <p><input checked="" type="radio"/> Data Catalog table</p>			
<p>Database Choose a database.</p> <p>data_lake_format_tables ▼ ↻</p> <p>▶ Use runtime parameters</p>			
<p>Table</p> <p>source_iceberg ▼ ↻</p> <p>▶ Use runtime parameters</p>			
<p>Format Apache Iceberg</p>			
<p>S3 URL s3://data-lake-format-data/iceberg/ ↗</p>			
<p>Partition predicate - optional Enter a boolean expression supported by Spark SQL, using only partition columns.</p> <p><input type="text"/></p> <p>Partition predicate syntax for Spark SQL is <code>year == year(date_sub(current_date, 7)) AND month == month(date_sub(current_date, 7)) AND day == day(date_sub(current_date, 7))</code>.</p>			

Amazon S3 데이터 소스에서 Apache Iceberg 프레임워크 사용

Apache Iceberg는 AWS Glue Studio에서 Amazon S3 소스 노드의 데이터 옵션으로 사용할 수 없습니다.

데이터 대상 노드 구성

데이터 대상은 작업이 변환된 데이터를 쓰는 위치입니다.

데이터 대상 옵션 개요

다음은 데이터 대상(데이터 싱크라고도 함)일 수 있습니다.

- [S3] - 작업이 선택한 Amazon S3 위치와 지정한 포맷의 파일에 데이터를 씁니다.

데이터 대상에 대한 파티션 열을 구성하면 작업이 파티션 키를 기반으로 디렉터리에 Amazon S3에 대한 데이터 집합을 씁니다.

- AWS Glue Data Catalog - 작업에서 데이터 카탈로그의 테이블과 연결된 정보를 사용하여 출력 데이터를 대상 위치에 씁니다.

수동으로 또는 크롤러를 사용하여 테이블을 생성할 수 있습니다. AWS CloudFormation 템플릿을 사용하여 데이터 카탈로그에서 테이블을 생성할 수도 있습니다.

- 커넥터 - 커넥터는 데이터 스토어와 AWS Glue 간의 통신을 용이하게 하는 코드입니다. 작업은 커넥터 및 연결된 연결을 사용하여 출력 데이터를 대상 위치에 씁니다. AWS Marketplace에서 제공되는 커넥터를 구독하거나 사용자 정의 커넥터를 생성할 수 있습니다. 자세한 내용은 [AWS Glue Studio에 커넥터 추가](#) 섹션을 참조하세요.

작업이 Amazon S3 데이터 대상에 쓸 때 데이터 카탈로그를 업데이트하도록 선택할 수 있습니다. 스키마 또는 파티션이 변경될 때 크롤러가 데이터 카탈로그를 업데이트하도록 요구하는 대신 이 옵션을 사용하면 테이블을 쉽게 최신 상태로 유지할 수 있습니다. 이 옵션은 필요에 따라 데이터 카탈로그에 새 테이블을 추가하고 테이블 파티션을 업데이트하며 작업에서 직접 테이블의 스키마를 업데이트하여 분석에 데이터를 사용할 수 있도록 하는 프로세스를 단순화합니다.

데이터 대상 노드 편집

데이터 대상은 작업이 변환된 데이터를 쓰는 위치입니다.

작업 다이어그램에서 데이터 대상 노드를 추가하거나 구성하려면

1. (선택 사항) 대상 노드를 추가해야 하는 경우 시각적 편집기 상단의 도구 모음에서 Target(대상)을 선택한 다음 S3 또는 Glue Data Catalog(Glue 데이터 카탈로그)를 선택합니다.
 - 대상으로 [S3]를 선택하면 작업은 지정한 Amazon S3 위치에 있는 하나 이상의 파일에 데이터 집합을 씁니다.
 - 대상으로 [AWS Glue Data Catalog]를 선택하면 작업은 데이터 카탈로그에서 선택한 테이블에 설명된 위치에 씁니다.
2. 작업 다이어그램에서 데이터 대상 노드를 선택합니다. 노드를 선택하면 페이지 오른쪽에 노드 세부 정보 패널이 나타납니다.
3. [노드 속성(Node properties)] 탭을 선택한 후 다음 정보를 입력합니다.
 - [이름(Name)]: 작업 다이어그램의 노드와 연결할 이름을 입력합니다.

- [노드 유형(Node type)]: 값이 이미 선택되어 있어야 하지만 필요에 따라 변경할 수 있습니다.
- [상위 노드(Node parents)]: 상위 노드는 대상 위치에 쓰려는 출력 데이터를 제공하는 작업 다이어그램의 노드입니다. 미리 채워진 작업 다이어그램의 경우 대상 노드에 이미 상위 노드가 선택되어 있어야 합니다. 표시되는 상위 노드가 없으면 목록에서 상위 노드를 선택합니다.

대상 노드에는 단일 상위 노드가 있습니다.

4. [데이터 대상 속성(Data target properties)] 정보를 구성합니다. 자세한 내용은 다음 단원을 참조하세요.

- [데이터 대상에 Amazon S3 사용](#)
- [데이터 대상에 데이터 카탈로그 테이블 사용](#)
- [데이터 대상에 커넥터 사용](#)

5. (선택 사항) 데이터 대상 노드 속성을 구성한 후 노드 세부 정보 패널에서 [출력 스키마(Output schema)] 탭을 선택하여 데이터에 대해 출력 스키마를 볼 수 있습니다. 작업의 노드에 대해 이 탭을 처음 선택하면 데이터 액세스를 위해 IAM 역할을 제공하라는 메시지가 나타납니다. [작업 세부 정보(Job details)] 탭에서 IAM 역할을 지정하지 않은 경우 여기에 IAM 역할을 입력하라는 메시지가 나타납니다.

데이터 대상에 Amazon S3 사용

Amazon S3와 커넥터를 제외한 모든 데이터 원본의 경우 선택한 원본 유형의 테이블이 AWS Glue Data Catalog에 있어야 합니다. AWS Glue Studio는 데이터 카탈로그 테이블을 생성하지 않습니다.

Amazon S3에 쓰는 데이터 대상 노드를 구성하려면

1. 새 작업 또는 저장된 작업의 시각적 편집기로 이동합니다.
2. 작업 다이어그램에서 데이터 원본 노드를 선택합니다.
3. [데이터 원본 속성(Data source properties)] 탭을 선택한 후 다음 정보를 입력합니다.
 - [포맷(Format)]: 목록에서 포맷을 선택합니다. 데이터 결과에 사용할 수 있는 포맷 유형은 다음과 같습니다.
 - [JSON]: JavaScript Object Notation.
 - [CSV]: 쉼표로 분리된 값.
 - [Avro]: Apache Avro JSON 바이너리입니다.
 - [Parquet]: Apache Parquet 컬럼 방식 스토리지.

- [Glue Parquet]: 데이터 포맷으로 DynamicFrames에 최적화된 사용자 정의 Parquet 라이터 유형입니다. 데이터에 대해 미리 계산된 스키마를 요구하는 대신 스키마를 동적으로 계산하고 수정합니다.
- [ORC]: Apache Optimized Row Columnar(ORC) 포맷입니다.

이러한 포맷 옵션에 대한 자세한 내용은 AWS Glue Developer Guide의 [Format Options for ETL Inputs and Outputs in AWS Glue](#)를 참조하세요.

- [압축 유형(Compression Type)]: gzip 또는 bzip2 포맷을 사용하여 데이터를 선택적으로 압축하도록 선택할 수 있습니다. 기본값은 압축 안 함 또는 [없음(None)]입니다.
- [S3 대상 위치(S3 Target Location)]: 데이터 출력을 위한 Amazon S3 버킷 및 위치입니다. [S3 찾아보기(Browse S3)] 버튼을 선택하여 액세스 권한이 있는 Amazon S3 버킷을 확인하고 그 중 하나를 대상으로 선택할 수 있습니다.
- 데이터 카탈로그 업데이트 옵션
 - [데이터 카탈로그 업데이트 안 함(Do not update the Data Catalog)]: (기본값) 스키마가 변경되거나 새 파티션이 추가된 경우에도 작업에서 데이터 카탈로그를 업데이트하지 않으려면 이 옵션을 선택합니다.
 - [데이터 카탈로그에 테이블 생성, 후속 실행 시 스키마 업데이트 및 새 파티션 추가(Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions)]: 이 옵션을 선택하면 작업이 처음 실행될 때 데이터 카탈로그에 테이블이 생성됩니다. 후속 작업 실행 시 스키마가 변경되거나 새 파티션이 추가되면 작업이 데이터 카탈로그 테이블을 업데이트합니다.

또한 데이터 카탈로그에서 데이터베이스를 선택하고 테이블 이름을 입력해야 합니다.

- [데이터 카탈로그에 테이블 생성, 기존 스키마 유지 및 새 파티션 추가(Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions)]: 이 옵션을 선택하면 작업이 처음 실행될 때 데이터 카탈로그에 테이블이 생성됩니다. 후속 작업 실행 시 작업은 새 파티션을 추가하기 위해서만 데이터 카탈로그 테이블을 업데이트합니다.

또한 데이터 카탈로그에서 데이터베이스를 선택하고 테이블 이름을 입력해야 합니다.

- [파티션 키(Partition keys)]: 출력에서 파티션 키로 사용할 열을 선택합니다. 파티션 키를 더 추가하려면 [파티션 키 추가(Add a partition key)]를 선택합니다.

데이터 대상에 데이터 카탈로그 테이블 사용

Amazon S3과 커넥터를 제외한 모든 데이터 원본의 경우 선택한 대상 유형의 테이블이 AWS Glue Data Catalog에 있어야 합니다. AWS Glue Studio는 데이터 카탈로그 테이블을 생성하지 않습니다.

데이터 카탈로그 테이블을 사용하는 대상에 대한 데이터 속성을 구성하려면

1. 새 작업 또는 저장된 작업의 시각적 편집기로 이동합니다.
2. 작업 다이어그램에서 데이터 대상 노드를 선택합니다.
3. [데이터 대상 속성(Data target properties)] 탭을 선택한 후 다음 정보를 입력합니다.
 - [데이터베이스(Database)]: 목록에서 대상으로 사용할 테이블이 포함된 데이터베이스를 선택합니다. 이 데이터베이스가 데이터 카탈로그에 이미 존재해야 합니다.
 - [테이블(Table)]: 목록에서 출력 데이터의 스키마를 정의하는 테이블을 선택합니다. 이 테이블이 데이터 카탈로그에 이미 존재해야 합니다.

데이터 카탈로그의 테이블은 열 이름, 데이터 유형 정의, 파티션 정보 및 대상 데이터 집합에 대한 기타 메타데이터로 구성됩니다. 작업은 데이터 카탈로그에서 이 테이블에 설명된 위치에 있습니다.

데이터 카탈로그에 테이블 생성에 대한 자세한 내용은 AWS Glue Developer Guide의 [Defining Tables in the Data Catalog](#)를 참조하세요.

- 데이터 카탈로그 업데이트 옵션
 - [테이블 정의 변경 안 함(Do not change table definition)]: (기본값) 스키마가 변경되거나 새 파티션이 추가된 경우에도 작업에서 데이터 카탈로그를 업데이트하지 않으려면 이 옵션을 선택합니다.
 - [스키마 업데이트 및 새 파티션 추가(Update schema and add new partitions)]: 이 옵션을 선택하면 스키마가 변경되거나 새 파티션이 추가될 때 작업이 데이터 카탈로그 테이블을 업데이트합니다.
 - [기존 스키마 유지 및 새 파티션 추가(Keep existing schema and add new partitions)]: 이 옵션을 선택하면 작업에서 새 파티션을 추가하기 위해서만 데이터 카탈로그 테이블을 업데이트합니다.
 - [파티션 키(Partition keys)]: 출력에서 파티션 키로 사용할 열을 선택합니다. 파티션 키를 더 추가하려면 [파티션 키 추가(Add a partition key)]를 선택합니다.

데이터 대상에 커넥터 사용

[노드 유형(Node type)]에 대한 커넥터를 선택하는 경우 [사용자 정의 커넥터로 작업 작성](#)의 지침에 따라 데이터 대상 속성 구성을 완료합니다.

작업 스크립트 편집 또는 업로드

AWS Glue Studio 시각적 편집기를 사용하여 작업 스크립트를 편집하거나 고유한 스크립트를 업로드합니다.

작업이 AWS Glue Studio로 생성된 경우에만 시각적 편집기를 사용하여 작업 노드를 편집할 수 있습니다. 작업이 AWS Glue 콘솔, API 명령 또는 명령줄 인터페이스(CLI)로 생성된 경우 AWS Glue Studio의 스크립트 편집기를 사용하여 작업 스크립트, 파라미터, 일정을 편집할 수 있습니다. 작업을 스크립트 전용 모드로 변환하여 AWS Glue Studio에서 생성된 작업의 스크립트를 편집할 수도 있습니다.

작업 스크립트를 편집하거나 자체 스크립트를 업로드하려면

1. 새 작업을 생성하는 경우 [작업(Jobs)] 페이지에서 [Spark 스크립트 편집기(Spark script editor)] 옵션을 선택하여 Spark 작업을 생성하거나 [Python 셸 스크립트 편집기(Python Shell script editor)]를 선택하여 Python 셸 작업을 생성합니다. 새 스크립트를 작성하거나 기존 스크립트를 업로드할 수 있습니다. [Spark 스크립트 편집기(Spark script editor)]를 선택하면 Scala 또는 Python 스크립트를 작성하거나 업로드할 수 있습니다. [Python Shell 스크립트 편집기(Python Shell script editor)]를 선택하면 Python 스크립트만 작성하거나 업로드할 수 있습니다.

새 작업을 만드는 옵션을 선택한 후 나타나는 [옵션(Options)] 섹션에서 시작 스크립트로 시작하거나([표준 문안 코드로 새 스크립트 생성(Create a new script with boilerplate code)]) 작업 스크립트로 사용할 로컬 파일을 업로드할 수 있습니다.

[Spark 스크립트 편집기(Spark script editor)]를 선택한 경우 Python 또는 Scala 스크립트 파일을 업로드할 수 있습니다. Scala 스크립트는 파일 확장명이 `.scala`여야 합니다. Python 스크립트는 Python 유형의 파일로 인식되어야 합니다. [Python Shell 스크립트 편집기(Python Shell script editor)]를 선택하면 Python 스크립트 파일만 업로드할 수 있습니다.

선택을 마쳤으면 [생성(Create)]을 선택하여 작업을 생성하고 시각적 편집기를 엽니다.

2. 새 작업 또는 저장된 작업에 대한 시각적 작업 편집기로 이동한 다음 [스크립트(Script)] 탭을 선택합니다.
3. 스크립트 편집기 옵션 중 하나를 사용하여 새 작업을 생성하지 않았고 기존 작업에 대한 스크립트를 편집한 적이 없는 경우 [스크립트(Script)] 탭에 머리글 [스크립트(잠김)(Script (Locked))]가 표시됩니다. 이는 스크립트 편집기가 읽기 전용 모드를 의미합니다. [스크립트 편집(Edit script)]을 선택하여 편집할 스크립트의 잠금을 해제합니다.

스크립트를 편집 가능하게 만들기 위해 AWS Glue Studio는 작업을 시각적 작업에서 스크립트 전용 작업으로 변환합니다. 편집을 위해 스크립트를 잠금 해제하면 저장한 후 이 작업에 대해 더 이상 시각적 편집기를 사용할 수 없습니다.

확인 창에서 [확인(Confirm)]을 선택하여 계속하거나 [취소(Cancel)]를 선택하여 작업을 시각적 편집에 사용할 수 있도록 유지합니다.

[확인(Confirm)]를 선택하면 [시각적(Visual)] 탭이 더 이상 편집기에 표시되지 않습니다. AWS Glue Studio를 사용하여 스크립트 편집기로 스크립트를 수정하거나, 작업 세부 정보 또는 일정을 수정하거나, 작업 실행을 볼 수 있습니다.

Note

작업을 저장할 때까지 스크립트 전용 작업으로의 변환은 영구적이지 않습니다. 콘솔 웹 페이지를 새로 고치거나 저장하기 전에 작업을 다고 시각적 편집기에서 다시 열면 시각적 편집기에서 개별 노드를 계속 편집할 수 있습니다.

4. 필요에 따라 스크립트를 편집합니다.

스크립트 편집을 마치면 [저장(Save)]을 선택하여 작업을 저장하고 시각적 객체에서 스크립트 전용으로 작업을 영구적으로 변환합니다.

5. (선택 사항) AWS Glue Studio 콘솔의 스크립트(Script) 탭에서 다운로드(Download) 버튼을 선택하여 스크립트를 다운로드할 수 있습니다. 이 버튼을 선택하면 새 브라우저 창이 열리고 Amazon S3의 해당 위치에 있는 스크립트가 표시됩니다. 작업의 [작업 세부 정보(Job details)] 탭에 있는 [스크립트 파일 이름(Script filename)] 및 [스크립트 경로(Script path)] 파라미터는 Amazon S3에 있는 스크립트 파일의 이름과 위치를 결정합니다.

Join test job2

Visual | Script | **Job details** | Runs | Schedules

▼ Advanced properties

Script filename

Join test job.py

Script path

S3 location of the script. Path must be in the form s3://bucket/prefix/path/. It must end with a slash (/) and not include any files.

🔍 s3://aws-glue-assets-111122223333-t ✕

View ↗

Browse S3

- Job metrics [Info](#)
Enable the creation of CloudWatch metrics when this job runs.
- Continuous logging [Info](#)
Enable logs in CloudWatch.
- Spark UI [Info](#)
Enable using Spark UI for monitoring this job.

작업을 저장할 때 AWS Glue는 이 필드에서 지정한 위치에 작업 스크립트를 저장합니다. Amazon S3 내의 이 위치에서 스크립트 파일을 수정할 경우 AWS Glue Studio는 다음에 작업을 편집할 때 수정된 스크립트를 로드합니다.

AWS Glue Studio에서 Scala 스크립트 생성 및 편집

작업 생성을 위해 스크립트 편집기를 선택하면 기본적으로 작업 프로그래밍 언어가 Python 3로 설정됩니다. 스크립트를 업로드하는 대신 새 스크립트를 작성하도록 선택할 경우 AWS Glue Studio는 Python으로 작성된 표준 문안 텍스트로 새 스크립트를 시작합니다. 대신 Scala 스크립트를 작성하려면 먼저 Scala를 사용하도록 스크립트 편집기를 구성해야 합니다.

i Note

Scala를 작업의 프로그래밍 언어로 선택하고 시각적 편집기를 사용하여 작업을 설계하면 생성된 작업 스크립트가 Scala로 작성되고 추가 작업이 필요하지 않습니다.

AWS Glue Studio에서 새 Scala 스크립트를 작성하려면

1. [Spark 스크립트 편집기(Spark script editor)] 옵션을 선택하여 새 작업을 생성합니다.
2. [옵션(Options)]에서 [표준 문안 코드로 새 스크립트 생성(Create a new script with boilerplate code)]을 선택합니다.
3. [작업 세부 정보(Job details)] 탭을 선택하고 [언어(Language)]를 Scala(Python 3 대신)로 설정합니다.

Note

작업을 생성하기 위해 [Spark 스크립트 편집기(Spark script editor)] 옵션을 선택하면 작업의 [유형(Type)] 속성이 Spark로 자동 설정됩니다.

4. [스크립트(Script)] 탭을 클릭합니다.
5. Python 표준 문안 텍스트를 제거합니다. 다음 Scala 표준 문안 텍스트로 바꿀 수 있습니다.

```
import com.amazonaws.services.glue.{DynamicRecord, GlueContext}
import org.apache.spark.SparkContext
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job

object MyScript {
  def main(args: Array[String]): Unit = {
    val sc: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(sc)

  }
}
```

6. 편집기에서 Scala 작업 스크립트를 작성합니다. 필요에 따라 import 문을 더 추가합니다.

AWS Glue Studio에서 Python 셸 작업 생성 및 편집

작업을 생성하기 위해 Python 셸 스크립트 편집기를 선택하면 기존 Python 스크립트를 업로드하거나 새 스크립트를 작성할 수 있습니다. 새 스크립트를 작성하도록 선택하면 표준 문안 코드가 새 Python 작업 스크립트에 추가됩니다.

새 Python 셸 작업을 생성하려면

[AWS Glue Studio에서 작업 시작](#)의 지침을 참조하세요.

Python 셸 작업에 지원되는 작업 속성은 Spark 작업에 지원되는 속성과 동일하지 않습니다. 다음 목록은 [작업 세부 정보(Job details)] 탭에서 Python 셸 작업에 사용 가능한 작업 파라미터의 변경 사항을 설명합니다.

- 작업의 [유형(Type)] 속성은 Python Shell로 자동 설정되며 변경할 수 없습니다.
- [언어(Language)] 대신 작업에 대한 [Python 버전(Python version)] 속성이 있습니다. 현재 AWS Glue Studio에서 생성된 Python 셸 작업은 Python 3.6을 사용합니다.
- [Glue 버전(Glue version)] 속성은 Python 셸 작업에 적용되지 않으므로 사용할 수 없습니다.
- [작업자 유형(Worker type)] 및 [작업자 수(Number of workers)] 대신 [데이터 처리 장치(Data processing units)] 속성이 표시됩니다. 이 작업 속성은 작업을 실행할 때 Python 셸에서 사용하는 DPU(데이터 처리 장치) 수를 결정합니다.
- [작업 북마크(Job bookmark)] 속성은 Python 셸 작업에 지원되지 않기 때문에 사용할 수 없습니다.
- [고급 속성(Advanced properties)]에서 Python 셸 작업에는 다음 속성을 사용할 수 없습니다.
 - 작업 지표
 - 연속 로깅
 - [Spark UI] 및 [Spark UI 로그 경로(Spark UI logs path)]
 - 머릿글 [라이브러리(Libraries)] 아래의 [종속 jar 경로(Dependent jars path)]

작업 다이어그램에서 노드의 상위 노드 변경

노드의 상위 항목을 변경하여 작업 다이어그램 내에서 노드를 이동하거나 노드의 데이터 원본을 변경할 수 있습니다.

상위 노드를 변경하려면

1. 수정할 작업 다이어그램의 노드를 선택합니다.
2. 노드 세부 정보 패널의 [노드 속성] 탭에 있는 [노드 상위 항목(Node parents)] 아래에서 노드의 현재 상위 항목을 제거합니다.
3. 목록에서 새 상위 노드를 선택합니다.
4. 새로 선택한 상위 노드와 일치하도록 필요에 따라 노드의 다른 속성을 수정합니다.

실수로 노드를 수정한 경우 도구 모음의 [실행 취소(Undo)] 버튼을 사용하여 작업을 되돌릴 수 있습니다.

작업 다이어그램에서 노드 삭제

Visual ETL 작업을 수행할 때 제거된 노드에 연결된 노드를 다시 추가하거나 재구성하지 않고도 캔버스에서 노드를 제거할 수 있습니다.

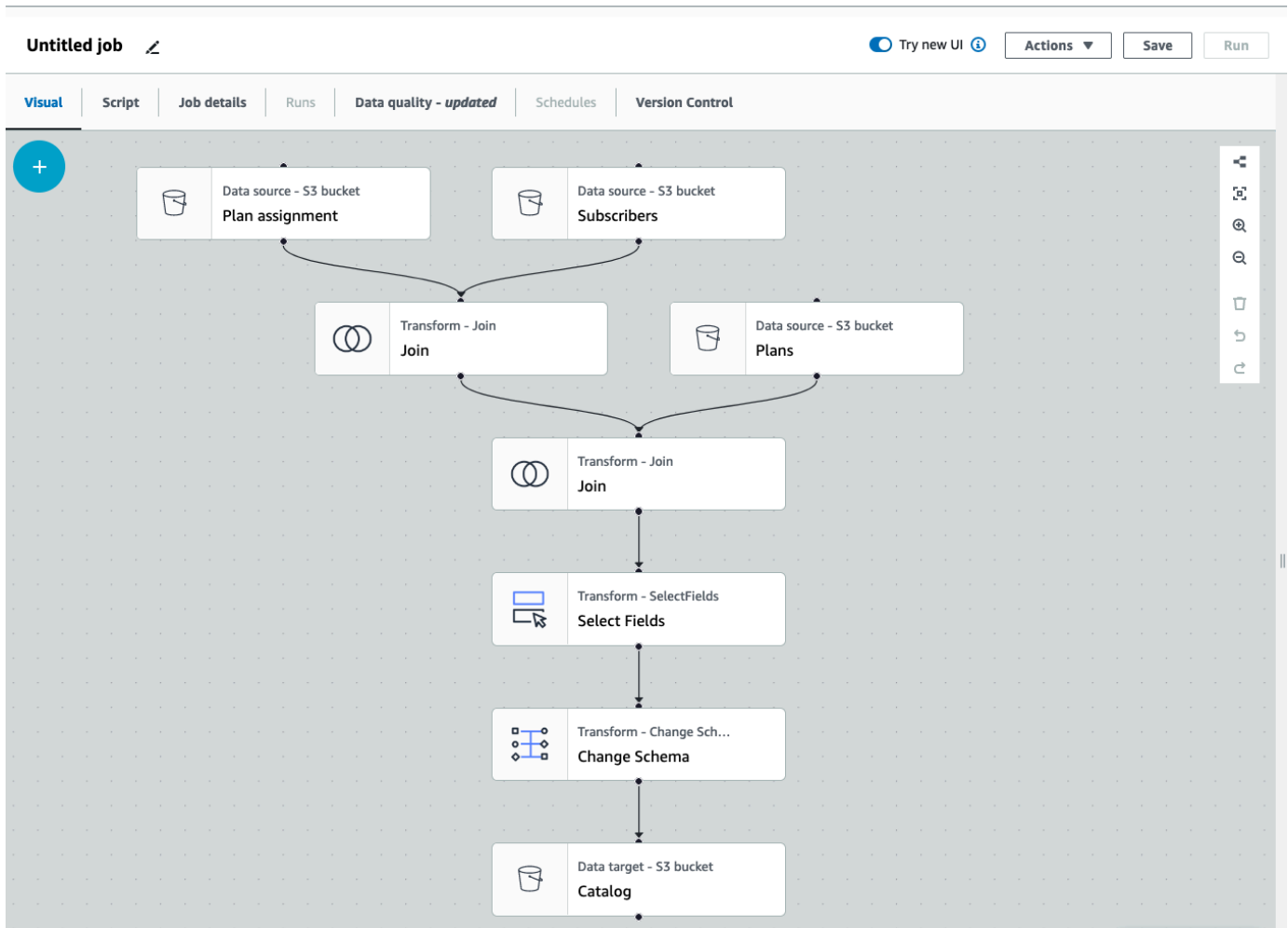
아래 예제에서는 ETL 작업 > Visual ETL을 선택한 다음, 예제 작업에서 여러 소스를 조인하는 Visual ETL 작업을 선택합니다. 예제 작업 생성을 선택하여 작업을 생성하고 아래 단계를 따르세요.

The screenshot shows the AWS Glue Studio interface. On the left is a navigation sidebar with 'Visual ETL' selected under 'Data Integration and ETL'. The main content area is titled 'AWS Glue Studio' and 'Jobs'. It features three main sections: 'Create job', 'Example jobs', and 'Your jobs (1)'. In the 'Create job' section, the 'Visual ETL' button is highlighted with a red box. In the 'Example jobs' section, the 'Visual ETL job to join multiple sources' job is highlighted with a red box. The 'Your jobs (1)' section shows a table with one job entry.

Job name	Type	Last modified	AWS Glue version
job_101521	Glue ETL	1/31/2022, 11:44:06 AM	2.0

캔버스에서 노드를 제거하려면

1. AWS Glue 콘솔의 탐색 메뉴에서 Visual ETL을 선택하고 기존 작업을 선택합니다. 작업 캔버스는 아래 그림과 같이 예제 작업을 표시합니다.



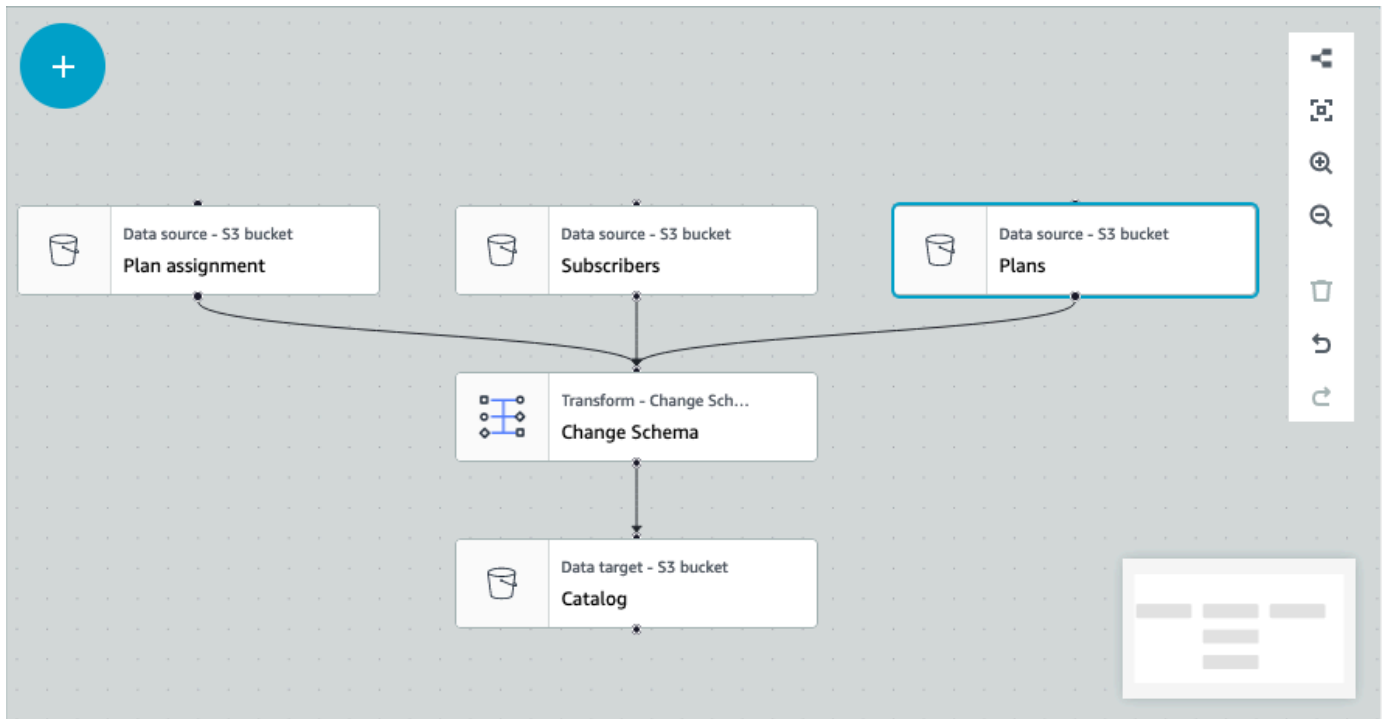
- 제거할 노드를 선택합니다. 캔버스가 노드로 확대됩니다. 캔버스 오른쪽에 있는 도구 모음에서 휴지통 아이콘을 선택합니다. 이렇게 하면 노드가 제거되고 노드에 연결된 모든 노드가 워크플로에서 해당 위치로 이동됩니다. 이 예제에서는 첫 번째 Join 노드가 캔버스에서 삭제되었습니다.

워크플로에서 노드를 삭제하면 AWS Glue에서 노드를 다시 정렬하여 워크플로가 유효하지 않은 방식으로 구성되도록 합니다. 여전히 노드 구성을 수정해야 할 수도 있습니다.

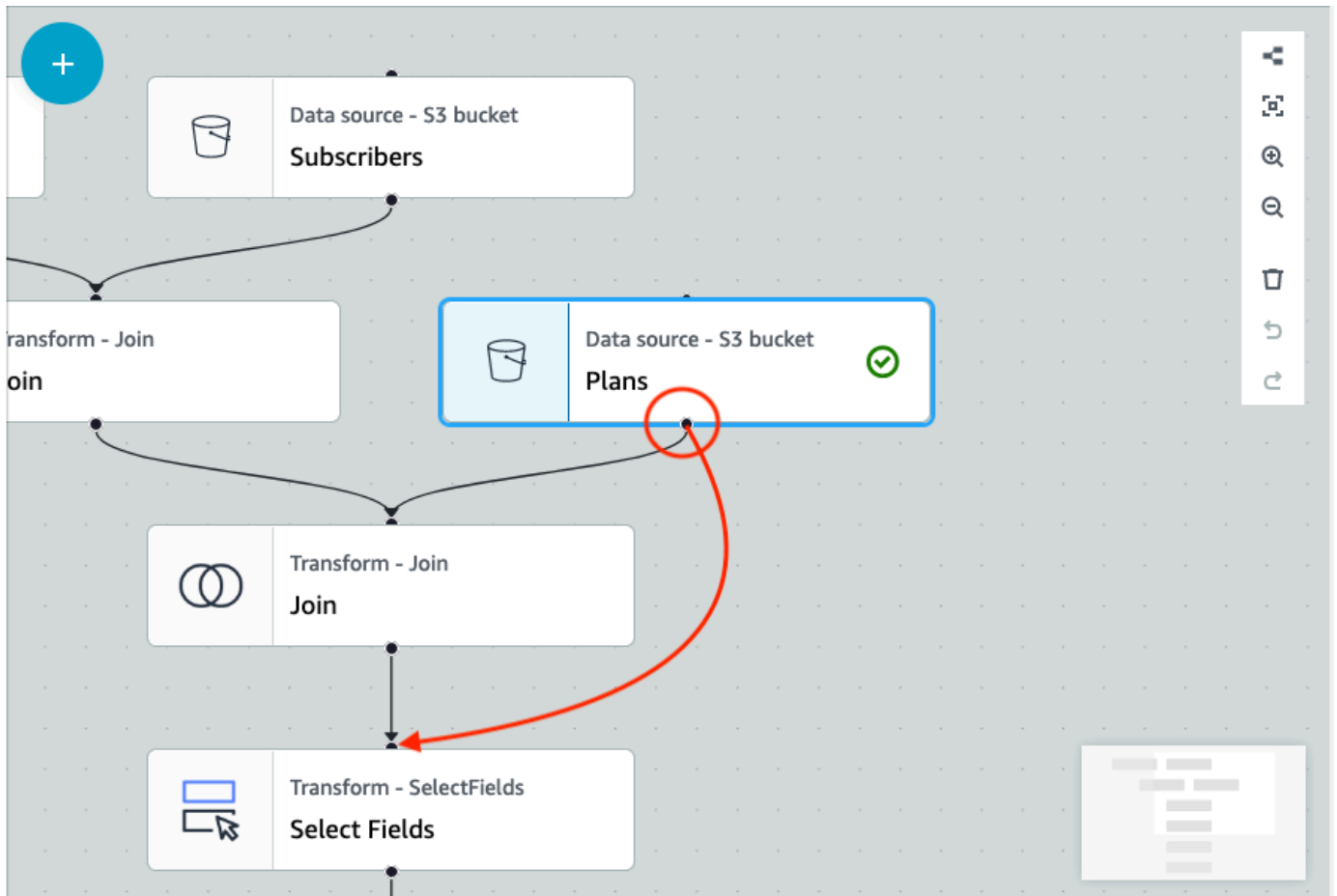
이 예제에서는 Subscribers 노드 아래의 Join 노드가 제거되었습니다. 따라서 Plans 소스 노드는 최상위 수준으로 이동되었으며 여전히 하위 Join 노드에 연결되어 있습니다. Join에는 선택한 테이블이 있는 두 개의 상위 소스 노드가 필요하므로 Join 노드에는 추가 구성이 필요합니다. 캔버스 오른쪽에 있는 변환 탭에는 조인 조건에서 누락된 요구 사항이 표시됩니다.

The screenshot shows the AWS Glue console interface for an 'Untitled job'. The workflow consists of several nodes: 'Data source - S3 bucket Plan assignment', 'Data source - S3 bucket Subscribers', 'Data source - S3 bucket Plans', 'Transform - Join', 'Transform - SelectFields Select Fields', and 'Transform - Change Sch... Change Schema'. The 'Join' node is highlighted with a red box. A yellow warning box at the bottom left states 'Node is misconfigured' and 'Data preview will be displayed when following node is correctly configured: Join'. The right-hand 'Transform' configuration panel shows the 'Join' node's settings, including 'Node parents' (Plan assignment, Subscribers, Plans), 'Join type' (Inner join), and 'Join conditions' (Insufficient source nodes). A red box highlights the 'Join conditions' section, which contains the message: 'The Join transform requires two parent source nodes with selected tables.'

3. 두 번째 Join 노드와 Select Fields 노드를 삭제합니다. 노드가 삭제되면 워크플로는 아래 예제와 같이 표시됩니다.



4. 노드 연결을 수정하려면 노드 핸들을 클릭하고 연결을 새 노드로 끌어 놓습니다. 이렇게 하면 노드를 삭제하고 논리적 흐름에서 노드를 다시 정렬할 수 있습니다. 예제에서는 빨간색 화살표로 표시된 대로 Plans 노드의 핸들을 클릭하고 연결을 Join 노드로 끌어서 새 연결을 생성합니다.



5. 작업을 취소해야 하는 경우 캔버스 오른쪽 도구 모음의 휴지통 아이콘 바로 아래에 있는 실행 취소 아이콘을 선택하세요.

AWS Glue 데이터 카탈로그 노드에 소스 및 대상 파라미터 추가

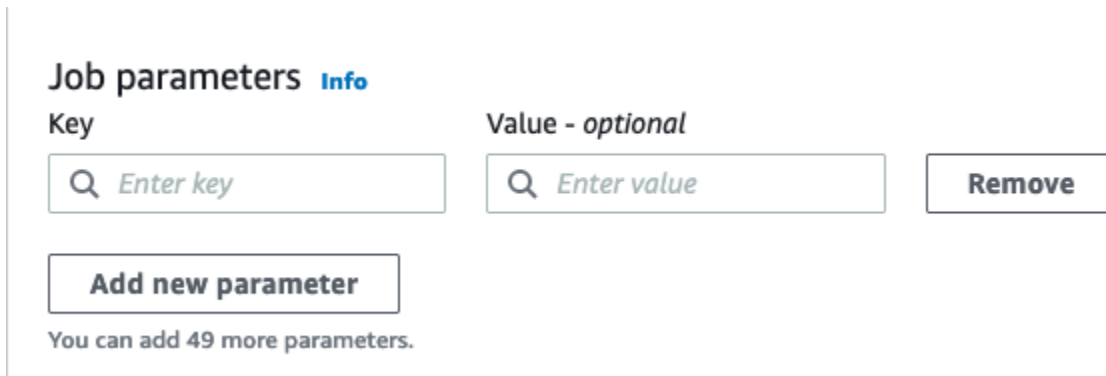
AWS Glue Studio은(는) 시각적 작업을 파라미터화할 수 있습니다. 프로덕션 및 개발 환경에서는 카탈로그 테이블 이름이 다를 수 있으므로 작업이 실행될 때 실행할 데이터베이스 및 테이블의 런타임 파라미터를 정의하고 선택할 수 있습니다.

작업 파라미터화를 사용하면 소스 및 대상을 파라미터화하고, AWS Glue 데이터 카탈로그 노드를 사용할 때 해당 파라미터를 작업에 저장할 수 있습니다. 소스 및 대상을 파라미터로 지정하면 특히 여러 환경에서 동일한 작업을 사용할 때 작업을 재사용할 수 있습니다. 소스 및 대상을 관리하는 데 드는 시간과 노력을 절약함으로써, 이는 배포 환경 전반에서 코드를 승격할 때 유용합니다. 또한 지정한 사용자 지정 파라미터는 AWS Glue 작업의 특정 실행에 대한 기본 인수를 재정의합니다.

소스 및 대상 파라미터 추가

AWS Glue 데이터 카탈로그 노드를 소스 또는 대상으로 사용하는지 관계없이 Job details(작업 세부 정보) 탭의 Advanced properties(고급 속성) 섹션에서 런타임 파라미터를 정의할 수 있습니다.

1. 소스 노드 또는 대상 노드로 AWS Glue 데이터 카탈로그를 선택합니다.
2. [작업 세부 정보(Job details)] 탭을 선택합니다.
3. Advanced properties(고급 속성)을 선택합니다.
4. 작업 파라미터 섹션에서 키 값을 입력합니다. 예를 들어, --db.source은(는) 데이터베이스 원본에 대한 파라미터가 될 것입니다. 키 이름 뒤에 'dash dash'가 오면 아무 이름이나 입력할 수 있습니다.



Job parameters [Info](#)

Key	Value - optional	
<input type="text" value="Q Enter key"/>	<input type="text" value="Q Enter value"/>	<input type="button" value="Remove"/>

You can add 49 more parameters.

5. 값을 입력합니다. 예를 들어, databasename은(는) 파라미터화되는 데이터베이스의 값이 될 것입니다.
6. 파라미터를 더 추가하고자 하는 경우 Add new parameter(새 파라미터 추가)를 선택합니다. 최대 50개의 파라미터를 추가할 수 있습니다. 키 값 쌍이 정의되면 AWS Glue 데이터 카탈로그 노드에서 파라미터를 사용할 수 있습니다.

런타임 파라미터 선택

Note

데이터베이스 및 테이블에 대한 런타임 파라미터를 선택하는 프로세스는 AWS Glue 데이터 카탈로그 노드가 소스 또는 대상인지와 상관없이 동일합니다.

1. 소스 노드 또는 대상 노드로 AWS Glue 데이터 카탈로그를 선택합니다.
2. Database(데이터베이스)의 Data source properties - Data Catalog(데이터 소스 속성 - 데이터 카탈로그) 탭에서 Use runtime parameters(런타임 파라미터 사용)을 선택합니다.

▼ **Use runtime parameters**

Select runtime parameter

Runtime parameters can be configured in the **Advanced properties** section on the **Job details** tab

Apply

Clear

3. 드롭다운 메뉴에서 파라미터를 선택합니다. 예를 들어, 소스 데이터베이스에 대해 정의한 파라미터를 선택하면 Apply(적용)을 선택하면 데이터베이스가 자동으로 데이터베이스 드롭다운 메뉴에 채워집니다.
4. 테이블(Table) 섹션에서 이미 소스 테이블로 정의한 파라미터를 선택합니다. Apply(적용)을 선택하면 테이블이 사용할 테이블로 자동으로 채워집니다.
5. 작업을 저장하고 실행하면 AWS Glue Studio이(가) 작업 실행 중에 선택한 파라미터를 참조합니다.

AWS Glue에서 Git 버전 제어 시스템 사용

i Note

노트북은 현재 AWS Glue Studio에서 버전 관리가 지원되지 않습니다. 하지만 AWS Glue 작업 스크립트 및 시각적 ETL 작업에 대한 버전 제어는 지원됩니다.

원격 리포지토리가 있고 리포지토리를 사용하여 AWS Glue 작업을 관리하려는 경우 AWS Glue Studio 또는 AWS CLI을(를) 사용하여 AWS Glue의 리포지토리 및 작업에 대한 변경 사항을 동기화할 수 있습니다. 이러한 방식으로 변경 내용을 동기화하면 작업을 AWS Glue Studio에서 리포지토리로 푸시하거나 리포지토리에서 AWS Glue Studio(으)로 가져옵니다.

AWS Glue Studio의 Git 통합을 통해 다음 작업을 수행할 수 있습니다.

- AWS CodeCommit, GitHub, GitLab 및 Bitbucket과 같은 Git 버전 관리 시스템과 통합
- 시각적 작업을 사용한 스크립트 작업을 사용한 AWS Glue Studio에서 AWS Glue 작업을 편집하고 이를 리포지토리에 동기화
- 작업의 소스 및 대상을 파라미터화

- 리포지토리에서 작업을 가져와 AWS Glue Studio에서 편집합니다.
- AWS Glue Studio의 다중 브랜치 워크플로를 활용하여 브랜치에서 가져오거나 브랜치로 푸시하여 작업 테스트
- 교차 계정 작업 생성에 대해 리포지토리에서 파일 다운로드 및 AWS Glue Studio(으)로 작업 업로드
- 선택한 자동화 도구 사용(예: Jenkins, AWS CodeDeploy, 등.)

이 비디오에서는 AWS Glue를 Git와 통합하고 협업에 기반한 지속적인 코드 파이프라인을 구축하는 방법을 보여줍니다.

IAM 권한

작업에 다음 IAM 권한 중 하나가 있는지 확인합니다. IAM 권한 설정 방법에 대한 자세한 내용을 알아보려면 [AWS Glue Studio에 대한 IAM 권한 설정](#)을 참조하세요.

- AWSGlueServiceRole
- AWSGlueConsoleFullAccess

Git 통합을 위해서는 최소한 다음과 같은 작업이 필요합니다.

- glue:UpdateJobFromSourceControl - 버전 관리 시스템에 있는 작업으로 AWS Glue을(를) 업데이트할 수 있습니다.
- glue:UpdateSourceControlFromJob - AWS Glue에 저장된 작업으로 버전 관리 시스템을 업데이트할 수 있습니다.
- s3:GetObject - 버전 관리 시스템으로 푸시하는 동안 작업에 대한 스크립트를 검색할 수 있습니다.
- s3:PutObject - 소스 제어 시스템에서 작업을 가져올 때 스크립트를 업데이트할 수 있습니다.

필수 조건

작업을 소스 제어 리포지토리로 푸시하려면 다음 사항이 필요합니다.

- 관리자가 이미 생성한 리포지토리
- 리포지토리 내 브랜치
- 개인용 액세스 토큰(Bitbucket의 경우 리포지토리 액세스 토큰)
- 리포지토리 소유자의 사용자 이름

- AWS Glue Studio의 리포지토리에 대해 읽기 및 쓰기가 허용되도록 리포지토리의 권한을 설정합니다
 - GitLab — 토큰 범위를 API, read_repository, write_repository로 설정
 - Bitbucket — 권한 설정 위치:
 - Workspace 멤버십 — 읽기, 쓰기
 - 프로젝트 — 쓰기, 관리자 읽기
 - 리포지토리 — 읽기, 쓰기, 관리, 삭제

Note

AWS CodeCommit 사용 시 개인 액세스 토큰 및 리포지토리 소유자는 필요하지 않습니다. [Git](#) 및 [AWS CodeCommit 시작하기](#)를 참조하세요.

AWS Glue Studio에서 소스 제어 리포지토리의 작업 사용

AWS Glue Studio에 없는 작업을 소스 제어 리포지토리에서 가져오고 AWS Glue Studio에서 작업을 사용하려고 하면 작업 유형에 따라 전제 조건이 달라집니다.

시각적 작업의 경우:

- 작업 이름과 일치하는 작업 정의의 폴더 및 JSON 파일이 필요합니다

예를 들어, 아래 작업 정의를 참조하세요. 리포지토리의 브랜치에는 폴더와 JSON 파일이 모두 작업 이름과 일치하는 경로 my-visual-job/my-visual-job.json이(가) 포함되어야 합니다

```
{
  "name" : "my-visual-job",
  "description" : "",
  "role" : "arn:aws:iam::aws_account_id:role/Rolename",
  "command" : {
    "name" : "glueetl",
    "scriptLocation" : "s3://foldername/scripts/my-visual-job.py",
    "pythonVersion" : "3"
  },
  "codegenConfigurationNodes" : "{\"node-nodeID\":{\"S3CsvSource\":{\"AdditionalOptions\":{\"EnableSamplePath\":false,\"SamplePath\":\"s3://notebook-test-input/netflix_titles.csv\"},\"Escaper\":\"\\\", \"Exclusions\":[],\"Name\":\"Amazon S3\", \"OptimizePerformance\":false,\"OutputSchemas\":[{\"Columns\":[{\"Name\":
```

```

{"show_id","\Type\":"string"},{"Name\":"type","\Type\":"string"},{"Name\":"title","\Type\":"choice"},{"Name\":"director","\Type\":"string"},{"Name\":"cast","\Type\":"string"},{"Name\":"country","\Type\":"string"},{"Name\":"date_added","\Type\":"string"},{"Name\":"release_year","\Type\":"bigint"},{"Name\":"rating","\Type\":"string"},{"Name\":"duration","\Type\":"string"},{"Name\":"listed_in","\Type\":"string"},{"Name\":"description","\Type\":"string"}]]],\Paths\":[\s3://dalamgir-notebook-test-input/netflix_titles.csv"],\QuoteChar\":"quote",\Recurse\":true,\Separator\":"comma",\WithHeader\":true}}}"
}

```

스크립트 작업의 경우:

- 폴더, 작업 정의의 JSON 파일 및 스크립트가 필요합니다.
- 폴더 및 JSON 파일은 작업 이름과 일치해야 합니다. 스크립트 이름은 파일 확장자와 함께 작업 정의의 `scriptLocation`와(과) 일치해야 합니다

예를 들어, 아래 작업 정의에서 리포지토리의 브랜치에는 경로 `my-script-job/my-script-job.json` 및 `my-script-job/my-script-job.py`이(가) 포함되어야 합니다. 스크립트 이름은 스크립트의 확장자를 포함하여 `scriptLocation`의 이름과 일치해야 합니다.

```

{
  "name" : "my-script-job",
  "description" : "",
  "role" : "arn:aws:iam::aws_account_id:role/Rolename",
  "command" : {
    "name" : "glueetl",
    "scriptLocation" : "s3://foldername/scripts/my-script-job.py",
    "pythonVersion" : "3"
  }
}

```

제한 사항

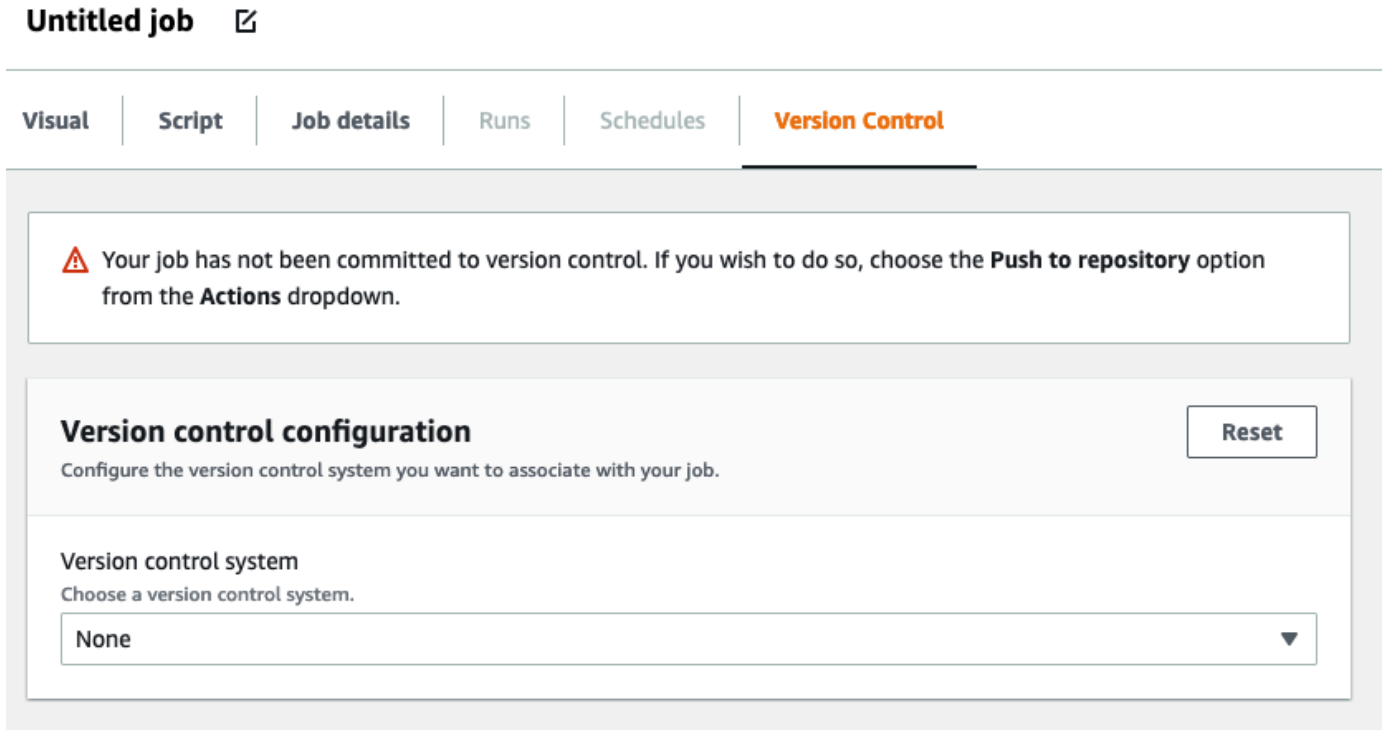
- AWS Glue는 현재 [GitLab-Groups](#)에서 푸시/풀링을 지원하지 않습니다.

버전 관리 리포지토리를 AWS Glue와(과) 연결

AWS Glue Studio 작업 편집기의 Version Control(버전 관리) 탭에서 버전 관리 리포지토리 세부 정보를 입력하고 관리할 수 있습니다. Git 리포지토리와 통합하려면 AWS Glue Studio에 로그인할 때마다 리포지토리에 연결해야 합니다.

Git 버전 관리 시스템 연결하기:

1. AWS Glue Studio에서 새 작업을 시작하고 Version Control(버전 관리) 탭을 선택합니다.



2. 버전 관리 시스템에서 다음 드롭다운 메뉴를 클릭하여 사용 가능한 옵션 중에서 Git Service를 선택합니다.

- AWS CodeCommit
- GitHub
- GitLab
- Bitbucket

3. 선택한 Git 버전 관리 시스템에 따라 완료해야 하는 필드가 달라집니다.

AWS CodeCommit의 경우

작업에 사용할 리포지토리와 브랜치를 선택하여 리포지토리 구성을 완료합니다.

- 리포지토리 - AWS CodeCommit에 리포지토리를 설정한 경우 드롭다운 메뉴에서 리포지토리를 선택합니다. 리포지토리가 목록에 자동으로 채워집니다.
- 브랜치 - 드롭다운 메뉴에서 브랜치를 선택합니다.
- 폴더 - 선택 사항- 작업을 저장할 폴더 이름을 입력합니다. 비워 두면 폴더가 자동으로 생성됩니다. 폴더 이름은 기본적으로 작업 이름입니다.

GitHub의 경우:

다음 필드를 작성하여 GitHub 구성을 완료합니다.

- Personal access token(개인 액세스 토큰) - GitHub 리포지토리에서 제공하는 토큰입니다. 개인 액세스 토큰에 대한 자세한 내용을 알아보려면 [GitHub 문서](#)를 참조하세요.
- 리포지토리 소유자 - GitHub 리포지토리의 소유자입니다.

GitHub에서 사용할 리포지토리와 브랜치를 선택하여 리포지토리 구성을 완료합니다.

- 리포지토리 - GitHub에 리포지토리를 설정한 경우 드롭다운 메뉴에서 리포지토리를 선택합니다. 리포지토리가 목록에 자동으로 채워집니다.
- 브랜치 - 드롭다운 메뉴에서 브랜치를 선택합니다.
- 폴더 - 선택 사항- 작업을 저장할 폴더 이름을 입력합니다. 비워 두면 폴더가 자동으로 생성됩니다. 폴더 이름은 기본적으로 작업 이름입니다.

GitLab의 경우:

Note

AWS Glue는 현재 [GitLab-Groups](#)에서 푸시/풀링을 지원하지 않습니다.

- 개인 액세스 토큰 - GitLab 리포지토리에서 제공하는 토큰입니다. 개인 액세스 토큰에 대한 자세한 내용은 [GitLab 개인 액세스 토큰](#)을 참조하세요
- 리포지토리 소유자 - GitLab 리포지토리의 소유자입니다.

GitLab에서 사용할 리포지토리와 브랜치를 선택하여 리포지토리 구성을 완료합니다.

- 리포지토리 - GitLab에 리포지토리를 설정한 경우 드롭다운 메뉴에서 리포지토리를 선택합니다. 리포지토리가 목록에 자동으로 채워집니다.
- 브랜치 - 드롭다운 메뉴에서 브랜치를 선택합니다.
- 폴더 - 선택 사항- 작업을 저장할 폴더 이름을 입력합니다. 비워 두면 폴더가 자동으로 생성됩니다. 폴더 이름은 기본적으로 작업 이름입니다.

Bitbucket의 경우:

- 앱 암호 - Bitbucket은 리포지토리 액세스 토큰이 아닌 앱 암호를 사용합니다. 앱 암호에 대한 자세한 내용은 [앱 암호](#)를 참조하세요.
- 리포지토리 소유자 - Bitbucket 리포지토리의 소유자입니다. Bitbucket에서 소유자는 리포지토리의 생성자입니다.

Bitbucket에서 사용할 워크스페이스, 리포지토리, 브랜치 및 폴더를 선택하여 리포지토리 구성을 완료합니다.

- 작업 영역 — Bitbucket에 작업 영역을 설정한 경우 드롭다운 메뉴에서 작업 영역을 선택합니다. 작업 공간이 자동으로 채워집니다
- 리포지토리- Bitbucket에 리포지토리를 설정한 경우 드롭다운 메뉴에서 리포지토리를 선택합니다. 리포지토리가 자동으로 채워집니다
- 브랜치 — 드롭다운 메뉴에서 브랜치를 선택합니다. 브랜치가 자동으로 채워집니다
- 폴더 - 선택 사항- 작업을 저장할 폴더 이름을 입력합니다. 비워 두면 작업 이름의 폴더가 자동으로 생성됩니다.

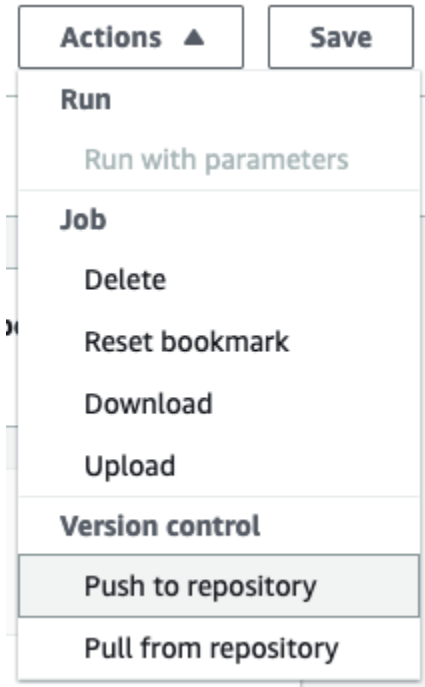
4. AWS Glue Studio 작업의 페이지 상단에서 Save(저장)을 선택합니다.

소스 리포지토리에 AWS Glue 작업 푸시

버전 관리 시스템의 세부 정보를 입력한 후에는 AWS Glue Studio에서 작업을 편집하고 작업을 소스 리포지토리로 푸시할 수 있습니다. 푸시 및 가져오기 같은 Git 개념에 익숙하지 않은 경우 [Git 및 AWS CodeCommit 시작하기](#)에 대한 이 자습서를 참조하세요.

작업을 리포지토리에 푸시하려면 버전 관리 시스템의 세부 정보를 입력하고 작업을 저장해야 합니다.

1. AWS Glue Studio 작업에서 Actions(작업)을 선택합니다. 그러면 추가 메뉴 옵션이 열립니다.



2. Push to repository(리포지토리로 푸시)를 선택합니다.

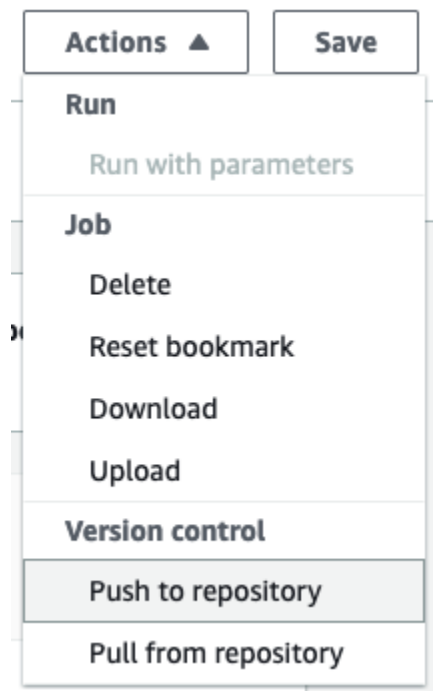
이 작업을 수행하면 작업이 저장됩니다. 리포지토리로 푸시하면 AWS Glue Studio이(가) 마지막으로 저장한 변경 내용을 푸시합니다. 리포지토리의 작업이 사용자 본인 또는 다른 사용자에게 의해 수정되었으며 AWS Glue Studio의 작업과 동기화되지 않은 경우 AWS Glue Studio에서 작업을 푸시할 때 AWS Glue Studio에서 저장된 작업으로 리포지토리의 작업이 덮어써 집니다.

3. Confirm(확인)을 선택하여 작업을 완료합니다. 이렇게 하면 리포지토리에서 새 커밋을 생성합니다. AWS CodeCommit을(를) 사용하는 경우 확인 메시지에 AWS CodeCommit에 대한 최신 커밋에 대한 링크가 표시됩니다.

소스 리포지토리에서 AWS Glue 작업 가져오기

Version control(버전 관리) 탭에 Git 리포지토리의 세부 정보를 입력한 후에는 리포지토리에서 작업을 가져와 AWS Glue Studio에서 편집할 수도 있습니다.

1. AWS Glue Studio 작업에서 Actions(작업)을 선택합니다. 그러면 추가 메뉴 옵션이 열립니다.



2. Pull from repository(리포지토리에서 가져오기)를 선택합니다.
3. 확인을 선택합니다. 이렇게 하면 리포지토리에서 최신 커밋을 가져와 AWS Glue Studio에서 작업을 업데이트합니다.
4. AWS Glue Studio에서 작업을 편집합니다. 변경할 경우 Actions(작업) 드롭다운 메뉴에서 Push to repository(리포지토리로 푸시)를 선택하여 작업을 리포지토리에 동기화할 수 있습니다.

AWS Glue Studio 노트북을 사용한 코드 작성

데이터 엔지니어는 AWS Glue Studio의 대화형 노트북 인터페이스 또는 AWS Glue의 대화형 세션을 사용하기 전보다 더 쉽고 빠르게 AWS Glue 작업을 작성할 수 있습니다.

주제

- [노트북 사용 개요](#)
- [AWS Glue Studio에서 노트북을 사용하여 ETL 작업 생성](#)
- [노트북 편집기 구성 요소](#)
- [노트북 및 작업 스크립트 저장](#)
- [노트북 세션 관리](#)
- [CodeWhisperer와 함께 사용 AWS Glue Studio notebooks](#)

노트북 사용 개요

AWS Glue Studio를 사용하면 Jupyter Notebook 기반의 노트북 인터페이스에서 작업을 대화형으로 작성할 수 있습니다. AWS Glue Studio의 노트북을 통해 작업 스크립트를 편집한 후 전체 작업을 실행하지 않고도 출력을 볼 수 있으며, 데이터 통합 코드를 편집한 후 전체 작업을 실행하지 않고도 출력을 볼 수 있으며, 마크다운을 추가하고 노트북을 .ipynb 파일과 작업 스크립트로 저장할 수 있습니다. 소프트웨어를 로컬로 설치하거나 서버를 관리하지 않고도 노트북을 시작할 수 있습니다. 코드가 만족스러우면 AWS Glue Studio에서 하나의 버튼을 클릭하여 노트북을 Glue 작업으로 변환할 수 있습니다.

노트북을 사용하는 경우의 몇 가지 이점은 다음과 같습니다.

- 프로비저닝하거나 관리할 클러스터가 없습니다.
- 요금을 결제할 유휴 클러스터가 없습니다.
- 사전 구성이 필요하지 않습니다.
- Jupyter Notebook을 설치할 필요가 없습니다.
- AWS Glue ETL과 동일한 런타임/플랫폼입니다.

AWS Glue Studio를 통해 노트북을 시작하는 경우 데이터를 탐색하고 몇 초 만에 작업 스크립트 개발을 시작할 수 있도록 모든 구성 단계가 완료됩니다. AWS Glue Studio는 AWS Glue Jupyter 커널을 사용하여 Jupyter Notebook을 구성합니다. 이 노트북을 사용하기 위해 VPC, 네트워크 연결 또는 개발 엔드포인트를 구성할 필요는 없습니다.

노트북 인터페이스를 사용하여 작업을 생성하려면

- 필요한 IAM 권한을 구성합니다.
- 노트북 세션을 시작하여 작업을 생성합니다.
- 노트북의 셀에 코드를 작성합니다.
- 코드를 실행하고 테스트하여 출력을 봅니다.
- 작업 저장

저장한 후의 노트북은 전체 AWS Glue 작업입니다. 작업 실행 예약, 작업 파라미터 설정, 노트북과 작업 실행 기록 나란히 보기 등 작업의 모든 측면을 관리할 수 있습니다.

AWS Glue Studio에서 노트북을 사용하여 ETL 작업 생성

AWS Glue Studio 콘솔에서 노트북 사용을 시작하려면

1. AWS Identity and Access Management 정책을 AWS Glue Studio 사용자에게 연결하고 ETL 작업 및 노트북에 대한 IAM 역할을 생성합니다.
2. [IAM 역할에 권한 부여](#)에 설명된 대로 노트북에 대한 추가 IAM 보안을 구성합니다.
3. <https://console.aws.amazon.com/gluestudio/>에서 AWS Glue Studio 콘솔을 엽니다.

Note

브라우저가 서드 파티 쿠키를 차단하지 않는지 확인합니다. 브라우저가 기본값 또는 사용자가 지정한 설정으로 서드 파티 쿠키를 차단하는 경우 노트북이 실행되지 않습니다. 쿠키 관리에 관한 자세한 내용은 다음을 참조하세요.

- [Chrome](#)
- [Firefox](#)
- [Safari](#)

4. 왼쪽 탐색 메뉴에서 작업(Jobs) 링크를 선택합니다.
5. Jupyter Notebook(Jupyter notebook)을 선택한 다음 생성(Create)을 선택하여 새 노트북 세션을 시작합니다.
6. Jupyter Notebook에서 작업 생성 페이지에서 작업 이름을 제공하고 사용할 IAM 역할을 선택합니다. 작업 생성(Create job)을 선택합니다.

잠시 후에 노트북 편집기가 나타납니다.

7. 코드를 추가한 후 셀을 실행하여 세션을 시작해야 합니다. 다음과 같은 여러 가지 방법으로 셀을 실행할 수 있습니다.
 - 재생 버튼을 누릅니다.
 - 키보드 단축키를 사용합니다.
 - MacOS에서 Command + Enter를 눌러 셀을 실행합니다.
 - Windows에서 Shift + Enter를 눌러 셀을 실행합니다.

Jupyter Notebook 인터페이스를 사용하여 코드를 작성하는 방법에 대한 자세한 내용은 [Jupyter Notebook 사용 설명서](#)를 참조하세요.

8. 스크립트를 테스트하려면 전체 스크립트 또는 개별 셀을 실행합니다. 모든 명령 출력은 셀 아래 영역에 표시됩니다.
9. 노트북 개발을 마친 후 작업을 저장하고 실행할 수 있습니다. 스크립트(Script) 탭에서 스크립트를 찾을 수 있습니다. 노트북에 추가한 모든 매직은 제거되고 생성된 AWS Glue 작업의 스크립트의 일부로 저장되지 않습니다. AWS Glue Studio는 노트북 내용에서 생성된 스크립트의 끝에 `job.commit()`을 자동으로 추가합니다.

작업 실행 방법에 대한 자세한 내용은 [작업 실행 시작](#) 섹션을 참조하세요.

노트북 편집기 구성 요소

노트북 편집기 인터페이스에는 다음과 같은 기본 섹션이 있습니다.

- 노트북 인터페이스(기본 패널) 및 도구 모음
- 작업 편집 탭

노트북 편집기

AWS Glue Studio 노트북 편집기는 Jupyter Notebook 애플리케이션을 기반으로 합니다. AWS Glue Studio 노트북 인터페이스는 [노트북 사용자 인터페이스](#) 섹션에 설명된, Jupyter Notebook에서 제공하는 인터페이스와 비슷합니다. 대화형 세션에서 사용하는 노트북은 Jupyter Notebook입니다.

AWS Glue Studio 노트북은 Jupyter 노트북과 비슷하지만 다음과 같은 몇 가지 주요 차이점이 있습니다.

- 현재 AWS Glue Studio 노트북은 확장을 설치할 수 없습니다.
- 여러 탭을 사용할 수 없습니다. 작업과 노트북 간에 1:1 관계입니다.
- AWS Glue Studio 노트북에는 Jupyter Notebook에 있는 것과 같은 상단 파일 메뉴가 없습니다.
- 현재 AWS Glue Studio 노트북은 AWS Glue 커널에서만 실행됩니다. 커널은 직접 업데이트할 수 없습니다.

AWS Glue Studio 작업 편집 탭

ETL 작업을 조작하는 데 사용하는 탭은 노트북 페이지 상단에 있습니다. AWS Glue Studio의 시각적 작업 편집기에 표시되는 탭과 비슷하며, 동일한 작업을 수행합니다.

- 노트북(Notebook) - 노트북 인터페이스를 사용하여 작업 스크립트를 보려면 이 탭을 사용합니다.
- 작업 세부 정보(Job details) - 작업 실행의 환경과 속성을 구성합니다.
- 실행(Runs) - 이 작업의 이전 실행 정보를 봅니다.
- 일정(Schedules) - 특정 시간에 작업 실행 일정을 구성합니다.

노트북 및 작업 스크립트 저장

생성 중인 작업 스크립트와 노트북을 언제든지 저장할 수 있습니다. 시각적 편집기나 스크립트 편집기를 사용하는 경우와 마찬가지로, 오른쪽 위에 있는 저장(Save) 버튼을 선택하면 됩니다.

저장을 선택하면 노트북 파일이 기본 위치에 저장됩니다.

- 기본적으로 작업 스크립트는 작업 세부 정보 탭의 고급 속성 아래 작업 세부 정보 속성 스크립트 경로에 표시된 Amazon S3 위치에 저장됩니다. 작업 스크립트는 Scripts라는 하위 폴더에 저장됩니다.
- 기본적으로 노트북 파일(.ipynb)은 작업 세부 정보 탭의 고급 속성 아래 작업 세부 정보 스크립트 경로에 표시된 Amazon S3 위치에 저장됩니다. 노트북 파일은 Notebooks라는 하위 폴더에 저장됩니다.

Note

작업을 저장하면 작업 스크립트에는 노트북의 코드 셀만 포함됩니다. 마크다운 셀 및 매직은 작업 스크립트에 포함되지 않습니다. 그러나 .ipynb 파일에는 마크다운과 매직이 포함됩니다.

작업을 저장한 후 노트북에서 생성한 스크립트를 사용하여 작업을 실행할 수 있습니다.

노트북 세션 관리

AWS Glue Studio의 노트북은 AWS Glue의 대화형 세션 기능을 기반으로 합니다. 대화형 세션을 사용하면 비용이 발생합니다. 비용을 관리하기 위해 계정에 대해 생성된 세션을 모니터링하고 모든 세션의 기본 설정을 구성할 수 있습니다.

모든 노트북 세션의 기본 시간 제한 변경

기본적으로 프로비저닝된 AWS Glue Studio 노트북은 노트북이 시작되고 셀이 실행되지 않은 경우 12시간 후에 시간 초과됩니다. 관련된 비용이 없으며 제한 시간을 구성할 수 없습니다.

셀을 실행하면 대화형 세션이 시작됩니다. 이 세션의 기본 제한 시간은 48시간입니다. 셀을 실행하기 전에 `%idle_timeout` 매직을 전달하여 이 제한 시간을 구성할 수 있습니다.

AWS Glue Studio에서 노트북의 기본 세션 시간 제한을 수정하려면

1. 노트북의 셀에 `%idle_timeout` 매직을 입력하고 시간 제한 값을 분 단위로 지정합니다.
2. 예: `%idle_timeout 15`는 기본 시간 제한을 15분으로 변경합니다. 15분 내에 세션을 사용하지 않으면 세션이 자동으로 중지됩니다.

추가 Python 모듈 설치

`pip`를 사용하여 세션에 추가 모듈을 설치하려는 경우 `%additional_python_modules`를 사용하여 세션에 추가하면 됩니다.

```
%additional_python_modules awswrangler, s3://mybucket/mymodule.whl
```

`additional_python_modules`에 대한 모든 인수가 `pip3 install -m <>`에 전달됩니다.

사용 가능한 Python 모듈 목록은 [AWS Glue에서 Python 라이브러리 사용](#)을 참조하세요.

AWS Glue 구성 변경

매직을 사용하여 AWS Glue 작업 구성 값을 제어할 수 있습니다. 작업 구성 값을 변경하려면 노트북에서 적절한 매직을 사용해야 합니다. [Jupyter용 AWS Glue 대화형 세션에서 지원되는 매직](#)을 참조하세요.

Note

실행 중인 세션의 속성 재정의는 더 이상 사용할 수 없습니다. 세션 구성을 변경하려면 세션을 중지하고 새 구성을 설정한 후 새 세션을 시작하면 됩니다.

AWS Glue는 다양한 작업자 유형을 지원합니다. `%worker_type`을 사용하여 작업자 유형을 설정할 수 있습니다. 예: `%worker_type G.2X`. 기본값은 `G.1X`입니다.

`%number_of_workers`를 사용하여 작업자 수도 지정할 수 있습니다. 예를 들어 40개의 작업자를 지정하려면 `%number_of_workers 40`을 사용합니다.

자세한 내용은 [작업 속성 수정](#)을 참조하세요.

노트북 세션 중지

노트북 세션을 중지하려면 `%stop_session` 매직을 사용합니다.

AWS 콘솔에서 노트북을 나가는 경우 경고 메시지가 표시되며, 여기서 세션 중지를 선택할 수 있습니다.

CodeWhisperer 와 함께 사용 AWS Glue Studio notebooks

AWS Glue Studio를 사용하면 Jupyter Notebook 기반의 노트북 인터페이스에서 작업을 대화형으로 작성할 수 있습니다. 를 사용하면 CodeWhisperer 노트북 내 작성 환경이 개선됩니다. AWS Glue Studio

Amazon CodeWhisperer 확장 프로그램은 코드 권장 사항을 생성하고 코드 문제와 관련된 개선 사항을 제안하여 코드 작성을 지원합니다.

아마존이란 CodeWhisperer 무엇입니까?

CodeWhisperer Amazon은 개발자의 생산성을 향상시키는 데 도움이 되는 기계 학습 기반 서비스입니다. CodeWhisperer 개발자의 자연 언어 설명과 IDE의 코드를 기반으로 코드 권장 사항을 생성하여 이를 달성합니다. 평가판 기간 동안 CodeWhisperer Amazon은 Java, Python JavaScript, C# 및 TypeScript 프로그래밍 언어로 제공됩니다. 이 서비스는 Amazon SageMaker Studio JupyterLab, Amazon SageMaker 노트북 인스턴스 및 기타 통합 개발 환경 (IDE) 과 통합됩니다.

자세한 내용은 [설정을 CodeWhisperer](#) 참조하십시오. AWS Glue Studio

콘솔의 AWS Glue 작업 실행 상태

AWS Glue 추출, 변환, 로드 작업이 실행되는 동안 또는 중지된 후 작업 상태를 볼 수 있습니다. AWS Glue 콘솔을 사용하여 상태를 볼 수 있습니다. 작업 실행 상태에 대한 자세한 내용은 [the section called “작업 실행 상태”](#) 섹션을 참조하십시오.

작업 모니터링 대시보드에 액세스

AWS Glue 탐색 창에서 모니터링(Monitoring) 링크를 선택하여 작업 모니터링 대시보드에 액세스합니다.

작업 모니터링 대시보드 개요

작업 모니터링 대시보드는 [실행 중(Running)], [취소됨(Canceled)], [성공(Success)] 또는 [실패(Failed)] 상태의 작업에 대한 집계와 함께 작업 실행에 대한 전체 요약を提供합니다. 추가 타일은 전체 작업 실행 성공률, 작업에 대한 예상 DPU 사용량, 작업 유형, 작업자 유형 및 개별 작업 상태 개수 분석을 제공합니다.

타일의 그래프는 대화형입니다. 그래프에서 블록을 선택하여 페이지 하단의 [작업 실행(Job runs)] 테이블에 해당 작업만 표시하는 필터를 실행할 수 있습니다.

[날짜 범위(Date range)] 선택기를 사용하여 이 페이지에 표시되는 정보의 날짜 범위를 변경할 수 있습니다. 날짜 범위를 변경하면 정보 타일이 조정되어 현재 날짜 이전의 지정된 일 수에 대한 값을 표시합니다. 날짜 범위 선택기에서 [사용자 정의(Custom)]를 선택하면 특정 날짜 범위를 사용할 수도 있습니다.

작업 실행 보기

Note

워크플로 및 작업 실행의 경우 작업 실행 기록을 90일 동안 액세스할 수 있습니다.

[작업 실행(Job runs)] 리소스 목록에는 지정된 날짜 범위 및 필터에 대한 작업이 표시됩니다.

상태, 작업자 유형, 작업 유형 및 작업 이름과 같은 추가 기준에 따라 작업을 필터링할 수 있습니다. 테이블 상단의 필터 상자에 필터로 사용할 텍스트를 입력할 수 있습니다. 텍스트를 입력할 때 일치하는 텍스트가 포함된 행으로 테이블 결과가 업데이트됩니다.

작업 모니터링 대시보드의 그래프에서 요소를 선택하여 작업의 하위 집합을 볼 수 있습니다. 예를 들어 [작업 실행 요약(Job runs summary)] 타일에서 실행 중인 작업 수를 선택하면 [작업 실행(Job runs)] 목록에 현재 상태가 Running인 작업만 표시됩니다. [작업자 유형 분석(Worker type breakdown)] 막대 차트에서 막대 중 하나를 선택하면 작업자 유형 및 상태가 일치하는 작업 실행만 [작업 실행(Job runs)] 목록에 표시됩니다.

[작업 실행(Job runs)] 리소스 목록에는 작업 실행에 대한 세부 정보가 표시됩니다. 열 머리글을 선택하여 테이블의 행을 정렬할 수 있습니다. 표에는 다음 정보가 포함되어 있습니다.

속성	설명
작업 이름	작업의 이름입니다.
유형	<p>작업 환경의 유형입니다.</p> <ul style="list-style-type: none"> [Glue ETL]: AWS Glue에서 관리하는 Apache Spark 환경에서 실행됩니다. [Glue 스트리밍(Glue Streaming)]: Apache Spark 환경에서 실행되고 데이터 스트림에서 ETL을 수행합니다. Python 셸: Python 스크립트를 셸로 실행합니다.
시작 시간	이 작업이 시작된 날짜 및 시간.
종료 시간	이 작업 실행이 완료된 날짜 및 시간입니다.
실행 상태	<p>작업 실행의 현재 상태입니다. 값은 다음과 같습니다.</p> <ul style="list-style-type: none"> STARTING RUNNING STOPPING STOPPED SUCCEEDED FAILED TIMEOUT

속성	설명
런타임	작업이 리소스를 소비한 시간입니다.
Capacity	이 작업 실행에 할당된 AWS Glue 데이터 처리 장치(DPU) 수입니다. 용량 계획에 대한 자세한 내용은 AWS Glue Developer Guide의 Monitoring for DPU Capacity Planning 을 참조하세요.

속성	설명
작업자 유형	<p>작업 실행 시 할당된 미리 정의된 작업자 유형입니다. 값은 G.1X, G.2X, G.4X 또는 G.8X일 수 있습니다.</p> <ul style="list-style-type: none"> G.1X – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 1개의 DPU(vCPU 4개, 메모리 16GB)에 매핑됩니다. 메모리 집약적인 작업의 경우 이 작업자 유형을 사용하는 것이 좋습니다. AWS Glue 버전 2.0 이상 작업의 기본 [작업자 유형(Worker type)]입니다. G.2X – 이 유형을 선택할 경우 [작업자 수(Number of workers)] 값도 제공합니다. 각 작업자는 128GB 디스크(약 77GB의 여유 공간)에서 2개의 DPU(vCPU 8개, 메모리 32GB)에 매핑됩니다. 메모리 집약적인 작업과 기계 학습 변환을 실행하는 작업의 경우 이 작업자 유형을 사용하는 것이 좋습니다. G.4X – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 256GB 디스크(약 235GB의 여유 공간)에서 4개의 DPU(vCPU 16개, 메모리 64GB)에 매핑됩니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 미국 동부(오하이오), 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄), 캐나다(중부), 유럽(프랑크푸르트), 유럽(아일랜드), 유럽(스톡홀름)과 같은 AWS 리전에서 AWS Glue 버전 3.0 이상 Spark ETL 작업에 대해서만 사용할 수 있습니다.

속성	설명
DPU 시간	<p>• G.8X – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 512GB 디스크(약 487GB의 여유 공간)에서 8개의 DPU(vCPU 32개, 메모리 128GB)에 매핑됩니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 G.4X 작업자 유형에 지원되는 동일한 AWS 리전에서 AWS Glue 버전 3.0 이상 Spark ETL 작업에 대해서만 사용할 수 있습니다.</p> <p>작업 실행에 사용된 예상 DPU입니다. DPU는 처리 능력의 상대적인 측정치입니다. DPU는 작업 실행 비용을 결정하는 데 사용됩니다. 자세한 내용은 AWS Glue 요금 페이지를 참조하십시오.</p>

목록에서 원하는 작업 실행을 선택하고 추가 정보를 볼 수 있습니다. 작업 실행을 선택하고 다음 중 하나를 수행합니다.

- [작업(Actions)] 메뉴와 [작업 보기(View job)] 옵션을 선택하여 시각적 편집기에서 작업을 봅니다.
- [작업(Actions)] 메뉴와 [실행 중지(Stop run)] 옵션을 선택하여 작업의 현재 실행을 중지합니다.
- [CloudWatch Logs 보기(View CloudWatch logs)] 버튼을 선택하여 해당 작업에 대한 작업 실행 로그를 봅니다.
- 세부 정보 보기를 선택하여 작업 실행 세부 정보 페이지를 봅니다.

작업 실행 로그 보기

다양한 방법으로 작업 로그를 볼 수 있습니다.

- [모니터링(Monitoring)] 페이지의 [작업 실행(Job runs)] 테이블에서 작업 실행을 선택한 다음 [CloudWatch Logs 보기(View CloudWatch logs)]를 선택합니다.
- 시각적 작업 편집기의 작업에 대한 [실행(Runs)] 탭에서 하이퍼링크를 선택하여 로그를 봅니다.

- [로그(Logs)] - 작업 실행에 연속 로깅이 사용될 때 작성된 Apache Spark 작업 로그에 대한 링크입니다. 이 링크를 선택하면 /aws-glue/jobs/logs-v2 로그 그룹의 Amazon CloudWatch 로그로 이동합니다. 기본적으로 로그는 유용하지 않은 Apache Hadoop YARN 하트비트 및 Apache Spark 드라이버 또는 실행기 로그 메시지를 제외합니다. 연속 로깅에 대한 자세한 내용은 AWS Glue Developer Guide의 [Continuous Logging for AWS Glue Jobs](#)를 참조하세요.
- [오류 로그(Error logs)] - 이 작업을 실행할 때 stderr에 작성되는 로그와 연결됩니다. 이 링크를 선택하면 /aws-glue/jobs/error 로그 그룹의 Amazon CloudWatch 로그로 이동합니다. 이 로그를 사용하여 작업 실행 중에 발생한 오류에 대한 세부 정보를 볼 수 있습니다.
- [출력 로그(Output logs)] - 이 작업을 실행할 때 stdout에 작성되는 로그와 연결됩니다. 이 링크를 선택하면 /aws-glue/jobs/output 로그 그룹의 Amazon CloudWatch 로그로 이동합니다. 이러한 로그를 사용하여 AWS Glue Data Catalog에서 생성된 테이블 및 발생한 오류에 대한 모든 세부 정보를 볼 수 있습니다.

작업 실행의 세부 정보 보기

[모니터링(Monitoring)] 페이지의 [작업 실행(Job runs)] 목록에서 작업을 선택한 다음 [실행 세부 정보 보기(View run details)]를 선택하여 해당 작업 실행에 대한 세부 정보를 볼 수 있습니다.

작업 실행 세부 정보 페이지에 표시되는 정보는 다음과 같습니다.

속성	설명
작업 이름	작업의 이름입니다.
실행 상태	작업 실행의 현재 상태입니다. 값은 다음과 같습니다. <ul style="list-style-type: none"> • STARTING • RUNNING • STOPPING • STOPPED • SUCCEEDED • FAILED • TIMEOUT
Glue 버전	작업 실행에 사용된 AWS Glue 버전입니다.

속성	설명
최근 시도	이 작업 실행에 대한 자동 재시도 횟수입니다.
시작 시간	이 작업이 시작된 날짜 및 시간.
종료 시간	이 작업 실행이 완료된 날짜 및 시간입니다.
시작 시간	작업 실행 준비에 소요된 시간입니다.
실행 시간	작업 스크립트 실행에 소요된 시간입니다.
트리거 이름	작업과 연결된 트리거의 이름입니다.
최근 수정 시간	작업이 마지막으로 수정된 날짜입니다.
보안 구성	Amazon S3 암호화, CloudWatch 암호화 및 작업 북마크 암호화 설정을 포함하는 작업에 대한 보안 구성입니다.
제한 시간	작업 실행 시간 제한 임계값입니다.
할당된 용량	이 작업 실행에 할당된 AWS Glue 데이터 처리 장치(DPU) 수입니다. 용량 계획에 대한 자세한 내용은 AWS Glue Developer Guide의 Monitoring for DPU Capacity Planning 을 참조하세요.
최대 용량	작업 실행에 사용할 수 있는 최대 용량.
작업자 수	작업 실행에 사용된 작업자 수입니다.

속성	설명
작업자 유형	<p>작업 실행에 할당된 사전 정의된 작업자 유형. 값은 G.1X 또는 G.2X일 수 있습니다.</p> <ul style="list-style-type: none"> • G.1X – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 1개의 DPU(vCPU 4개, 16GB 메모리, 64GB 디스크)에 매핑되고, 작업자당 실행기 1개를 제공합니다. 메모리 집약적인 작업의 경우 이 작업자 유형을 사용하는 것이 좋습니다. AWS Glue 버전 2.0 이상 작업의 기본 [작업자 유형(Worker type)]입니다. • G.2X – 이 유형을 선택할 경우 [작업자 수 (Number of workers)] 값도 제공합니다. 각 작업자는 2개의 DPU(vCPU 8개, 32GB 메모리, 128GB 디스크)에 매핑되고, 작업자당 실행기 1개를 제공합니다. 메모리 집약적인 작업과 기계 학습 변환을 실행하는 작업의 경우 이 작업자 유형을 사용하는 것이 좋습니다.
로그	연속 로깅을 위한 작업 로그에 대한 링크(/aws-glue/jobs/logs-v2)입니다.
출력 로그	작업 출력 로그 파일(/aws-glue/jobs/output)에 대한 링크입니다.
오류 로그	작업 오류 로그 파일(/aws-glue/jobs/error)에 대한 링크입니다.

최근 작업 실행에 대한 정보를 볼 때 제공되는 다음의 추가 항목도 볼 수 있습니다. 자세한 내용은 [the section called “최근 작업 실행에 대한 정보 보기”](#) 단원을 참조하십시오.

- 입력 인수
- 연속 로그
- 지표 - 기본 지표의 시각화를 볼 수 있습니다. 포함된 지표에 대한 자세한 내용은 [the section called “Spark 작업 실행에 대한 Amazon CloudWatch 지표 보기”](#) 섹션을 참조하세요.

- Spark UI - Spark UI에서 작업에 대한 Spark 로그를 시각화할 수 있습니다. Spark Web UI에 관해 자세한 내용은 [the section called “Spark UI로 모니터링”](#)를 참조하십시오. [the section called “작업을 위한 Spark UI 사용 설정”](#)의 절차에 따라 이 기능을 활성화합니다.

Spark 작업 실행에 대한 Amazon CloudWatch 지표 보기

작업 실행에 대한 세부 정보 페이지의 실행 세부 정보(Run details) 섹션 아래에서 작업 지표를 볼 수 있습니다. AWS Glue Studio는 모든 작업 실행의 작업 지표를 Amazon CloudWatch로 보냅니다.

AWS Glue은 30초마다 Amazon CloudWatch에 지표를 보고합니다. AWS Glue 지표는 이전에 보고한 값의 델타 값을 나타냅니다. 적절한 경우 지표 대시보드는 30초 값을 집계(합)하여 마지막 1분 전체에 대한 값을 얻습니다. 그러나 AWS Glue가 Amazon CloudWatch에 전달하는 Apache Spark 지표는 일반적으로 보고되는 시점의 현재 상태를 나타내는 절대값입니다.

Note

Amazon CloudWatch에 액세스하려면 계정을 구성해야 합니다.

지표는 다음과 같은 작업 실행에 대한 정보를 제공합니다.

- [ETL 데이터 이동(ETL Data Movement)] – Amazon S3에서 읽거나 쓴 바이트 수입니다.
- [메모리 프로파일: 사용된 힙(Memory Profile: Heap used)] – Java 가상 머신(JVM) 힙에서 사용하는 메모리 바이트 수입니다.
- [메모리 프로파일: 힙 사용량(Memory Profile: heap usage)] – JVM 힙에서 사용하는 메모리 부분(규모: 0~1, 백분율로 표시)입니다.
- [CPU 로드(CPU Load)] – 사용된 CPU 시스템 로드 부분(규모: 0~1, 백분율로 표시)입니다.

Ray 작업 실행에 대한 Amazon CloudWatch 지표 보기

작업 실행에 대한 세부 정보 페이지의 실행 세부 정보(Run details) 섹션 아래에서 작업 지표를 볼 수 있습니다. AWS Glue Studio는 모든 작업 실행의 작업 지표를 Amazon CloudWatch로 보냅니다.

AWS Glue은 30초마다 Amazon CloudWatch에 지표를 보고합니다. AWS Glue 지표는 이전에 보고한 값의 델타 값을 나타냅니다. 적절한 경우 지표 대시보드는 30초 값을 집계(합)하여 마지막 1분 전체에 대한 값을 얻습니다. 그러나 AWS Glue가 Amazon CloudWatch에 전달하는 Apache Spark 지표는 일반적으로 보고되는 시점의 현재 상태를 나타내는 절대값입니다.

Note

다음 사항에 설명된 대로 Amazon CloudWatch에 액세스하려면 계정을 구성해야 합니다.

Ray 작업에서는 다음과 같은 집계된 지표 그래프를 볼 수 있습니다. 이를 통해 클러스터 및 작업의 프로파일을 구축하고 각 노드에 대한 세부 정보에 액세스할 수 있습니다. 이 그래프를 지원하는 시계열 데이터는 추가 분석을 위해 CloudWatch에서 사용할 수 있습니다.

작업 프로파일: 작업 상태

시스템에 있는 Ray 작업 수를 표시합니다. 각 작업 수명 주기에는 고유한 시계열이 지정됩니다.

작업 프로파일: 작업 이름

시스템에 있는 Ray 작업 수를 표시합니다. 대기 중인 작업과 진행 중인 작업만 표시됩니다. 각 작업 유형(이름 기준)에는 고유한 시계열이 지정됩니다.

클러스터 프로파일: 사용 중인 CPU

사용된 CPU 코어 수를 표시합니다. 각 노드에는 고유한 시계열이 지정됩니다. 노드는 임시 IP 주소로 식별되며, IP 주소는 임시 주소이며 식별용으로만 사용됩니다.

클러스터 프로파일: 객체 스토어 메모리 사용

Ray 객체 캐시의 메모리 사용을 보여줍니다. 각 메모리 위치(물리적 메모리, 디스크에 캐시됨, Amazon S3에서 유출됨)에는 고유한 시계열이 지정됩니다. 객체 스토어는 클러스터의 모든 노드에서 데이터 스토리지를 관리합니다. 자세한 내용은 Ray 설명서의 [Objects](#)를 참조하세요.

클러스터 프로파일: 노드 수

클러스터에 프로비저닝된 노드 수를 표시합니다.

노드 세부 정보: CPU 사용

각 노드에서 CPU 사용률을 백분율로 표시합니다. 각 시리즈는 노드에 있는 모든 코어에서 집계된 CPU 사용률을 보여줍니다.

노드 세부 정보: 메모리 사용

각 노드에서 메모리 사용(GB)을 표시합니다. 각 시리즈는 Ray 작업 및 Plasma 저장 프로세스를 포함하여 노드의 모든 프로세스 사이에서 집계된 메모리를 보여줍니다. 디스크에 저장되거나 Amazon S3로 유출된 객체는 반영되지 않습니다.

노드 세부 정보: 디스크 사용

각 노드에서 디스크 사용(GB)을 표시합니다.

노드 세부 정보: 디스크 I/O 속도

각 노드에서 디스크 I/O(KB/s)를 표시합니다.

노드 세부 정보: 네트워크 I/O 처리량

각 노드에서 네트워크 I/O(KB/s)를 표시합니다.

노드 세부 정보: Ray 구성 요소별 CPU 사용

CPU 사용을 코어의 비율로 표시합니다. 각 노드의 각 Ray 구성 요소에 고유한 시계열이 지정됩니다.

노드 세부 정보: Ray 구성 요소별 메모리 사용

메모리 사용(GiB)을 표시합니다. 각 노드의 각 Ray 구성 요소에 고유한 시계열이 지정됩니다.

민감한 데이터 감지 및 처리

PII 탐지 변환은 데이터 원본에서 개인 식별 정보(PII)를 식별합니다. 식별할 PII 엔터티, 데이터를 스캔하는 방법, PII 탐지 변환에서 식별된 PII 엔터티로 수행할 작업을 선택합니다.

PII 탐지 변환은 사용자가 정의하거나 AWS에서 사전 정의한 엔터티를 탐지, 마스크 또는 제거하는 기능을 제공합니다. 이를 통해 규정 준수 수준을 높이고 책임 부담을 줄일 수 있습니다. 예를 들어 읽을 수 있는 개인 식별 정보가 데이터에 존재하지 않도록 하고 주민등록번호 (예: xxx-xx-xxxx), 전화번호 또는 주소를 고정된 문자열 (예:) 로 숨기고 싶을 수 있습니다.

AWS Glue Studio 외부에서 민감한 데이터를 처리하려면 [AWS Glue Studio 외부에서 민감한 데이터 탐지 사용](#)을 참조하십시오

주제

- [데이터를 스캔하는 방법 선택](#)
- [탐지할 PII 엔터티 선택](#)
- [탐지 민감도 수준 지정](#)
- [식별된 PII 데이터로 수행할 작업 선택](#)
- [세분화된 작업 오버라이드 추가](#)

데이터를 스캔하는 방법 선택

개인 식별 정보(PII)와 같은 민감한 데이터가 있는지 데이터 세트를 스캔할 때 각 행에서 PII를 탐지할지 아니면 PII 데이터가 포함된 열을 검색할지 선택할 수 있습니다.

<input type="radio"/> Detect PII in each cell Scan the entire data set, and act on each occurrence individually.	<input checked="" type="radio"/> Detect fields containing PII To reduce costs and improve performance, sample only a portion of the data and act on fields across all records.
--	--

Sample portion

The percentage of rows to sample out of the entire data set.

 %

Between 1 and 100.

Detection threshold

To consider a field as containing PII, set the minimum percentage of detected rows out of the sampled rows.

 %

Between 1 and 100.

각 셀에서 PII 탐지(Detect PII in each cell)를 선택할 경우 데이터 원본의 모든 행을 스캔하도록 선택하는 것입니다. PII 엔터티를 식별하기 위한 포괄적인 스캔입니다.

PII가 포함된 필드 탐지(Detect fields containing PII)를 선택할 경우 행 샘플에서 PII 엔터티를 스캔하도록 선택하는 것입니다. PII 엔터티가 있는 필드도 식별하면서 비용과 리소스를 낮게 유지하는 방법입니다.

PII가 포함된 필드를 탐지하도록 선택할 경우 행의 일부를 샘플링하여 비용을 절감하고 성능을 향상시킬 수 있습니다. 이 옵션을 선택하면 다음과 같은 추가 옵션을 지정할 수 있습니다.

- **부분 샘플링(Sample portion):** 샘플링할 행의 백분율을 지정할 수 있습니다. 예를 들어 '50'을 입력할 경우 50%의 행에서 PII 엔터티를 스캔하도록 지정하는 것입니다.
- **탐지 임계값(Detection threshold):** 전체 열이 PII 엔터티를 포함하는 것으로 식별되기 위해 PII 엔터티를 포함해야 하는 행의 백분율을 지정할 수 있습니다. 예를 들어 '10'을 입력할 경우 필드가 PII 엔터티(미국 전화)를 포함하는 것으로 식별되려면 스캔된 행의 PII 엔터티(미국 전화) 수가 10% 이상이어야 한다고 지정하는 것입니다. PII 엔터티를 포함하는 행의 백분율이 10% 미만일 경우 해당 필드는 PII 엔터티(미국 전화)를 포함하는 것으로 레이블이 지정되지 않습니다.

탐지할 PII 엔터티 선택

각 셀에서 PII 탐지(Detect PII in each cell)를 선택한 경우 다음 세 옵션 중 하나를 선택할 수 있습니다.

- 사용 가능한 모든 PII 패턴 - 여기에는 엔터티가 포함됩니다 AWS .
- 카테고리 선택 - 범주를 선택하면 PII 패턴이 선택한 범주의 패턴을 자동으로 포함합니다.
- 특정 패턴 선택(Select specific patterns) - 선택한 패턴만 탐지됩니다.

민감한 관리형 데이터 형식의 전체 목록은 [관리형 데이터 형식](#)을 참조하세요.

사용 가능한 모든 PII 패턴 중에서 선택

사용 가능한 모든 PII 패턴을 선택하는 경우 에 의해 사전 정의된 엔터티를 선택합니다. AWS엔터티를 하나 이상 또는 모두 선택할 수 있습니다.

Select entities to detect



Available entities (19)



Select all

Clear all

Create new

Manage

All categories ▼

< 1 >

<input type="checkbox"/>	Entity name ▼	Category ▲
<input type="checkbox"/>	Person's name	Universal, HIPAA
<input type="checkbox"/>	Email (General)	Universal
<input type="checkbox"/>	Credit Card	Universal
<input type="checkbox"/>	IP Address	Networking
<input type="checkbox"/>	MAC Address	Networking
<input type="checkbox"/>	US Phone	United States, HIPAA
<input type="checkbox"/>	US Passport	United States
<input type="checkbox"/>	Social Security Number (SSN)	United States, HIPAA
<input type="checkbox"/>	US Individual Taxpayer Identification Number (ITIN)	United States, HIPAA
<input type="checkbox"/>	US/Canada bank account	United States, HIPAA
<input type="checkbox"/>	US driving license	HIPAA
<input type="checkbox"/>	Healthcare Common Procedure Coding System (HCPCS) code	HIPAA
<input type="checkbox"/>	National Drug Code (NDC)	HIPAA
<input type="checkbox"/>	National Provider Identifier (NPI)	HIPAA
<input type="checkbox"/>	Drug Enforcement Agency (DEA) Registration Number	HIPAA
<input type="checkbox"/>	Health Insurance Claim Number (HICN)	HIPAA
<input type="checkbox"/>	Medicare Beneficiary Identifier	HIPAA

카테고리 선택

탐지할 PII 패턴으로 카테고리 선택(Select categories)을 선택한 경우 드롭다운 메뉴의 옵션에서 선택할 수 있습니다. 일부 엔터티는 둘 이상의 카테고리에 속할 수 있습니다. 예를 들어, 사람 이름(Person's name)은 일반(Universal) 및 HIPAA 카테고리에 속하는 엔터티입니다.

- 일반(예: 이메일, 신용카드)
- HIPAA(예: 미국 운전 면허증, HCPCS(Healthcare Common Procedure Coding System) 코드)
- 네트워킹(예: IP 주소, MAC 주소)
- 아르헨티나
- 호주
- 오스트리아
- 벨기에
- 보스니아
- 불가리아
- 캐나다
- 칠레
- 콜롬비아
- 크로아티아
- 사이프러스
- 체코
- 덴마크
- 에스토니아
- 핀란드
- 프랑스
- 독일
- 그리스
- 헝가리
- 아일랜드
- 한국
- 일본

- 멕시코
- 네덜란드
- 뉴질랜드
- 노르웨이
- 포르투갈
- 루마니아
- 싱가포르
- 슬로바키아
- 슬로베니아
- 스페인
- 스웨덴
- 스위스
- 터키
- 우크라이나
- 미국
- 영국
- 베네수엘라

특정 패턴 선택

탐지할 PII 패턴으로 특정 패턴 선택(Select specific patterns)을 선택한 경우 이미 만든 패턴 목록에서 검색하거나 찾아볼 수 있으며, 새 탐지 엔터티 패턴을 만들 수도 있습니다.

아래 단계에서는 민감한 데이터를 감지하기 위한 새로운 사용자 정의 패턴을 생성하는 방법을 설명합니다. 사용자 지정 패턴의 이름을 입력하여 사용자 지정 패턴을 생성하고 정규식을 추가하고 선택 사항으로 컨텍스트 단어를 정의합니다.

1. 새 패턴을 생성하려면 새로 생성(Create new) 버튼을 클릭합니다.

Select patterns

Browse

Create new

2. 탐지 엔터티 생성 페이지에서 엔터티 이름과 정규 표현식을 입력합니다. 정규 표현식(Regex)은 AWS Glue가 엔터티를 일치시키는 데 사용됩니다.
3. 검증(Validate)을 클릭합니다. 검증에 성공하면 문자열이 유효한 정규 표현식이라는 확인 메시지가 표시됩니다. 검증에 실패하면 문자열이 적절한 형식 및 허용된 문자 리터럴, 연산자 또는 구조를 준수하지 않는다는 메시지가 표시됩니다.
4. 정규 표현식 외에 컨텍스트 단어를 추가하도록 선택할 수 있습니다. 컨텍스트 단어는 일치 가능성을 높일 수 있습니다. 필드 이름이 엔터티를 설명하지 않는 경우에 유용할 수 있습니다. 예를 들어, 사회보장번호의 이름은 'SSN' 또는 'SS'일 수 있습니다. 이러한 컨텍스트 단어를 추가하면 엔터티를 일치시키는 데 도움이 될 수 있습니다.
5. 생성(Create)을 클릭하여 탐지 엔터티를 생성합니다. 생성된 모든 엔터티는 AWS Glue Studio 콘솔에 표시됩니다. 왼쪽 탐색 메뉴에서 탐지 엔터티(Detection entities)를 클릭합니다.

탐지 엔터티(Detection entities) 페이지에서 탐지 엔터티를 편집, 삭제 또는 생성할 수 있습니다. 검색 필드를 사용하여 패턴을 검색할 수도 있습니다.

탐지 민감도 수준 지정

민감한 데이터 탐지를 사용할 때 민감도 수준을 설정할 수 있습니다.

- 높음 - (기본값) 더 높은 수준의 민감도가 필요한 사용 사례에서 더 많은 개체를 탐지합니다. 2023년 11월 이후에 생성된 모든 AWS Glue 작업에는 이 설정이 자동으로 적용됩니다.
- 낮음 - 탐지되는 개체 수를 줄이고 오탐을 줄입니다.

Select global detection sensitivity

Choose the level of detection sensitivity to apply to your data set.

- High (default)**
Detects more entities for use cases that require a higher level of sensitivity.
- Low**
Detects fewer entities and reduces false positives.

식별된 PII 데이터로 수행할 작업 선택

전체 데이터 소스에서 PII를 탐지하도록 선택한 경우 적용할 글로벌 옵션을 선택할 수 있습니다.

- 탐지 결과로 데이터 보강(Enrich data with detection results): 각 셀에서 PII 탐지(Detect PII in each cell)를 선택한 경우 탐지된 엔터티를 새 열에 저장할 수 있습니다.

- **탐지된 텍스트 교정(Redact detected text):** 탐지된 PII 값을 선택적 바꾸기(Replacing) 텍스트 입력 필드에 지정한 문자열로 바꿀 수 있습니다. 문자열을 지정하지 않으면 탐지된 PII 엔터티가 '*****'로 바뀝니다.
- **탐지된 텍스트 부분 교정:** 탐지된 PII 값을 지정한 문자열로 바꿀 수 있습니다. 두 가지 옵션이 있습니다. 끝을 마스킹하지 않은 상태로 두거나 명시적인 정규식 패턴을 제공하여 마스킹하는 것입니다. 이 기능은 AWS Glue 2.0에서는 아직 사용할 수 없습니다.
- **Apply cryptographic hash(암호화 해시 적용):** 탐지된 PII 값을 SHA-256 암호화 해시 함수에 전달하고 이 값을 함수의 출력으로 바꿀 수 있습니다.

Select global action (required)

Choose an action to take on detected entities.

- DETECT. Enrich data with detection results.**
Create a new column that will contain any entity type detected in that row.
- REDACT. Redact detected text.**
Replace detected entity with a string you choose.
- PARTIAL_REDACT. Partially redact detected text.**
Replace part of a detected entity with a string you choose.
- SHA256_HASH. Apply cryptographic hash.**
Apply a SHA-256 cryptographic hash function to the input string.

AWS Glue 버전 2.0과 3.0 이상의 차이점

AWS Glue 2.0 작업은 각 열에 대해 탐지된 PII 정보가 DataFrame 포함된 새 작업을 보조 열에 반환합니다. 모든 수정 또는 해시 작업은 비주얼 탭의 AWS Glue 스크립트에서 볼 수 있습니다.

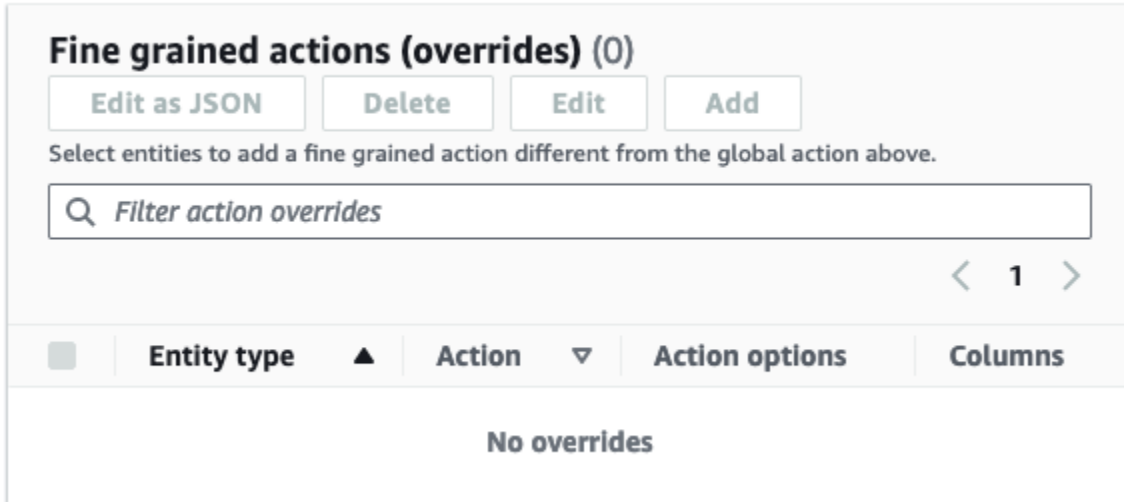
AWS Glue 3.0 및 4.0 작업에서는 이와 동일한 보조 열이 DataFrame 포함된 새 작업이 반환됩니다. “actionUsed”의 새 키가 존재하며, 이 키는 DETECT, REDACT, PARTIAL_REDACT, SHA256_HASH 중 하나가 될 수 있습니다. 마스킹 작업을 선택하면 민감한 데이터가 마스킹된 상태로 데이터가 DataFrame 반환됩니다.

세분화된 작업 오버라이드 추가

세부 조치 재정의 테이블에 탐지 및 조치 설정을 추가할 수 있습니다. 다음 작업을 수행할 수 있습니다.

- 특정 열을 탐지에서 포함 또는 제외 - 데이터 소스에서 추론된 스키마가 테이블을 사용 가능한 열로 채웁니다.
- 글로벌 액션을 사용하는 것보다 더 세밀한 특정 설정을 지정 - 예를 들어 엔터티 유형별로 다른 수정 텍스트 설정을 지정할 수 있습니다.

- 글로벌 액션과 다른 액션 지정 - 다른 민감한 데이터 유형에 다른 액션을 적용하려는 경우 여기에서 수행할 수 있습니다. 동일한 열에 두 가지 다른 edit-in-place 작업 (수정 및 해싱) 을 사용할 수는 없지만 감지는 항상 사용할 수 있습니다.



AWS Glue Studio를 사용하여 ETL 작업 관리

AWS Glue Studio의 간단한 그래픽 인터페이스를 사용하여 ETL 작업을 관리할 수 있습니다. 탐색 메뉴를 사용하여 [작업(Jobs)]을 선택하여 [작업(Jobs)] 페이지를 봅니다. 이 페이지에서 AWS Glue Studio 또는 AWS Glue 콘솔로 생성한 모든 작업을 볼 수 있습니다. 이 페이지에서 작업을 보고, 관리하고, 실행할 수 있습니다.

이 페이지에서 다음 태스크도 수행할 수 있습니다.

- [작업 실행 시작](#)
- [작업 실행 예약](#)
- [작업 일정 관리](#)
- [작업 실행 중지](#)
- [작업 보기](#)
- [최근 작업 실행에 대한 정보 보기](#)
- [작업 스크립트 보기](#)
- [작업 속성 수정](#)
- [작업 저장](#)
- [작업 복제](#)

- [작업 삭제](#)

작업 실행 시작

AWS Glue Studio에서 온디맨드로 작업을 실행할 수 있습니다. 작업은 여러 번 실행할 수 있으며 작업을 실행할 때마다 AWS Glue는 작업 활동 및 성능에 대한 정보를 수집합니다. 이 정보를 작업 실행이라고 하며 작업 실행 ID로 식별됩니다.

다음과 같은 방법으로 AWS Glue Studio에서 작업 실행을 시작할 수 있습니다.

- [작업(Jobs)] 페이지에서 시작할 작업을 선택한 다음 [작업 실행(Run job)] 버튼을 선택합니다.
- 비주얼 편집기에서 작업을 보고 있고 작업이 저장된 경우 [실행(Run)] 버튼을 선택하여 작업 실행을 시작할 수 있습니다.

작업 실행에 대한 자세한 내용은 AWS Glue Developer Guide의 [Working with Jobs on the AWS Glue Console](#)을 참조하세요.

작업 실행 예약

AWS Glue Studio에서 특정 시간에 작업이 실행되도록 일정을 생성할 수 있습니다. 작업이 실행되는 횟수, 실행되는 요일 및 실행되는 시간과 같은 제약 조건을 지정할 수 있습니다. 이러한 제약 조건은 cron을 기준으로 하며 cron과 동일한 제약 조건을 갖습니다. 예를 들어, 매월 31일에 작업을 실행하고자 한다면 매월 31일이 없다는 점을 유의하기 바랍니다. cron에 대한 자세한 내용은 AWS Glue Developer Guide의 [Cron Expressions](#)를 참조하세요.

일정에 따라 작업을 실행하려면

1. 다음 방법 중 하나를 사용하여 작업 일정을 생성합니다.
 - [작업(Jobs)] 페이지에서 일정을 생성할 작업을 선택하고 [작업(Actions)]을 선택한 다음 [작업 예약(Schedule job)]을 선택합니다.
 - 시각적 편집기에서 작업을 보고 있고 작업이 저장된 경우 [일정(Schedules)] 탭을 선택합니다. 그런 다음 [일정 생성(Create Schedule)]을 선택합니다.
2. [작업 실행 예약(Schedule job run)] 페이지에서 다음 정보를 입력합니다.
 - [이름(Name)]: 작업 예약의 이름을 입력합니다.
 - [빈도(Frequency)]: 작업 일정의 빈도를 입력합니다. 다음을 선택할 수 있습니다.

- [매시간(Hourly)]: 작업이 특정 분에 시작하여 매시간 실행됩니다. 작업을 실행해야 하는 [분(Minute)]을 지정할 수 있습니다. 기본적으로 매시간을 선택하면 작업이 시작 시간(0분)에 실행됩니다.
- [매일(Daily)]: 작업이 특정 시간에 시작하여 매일 실행됩니다. 작업을 실행해야 하는 [분(Minute)]과 작업의 [시작 시간(Start hour)]을 지정할 수 있습니다. 시간은 23시간제를 사용하여 지정되며 오후 시간에는 13~23의 숫자를 사용합니다. 분 및 시간의 기본값은 0입니다. 즉, [매일(Daily)]을 선택하면 기본적으로 작업이 자정에 실행됩니다.
- [매주(Weekly)]: 작업이 매주 하나 이상의 요일에 실행됩니다. 매일에 대해 앞에서 설명한 것과 동일한 설정 외에도 작업을 실행할 요일을 선택할 수 있습니다. 하루 이상을 선택할 수 있습니다.
- [매월(Monthly)]: 작업이 매월 특정 날짜에 실행됩니다. 매일에 대해 앞에서 설명한 것과 동일한 설정 외에도 작업을 실행할 날짜를 선택할 수 있습니다. 1~31의 숫자 값으로 날짜를 지정합니다. 존재하지 않는 날짜(예: 2월 30일)를 선택하면 해당 달에 작업이 실행되지 않습니다.
- [사용자 정의(Custom)]: cron 구문을 사용하여 작업 일정에 대한 표현식을 입력합니다. Cron 표현식을 사용하면 매월 말일(특정 날짜 대신) 또는 3개월마다 해당 월의 7일과 21일 같은 더 복잡한 일정을 생성할 수 있습니다.

AWS Glue Developer Guide의 [Cron Expressions](#)를 참조하세요.

- [설명(Description)]: 선택적으로 작업 일정에 대한 설명을 입력할 수 있습니다. 여러 작업에 대해 동일한 일정을 사용하려는 경우 설명이 있으면 작업 일정이 수행하는 작업을 더 쉽게 결정할 수 있습니다.
3. [일정 생성(Create schedule)]을 선택하여 작업 일정을 저장합니다.
 4. 일정을 생성하면 콘솔 페이지 상단에 성공 메시지가 나타납니다. 이 배너에서 [작업 세부 정보(Job details)]를 선택하여 작업 세부 정보를 볼 수 있습니다. 그러면 [일정(Schedules)] 탭이 선택된 상태로 시각적 작업 편집기 페이지가 열립니다.

작업 일정 관리

작업에 대한 일정을 생성한 후 시각적 편집기에서 작업을 열고 [일정(Schedules)] 탭을 선택하여 일정을 관리할 수 있습니다.

시각적 편집기의 [일정(Schedules)] 탭에서 다음 태스크를 수행할 수 있습니다.

- 새 예약을 생성합니다.

[일정 생성(Create schedule)]을 선택하고 [the section called “작업 실행 예약”](#)에 설명된 대로 일정 정보를 입력합니다.

- 기존 일정을 편집합니다.

편집할 일정을 선택한 다음 [작업(Action)]과 [일정 편집(Edit schedule)]을 차례로 선택합니다. 기존 일정을 편집하도록 선택하면 [빈도(Frequency)]가 [사용자 정의(Custom)]로 표시되고 일정이 cron 표현식으로 표시됩니다. cron 표현식을 수정하거나 [빈도(Frequency)] 버튼을 사용하여 새 일정을 지정할 수 있습니다. 변경을 마쳤으면 [예약 업데이트(Update schedule)]를 선택합니다.

- 활성 일정을 일시 중지합니다.

활성 일정을 선택한 다음 [작업(Action)]과 [일정 일시 중지(Pause schedule)]를 차례로 선택합니다. 일정이 즉시 비활성화됩니다. 새로 고침(다시 로드) 단추를 선택하여 업데이트된 작업 일정 상태를 봅니다.

- 일시 중지된 일정을 재개합니다.

비활성화된 일정을 선택한 다음 [작업(Action)]과 [일정 재개(Resume schedule)]를 차례로 선택합니다. 일정이 즉시 활성화됩니다. 새로 고침(다시 로드) 단추를 선택하여 업데이트된 작업 일정 상태를 봅니다.

- 일정을 삭제합니다.

제거할 일정을 선택한 다음 [작업(Action)]과 [일정 삭제>Delete schedule)]를 차례로 선택합니다. 일정이 즉시 삭제됩니다. 새로 고침(다시 로드) 단추를 선택하여 업데이트된 작업 일정 목록을 봅니다. 일정은 완전히 제거될 때까지 [삭제 중(Deleting)] 상태로 표시됩니다.

작업 실행 중지

작업 실행이 완료되기 전에 작업을 중지할 수 있습니다. 작업이 제대로 구성되지 않았거나 작업을 완료하는 데 너무 오래 걸리는 경우 이 옵션을 선택할 수 있습니다.

[모니터링(Monitoring)] 페이지의 [작업 실행(Job runs)] 목록에서 중지할 작업을 선택하고 [작업(Actions)]을 선택한 다음 [실행 중지(Stop run)]를 선택합니다.

작업 보기

[작업(Jobs)] 페이지에서 모든 작업을 볼 수 있습니다. 탐색 창에서 [작업(Jobs)]을 선택하여 이 페이지에 액세스할 수 있습니다.


[작업(Jobs)] 페이지에서 계정에 생성된 모든 작업을 볼 수 있습니다. [작업(Your jobs)] 목록에는 작업 이름, 유형, 해당 작업의 마지막 실행 상태 및 작업이 생성되고 마지막으로 수정된 날짜가 표시됩니다. 작업의 이름을 선택하여 해당 작업에 대한 자세한 정보를 볼 수 있습니다.

모니터링 대시보드를 사용하여 모든 작업을 볼 수도 있습니다. 탐색 창에서 [모니터링(Monitoring)]을 선택하여 대시보드에 액세스할 수 있습니다.

작업 표시 사용자 지정

[작업(Jobs)] 페이지의 [작업(Your jobs)] 섹션에 작업이 표시되는 방식을 사용자 지정할 수 있습니다. 또한 검색 텍스트 필드에 텍스트를 입력하여 해당 텍스트가 포함된 이름의 작업만 표시할 수 있습니다.

작업(Your jobs) 섹션에서 설정 아이콘

() 을 선택한 경우 AWS Glue Studio가 테이블에 정보를 표시하는 방식을 사용자 지정할 수 있습니다. 표시에서 텍스트 줄 바꿈을 선택하고, 페이지에 표시되는 작업 수를 변경하고, 표시할 열을 지정할 수 있습니다.

최근 작업 실행에 대한 정보 보기

소스 위치에 새 데이터가 추가되면 작업이 여러 번 실행될 수 있습니다. 작업이 실행될 때마다 작업 실행에 고유한 ID가 할당되고 해당 작업 실행에 대한 정보가 수집됩니다. 다음 방법을 사용하여 이 정보를 볼 수 있습니다.

- 현재 표시된 작업에 대한 작업 실행 정보를 보려면 시각적 편집기의 [실행(Runs)] 탭을 선택합니다.

[실행(Runs)] 탭([최근 작업 실행(Recent job runs)] 페이지)에는 각 작업 실행에 대한 카드가 있습니다. [실행(Runs)] 탭에 표시되는 정보는 다음과 같습니다.

- 작업 실행 ID
- 이 작업을 실행하려는 시도 횟수
- 작업 실행 상태
- 작업 실행의 시작 및 종료 시간
- 작업 실행을 위한 런타임
- 작업 로그 파일에 대한 링크
- 작업 오류 로그 파일에 대한 링크
- 실패한 작업에 대해 반환된 오류
- 작업 실행을 선택하면 다음을 포함하여 작업에 대한 추가 정보를 볼 수 있습니다.

- 입력 인수
- 연속 로그
- 지표 - 기본 지표의 시각화를 볼 수 있습니다. 포함된 지표에 대한 자세한 내용은 [the section called “Spark 작업 실행에 대한 Amazon CloudWatch 지표 보기”](#) 섹션을 참조하세요.
- Spark UI - Spark UI에서 작업에 대한 Spark 로그를 시각화할 수 있습니다. Spark Web UI에 관해 자세한 내용은 [the section called “Spark UI로 모니터링”](#)를 참조하십시오. [the section called “작업을 위한 Spark UI 사용 설정”](#)의 절차에 따라 이 기능을 활성화합니다.

세부 정보 보기를 선택하여 작업 실행 세부 정보 페이지에서 유사한 정보를 볼 수 있습니다. 모니터링 페이지를 통해 작업 실행 세부 정보 페이지로 이동할 수도 있습니다. 탐색 창에서 모니터링을 선택합니다. [작업 실행(Job runs)] 목록까지 아래로 스크롤합니다. 작업을 선택한 후 [실행 세부 정보 보기(View run details)]를 선택합니다. 내용은 [작업 실행의 세부 정보 보기](#)에 설명되어 있습니다.

작업 로그에 대한 자세한 내용은 [작업 실행 로그 보기](#) 섹션을 참조하세요.

작업 스크립트 보기

작업의 모든 노드에 대한 정보를 제공하면 AWS Glue Studio는 작업이 소스에서 데이터를 읽고, 데이터를 변환하고, 대상 위치에 데이터를 쓰는 데 사용하는 스크립트를 생성합니다. 작업을 저장하면 이 스크립트를 언제든지 볼 수 있습니다.

작업에 대해 생성된 스크립트를 보려면

1. 탐색 창에서 [작업(Jobs)]을 선택합니다.
2. [작업(Jobs)] 페이지의 [작업 목록(Your Jobs)]에서 검토할 작업의 이름을 선택합니다. 또는 목록에서 작업을 선택하고 [작업(Actions)] 메뉴를 선택한 다음 [작업 편집(Edit job)]을 선택할 수 있습니다.
3. 시각적 편집기 페이지에서 맨 위에 있는 [스크립트(Script)] 탭을 선택하여 작업 스크립트를 봅니다.

작업 스크립트를 편집하려면 [AWS Glue 프로그래밍 안내서](#) 섹션을 참조하세요.

작업 속성 수정

작업 다이어그램의 노드는 작업에 의해 수행되는 작업을 정의하지만 작업에 대해 구성할 수 있는 몇 가지 속성도 있습니다. 이러한 속성은 작업이 실행되는 환경, 작업이 사용하는 리소스, 임계값 설정, 보안 설정 등을 결정합니다.

작업 실행 환경을 사용 지정하려면

1. 탐색 창에서 [작업(Jobs)]을 선택합니다.
2. [작업(Jobs)] 페이지의 [작업 목록(Your Jobs)]에서 검토할 작업의 이름을 선택합니다.
3. 시각적 편집기 페이지에서 작업 편집 창 상단의 [작업 세부 정보(Job details)] 탭을 선택합니다.
4. 필요에 따라 작업 속성을 수정합니다.

작업 속성에 대한 자세한 내용은 AWS Glue Developer Guide의 [Defining Job Properties](#)를 참조하세요.

5. 다음과 같은 추가 작업 속성을 지정해야 하는 경우 [고급 속성(Advanced properties)] 섹션을 확장합니다.
 - [스크립트 파일 이름(Script filename)] - Amazon S3에 작업 스크립트를 저장하는 파일의 이름입니다.
 - [스크립트 경로(Script path)] - 작업 스크립트가 저장되는 Amazon S3 위치입니다.
 - [작업 지표(Job metrics)] - (Python 셸 작업에는 사용할 수 없음) 이 작업이 실행될 때 Amazon CloudWatch 지표 생성을 설정합니다.
 - [연속 로깅(Continuous logging)] - (Python 셸 작업에는 사용할 수 없음) CloudWatch에 대한 연속 로깅을 설정하여 작업이 완료되기 전에 로그를 볼 수 있습니다.
 - [Spark UI] 및 [Spark UI 로그 경로(Spark UI logs path)] - (Python 셸 작업에는 사용할 수 없음) 이 작업을 모니터링하기 위해 Spark UI를 사용하도록 설정하고 Spark UI 로그의 위치를 지정합니다.
 - [최대 동시성(Maximum concurrency)] - 이 작업에 허용된 최대 동시 실행 수를 설정합니다.
 - [임시 경로(Temporary path)] - AWS Glue가 스크립트를 실행할 때 임시 중간 결과가 작성된 Amazon S3의 작업 디렉터리의 위치를 제공합니다.
 - [지연 알림 임계값(분)(Delay notification threshold (minutes))] - 작업에 대한 지연 임계값을 지정합니다. 작업이 임계값에 지정된 시간보다 오래 실행되면 AWS Glue는 작업에 대한 지연 알림을 CloudWatch로 보냅니다.
 - [보안 구성(Security configuration)] 및 [서버 측 암호화(Server-side encryption)] - 이 필드를 사용하여 작업에 대한 암호화 옵션을 선택합니다.
 - [Glue Data Catalog를 Hive 메타스토어로 사용(Use Glue Data Catalog as the Hive metastore)] - Apache Hive Metastore의 대안으로 AWS Glue Data Catalog를 사용하려면 이 옵션을 선택합니다.
 - [추가 네트워크 연결(Additional network connection)] - VPC의 데이터 원본에 대해 Network 유형의 연결을 지정하여 작업이 VPC를 통해 데이터에 액세스할 수 있도록 할 수 있습니다.

- [Python 라이브러리 경로(Python library path)], [종속 jar 경로(Dependent jars path)](Python 셸 작업에는 사용할 수 없음) 또는 [참조된 파일 경로(Referenced files) path] – 이 필드를 사용하여 스크립트를 실행할 때 작업에서 사용하는 추가 파일의 위치를 지정합니다.
- [작업 파라미터(Job Parameters)] – 이름이 지정된 파라미터로 작업 스크립트에 전달되는 키-값 페어의 집합을 추가할 수 있습니다. Python은 AWS Glue API를 호출할 경우, 파라미터를 이름에 따라 정확하게 통과하는 것이 최선의 방법입니다. 작업 스크립트에서 파라미터 사용에 대한 자세한 내용은 AWS Glue Developer Guide의 [Passing and Accessing Python Parameters in AWS Glue](#)를 참조하세요.
- [태그(Tags)] - 작업에 태그를 추가하면 태그를 구성하고 식별하는 데 도움이 됩니다.

6. 작업 속성을 수정한 후 작업을 저장합니다.

Amazon S3에 Spark 셔플 파일 저장

일부 ETL 작업은 예를 들어 조인 변환을 사용할 때 여러 파티션에서 정보를 읽고 결합해야 합니다. 이 작업을 셔플링이라고 합니다. 셔플 중에 데이터가 디스크에 기록되고 네트워크를 통해 전송됩니다. AWS Glue 버전 3.0을 사용하면 Amazon S3를 이러한 파일의 저장 위치로 구성할 수 있습니다. AWS Glue는 Amazon S3에서 셔플 파일을 쓰고 읽는 셔플 관리자를 제공합니다. Amazon S3에서 셔플 파일 쓰기 및 읽기는 로컬 디스크(또는 Amazon EC2에 크게 최적화된 Amazon EBS)에 비해 느립니다 (5~20%). 그러나 Amazon S3는 무제한 스토리지 용량을 제공하므로 작업을 실행할 때 "No space left on device" 오류에 대해 걱정할 필요가 없습니다.

셔플 파일에 Amazon S3를 사용하도록 작업을 구성하려면

1. [작업(Jobs)] 페이지의 [작업 목록(Your Jobs)]에서 수정할 작업의 이름을 선택합니다.
2. 시각적 편집기 페이지에서 작업 편집 창 상단의 [작업 세부 정보(Job details)] 탭을 선택합니다.

[작업 파라미터(Job parameters)] 섹션으로 스크롤을 내립니다.

3. 다음 키-값 페어를 지정합니다.

- `--write-shuffle-files-to-s3 — true`

AWS Glue의 셔플 관리자가 Amazon S3 버킷을 사용하여 셔플 데이터를 쓰고 읽도록 구성하는 주요 파라미터입니다. 이 파라미터의 기본값은 `false`입니다.

- (선택 사항) `--write-shuffle-spills-to-s3 - true`

이 파라미터를 사용하면 유출 파일을 Amazon S3 버킷으로 오프로드할 수 있으므로 AWS Glue 에서 Spark 작업에 대한 추가 복원력을 제공합니다. 이는 많은 데이터를 디스크로 유출하는 대규모 워크로드에만 필요합니다. 이 파라미터의 기본값은 `false`입니다.

- (선택 사항) `--conf spark.shuffle.glue.s3ShuffleBucket - S3://<shuffle-bucket>`

이 파라미터는 셔플 파일을 작성할 때 사용할 Amazon S3 버킷을 지정합니다. 이 파라미터를 설정하지 않으면 위치는 [임시 경로(Temporary path)](`--TempDir`)에 지정된 위치의 `shuffle-data` 폴더입니다.

Note

셔플 버킷의 위치가 작업이 실행되는 동일한 AWS 리전에 있는지 확인합니다. 또한 셔플 서비스는 작업 실행이 완료된 후 파일을 정리하지 않으므로 셔플 버킷 위치에 Amazon S3 스토리지 수명 주기 정책을 구성해야 합니다. 자세한 내용은 Amazon S3 사용 설명서의 [스토리지 수명 주기 관리](#)를 참조하세요.

작업 저장

작업을 저장할 때까지 빨간색 [작업이 저장되지 않음(Job has not been saved)] 설명선이 [저장(Save)] 버튼 왼쪽에 표시됩니다.

Job has not been saved

Save

작업을 저장하려면

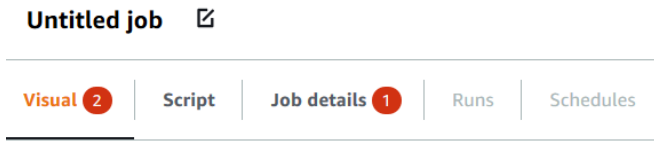
1. [시각적(Visual)] 및 [작업 세부 정보(Job details)] 탭에 필요한 모든 정보를 제공합니다.
2. 저장 버튼을 선택합니다.

작업을 저장한 후 '저장되지 않음(not saved)' 설명선이 변경되어 작업이 마지막으로 저장된 시간과 날짜를 표시합니다.

작업을 저장하기 전에 AWS Glue Studio를 종료하면 다음에 AWS Glue Studio에 로그인할 때 알림이 표시됩니다. 알림은 저장되지 않은 작업이 있음을 나타내며 복원할 것인지 묻습니다. 작업을 복원하도록 선택한 경우 작업을 계속 편집할 수 있습니다.

작업 저장 시 발생하는 오류 문제 해결

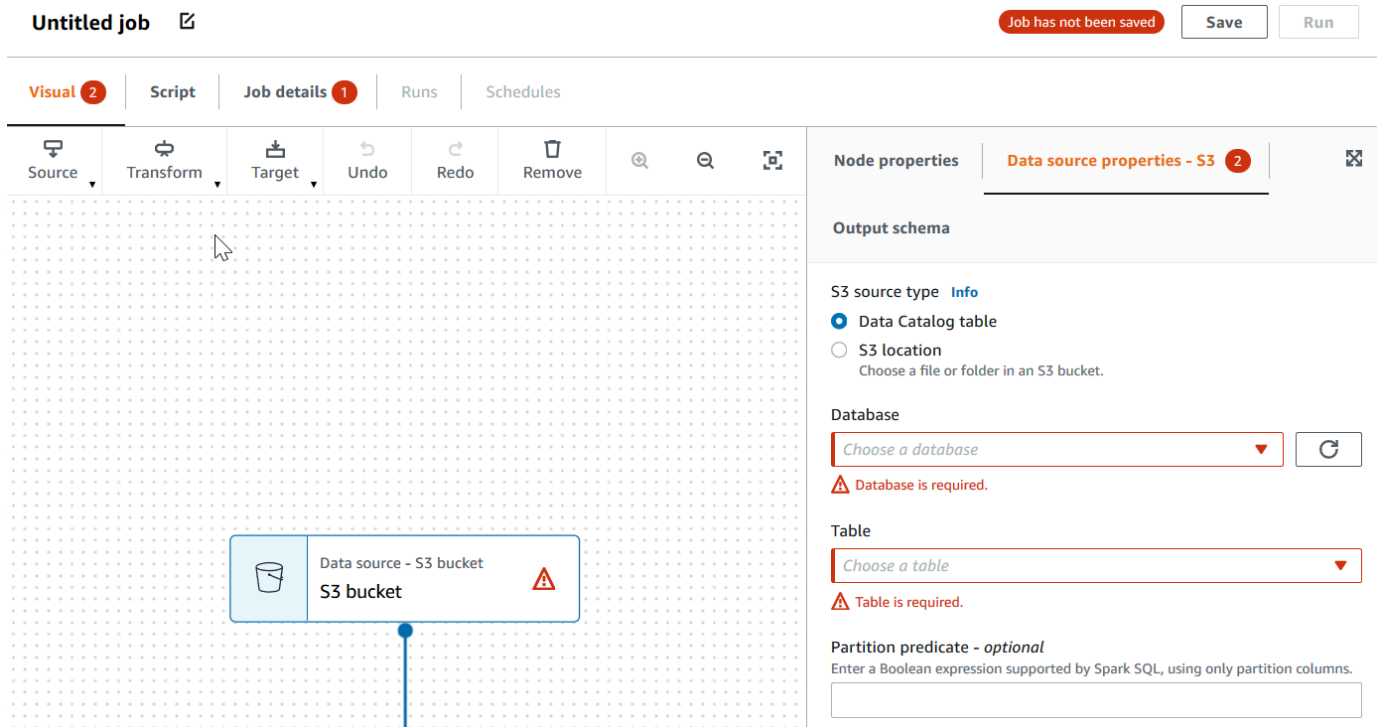
[저장(Save)] 버튼을 선택했지만 작업에 일부 필수 정보가 누락된 경우 정보가 누락된 탭에 빨간색 설명선이 나타납니다. 설명선의 숫자는 감지된 누락된 필드 수를 나타냅니다.




- 시각적 편집기의 노드가 올바르게 구성되지 않은 경우 [시각적(Visual)] 탭은 빨간색 설명선을 표시하고 오류가 있는 노드는 경고 기호

△ 표시합니다.

- 노드를 선택합니다. 노드 세부 정보 패널에서 누락되거나 잘못된 정보가 있는 탭에 빨간색 설명선이 나타납니다.
- 노드 세부 정보 패널에서 빨간색 설명선이 표시된 탭을 선택한 다음 강조 표시된 문제 필드를 찾습니다. 필드 아래의 오류 메시지는 문제에 대한 추가 정보를 제공합니다.



- 작업 속성에 문제가 있는 경우 [작업 세부 정보(Job details)] 탭에 빨간색 설명선이 표시됩니다. 해당 탭을 선택하고 강조 표시된 문제 필드를 찾습니다. 필드 아래의 오류 메시지는 문제에 대한 추가 정보를 제공합니다.

Untitled job 

Visual **2** | Script | **Job details 1** | Runs | Schedules

Basic properties [Info](#)

Name

Description - optional

Descriptions can be up to 2048 characters long.

IAM Role

Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job.

 IAM Role is required.

Type

The type of ETL job. This is set automatically based on the types of data sources you have selected.

작업 복제

[작업 복제(Clone job)] 작업을 사용하여 기존 작업을 새 작업으로 복사할 수 있습니다.

기존 작업을 복사하여 새 작업을 생성하려면

1. [작업(Jobs)] 페이지의 [작업 목록(Your jobs)]에서 복제할 작업을 선택합니다.
2. [작업(Actions)] 메뉴에서 [작업 복제(Clone job)]를 선택합니다.
3. 새 작업의 이름을 입력합니다. 그런 다음 작업을 저장하거나 편집할 수 있습니다.

작업 삭제

더 이상 필요 없는 작업을 제거할 수 있습니다. 단일 작업으로 하나 이상의 작업을 삭제할 수 있습니다.

AWS Glue Studio에서 작업을 제거하려면

1. [작업(Jobs)] 페이지의 [작업 목록(Your jobs)]에서 삭제할 작업을 선택합니다.

2. [작업(Actions)] 메뉴에서 [작업 삭제>Delete job)]를 선택합니다.
3. **delete**를 입력하여 작업 삭제를 확인합니다.

시각적 편집기에서 해당 작업에 대한 [작업 세부 정보(Job details)] 탭을 볼 때 저장된 작업을 삭제할 수도 있습니다.

AWS Glue에서 작업 사용

다음 섹션에서는 AWS Glue에서 ETL 및 Ray 작업에 대한 정보를 제공합니다.

주제

- [AWS Glue 버전](#)
- [에서 Spark 채용공고를 다루기 AWS Glue](#)
- [AWS Glue에서 Ray 작업 사용](#)
- [AWS Glue에서 Python 셸 작업에 대한 작업 속성 구성](#)
- [AWS Glue 모니터링](#)
- [AWS Glue 작업 실행 상태](#)

AWS Glue 버전

작업을 추가하거나 업데이트할 때 AWS Glue 버전 파라미터를 구성할 수 있습니다. AWS Glue 버전은 AWS Glue가 지원하는 Apache Spark 및 Python 버전을 결정합니다. Python의 버전으로 Spark 유형의 작업에 대해 지원되는 버전을 확인할 수 있습니다. 다음 테이블에는 이용 가능한 AWS Glue 버전과 그에 상응하는 Spark 및 Python 버전, 그리고 다른 기능 변경 사항이 나열됩니다.

AWS Glue 버전

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
AWS Glue4.0	Spark 환경 버전 <ul style="list-style-type: none"> • Spark 3.3.0 • Python 3.10 	Java 8	AWS Glue의 최신 버전은 AWS Glue 4.0입니다. 이 AWS Glue 릴리스에는 다음과 같은 몇 가지 최적화 및 업그레이드가 기본으로 제공됩니다. <ul style="list-style-type: none"> • Spark 3.1에서 Spark 3.3으로 다양

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			<p>한 Spark 기능 업그레이드</p> <ul style="list-style-type: none"> • Pandas와 연결할 경우 몇 가지 기능이 개선됩니다. 자세한 내용은 Spark 3.3의 새로운 기능을 참조하세요. • Amazon EMR에서 개발된 추가 최적화 포함 • EMR 파일 시스템 (EMRFS) 2.53으로 업그레이드 • Log4j 1.x에서 Log4j 2로 마이그레이션 • Boto의 업그레이드 버전과 같은 AWS Glue 3.0의 여러 Python 모듈 업데이트 • 기본 Amazon Redshift 커넥터를 비롯한 여러 커넥터 업그레이드 부록 C: 커넥터 업그레이드 섹션을 참조하십시오. • 여러 JDBC 드라이버 업그레이드 부록 B: JDBC 드라이버

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			<p>업그레이드 섹션을 참조하십시오.</p> <ul style="list-style-type: none"> • 새로운 Amazon Redshift 커넥터와 JDBC 드라이버로 업데이트 • Apache Hudi, Delta Lake 및 Apache Iceberg를 통해 개방형 데이터 레이크 프레임워크 기본 지원 • Amazon S3를 사용하여 셔플링 및 탄력적인 스토리지 용량을 지원할 수 있도록 Amazon S3 기반 클라우드 셔플 스토리지 플러그인(Apache Spark 플러그인) 기본 지원 <p>제한 사항</p> <p>다음은 AWS Glue 4.0의 제한 사항입니다.</p> <ul style="list-style-type: none"> • AWS Glue 기계 학습 및 개인 식별 정보(PII) 변환은 AWS Glue 4.0에서 아직 사용할 수 없습니다.

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			AWS Glue 버전 4.0으로 마이그레이션에 대한 자세한 내용은 AWS Glue 버전 4.0으로 AWS Glue for Spark 작업 마이그레이션 섹션을 참조하세요.

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
	Ray 환경 버전 <ul style="list-style-type: none"> • Ray 2.4.0 Python 3.9	N/A	Ray를 사용하여 AWS Glue 분산 Python 애플리케이션을 빌드하고 실행합니다. <ul style="list-style-type: none"> • Python 3.9에서 Ray-2.4.0 데이터 배포(ray[data])를 지원합니다. 이번 Ray 릴리스에 대한 자세한 내용은 Ray 리포지토리의 Ray-2.4.0을 참조하십시오. GitHub • Ray2.4 런타임 환경에 추가 Python 라이브러리 설치를 지원합니다. 자세한 정보는 the section called “Ray 작업을 위한 추가 Python 모듈”을 참조하세요. • Ray 작업의 로그와 지표를 CloudWatch Amazon과 통합합니다. 자세한 내용은 the section called “Ray 오류 해결” 및 the section called “Ray 작업 지표” 섹션을 참조하세요. • 각 작업 실행 페이지에서 Ray 작업에 대


AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			<p>한 지표를 집계하고 시각화합니다. AWS Glue Studio</p> <ul style="list-style-type: none"> 클러스터의 각 작업 디렉터리에 파일을 배포하고, Ray 객체 스토어에서 Amazon S3로 객체를 유출하며, Ray 작업에 할당된 워커 노드의 최소 수를 제어하도록 지원합니다. 자세한 정보는 the section called "Ray 작업 파라미터"을 참조하세요. <p>AWS Glue 4.0에서 Ray 작업 제한</p> <ul style="list-style-type: none"> AWS Glue Ray의 대화형 세션은 이번 릴리스에서 계속 프리뷰로 제공됩니다. AWS Glue Amazon VPC와의 Ray 통합은 현재 사용할 수 없습니다. 공개 경로가 없으면 VPC의 AWS 리소스에 액세스할 수 없습니다. Amazon AWS Glue VPC와 함께 사용하

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			<p>는 방법에 대한 자세한 내용은 을 참조하십시오. <u>the section called “VPC 엔드포인트(AWS PrivateLink)”</u></p> <ul style="list-style-type: none"> • AWS Glue for Ray 는 미국 동부 (버지니아 북부), 미국 동부 (오하이오), 미국 서부 (오레곤), 아시아 태평양 (도쿄), 유럽 (아일랜드) 에서 사용할 수 있습니다.

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
AWS Glue3.0	<ul style="list-style-type: none"> • Spark 3.1.1 • Python 3.7 	Java 8	<p>Spark 엔진을 3.0으로 업그레이드하는 것 외에도 이 AWS Glue 릴리스에는 다음과 같은 최적화 및 업그레이드가 내장되어 있습니다.</p> <ul style="list-style-type: none"> • Spark의 주요 릴리스인 Spark 3.0에 대해 AWS Glue ETL 라이브러리를 구축합니다. • 스트리밍 작업은 AWS Glue 3.0에서 지원됩니다. • 성능과 안정성을 위한 새로운 AWS Glue Spark 런타임 최적화를 포함합니다. <ul style="list-style-type: none"> • CSV 데이터 읽기를 위한 Apache Arrow 기반의 더 빠른 인메모리 열 형식 처리. • CSV 데이터로 벡터화된 읽기를 위한 SIMD 기반 실행. • Spark 업그레이드에는 Amazon EMR에서 개발된

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			<p>추가 최적화도 포함됩니다.</p> <ul style="list-style-type: none"> EMRFS가 2.38에서 2.46으로 업그레이드되어 Amazon S3 액세스에 대한 새로운 기능과 버그 수정이 가능합니다. 새 Spark 버전에 필요한 여러 종속성이 업그레이드되었습니다. 부록 A: 중요한 종속성 업그레이드 섹션을 참조하십시오. 기본적으로 지원되는 데이터 원본용으로 JDBC 드라이버가 업그레이드되었습니다. 부록 B: JDBC 드라이버 업그레이드 섹션을 참조하십시오. <p>제한 사항</p> <p>다음은 AWS Glue 3.0의 제한 사항입니다.</p> <ul style="list-style-type: none"> AWS Glue 기계 학습 변환은 아직 AWS Glue 3.0에서 사용할 수 없습니다.

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			<ul style="list-style-type: none"> 일부 사용자 정의 Spark 커넥터는 Spark 2.4에 종속되고 Spark 3.1과 호환되지 않는 경우 AWS Glue 3.0에서 작동하지 않습니다. <p>AWS Glue 버전 3.0으로 마이그레이션에 대한 자세한 내용은 AWS Glue 버전 3.0으로 AWS Glue for Spark 작업 마이그레이션 섹션을 참조하세요.</p>

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
<p>AWS Glue 2.0(지원 중 단됨, 지원 종료)</p>	<ul style="list-style-type: none"> • Spark 2.4.3 • Python 3.7 	<p>N/A</p>	<p>AWS Glue 버전 1.0에서 제공되는 기능 외에도 AWS Glue 버전 2.0은 다음을 제공합니다.</p> <ul style="list-style-type: none"> • AWS Glue에서 Apache Spark ETL 작업을 실행하고 시작 시간을 단축하기 위한 업그레이드된 인프라. • 기본 로깅은 이제 드라이버와 실행기, 출력 및 오류에 대한 별도의 스트림을 사용하여 실시간으로 이루어집니다. • 작업 수준에서 추가 Python 모듈 또는 다른 버전 지정 지원. <div data-bbox="1187 1291 1511 1852" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; margin-top: 10px;"> <p> Note</p> <p>AWS Glue 버전 2.0은 기본 아키텍처 변경으로 인해 일부 종속성 및 버전에 대해 AWS Glue 버전 1.0과 다릅니다. 주요 AWS Glue 버</p> </div>

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			<p>전 릴리스 간에 마이그레이션하기 전에 AWS Glue 작업을 검증합니다.</p> <p>AWS Glue 버전 2.0의 기능 및 제한 사항에 대한 자세한 내용은 단축된 시작 시간으로 Spark ETL 작업 실행 섹션을 참조하세요.</p>

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
<p>AWS Glue 1.0(지원 중 단됨, 지원 종료)</p>	<ul style="list-style-type: none"> • Spark 2.4.3 • Python 2.7 • Python 3.6 	<p>N/A</p>	<p>AWS Glue ETL 작업에서는 Parquet 및 ORC 포맷에 대해 작업 북마크를 유지할 수 있습니다(AWS Glue 버전 1.0 사용). 이전에는 AWS Glue ETL 작업에서 JSON, CSV, Apache Avro, XML처럼 일반적인 Amazon S3 소스 형식만 북마크가 가능했습니다.</p> <p>ETL 입력 및 출력의 포맷 옵션을 설정할 때 Apache Avro 리더/라이터 포맷 1.8을 사용해 Avro 논리적 유형 읽기 및 쓰기를 지원하도록 지정할 수 있습니다(AWS Glue 버전 1.0 사용). 이전에는 버전 1.7 Avro 리더/라이터 포맷만 지원되었습니다.</p> <p>DynamoDB 연결 유형은 라이터 옵션(AWS Glue 버전 1.0 사용)을 지원합니다.</p> <p>제한 사항</p> <p>다음은 AWS Glue 1.0의 제한 사항입니다.</p>

AWS Glue 버전	지원되는 런타임 환경 버전	지원되는 자바 버전	기능 변경 사항
			<ul style="list-style-type: none"> AWS Glue 버전 0.9 및 1.0은 아시아 태평양(자카르타)(ap-southeast-3), 중동(UAE)(me-central-1) 또는 향후 기타 신규 리전에서 사용할 수 없습니다.
<p>AWS Glue 0.9(지원 중 단됨, 지원 종료)</p>	<ul style="list-style-type: none"> Spark 2.2.1 Python 2.7 	<p>N/A</p>	<p>AWS Glue 버전 지정 없이 생성된 작업은 AWS Glue 0.9로 기본 지정됩니다.</p> <p>제한 사항</p> <p>다음은 AWS Glue 0.9의 제한 사항입니다.</p> <ul style="list-style-type: none"> AWS Glue 버전 0.9 및 1.0은 아시아 태평양(자카르타)(ap-southeast-3), 중동(UAE)(me-central-1) 또는 향후 기타 신규 리전에서 사용할 수 없습니다.

단축된 시작 시간으로 Spark ETL 작업 실행

AWS Glue 버전 2.0 이상에서는 시작 시간을 단축하여 AWS Glue에서 Apache Spark ETL(추출, 변환, 로드) 작업을 실행하기 위한 업그레이드된 인프라를 제공합니다. 대기 시간이 단축되어 데이터 엔지니어

어는 생산성을 높이고 AWS Glue와의 상호 작용성을 높일 수 있습니다. 작업 시작 시간의 편차를 줄이면 분석에 사용할 수 있는 데이터의 SLA를 충족하거나 초과할 수 있습니다.

AWS Glue ETL 작업에 이 기능을 사용하려면 작업 생성 시 Glue version에 대해 **2.0** 버전 이상을 선택합니다.

주제

- [지원되는 새로운 기능](#)
- [로깅 동작](#)
- [지원되지 않는 기능](#)

지원되는 새로운 기능

이 섹션에서는 AWS Glue 버전 2.0 이상에서 지원되는 새로운 기능에 대해 설명합니다.

작업 수준에서 추가 Python 모듈 지정 지원

AWS Glue 버전 2.0 이상을 사용하면 작업 수준에서 추가 Python 모듈이나 다른 버전을 제공할 수도 있습니다. `--additional-python-modules` 옵션을 쉼표로 구분된 Python 모듈 목록과 함께 사용하여 새 모듈을 추가하거나 기존 모듈의 버전을 변경할 수 있습니다.

예를 들어 새 `scikit-learn` 모듈을 업데이트하거나 추가하려면 키/값 `--additional-python-modules`, `"scikit-learn==0.21.3"`을 사용합니다.

또한 `--additional-python-modules` 옵션 내에서 Python 휠 모듈에 대한 Amazon S3 경로를 지정할 수 있습니다. 예:

```
--additional-python-modules s3://aws-glue-native-spark/tests/j4.2/ephem-3.7.7.1-cp37-cp37m-linux_x86_64.whl,s3://aws-glue-native-spark/tests/j4.2/fbprophet-0.6-py3-none-any.whl,scikit-learn==0.21.3
```

AWS Glue는 Python 패키지 설치 프로그램(pip3)을 사용하여 추가 모듈을 설치합니다. 모듈 설치를 위해 `python-modules-installer-option`으로 지정된 추가 옵션을 pip3에 전달할 수 있습니다. pip3의 비호환성 또는 제한 사항이 적용됩니다.

AWS Glue 버전 2.0에서 이미 제공되는 Python 모듈

AWS Glue 버전 2.0은 기본적으로 다음과 같은 Python 모듈을 지원합니다.

- `setuptools`—45.2.0

- subprocess32—3.5.4
- ptvsd—4.3.2
- pydevd—1.9.0
- PyMySQL—0.9.3
- docutils—0.15.2
- jmespath—0.9.4
- six—1.14.0
- python_dateutil—2.8.1
- urllib3—1.25.8
- botocore—1.15.4
- s3transfer—0.3.3
- boto3—1.12.4
- certifi—2019.11.28
- chardet—3.0.4
- idna—2.9
- requests—2.23.0
- pyparsing—2.4.6
- enum34—1.1.9
- pytz—2019.3
- numpy—1.18.1
- cyclcr—0.10.0
- kiwisolver—1.1.0
- scipy—1.4.1
- pandas—1.0.1
- pyarrow—0.16.0
- matplotlib—3.1.3
- pyhocon—0.3.54
- mpmath—1.1.0
- sympy—1.5.1
- patsy—0.5.1

- statsmodels—0.11.1
- fsspec—0.6.2
- s3fs—0.4.0
- Cython—0.29.15
- joblib—0.14.1
- pmdarima—1.5.3
- scikit-learn—0.22.1
- tbats—1.0.9

로깅 동작

AWS Glue 버전 2.0 이상에서는 다른 기본 로깅 동작을 지원합니다. 차이점은 다음과 같습니다.

- 로깅이 실시간으로 수행됩니다.
- 드라이버와 실행기를 위한 별도의 스트림이 있습니다.
- 각 드라이버와 실행기에는 출력 스트림과 오류 스트림의 두 가지 스트림이 있습니다.

드라이버 및 실행기 스트림

드라이버 스트림은 작업 실행 ID로 식별됩니다. 실행기 스트림은 작업 `<run id>_<executor task id>`로 식별됩니다. 예:

- "logStreamName":
"jr_8255308b426fff1b4e09e00e0bd5612b1b4ec848d7884cebe61ed33a31789..._g-f65f617bd31d54bd94482af755b6cdf464542..."

출력 및 오류 스트림

출력 스트림에는 코드의 표준 출력(stdout)이 있습니다. 오류 스트림에는 코드/라이브러리의 로깅 메시지가 있습니다.

- 로그 스트림:
 - 드라이버 로그 스트림에는 `<jr>`이 있으며, 여기서 `<jr>`은 작업 실행 ID입니다.
 - 실행기 로그 스트림에는 `<jr>_<g>`가 있습니다. 여기서 `<g>`는 실행기의 태스크 ID입니다. 드라이버 오류 로그에서 실행기 태스크 ID를 조회할 수 있습니다.

AWS Glue 버전 2.0의 기본 로그 그룹은 다음과 같습니다.

- 출력의 경우 `/aws-glue/jobs/logs/output`
- 오류의 경우 `/aws-glue/jobs/logs/error`

보안 구성이 제공되면 로그 그룹 이름이 다음으로 변경됩니다.

- `/aws-glue/jobs/<security configuration>-role/<Role Name>/output`
- `/aws-glue/jobs/<security configuration>-role/<Role Name>/error`

콘솔에서 [로그(Logs)] 링크는 출력 로그 그룹을 가리키고 [오류(Error)] 링크는 오류 로그 그룹을 가리킵니다. 연속 로깅을 사용하면 [로그(Logs)] 링크는 연속 로그 그룹을 가리키고 [출력(Output)] 링크는 출력 로그 그룹을 가리킵니다.

로깅 규칙

Note

연속 로깅의 기본 로그 그룹 이름은 `/aws-glue/jobs/logs-v2`입니다.

AWS Glue 버전 2.0 이상에서 연속 로깅은 AWS Glue 버전 1.0과 동일한 동작을 수행합니다.

- 기본 로그 그룹: `/aws-glue/jobs/logs-v2`
- 드라이버 로그 스트림: `<jr>-driver`
- 실행기 로그 스트림: `<jr>-<executor ID>`

`--continuous-log-logGroupName`을 설정하여 로그 그룹 이름을 변경할 수 있습니다.

`--continuous-log-logStreamPrefix`를 설정하여 로그 스트림 이름에 접두사를 붙일 수 있습니다.

지원되지 않는 기능

다음 AWS Glue 기능은 지원되지 않습니다.

- 개발 엔드포인트

- AWS Glue 버전 2.0 이상은 Apache YARN에서 실행되지 않으므로 YARN 설정이 적용되지 않습니다.
- AWS Glue 버전 2.0 이상에는 Hadoop 분산 파일 시스템(HDFS)이 없습니다.
- AWS Glue 버전 2.0 이상에서는 동적 할당을 사용하지 않으므로 ExecutorAllocationManager 지표가 제공되지 않습니다.
- AWS Glue 버전 2.0 이상 작업의 경우 작업자 수와 작업자 유형을 지정하지만 maxCapacity는 지정하지 않습니다.
- AWS Glue 버전 2.0 이상에서는 기본값으로 s3n을 지원하지 않습니다. s3 또는 s3a를 사용하는 것이 좋습니다. 어떤 이유로 작업에서 s3n을 사용해야 하는 경우 다음 추가 인수를 전달할 수 있습니다.

```
--conf spark.hadoop.fs.s3n.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem
```

AWS Glue 버전 3.0으로 AWS Glue for Spark 작업 마이그레이션

이 주제에서는 Spark 애플리케이션 및 ETL 작업을 AWS Glue 3.0으로 마이그레이션할 수 있도록 하는 AWS Glue 버전 0.9, 1.0, 2.0 및 3.0 간의 변경 사항에 대해 설명합니다.

AWS Glue ETL 작업에 이 기능을 사용하려면 작업 생성 시 Glue version으로 **3.0**을 선택합니다.

주제

- [지원되는 새로운 기능](#)
- [AWS Glue 3.0으로 마이그레이션할 작업](#)
- [마이그레이션 체크리스트](#)
- [AWS Glue 0.9에서 AWS Glue 3.0으로 마이그레이션](#)
- [AWS Glue 1.0에서 AWS Glue 3.0으로 마이그레이션](#)
- [AWS Glue 2.0에서 AWS Glue 3.0으로 마이그레이션](#)
- [부록 A: 중요한 종속성 업그레이드](#)
- [부록 B: JDBC 드라이버 업그레이드](#)

지원되는 새로운 기능

이 섹션에서는 AWS Glue 버전 3.0의 새로운 기능과 장점에 대해 설명합니다.

- 오픈 소스 Spark에서 최적화되고 적응형 쿼리 실행, 벡터화된 리더, 최적화된 셔플 및 파티션 병합과 같은 AWS Glue 및 EMR 서비스에 의해 개발된 Apache Spark 3.1.1을 기반으로 합니다.
- MySQL, Microsoft SQL Server, Oracle, PostgreSQL, MongoDB 및 Spark 3.1.1에서 가져온 업그레이드된 Spark 라이브러리 및 종속성을 포함한 모든 Glue 기본 소스용 JDBC 드라이버가 업그레이드되었습니다.
- 업그레이드된 EMRFS로 Amazon S3 액세스가 최적화되었으며 Amazon S3 최적화 출력 커미터가 기본적으로 사용됩니다.
- 파티션 인덱스, 푸시다운 조건자, 파티션 목록 및 업그레이드된 Hive 메타스토어 클라이언트로 Data Catalog 액세스가 최적화되었습니다.
- 셀 수준 필터링 및 데이터 레이크 트랜잭션이 있는 관리되는 카탈로그 테이블을 위해 Lake Formation과 통합됩니다.
- 새로운 Spark 실행기 메모리 지표 및 Spark 정형 스트리밍 지표를 사용하여 Spark 3.1.1의 Spark UI 환경이 개선되었습니다.
- AWS Glue 2.0과 유사하게 시작 대기 시간이 단축되어 전반적인 작업 완료 시간과 상호 작용성이 개선되었습니다.
- Spark 작업은 AWS Glue 2.0과 유사하게 최소 10분에서 1분으로 10배 더 짧은 최소 청구 기간으로 1초 단위로 청구됩니다.

AWS Glue 3.0으로 마이그레이션할 작업

기존 작업의 경우 작업 구성에서 Glue version을 이전 버전에서 Glue 3.0으로 변경합니다.

- 콘솔의 Glue version에서 Spark 3.1, Python 3 (Glue Version 3.0) or Spark 3.1, Scala 2 (Glue Version 3.0)를 선택합니다.
- AWS Glue Studio의 Glue version에서 Glue 3.0 - Supports spark 3.1, Scala 2, Python 3을 선택합니다.
- [UpdateJob](#) API의 GlueVersion 파라미터에서 **3.0**을 선택합니다.

새 작업의 경우 작업을 생성할 때 Glue 3.0을 선택합니다.

- 콘솔의 Glue version에서 Spark 3.1, Python 3 (Glue Version 3.0) or Spark 3.1, Scala 2 (Glue Version 3.0)를 선택합니다.
- AWS Glue Studio의 Glue version에서 Glue 3.0 - Supports spark 3.1, Scala 2, Python 3을 선택합니다.

- [CreateJob](#) API의 GlueVersion 파라미터에서 **3.0**을 선택합니다.

AWS Glue 3.0의 Spark 이벤트 로그를 보려면 [CloudFormation](#) 또는 [Docker](#)를 사용하여 [Glue 3.0용으로 업그레이드된 Spark 기록 서버를 시작](#)합니다.

마이그레이션 체크리스트

마이그레이션을 위해 이 체크리스트를 검토합니다.

- 작업이 HDFS에 종속되나요? 그렇다면 HDFS를 S3로 교체해 봅니다.
 - 작업 스크립트 코드에서 DFS 경로로 `hdfs://` 또는 `/`로 시작하는 파일 시스템 경로를 검색합니다.
 - 기본 파일 시스템이 HDFS로 구성되어 있지 않은지 확인합니다. 명시적으로 구성된 경우 `fs.defaultFS` 구성을 제거해야 합니다.
 - 작업에 `dfs.*` 파라미터가 포함되어 있는지 확인합니다. 포함된 경우 파라미터를 사용 중지해도 괜찮은지 확인해야 합니다.
- 작업이 YARN에 종속되나요? 그렇다면 작업에 다음 파라미터가 포함되어 있는지 확인하여 영향을 확인합니다. 포함된 경우 파라미터를 사용 중지해도 괜찮은지 확인해야 합니다.
 - `spark.yarn.*`

예:

```
spark.yarn.executor.memoryOverhead
spark.yarn.driver.memoryOverhead
spark.yarn.scheduler.reporterThread.maxFailures
```

- `yarn.*`

예:

```
yarn.scheduler.maximum-allocation-mb
yarn.nodemanager.resource.memory-mb
```

- 작업이 Spark 2.2.1 또는 Spark 2.4.3에 종속되나요? 그렇다면 작업이 Spark 3.1.1에서 변경된 기능을 사용하는지 확인하여 영향을 확인합니다.
 - <https://spark.apache.org/docs/latest/sql-migration-guide.html#upgrading-from-spark-sql-22-to-23>

예를 들어 `percentile_approx` 함수 또는 기존 `SparkContext`가 있는 경우 `SparkSession`와 `SparkSession.builder.getOrCreate()`이 있습니다.

- <https://spark.apache.org/docs/latest/sql-migration-guide.html#upgrading-from-spark-sql-23-to-24>

예를 들어 `array_contains` 함수 또는 `spark.sql.caseSensitive=true`가 있는 `CURRENT_DATE`, `CURRENT_TIMESTAMP` 함수입니다.

- 작업의 추가 jar가 Glue 3.0에서 충돌합니까?
 - AWS Glue 0.9/1.0부터: 기존 AWS Glue 0.9/1.0 작업에 제공된 추가 jar는 Glue 3.0에서 사용할 수 있는 업그레이드된 종속성이나 새로운 종속성으로 인해 클래스 경로 충돌을 일으킬 수 있습니다. AWS Glue 3.0에서 `--user-jars-first` AWS Glue 작업 파라미터를 사용하거나 종속성을 셰이딩하여 클래스 경로 충돌을 피할 수 있습니다.
 - AWS Glue 2.0부터: AWS Glue 3.0에서 `--user-jars-first` AWS Glue 작업 파라미터를 사용하거나 종속성을 셰이딩하여 클래스 경로 충돌을 피할 수 있습니다.
- 작업이 Scala 2.11에 종속되나요?
 - AWS Glue 3.0은 Scala 2.12를 사용하므로 라이브러리가 Scala 2.11에 종속되는 경우 Scala 2.12로 라이브러리를 다시 구축해야 합니다.
- 작업의 외부 Python 라이브러리가 Python 2.7/3.6에 종속되나요?
 - Python 라이브러리 경로에 `egg/wheel/zip` 파일을 설정하는 대신 `--additional-python-modules` 파라미터를 사용합니다.
 - Spark 3.1.1에서 Python 2.7 지원이 제거되었으므로 종속 라이브러리를 Python 2.7/3.6에서 Python 3.7로 업데이트합니다.

AWS Glue 0.9에서 AWS Glue 3.0으로 마이그레이션

마이그레이션할 때 다음 변경 사항에 유의합니다.

- AWS Glue 0.9은 오픈 소스 Spark 2.2.1을 사용하고 AWS Glue 3.0은 EMR 최적화 Spark 3.1.1을 사용합니다.
- 몇 가지 Spark 변경만으로도 제거된 기능이 참조되지 않도록 스크립트를 수정해야 할 수 있습니다.
- 예를 들어 Spark 3.1.1은 Scala 유형이 지정되지 않은 UDF를 사용하지 않지만 Spark 2.2는 이를 허용합니다.

- AWS Glue 3.0의 모든 작업은 시작 시간이 크게 개선되어 실행됩니다. Spark 작업은 시작 대기 시간이 최대 10분에서 1분으로 단축되어 최소 청구 기간이 10배 더 짧으며 1초 단위로 청구됩니다.
- AWS Glue 2.0부터 로깅 동작이 변경되었습니다.
- [부록 A: 중요한 종속성 업그레이드](#)에서 강조 표시된 여러 종속성 업데이트
- Scala도 2.11에서 2.12로 업데이트되었으며 Scala 2.12는 Scala 2.11과 역호환되지 않습니다.
- Python 3.7은 Python 스크립트에 사용되는 기본 버전이기도 합니다. AWS Glue 0.9는 Python 2만 사용했기 때문입니다.
 - Python 2.7은 Spark 3.1.1에서 지원되지 않습니다.
 - 추가 Python 모듈을 설치하는 새로운 메커니즘을 사용할 수 있습니다.
- AWS Glue 3.0은 Apache YARN에서 실행되지 않으므로 YARN 설정이 적용되지 않습니다.
- AWS Glue 3.0에는 Hadoop 분산 파일 시스템(HDFS)이 없습니다.
- 기존 AWS Glue 0.9 작업에 제공된 추가 jar는 0.9에서 3.0의 여러 종속성에서 업그레이드가 있었기 때문에 종속성 충돌을 일으킬 수 있습니다. AWS Glue 3.0에서 --user-jars-first AWS Glue 작업 파라미터를 사용하여 클래스 경로 충돌을 피할 수 있습니다.
- AWS Glue 3.0은 동적 할당을 사용하지 않으므로 ExecutorAllocationManager 지표를 사용할 수 없습니다.
- AWS Glue 버전 3.0 작업에서는 작업자 수와 작업자 유형을 지정하지만 maxCapacity를 지정하지 않습니다.
- AWS Glue 3.0은 아직 기계 학습 변환을 지원하지 않습니다.
- AWS Glue 3.0은 아직 개발 엔드포인트를 지원하지 않습니다.

Spark 마이그레이션 설명서를 참조하세요.

- [Spark SQL 2.2에서 2.3으로 업그레이드](#) 참조
- [Spark SQL 2.3에서 2.4로 업그레이드](#) 참조
- [Spark SQL 2.4에서 3.0으로 업그레이드](#) 참조
- [Spark SQL 3.0에서 3.1으로 업그레이드](#) 참조
- [Spark 3.0 이후 예상되는 Datetime 동작의 변경 사항](#) 참조

AWS Glue 1.0에서 AWS Glue 3.0으로 마이그레이션

마이그레이션할 때 다음 변경 사항에 유의합니다.

- AWS Glue 1.0은 오픈 소스 Spark 2.4를 사용하고 AWS Glue 3.0은 EMR 최적화 Spark 3.1.1을 사용합니다.
- 몇 가지 Spark 변경만으로도 제거된 기능이 참조되지 않도록 스크립트를 수정해야 할 수 있습니다.
- 예를 들어 Spark 3.1.1은 Scala 유형이 지정되지 않은 UDF를 사용하지 않지만 Spark 2.4는 이를 허용합니다.
- AWS Glue 3.0의 모든 작업은 시작 시간이 크게 개선되어 실행됩니다. Spark 작업은 시작 대기 시간이 최대 10분에서 1분으로 단축되어 최소 청구 기간이 10배 더 짧으며 1초 단위로 청구됩니다.
- AWS Glue 2.0부터 로깅 동작이 변경되었습니다.
- 에서 강조 표시된 여러 종속성 업데이트
- Scala도 2.11에서 2.12로 업데이트되었으며 Scala 2.12는 Scala 2.11과 역호환되지 않습니다.
- Python 3.7은 Python 스크립트에 사용되는 기본 버전이기도 합니다. AWS Glue 0.9는 Python 2만 사용했기 때문입니다.
 - Python 2.7은 Spark 3.1.1에서 지원되지 않습니다.
 - 추가 Python 모듈을 설치하는 새로운 메커니즘을 사용할 수 있습니다.
- AWS Glue 3.0은 Apache YARN에서 실행되지 않으므로 YARN 설정이 적용되지 않습니다.
- AWS Glue 3.0에는 Hadoop 분산 파일 시스템(HDFS)이 없습니다.
- 기존 AWS Glue 1.0 작업에 제공된 추가 jar는 1.0에서 3.0의 여러 종속성에서 업그레이드가 있었기 때문에 종속성 충돌을 일으킬 수 있습니다. AWS Glue 3.0에서 `--user-jars-first` AWS Glue 작업 파라미터를 사용하여 클래스 경로 충돌을 피할 수 있습니다.
- AWS Glue 3.0은 동적 할당을 사용하지 않으므로 `ExecutorAllocationManager` 지표를 사용할 수 없습니다.
- AWS Glue 버전 3.0 작업에서는 작업자 수와 작업자 유형을 지정하지만 `maxCapacity`를 지정하지 않습니다.
- AWS Glue 3.0은 아직 기계 학습 변환을 지원하지 않습니다.
- AWS Glue 3.0은 아직 개발 엔드포인트를 지원하지 않습니다.

Spark 마이그레이션 설명서를 참조하세요.

- [Spark SQL 2.4에서 3.0으로 업그레이드](#) 참조
- [Spark 3.0 이후 예상되는 Datetime 동작의 변경 사항](#) 참조

AWS Glue 2.0에서 AWS Glue 3.0으로 마이그레이션

마이그레이션할 때 다음 변경 사항에 유의합니다.

- AWS Glue 2.0에 존재하는 모든 기존 작업 파라미터와 주요 기능은 AWS Glue 3.0에 존재합니다.
 - AWS Glue 3.0에서는 Amazon S3에 Parquet 데이터를 쓰기 위한 EMRFS S3 최적화 커미터가 기본값으로 사용됩니다. 그러나 `--enable-s3-parquet-optimized-commmitter`를 `false`로 설정하여 사용하지 않을 수도 있습니다.
- AWS Glue 2.0은 오픈 소스 Spark 2.4를 사용하고 AWS Glue 3.0은 EMR 최적화 Spark 3.1.1을 사용합니다.
 - 몇 가지 Spark 변경만으로도 제거된 기능이 참조되지 않도록 스크립트를 수정해야 할 수 있습니다.
 - 예를 들어 Spark 3.1.1은 Scala 유형이 지정되지 않은 UDF를 사용하지 않지만 Spark 2.4는 이를 허용합니다.
- AWS Glue 3.0에는 EMRFS 업데이트, JDBC 드라이버 업데이트, AWS Glue에서 제공하는 Spark 자체에 대한 추가 최적화 기능도 포함되어 있습니다.
- AWS Glue 3.0의 모든 작업은 시작 시간이 크게 개선되어 실행됩니다. Spark 작업은 시작 대기 시간이 최대 10분에서 1분으로 단축되어 최소 청구 기간이 10배 더 짧으며 1초 단위로 청구됩니다.
- Python 2.7은 Spark 3.1.1에서 지원되지 않습니다.
- [부록 A: 중요한 종속성 업그레이드](#)에서 강조 표시된 여러 종속성 업데이트
- Scala도 2.11에서 2.12로 업데이트되었으며 Scala 2.12는 Scala 2.11과 역호환되지 않습니다.
- 기존 AWS Glue 2.0 작업에 제공된 추가 jar는 2.0에서 3.0의 여러 종속성에서 업그레이드가 있었기 때문에 종속성 충돌을 일으킬 수 있습니다. AWS Glue 3.0에서 `--user-jars-first` AWS Glue 작업 파라미터를 사용하여 클래스 경로 충돌을 피할 수 있습니다.
- AWS Glue 3.0에서는 드라이버/실행기 구성에 대한 Spark 태스크 병렬 처리가 AWS Glue 2.0과는 다른 방식으로 수행되며 성능이 향상되고 사용 가능한 리소스가 더 효율적으로 활용됩니다. AWS Glue 3.0에서는 `spark.driver.cores`와 `spark.executor.cores`가 모두 코어 수로 구성됩니다(표준 및 G.1X 작업자의 경우 4개, G.2X 작업자의 경우 8개). 이러한 구성은 AWS Glue 작업의 작업자 유형이나 하드웨어를 변경하지 않습니다. 이 구성을 사용하여 Spark 애플리케이션의 Spark 태스크 병렬 처리에 맞는 파티션 또는 분할 수를 계산할 수 있습니다.

일반적으로 작업 성능은 AWS Glue 2.0과 비슷하거나 개선됩니다. 작업 실행 속도가 느린 경우 다음 작업 인수를 전달하여 작업 병렬 처리 성능을 높일 수 있습니다.

- key: `--executor-cores` value: *<desired number of tasks that can run in parallel>*

- 값은 작업자 유형에 있는 vCPU 수의 2배(G.1X의 경우 8, G.2X의 경우 16, G.4X의 경우 32, G.8X의 경우 64)를 초과해서는 안 됩니다. 이 구성을 업데이트할 때는 주의해야 합니다. 병렬 처리 성능을 높이면 메모리 및 디스크 부담이 커질 뿐 아니라 소스 및 대상 시스템에서 제한이 발생할 수 있으므로 작업 성능에 영향을 미칠 수 있습니다.
- AWS Glue 3.0에서는 Spark 3.1을 사용하므로 parquet 파일에서 타임스탬프를 로드/저장하는 동작이 변경됩니다. 자세한 내용은 [Spark SQL 3.0에서 3.1로 업그레이드](#)를 참조하세요.

타임스탬프 열이 포함된 parquet 데이터를 읽거나 쓸 때는 다음 파라미터를 설정하는 것이 좋습니다. 이러한 파라미터를 설정하면 Spark 2에서 Spark 3으로 업그레이드하는 동안 AWS Glue 동적 프레임과 Spark 데이터 프레임 둘 다에서 발생하는 달력 비호환성 문제를 해결할 수 있습니다. datetime 값을 있는 그대로 읽으려면 CORRECTED 옵션을 사용하고, 읽는 동안 달력 차이를 기준으로 datetime 값을 다시 지정하려면 LEGACY 옵션을 사용합니다.

```
- Key: --conf
- Value: spark.sql.legacy.parquet.int96RebaseModeInRead=[CORRECTED|LEGACY] --conf spark.sql.legacy.parquet.int96RebaseModeInWrite=[CORRECTED|LEGACY] --conf spark.sql.legacy.parquet.datetimeRebaseModeInRead=[CORRECTED|LEGACY]
```

Spark 마이그레이션 설명서를 참조하세요.

- [Spark SQL 2.4에서 3.0으로 업그레이드](#) 참조
- [Spark 3.0 이후 예상되는 Datetime 동작의 변경 사항](#) 참조

부록 A: 중요한 종속성 업그레이드

다음은 종속성 업그레이드입니다.

종속성	AWS Glue 0.9 버전	AWS Glue 1.0 버전	AWS Glue 2.0 버전	AWS Glue 3.0 버전
Spark	2.2.1	2.4.3	2.4.3	3.1.1-amzn-0
Hadoop	2.7.3-amzn-6	2.8.5-amzn-1	2.8.5-amzn-5	3.2.1-amzn-3
Scala	2.11	2.11	2.11	2.12
Jackson	2.7.x	2.7.x	2.7.x	2.10.x

종속성	AWS Glue 0.9 버전	AWS Glue 1.0 버전	AWS Glue 2.0 버전	AWS Glue 3.0 버전
Hive	1.2	1.2	1.2	2.3.7-amzn-4
EMRFS	2.20.0	2.30.0	2.38.0	2.46.0
Json4s	3.2.x	3.5.x	3.5.x	3.6.6
화살표	N/A	0.10.0	0.10.0	2.0.0
AWS Glue 카탈로그 클라이언트	N/A	N/A	1.10.0	3.0.0

부록 B: JDBC 드라이버 업그레이드

다음은 JDBC 드라이버 업그레이드입니다.

드라이버	과거 AWS Glue 버전의 JDBC 드라이버 버전	AWS Glue 3.0의 JDBC 드라이버 버전
MySQL	5.1	8.0.23
Microsoft SQL Server	6.1.0	7.0.0
Oracle Database	11.2	21.1
PostgreSQL	42.1.0	42.2.18
MongoDB	2.0.0	4.0.0

AWS Glue 버전 4.0으로 AWS Glue for Spark 작업 마이그레이션

이 주제에서는 Spark 애플리케이션 및 ETL 작업을 AWS Glue 4.0으로 마이그레이션할 수 있도록 하는 AWS Glue 버전 0.9, 1.0, 2.0 및 3.0 간의 변경 사항에 대해 설명합니다. 또한 AWS Glue 4.0의 기능과 이를 사용할 때의 이점에 대해 설명합니다.

AWS Glue ETL 작업에 이 기능을 사용하려면 작업 생성 시 Glue version으로 **4.0**을 선택합니다.

주제

- [지원되는 새로운 기능](#)
- [AWS Glue 4.0으로 마이그레이션할 작업](#)
- [마이그레이션 체크리스트](#)
- [AWS Glue 3.0에서 AWS Glue 4.0으로 마이그레이션](#)
- [AWS Glue 2.0에서 AWS Glue 4.0으로 마이그레이션](#)
- [AWS Glue 1.0에서 AWS Glue 4.0으로 마이그레이션](#)
- [AWS Glue 0.9에서 AWS Glue 4.0으로 마이그레이션](#)
- [AWS Glue 4.0용 커넥터 및 JDBC 드라이버 마이그레이션](#)
- [부록 A: 중요한 종속성 업그레이드](#)
- [부록 B: JDBC 드라이버 업그레이드](#)
- [부록 C: 커넥터 업그레이드](#)

지원되는 새로운 기능

이 섹션에서는 AWS Glue 버전 4.0의 새로운 기능과 장점에 대해 설명합니다.

- Apache Spark 3.3.0을 기반으로 하지만 적응형 쿼리 실행, 벡터화된 리더, 최적화된 셔플, 파티션 병합과 같은 Amazon EMR 및 AWS Glue의 최적화를 포함합니다.
- MySQL, Microsoft SQL Server, Oracle, PostgreSQL, MongoDB 및 Spark 3.3.0에서 가져온 업그레이드된 Spark 라이브러리 및 종속성을 포함한 모든 AWS Glue 기본 소스용 JDBC 드라이버가 업그레이드되었습니다.
- 새로운 Amazon Redshift 커넥터와 JDBC 드라이버로 업데이트.
- 업그레이드된 EMR 파일 시스템(EMRFS)으로 Amazon S3 액세스가 최적화되었으며 Amazon S3 최적화 출력 커미터가 기본적으로 사용됩니다.
- 파티션 인덱스, 푸시다운 조건자, 파티션 목록 및 업그레이드된 Hive 메타스토어 클라이언트로 Data Catalog 액세스가 최적화되었습니다.
- 셀 수준 필터링 및 데이터 레이크 트랜잭션이 있는 관리되는 카탈로그 테이블을 위해 Lake Formation과 통합됩니다.
- 시작 대기 시간이 단축되어 전반적인 작업 완료 시간과 상호 작용성이 개선되었습니다.
- Spark 작업은 최소 10분에서 1분으로 10배 더 짧은 최소 청구 기간으로 1초 단위로 청구됩니다.
- Apache Hudi, Delta Lake 및 Apache Iceberg를 통해 개방형 데이터 레이크 프레임워크 기본 지원.

- Amazon S3를 사용하여 셔플링 및 탄력적인 스토리지 용량을 지원할 수 있도록 Amazon S3 기반 클라우드 셔플 스토리지 플러그인(Apache Spark 플러그인) 기본 지원.

Spark 3.1.1에서 Spark 3.3.0으로 개선된 주요 기능

다음과 같은 개선 사항에 유의합니다.

- 행 수준 런타임 필터링([SPARK-32268](#)).
- ANSI 개선 사항([SPARK-38860](#)).
- 오류 메시지 개선 사항([SPARK-38781](#)).
- Parquet 벡터화된 리더([SPARK-34863](#))에 대해 복합 유형을 지원합니다.
- Spark SQL([SPARK-37273](#))에 대해 숨김 파일 메타데이터를 지원합니다.
- Python/Pandas UDF([SPARK-37443](#))에 대한 프로파일러를 제공합니다.
- 여러 배치에서 Trigger.Once과 같은 스트리밍 쿼리를 실행하기 위해 Trigger.AvailableNow를 소개합니다([SPARK-36533](#)).
- 보다 포괄적인 Datasource V2 푸시다운 기능([SPARK-38788](#)).
- log4j 1에서 log4j 2([SPARK-37814](#))로 마이그레이션합니다.

기타 주요 변경 사항

다음과 같은 변경 사항에 유의하세요.

- 호환성에 영향을 미치는 변경
 - 문서 및 Python/문서([SPARK-36977](#))에서 Python 3.6 지원에 대한 참조를 삭제합니다.
 - 기본 제공 pickle을 cloudpickle([SPARK-32079](#))로 대체하여 명명된 tuple hack을 제거합니다.
 - 최소 pandas 버전을 1.0.5([SPARK-37465](#))로 올립니다.

AWS Glue 4.0으로 마이그레이션할 작업

기존 작업의 경우 작업 구성에서 Glue version을 이전 버전에서 Glue 4.0으로 변경합니다.

- AWS Glue Studio의 Glue version에서 Glue 4.0 - Supports Spark 3.3, Scala 2, Python 3을 선택합니다.
- API에서 [UpdateJob](#) API 작업의 GlueVersion 파라미터에서 **4.0**을 선택합니다.

새 작업의 경우 작업을 생성할 때 Glue 4.0을 선택합니다.

- 콘솔의 Glue version에서 Spark 3.3, Python 3 (Glue Version 4.0) or Spark 3.3, Scala 2 (Glue Version 3.0)를 선택합니다.
- AWS Glue Studio의 Glue version에서 Glue 4.0 - Supports Spark 3.3, Scala 2, Python 3을 선택합니다.
- API에서 [CreateJob](#) API 작업의 GlueVersion 파라미터에서 4.0을 선택합니다.

AWS Glue 2.0 이전 버전에서 가져온 AWS Glue 4.0의 Spark 이벤트 로그를 보려면 [AWS CloudFormation 또는 Docker를 사용하여 AWS Glue 4.0용으로 업그레이드된 Spark 기록 서버를 시작합니다.](#)

마이그레이션 체크리스트

마이그레이션을 위해 이 체크리스트를 검토합니다.

Note

AWS Glue 3.0과 관련된 체크리스트 항목은 [마이그레이션 체크리스트](#) 섹션을 참조하세요.

- 작업의 외부 Python 라이브러리가 Python 2.7/3.6에 종속되나요?
 - Spark 3.3.0에서 Python 2.7 및 3.6 지원이 완전히 제거되었으므로 종속 라이브러리를 Python 2.7/3.6에서 Python 3.10으로 업데이트합니다.

AWS Glue 3.0에서 AWS Glue 4.0으로 마이그레이션

마이그레이션할 때 다음 변경 사항에 유의합니다.

- AWS Glue 3.0에 존재하는 모든 기존 작업 파라미터와 주요 기능은 AWS Glue 4.0에 존재합니다.
- AWS Glue 3.0은 Amazon EMR 최적화 Spark 3.1.1을 사용하고 AWS Glue 4.0은 Amazon EMR 최적화 Spark 3.3.0을 사용합니다.

몇 가지 Spark 변경만으로도 제거된 기능이 참조되지 않도록 스크립트를 수정해야 할 수 있습니다.

- AWS Glue 4.0에는 EMRFS와 Hadoop에 대한 업데이트도 제공됩니다. 특정 버전 번호는 [부록 A: 중요한 종속성 업그레이드](#) 섹션을 참조하세요.
- ETL 작업에 제공되는 AWS SDK가 이제 1.11에서 1.12로 업그레이드되었습니다.

- 모든 Python 작업에서는 Python 버전 3.10을 사용합니다. 이전에는 Python 3.7이 AWS Glue 3.0에서 사용되었습니다.

따라서 AWS Glue에서 기본적으로 가져오는 일부 pymodule이 업그레이드됩니다.

- Log4j가 Log4j2로 업그레이드되었습니다.
 - Log4j2 마이그레이션 경로에 대한 자세한 내용은 [Log4j 설명서](#)를 참조하세요.
 - 대신 사용자 지정 log4j.properties 파일의 이름을 적절한 log4j2 속성을 사용하여 log4j2.properties 파일로 바꿔야 합니다.
- 특정 커넥터를 마이그레이션하려면 [AWS Glue 4.0용 커넥터 및 JDBC 드라이버 마이그레이션](#) 섹션을 참조하세요.
- AWS 암호화 SDK가 1.x에서 2.x로 업그레이드되었습니다. AWS Glue보안 구성을 사용하는 AWS Glue 작업과 런타임에 제공된 AWS 암호화 SDK 종속성에 종속되는 작업이 영향을 받습니다. AWS Glue 작업 마이그레이션 지침을 참조하세요.

AWS Glue 2.0/3.0에는 이미 AWS 암호화 SDK 브리지 버전이 포함되어 있으므로 AWS Glue 2.0/3.0 작업을 AWS Glue 4.0 작업으로 안전하게 업그레이드할 수 있습니다.

Spark 마이그레이션 설명서를 참조하세요.

- [Spark SQL 3.1에서 3.2로 업그레이드](#)
- [Spark SQL 3.2에서 3.3으로 업그레이드](#)

AWS Glue 2.0에서 AWS Glue 4.0으로 마이그레이션

마이그레이션할 때 다음 변경 사항에 유의합니다.

Note

AWS Glue 3.0과 관련된 마이그레이션 단계는 [AWS Glue 3.0에서 AWS Glue 4.0으로 마이그레이션](#) 섹션을 참조하세요.

- AWS Glue 2.0에 존재하는 모든 기존 작업 파라미터와 주요 기능은 AWS Glue 4.0에 존재합니다.
- AWS Glue 3.0 이상에서는 Amazon S3에 Parquet 데이터를 쓰기 위한 EMRFS S3 최적화 커미터가 기본값으로 사용됩니다. 그러나 `--enable-s3-parquet-optimized-commmitter`를 `false`로 설정하여 사용하지 않을 수도 있습니다.

- AWS Glue 2.0은 오픈 소스 Spark 2.4를 사용하고 AWS Glue 4.0은 Amazon EMR 최적화 Spark 3.3.0을 사용합니다.
- 몇 가지 Spark 변경만으로도 제거된 기능이 참조되지 않도록 스크립트를 수정해야 할 수 있습니다.
- 예를 들어 Spark 3.3.0은 Scala 유형이 지정되지 않은 UDF를 사용하지 않지만 Spark 2.4는 이를 허용합니다.
- ETL 작업에 제공되는 AWS SDK가 이제 1.11에서 1.12로 업그레이드되었습니다.
- AWS Glue 4.0에는 EMRFS 업데이트, JDBC 드라이버 업데이트, AWS Glue에서 제공하는 Spark 자체에 대한 추가 최적화 기능도 포함되어 있습니다.
- Scala는 2.11에서 2.12로 업데이트되었으며 Scala 2.12는 Scala 2.11과 역호환되지 않습니다.
- Python 3.10은 Python 스크립트에 사용되는 기본 버전입니다. AWS Glue 2.0은 Python 3.7 및 2.7만 사용했기 때문입니다.
- Python 2.7은 Spark 3.3.0에서 지원되지 않습니다. 작업 구성에서 Python 2를 요청하는 작업은 `IllegalArgument`Exception과 함께 실패합니다.
- AWS Glue 2.0 이상에서는 추가 Python 모듈을 설치하는 새로운 메커니즘을 사용할 수 있습니다.
- [부록 A: 중요한 종속성 업그레이드](#)에서 강조 표시된 여러 종속성 업데이트
- 기존 AWS Glue 2.0 작업에 제공된 추가 JAR 파일은 2.0에서 4.0의 여러 종속성에서 업그레이드가 있었기 때문에 종속성 충돌을 일으킬 수 있습니다. AWS Glue 4.0에서 `--user-jars-first` AWS Glue 작업 파라미터를 사용하여 클래스 경로 충돌을 피할 수 있습니다.
- AWS Glue 4.0은 Spark 3.3을 사용합니다. Spark 3.1부터 parquet 파일에서 타임스탬프를 로드하거나 해당 파일로 저장하는 동작이 변경되었습니다. 자세한 내용은 [Spark SQL 3.0에서 3.1로 업그레이드](#)를 참조하세요.

타임스탬프 열이 포함된 parquet 데이터를 읽거나 쓸 때는 다음 파라미터를 설정하는 것이 좋습니다. 이러한 파라미터를 설정하면 Spark 2에서 Spark 3으로 업그레이드하는 동안 AWS Glue 동적 프레임과 Spark 데이터 프레임 둘 다에서 발생하는 달력 비호환성 문제를 해결할 수 있습니다. `datetime` 값을 있는 그대로 읽으려면 `CORRECTED` 옵션을 사용하고, 읽는 동안 달력 차이를 기준으로 `datetime` 값을 다시 지정하려면 `LEGACY` 옵션을 사용합니다.

```
- Key: --conf
- Value: spark.sql.legacy.parquet.int96RebaseModeInRead=[CORRECTED|LEGACY] --conf spark.sql.legacy.parquet.int96RebaseModeInWrite=[CORRECTED|LEGACY] --conf spark.sql.legacy.parquet.datetimeRebaseModeInRead=[CORRECTED|LEGACY]
```

- 특정 커넥터를 마이그레이션하려면 [AWS Glue 4.0용 커넥터 및 JDBC 드라이버 마이그레이션](#) 섹션을 참조하세요.

- AWS 암호화 SDK가 1.x에서 2.x로 업그레이드되었습니다. AWS Glue보안 구성을 사용하는 AWS Glue 작업과 런타임에 제공된 AWS 암호화 SDK 종속성에 종속되는 작업이 영향을 받습니다. AWS Glue 작업 마이그레이션 지침을 참조하세요.
- AWS Glue 2.0에는 이미 AWS 암호화 SDK 브리지 버전이 포함되어 있으므로 AWS Glue 2.0 작업을 AWS Glue 4.0 작업으로 안전하게 업그레이드할 수 있습니다.

Spark 마이그레이션 설명서를 참조하세요.

- [Spark SQL 2.4에서 3.0으로 업그레이드](#)
- [Spark SQL 3.1에서 3.2로 업그레이드](#)
- [Spark SQL 3.2에서 3.3으로 업그레이드](#)
- [Spark 3.0 이후 예상되는 Datetime 동작의 변경 사항](#)

AWS Glue 1.0에서 AWS Glue 4.0으로 마이그레이션

마이그레이션할 때 다음 변경 사항에 유의합니다.

- AWS Glue 1.0은 오픈 소스 Spark 2.4를 사용하고 AWS Glue 4.0은 Amazon EMR 최적화 Spark 3.3.0을 사용합니다.
 - 몇 가지 Spark 변경만으로도 제거된 기능이 참조되지 않도록 스크립트를 수정해야 할 수 있습니다.
 - 예를 들어 Spark 3.3.0은 Scala 유형이 지정되지 않은 UDF를 사용하지 않지만 Spark 2.4는 이를 허용합니다.
- AWS Glue 4.0의 모든 작업은 시작 시간이 크게 개선되어 실행됩니다. Spark 작업은 시작 대기 시간이 최대 10분에서 1분으로 단축되어 최소 청구 기간이 10배 더 짧으며 1초 단위로 청구됩니다.
- AWS Glue 4.0에서는 로깅 동작이 크게 변경되었습니다. Spark 3.3.0의 최소 요구 사항은 Log4j2입니다.
- 부록에서 강조 표시된 여러 종속성 업데이트.
- Scala도 2.11에서 2.12로 업데이트되었으며 Scala 2.12는 Scala 2.11과 역호환되지 않습니다.
- Python 3.10은 Python 스크립트에 사용되는 기본 버전이기도 합니다. AWS Glue 0.9는 Python 2만 사용했기 때문입니다.

Python 2.7은 Spark 3.3.0에서 지원되지 않습니다. 작업 구성에서 Python 2를 요청하는 작업은 `IllegalArgumentExpection`과 함께 실패합니다.

- AWS Glue 2.0 이상에서는 pip를 통해 추가 Python 모듈을 설치하는 새로운 메커니즘을 사용할 수 있습니다. 자세한 내용은 [AWS Glue 2.0+에서 pip를 사용하여 추가 Python 모듈 설치](#)를 참조하세요.
- AWS Glue 4.0은 Apache YARN에서 실행되지 않으므로 YARN 설정이 적용되지 않습니다.
- AWS Glue 4.0에는 Hadoop 분산 파일 시스템(HDFS)이 없습니다.
- 기존 AWS Glue 1.0 작업에 제공된 추가 JAR 파일은 1.0에서 4.0의 여러 종속성에서 업그레이드가 있었기 때문에 종속성 충돌을 일으킬 수 있습니다. 이 문제를 방지하기 위해 기본적으로 `--user-jars-first` AWS Glue 작업 파라미터를 사용하여 AWS Glue 4.0을 활성화합니다.
- AWS Glue 4.0은 Auto Scaling을 지원합니다. 따라서 auto Scaling이 활성화된 경우 `ExecutorAllocationManager` 지표를 사용할 수 있습니다.
- AWS Glue 버전 4.0 작업에서는 작업자 수와 작업자 유형을 지정하지만 `maxCapacity`를 지정하지 않습니다.
- AWS Glue 4.0은 아직 기계 학습 변환을 지원하지 않습니다.
- 특정 커넥터를 마이그레이션하려면 [AWS Glue 4.0용 커넥터 및 JDBC 드라이버 마이그레이션](#) 섹션을 참조하세요.
- AWS 암호화 SDK가 1.x에서 2.x로 업그레이드되었습니다. AWS Glue보안 구성을 사용하는 AWS Glue 작업과 런타임에 제공된 AWS 암호화 SDK 종속성에 종속되는 작업이 영향을 받습니다. AWS Glue 작업 마이그레이션 지침을 참조하세요.
 - AWS Glue 0.9/1.0 작업을 AWS Glue 4.0 작업으로 직접 마이그레이션할 수 없습니다. 이는 버전 2.x 이상으로 직접 업그레이드하고 모든 새 기능을 즉시 활성화하면 AWS Encryption SDK가 이전 버전의 AWS Encryption SDK에서 암호화된 사이버텍스트를 해독할 수 없기 때문입니다.
 - 안전하게 업그레이드하려면 먼저 AWS Encryption SDK 브리지 버전이 포함된 AWS Glue 2.0/3.0 작업으로 마이그레이션하는 것이 좋습니다. 작업을 한 번 실행하여 AWS Encryption SDK 브리지 버전을 활용합니다.
 - 완료되면 AWS Glue 2.0/3.0 작업을 AWS Glue 4.0으로 안전하게 마이그레이션할 수 있습니다.

Spark 마이그레이션 설명서를 참조하세요.

- [Spark SQL 2.4에서 3.0으로 업그레이드](#)
- [Spark SQL 3.0에서 3.1로 업그레이드](#)
- [Spark SQL 3.1에서 3.2로 업그레이드](#)
- [Spark SQL 3.2에서 3.3으로 업그레이드](#)
- [Spark 3.0 이후 예상되는 Datetime 동작의 변경 사항](#)

AWS Glue 0.9에서 AWS Glue 4.0으로 마이그레이션

마이그레이션할 때 다음 변경 사항에 유의합니다.

- AWS Glue 0.9는 오픈 소스 Spark 2.2.1을 사용하고 AWS Glue 4.0은 Amazon EMR 최적화 Spark 3.3.0을 사용합니다.
 - 몇 가지 Spark 변경만으로도 제거된 기능이 참조되지 않도록 스크립트를 수정해야 할 수 있습니다.
 - 예를 들어 Spark 3.3.0은 Scala 유형이 지정되지 않은 UDF를 사용하지 않지만 Spark 2.2는 이를 허용합니다.
- AWS Glue 4.0의 모든 작업은 시작 시간이 크게 개선되어 실행됩니다. Spark 작업은 시작 대기 시간이 최대 10분에서 1분으로 단축되어 최소 청구 기간이 10배 더 짧으며 1초 단위로 청구됩니다.
- AWS Glue 4.0 이후 로깅 동작이 크게 변경되었습니다. Spark 3.3.0에는 여기서(<https://spark.apache.org/docs/latest/core-migration-guide.html#upgrading-from-core-32-to-33>) 언급한 것처럼 Log4j2의 최소 요구 사항이 있습니다.
- 부록에서 강조 표시된 여러 종속성 업데이트
- Scala도 2.11에서 2.12로 업데이트되었으며 Scala 2.12는 Scala 2.11과 역호환되지 않습니다.
- Python 3.10은 Python 스크립트에 사용되는 기본 버전이기도 합니다. AWS Glue 0.9는 Python 2만 사용했기 때문입니다.
 - Python 2.7은 Spark 3.3.0에서 지원되지 않습니다. 작업 구성에서 Python 2를 요청하는 작업은 `IllegalArgumentExpection`과 함께 실패합니다.
 - pip를 통해 추가 Python 모듈을 설치하는 새로운 메커니즘을 사용할 수 있습니다.
- AWS Glue 4.0은 Apache YARN에서 실행되지 않으므로 YARN 설정이 적용되지 않습니다.
- AWS Glue 4.0에는 Hadoop 분산 파일 시스템(HDFS)이 없습니다.
- 기존 AWS Glue 0.9 작업에 제공된 추가 JAR 파일은 0.9에서 3.0의 여러 종속성에서 업그레이드가 있었기 때문에 종속성 충돌을 일으킬 수 있습니다. AWS Glue 3.0에서 `--user-jars-first` AWS Glue 작업 파라미터를 사용하여 클래스 경로 충돌을 피할 수 있습니다.
- AWS Glue 4.0은 Auto Scaling을 지원합니다. 따라서 auto Scaling이 활성화된 경우 `ExecutorAllocationManager` 지표를 사용할 수 있습니다.
- AWS Glue 버전 4.0 작업에서는 작업자 수와 작업자 유형을 지정하지만 `maxCapacity`를 지정하지 않습니다.
- AWS Glue 4.0은 아직 기계 학습 변환을 지원하지 않습니다.
- 특정 커넥터를 마이그레이션하려면 [AWS Glue 4.0용 커넥터 및 JDBC 드라이버 마이그레이션](#) 섹션을 참조하세요.

- AWS 암호화 SDK가 1.x에서 2.x로 업그레이드되었습니다. AWS Glue보안 구성을 사용하는 AWS Glue 작업과 런타임에 제공된 AWS 암호화 SDK 종속성에 종속되는 작업이 영향을 받습니다. AWS Glue 작업 마이그레이션 지침을 참조하세요.
- AWS Glue 0.9/1.0 작업을 AWS Glue 4.0 작업으로 직접 마이그레이션할 수 없습니다. 이는 버전 2.x 이상으로 직접 업그레이드하고 모든 새 기능을 즉시 활성화하면 AWS Encryption SDK가 이전 버전의 AWS Encryption SDK에서 암호화된 사이버텍스트를 해독할 수 없기 때문입니다.
- 안전하게 업그레이드하려면 먼저 AWS Encryption SDK 브리지 버전이 포함된 AWS Glue 2.0/3.0 작업으로 마이그레이션하는 것이 좋습니다. 작업을 한 번 실행하여 AWS Encryption SDK 브리지 버전을 활용합니다.
- 완료되면 AWS Glue 2.0/3.0 작업을 AWS Glue 4.0으로 안전하게 마이그레이션할 수 있습니다.

Spark 마이그레이션 설명서를 참조하세요.

- [Spark SQL 2.2에서 2.3으로 업그레이드](#)
- [Spark SQL 2.3에서 2.4로 업그레이드](#)
- [Spark SQL 2.4에서 3.0으로 업그레이드](#)
- [Spark SQL 3.0에서 3.1로 업그레이드](#)
- [Spark SQL 3.1에서 3.2로 업그레이드](#)
- [Spark SQL 3.2에서 3.3으로 업그레이드](#)
- [Spark 3.0 이후 예상되는 Datetime 동작의 변경 사항](#)

AWS Glue 4.0용 커넥터 및 JDBC 드라이버 마이그레이션

업그레이드된 JDBC 및 데이터 레이크 커넥터 버전은 다음을 참조하세요.

- [부록 B: JDBC 드라이버 업그레이드](#)
- [부록 C: 커넥터 업그레이드](#)

Hudi

- Spark SQL 지원 개선 사항:
 - Call Procedure 명령을 통해 업그레이드, 다운그레이드, 부트스트랩, 정리 및 복구에 대한 지원이 추가되었습니다. Spark SQL에서는 Create/Drop/Show/Refresh Index 구문이 가능합니다.

- Spark SQL과 달리 Spark DataSource를 통한 사용 간 성능 격차가 좁혀졌습니다. 과거에는 데이터 소스 쓰기가 SQL보다 빨랐습니다.
- 모든 기본 제공 키 생성기는 성능이 우수한 Spark 전용 API 작업을 구현합니다.
- 대량 insert 작업에서 UDF 변환을 RDD 변환으로 대체하여 SerDe 사용 비용을 줄였습니다.
- Hudi에서 Spark SQL을 사용하려면 SQL 문에서 primaryKey를 tblproperties 또는 옵션으로 지정해야 합니다. 업데이트 및 삭제 작업의 경우에도 preCombineField가 필요합니다.
- primaryKey 없이 버전 0.10.0 이전에 생성한 Hudi 테이블은 버전 0.10.0 이후의 primaryKey 필드를 사용하여 다시 생성해야 합니다.

PostgreSQL

- 여러 취약점(CVE)이 해결되었습니다.
- Java 8은 기본적으로 지원됩니다.
- 작업에서 바이트 배열을 제외한 배열 배열을 사용하는 경우 이 시나리오를 다차원 배열로 간주할 수 있습니다.

MongoDB

- 현재 MongoDB 커넥터는 Spark 버전 3.1 이상과 MongoDB 버전 4.0 이상을 지원합니다.
- 커넥터 업그레이드로 인해 몇 가지 속성 이름이 변경되었습니다. 예를 들어 URI 속성 이름이 `connection.uri`로 변경되었습니다. 현재 옵션에 대한 자세한 내용은 [MongoDB Spark 커넥터 블로그](#)를 참조하세요.
- Amazon DocumentDB에서 호스팅하는 MongoDB 4.0을 사용하면 기능상 몇 가지 차이점이 있습니다. 자세한 내용은 다음 주제를 참조하십시오.
 - [기능적 차이: Amazon DocumentDB 및 MongoDB](#)
 - [지원되는 MongoDB API, 작업 및 데이터 형식](#)
- “파티셔너” 옵션은 ShardedPartitioner, PaginateIntoPartitionsPartitioner 및 SinglePartitionPartitioner로 제한됩니다. 스테이지 오퍼레이터가 MongoDB API를 지원하지 않기 때문에 Amazon DocumentDB에는 기본값 SamplePartitioner 및 PaginateBySizePartitioner를 사용할 수 없습니다. 자세한 내용은 [지원되는 MongoDB API, 작업 및 데이터 형식](#)을 참조하세요.

Delta Lake

- Delta Lake는 이제 [SQL에서 시간 여행](#)을 지원하여 오래된 데이터를 쉽게 쿼리할 수 있습니다. 이번 업데이트를 통해 이제 Spark SQL과 DataFrame API를 통해 시간 여행을 사용할 수 있습니다. SQL에서 현재 버전의 TIMESTAMP에 대한 Support 추가되었습니다.
- Spark 3.3에는 배치 쿼리를 위한 Trigger.Once와 동일한 방식으로 스트리밍 쿼리를 실행할 수 있는 [Trigger.AvailableNow](#)가 도입되었습니다. 이 지원은 Delta 테이블을 스트리밍 소스로 사용하는 경우에도 사용할 수 있습니다.
- 테이블의 열 목록을 반환하는 SHOW COLUMNS를 지원합니다.
- Scala 및 Python DeltaTable API에서 [DESCRIBE DETAIL](#)을 지원합니다. DeltaTable API 또는 Spark SQL을 사용하여 Delta 테이블에 대한 세부 정보를 검색합니다.
- SQL [Delete](#), [Merge](#) 및 [Update](#) 명령에서 작업 지표 반환을 지원합니다. 이전에는 이러한 SQL 명령이 빈 DataFrame을 반환했지만 이제 수행된 작업에 대한 유용한 지표가 포함된 DataFrame을 반환합니다.
- 성능 개선 최적화:
 - 여러 개의 작은 파일을 압축할 때 성능을 높이려면 Optimize 명령에서 coalesce(1) 대신 repartition(1)을 사용하도록 spark.databricks.delta.optimize.repartition.enabled=true 구성 옵션을 설정합니다.
 - 큐 기반 접근 방식을 사용하여 압축 작업을 병렬화하여 [성능을 개선했습니다](#).
- 기타 주요 변경 사항:
 - VACUUM 및 OPTIMIZE SQL 명령에서 [변수 사용을 지원합니다](#).
 - 카탈로그 테이블을 통한 CONVERT TO DELTA 관련 개선 사항은 다음과 같습니다.
 - 제공되지 않은 경우 카탈로그에서 [파티션 스키마를 자동으로 채웁니다](#).
 - 전체 디렉터리 스캔을 수행하는 대신 카탈로그의 [파티션 정보를 사용하여](#) 커밋할 데이터 파일을 찾습니다. 테이블 디렉터리의 모든 데이터 파일을 커밋하는 대신 활성 파티션의 디렉터리에 있는 데이터 파일만 커밋됩니다.
 - DROP COLUMN 및 RENAME COLUMN을 사용하지 않은 경우 열 매핑이 가능한 테이블에 대한 [CDF\(Change Data Feed\) 배치 읽기를 지원합니다](#). 자세한 내용은 [Delta Lake 설명서](#)를 참조하세요.
 - 첫 번째 통과에서 스키마 프루닝을 활성화하여 [Update 명령 성능을 개선합니다](#).

Apache Iceberg

- 스캔 계획 및 Spark 쿼리에 대한 몇 가지 [성능 개선 사항](#)이 추가되었습니다.
- 변경 기반 커밋을 사용하여 서비스 측의 커밋 충돌을 해결하는 일반 REST 카탈로그 클라이언트를 추가했습니다.
- AS OF SQL 시간 여행 쿼리 구문이 지원됩니다.
- MERGE 및 UPDATE 쿼리에 대한 읽기 중 병합 지원을 추가했습니다.
- Z축을 통한 파티션 재작성 지원을 추가했습니다.
- [Theta 스케치](#) 또는 블룸 필터와 같은 대형 통계 및 인덱스 blob을 위한 형식인 Puffin의 사양 및 구현을 추가했습니다.
- 데이터를 점진적으로 소비하기 위한 새 인터페이스를 추가했습니다(추가 및 변경 로그 스캔 모두).
- 대량 작업 및 FileIO 인터페이스에 대한 범위 읽기 지원이 추가되었습니다.
- 메타데이터 트리에서 삭제 파일을 표시하는 메타데이터 테이블을 더 추가했습니다.
- 드롭 테이블 동작이 변경되었습니다. Iceberg 0.13.1에서 DROP TABLE을 실행하면 카탈로그에서 테이블이 제거되고 테이블 내용도 삭제됩니다. Iceberg 1.0.0에서 DROP TABLE은 카탈로그에서만 테이블을 제거합니다. 테이블 내용을 삭제하려면 DROP TABLE PURGE를 사용합니다.
- Parquet 벡터화된 읽기는 Iceberg 1.0.0에서 기본적으로 활성화됩니다. 벡터화된 읽기를 비활성화하려면 `read.parquet.vectorization.enabled`를 `false`로 설정합니다.

Oracle

변경 사항은 미미합니다.

MySQL

변경 사항은 미미합니다.

Amazon Redshift

AWS Glue 4.0에는 새로운 JDBC 드라이버가 포함된 새로운 Amazon Redshift 커넥터가 있습니다. 향상된 기능 및 이전 AWS Glue 버전에서 마이그레이션하는 방법에 대한 자세한 내용은 [the section called “Redshift 연결”](#) 섹션을 참조하세요.

부록 A: 중요한 종속성 업그레이드

다음은 종속성 업그레이드입니다.

종속성	AWS Glue 4.0 버전	AWS Glue 3.0 버전	AWS Glue 2.0 버전	AWS Glue 1.0 버전
Spark	3.3.0-amzn-1	3.1.1-amzn-0	2.4.3	2.4.3
Hadoop	3.3.3-amzn-0	3.2.1-amzn-3	2.8.5-amzn-5	2.8.5-amzn-1
Scala	2.12	2.12	2.11	2.11
Jackson	2.13.3	2.10.x	2.7.x	2.7.x
Hive	2.3.9-amzn-2	2.3.7-amzn-4	1.2	1.2
EMRFS	2.54.0	2.46.0	2.38.0	2.30.0
Json4s	3.7.0-M11	3.6.6	3.5.x	3.5.x
화살표	7.0.0	2.0.0	0.10.0	0.10.0
AWS Glue 데이터 카탈로그 클라이언트	3.7.0	3.0.0	1.10.0	N/A
Python	3.10	3.7	2.7 및 3.6	2.7 및 3.6
Boto	1.26	1.18	1.12	N/A

부록 B: JDBC 드라이버 업그레이드

다음은 JDBC 드라이버 업그레이드입니다.

드라이버	과거 AWS Glue 버전의 JDBC 드라이버 버전	AWS Glue 3.0의 JDBC 드라이버 버전	AWS Glue 4.0의 JDBC 드라이버 버전
MySQL	5.1	8.0.23	8.0.23
Microsoft SQL Server	6.1.0	7.0.0	9.4.0

드라이버	과거 AWS Glue 버전의 JDBC 드라이버 버전	AWS Glue 3.0의 JDBC 드라이버 버전	AWS Glue 4.0의 JDBC 드라이버 버전
Oracle Database	11.2	21.1	21.7
PostgreSQL	42.1.0	42.2.18	42.3.6
MongoDB	2.0.0	4.0.0	4.7.2
Amazon Redshift	redshift-jdbc41-1.2.12.1017	redshift-jdbc41-1.2.12.1017	redshift-jdbc42-2.1.0.16

부록 C: 커넥터 업그레이드

다음은 커넥터 업그레이드입니다.

드라이버	AWS Glue 3.0의 커넥터 버전	AWS Glue 4.0의 커넥터 버전
MongoDB	3.0.0	10.0.4
Hudi	0.10.1	0.12.1
Delta Lake	1.0.0	2.1.0
Iceberg	0.13.1	1.0.0
DynamoDB	1.11	1.12

AWS Glue for Ray를 평가판에서 Ray2.4 런타임 환경으로 마이그레이션

Warning

AWS Glue for Ray(평가판) 작업을 AWS Glue Studio에 저장하면 해당 작업이 Ray2.4 런타임으로 자동 업그레이드됩니다. 스크립트와 호환성 문제가 발생하는 경우 지원팀에 문의하세요.

AWS Glue for Ray(평가판)에서 생성한 AWS Glue 작업은 AWS Glue for Ray로 마이그레이션해야 합니다. 여기에는 작업 구성의 몇 가지 동시 변경이 포함됩니다.

- Runtime 필드에 Ray2.4 런타임 값을 입력합니다. 그러면 기본 Ray 버전이 2.0.0에서 2.4.0으로 업그레이드됩니다.
- 평가판에 기본적으로 포함된 특정 Python 라이브러리는 더 이상 제공되지 않습니다. 작업에서 pandas(aws wrangler), dask, modin 또는 pymars에 대한 AWS SDK를 사용하는 경우 이를 추가 라이브러리로 포함해야 합니다. 추가 Python 라이브러리를 포함하는 방법에 대한 자세한 내용은 [the section called “Ray 작업을 위한 추가 Python 모듈”](#) 섹션을 참조하세요.
- `--additional-python-modules` 파라미터를 사용하는 경우 이 워크플로를 지원하는 데 사용되는 파라미터는 `--pip-install` 및 `--s3-py-modules`로 구분됩니다. 이런 파라미터에 대한 자세한 내용은 [the section called “Ray 작업을 위한 추가 Python 모듈”](#) 섹션을 참조하세요.
- `--auto-scaling-ray-min-workers` 파라미터를 사용하는 경우 해당 이름이 `--min-workers`로 변경됩니다.

AWS Glue 버전 지원 정책

AWS Glue는 분석, 기계 학습, 애플리케이션 개발을 위해 데이터를 쉽게 검색, 준비, 결합할 수 있게 해 주는 서버리스 데이터 통합 서비스입니다. AWS Glue 작업에는 AWS Glue에서 데이터 통합 작업을 수행하는 비즈니스 로직이 포함되어 있습니다. AWS Glue에는 Spark(배치 및 스트리밍), Ray 및 Python 셸이라는 세 가지 작업 유형이 있습니다. 작업을 정의할 때 기본 Spark, Ray 또는 Python 런타임 환경에서 버전을 구성하는 AWS Glue 버전을 지정합니다. 예: AWS Glue 버전 2.0 Spark 작업은 Spark 2.4.3 및 Python 3.7을 지원합니다.

지원 정책

AWS Glue에서 이전 AWS Glue 버전에 대한 지원을 중단하는 경우가 있습니다. 그러나 사용되지 않는 버전에서 실행되는 작업은 더 이상 기술 지원을 받을 수 없습니다. AWS Glue는 사용되지 않는 버전에 보안 패치 또는 기타 업데이트를 더 이상 적용하지 않습니다. AWS Glue 역시 더 이상 사용되지 않는 버전에서 작업이 실행될 때 SLA를 준수하지 않습니다.

AWS Glue 버전 2.0 이상에서 지원이 종료되면 작업을 생성할 수 없고 작업을 편집하거나 실행할 수만 있습니다.

다음 AWS Glue 버전의 지원 종료 시점에 도달했거나 예정되어 있습니다. 지원 종료는 지정된 날짜의 자정(태평양 표준시)부터 시작됩니다.

Type	Glue 버전	지원 종료
Spark	Spark 2.2, Scala 2(Glue 버전 0.9)	2022년 6월 1일
Spark	Spark 2.2, Python 2(Glue 버전 0.9)	2022년 6월 1일
Spark	Spark 2.4, Python 2(Glue 버전 1.0)	2022년 6월 1일
Spark	Spark 2.4, Python 3(Glue 버전 1.0)	2022년 9월 30일
Spark	Spark 2.4, Scala 2(Glue 버전 1.0)	2022년 9월 30일
Spark	Glue 버전 2.0	2024년 1월 31일
Type	Python 버전	지원 종료
Python 셸	Python 2(Glue 버전 1.0)	2022년 6월 1일
Type	노트북 버전	지원 종료
개발 엔드포인트	Zeppelin 노트북	2022년 9월 30일

AWS는 해당 작업을 지원되는 버전으로 마이그레이션할 것을 적극 권장합니다.

Spark 작업을 최신 AWS Glue 버전으로 마이그레이션하는 방법에 대한 자세한 내용은 [AWS Glue 작업을 AWS Glue 버전 4.0으로 마이그레이션](#)을 참조하세요.

Python 셸 작업을 최신 AWS Glue 버전으로 마이그레이션하는 경우:

- 콘솔에서 Python 3 (Glue Version 4.0)을 선택합니다.
- [CreateJob/UpdateJob](#) API에서 GlueVersion 파라미터를 2.0으로 설정하고 Command 파라미터에서 PythonVersion을 3으로 설정합니다. GlueVersion 구성은 Python 셸 작업의 동작에 영향을 미치지 않으므로 GlueVersion을 늘려도 이점이 없습니다.
- 작업 스크립트가 Python 3과 호환되도록 해야 합니다.

Note

2022년 8월 인도네시아 자카르타(ap-southeast-3) 리전이 출시되기 전에 시작된 모든 AWS 리전에는 AWS Glue 버전 0.9/1.0 작업 실행을 허용하는 고객의 허용 목록이 있습니다. 이러한 이전 리전에서는 null 값을 포함하는 작업을 생성할 수 있으며 리전에 따라 기본적으로 버전 0.9/1.0으로 설정됩니다. 이후에 출시되는 AWS 리전의 경우 API에서 AWS Glue 버전을 명시적으로 설정해야 합니다. AWS Glue에서는 더 이상 null 파라미터를 허용하지 않습니다. 파라미터에서 0.9 또는 1.0을 전달하면 'Glue Version 0.9 (or) 1.0 is not supported' 오류가 발생합니다.

에서 Spark 채용공고를 다루기 AWS Glue

Spark ETL 작업에 AWS Glue 대한 정보를 제공합니다.

주제

- [AWS Glue 작업 파라미터](#)
- [AWS Glue 스파크 앤 PySpark 잡스](#)
- [AWS Glue에서 스트리밍 ETL 작업](#)
- [AWS Lake Formation FindMatches로 레코드 매칭](#)
- [Apache Spark 프로그램을 AWS Glue로 마이그레이션](#)

AWS Glue 작업 파라미터

AWS Glue 작업을 생성할 때, Role 및 와 같은 몇 가지 표준 필드를 설정합니다 WorkerType. Argument 필드(콘솔의 작업 파라미터)를 통해 추가 구성 정보를 제공할 수 있습니다. 이 필드에서 이 항목에 나열된 인수 (파라미터) 와 함께 AWS Glue 작업을 제공할 수 있습니다. AWS Glue Job API에 대한 자세한 내용은 을 참조하십시오 [the section called “작업”](#).

작업 파라미터 설정

Job Parameters(작업 파라미터) 제목 아래의 Job details(작업 세부 정보) 탭에서 콘솔을 통해 작업을 구성할 수 있습니다. 작업 설정 DefaultArguments 또는 작업 실행 NonOverridableArguments 시 설정을 AWS CLI 통해 작업을 구성할 수도 있습니다. Arguments 작업에 설정된 인수는 작업이 실행될 때마다 전달되지만, 작업 실행에 설정된 인수는 해당 개별 실행에 대해서만 전달됩니다.

예를 들어 다음은 작업 파라미터를 설정하기 위해 `--arguments`를 사용하여 작업을 실행하는 구문입니다.

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://my_glue/libraries/test_lib.py"'
```

작업 파라미터 액세스

AWS Glue 스크립트를 작성할 때 작업 매개변수 값에 액세스하여 자체 코드의 동작을 변경하는 것이 좋습니다. 라이브러리에서 이를 수행하기 위한 도우미 메서드가 제공됩니다. 이러한 메서드는 작업 파라미터 값을 재정의하는 작업 실행 파라미터 값을 해석합니다. 여러 위치에 설정된 파라미터를 해석하는 경우 `NonOverridableArguments` 작업은 `Arguments` 작업 실행을 재정의하고 이 작업 실행은 `DefaultArguments` 작업을 재정의합니다.

Python에서:

Python 작업에서는 `getResolvedParameters` 함수가 제공됩니다. 자세한 정보는 [the section called “getResolvedOptions 옵션”](#)을 참조하세요. 작업 파라미터는 `sys.argv` 변수에서 사용할 수 있습니다.

Scala에서:

Scala 작업에서는 `GlueArgParser` 객체가 제공됩니다. 자세한 정보는 [the section called “GlueArgParser”](#)을 참조하세요. 작업 파라미터는 `sys.Args` 변수에서 사용할 수 있습니다.

작업 파라미터 참조

AWS Glue에서는 작업 및 작업 실행에 대한 스크립트 환경을 설정하는 데 사용할 수 있는 다음과 같은 인수 이름을 인식합니다.

--additional-python-modules

설치할 Python 패키지 세트를 나타내는 쉼표로 구분된 목록입니다. PyPI에서 패키지를 설치하거나 사용자 지정 배포를 제공할 수 있습니다. PyPI 패키지 항목은 `package==version` 형식(대상 패키지의 PyPI 이름 및 버전)입니다. 사용자 지정 배포 항목은 배포에 대한 S3 경로입니다.

항목은 Python 버전 일치를 사용하여 패키지와 버전을 일치시킵니다. 즉, 2개의 등호(예: `==`)를 사용해야 합니다. 다른 버전 일치 연산자도 있습니다. 자세한 내용을 알아보려면 [PEP 440](#)을 참조하세요.

모듈 설치 옵션을 `pip3`에 전달하려면 [--python-modules-installer-option](#) 파라미터를 사용합니다.

--auto-scale-within-microbatch

기본값은 false입니다. 이 매개변수는 스트리밍 데이터를 일련의 마이크로 배치로 처리하는 AWS Glue 스트리밍 작업에만 사용할 수 있으며, Auto Scaling을 활성화해야 합니다. 이 값을 false로 설정한 경우 완료된 마이크로 배치에 대한 배치 지속 시간의 지수 이동 평균을 계산하고 이 값을 창 크기와 비교하여 실행기 수의 스케일 업 또는 스케일 다운 여부를 결정합니다. 마이크로 배치가 완료될 때만 규모가 조정됩니다. 이 값을 true로 설정한 경우 마이크로 배치 중에 Spark 작업 수가 30초 동안 동일하게 유지되거나 현재 배치 처리가 창 크기보다 클 때 스케일 업됩니다. 실행기가 60초 넘게 유휴 상태이거나 배치 지속 시간의 지수 이동 평균이 낮으면 실행기 수가 감소합니다.

--class

Scala 스크립트 진입점으로써 Scala 클래스. 이는 --job-language가 scala로 설정된 경우에만 적용됩니다.

--continuous-log-conversionPattern

연속 로깅에 대해 활성화된 작업의 사용자 지정 변환 로그 패턴을 지정합니다. 변환 패턴은 드라이버 로그와 실행기 로그에만 적용됩니다. AWS Glue 진행률 표시줄에는 영향을 주지 않습니다.

--continuous-log-logGroup

연속 로깅이 활성화된 작업의 사용자 지정 Amazon CloudWatch 로그 그룹 이름을 지정합니다.

--continuous-log-logStreamPrefix

연속 로깅이 활성화된 작업의 사용자 지정 CloudWatch 로그 스트림 접두사를 지정합니다.

--customer-driver-env-vars 및 **--customer-executor-env-vars**

이러한 파라미터는 각 작업자(드라이버 또는 실행자)에 대해 각각 운영 체제의 환경 변수를 설정합니다. Glue를 기반으로 플랫폼 및 사용자 지정 프레임워크를 구축할 때 이러한 매개변수를 사용하여 사용자가 AWS Glue 위에 작업을 작성하도록 할 수 있습니다. 이 두 플래그를 활성화하면 작업 스크립트 자체에 동일한 논리를 삽입하지 않고도 드라이버와 실행기에 각각 다른 환경 변수를 설정할 수 있습니다.

사용 예

다음은 이러한 파라미터를 사용하는 예제입니다.

```
"--customer-driver-env-vars", "CUSTOMER_KEY1=VAL1,CUSTOMER_KEY2=\"val2,val2 val2\"",
"--customer-executor-env-vars", "CUSTOMER_KEY3=VAL3,KEY4=VAL4"
```

작업 실행 인수에서 이를 설정하는 것은 다음 명령을 실행하는 것과 동일합니다.

드라이버에서:

- `export CUSTOMER_KEY1=VAL1`
- `export CUSTOMER_KEY2="val2,val2 val2"`

실행기에서:

- `export CUSTOMER_KEY3=VAL3`

그런 다음 작업 스크립트 자체에서 `os.environ.get("CUSTOMER_KEY1")` 또는 `System.getenv("CUSTOMER_KEY1")`을 사용하여 환경 변수를 검색할 수 있습니다.

적용된 구문

환경 변수를 정의할 때는 다음 표준을 준수하세요.

- 각 키에는 `CUSTOMER_ prefix`가 있어야 합니다.

예를 들어 `"CUSTOMER_KEY3=VAL3,KEY4=VAL4"`의 경우 `KEY4=VAL4`는 무시되고 설정되지 않습니다.

- 각 키와 값 쌍은 단일 쉼표로 구분되어야 합니다.

예: `"CUSTOMER_KEY3=VAL3, CUSTOMER_KEY4=VAL4"`

- '값'에 공백이나 쉼표가 있는 경우 따옴표 안에 정의해야 합니다.

예: `CUSTOMER_KEY2=\"val2,val2 val2\"`

이 구문은 `bash` 환경 변수 설정의 표준에 가깝게 모델링합니다.

--datalake-formats

AWS Glue 3.0 이상 버전에서 지원됩니다.

사용할 데이터 레이크 프레임워크를 지정합니다. AWS Glue는 지정한 프레임워크에 필요한 JAR 파일을 `classpath`에 추가합니다. 자세한 정보는 [AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용](#)을 참조하세요.

다음 값 중 하나 이상을 쉼표로 구분하여 지정할 수 있습니다.

- `hudi`
- `delta`
- `iceberg`

예를 들어 다음 인수를 전달하여 세 프레임워크를 모두 지정합니다.

```
'--datalake-formats': 'hudi,delta,iceberg'
```

--disable-proxy-v2

VPC를 통해 스크립트에서 AWS Glue 시작되는 Amazon S3로의 AWS 서비스 호출을 허용하려면 서비스 프록시를 비활성화하십시오. CloudWatch 자세한 내용은 [VPC를 통과하도록 AWS 호출 구성](#)을 참조하세요. 서비스 프록시를 비활성화하려면 이 파라미터 값을 true로 설정합니다.

--enable-auto-scaling

이 값을 true로 설정할 경우 Auto Scaling과 작업자별 청구를 사용하는 기능을 켭니다.

--enable-continuous-cloudwatch-log

AWS Glue 작업에 대한 실시간 연속 로깅을 활성화합니다. 로그인한 실시간 Apache Spark 작업 로그를 볼 수 있습니다. CloudWatch

--enable-continuous-log-filter

지속 로깅을 활성화하여 작업을 생성하거나 편집할 때 표준 필터(true) 또는 필터 없음(false)을 지정합니다. 표준 필터를 선택하면 유용하지 않은 Apache Spark 드라이버/실행기 및 Apache Hadoop YARN 하트비트 로그 메시지가 제거됩니다. 필터 없음을 선택하면 모든 로그 메시지가 제공됩니다.

--enable-glue-datacatalog

AWS Glue 데이터 카탈로그를 Apache Spark Hive 메타스토어로 사용할 수 있습니다. 이 기능을 활성화하려면 값을 true로 설정합니다.

--enable-job-insights

AWS Glue 작업 실행 통찰력을 통해 추가 오류 분석 모니터링을 수행할 수 있습니다. 자세한 내용은 [the section called “AWS Glue 작업 실행 인사이트를 사용한 모니터링”](#) 단원을 참조하세요. 기본적으로 이 값은 true로 설정되고 작업 실행 인사이트가 활성화됩니다.

이 옵션은 AWS Glue 버전 2.0과 3.0에서 사용할 수 있습니다.

--enable-metrics

이 작업 실행을 위해 작업 프로파일링용 측정치 수집을 활성화합니다. 이러한 지표는 AWS Glue 콘솔과 Amazon CloudWatch 콘솔에서 사용할 수 있습니다. 이 파라미터의 값은 관련이 없습니다. 이 기능을 활성화하려면 이 파라미터에 어떤 값이든 제공할 수 있지만 명확성을 위해 true를 권장합니다. 이 기능을 비활성화하려면 작업 구성에서 이 파라미터를 제거하세요.

--enable-observability-metrics

AWS Glue 콘솔 및 콘솔의 Job Runs Monitoring 페이지에서 일련의 관찰 가능성 지표를 사용하여 각 작업 실행 내에서 발생하는 상황에 대한 통찰력을 생성할 수 있습니다. Amazon CloudWatch 이 기능을 활성화하려면 이 파라미터의 값을 `true`로 설정합니다. 이 기능을 비활성화하려면 `false`로 설정하거나 작업 구성에서 이 파라미터를 제거하세요.

--enable-rename-algorithm-v2

EMRFS 이름 바꾸기 알고리즘 버전을 버전 2로 설정합니다. Spark 작업에서 동적 파티션 덮어쓰기 모드를 사용하는 경우 중복 파티션이 생성될 가능성이 있습니다. 예를 들어, `s3://bucket/table/location/p1=1/p1=1`과 같은 중복 파티션으로 끝날 수 있습니다. 여기서 P1은 덮어쓰는 파티션입니다. 이름 바꾸기 알고리즘 버전 2가 이 문제를 해결합니다.

이 옵션은 AWS Glue 버전 1.0에서만 사용할 수 있습니다.

--enable-s3-parquet-optimized-commmitter

Amazon S3에 Parquet 데이터를 쓸 수 있도록 EMRFS S3 최적화 커미터를 사용합니다. AWS Glue 작업을 생성하거나 업데이트할 때 AWS Glue 콘솔을 통해 파라미터/값 쌍을 제공할 수 있습니다. 값을 `true`로 설정하면 커미터가 활성화됩니다. 기본적으로 플래그는 AWS Glue 3.0에서는 켜져 있고 Glue 2.0에서는 AWS 꺼져 있습니다.

자세한 내용은 [EMRFS S3 최적화 커미터 사용](#)을 참조하십시오.

--enable-spark-ui

`true`로 설정하면 Spark UI를 사용하여 AWS Glue ETL 작업을 모니터링하고 디버그하는 기능을 켭니다.

--executor-cores

병렬로 실행할 수 있는 Spark 작업 수입니다. 이 옵션은 AWS Glue 3.0 이상에서 지원됩니다. 값은 작업자 유형에 있는 vCPU 수의 2배(G.1X의 경우 8, G.2X의 경우 16, G.4X의 경우 32, G.8X의 경우 64)를 초과해서는 안 됩니다. 이 구성을 업데이트할 때는 주의해야 합니다. 병렬 처리 성능을 높이면 메모리 및 디스크 부담이 커질 뿐 아니라 소스 및 대상 시스템에서 제한이 발생할 수 있으므로 작업 성능에 영향을 미칠 수 있습니다(예: Amazon RDS에서 동시 연결이 늘어남).

--extra-files

스크립트를 실행하기 전에 AWS Glue가 스크립트의 작업 디렉터리에 복사하는 구성 파일과 같은 추가 파일에 대한 Amazon S3 경로입니다. 여러 값은 쉼표(,)로 구분된 완전한 경로여야 합니다. 개별 파일만 지원되면 디렉터리 경로는 지원되지 않습니다. 이 옵션은 Python 셸 작업 유형에서 지원되지 않습니다.

--extra-jars

스크립트를 실행하기 전에 AWS Glue가 Java 클래스 경로에 추가하는 추가 Java .jar 파일에 대한 Amazon S3 경로입니다. 여러 값은 쉼표(,)로 구분된 완전한 경로여야 합니다.

--extra-py-files

스크립트를 실행하기 전에 AWS Glue가 Python 경로에 추가하는 추가 Python 모듈에 대한 Amazon S3 경로입니다. 여러 값은 쉼표(,)로 구분된 완전한 경로여야 합니다. 개별 파일만 지원되면 디렉터리 경로는 지원되지 않습니다.

--job-bookmark-option

작업 북마크 동작을 제어합니다. 다음 옵션 값을 설정할 수 있습니다.

-- job-bookmark-option 값	설명
job-bookmark-enable	이전에 처리된 데이터를 계속 추적합니다. 작업이 실행되면 마지막 체크포인트 이후의 새로운 데이터를 진행합니다.
job-bookmark-disable	항상 전체 데이터셋을 진행합니다. 이전에 실행된 작업의 출력값 관리는 여러분이 직접 수행해야 합니다.
job-bookmark-pause	<p>마지막 북마크의 상태를 업데이트하지 않고 마지막으로 성공한 실행 이후의 증분 데이터 또는 다음 하위 옵션으로 식별된 범위의 데이터를 처리합니다. 이전에 실행된 작업의 출력값 관리는 여러분이 직접 수행해야 합니다. 두 하위 옵션은 다음과 같습니다.</p> <ul style="list-style-type: none"> • job-bookmark-from <from-value> 는 이전에 마지막으로 성공한 실행까지 처리된 모든 입력을 나타내는 실행 ID이며, 지정된 실행 ID를 포함합니다. 해당하는 입력은 무시됩니다. • job-bookmark-to <to-value> 는 이전에 마지막으로 성공한 실행까지 처리된 모든 입력을 나타내는 실행 ID이며, 지정된 실행 ID를 포함합니다. <from-value> 에 의해 식별된 입력을 제외한 해당하는 입력이 작업에 의해 처리됩니다. 이 입력 이후의 입력도 처리에서 제외됩니다.

--job-bookmark-option 값	설명
	<p>이 옵션 세트를 지정하면 작업 북마크 상태가 업데이트되지 않습니다.</p> <p>하위 옵션은 선택 사항입니다. 그러나 사용할 경우 두 하위 옵션 모두 제공해야 합니다.</p>

예를 들어, 작업 북마크를 활성화하려면 다음 인수를 전달합니다.

```
'--job-bookmark-option': 'job-bookmark-enable'
```

--job-language

스크립트 프로그래밍 언어. 이 값은 scala 또는 python이어야 합니다. 이 파라미터가 존재하지 않을 경우 기본값은 python입니다.

--python-modules-installer-option

[--additional-python-modules](#)로 모듈을 설치할 때 pip3에 전달할 옵션을 정의하는 일반 텍스트 문자열입니다. 명령줄에서와 같이 공백으로 구분되고 접두사가 대시로 된 옵션을 제공합니다. 사용에 대한 자세한 내용을 알아보려면 [the section called “AWS Glue 2.0+에서 pip를 사용하여 추가 Python 모듈 설치”](#)를 참조하세요.

Note

Python 3.9를 사용하는 AWS Glue 작업에는 이 옵션이 지원되지 않습니다.

--scriptLocation

ETL 스크립트가 있는 Amazon Simple Storage Service(Amazon S3) 위치입니다(s3://path/to/my/script.py 형식). 이 파라미터는 JobCommand 객체에 설정된 스크립트 위치를 재정의합니다.

--spark-event-logs-path

Amazon S3 경로를 지정합니다. Spark UI 모니터링 기능을 사용하는 경우 AWS Glue는 Spark UI 이벤트를 저장하기 위해 임시 디렉터리로 사용할 수 있는 버킷에 이 Amazon S3 경로에 대한 Spark 이벤트 로그를 30초마다 플러시합니다.

--TempDir

작업의 임시 디렉터리로 사용될 수 있는 버킷에 대한 Amazon S3 경로를 지정합니다.

예를 들어, 임시 디렉터리를 설정하려면 다음 인수를 전달합니다.

```
'--TempDir': 's3-path-to-directory'
```

Note

리전에 버킷이 아직 없는 경우 AWS Glue에서 작업에 대한 임시 버킷을 생성합니다. 이 버킷은 퍼블릭 액세스를 허용할 수 있습니다. Amazon S3에서 버킷을 수정하여 퍼블릭 액세스 차단 설정하거나 나중에 해당 리전의 모든 작업이 완료된 후 버킷을 삭제할 수 있습니다.

--use-postgres-driver

이 값을 true로 설정하면 Amazon Redshift JDBC 드라이버와의 충돌을 피하기 위해 클래스 경로에서 Postgres JDBC 드라이버의 우선순위를 지정합니다. 이 옵션은 AWS Glue 버전 2.0에서만 사용할 수 있습니다.

--user-jars-first

이 값을 true로 설정하면 클래스 경로에 있는 고객의 추가 JAR 파일의 우선순위를 지정합니다. 이 옵션은 AWS Glue 버전 2.0 이상에서만 사용할 수 있습니다.

--conf

Spark 구성 파라미터를 제어합니다. 고급 사용 사례에 해당됩니다.

--encryption-type

레거시 파라미터. 보안 구성을 사용하여 해당 동작을 구성해야 합니다. 보안 구성에 대한 자세한 내용은 [the section called “AWS Glue에서 작성한 데이터 암호화”](#) 섹션을 참조하세요.

AWS Glue는 다음 인수를 내부적으로 사용하므로 절대 사용하지는 않습니다.

- --debug - AWS Glue 내부 설정해서는 안 됩니다.
- --mode - AWS Glue 내부 설정해서는 안 됩니다.

- --JOB_NAME - AWS Glue 내부 설정해서는 안 됩니다.
- --endpoint - AWS Glue 내부 설정해서는 안 됩니다.

AWS Glue에서는 사이트별 사용자 지정을 수행하기 위해 `sitecustomize`를 사용하여 Python `site` 모듈에서 환경을 부트스트래핑할 수 있도록 지원합니다. 자체 초기화 함수의 부트스트래핑은 고급 사용 사례에만 권장되며, AWS Glue 4.0에서 효과가 가장 뛰어납니다.

환경 변수 접두사(`GLUE_CUSTOMER`)는 고객 전용으로 예약되어 있습니다.

AWS Glue 스파크 앤 PySpark 잡스

다음 섹션에서는 AWS Glue Spark 및 PySpark 작업에 대한 정보를 제공합니다.

주제

- [AWS Glue에서 Spark 및 PySpark 작업 추가](#)
- [처리된 데이터를 작업 북마크로 추적](#)
- [Amazon S3의 AWS Glue Spark 셔플 플러그인](#)
- [AWS Glue Spark 작업 모니터링](#)

AWS Glue에서 Spark 및 PySpark 작업 추가

다음 섹션에서는 AWS Glue에서 Spark 및 PySpark 작업을 추가하는 방법에 대해 설명합니다.

주제

- [에서 Spark 작업에 대한 작업 속성 구성 AWS Glue](#)
- [AWS Glue 콘솔에서 Spark 스크립트 편집](#)
- [작업\(레거시\)](#)

에서 Spark 작업에 대한 작업 속성 구성 AWS Glue

AWS Glue 작업은 소스 데이터에 연결하여 처리한 다음 데이터 대상에 작성하는 스크립트를 캡슐화합니다. 일반적으로 작업은 추출, 변환 및 로드(ETL) 스크립트를 실행합니다. 작업은 범용 Python 스크립트(Python 셸 작업)를 실행할 수도 있습니다. AWS Glue 트리거는 일정 또는 이벤트에 따라 또는 필요

에 따라 작업을 시작할 수 있습니다. 작업 실행을 모니터링하여 완료 상태, 지속 시간, 시작 시간 같은 실행 시간 지표를 이해할 수 있습니다.

AWS Glue에서 생성하는 스크립트를 사용하거나 직접 제공할 수 있습니다. 소스 스키마와 대상 위치 또는 스키마를 사용하여 AWS Glue 코드 생성기는 Apache Spark API (PySpark) 스크립트를 자동으로 만들 수 있습니다. 이 스크립트를 시작 포인트로 사용할 수 있고 목적에 부합하기 위해 편집할 수도 있습니다.

AWS Glue는 JSON, CSV, ORC(Optimized Row Columnar), Apache Parquet 및 Apache Avro를 포함해 몇 가지 데이터 포맷으로 출력 파일을 작성할 수 있습니다. 몇 가지 데이터 포맷의 경우, 일반 압축 포맷이 작성될 수 있습니다.

AWS Glue의 작업은 Spark, 스트리밍 ETL, Python 셸의 세 가지 유형이 있습니다.

- Spark 작업은 에서 관리하는 Apache Spark 환경에서 실행됩니다. AWS Glue데이터를 배치로 처리합니다.
- 스트리밍 ETL 작업은 데이터 스트림에 대해 ETL을 수행한다는 점을 제외하고 Spark 작업과 유사합니다. 이는 Apache Spark Structured Streaming 프레임워크를 사용합니다. 스트리밍 ETL 작업에 일부 Spark 작업 기능을 사용할 수 없습니다.
- Python 셸 작업은 Python 스크립트를 셸로 실행하고 사용 중인 AWS Glue 버전에 따라 다른 Python 버전을 지원합니다. 이러한 작업을 사용하여 Apache Spark 환경이 필요하지 않은 작업을 예약하고 실행할 수 있습니다.

Spark 작업에 대한 작업 속성 정의

작업을 AWS Glue 콘솔에서 정의할 경우 AWS Glue 런타임 환경을 제어하기 위한 속성 값을 제공합니다.

다음 목록에서는 Spark 작업의 속성을 설명합니다. Python 셸 작업의 속성은 [Python 셸 작업에 대한 작업 속성 정의](#) 단원을 참조하십시오. 스트리밍 ETL 작업의 속성은 [the section called “스트리밍 ETL 작업에 대한 작업 속성 정의”](#) 단원을 참조하십시오.

속성은 AWS Glue 콘솔의 [작업 추가(Add job)] 마법사에 나타나는 순서대로 나열됩니다.

명칭

최대 255자 길이의 UTF-8 문자열을 제공합니다.

설명

최대 2048자의 설명을 입력합니다(선택 사항).

IAM 역할

작업을 실행하고 데이터 스토어에 액세스하는 데 사용되는 리소스에 대한 권한 부여용 IAM 역할을 지정합니다. AWS Glue 작업을 실행하는 권한에 대한 자세한 내용은 [AWS Glue의 ID 및 액세스 관리](#) 단원을 참조하십시오.

유형

ETL 작업의 유형입니다. 이는 선택한 데이터 소스 유형에 따라 자동으로 설정됩니다.

- Spark는 작업 명령 glueetl로 Apache Spark ETL 스크립트를 실행합니다.
- Spark Streaming은 gluestreaming 작업 명령으로 Apache Spark 스트리밍 ETL 스크립트를 실행합니다. 자세한 정보는 [the section called “스트리밍 ETL 작업”](#)을 참조하십시오.
- Python 셸에서 작업 명령 pythonshell로 Python 스크립트를 실행합니다. 자세한 정보는 [AWS Glue에서 Python 셸 작업에 대한 작업 속성 구성](#)을 참조하십시오.

AWS Glue 버전

AWS Glue 버전이 다음 표에 지정된 대로 작업에 사용할 수 있는 Apache Spark 및 Python의 버전을 결정합니다.

AWS Glue 버전	지원되는 Spark 및 Python 버전
4.0	<ul style="list-style-type: none"> • Spark 3.3.0 • Python 3.10
3.0	<ul style="list-style-type: none"> • Spark 3.1.1 • Python 3.7
2.0	<ul style="list-style-type: none"> • Spark 2.4.3 • Python 3.7
1.0	<ul style="list-style-type: none"> • Spark 2.4.3 • Python 2.7 • Python 3.6
0.9	<ul style="list-style-type: none"> • Spark 2.2.1 • Python 2.7

작업자 유형

다음 작업자 유형을 사용할 수 있습니다.

AWS Glue 작업자가 사용할 수 있는 리소스는 DPU 단위로 측정됩니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다.

- **G.1X** – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 1개의 DPU(vCPU 4개, 메모리 16GB)에 매핑됩니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- **G.2X** – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 128GB 디스크(약 77GB의 여유 공간)에서 2개의 DPU(vCPU 8개, 메모리 32GB)에 매핑됩니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- **G.4X** – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 256GB 디스크(약 235GB의 여유 공간)에서 4개의 DPU(vCPU 16개, 메모리 64GB)에 매핑됩니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 미국 동부 (오하이오), 미국 동부 (버지니아 북부), 미국 서부 (오레곤), 아시아 태평양 (싱가포르), 아시아 태평양 (시드니), 아시아 태평양 (도쿄), 캐나다 (중부), 유럽 (아일랜드) 및 유럽 (스톡홀름) 의 AWS Glue 버전 3.0 이상의 Spark ETL 작업에만 사용할 수 있습니다. AWS
- **G.8X** – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 512GB 디스크(약 487GB의 여유 공간)에서 8개의 DPU(vCPU 32개, 메모리 128GB)에 매핑됩니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 작업자 유형에 지원되는 AWS 지역과 동일한 지역에서 AWS Glue 버전 3.0 이상의 Spark ETL G.4X 작업에만 사용할 수 있습니다.
- **G.025X** – 이 유형을 선택할 경우 Number of workers(작업자 수) 값도 제공합니다. 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 0.25개의 DPU(vCPU 2개, 메모리 4GB)에 매핑됩니다. 볼륨이 낮은 스트리밍 작업에 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 AWS Glue 버전 3.0 스트리밍 작업에만 사용할 수 있습니다.

ETL 작업을 실행하는 데 사용된 DPU의 개수에 따라 시간당 비용이 부과됩니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

AWS Glue 버전 1.0 이전 작업의 경우 콘솔을 사용하여 작업을 구성하고 [작업자 유형(Worker type)]을 [표준(Standard)]로 지정하면, [최대 용량(Maximum capacity)]이 설정되고 [작업자 수 (Number of workers)]는 [최대 용량(Maximum capacity)] - 1이 됩니다. AWS Command Line

Interface (AWS CLI) 또는 AWS SDK를 사용하는 경우 최대 용량 매개변수를 지정하거나 작업자 유형과 작업자 수를 모두 지정할 수 있습니다.

AWS Glue 버전 2.0 이상 작업의 경우 최대 용량을 지정할 수 없습니다. 대신 [작업자 유형(Worker type)] 및 [작업자 수(Number of workers)]를 지정해야 합니다.

언어

ETL 스크립트의 코드는 작업 논리를 정의합니다. 이 스크립트는 Python 또는 Scala에 코딩될 수 있습니다. 작업이 실행하는 스크립트가 AWS Glue가 생성했는지 여러분이 제공했는지 선택할 수 있습니다. Amazon Simple Storage Service(Amazon S3)에서 스크립트 이름과 위치를 제공합니다. 경로의 스크립트 디렉터리와 동일한 이름의 파일이 없다는 것을 확인합니다. 스크립트 작성에 대한 자세한 내용은 [AWS Glue 프로그래밍 안내서](#) 단원을 참조하십시오.

요청된 작업자 수

대부분의 작업자 유형에서 작업이 실행될 때 할당되는 작업자 수를 지정해야 합니다.

작업 북마크

작업 실행 시 AWS Glue가 상태 정보를 진행하는 방법을 지정합니다. 이전에 처리된 데이터를 기억하거나, 상태 정보를 업데이트하거나, 상태 정보를 무시할 수 있습니다. 자세한 정보는 [the section called “처리된 데이터를 작업 북마크로 추적”](#)을 참조하세요.

유연한 실행

AWS Studio 또는 API를 사용하여 작업을 구성할 때 표준 또는 유연한 작업 실행 클래스를 지정할 수 있습니다. 작업의 우선순위와 시간 민감도는 다를 수 있습니다. 표준 실행 클래스는 빠른 작업 시작 및 전용 리소스가 필요한 시간에 민감한 워크로드에 적합합니다.

유연한 실행 클래스는 사전 프로덕션 작업, 테스트 및 일회성 데이터 로드와 같은 긴급하지 않은 작업에 적합합니다. 유연한 작업 실행은 AWS Glue 버전 3.0 이상 및 G.1X 또는 G.2X 작업자 유형을 사용하는 작업에 지원됩니다.

유연한 작업 실행은 한 번에 실행하는 작업자 수를 기준으로 요금이 청구됩니다. 유연한 작업 실행에 대한 작업자 수를 추가하거나 제거할 수 있습니다. Max Capacity*Execution Time의 간단한 계산으로 청구하는 대신 각 작업자는 작업 실행 중에 실행된 시간으로 요금이 청구됩니다. 청구 금액은 (Number of DPUs per worker*time each worker ran)의 합입니다.

자세한 내용은 AWS Studio 또는 [작업](#) 및 [의 도움말 패널](#)을 참조하십시오 [작업 실행](#).

재시도 횟수

AWS Glue가 자동적으로 작업을 재시작하지 못하면 재시작하도록 시간 수를 0~10사이에서 지정합니다. 시간 제한에 도달한 작업은 다시 시작되지 않습니다.

작업 제한 시간

최대 실행 제한을 분으로 설정합니다. 배치 작업의 경우 기본값은 2,880분(48시간)입니다. 작업 실행 시간이 이 제한을 초과하면 작업 실행 상태가 TIMEOUT으로 변경됩니다.

스트리밍 작업의 제한 시간 값은 7일 또는 100-80분 미만이어야 합니다. 값을 비워 두면 유지 관리 기간을 설정하지 않은 경우에 따라 7일 후에 작업이 다시 시작됩니다. 유지 관리 기간을 설정한 경우 7일 후 유지 관리 기간 중에 다시 시작됩니다.

작업 시간 초과에 대한 모범 사례

작업은 실행 시간을 기준으로 요금이 청구됩니다. 예상치 못한 요금이 부과되지 않도록 하려면 작업의 예상 실행 시간에 적합한 제한 시간 값을 구성하세요.

고급 속성

스크립트 파일 이름

작업의 고유한 스크립트 이름입니다. Untitled job으로 지정할 수 없습니다.

스크립트 경로

스크립트의 Amazon S3 위치입니다. 경로는 s3://bucket/prefix/path/ 형식이어야 합니다. 슬래시(/)로 끝나야 하며 파일을 포함하지 않아야 합니다.

작업 지표

이 작업이 실행될 때 Amazon CloudWatch 지표 생성을 켜거나 끕니다. 프로파일링 데이터를 보려면 이 옵션을 활성화해야 합니다. 지표를 설정하고 시각화하는 방법에 대한 자세한 내용은 [작업 모니터링 및 디버깅](#) 섹션을 참조하세요.

작업 관찰성 지표

이 작업이 실행되면 추가 관찰 가능성 CloudWatch 지표 생성을 활성화하십시오. 자세한 정보는 [the section called “AWS Glue 관찰성 메트릭을 사용한 모니터링”](#)을 참조하세요.

연속 로깅

Amazon에 대한 연속 로깅을 CloudWatch 활성화합니다. 이 옵션이 활성화되지 않으면 작업이 완료된 후에만 로그를 사용할 수 있습니다. 자세한 정보는 [the section called “AWS Glue 작업에 대한 지속 로깅”](#)을 참조하세요.

Spark UI

이 작업 모니터링에 Spark UI 사용을 설정합니다. 자세한 정보는 [AWS Glue 작업을 위한 Apache Spark 웹 UI 사용 설정](#)을 참조하세요.

Spark UI 로그 경로

Spark UI를 사용하도록 설정한 경우 로그를 작성할 경로입니다.

Spark UI 로깅 및 모니터링 구성

다음 옵션 중 하나를 선택합니다:

- 표준: AWS Glue 작업 실행 ID를 파일 이름으로 사용하여 로그를 작성합니다. 콘솔에서 Spark UI 모니터링을 켜십시오. AWS Glue
- 레거시: 'spark-application-{timestamp}'를 파일 이름으로 사용하여 로그를 작성합니다. Spark UI 모니터링을 켜지 마세요.
- 표준 및 레거시: 표준 위치와 레거시 위치 모두에 로그를 작성합니다. 콘솔에서 Spark UI 모니터링을 켜세요. AWS Glue

최대 동시성

이 작업에 허용된 최대 동시 실행 수를 설정합니다. 기본 값은 1입니다. 이 임계값에 도달하면 오류가 반환됩니다. 지정할 수 있는 최대값은 서비스 제한에 따라 통제됩니다. 예를 들어, 새로운 인스턴스가 시작된 후에도 이전 작업이 여전히 실행되고 있다면 오류를 반환하여 동일한 작업의 두 개의 인스턴스를 동시에 실행하지 않도록 방지합니다.

임시 경로

AWS Glue가 스크립트를 실행할 때 임시 중간 결과가 작성된 Amazon S3의 작업 디렉터리의 위치를 제공합니다. 경로의 임시 디렉터리와 동일한 이름의 파일이 없다는 것을 확인합니다. 이 디렉터리는 AWS Glue가 Amazon Redshift에 읽고 쓸 때 및 일부 AWS Glue 변환에 사용됩니다.

Note

리전에 버킷이 아직 없는 경우 AWS Glue에서 작업에 대한 임시 버킷을 생성합니다. 이 버킷은 퍼블릭 액세스를 허용할 수 있습니다. Amazon S3에서 버킷을 수정하여 퍼블릭 액세스 차단 설정하거나 나중에 해당 리전의 모든 작업이 완료된 후 버킷을 삭제할 수 있습니다.

지연 알림 임계값(분)

지연 알림이 전송되기 전까지의 임계값(분)을 설정합니다. RUNNING, STARTING 또는 STOPPING 작업 실행이 예상 시간(분)을 초과할 때 알림을 전송하도록 이 임계값을 설정할 수 있습니다.

보안 구성

목록에서 보안 구성을 선택합니다. 보안 구성은 암호화 없음, AWS KMS관리형 키(SSE-KMS)를 사용한 서버 측 암호화, Amazon S3 관리형 암호화 키(SSE-S3) 등 Amazon S3 대상의 데이터 암호화 방법을 지정합니다.

서버 측 암호화

이 옵션을 선택할 경우, ETL 작업을 Amazon S3에 작성하면 SSE-S3 암호화를 사용하여 데이터는 암호화됩니다. Amazon S3 임시 디렉터리에 쓴 모든 데이터와 Amazon S3 데이터 대상 모두 암호화됩니다. 이 옵션은 작업 파라미터로 전달됩니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [Amazon S3 관리형 암호화 키\(SSE-S3\)로 서버 측 암호화를 사용하여 데이터 보호](#)를 참조하세요.

Important

보안 구성이 지정된 경우 이 옵션은 무시됩니다.

Glue 데이터 카탈로그를 Hive 메타스토어로 사용

이를 선택하여 AWS Glue 데이터 카탈로그를 Hive 메타스토어로 사용합니다. 작업에 사용되는 IAM 역할에는 `glue:CreateDatabase` 권한이 있어야 합니다. "기본값"이라는 데이터베이스가 없는 경우에 데이터 카탈로그에서 생성됩니다.

연결

Virtual Private Cloud(VPC)에 있는 Amazon S3 데이터 소스에 액세스하려면 VPC 구성을 선택합니다. AWS Glue에서 네트워크 연결을 생성하고 관리할 수 있습니다. 자세한 정보는 [데이터에 연결](#)을 참조하세요.

Libraries

Python 라이브러리 경로, 종속 JAR 경로 및 참조된 파일 경로

스크립트에 필요한 경우 이러한 옵션을 지정합니다. 작업을 정의할 때 이러한 옵션에 대해 쉘표로 구분된 Amazon S3 경로를 정의할 수 있습니다. 작업을 실행할 때 이 경로를 무시할 수 있습니다. 자세한 정보는 [사용자 지정 스크립트 제공](#)을 참조하세요.

작업 파라미터

이름이 지정된 파라미터로 스크립트에 전달되는 키-값 쌍의 집합입니다. 이러한 값은 스크립트가 실행될 때 사용되는 기본값이지만 트리거에서 또는 작업을 실행할 때 재정의할 수 있습니다. 키 이름 앞에 --를 붙여야 합니다(예: --myKey). 를 사용할 때는 작업 매개변수를 맵으로 전달합니다. AWS Command Line Interface

예를 들어 [AWS Glue에서 Python 파라미터 전달 및 액세스](#)의 Python 파라미터를 참조하십시오.

Tags

태그 키와 선택 사항인 태그 값으로 작업에 태그를 지정합니다. 생성된 태그 키는 읽기 전용입니다. 일부 리소스에서 태그를 이용하면 리소스를 정리하고 식별하는 데 도움이 됩니다. 자세한 정보는 [AWS Glue의 AWS 태그](#)을 참조하세요.

Lake Formation 관리형 테이블에 액세스하는 작업에 대한 제한 사항

에서 관리하는 테이블에서 읽거나 테이블에 쓰는 작업을 생성할 때는 다음 참고 및 제한 사항을 염두에 두십시오 AWS Lake Formation.

- 셀 수준 필터가 있는 테이블에 액세스하는 작업에서는 다음 기능이 지원되지 않습니다.
 - [작업 북마크 및 제한된 실행](#)
 - [푸시다운 조건자](#)
 - [서버 측 카탈로그 파티션 조건자](#)
 - [enableUpdateCatalog](#)

AWS Glue 콘솔에서 Spark 스크립트 편집

스크립트에는 소스에서 데이터를 추출하고, 변환하고, 대상으로 로드하는 코드가 포함되어 있습니다. AWS Glue 작업을 시작할 때 스크립트를 실행합니다.

AWS Glue ETL 스크립트는 Python 또는 Scala에 코딩될 수 있습니다. Python 스크립트는 추출, 변환, 로드 (ETL) 작업에 PySpark Python 언어를 확장한 언어를 사용합니다. 스크립트는 ETL 변환을 다루는 확장된 구조를 포함합니다. 자동적으로 작업의 소스 코드 논리를 생성하면 스크립트가 생성됩니다. 스크립트를 편집하거나 자체 스크립트를 제공하여 ETL 작업을 실행할 수 있습니다.

AWS Glue에서 스크립트를 정의하고 편집하는 방법에 대한 자세한 내용은 [AWS Glue 프로그래밍 안내서](#) 섹션을 참조하세요.

추가 라이브러리 또는 파일

스크립트가 추가 라이브러리 혹은 파일은 요구하면 다음과 같이 지정합니다.

Python 라이브러리 경로

스크립트에 필요한 Python 라이브러리로 이동하는, 심포로 구분된 Amazon Simple Storage Service(Amazon S3) 경로입니다.

Note

순수 Python 라이브러리만 사용할 수 있습니다. pandas Python 데이터 분석 라이브러리 등 C 확장을 활용하는 라이브러리는 아직 지원되지 않습니다.

종속된 jars 경로

스크립트에 필요한 JAR 파일로 이동하는, 심포로 구분된 Amazon S3 경로입니다.

Note

현재 순수 Java 또는 Scala(2.11) 라이브러리만 사용할 수 있습니다.

참조된 파일 경로

스크립트에 필요한 추가 파일(예: 구성 파일)로 이동하는, 심포로 구분된 Amazon S3 경로입니다.

작업(레거시)

스크립트는 작업을 추출, 변환 및 로드(ETL)하는 코드를 포함합니다. 사용자가 스크립트를 제공하거나 AWS Glue가 사용자의 지시에 따라 스크립트를 생성할 수 있습니다. 스크립트 생성에 대한 자세한 내용은 [사용자 지정 스크립트 제공](#) 섹션을 참조하십시오.

AWS Glue 콘솔의 스크립트를 편집할 수 있습니다. 스크립트를 편집하려면 원본, 대상 및 변환을 추가할 수 있습니다.

스트립트 편집

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다. 그런 다음 [Jobs(작업)] 탭을 선택합니다.

2. 목록에서 작업을 선택한 다음 [Action(작업)]과 [Edit script(스크립트 편집)]을 선택하여 스크립트 에디터를 엽니다.

작업 상세 정보 페이지에서 스크립트 에디터로 액세스할 수 있습니다. [스크립트(Script)] 탭을 선택한 다음 [스크립트 편집(Edit Script)]을 선택합니다.

스크립트 에디터

AWS Glue 스크립트 편집기를 사용하여 스트림에서 원본, 대상 및 변환을 삽입, 수정 및 삭제할 수 있습니다. 스크립트 편집기는 데이터 흐름을 시각화하여 스크립트와 다이어그램을 모두 보여줍니다.

스크립트용 다이어그램을 생성하려면 다이어그램 생성을 선택합니다. AWS Glue는 ##로 시작하는 스크립트 문장을 사용하여 다이어그램을 만듭니다. 다이어그램에서 스크립트를 정확하게 나타내기 위해서 파라미터를 주석에 있도록 하고 Apache Spark 코드와 동기화되도록 합니다.

스크립트 편집기는 커서가 스크립트 어디에 있는지 코드 템플릿을 추가합니다. 편집기 상단에서 다음 옵션을 선택합니다.

- [Source(원본)]을 선택하여 원본 테이블을 스크립트에 추가합니다.
- [Target(대상)]을 선택하여 대상 테이블을 스크립트에 추가합니다.
- [Target location(대상 위치)]을 선택하여 대상 위치를 스크립트에 추가합니다.
- [Transform(변환)]을 선택하여 변환을 스크립트에 추가합니다. 스크립트에 호출된 함수에 대한 자세한 내용은 [에서 AWS Glue ETL 스크립트 프로그래밍 PySpark](#) 단원을 참조하십시오.
- [Spigot(스피곳)]을 선택하여 스피곳 변환을 스크립트에 추가합니다.

삽입된 코드의 경우, 주석과 Apache Spark 코드 모두에서 parameters를 수정합니다. 예를 들어, [Spigot(스피곳)] 변환을 추가하면 path가 @args 주석과 output 코드 라인으로 바뀌었는지 확인합니다.

[Logs(로그)] 탭은 작업이 실행되면서 작업과 관련된 로그를 보여줍니다. 최신 1,000 라인이 표시됩니다.

[스키마(Schema)] 탭에는 선택한 원본 및 대상의 스키마가 표시됩니다(Data Catalog에서 사용 가능한 경우).

처리된 데이터를 작업 북마크로 추적

AWS Glue는 작업 실행의 상태 정보를 유지하여 이전에 ETL 작업을 실행할 때 이미 처리된 데이터를 추적합니다. 이와 같은 지속 상태 정보를 작업 북마크라고 합니다. AWS Glue는 작업 북마크로 상태 정보를 유지하고 이전 데이터의 재처리를 방지합니다. 작업 북마크를 사용하면 예약된 간격으로 재실행 중인 새 데이터를 처리할 수 있습니다. 작업 북마크는 원본, 변환 및 대상과 같은 다양한 작업 요소의 상태로 구성됩니다. 예를 들어, ETL 작업에서 Amazon S3 파일의 새 파티션을 읽어야 할 수 있습니다. AWS Glue는 그 작업에서 처리한 파티션을 추적하여 중복 실행을 방지하고 작업 대상인 데이터 스토어에 데이터를 복제합니다.

작업 북마크는 JDBC 데이터 원본, Relationalize 변환 및 일부 Amazon Simple Storage Service(Amazon S3) 소스에 대해 구현됩니다. 다음 테이블에는 AWS Glue가 작업 북마크에 대해 지원하는 Amazon S3 소스 포맷이 나열됩니다.

AWS Glue 버전	Amazon S3 소스 포맷
버전 0.9	JSON, CSV, Apache Avro, XML
버전 1.0 이상	JSON, CSV, Apache Avro, XML, Parquet, ORC

AWS Glue 버전에 대한 자세한 내용은 [Spark 작업에 대한 작업 속성 정의](#) 섹션을 참조하세요.

AWS Glue 스크립트를 통해 액세스할 경우 작업 북마크 기능에는 추가 기능이 있습니다. 생성된 스크립트를 찾아보면 이 기능과 관련된 변환 컨텍스트가 표시될 수 있습니다. 자세한 내용은 [the section called “작업 북마크 사용”](#) 단원을 참조하십시오.

주제

- [AWS Glue에서 작업 북마크 사용](#)
- [작업 북마크 기능의 운영 세부 정보](#)

AWS Glue에서 작업 북마크 사용

작업이 시작되면 작업 북마크 옵션을 파라미터로 전달합니다. 다음 표에 AWS Glue 콘솔에서 작업 북마크를 설정하기 위한 옵션이 설명되어 있습니다.

작업 북마크	설명
Enable	이전에 처리된 데이터를 추적하기 위해 작업 실행 후 상태를 업데이트하도록 합니다. 작업 북마크가 지원되는 소스의 작업이라면 이미 처리된 데이터를 추적하고, 작업 실행 시 마지막 체크포인트 이후에 받은 새 데이터를 처리합니다.
비활성화	작업 북마크를 사용하지 않고, 작업에서 항상 전체 데이터 세트를 처리합니다. 이전에 실행된 작업의 출력값 관리는 여러분이 직접 수행해야 합니다. 이 값이 기본값입니다.
일시 중지	<p>마지막 북마크의 상태를 업데이트하지 않고 마지막으로 성공한 실행 이후의 증분 데이터 또는 다음 하위 옵션으로 식별된 범위의 데이터를 처리합니다. 이전에 실행된 작업의 출력값 관리는 여러분이 직접 수행해야 합니다. 두 개의 하위 옵션은 다음과 같습니다.</p> <ul style="list-style-type: none"> • <code>job-bookmark-from<시작 값></code>은 마지막으로 성공한 실행 전까지 처리된 모든 입력을 나타내는 실행 ID이며, 지정된 실행 ID를 포함합니다. 해당하는 입력은 무시됩니다. • <code>job-bookmark-to<종료 값></code>는 마지막으로 성공한 실행 전까지 처리된 모든 입력을 나타내는 실행 ID이며, 지정된 실행 ID를 포함합니다. <시작 값>에 의해 식별된 입력을 제외한 해당하는 입력이 작업에서 처리됩니다. 이 입력 이후의 입력도 처리에서 제외됩니다. <p>이 옵션 세트를 지정하면 작업 북마크 상태가 업데이트되지 않습니다.</p> <p>하위 옵션은 선택 사항이지만, 두 하위 옵션이 모두 사용되는 경우에는 제공해야 합니다.</p>

명령줄에서 작업에 전달된 파라미터, 특히 작업 북마크에 대한 자세한 내용은 [AWS Glue 작업 파라미터](#) 단원을 참조하십시오.

Amazon S3 입력 소스의 경우 AWS Glue 작업 북마크는 객체의 마지막 수정 시간을 검사해 어떤 객체를 재처리해야 하는지 확인합니다. 마지막 작업 실행 이후로 입력 소스 데이터가 수정된 경우, 작업을 다시 실행하면 그 파일이 재처리됩니다.

JDBC 소스의 경우 다음 규칙이 적용됩니다.

- 각 테이블에 대해 AWS Glue는 하나 이상의 열을 북마크 키로 사용하여 새 데이터와 처리된 데이터를 결정합니다. 북마크 키가 결합되어 단일 복합 키를 형성합니다.
- AWS Glue에서는 기본적으로 프라이머리 키를 북마크 키로 사용합니다. 단, 해당 키는 간격 없이 순차적으로 증가하거나 감소합니다.
- AWS Glue 스크립트에서 북마크 키로 사용할 열을 지정할 수 있습니다. AWS Glue 스크립트에서 작업 북마크를 사용하는 방법에 대한 자세한 내용은 [the section called “작업 북마크 사용”](#) 섹션을 참조하세요.
- AWS Glue에서는 대소문자를 구분하는 열을 작업 북마크 키로 사용하는 것을 지원하지 않습니다.

AWS Glue Spark ETL 작업에 대한 작업 북마크를 이전 작업 실행으로 되돌릴 수 있습니다. 작업 북마크를 이전의 작업 실행으로 되돌릴 수 있어 후속 작업 실행 시 북마크로 지정된 작업 실행의 데이터만 다시 처리하므로 데이터 다시 채우기 시나리오를 훨씬 효율적으로 지원할 수 있습니다.

모든 데이터를 동일한 작업에서 재처리하려는 경우에는 작업 북마크를 재설정합니다. 작업 북마크 상태를 재설정하려면 AWS Glue 콘솔, [ResetJobBookmark 액션 \(Python: `reset_job_bookmark`\)](#) API 작업 또는 AWS CLI를 사용합니다. 예를 들어, AWS CLI에서는 다음 명령을 입력합니다.

```
aws glue reset-job-bookmark --job-name my-job-name
```

북마크를 되감거나 재설정할 때 여러 대상이 있을 수 있고 대상이 작업 북마크로 추적되지 않기 때문에 AWS Glue는 대상 파일을 정리하지 않습니다. 소스 파일만 북마크로 추적됩니다. 출력에서 중복 데이터를 피하기 위해 소스 파일을 되감고 다시 처리할 때 다른 출력 대상을 생성할 수 있습니다.

AWS Glue는 작업 북마크를 작업별로 추적합니다. 작업을 삭제하면 작업 북마크가 삭제됩니다.

AWS Glue 작업 북마크를 활성화했는데도 ETL 작업이 앞선 실행에서 이미 처리된 데이터를 재처리하는 경우가 있을 수 있습니다. 이러한 오류의 일반적인 원인 해결에 대한 자세한 내용은 [AWS Glue Spark의 오류 문제 해결](#) 단원을 참조하십시오.

작업 북마크 기능의 운영 세부 정보

이 단원에서는 작업 북마크 사용의 실무적 측면에 대해 좀 더 자세히 설명합니다.

작업 북마크는 작업의 상태를 저장합니다. 각각의 상태 인스턴스를 작업 이름과 버전 번호로 입력합니다. 스크립트가 `job.init`를 호출하면 상태를 검색해 항상 최신 버전을 가져옵니다. 상태 안에는 여러

가지 상태 요소가 있는데, 이러한 요소는 해당 스크립트에서 소스, 변환 및 싱크 인스턴스별로 고유합니다. 이러한 상태 요소는 스크립트의 해당 요소(소스, 변환 또는 싱크)에 연결된 변환 컨텍스트로 식별할 수 있습니다. 상태 요소는 사용자 스크립트에서 `job.commit`를 호출할 때 기본적으로 저장됩니다. 이 스크립트는 인수에서 작업 이름과 작업 북마크에 대한 제어 옵션을 가져옵니다.

작업 북마크의 상태 요소는 소스, 변환 또는 싱크 관련 데이터입니다. 예를 들어, 업스트림 작업 또는 프로세스에서 끊임 없이 쓰는 Amazon S3 위치에서 증분 데이터를 읽으려 한다고 가정해 보겠습니다. 이 경우, 스크립트는 지금까지 처리된 항목을 확인해야 합니다. Amazon S3 소스에 대해 작업 북마크를 구현하면 정보를 저장하기 때문에 작업을 다시 실행할 때 저장된 정보를 사용해 새 객체만 필터링하여 작업의 다음 실행을 위해 상태를 재컴퓨팅할 수 있습니다. 타임스탬프는 새 파일을 필터링하는 데 사용됩니다.

작업 북마크에는 상태 요소 외에도 실행 횟수, 시도 횟수 및 버전 번호가 포함되어 있습니다. 실행 횟수는 작업의 실행을 추적하고, 시도 횟수는 작업 실행을 시도한 횟수를 기록합니다. 작업 실행 횟수는 실행에 성공할 때마다 단순히 늘어나는 숫자입니다. 시도 횟수는 각각의 실행 시도를 추적하며, 시도 실패 후 실행하는 경우에만 숫자가 늘어납니다. 버전 번호는 단순히 증가하고, 작업 북마크에 대한 업데이트를 추적합니다.

AWS Glue 서비스 데이터베이스에서 모든 변환에 대한 북마크 상태는 키-값 쌍으로 함께 저장됩니다.

```
{
  "job_name" : ...,
  "run_id": ...,
  "run_number": ..,
  "attempt_number": ...
  "states": {
    "transformation_ctx1" : {
      bookmark_state1
    },
    "transformation_ctx2" : {
      bookmark_state2
    }
  }
}
```

모범 사례

다음은 작업 북마크 사용에 관한 모범 사례입니다.

- 북마크가 사용 설정된 상태에서 데이터 원본 속성을 변경하지 마세요. 예를 들어, Amazon S3 입력 경로 A를 가리키는 `datasource0`가 있으며 작업은 북마크가 사용 설정된 상태에서 여러 라운드 동안

실행 중인 소스에서 읽어오고 있습니다. `transformation_ctx`를 변경하지 않고 `datasource0`의 입력 경로를 Amazon S3 경로 B로 변경하는 경우, AWS Glue 작업은 저장된 이전 북마크 상태를 사용합니다. 그러면 AWS Glue가 해당 파일이 이전 실행에서 처리되었다고 가정하여 입력 경로 B에서 파일이 누락되거나 파일을 건너뛰게 됩니다.

- 더 효과적인 파티션 관리를 위해 북마크가 있는 카탈로그 테이블을 사용하세요. 책갈피는 Data Catalog 또는 옵션의 데이터 원본에 모두 작동합니다. 그러나 옵션 접근 방식을 사용하면 새 파티션을 제거/추가하는 것은 어렵습니다. 크롤러로 카탈로그 테이블을 사용하면 새로 추가된 [파티션](#)을 추적하는 더 향상된 자동화를 제공할 수 있으며, [푸시다운 조건자](#)로 특정 파티션을 선택할 수 있는 유연성이 확보됩니다.
- 대용량 데이터 집합에 대해 [AWS Glue Amazon S3 파일 리스터](#)를 사용하세요. 북마크는 각 입력 파티션 아래의 모든 파일을 나열하고 필터링을 수행하므로 단일 파티션 아래에 파일이 너무 많으면 북마크가 드라이버 OOM으로 실행될 수 있습니다. 메모리에 있는 모든 파일을 한 번에 나열하지 않도록 AWS Glue Amazon S3 파일 리스터를 사용하세요.

Amazon S3의 AWS Glue Spark 셔플 플러그인

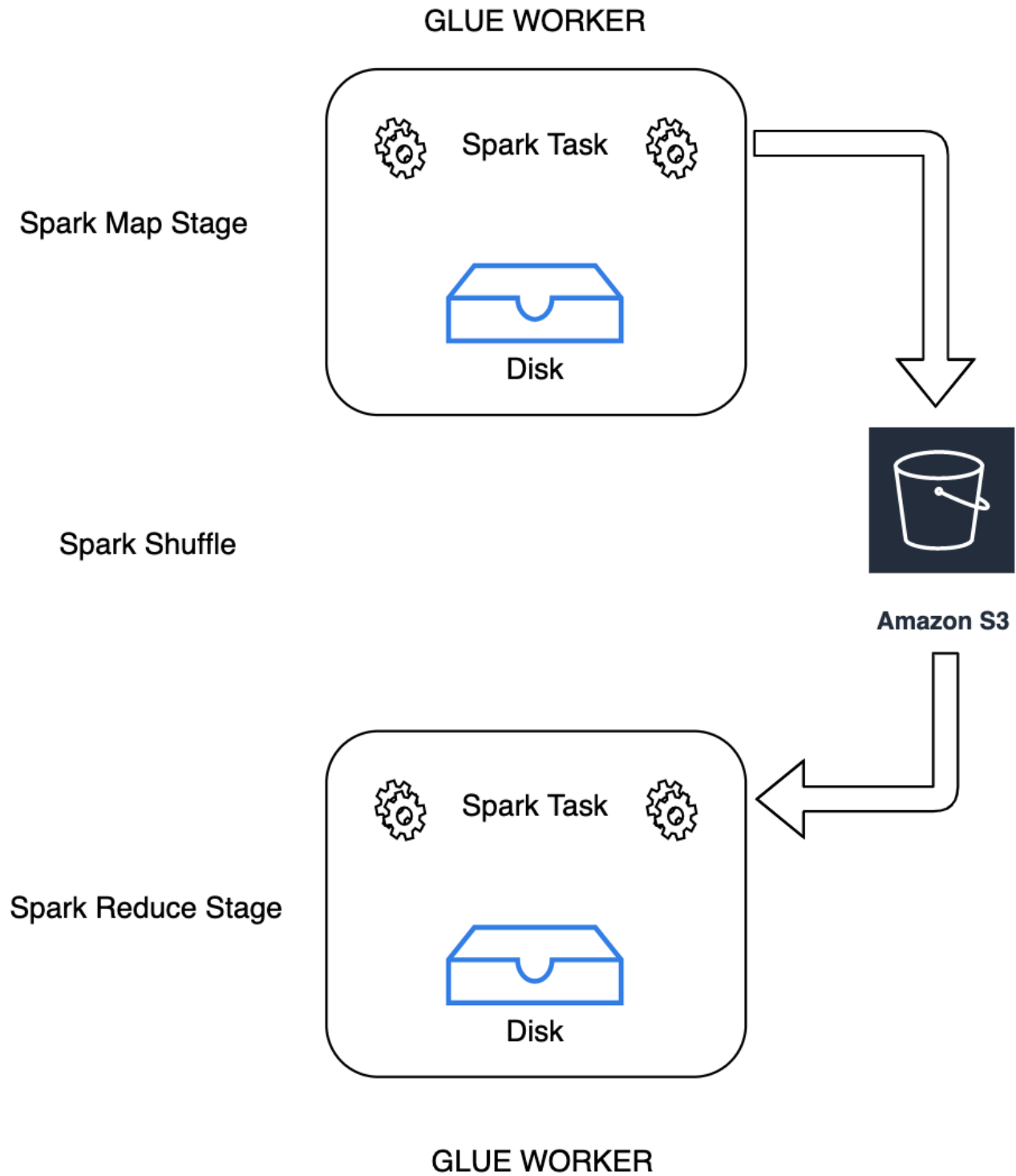
셔플링은 데이터가 파티션 간에 재배열될 때마다 Spark 작업에서 중요한 단계입니다. 이는 `join`, `groupByKey`, `reduceByKey`, `repartition` 등의 광범위한 변환이 처리를 완료하기 위해 다른 파티션의 정보를 필요로 하기 때문에 필요합니다. Spark는 각 파티션에서 필요한 데이터를 수집하여 새 파티션으로 결합합니다. 셔플 중에 데이터가 디스크에 기록되고 네트워크를 통해 전송됩니다. 결과적으로 셔플 작업은 로컬 디스크 용량에 바인딩됩니다. Spark는 실행기에 디스크 공간이 충분하지 않고 복구가 없을 때 `No space left on device` 또는 `MetadataFetchFailedException` 오류를 발생시킵니다.

Note

Amazon S3의 AWS Glue Spark 셔플 플러그인은 AWS Glue ETL 작업에서만 지원됩니다.

Solution

AWS Glue에서는 이제 Amazon S3를 사용하여 Spark 셔플 데이터를 저장할 수 있습니다. Amazon S3는 업계 최고의 확장성, 데이터 가용성, 보안 및 성능을 제공하는 객체 스토리지 서비스입니다. 이 솔루션은 Spark 작업에 대한 컴퓨팅 및 스토리지를 분해하고 완전한 탄력성과 저비용 셔플 스토리지를 제공하여 가장 셔플 집약적인 워크로드를 안정적으로 실행할 수 있도록 합니다.



Amazon S3를 사용하기 위한 새로운 Apache Spark용 클라우드 셔플 스토리지 플러그인을 소개합니다. 대규모 셔플 작업을 위한 로컬 디스크 용량에 의해 제한되는 것으로 알려진 경우 Amazon S3 셔플링을 켜서 AWS Glue 작업을 실패 없이 안정적으로 실행할 수 있습니다. 경우에 따라 Amazon S3에 기

록된 작은 파티션이나 셔플 파일이 많은 경우 Amazon S3로의 셔플링은 로컬 디스크(또는 EBS)보다 약간 느립니다.

클라우드 셔플 스토리지 플러그인 사용을 위한 필수 조건

클라우드 셔플 스토리지 플러그인을 AWS Glue ETL 작업에서 사용하려면 다음이 필요합니다.

- 중간 셔플 및 유출된 데이터를 저장할, 작업 실행과 동일한 리전에 위치한 Amazon S3 버킷. 셔플 스토리지의 Amazon S3 접두사는 다음 예제와 같이 `--conf spark.shuffle.glue.s3ShuffleBucket=s3://shuffle-bucket/prefix/`으로 지정될 수 있습니다.

```
--conf spark.shuffle.glue.s3ShuffleBucket=s3://glue-shuffle-123456789-us-east-1/glue-shuffle-data/
```

- 셔플 관리자는 작업이 완료된 후 파일을 정리하지 않으므로 접두사(예: `glue-shuffle-data`)에서 Amazon S3 스토리지 수명 주기 정책을 설정합니다. 중간 셔플 및 유출된 데이터는 작업 완료 후 삭제해야 합니다. 사용자는 접두사에 간단한 수명 주기 정책을 설정할 수 있습니다. Amazon S3 수명 주기 정책 설정 지침은 Amazon Simple Storage Service Console 사용 설명서의 [버킷에서 수명 주기 구성 설정](#)을 참조하세요.

AWS 콘솔에서 AWS Glue Spark 셔플 관리자 사용

작업을 구성할 때 AWS Glue 콘솔 또는 AWS Glue Studio를 사용하여 AWS Glue Spark 셔플 관리자를 설정하려면 `--write-shuffle-files-to-s3` 작업 파라미터를 선택하여 작업에 대해 Amazon S3 셔플을 설정합니다.

Job parameters

Key	Value - optional
<input type="text" value="--write-shuffle-files-"/>	<input type="text"/>

[Add new parameter](#)

You can add 49 more parameters.

AWS Glue Spark 셔플 플러그인 사용

다음 작업 파라미터는 AWS Glue 셔플 매니저를 설정하고 조정합니다. 이러한 파라미터는 플래그이므로 제공된 값은 고려되지 않습니다.

- `--write-shuffle-files-to-s3` - AWS Glue Spark 셔플 관리자가 Amazon S3 버킷을 사용하여 셔플 데이터를 쓰고 읽을 수 있도록 하는 기본 플래그입니다. 플래그가 지정되지 않은 경우 셔플 관리자가 사용되지 않습니다.
- `--write-shuffle-spills-to-s3` - (AWS Glue 버전 2.0에서만 지원됨). 유출 파일을 Amazon S3 버킷으로 오프로드하도록 허용하는 선택적 플래그로, Spark 작업에 복원력을 추가로 제공합니다. 이는 많은 데이터를 디스크로 유출하는 대규모 워크로드에만 필요합니다. 플래그를 지정하지 않으면 중간 유출 파일이 작성되지 않습니다.
- `--conf spark.shuffle.glue.s3ShuffleBucket=s3://<shuffle-bucket>` - 셔플 파일을 작성하는 Amazon S3 버킷을 지정하는 또 다른 선택적 플래그입니다. 기본값은 `--TempDir /shuffle-data`입니다. AWS Glue 3.0 이상에서는 `--conf spark.shuffle.glue.s3ShuffleBucket=s3://shuffle-bucket-1/prefix,s3://shuffle-bucket-2/prefix`/에서와 같이 쉼표 구분자를 사용해 버킷을 지정함으로써 여러 버킷에 셔플 파일을 쓸 수 있도록 지원합니다. 버킷을 여러 개 사용하면 성능이 향상됩니다.

셔플 데이터의 유휴 암호화를 활성화하려면 보안 구성 설정을 제공해야 합니다. 보안 구성에 대한 자세한 내용은 [the section called “암호화 설정”](#) 섹션을 참조하세요. AWS Glue는 Spark에서 제공하는 다른 모든 셔플 관련 구성을 지원합니다.

클라우드 셔플 스토리지 플러그인을 위한 소프트웨어 바이너리

또한 Apache 2.0 라이선스에 따라 Apache Spark용 클라우드 셔플 스토리지 플러그인의 소프트웨어 바이너리를 다운로드하여 Spark 환경에서 실행할 수 있습니다. 새 플러그인은 Amazon S3에 대한 기본 지원과 함께 제공되며 [Google Cloud Storage 및 Microsoft Azure Blob Storage](#)와 같은 다른 형태의 클라우드 스토리지를 사용하도록 쉽게 구성할 수도 있습니다. 자세한 내용은 [Apache Spark용 클라우드 셔플 스토리지 플러그인](#)을 참조하세요.

참고 및 제한 사항

다음은 AWS Glue 셔플 관리자에 대한 참고 또는 제한 사항입니다.

- AWS Glue 셔플 관리자는 작업이 완료된 후 Amazon S3 버킷에 저장된 (임시) 셔플 데이터 파일을 자동으로 삭제하지 않습니다. 데이터를 보호하려면 클라우드 셔플 스토리지 플러그인을 활성화하기 전에 [클라우드 셔플 스토리지 플러그인 사용을 위한 필수 조건](#)의 지침을 따릅니다.
- 데이터가 왜곡된 경우 이 기능을 사용할 수 있습니다.

Apache Spark용 클라우드 셔플 스토리지 플러그인

클라우드 셔플 스토리지 플러그인은 [ShuffleDataIO API](#)와 호환되는 Apache Spark 플러그인으로, 셔플 데이터를 클라우드 스토리지 시스템(예: Amazon S3)에 저장할 수 있습니다. Spark 애플리케이션에서 일반적으로 join, reduceByKey, groupByKey, repartition과 같은 변환에 의해 트리거되는 대규모 셔플 작업을 위해 로컬 디스크 스토리지 용량을 보충하거나 교체하여 서버리스 데이터 분석 작업 및 파이프라인의 일반적인 장애 또는 가격 대비 성능 문제를 줄일 수 있습니다.

AWS Glue

AWS Glue 버전 3.0 및 4.0에는 플러그인이 미리 설치되어 있으며 추가 단계 없이 Amazon S3로 셔플링할 수 있습니다. 자세한 내용을 보려면 [Amazon S3의 AWS Glue Spark 셔플 클러그인](#)을 참조하여 Spark 애플리케이션에 대한 기능을 활성화하세요.

기타 Spark 환경

플러그인을 사용하려면 다른 Spark 환경에서 다음과 같은 Spark 구성을 설정해야 합니다.

- `--conf spark.shuffle.sort.io.plugin.class=com.amazonaws.spark.shuffle.io.cloud.ChoppedShuffleIO`: 셔플 IO에 이 플러그인을 사용하도록 Spark에 알려줍니다.
- `--conf spark.shuffle.storage.path=s3://bucket-name/shuffle-file-dir`: 셔플 파일이 저장되는 경로입니다.

Note

플러그인은 Spark 코어 클래스 하나를 덮어씁니다. 따라서 플러그인 jar을 Spark jar보다 먼저 로드해야 합니다. 플러그인을 AWS Glue 외부에서 사용하는 경우 온-프레미스 YARN 환경에서 `userClassPathFirst`를 사용하여 이 작업을 수행할 수 있습니다.

Spark 애플리케이션과 플러그인 번들링

Spark 애플리케이션을 로컬에서 개발하는 동안 Maven pom.xml에 플러그인 종속성을 추가하여 플러그인을 Spark 애플리케이션 및 Spark 배포(버전 3.1 이상)와 함께 번들로 제공할 수 있습니다. 플러그인 및 Spark 버전에 대한 자세한 내용은 [플러그인 버전](#) 섹션을 참조하세요.

```
<repositories>
  ...
  <repository>
```

```

    <id>aws-glue-etl-artifacts</id>
    <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/ </url>
  </repository>
</repositories>
...
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>chopper-plugin</artifactId>
  <version>3.1-amzn-LATEST</version>
</dependency>

```

또는 다음과 같이 AWS Glue Maven 아티팩트에서 바이너리를 직접 다운로드하여 Spark 애플리케이션에 포함시킬 수도 있습니다.

```

#!/bin/bash
sudo wget -v https://aws-glue-etl-artifacts.s3.amazonaws.com/release/com.amazonaws/
chopper-plugin/3.1-amzn-LATEST/chopper-plugin-3.1-amzn-LATEST.jar -P /usr/lib/spark/
jars/

```

예제 spark-submit

```

spark-submit --deploy-mode cluster \
--conf spark.shuffle.storage.s3.path=s3://<ShuffleBucket>/<shuffle-dir> \
--conf spark.driver.extraClassPath=<Path to plugin jar> \
--conf spark.executor.extraClassPath=<Path to plugin jar> \
--class <your test class name> s3://<ShuffleBucket>/<Your application jar> \

```

구성 옵션

Amazon S3 셔플 동작을 제어하는 구성 옵션 값이 있습니다.

- `spark.shuffle.storage.s3.enableServerSideEncryption`: 셔플 및 유출 파일에 대한 S3 SSE를 활성화/비활성화합니다. 기본값은 true입니다.
- `spark.shuffle.storage.s3.serverSideEncryption.algorithm`: 사용할 SSE 알고리즘입니다. 기본값은 AES256입니다.
- `spark.shuffle.storage.s3.serverSideEncryption.kms.key`: SSE aws:kms가 활성화된 경우의 KMS 키 ARN입니다.

이러한 구성과 함께 `spark.hadoop.fs.s3.enableServerSideEncryption` 및 기타 환경별 구성과 같은 구성을 설정하여 사용 사례에 적합한 암호화가 적용되도록 해야 할 수도 있습니다.

플러그인 버전

이 플러그인은 각 AWS Glue 버전과 연결된 Spark 버전에서 지원됩니다. 다음 표에는 플러그인 소프트웨어 바이너리의 AWS Glue 버전, Spark 버전 및 Amazon S3 위치와 연결된 플러그인 버전이 나와 있습니다.

AWS Glue 버전	Spark 버전	플러그인 버전	Amazon S3 위치
3.0	3.1	3.1-amzn-LATEST	s3://aws-glue-etl-artifacts/release/com/amazonaws/chopper-plugin/3.1-amzn-0/chopper-plugin-3.1-amzn-LATEST.jar
4.0	3.3	3.3-amzn-LATEST	s3://aws-glue-etl-artifacts/release/com/amazonaws/chopper-plugin/3.3-amzn-0/chopper-plugin-3.3-amzn-LATEST.jar

라이선스

이 플러그인의 소프트웨어 바이너리는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

AWS Glue Spark 작업 모니터링

주제

- [스파크 메트릭스는 다음에서 사용할 수 있습니다. AWS Glue Studio](#)
- [Apache Spark 웹 UI를 사용하여 작업 모니터링](#)
- [AWS Glue 작업 실행 인사이트를 사용한 모니터링](#)
- [Amazon CloudWatch를 사용한 모니터링](#)
- [작업 모니터링 및 디버깅](#)

스파크 메트릭스는 다음에서 사용할 수 있습니다. AWS Glue Studio

[지표(Metrics)] 탭에는 작업이 실행되고 프로파일링이 활성화될 때 수집되는 지표가 표시됩니다. 다음 그래프가 Spark 작업에 표시됩니다.

- ETL 데이터 이동
- 메모리 프로필: 드라이버 및 실행기

다음 그래프를 표시하려면 View additional metrics(추가 측정치 보기)를 선택합니다.

- ETL 데이터 이동
- 메모리 프로필: 드라이버 및 실행기
- 실행기 간의 데이터 셔플
- CPU 부하: 드라이버 및 실행기
- 작업 실행: 활성 실행기, 완료된 단계 및 최대 필요 실행기

측정치를 수집하도록 작업이 구성된 경우 이러한 그래프의 데이터는 CloudWatch 지표로 푸시됩니다. 지표를 설정하고 그래프를 해석하는 방법에 대한 자세한 내용은 [작업 모니터링 및 디버깅](#) 섹션을 참조하세요.

Example ETL 데이터 이동 그래프

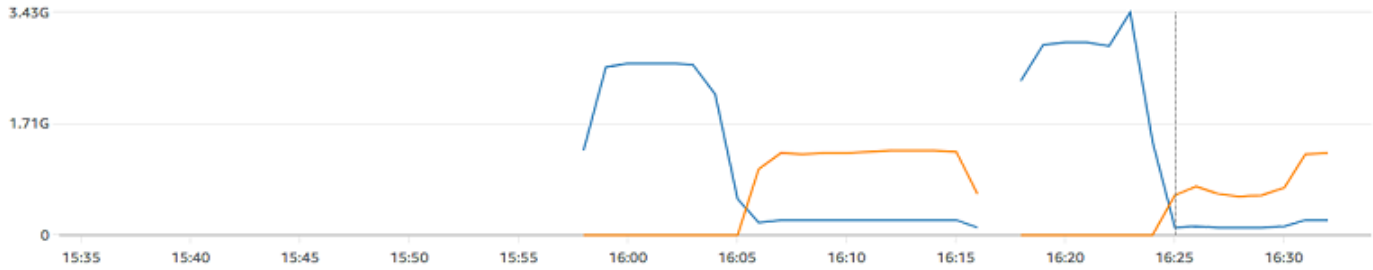
ETL 데이터 이동 그래프는 다음 측정치를 표시합니다.

- 모든 실행기가 Amazon S3에서 읽은 바이트 수 - [glue.ALL.s3.filesystem.read_bytes](#)
- 모든 실행기가 Amazon S3에 쓴 바이트 수 - [glue.ALL.s3.filesystem.write_bytes](#)

Jobs > e2e-straggler

Detailed job metrics

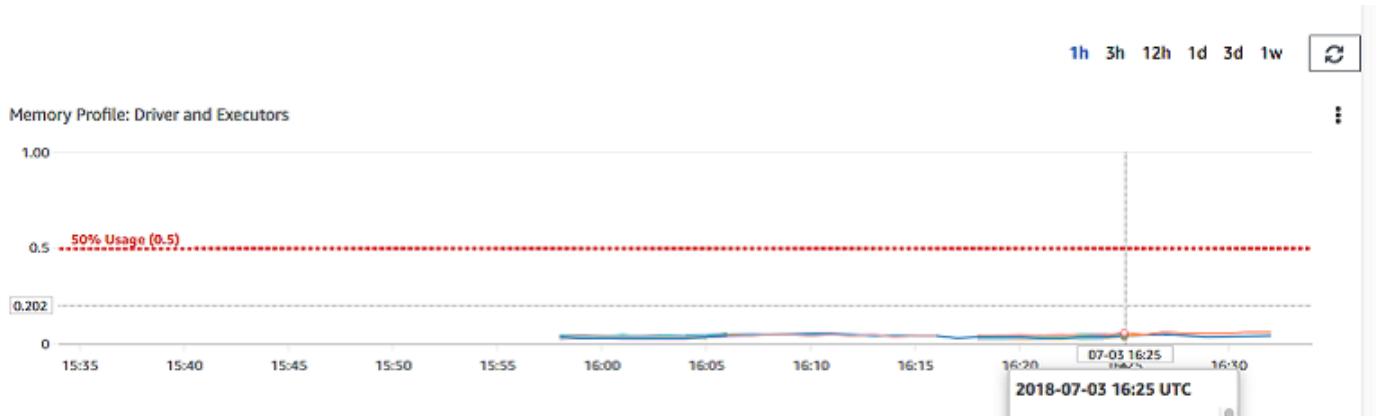
ETL Data Movement



Example 메모리 프로파일 그래프

메모리 프로파일 그래프는 다음 측정치를 표시합니다.

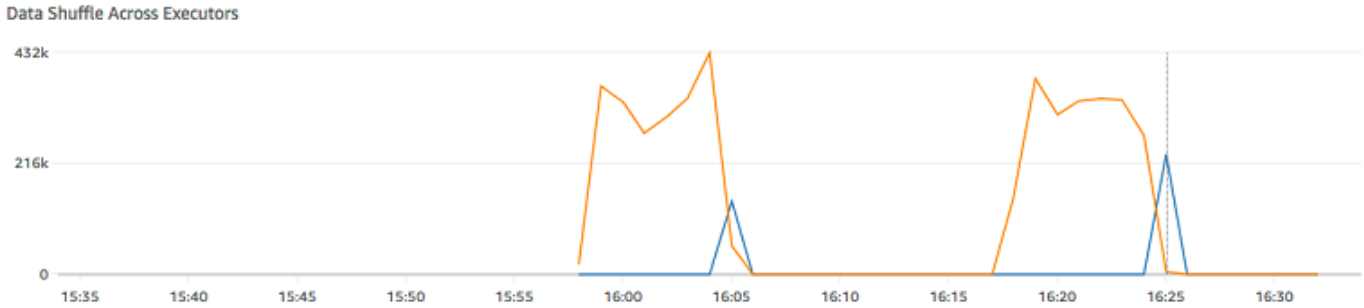
- 드라이버에 의해 이 드라이버용 JVM 힙에 사용되는 메모리 부분(규모: 0~1), executorId에 의해 식별되는 실행기, 또는 모든 실행기 -
 - [glue.driver.jvm.heap.usage](#)
 - [glue.executorId.jvm.heap.usage](#)
 - [glue.ALL.jvm.heap.usage](#)



Example 실행기 간의 데이터 셔플 그래프

실행기 간의 데이터 셔플 그래프는 다음 측정치를 표시합니다.

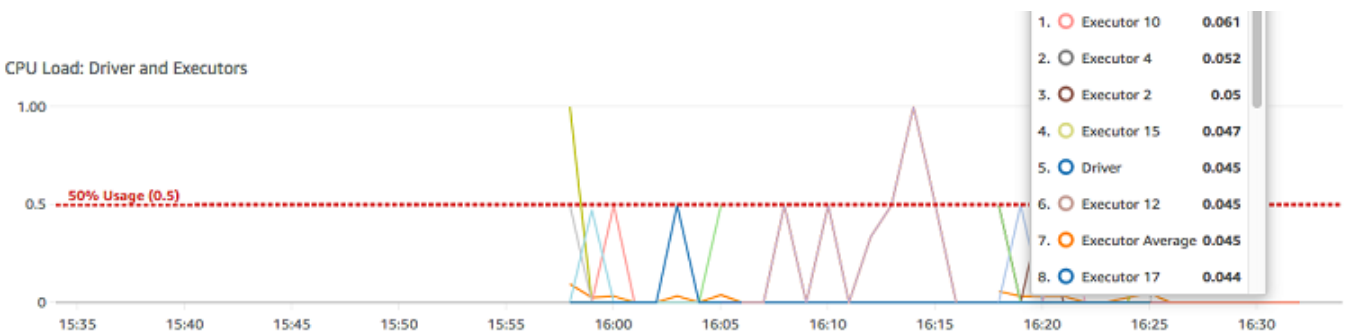
- 서로 간에 데이터를 셔플링하기 위해 모든 실행기가 읽은 바이트 수 - [glue.driver.aggregate.shuffleLocalBytesRead](#)
- 서로 간에 데이터를 셔플링하기 위해 모든 실행기가 쓴 바이트 수 - [glue.driver.aggregate.shuffleBytesWritten](#)



Example CPU 부하 그래프

CPU 부하 그래프는 다음 측정치를 표시합니다.

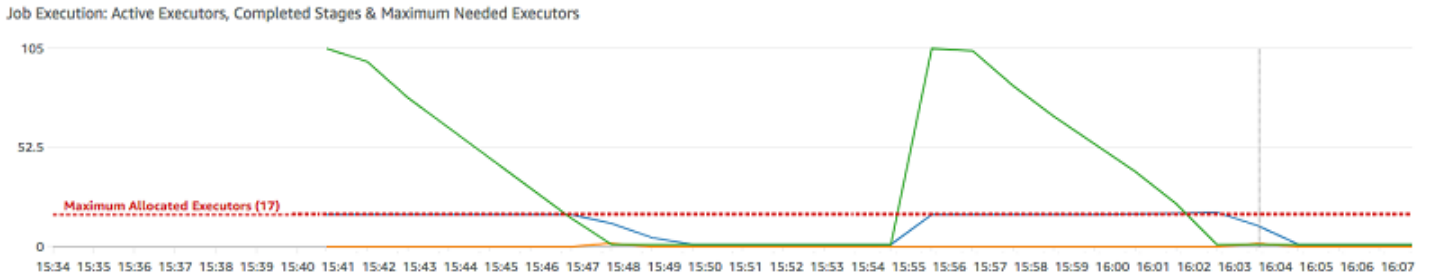
- 드라이버, executorId로 식별되는 실행기 또는 모든 실행기가 사용한 CPU 시스템 로드 부분(규모: 0~1).
 - [glue.driver.system.cpuSystemLoad](#)
 - [glue.executorId.system.cpuSystemLoad](#)
 - [glue.ALL.system.cpuSystemLoad](#)



Example 작업 실행 그래프

작업 실행 그래프는 다음 측정치를 표시합니다.

- 능동적으로 실행 중인 실행기 수 - [glue.driver.ExecutorAllocationManager.executors.numberAllExecutors](#)
- 완료된 단계 수 - [glue.aggregate.numCompletedStages](#)
- 최대 필요 실행기 수 - [glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors](#)



Apache Spark 웹 UI를 사용하여 작업 모니터링

Apache Spark 웹 UI를 사용하여 AWS Glue 작업 시스템에서 실행 중인 AWS Glue ETL 작업과 AWS Glue 개발 엔드포인트에서 실행 중인 Spark 애플리케이션을 모니터링하고 디버그할 수 있습니다. Spark UI를 사용하면 각 작업에 대해 다음을 확인할 수 있습니다.

- 각 Spark 단계의 이벤트 타임라인
- 작업의 방향성 비순환 그래프(DAG)
- SparkSQL 쿼리에 대한 물리적, 논리적 계획
- 각 작업에 대한 기본 Spark 환경 변수

Spark 웹 UI 사용에 대한 자세한 내용은 Spark 설명서의 [웹 UI](#)를 참조하십시오. Spark UI 결과를 해석하여 작업 성능을 개선하는 방법에 [대한 지침은 규범적 지침의 Apache Spark 작업 성능 조정 AWS Glue 모범 사례를](#) 참조하십시오. AWS

콘솔에서 Spark UI를 볼 수 있습니다. AWS Glue 이식 레거시가 아닌 표준 형식으로 생성된 로그를 사용하여 AWS Glue 3.0 이상 버전에서 AWS Glue 작업을 실행할 때 사용할 수 있습니다. 이 형식은 새 작업의 기본값입니다. 0.5GB보다 큰 로그 파일이 있는 경우 AWS Glue 4.0 이상 버전에서의 작업 실행에 대한 롤링 로그 지원을 활성화하여 로그 보관, 분석 및 문제 해결을 단순화할 수 있습니다.

AWS Glue 콘솔이나 AWS Command Line Interface (AWS CLI)를 사용하여 Spark UI를 활성화할 수 있습니다. Spark UI를 사용하면 AWS Glue 개발 엔드포인트의 AWS Glue ETL 작업과 Spark 애플리케이션은 Amazon Simple Storage Service(Amazon S3)에서 지정한 위치에 Spark 이벤트 로그를 백업할

수 있습니다. 작업이 작동 중일 때와 완료된 후 모두 실시간으로 Amazon S3에서 Spark UI를 통해 백업된 이벤트 로그를 사용할 수 있습니다. Amazon S3에 로그가 남아 있는 동안에는 AWS Glue 콘솔의 Spark UI에서 로그를 볼 수 있습니다.

권한

AWS Glue 콘솔에서 Spark UI를 사용하려면 모든 개별 서비스 API를 UseGlueStudio 사용하거나 추가할 수 있습니다. Spark UI를 완전히 사용하려면 모든 API가 필요하지만, 사용자는 세분화된 액세스를 위해 IAM 권한에 해당 서비스 API를 추가하여 SparkUI 기능에 액세스할 수 있습니다.

RequestLogParsing은 로그 구문 분석을 수행하므로 가장 중요합니다. 나머지 API는 각각의 구문 분석된 데이터를 읽는 데 사용됩니다. 예를 들어 GetStages는 Spark 작업의 모든 단계에 대한 데이터에 액세스할 수 있습니다.

UseGlueStudio에 매핑된 Spark UI 서비스 API 목록은 아래 샘플 정책에 나와 있습니다. 아래 정책은 Spark UI 기능만 사용할 수 있는 액세스 권한을 제공합니다. Amazon S3 및 IAM과 같은 권한을 더 추가하려면 [사용자 지정 IAM 정책 생성](#)을 참조하십시오. AWS Glue Studio

UseGlueStudio에 매핑된 Spark UI 서비스 API 목록은 아래 샘플 정책에 나와 있습니다. Spark UI 서비스 API를 사용할 때는 다음 네임스페이스를 사용하세요. glue:<ServiceAPI>

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGlueStudioSparkUI",
      "Effect": "Allow",
      "Action": [
        "glue:RequestLogParsing",
        "glue:GetLogParsingStatus",
        "glue:GetEnvironment",
        "glue:GetJobs",
        "glue:GetJob",
        "glue:GetStage",
        "glue:GetStages",
        "glue:GetStageFiles",
        "glue:BatchGetStageFiles",
        "glue:GetStageAttempt",
        "glue:GetStageAttemptTaskList",
        "glue:GetStageAttemptTaskSummary",
        "glue:GetExecutors",
        "glue:GetExecutorsThreads",
        "glue:GetStorage",

```

```

        "glue:GetStorageUnit",
        "glue:GetQueries",
        "glue:GetQuery"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

제한 사항

- 2023년 11월 20일 이전에 발생한 작업은 레거시 로그 형식이기 때문에 AWS Glue 콘솔의 Spark UI는 사용할 수 없습니다.
- AWS Glue 콘솔의 Spark UI는 스트리밍 작업에서 기본적으로 생성되는 로그와 같은 AWS Glue 4.0용 롤링 로그를 지원합니다. 생성된 모든 롤링 로그 이벤트 파일의 최대 합계는 2GB입니다. 롤링 로그가 지원되지 않는 AWS Glue 작업의 경우 SparkUI에 지원되는 최대 로그 이벤트 파일 크기는 0.5GB입니다.
- VPC만 액세스할 수 있는 Amazon S3 버킷에 저장된 Spark 이벤트 로그에는 서버리스 Spark UI를 사용할 수 없습니다.

예: Apache Spark 웹 UI

이 예제에서는 Spark UI를 사용하여 작업 성과를 이해하는 방법을 보여줍니다. 스크린샷은 자체 관리형 Spark 기록 서버에서 제공하는 Spark 웹 UI를 보여줍니다. 콘솔의 Spark UI에서도 비슷한 보기를 제공합니다. AWS Glue Spark 웹 UI 사용에 대한 자세한 내용은 Spark 설명서의 [웹 UI](#)를 참조하십시오.

다음은 Spark 애플리케이션에서 두 데이터 소스의 데이터를 읽고 조인 변환을 수행하여 Parquet 형식으로 Amazon S3에 쓰는 예제입니다.

```

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import count, when, expr, col, sum, isnull
from pyspark.sql.functions import countDistinct

```

```
from awsglue.dynamicframe import DynamicFrame

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'])

df_persons = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
persons.json")
df_memberships = spark.read.json("s3://awsglue-datasets/examples/us-legislators/all/
memberships.json")

df_joined = df_persons.join(df_memberships, df_persons.id == df_memberships.person_id,
'fullouter')
df_joined.write.parquet("s3://aws-glue-demo-sparkui/output/")

job.commit()
```

다음 DAG 시각화는 이 Spark 작업의 여러 단계를 보여줍니다.

APACHE **Spark** 2.2.1 tape-sparksql-jr_80b2f86d42bfb62... application UI

Jobs Stages Storage Environment Executors SQL

Details for Job 2

Status: SUCCEEDED
Completed Stages: 3

- ▶ Event Timeline
- ▼ DAG Visualization

▶ **Completed Stages (3)**

작업에 대한 다음 이벤트 타임라인은 다양한 Spark 실행기의 시작, 실행 및 종료를 보여줍니다.



- Jobs
- Stages
- Storage
- Environment
- Executors
- SQL

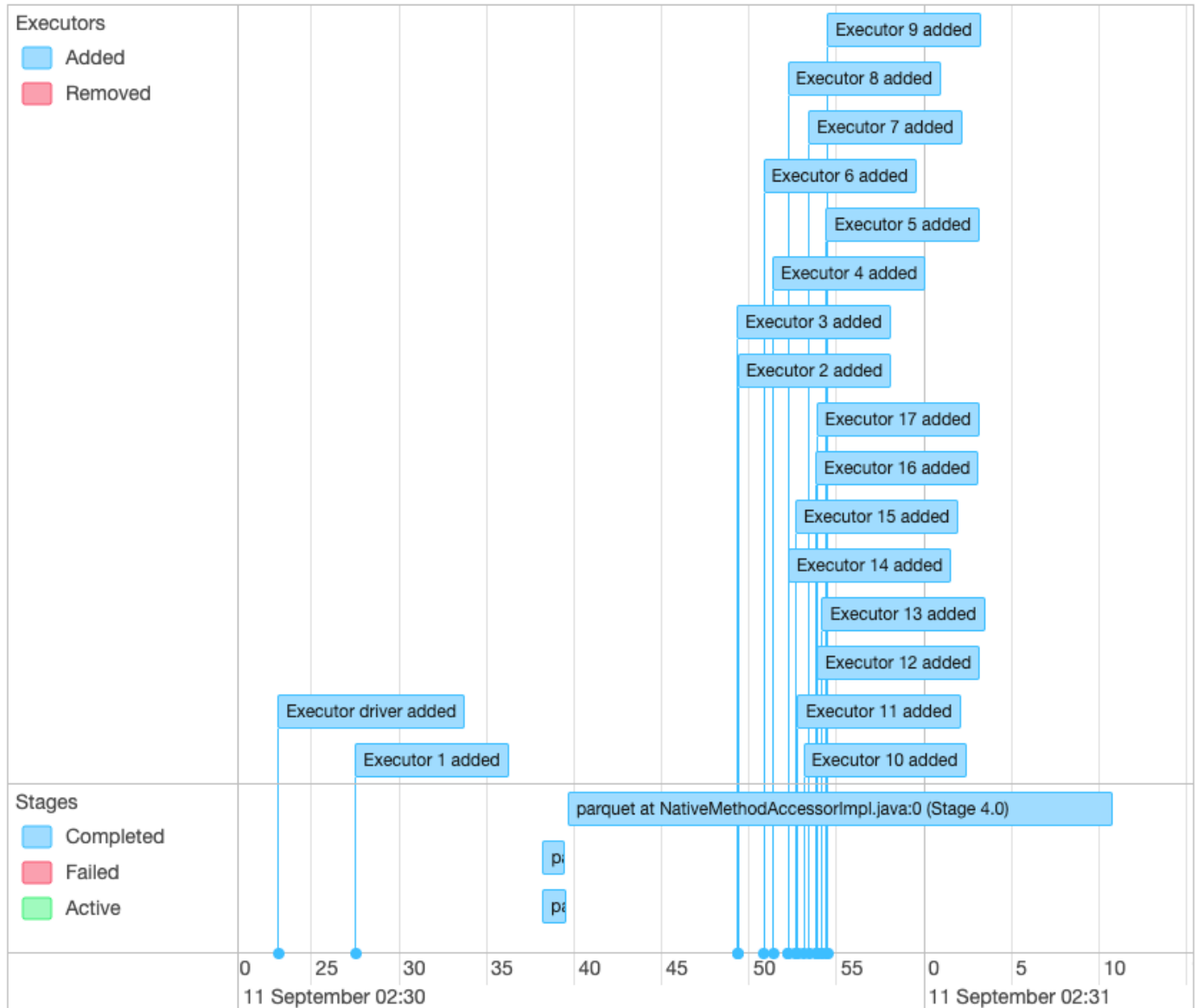
Details for Job 2

Status: SUCCEEDED

Completed Stages: 3

Event Timeline

Enable zooming



▶ DAG Visualization

▶ Completed Stages (3)

다음 화면은 SparkSQL 쿼리 계획의 세부 정보를 보여줍니다.

- 구문 분석된 논리적 계획
- 분석된 논리적 계획
- 최적화된 논리적 계획
- 실행할 물리적 계획



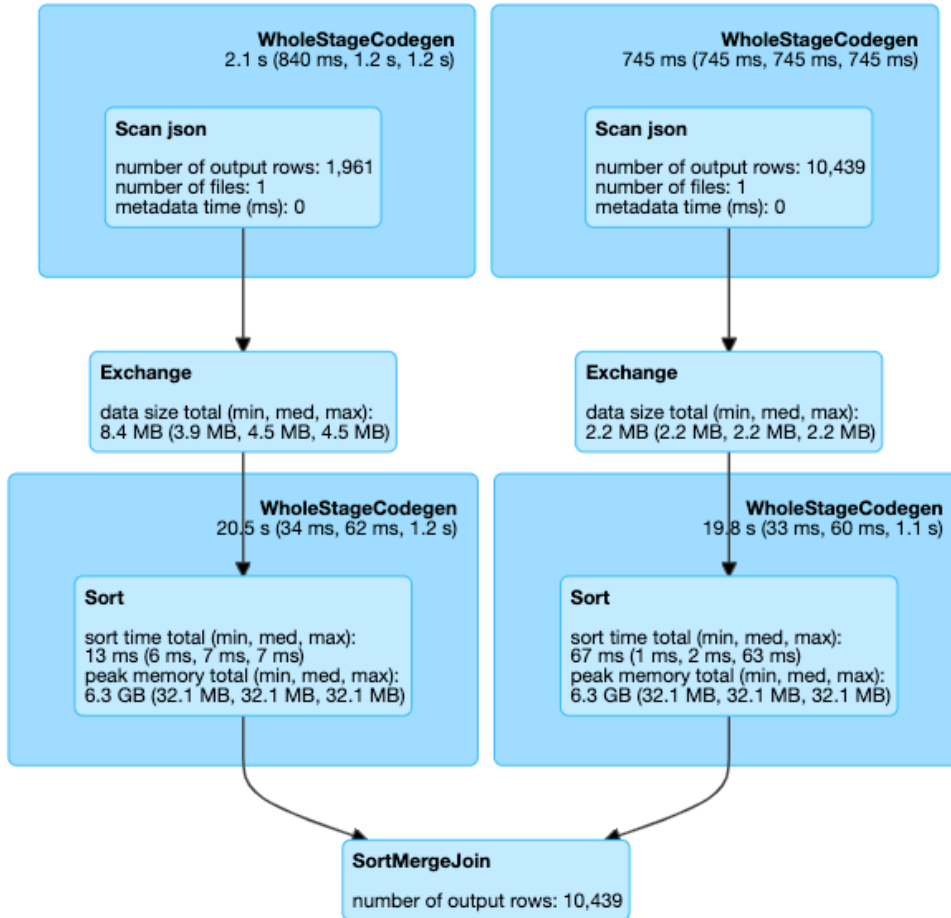
- Jobs
- Stages
- Storage
- Environment
- Executors
- SQL

Details for Query 0

Submitted Time: 2019/09/11 02:30:37

Duration: 34 s

Succeeded Jobs: 2



Details

```

== Parsed Logical Plan ==
Join FullOuter, (id#14 = person_id#50)
:-
Relation[birth_date#8,contact_details#9,death_date#10,family_name#11,gender#12,given_name#13,id#14,identifiers#15
,image#16,images#17,links#18,name#19,other_names#20,sort_name#21] json
+-
Relation[area_id#45,end_date#46,legislative_period_id#47,on_behalf_of_id#48,organization_id#49,person_id#50,role#
51,start_date#52] json

== Analyzed Logical Plan ==
birth_date: string, contact_details: array<struct<type:string,value:string>>, death_date: string, family_name:
string, gender: string, given_name: string, id: string, identifiers:
array<struct<identifier:string,scheme:string>>, image: string, images: array<struct<url:string>>, links:
array<struct<url:string>>, name: string, other_names:
array<struct<lang:string,name:string,note:string>>, sort_name: string, area_id: string, end_date: string,
legislative_period_id: string, on_behalf_of_id: string, organization_id: string, person_id: string, role: string,
start_date: string
Join FullOuter, (id#14 = person_id#50)

```

주제

- [AWS Glue 작업을 위한 Apache Spark 웹 UI 사용 설정](#)
- [Spark 기록 서버 시작](#)

AWS Glue 작업을 위한 Apache Spark 웹 UI 사용 설정

Apache Spark 웹 UI를 사용하여 AWS Glue 작업 시스템에서 실행 중인 AWS Glue ETL 작업을 모니터링하고 디버그할 수 있습니다. AWS Glue 콘솔 또는 AWS Command Line Interface (AWS CLI)를 사용하여 Spark UI를 구성할 수 있습니다.

30초마다 AWS Glue가 Spark 이벤트 로그를 지정한 Amazon S3 경로로 백업합니다.

주제

- [Spark UI 구성\(콘솔\)](#)
- [Spark UI 구성\(AWS CLI\)](#)
- [노트북을 사용하여 세션에 대한 Spark UI 구성](#)
- [롤링 로그 활성화](#)

Spark UI 구성(콘솔)

AWS Management Console을 사용하여 Spark UI를 구성하려면 다음 단계를 따르세요. AWS Glue 작업을 생성할 때 Spark UI는 기본적으로 활성화됩니다.

작업을 생성하거나 편집할 때 Spark UI를 켜려면

1. <https://console.aws.amazon.com/glue/> 에서 AWS Management Console 로그인하고 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서, 작업을 선택합니다.
3. 작업 추가를 선택하거나 기존 작업을 선택합니다.
4. 작업 세부 정보에서 고급 속성을 엽니다.
5. Spark UI 탭에서 Amazon S3에 Spark UI 로그 쓰기를 선택합니다.
6. 작업의 Spark 이벤트 로그를 저장할 Amazon S3 경로를 지정합니다. 작업에서 보안 구성을 사용하는 경우 암호화는 Spark UI 로그 파일에도 적용됩니다. 자세한 정보는 [AWS Glue에서 작성한 데이터 암호화](#)을 참조하세요.
7. Spark UI 로깅 및 모니터링 구성에서:

- AWS Glue 콘솔에서 볼 로그를 생성하려면 표준을 선택합니다.
- Spark 기록 서버에서 볼 로그를 생성하는 경우 레거시를 선택합니다.
- 둘 다 생성하도록 선택할 수도 있습니다.

Spark UI 구성(AWS CLI)

Spark UI로 볼 로그를 AWS CLI 생성하려면 AWS Glue 콘솔에서 를 사용하여 다음 작업 매개변수를 작업에 전달하십시오. AWS Glue 자세한 정보는 [the section called “작업 파라미터”](#)을 참조하세요.

```
'--enable-spark-ui': 'true',
'--spark-event-logs-path': 's3://s3-event-log-path'
```

기존 위치에 로그를 배포하려면 `--enable-spark-ui-legacy-path` 파라미터를 "true"로 설정합니다. 두 가지 형식으로 로그를 생성하지 않으려면 `--enable-spark-ui` 매개변수를 제거하십시오.

노트북을 사용하여 세션에 대한 Spark UI 구성

Warning

AWS Glue 대화형 세션은 현재 콘솔에서 Spark UI를 지원하지 않습니다. Spark 기록 서버를 구성합니다.

AWS Glue 노트북을 사용하는 경우 세션을 시작하기 전에 SparkUI 구성을 설정하십시오. 이렇게 하려면 `%%configure` 셀 매직을 사용합니다.

```
%%configure { "--enable-spark-ui": "true", "--spark-event-logs-path": "s3://path" }
```

롤링 로그 활성화

AWS Glue 작업에 대해 SparkUI 및 롤링 로그 이벤트 파일을 활성화하면 다음과 같은 여러 가지 이점이 있습니다.

- 롤링 로그 이벤트 파일 - 롤링 로그 이벤트 파일을 활성화하면 작업 실행의 각 단계에 대해 별도의 로그 파일이 AWS Glue 생성되므로 특정 단계 또는 변환과 관련된 문제를 쉽게 식별하고 해결할 수 있습니다.

- 향상된 로그 관리 - 롤링 로그 이벤트 파일은 로그 파일을 보다 효율적으로 관리하는 데 도움이 됩니다. 크기가 클 수 있는 단일 로그 파일이 아니라 작업 실행 단계에 따라 로그를 더 작고 관리하기 쉬운 파일로 분할합니다. 이를 통해 로그 보관, 분석 및 문제 해결을 단순화할 수 있습니다.
- 내결함성 향상 - AWS Glue 작업이 실패하거나 중단된 경우 롤링 로그 이벤트 파일이 마지막 성공 단계에 대한 중요한 정보를 제공할 수 있으므로 처음부터 작업을 시작하는 대신 해당 시점부터 작업을 재개하기가 더 쉬워집니다.
- 비용 최적화 - 롤링 로그 이벤트 파일을 활성화하면 로그 파일과 관련된 스토리지 비용을 절감할 수 있습니다. 크기가 클 수 있는 단일 로그 파일을 저장하는 대신 더 작고 관리하기 쉬운 로그 파일을 저장하므로 특히 오래 실행되거나 복잡한 작업의 경우 비용 효율적일 수 있습니다.

새 환경에서 사용자는 다음을 통해 롤링 로그를 명시적으로 활성화할 수 있습니다.

```
'-conf': 'spark.eventLog.rolling.enabled=true'
```

또는

```
'-conf': 'spark.eventLog.rolling.enabled=true -conf
spark.eventLog.rolling.maxFileSize=128m'
```

롤링 로그가 활성화되면 롤오버되기 전 이벤트 로그 파일의 최대 크기를 `spark.eventLog.rolling.maxFileSize` 지정합니다. 이 선택적 매개 변수를 지정하지 않은 경우 기본값은 128MB입니다. 최소값은 10MB입니다.

생성된 모든 롤링 로그 이벤트 파일의 최대 합계는 2GB입니다. 롤링 로그를 지원하지 않는 AWS Glue 작업의 경우 SparkUI에 지원되는 최대 로그 이벤트 파일 크기는 0.5GB입니다.

추가 구성을 전달하여 스트리밍 작업에 대한 롤링 로그를 끌 수 있습니다. 로그 파일이 매우 크면 유지 관리 비용이 많이 들 수 있습니다.

롤링 로그를 끄려면 다음 구성을 제공하십시오.

```
'--spark-ui-event-logs-path': 'true',
'--conf': 'spark.eventLog.rolling.enabled=false'
```

Spark 기록 서버 시작

Spark 기록 서버를 사용하여 자체 인프라에서 Spark 로그를 시각화할 수 있습니다. AWS Glue 콘솔에서는 AWS Glue 4.0 이상 버전의 AWS Glue 작업 실행에 대한 동일한 시각화를 레거시 형식이 아닌 표

준 형식으로 생성된 로그와 함께 볼 수 있습니다. 자세한 내용은 [the section called “Spark UI로 모니터링” 단원을 참조하십시오.](#)

EC2 인스턴스에서 서버를 호스팅하는 AWS CloudFormation 템플릿을 사용하여 Spark 기록 서버를 시작하거나 Docker를 사용하여 로컬로 시작할 수 있습니다.

주제

- [AWS CloudFormation을 사용하여 Spark 기록 서버 시작 및 Spark UI 보기](#)
- [Docker를 사용하여 Spark 기록 서버 시작 및 Spark UI 보기](#)

AWS CloudFormation을 사용하여 Spark 기록 서버 시작 및 Spark UI 보기

AWS CloudFormation 템플릿을 사용하여 Apache Spark 기록 서버를 시작하고 Spark 웹 UI를 볼 수 있습니다. 이 템플릿은 요구 사항에 맞게 수정해야 하는 샘플입니다.

AWS CloudFormation을 사용하여 Spark History Server를 시작하고 Spark UI를 보려면

1. 다음 표에서 스택 시작 버튼 중 하나를 선택하십시오. 그러면 AWS CloudFormation 콘솔에서 스택이 시작됩니다.

지역	시작
미국 동부(오하이오)	
미국 동부(버지니아 북부)	
미국 서부(캘리포니아 북부)	
미국 서부(오레곤)	
아프리카(케이프타운)	
아시아 태평양(홍콩)	
아시아 태평양(뭄바이)	

지역	시작
아시아 태평양(오사카)	Launch Stack
아시아 태평양(서울)	Launch Stack
아시아 태평양(싱가포르)	Launch Stack
아시아 태평양(시드니)	Launch Stack
아시아 태평양(도쿄)	Launch Stack
캐나다(중부)	Launch Stack
유럽(프랑크푸르트)	Launch Stack
유럽(아일랜드)	Launch Stack
유럽(런던)	Launch Stack
유럽(밀라노)	Launch Stack
유럽(파리)	Launch Stack
유럽(스톡홀름)	Launch Stack
중동(바레인)	Launch Stack
남아메리카(상파울루)	Launch Stack

2. Specify template(템플릿 지정) 페이지에서 다음을 선택합니다.

3. 스택 세부 정보 지정 페이지에서 스택 이름을 입력합니다. 파라미터(Parameters) 아래에 추가 정보를 입력합니다.

a. Spark UI 구성

다음 정보를 제공합니다.

- [IP 주소 범위(IP address range)] - Spark UI를 보는 데 사용할 수 있는 IP 주소 범위입니다. 특정 IP 주소 범위의 액세스를 제한하려면 사용자 지정 값을 사용해야 합니다.
- [기록 서버 포트(History server port)] - Spark UI의 포트입니다. 기본 값을 사용할 수 있습니다.
- [이벤트 로그 디렉터리(Event log directory)] - AWS Glue 작업 또는 개발 엔드포인트에서 Spark 이벤트 로그가 저장되는 위치를 선택합니다. 이벤트 로그 경로 체계에 **s3a://**를 사용해야 합니다.
- [Spark 패키지 위치(Spark package location)] - 기본값을 사용할 수 있습니다.
- [키 스토어 경로(Keystore path)] - HTTPS의 SSL/TLS 키 스토어 경로입니다. 사용자 지정 키 스토어 파일을 사용하려면 여기에서 S3 경로 **s3://path_to_your_keystore_file**을 지정하면 됩니다. 이 파라미터를 비워두면 자체 서명된 인증서 기반 키 스토어가 생성되어 사용됩니다.
- 키 스토어 암호(Keystore password) - HTTPS에 대한 SSL/TLS 키 스토어 암호를 입력합니다.

b. EC2 인스턴스 구성

다음 정보를 제공합니다.

- [인스턴스 유형(Instance type)] - Spark 기록 서버를 호스팅하는 Amazon EC2 인스턴스 유형입니다. 이 템플릿은 계정에서 Amazon EC2 인스턴스를 시작하므로 계정에 별도로 Amazon EC2 비용이 청구됩니다.
- [최신 AMI ID(Latest AMI ID)] - Spark 기록 서버 인스턴스에 대한 Amazon Linux 2의 AMI ID입니다. 기본 값을 사용할 수 있습니다.
- [VPC ID] - Spark 기록 서버 인스턴스의 Virtual Private Cloud(VPC) ID입니다. 계정에 제공된 모든 VPC를 사용할 수 있습니다. [기본 네트워크 ACL](#)과 함께 기본 VPC를 사용하지 않는 것이 좋습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [기본 VPC 및 서브넷](#)과 [VPC 생성](#)을 참조하세요.
- [서브넷 ID(Subnet ID)] - Spark 기록 서버 인스턴스의 ID입니다. VPC의 모든 서브넷을 사용할 수 있습니다. 클라이언트에서 서브넷으로 네트워크에 도달할 수 있어야 합니다. 인터넷

을 통해 액세스하려면 라우팅 테이블에 인터넷 게이트웨이가 있는 퍼블릭 서브넷을 사용해야 합니다.

c. 다음을 선택합니다.

4. 스택 옵션 구성(Configure stack options) 페이지에서 현재 사용자 자격 증명을 사용하여 CloudFormation이 스택의 리소스를 생성, 수정 또는 삭제하는 방식을 확인하려면 다음(Next)을 선택합니다. 권한 섹션에서 현재 사용자 권한 대신 사용할 역할을 지정하고 다음을 선택할 수도 있습니다.
5. 검토 페이지에서 설정을 확인합니다.

AWS CloudFormation에서 IAM 리소스를 생성할 수 있음을 승인합니다를 선택하고 스택 생성을 선택합니다.

6. 스택이 생성되기를 기다립니다.
7. 출력 탭을 클릭합니다.
 - a. 퍼블릭 서브넷을 사용하는 경우 SparkUiPublicUrl의 URL을 복사하십시오.
 - b. 프라이빗 서브넷을 사용하는 경우 SparkUiPrivateUrl의 URL을 복사하십시오.
8. 웹 브라우저를 열고 URL에 붙여 넣습니다. 지정된 포트에서 HTTPS를 사용하여 서버에 액세스할 수 있습니다. 브라우저가 서버 인증을 인식하지 못할 수 있어 서버 보호를 재정의하여 실행해야 합니다.

Docker를 사용하여 Spark 기록 서버 시작 및 Spark UI 보기

로컬 액세스를 선호하는 경우(Apache Spark 기록 서버에 EC2 인스턴스가 없는 경우) Docker를 사용하여 Apache Spark 기록 서버를 시작하고 Spark UI를 로컬로 볼 수도 있습니다. 이 Dockerfile은 요구 사항에 맞게 수정해야 하는 샘플입니다.

사전 조건

노트북에 도커를 설치하는 방법에 대한 자세한 내용은 [Docker Engine community](#)를 참조하십시오.

Docker를 사용하여 로컬로 Spark History Server를 시작하고 Spark UI를 보려면

1. GitHub에서 파일을 다운로드하십시오.

[AWS Glue 코드 샘플](#)에서 Dockerfile과 pom.xml을 다운로드합니다.

2. AWS에 액세스하는 데 사용자 자격 증명을 사용할지 또는 페더레이션 사용자 자격 증명을 사용할지 결정합니다.

- 현재 사용자 자격 증명을 사용하여 AWS에 액세스하려면 `docker run` 명령의 `AWS_ACCESS_KEY_ID`와 `AWS_SECRET_ACCESS_KEY`에 사용할 값을 가져옵니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자의 액세스 키 관리](#)를 참조하세요.
 - SAML 2.0 페더레이션 사용자를 사용하여 AWS에 액세스하려면 `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` 및 `AWS_SESSION_TOKEN`의 값을 가져옵니다. 자세한 내용은 [임시 보안 자격 증명 요청](#)을 참조하세요.
3. `docker run` 명령에서 사용할 이벤트 로그 디렉터리의 위치를 결정합니다.
 4. 이름 `glue/sparkui` 및 태그 `latest`를 사용하여 로컬 디렉터리의 파일을 통해 Docker 이미지를 구축합니다.

```
$ docker build -t glue/sparkui:latest .
```

5. Docker 컨테이너를 생성하고 시작합니다.

다음 명령에서는 이전 2단계와 3단계에서 얻은 값을 사용합니다.

- a. 사용자 자격 증명을 사용하여 Docker 컨테이너를 생성하려면 다음과 비슷한 명령을 사용합니다.

```
docker run -itd -e SPARK_HISTORY_OPTS="$SPARK_HISTORY_OPTS -
Dspark.history.fs.logDirectory=s3a://path_to_eventlog
-Dspark.hadoop.fs.s3a.access.key=AWS_ACCESS_KEY_ID -
Dspark.hadoop.fs.s3a.secret.key=AWS_SECRET_ACCESS_KEY"
-p 18080:18080 glue/sparkui:latest "/opt/spark/bin/spark-class
org.apache.spark.deploy.history.HistoryServer"
```

- b. 임시 자격 증명을 사용하여 Docker 컨테이너를 생성하려면 `org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider`를 공급자로 사용하고 2단계에서 얻은 자격 증명 값을 제공합니다. 자세한 내용은 Hadoop: Amazon Web Services와 통합 설명서의 [TemporaryAWSCredentialsProvider로 세션 자격 증명 사용](#)을 참조하세요.

```
docker run -itd -e SPARK_HISTORY_OPTS="$SPARK_HISTORY_OPTS -
Dspark.history.fs.logDirectory=s3a://path_to_eventlog
-Dspark.hadoop.fs.s3a.access.key=AWS_ACCESS_KEY_ID -
Dspark.hadoop.fs.s3a.secret.key=AWS_SECRET_ACCESS_KEY
-Dspark.hadoop.fs.s3a.session.token=AWS_SESSION_TOKEN
-
Dspark.hadoop.fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.TemporaryAWSCred
```

```
-p 18080:18080 glue/sparkui:latest "/opt/spark/bin/spark-class
org.apache.spark.deploy.history.HistoryServer"
```

Note

이러한 구성 파라미터는 [Hadoop-AWS 모듈](#)에서 나옵니다. 사용 사례를 바탕으로 특정 구성을 추가해야 할 수 있습니다. 예를 들어, 격리된 리전에 있는 사용자는 `spark.hadoop.fs.s3a.endpoint`를 구성해야 합니다.

6. 브라우저에서 `http://localhost:18080`을 열어 로컬로 Spark UI를 봅니다.

AWS Glue 작업 실행 인사이트를 사용한 모니터링

AWS Glue 작업 실행 인사이트는 작업의 작업 디버깅 및 최적화를 AWS Glue 간소화하는 기능입니다. AWS Glue [Spark UI](#)와 작업 모니터링을 위한 [CloudWatch 로그 및 지표](#)를 제공합니다. AWS Glue 이 기능을 사용하면 AWS Glue 작업 실행에 대한 다음과 같은 정보를 얻을 수 있습니다.

- 실패가 발생한 AWS Glue 작업 스크립트의 행 번호
- 작업 실패가 발생하기 직전에 Spark 쿼리 계획에서 마지막으로 실행된 Spark 작업
- 실패와 관련된 Spark 예외 이벤트의 시간순으로 된 로그 스트림
- 근본 원인 분석 및 문제 해결을 위한 권장 작업(예: 스크립트 튜닝)
- 일반적인 Spark 이벤트(Spark 작업과 관련된 로그 메시지)와 근본 원인 해결을 위한 권장 작업

AWS Glue 작업 CloudWatch 로그의 두 가지 새로운 로그 스트림을 사용하여 이러한 모든 통찰력을 얻을 수 있습니다.

요구 사항

AWS Glue 작업 실행 인사이트 기능은 AWS Glue 버전 2.0, 3.0, 4.0에서 사용할 수 있습니다. 기존 작업의 [마이그레이션 가이드](#)에 따라 이전 버전의 AWS Glue에서 업그레이드할 수 있습니다.

AWS Glue ETL 작업에 대한 작업 실행 인사이트 활성화

작업 실행 인사이트는 AWS Glue Studio 또는 CLI를 통해 활성화할 수 있습니다.

AWS Glue Studio

AWS Glue Studio를 통해 작업을 생성할 때는 Job Details(작업 세부 정보) 탭에서 작업 실행 인사이트를 활성화 또는 비활성화할 수 있습니다. 작업 인사이트 생성 상자가 선택되어 있는지 확인합니다.

Requested number of workers

The number of workers you want AWS Glue to allocate to this job.

The maximums are 299 for G.1X and 149 for G.2X, and the minimum is 2.

Generate job insights

AWS Glue will analyze your job runs and provide insights on how to optimize your jobs and the reasons for job failures.

명령줄

CLI를 통해 작업을 생성할 때는 새로 추가된 단일 [작업 파라미터](#) `--enable-job-insights = true`를 사용하여 작업 실행을 시작할 수 있습니다.

기본적으로 작업 실행 인사이트 로그 스트림은 [AWS Glue 연속 로깅](#)에서 사용되는 것과 동일한 기본 로그 그룹인 `/aws-glue/jobs/logs-v2/` 아래에 생성됩니다. 연속 로깅에 사용된 것과 동일한 인수 세트를 사용하여 사용자 지정 로그 그룹 이름, 로그 필터 및 로그 그룹 구성을 설정할 수 있습니다. 자세한 내용은 [Enabling Continuous Logging for AWS Glue Jobs](#)(작업에 대해 연속 로깅 활성화)를 참조하세요.

작업 실행 인사이트 로그 스트림에 액세스하려면 CloudWatch

작업 실행 인사이트 기능이 활성화된 상태에서 작업 실행이 실패할 경우 2가지 로그 스트림이 생성될 수 있습니다. 작업이 성공적으로 완료되면 2가지 스트림 모두 생성되지 않습니다.

- 예외 분석 로그 스트림: `<job-run-id>-job-insights-rca-driver`. 이 스트림은 다음과 같은 정보를 제공합니다.
 - 실패의 원인이 된 AWS Glue 작업 스크립트의 행 번호.
 - Spark 쿼리 계획(DAG)에서 마지막으로 실행된 Spark 작업.
 - Spark 드라이버 및 실행기의 예외와 관련 있는 시간순 이벤트. 필요한 경우 전체 오류 메시지, 실패한 Spark 작업과 해당 실행기 ID(특정 실행기의 로그 스트림을 살펴보는 데 사용)와 같은 세부 정보를 볼 수 있습니다.
- 규칙 기반 인사이트 스트림:
 - 근본 원인 분석 및 오류 수정을 위한 권장 사항(예: 특정 작업 파라미터를 사용하여 성능 최적화).
 - 근본 원인 분석의 기반이 되는 관련 있는 Spark 이벤트 및 권장 작업.

Note

첫 번째 스트림은 실패한 작업 실행의 예외 Spark 이벤트가 있는 경우에만 생성되고, 두 번째 스트림은 실패한 작업 실행에 대한 인사이트가 있는 경우에만 생성됩니다. 예를 들어, 작업이 성공적으로 완료되었다면 2가지 스트림 모두 생성되지 않고, 작업이 실패했으나 실패 시나리오와 일치하는 서비스 정의 규칙이 없는 경우에는 첫 번째 스트림만 생성됩니다.

작업이 AWS Glue Studio에서 생성되었다면 Job Run Details(작업 실행 세부 정보) 탭 아래의 Job run insights(작업 실행 인사이트)에 위 스트림의 링크가 각각 'Concise and consolidated error logs'(간결하고 통합된 오류 로그) 및 'Error analysis and guidance'(오류 분석 및 가이드)로 표시됩니다.

Job run - jr_ [REDACTED]

Run details [Info](#) ↻

⊗ An error occurred while calling o134.pyWriteDynamicFrame. No such file or directory 's3://[REDACTED]'

<p>Job name [REDACTED]</p> <p>Start time May 17, 2021 1:10 PM</p> <p>Trigger name -</p> <p>Allocated capacity 10</p> <p>Cloudwatch logs All logs Output logs Error logs</p>	<p>Run status ✔ Success</p> <p>End time May 17, 2021 1:10 PM</p> <p>Last modified on May 17, 2021 1:10 PM</p> <p>Max capacity 10</p> <p>Job run insights Info Concise and consolidated error logs Error analysis and guidance</p>	<p>Glue version 2.0</p> <p>Start-up time 4 seconds</p> <p>Security configuration -</p> <p>Number of workers 10</p>	<p>Recent attempt 2</p> <p>Execution time 1 minute</p> <p>Timeout 2880 minutes</p> <p>Worker type G.1X</p>
---	--	--	--

AWS Glue 작업 실행 인사이트의 예

이 섹션에서는 작업 실행 인사이트 기능을 사용하여 실패한 작업의 문제를 해결하는 방법을 예를 통해 살펴봅니다. 이 예제에서는 사용자가 AWS Glue 작업에서 데이터에 대한 기계 학습 모델을 분석 및 빌드하는 데 필요한 모듈(tensorflow)을 가져오지 않았습니니다.

```
import sys
from awsglue.transforms import *
```

```

from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.types import *
from pyspark.sql.functions import udf,col

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

data_set_1 = [1, 2, 3, 4]
data_set_2 = [5, 6, 7, 8]

scoresDf = spark.createDataFrame(data_set_1, IntegerType())

def data_multiplier_func(factor, data_vector):
    import tensorflow as tf
    with tf.compat.v1.Session() as sess:
        x1 = tf.constant(factor)
        x2 = tf.constant(data_vector)
        result = tf.multiply(x1, x2)
        return sess.run(result).tolist()

data_multiplier_udf = udf(lambda x:data_multiplier_func(x, data_set_2),
    ArrayType(IntegerType(),False))
factoredDf = scoresDf.withColumn("final_value", data_multiplier_udf(col("value")))
print(factoredDf.collect())

```

작업 실행 인사이트 기능이 없다면 작업이 실패해도 Spark에 의해 발생한 다음과 같은 메시지만 볼 수 있습니다.

```
An error occurred while calling o111.collectToPython. Traceback (most recent call last):
```

이 메시지는 명확하지 않으므로 디버깅에 큰 도움이 되지 않습니다. 이 경우 이 기능을 사용하면 두 개의 CloudWatch 로그 스트림에서 추가 통찰력을 얻을 수 있습니다.

1. job-insights-rca-driver 로그 스트림:

- 예외 이벤트: 이 로그 스트림에서 Spark 드라이버 및 여러 분산 작업자로부터 수집된, 실패와 관련 있는 Spark 예외 이벤트를 볼 수 있습니다. 이러한 이벤트를 검토하면 여러 AWS Glue 작업자에 분산된 Spark 작업, 실행기 및 단계에서 결함 있는 코드가 실행되는 과정에서 예외가 전파된 상황을 시간순으로 살펴볼 수 있습니다.
- 행 번호: 이 로그 스트림에서 누락된 Python 모듈 가져오기를 호출하여 실패의 원인이 된 것이 21 번 행이고 스크립트에서 마지막으로 실행된 Spark 작업 collect()의 호출은 24번 행인 것을 알 수 있습니다.

Timestamp	Message
	No older events at this moment. Retry
2022-01-31T06:07:04.750-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisTaskFailed Failure Reason: Traceb...
2022-01-31T06:07:04.870-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisTaskFailed Stage ID: 0, Task ID: ...
2022-01-31T06:07:04.888-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisTaskFailed Stage ID: 0, Task ID: ...
2022-01-31T06:07:04.940-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisTaskFailed Stage ID: 0, Task ID: ...
2022-01-31T06:07:04.998-08:00	22/01/31 14:07:04 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisStageFailed Failure Reason: Job a...
2022-01-31T06:07:05.044-08:00	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueExceptionAnalysisJobFailed Failure Reason: JobFail...
2022-01-31T06:07:05.105-08:00	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Root Cause Analysis Result: line 24 in script jobInsightsDemo... 22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Root Cause Analysis Result: line 24 in script jobInsightsDemo.py.
2022-01-31T06:07:05.427-08:00	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Event: GlueETLJobExceptionEvent Failure Reason: Traceback (mo...
2022-01-31T06:07:05.430-08:00	22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Last Executed Line number from script jobInsightsDemo.py: 33 22/01/31 14:07:05 ERROR GlueExceptionAnalysisListener: [Glue Exception Analysis] Last Executed Line number from script jobInsightsDemo.py: 33

2. job-insights-rule-driver 로그 스트림:

- 근본 원인 분석 및 권장 사항: 스크립트에서 결함이 되는 행 번호와 마지막으로 실행된 행 번호 외에도 이 로그 스트림에서는 근본 원인 분석과 권장 사항(AWS Glue 작업에서 추가 Python 모듈을 사용하려면 AWS Glue 문서에 따라 필요한 작업 파라미터를 설정할 것)을 볼 수 있습니다.
- 기본 이벤트: 이 로그 스트림에서는 근본 원인을 유추할 수 있도록, 서비스 정의 규칙을 사용하여 평가된 Spark 예외 이벤트를 볼 수 있으며 권장 사항을 제공합니다.

2022-01-31T06:07:05.499-08:00	22/01/31 14:07:05 ERROR Analyzer: 2022-01-31 14:07:05,499 ERROR [pool-2-thread-1] app.GlueJobAnalyzerApp\$ (Logging.scala:logError(9)) - [Glue ... 22/01/31 14:07:05 ERROR Analyzer: 2022-01-31 14:07:05,499 ERROR [pool-2-thread-1] app.GlueJobAnalyzerApp\$ (Logging.scala:logError(9)) - [Glue Insights]
	<pre> { "details": { "time": 1643638025489, "rootCauseAnalysis": "Module that is referenced in Glue job was not found.", "action": "Include all modules used in Glue job, refer documentation on how to include external modules, https://aws.amazon.com/premiumsupport/knowledge-center/glue-version2-external-python-libraries/" }, "cause": { "module": "data_multiplier_func", "issue": "ModuleNotFoundError: No module named 'tensorflow'", "fileName": "jobInsightsDemo.py", "lineOfCode": 24 }, "basis": [{ "event": { "timestamp": 1643638024940, "failureReason": "Traceback (most recent call last):\n File \"/opt/amazon/spark/python/lib/pyspark.zip/pyspark/worker.py", line 377, in main\n process()\n File \"/opt/amazon/spark/python/lib/pyspark.zip/pyspark/worker.py", line 372, in process\n serializer.dump_stream(func(split_index, iterator),\n outfile)\n File \"/opt/amazon/spark/python/lib/pyspark.zip/pyspark/serializers.py", line 345, in dump_stream\n self.serializer.dump_stream(self._batched(iterator), stream)\n File \"/opt/amazon/spark/python/lib/pyspark.zip/pyspark/serializers.py", line 141, in dump_stream\n for obj in iterator:\n File \"/opt/amazon/spark/python/lib/pyspark.zip/pyspark/serializers.py", line 334, in _batched\n for item in iterator:\n File\n "<string>", line 1, in <lambda>\n File \"/opt/amazon/spark/python/lib/pyspark.zip/pyspark/worker.py", line 85, in <lambda>\n return lambda *a: f(*a)\n File\n \"/opt/amazon/spark/python/lib/pyspark.zip/pyspark/util.py", line 99, in wrapper\n return f(*args, **kwargs)\n File \"/tmp/jobInsightsDemo.py", line 31, in\n <lambda>\n File \"/tmp/jobInsightsDemo.py", line 24, in data_multiplier_func\nModuleNotFoundError: No module named 'tensorflow'\n", "stackTrace": [{ "declaringClass": "data_multiplier_func", "methodName": "ModuleNotFoundError: No module named 'tensorflow'", "fileName": "/tmp/jobInsightsDemo.py", "lineNumber": 24 }] } }] } </pre>

Amazon CloudWatch를 사용한 모니터링

AWS Glue에서 원시 데이터를 수집하여 읽기 가능하며 실시간에 가까운 지표로 처리하는 Amazon CloudWatch를 통해 AWS Glue를 모니터링할 수 있습니다. 이러한 통계는 2주간 기록되므로 기록 정보에 액세스하여 웹 애플리케이션 또는 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 기본적으로 AWS Glue 지표 데이터는 CloudWatch에 자동으로 전송됩니다. 자세한 내용은 [AWS Glue 지표](#) 및 Amazon CloudWatch User Guide의 [What Is Amazon CloudWatch?](#)를 참조하세요.

연속 로깅

AWS Glue에서도 AWS Glue 작업에 대한 실시간 지속 로깅을 지원합니다. 작업에 대한 지속 로깅이 활성화되어 있으면 AWS Glue 콘솔 또는 CloudWatch 콘솔 대시보드에서 실시간 로그를 볼 수 있습니다. 자세한 내용은 [AWS Glue 작업에 대한 지속 로깅](#) 단원을 참조하십시오.

관찰성 메트릭

작업 관찰성 지표가 활성화되면 작업이 실행될 때 추가 Amazon CloudWatch 지표가 생성됩니다. AWS Glue 관찰성 메트릭을 사용하면 AWS Glue 내부에서 일어나는 일에 대한 통찰력을 얻어 문제의 분류 및 분석을 개선할 수 있습니다.

주제

- [Amazon CloudWatch 지표를 사용하여 AWS Glue 모니터링](#)
- [AWS Glue 작업 프로파일에서 Amazon CloudWatch 경보 설정](#)
- [AWS Glue 작업에 대한 지속 로깅](#)
- [AWS Glue 관찰성 메트릭을 사용한 모니터링](#)

Amazon CloudWatch 지표를 사용하여 AWS Glue 모니터링

AWS Glue 작업 프로파일러를 사용하여 AWS Glue 작업을 프로파일링하고 모니터링할 수 있습니다. AWS Glue 작업에서 원시 데이터를 수집한 후 Amazon CloudWatch에 저장된 판독이 가능한 지표로 실시간에 가깝게 처리합니다. 이러한 통계는 CloudWatch에서 보관 및 집계되므로 기록 정보에 액세스하여 애플리케이션이 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다.

Note

작업 지표를 활성화하고 CloudWatch 사용자 지정 지표를 생성하면 추가 요금이 발생할 수 있습니다. 자세한 내용은 [Amazon CloudWatch 요금](#)을 참조하세요.

AWS Glue 지표 개요

AWS Glue는 사용자와 상호 작용할 때 지표를 CloudWatch로 전송합니다. AWS Glue 콘솔(기본 방법), CloudWatch 콘솔 대시보드 또는 AWS Command Line Interface(AWS CLI)를 사용하여 이러한 지표를 볼 수 있습니다.

AWS Glue 콘솔 대시보드를 사용하여 지표를 보려면

작업에 대한 요약이나 세부 그래프, 또는 작업 실행에 대한 세부 그래프를 볼 수 있습니다.

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 작업 실행 모니터링을 선택합니다.
3. 작업 실행에서 현재 실행 중인 작업을 중지하거나, 작업을 보거나, 작업 북마크를 되돌립니다.
4. 작업을 선택한 다음 실행 세부 정보 보기를 선택하여 작업 실행에 대한 추가 정보를 확인합니다.

CloudWatch 콘솔 대시보드를 사용하여 지표를 보려면

지표는 먼저 서비스 네임스페이스별로 그룹화된 다음 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

1. <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 지표를 선택합니다.
3. Glue 네임스페이스를 선택합니다.

AWS CLI을(를) 사용하여 지표를 보려면

- 명령 프롬프트에서 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace Glue
```

AWS Glue는 30초마다 CloudWatch에 지표를 보고하며, CloudWatch 지표 대시보드는 1분마다 이러한 지표를 표시하도록 구성되어 있습니다. AWS Glue 지표는 이전에 보고한 값의 델타 값을 나타냅니다. 적절한 경우 지표 대시보드는 30초 값을 집계(합)하여 마지막 1분 전체에 대한 값을 얻습니다.

Spark 작업에 대한 AWS Glue 지표 동작

AWS Glue 지표는 스크립트에서 `GlueContext`를 초기화할 때 활성화되며 일반적으로 Apache Spark 작업이 끝날 때만 업데이트됩니다. 이러한 지표는 지금까지 완료된 모든 Spark 작업의 집계 값을 나타냅니다.

그러나 AWS Glue가 CloudWatch로 전달하는 Spark 지표는 일반적으로 보고되는 시점의 현재 상태를 나타내는 절대 값입니다. AWS Glue는 이 지표를 30초마다 CloudWatch로 보고하며, 일반적으로 지표 대시보드에는 마지막 1분 동안 받은 데이터 포인트의 평균이 표시됩니다.

AWS Glue 지표 이름은 모두 다음 접두사 유형 중 하나로 시작합니다.

- `glue.driver.` – 이 접두사로 시작하는 이름의 지표는 Spark 드라이버의 모든 실행기에서 집계된 AWS Glue 지표 또는 Spark 드라이버에 해당하는 Spark 지표를 나타냅니다.
- `glue.executorId.` – `executorId`는 특정 Spark 실행기의 번호입니다. 이는 로그에 나열된 실행기에 해당됩니다.
- `glue.ALL.` – 이름이 이 접두사로 시작하는 지표는 모든 Spark 실행기의 값을 집계합니다.

AWS Glue 지표

AWS Glue는 30초 간격으로 다음 지표를 프로파일링하여 CloudWatch로 전송하며, AWS Glue 지표 대시보드는 1분마다 이들 지표를 보고합니다.

지표	설명
<code>glue.driver.aggregate.bytesRead</code>	<p>모든 실행기에서 실행되는 모든 완료된 Spark 태스크가 모든 데이터 원본에서 읽은 바이트 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 바이트</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 읽은 바이트 수.

지표	설명
	<ul style="list-style-type: none"> • 작업 진행 상황. • JDBC 데이터 원본. • 작업 북마크 문제. • 작업 실행 간 편차. <p>이 지표는 <code>glue.ALL.s3.filesystem.read_bytes</code> 지표와 동일한 방식으로 사용할 수 있으며 (차이는 이 지표가 Spark 작업 종료 시 업데이트된다는 것) 비 S3 데이터 원본도 캡처합니다.</p>
<code>glue.driver.aggregate.elapsedTime</code>	<p>밀리초 단위의 ETL 경과 시간(작업 부트스트랩 시간은 포함하지 않음).</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 밀리초</p> <p>작업 실행이 평균적으로 얼마나 오래 실행되는지 측정하는 데 사용할 수 있습니다.</p> <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • 스트래글러에 경보를 설정합니다. • 작업 실행 간 편차를 측정합니다.

지표	설명
<code>glue.driver.aggregate.numCompletedStages</code>	<p>작업에서 완료된 단계 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> 작업 진행 상황. 다른 지표와 상관관계가 계산되는 경우, 작업 실행의 단계별 타임라인. <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> 작업 실행에서 가장 까다로운 단계를 식별합니다. 여러 작업 실행에서 상관관계가 있는 스파이크(까다로운 단계)에 대해 경보를 설정합니다.

지표	설명
<code>glue.driver.aggregate.numCompletedTasks</code>	<p>작업에서 상관관계가 있는 태스크 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 작업 진행 상황. • 특정 단계 내 병렬화.
<code>glue.driver.aggregate.numFailedTasks</code>	<p>실패한 태스크 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 작업 태스크가 실패한 원인인 데이터 이상 현상. • 작업 태스크가 실패한 원인인 클러스터 이상 현상. • 작업 태스크가 실패한 원인인 스크립트 이상 현상. <p>이 데이터를 사용하여 데이터, 클러스터 또는 스크립트 이상을 암시하는 실패 증가에 대해 경보를 설정할 수 있습니다.</p>

지표	설명
<code>glue.driver.aggregate.numKilledTasks</code>	<p>중지된 태스크 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 태스크가 중지된 예외(OOM)가 발생한 데이터 스쿠 이상 현상. • 태스크가 중지된 예외(OOM)가 발생한 스크립트 이상 현상. <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • 데이터 이상 현상을 나타내는 실패 증가에 대한 경보를 설정합니다. • 클러스터 이상 현상을 나타내는 실패 증가에 대한 경보를 설정합니다. • 스크립트 이상 현상을 나타내는 실패 증가에 대한 경보를 설정합니다.

지표	설명
<code>glue.driver.aggregate.recordsRead</code>	<p>모든 실행기에서 실행되는 모든 완료된 Spark 작업이 모든 데이터 원본에서 읽은 레코드 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 읽은 레코드 수. • 작업 진행 상황. • JDBC 데이터 원본. • 작업 북마크 문제. • 며칠에 걸친 작업 실행의 스큐. <p>이 지표는 <code>glue.ALL.s3.filesystem.read_bytes</code> 지표와 동일한 방식으로 사용할 수 있으며, 차이는 이 지표가 Spark 작업 종료 시 업데이트된다는 것입니다.</p>

지표	설명
<code>glue.driver.aggregate.shuffleBytesWritten</code>	<p>이전 보고 이후 데이터 셔플을 위해 모든 실행기가 기록한 바이트 수(AWS Glue 지표 대시보드가 이 목적으로 이전 1분 동안 기록된 바이트 수로 집계).</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 바이트</p> <p>다음을 모니터링하는 데 사용: 작업(대규모 joins, groupBy, repartition, coalesce) 내 데이터 셔플.</p> <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • 대규모 입력 파일을 추가 처리 전에 재파티셔닝 또는 압축 해제합니다. • 데이터를 보다 균일하게 다시 파티셔닝하여 핫 키를 방지합니다. • joins 또는 groupBy 작업 전에 데이터를 사전 필터링합니다.

지표	설명
<code>glue.driver.aggregate.shuffleLocalBytesRead</code>	<p>이전 보고 이후 데이터 셔플을 위해 모든 실행기가 읽은 바이트 수(AWS Glue 지표 대시보드가 이 목적으로 이전 1분 동안 읽은 바이트 수로 집계).</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다.</p> <p>단위: 바이트</p> <p>다음을 모니터링하는 데 사용: 작업(대규모 joins, groupBy, repartition, coalesce) 내 데이터 셔플.</p> <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • 대규모 입력 파일을 추가 처리 전에 재파티셔닝 또는 압축 해제합니다. • 핫 키를 사용하여 데이터를 보다 균일하게 다시 파티셔닝합니다. • joins 또는 groupBy 작업 전에 데이터를 사전 필터링합니다.

지표	설명
<code>glue.driver.BlockManager.disk.diskSpaceUsed_MB</code>	<p>모든 실행기에서 사용된 디스크 공간의 메가바이트 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(게이지).</p> <p>유효 통계: Average. 이 Spark 지표는 절대값으로 보고됩니다.</p> <p>단위: 메가바이트</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 캐싱된 RDD 파티션을 나타내는 블록에 사용된 디스크 공간. • 중간 셔플 출력을 나타내는 블록에 사용된 디스크 공간. • 브로드캐스트를 나타내는 블록에 사용된 디스크 공간. <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • 디스크 사용량 증가로 인한 작업 실패를 식별합니다. • 분산 또는 셔플이 발생하는 대규모 파티션을 식별합니다. • 이 문제를 해결하려면 프로비저닝된 DPU 용량을 증가시킵니다.

지표	설명
<pre>glue.driver.ExecutorAllocationManager.executors.numberAllExecutors</pre>	<p>능동적으로 실행 중인 실행기 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(게이지).</p> <p>유효 통계: Average. 이 Spark 지표는 절대값으로 보고됩니다.</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 작업 활동. • 스트래글링 실행기(몇 개의 실행기만 실행 중) • 현재의 실행기 수준 병렬화. <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • 클러스터 활용도가 저하되기 전에 대규모 입력 파일을 재파티셔닝 또는 압축 해제합니다. • 스트래글러 시나리오로 인한 단계 또는 작업 예외 지연을 식별합니다. • 추가 DPU를 프로비저닝해야 하는 백로그를 이해하기 위해 numberMaxNeededExecutors와 비교합니다.

지표	설명
<pre>glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors</pre>	<p>현재 로드를 충족하는 데 필요한 최대 작업 (능동 실행 및 보류) 실행기 수.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(게이지).</p> <p>유효한 통계: Maximum. 이 Spark 지표는 절대값으로 보고됩니다.</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> 작업 활동. DPU 용량 또는 중지/실패한 실행기로 인해 가용 실행기가 없어 아직 예약되지 않은 보류 태스크의 현재 실행기 수준 병렬화 및 백로그. <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> 예약 대기열의 보류/백로그를 식별합니다. 스트래글러 시나리오로 인한 단계 또는 작업 예외 지연을 식별합니다. 추가 DPU를 프로비저닝해야 하는 백로그를 이해하기 위해 numberAllExecutors와 비교합니다. 보류 실행기 백로그를 교정하려면 프로비저닝된 DPU 용량을 증가시킵니다.

지표	설명
<code>glue.driver.jvm.heap.usage</code>	드라이버, <code>executorId</code> 로 식별되는 실행기 또는 모든 실행기에 대해 이 드라이버용 JVM 힙이 사용하는 메모리 부분(규모: 0-1).
<code>glue.executorId.jvm.heap.usage</code>	유효 차원: <code>JobName</code> (AWS Glue 작업 이름), <code>JobRunId</code> (<code>JobRun ID</code> . 또는 <code>ALL</code>), <code>Type</code> (게이지).
<code>glue.ALL.jvm.heap.usage</code>	<p>유효 통계: <code>Average</code>. 이 Spark 지표는 절대값으로 보고됩니다.</p> <p>단위: 퍼센트</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> <code>glue.driver.jvm.heap.usage</code> 를 사용하여 드라이버 메모리 부족 상태(OOM). <code>glue.ALL.jvm.heap.usage</code> 를 사용하여 실행기 메모리 부족 상태(OOM). <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> 메모리 사용 실행기 ID 및 단계를 식별합니다. 스트래글링 실행기 ID 및 단계를 식별합니다. 드라이버 메모리 부족(OOM) 상태를 식별합니다. 실행기 메모리 부족(OOM) 상태를 식별하고 실행기 로그에서 스택 트레이스를 가져올 수 있도록 해당 실행기 ID를 확인합니다. 스트래글러 또는 메모리 부족 상태(OOM)가 발생하는 데이터 스큐를 가질 수 있는 파일 또는 파티션을 식별합니다.

지표	설명
glue.driver.jvm.heap.used	드라이버, executorId로 식별되는 실행기 또는 모든 실행기에 대해 JVM 힙이 사용하는 메모리 바이트 수.
glue.executorId.jvm.heap.used	유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(게이지).
glue.ALL.jvm.heap.used	<p>유효 통계: Average. 이 Spark 지표는 절대값으로 보고됩니다.</p> <p>단위: 바이트</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 드라이버 메모리 부족 상태(OOM). • 실행기 메모리 부족 상태(OOM). <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • 메모리 사용 실행기 ID 및 단계를 식별합니다. • 스트래글링 실행기 ID 및 단계를 식별합니다. • 드라이버 메모리 부족(OOM) 상태를 식별합니다. • 실행기 메모리 부족(OOM) 상태를 식별하고 실행기 로그에서 스택 트레이스를 가져올 수 있도록 해당 실행기 ID를 확인합니다. • 스트래글러 또는 메모리 부족 상태(OOM)가 발생하는 데이터 스큐를 가질 수 있는 파일 또는 파티션을 식별합니다.

지표	설명
<code>glue.driver.s3.filesystem.read_bytes</code>	<p>이전 보고 이후 드라이버, <code>executorId</code>로 식별되는 실행기 또는 모든 실행기가 Amazon S3에서 읽은 바이트 수(AWS Glue 지표 대시보드가 이 목적으로 이전 1분 동안 읽은 바이트 수로 집계).</p> <p>유효 차원: JobName, JobRunId, Type(게이지).</p>
<code>glue.executorId.s3.filesystem.read_bytes</code>	<p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다. AWS Glue 지표 대시보드에서 곡선 아래 영역은 두 작업 실행이 읽은 바이트 수를 시각적으로 비교하는 데 사용할 수 있습니다.</p>
<code>glue.ALL.s3.filesystem.read_bytes</code>	<p>단위: 바이트</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • ETL 데이터 이동. • 작업 진행 상황. • 작업 복마크 문제(처리, 재처리 및 건너뛴 데이터). • 외부 데이터 원본으로부터 수집 속도와 읽기를 비교. • 작업 실행 간 편차. <p>결과 데이터는 다음 용도로 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • DPU 용량 계획. • 작업 실행 및 작업 단계를 위해 읽은 데이터의 대규모 스파이크 또는 딥에 대한 경보를 설정.

지표	설명
<p><code>glue.driver.s3.filesystem.write_bytes</code></p> <p><code>glue.executorId.s3.filesystem.write_bytes</code></p> <p><code>glue.ALL.s3.filesystem.write_bytes</code></p>	<p>이전 보고 이후 드라이버, <code>executorId</code>로 식별되는 실행기 또는 모든 실행기가 Amazon S3에 쓴 바이트 수 (AWS Glue 지표 대시보드가 이 목적으로 이전 1분 동안 기록된 바이트 수로 집계).</p> <p>유효 차원: <code>JobName</code>, <code>JobRunId</code>, <code>Type</code>(게이지).</p> <p>유효 통계: SUM. 이 지표는 마지막으로 보고된 값의 델타 값입니다. 그러므로 AWS Glue 지표 대시보드에서 SUM 통계가 집계에 사용됩니다. AWS Glue 지표 대시보드에서 곡선 아래 영역은 두 작업 실행이 기록한 바이트 수를 시각적으로 비교하는 데 사용할 수 있습니다.</p> <p>단위: 바이트</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • ETL 데이터 이동. • 작업 진행 상황. • 작업 복마크 문제(처리, 재처리 및 건너뛴 데이터). • 외부 데이터 원본으로부터 수집 속도와 읽기를 비교. • 작업 실행 간 편차. <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> • DPU 용량 계획. • 작업 실행 및 작업 단계를 위해 읽은 데이터의 대규모 스파이크 또는 딥에 대한 경보를 설정.

지표	설명
<code>glue.driver.streaming.numRecords</code>	<p>마이크로 배치로 수신된 레코드 수입니다. 이 지표는 AWS Glue 버전 2.0 이상의 AWS Glue 스트리밍 작업에만 사용할 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: Sum, Maximum, Minimum, Average, Percentile</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 읽은 레코드 수. • 작업 진행 상황.
<code>glue.driver.streaming.batchProcessingTimeInMs</code>	<p>배치를 처리하는 데 걸리는 시간(밀리초)입니다. 이 지표는 AWS Glue 버전 2.0 이상의 AWS Glue 스트리밍 작업에만 사용할 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID. 또는 ALL), Type(개수).</p> <p>유효 통계: Sum, Maximum, Minimum, Average, Percentile</p> <p>단위: 수</p> <p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> • 작업 진행 상황. • 스크립트 성능입니다.

지표	설명
<code>glue.driver.system.cpuSystemLoad</code>	<p>드라이버, <code>executorId</code>로 식별되는 실행기 또는 모든 실행기가 사용한 CPU 시스템 부하 부분(규모: 0-1).</p> <p>유효 차원: <code>JobName</code>(AWS Glue 작업 이름), <code>JobRunId</code>(<code>JobRun ID</code>. 또는 <code>ALL</code>), <code>Type</code>(게이지).</p>
<code>glue.executorId.system.cpuSystemLoad</code>	<p>유효 통계: <code>Average</code>. 이 지표는 절대값으로 보고됩니다.</p> <p>단위: 퍼센트</p>
<code>glue.ALL.system.cpuSystemLoad</code>	<p>다음을 모니터링하는 데 사용:</p> <ul style="list-style-type: none"> 드라이버 CPU 부하. 실행기 CPU 로드. 작업에서 CPU 바운드 또는 IO 바운드 실행기를 탐지. <p>이 데이터는 다음과 같이 사용할 수 있습니다.</p> <ul style="list-style-type: none"> IO 지표(읽은 바이트/서플 바이트, 태스크 병렬화) 및 최대 필요 실행기 지표와 함께 DPU 용량 계획. CPU/IO 바운드 비율을 식별합니다. 이를 통해 CPU 사용률이 저조한, 분리 가능 데이터 세트의 장기 실행 작업을 위해 프로비저닝된 용량을 재파티셔닝 및 증가할 수 있습니다.

AWS Glue 지표의 차원

AWS Glue 지표는 AWS Glue 네임스페이스를 사용하며 다음 차원의 지표를 제공합니다.

측정기준	설명
<code>JobName</code>	이 차원은 특정 AWS Glue 작업 내 모든 작업 실행의 지표를 필터링합니다.

측정기준	설명
JobRunId	이 차원은 JobRun ID에 의해 실행되는 특정 AWS Glue 작업 또는 ALL의 지표를 필터링합니다.
Type	이 차원은 count(집계 값) 또는 gauge(특정 시점 값) 기준 지표를 필터링합니다.

자세한 내용은 [CloudWatch 사용 설명서](#)를 참조하세요.

AWS Glue 작업 프로파일에서 Amazon CloudWatch 경보 설정

AWS Glue 지표는 Amazon CloudWatch에서도 사용할 수 있습니다. 예약된 작업의 AWS Glue 측정치에 대해 경보를 설정할 수 있습니다.

경보 설정을 위한 몇 가지 일반적인 시나리오는 다음과 같습니다.

- 메모리 부족(OOM) 발생 작업: 메모리 사용량이 AWS Glue 작업에 대한 드라이버 또는 실행기에 대해 정상 평균을 초과할 때 경보를 설정합니다.
- 스트래글링 실행기: AWS Glue 작업에서 실행기 수가 오랜 시간 동안 특정 임계값보다 낮을 때 경보를 설정합니다.
- 데이터 백로그 또는 재처리: CloudWatch 수학적 표현식을 사용하여 워크플로 내 개별 작업의 지표를 비교합니다. 결과 표현식 값(작업이 쓴 바이트 수와 다음 작업이 읽은 바이트 수의 비율 등)에 대해 경보를 트리거할 수 있습니다.

경보 설정에 대한 자세한 지침은 [Amazon CloudWatch Events User Guide](#)의 [Create or Edit a CloudWatch Alarm](#)을 참조하세요.

CloudWatch를 사용한 모니터링 및 디버깅 시나리오는 [작업 모니터링 및 디버깅](#) 섹션을 참조하세요.

AWS Glue 작업에 대한 지속 로깅

AWS Glue는 AWS Glue 작업에 대한 실시간 지속 로깅을 제공합니다. Amazon CloudWatch에서 드라이버 로그, 실행기 로그 및 Apache Spark 작업 진행률 표시줄을 포함한 실시간 Apache Spark 작업 로그를 볼 수 있습니다. 실시간 로그를 보면 실행 중인 작업을 보다 잘 파악할 수 있습니다.

AWS Glue 작업을 시작하면 Spark 애플리케이션 실행이 시작된 후(5초마다 그리고 실행기가 종료되기 전) 실시간 로깅 정보가 CloudWatch에 전송됩니다. AWS Glue 콘솔 또는 CloudWatch 콘솔 대시보드에서 로그를 볼 수 있습니다.

지속 로깅 기능에는 다음 기능이 포함되어 있습니다.

- 연속 로깅
- 애플리케이션별 메시지를 로깅하는 사용자 지정 스크립트 로거
- 현재 AWS Glue 작업의 실행 상태를 추적하는 콘솔 진행률 표시줄

AWS Glue 버전 2.0에서 연속 로깅이 지원되는 방식에 대한 자세한 내용은 [시작 시간이 단축된 Spark ETL 작업 실행](#)을 참조하세요.

IAM 역할이 로그를 읽을 수 있도록 CloudWatch 로그 그룹 또는 스트림에 대한 액세스를 제한할 수 있습니다. 액세스 제한 방법에 대한 자세한 내용은 CloudWatch 설명서의 [CloudWatch Logs에 대한 자격 증명 기반 정책\(IAM 정책\) 사용](#)을 참조하세요.

Note

연속 로깅을 활성화하고 추가 CloudWatch 로그 이벤트가 생성되면 추가 요금이 발생할 수 있습니다. 자세한 내용은 [Amazon CloudWatch 요금](#)을 참조하세요.

주제

- [AWS Glue 작업에 대한 지속 로깅 사용 설정](#)
- [AWS Glue 작업에 대한 지속 로깅 보기](#)

AWS Glue 작업에 대한 지속 로깅 사용 설정

AWS Glue 콘솔을 사용하거나 AWS Command Line Interface(AWS CLI)를 통해 지속 로깅을 활성화할 수 있습니다.

새 작업을 만들거나 기존 작업을 편집할 때 또는 AWS CLI를 통해 연속 로깅을 사용 설정할 수 있습니다.

Amazon CloudWatch 로그 그룹 이름, AWS Glue 작업 실행 ID 드라이버/실행기 ID 앞의 CloudWatch 로그 스트림 접두사 및 로그 메시지에 대한 로그 변환 패턴 등의 사용자 정의 구성 옵션을 지정할 수도 있습니다. 이러한 구성을 사용하면 만료 정책이 서로 다른 사용자 지정 CloudWatch 로그 그룹에 집계 로그를 설정하고 사용자 지정 로그 스트림 접두사 및 변환 패턴을 사용하여 더 자세히 분석할 수 있습니다.

주제

- [AWS Management Console 사용](#)
- [사용자 지정 스크립트 로거를 사용하여 애플리케이션별 메시지 로깅](#)
- [작업 진행률을 보여주는 진행률 표시줄 사용 설정](#)
- [지속 로깅을 사용한 보안 구성](#)

AWS Management Console 사용

AWS Glue 작업을 생성하거나 편집할 때 콘솔에서 지속 로깅을 활성화하려면 다음 절차를 수행합니다.

지속 로깅으로 새 AWS Glue 작업을 생성하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 ETL 작업을 선택합니다.
3. 시각적 ETL을 선택합니다.
4. 작업 세부 정보 탭에서 고급 속성 섹션을 확장합니다.
5. 연속 로깅에서 CloudWatch에서 로그 활성화를 선택합니다.

기존 AWS Glue 작업에 대한 지속 로깅을 활성화하려면

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서, 작업을 선택합니다.
3. 작업 목록에서 기존 작업을 선택합니다.
4. 작업, 작업 편집을 선택합니다.
5. 작업 세부 정보 탭에서 고급 속성 섹션을 확장합니다.
6. 연속 로깅에서 CloudWatch에서 로그 활성화를 선택합니다.

AWS CLI 사용

지속 로깅을 활성화하려면 작업 파라미터를 AWS Glue 작업에 전달합니다. 다른 AWS Glue 작업 파라미터와 유사한 다음 특수 작업 파라미터를 전달합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#) 단원을 참조하십시오.

```
'--enable-continuous-cloudwatch-log': 'true'
```

사용자 정의 Amazon CloudWatch 로그 그룹 이름을 지정할 수 있습니다. 지정하지 않으면 기본 로그 그룹 이름은 `/aws-glue/jobs/logs-v2/`입니다.

```
'--continuous-log-logGroup': 'custom_log_group_name'
```

사용자 정의 Amazon CloudWatch 로그 스트림 접두사를 지정할 수 있습니다. 지정하지 않으면 기본 로그 스트림 접두사는 작업 실행 ID입니다.

```
'--continuous-log-logStreamPrefix': 'custom_log_stream_prefix'
```

사용자 지정 지속적 로깅 변환 규칙을 지정할 수 있습니다. 지정하지 않으면 기본 변환 패턴은 `%d{yy}/MM/dd HH:mm:ss} %p %c{1}: %m%n`입니다. 변환 패턴은 드라이버 로그 및 실행기 로그에만 적용되며 AWS Glue 진행률 표시줄에는 영향을 주지 않습니다.

```
'--continuous-log-conversionPattern': 'custom_log_conversion_pattern'
```

사용자 지정 스크립트 로거를 사용하여 애플리케이션별 메시지 로깅

AWS Glue 로거를 사용하면 드라이버 로그 스트림에 실시간으로 전송되는 모든 애플리케이션별 메시지를 스크립트에서 로깅할 수 있습니다.

다음 예제에서는 Python 스크립트를 보여줍니다.

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)
logger = glueContext.get_logger()
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
```

다음 예제에서는 Scala 스크립트를 보여줍니다.

```
import com.amazonaws.services.glue.log.GlueLogger

object GlueApp {
  def main(sysArgs: Array[String]) {
```

```

val logger = new GlueLogger
logger.info("info message")
logger.warn("warn message")
logger.error("error message")
}
}

```

작업 진행률을 보여주는 진행률 표시줄 사용 설정

AWS Glue는 AWS Glue 작업 실행 상태를 확인하기 위해 JOB_RUN_ID-progress-bar 로그 스트림 아래에 실시간 진행률 표시줄을 제공합니다. 현재 glueContext를 초기화하는 작업만 지원합니다. glueContext를 초기화하지 않고 기본 Spark 작업을 실행하는 경우에는 AWS Glue 진행률 표시줄이 나타나지 않습니다.

이 진행률 표시줄에서는 5초마다 다음 진행률 업데이트를 표시합니다.

```

Stage Number (Stage Name): > (numCompletedTasks + numActiveTasks) /
totalNumOfTasksInThisStage]

```

지속 로깅을 사용한 보안 구성

CloudWatch 로그에 보안 구성이 사용되는 경우 AWS Glue는 연속 로그에 대해 다음과 같은 이름의 로그 그룹을 생성합니다.

```
<Log-Group-Name>-<Security-Configuration-Name>
```

기본 및 사용자 정의 로그 그룹은 다음과 같습니다.

- 기본 연속 로그 그룹은 /aws-glue/jobs/logs-v2-*<Security-Configuration-Name>*입니다.
- 사용자 정의 연속 로그 그룹은 *<custom-log-group-name>*-*<Security-Configuration-Name>*입니다.

CloudWatch Logs로 보안 구성을 사용하는 경우 IAM 역할 권한에 logs:AssociateKmsKey를 추가해야 합니다. 해당 권한이 포함되지 않으면 연속 로깅이 사용 중지됩니다. 또한 CloudWatch Logs에 대한 암호화를 구성하려면 Amazon CloudWatch Logs User Guide의 [Encrypt Log Data in CloudWatch Logs Using AWS Key Management Service](#)를 참조하세요.

보안 구성 생성에 대한 자세한 내용은 [AWS Glue 콘솔의 보안 구성 관련 작업](#) 섹션을 참조하세요.

AWS Glue 작업에 대한 지속 로깅 보기

AWS Glue 콘솔 또는 Amazon CloudWatch 콘솔을 사용하여 실시간 로그를 볼 수 있습니다.

AWS Glue 콘솔 대시보드를 사용하여 실시간 로그를 보려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서, 작업을 선택합니다.
3. 기존 작업을 추가하거나 시작합니다. 작업, 작업 실행을 선택합니다.

작업 실행을 시작할 때 실행 중인 작업에 대한 정보가 포함된 페이지로 이동합니다.

- 로그 탭에 이전에 집계된 애플리케이션 로그가 표시됩니다.
 - Continuous logging(지속 로깅) 탭에는 glueContext가 초기화된 작업이 실행 중일 때 실시간 진행률 표시줄이 표시됩니다.
 - Continuous logging(지속 로깅) 탭에는 실시간 Apache Spark 드라이버 로그를 캡처하는 드라이버 로그 및 작업이 실행 중일 때 AWS Glue 애플리케이션 로거를 사용하여 로깅한 스크립트의 애플리케이션 로그도 포함됩니다.
4. 이전 작업의 경우 [로그(Logs)]를 선택하여 [작업 기록(Job History)] 보기에서 실시간 로그를 볼 수도 있습니다. 이러한 작업을 수행하면 해당 작업 실행에 대한 모든 Spark 드라이버, 실행기 및 진행률 표시줄 로그 스트림을 표시하는 CloudWatch 콘솔로 이동합니다.

CloudWatch 콘솔 대시보드를 사용하여 실시간 로그를 보려면

1. <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 로그를 선택합니다.
3. /aws-glue/jobs/logs-v2/ 로그 그룹을 선택합니다.
4. 필터 상자에 작업 실행 ID를 붙여넣습니다.

드라이버 로그, 실행기 로그 및 진행률 표시줄을 볼 수 있습니다(Standard filter(표준 필터)를 사용하는 경우).

AWS Glue 관찰성 메트릭을 사용한 모니터링

Note

AWS Glue 관찰성 지표는 AWS Glue 4.0 이상 버전에서 사용할 수 있습니다.

AWS Glue 관찰성 메트릭을 사용하면 Apache Spark의 AWS Glue 내부에서 일어나는 일에 대한 통찰력을 얻어 문제의 분류 및 분석을 개선할 수 있습니다. 관찰성 지표는 Amazon CloudWatch 대시보드를 통해 시각화되며 오류의 근본 원인 분석을 수행하고 성능 병목 현상을 진단하는 데 사용할 수 있습니다. 대규모 문제 디버깅에 소요되는 시간을 줄여 문제를 더 빠르고 효과적으로 해결하는 데 집중할 수 있습니다.

AWS Glue 관찰성은 다음 네 그룹으로 분류된 Amazon CloudWatch 메트릭을 제공합니다.

- 신뢰성(예: 오류 클래스) - 주어진 시간 범위에서 해결해야 할 가장 일반적인 장애 원인을 쉽게 식별할 수 있습니다.
- 성능(예: Skewness) - 성능 병목 지점을 식별하고 조정 기법을 적용합니다. 예를 들어 작업 왜곡으로 인해 성능이 저하되는 경우 Spark Adaptive Query Execution을 활성화하고 스쿼 조인 임계값을 미세 조정하는 것이 좋습니다.
- 처리량(예: 소스/싱크당 처리량) - 데이터 읽기 및 쓰기 추세를 모니터링합니다. 또한 이상 현상에 대한 Amazon CloudWatch 경보를 구성할 수 있습니다.
- 리소스 사용률(예: 작업자, 메모리 및 디스크 사용률) - 용량 사용률이 낮은 작업을 효율적으로 찾을 수 있습니다. 이러한 작업에 대해 AWS Glue Auto Scaling을 활성화할 수 있습니다.

AWS Glue 관찰성 메트릭 시작하기

Note

새로운 지표는 AWS Glue Studio 콘솔에서 기본적으로 활성화됩니다.

AWS Glue Studio에서 관찰성 지표를 구성하려면 다음을 수행합니다.

1. AWS Glue 콘솔에 로그인하고 콘솔 메뉴에서 ETL 작업을 선택합니다.
2. 내 작업 섹션에서 작업 이름을 클릭하여 작업을 선택합니다.
3. [작업 세부 정보(Job details)] 탭을 선택합니다.

4. 하단으로 스크롤하여 고급 속성을 선택한 다음 작업 관찰성 지표를 선택합니다.

obs-test Last modified on 10/10/2023, 2:04:44 PM [Try new UI](#) [Load JSON](#) [De](#)

Visual | Script | **Job details** | Runs | Data quality New | Schedules | Version Control

▼ **Advanced properties**

Script filename
obs-test.py

Script path
S3 location of the script. Path must be in the form s3://bucket/prefix/path/. It must end with a slash (/) and not include any files.
s3://aws-glue-assets-590186200215-us-east-1/scripts/ [View](#) [Browse S3](#)

Job metrics [Info](#)
 Enable the creation of CloudWatch metrics when this job runs.

Job observability metrics [Info](#)
 Enable the creation of additional observability CloudWatch metrics when this job runs.

Continuous logging [Info](#)
 Enable logs in CloudWatch.

Spark UI [Info](#)
 Enable using Spark UI for monitoring this job.

Serverless Spark UI [Info](#)
 Enable using Serverless Spark UI for monitoring this job.

Spark UI logs path
s3://aws-glue-assets-590186200215-us-east-1/sparkHistoryLogs/ [View](#) [Browse S3](#)

Maximum concurrency
Sets the maximum number of concurrent runs that are allowed for this job. An error is returned when this threshold is reached.
1

Temporary path
Working directory. Path must be in the form s3://bucket/prefix/path/. It must end with a slash (/) and not include any files.
s3://aws-glue-assets-590186200215-us-east-1/temporary/ [View](#) [Browse S3](#)

Delay notification threshold (minutes) | 1

AWS CLI를 사용하여 AWS Glue 관찰성 지표를 활성화하려면 다음을 수행합니다.

- 입력 `--default-arguments` JSON 파일의 다음 키값을 맵에 추가합니다.

```
--enable-observability-metrics, true
```

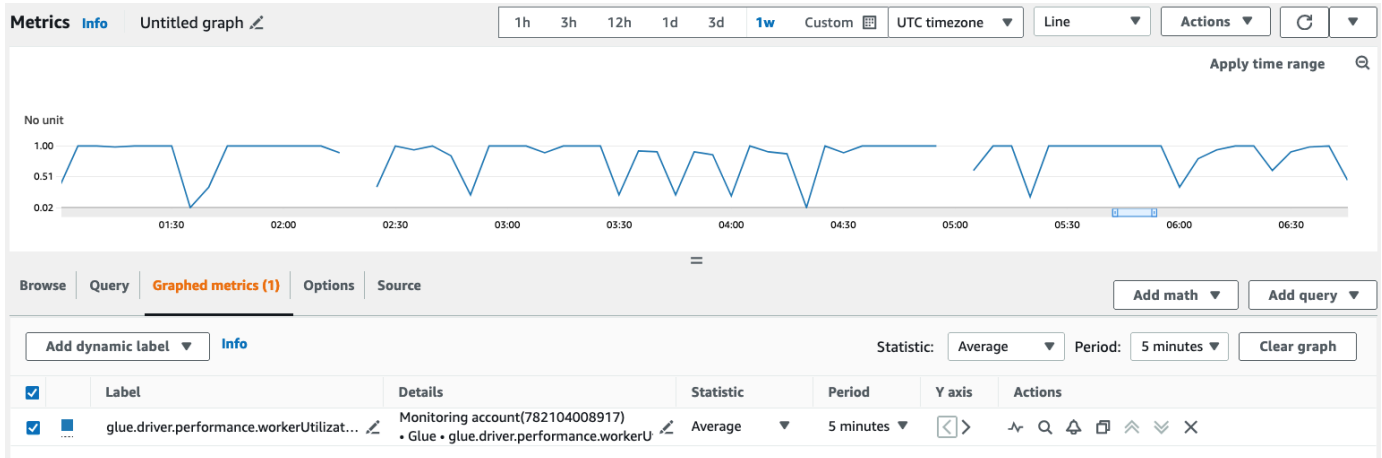
AWS Glue 관찰성 사용

AWS Glue 관찰성 지표는 Amazon CloudWatch를 통해 제공되므로 Amazon CloudWatch 콘솔, AWS CLI, SDK 또는 API를 사용하여 관찰성 지표 데이터 포인트를 쿼리할 수 있습니다. AWS Glue 관찰성 지표를 사용하는 사용 사례의 예는 [Using Glue Observability for monitoring resource utilization to reduce cost](#)를 참조하세요.

Amazon CloudWatch 콘솔에서 AWS Glue 관찰성 사용

Amazon CloudWatch 콘솔에서 메트릭을 쿼리하고 시각화는 방법:

1. Amazon CloudWatch 콘솔을 열고 모든 지표를 선택합니다.
2. 사용자 지정 네임스페이스에서 AWS Glue를 선택합니다.
3. 작업 관찰성 지표, 소스별 관찰 가능성 지표 또는 싱크당 관찰 가능성 지표를 선택합니다.
4. 특정 지표 이름, 작업 이름, 작업 실행 ID를 검색하고 선택합니다.
5. 그래프로 표시된 지표 탭에서 원하는 통계, 기간 및 기타 옵션을 구성합니다.



AWS CLI를 사용하여 관찰성 지표를 쿼리하려면 다음을 수행합니다.

1. 지표 정의 JSON 파일을 생성하고 `your-Glue-job-name` 및 `your-Glue-job-run-id`를 변경합니다.

```
$ cat multiplequeries.json
[
  {
    "Id": "avgWorkerUtil_0",
    "MetricStat": {
      "Metric": {
```

```

        "Namespace": "Glue",
        "MetricName": "glue.driver.workerUtilization",
        "Dimensions": [
            {
                "Name": "JobName",
                "Value": "<your-Glue-job-name-A>"
            },
            {
                "Name": "JobRunId",
                "Value": "<your-Glue-job-run-id-A>"
            },
            {
                "Name": "Type",
                "Value": "gauge"
            },
            {
                "Name": "ObservabilityGroup",
                "Value": "resource_utilization"
            }
        ]
    },
    "Period": 1800,
    "Stat": "Minimum",
    "Unit": "None"
},
{
    "Id": "avgWorkerUtil_1",
    "MetricStat": {
        "Metric": {
            "Namespace": "Glue",
            "MetricName": "glue.driver.workerUtilization",
            "Dimensions": [
                {
                    "Name": "JobName",
                    "Value": "<your-Glue-job-name-B>"
                },
                {
                    "Name": "JobRunId",
                    "Value": "<your-Glue-job-run-id-B>"
                },
                {
                    "Name": "Type",
                    "Value": "gauge"
                }
            ]
        }
    }
}

```

```

        },
        {
            "Name": "ObservabilityGroup",
            "Value": "resource_utilization"
        }
    ]
},
"Period": 1800,
"Stat": "Minimum",
"Unit": "None"
}
}
]

```

2. get-metric-data 명령 실행:

```

$ aws cloudwatch get-metric-data --metric-data-queries file: //multiplequeries.json \
\
  --start-time '2023-10-28T18: 20' \
  --end-time '2023-10-28T19: 10' \
  --region us-east-1
{
  "MetricDataResults": [
    {
      "Id": "avgWorkerUtil_0",
      "Label": "<your-label-for-A>",
      "Timestamps": [
        "2023-10-28T18:20:00+00:00"
      ],
      "Values": [
        0.06718750000000001
      ],
      "StatusCode": "Complete"
    },
    {
      "Id": "avgWorkerUtil_1",
      "Label": "<your-label-for-B>",
      "Timestamps": [
        "2023-10-28T18:50:00+00:00"
      ],
      "Values": [
        0.5959183673469387
      ]
    }
  ]
}

```

```

        ],
        "StatusCode": "Complete"
    }
],
"Messages": []
}
    
```

관찰성 메트릭

AWS Glue 관찰성은 다음 지표의 프로필을 작성하고 30초마다 Amazon CloudWatch로 전송하며, 이러한 지표 중 일부는 AWS Glue Studio 작업 실행 모니터링 페이지에서 확인할 수 있습니다.

지표	설명	범주
glue.driver.skewness.stage	<p>지표 범주: job_performance</p> <p>Spark 스테이지 실행 왜도: 이 지표는 입력 데이터 왜도 또는 변환(예: 왜곡된 조인)으로 인해 발생할 수 있는 실행 왜도를 캡처합니다. 이 지표의 값은 [0, 무한] 범위에 속합니다. 여기서 0은 스테이지의 모든 작업 중 최대 작업 실행 시간과 중간 작업 실행 시간의 비율이 특정 스테이지 왜도 인자보다 작다는 것을 의미합니다. 기본 스테이지 왜도 인자는 `5`이며 spark conf: spark.metrics.conf.driver.source.glue.jobPerformance.skewnessFactor를 통해 덮어쓸 수 있습니다.</p> <p>스테이지 왜도 값이 1이면 비율이 단계 왜도 인자의 두 배임을 의미합니다.</p>	job_performance

지표	설명	범주
	<p>스태이지 왜도 값은 현재 왜도를 반영하여 30초마다 업데이트됩니다. 스태이지가 끝날 때의 값은 최종 스태이지 왜도를 반영합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(Job_Performance)</p> <p>유효 통계: 평균, 최대, 최소, 백분위수</p> <p>단위: 수</p>	

지표	설명	범주
glue.driver.skewness.job	<p>지표 범주: job_performance</p> <p>작업 왜도는 작업 스테이지 왜도의 가중치 평균입니다. 가중치 평균은 실행하는 데 시간이 오래 걸리는 스테이지에 더 많은 가중치를 부여합니다. 이는 매우 왜곡된 스테이지가 실제로 다른 스테이지에 비해 매우 짧은 시간 동안 실행되는 경우를 피하기 위한 것입니다. 따라서 왜도는 전체 작업 성능에 있어 중요하지 않으며 왜도를 해결하려고 노력할 필요가 없습니다.</p> <p>이 지표는 각 단계가 완료될 때마다 업데이트되므로 마지막 값은 실제 전체 작업 왜도를 반영합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(Job_Performance)</p> <p>유효 통계: 평균, 최대, 최소, 백분위수</p> <p>단위: 수</p>	job_performance

지표	설명	범주
glue.succeed.ALL	<p>지표 범주: 오류</p> <p>실패 범주의 그림을 완성하기 위한 총 작업 실행 성공 횟수</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(개수), ObservabilityGroup(오류)</p> <p>유효 통계: SUM</p> <p>단위: 수</p>	error
glue.error.ALL	<p>지표 범주: 오류</p> <p>실패 범주의 그림을 완성하기 위한 총 작업 실행 오류 횟수</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(개수), ObservabilityGroup(오류)</p> <p>유효 통계: SUM</p> <p>단위: 수</p>	error

지표	설명	범주
glue.error.[error category]	<p>지표 범주: 오류</p> <p>이것은 실제로 작업 실행이 실패할 때만 업데이트되는 지표 세트입니다. 오류 분류는 분류 및 디버깅에 도움이 됩니다. 작업 실행이 실패하면 실패의 원인이 되는 오류가 분류되고 해당 오류 범주 지표가 1로 설정됩니다. 이를 통해 시간 경과에 따른 오류 분석은 물론 모든 작업에 대한 오류 분석을 수행하여 가장 일반적인 오류 범주를 식별하고 문제 해결을 시작할 수 있습니다. AWS Glue에는 OUT_OF_MEMORY(드라이버 및 실행기), PERMISSION, SYNTAX 및 THROTTLING 오류 범주를 포함한 28개의 오류 범주가 있습니다. 오류 범주는 COMPILATION, LAUNCH, TIMEOUT 오류 범주도 포함합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(개수), ObservabilityGroup(오류)</p> <p>유효 통계: SUM</p> <p>단위: 수</p>	error

지표	설명	범주
glue.driver.workerUtilization	<p>지표 범주: resource_utilization</p> <p>할당된 작업자 중 실제로 사용된 작업자의 비율입니다. 상황이 좋지 않다면 Auto Scaling이 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>유효 통계: 평균, 최대, 최소, 백분위수</p> <p>단위: 퍼센트</p>	resource_utilization

지표	설명	범주
glue.driver.memory.heap.[available used]	<p>지표 범주: resource_utilization</p> <p>작업 실행 중 드라이버의 사용 가능/사용된 힙 메모리입니다. 이는 특히 시간 경과에 따른 메모리 사용량 추세를 파악하는데 도움이 되며 메모리 관련 오류를 디버깅할 뿐만 아니라 잠재적인 오류를 방지하는 데도 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 바이트</p>	resource_utilization

지표	설명	범주
glue.driver.memory.heap.used.percentage	<p>지표 범주: resource_utilization</p> <p>작업 실행 중 드라이버가 사용한 힙 메모리(%)입니다. 이는 특히 시간 경과에 따른 메모리 사용량 추세를 파악하는 데 도움이 되며 메모리 관련 오류를 디버깅할 뿐만 아니라 잠재적인 오류를 방지하는 데도 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 퍼센트</p>	resource_utilization

지표	설명	범주
glue.driver.memory.non-heap. [available used]	<p>지표 범주: resource_utilization</p> <p>작업 실행 중 드라이버가 사용 가능한 또는 사용한 힙이 아닌 메모리입니다. 이는 특히 시간 경과에 따른 메모리 사용량 추세를 파악하는 데 도움이 되며 메모리 관련 오류를 디버깅할 뿐만 아니라 잠재적인 오류를 방지하는 데도 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 바이트</p>	resource_utilization

지표	설명	범주
glue.driver.memory.non-heap.used.percentage	<p>지표 범주: resource_utilization</p> <p>작업 실행 중 드라이버가 사용한 힙이 아닌 메모리(%)입니다. 이는 특히 시간 경과에 따른 메모리 사용량 추세를 파악하는데 도움이 되며 메모리 관련 오류를 디버깅할 뿐만 아니라 잠재적인 오류를 방지하는 데도 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 퍼센트</p>	resource_utilization

지표	설명	범주
glue.driver.memory.total.[available used]	<p>지표 범주: resource_utilization</p> <p>작업 실행 중 드라이버가 사용할 수 있는 또는 사용한 총 메모리입니다. 이는 특히 시간 경과에 따른 메모리 사용량 추세를 파악하는 데 도움이 되며 메모리 관련 오류를 디버깅할 뿐만 아니라 잠재적인 오류를 방지하는 데도 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 바이트</p>	resource_utilization

지표	설명	범주
<p>glue.driver.memory.total.used.percentage</p>	<p>지표 범주: resource_utilization</p> <p>작업 실행 중 드라이버가 사용한 총 메모리(%)입니다. 이는 특히 시간 경과에 따른 메모리 사용량 추세를 파악하는 데 도움이 되며 메모리 관련 오류를 디버깅할 뿐만 아니라 잠재적인 오류를 방지하는 데도 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 퍼센트</p>	<p>resource_utilization</p>
<p>glue.ALL.memory.heap.[available used]</p>	<p>지표 범주: resource_utilization</p> <p>실행기의 사용 가능한/사용된 힙 메모리입니다. ALL은 모든 실행기를 의미합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 바이트</p>	<p>resource_utilization</p>

지표	설명	범주
glue.ALL.memory.heap.used.percentage	<p>지표 범주: resource_utilization</p> <p>실행기가 사용한 힙 메모리(%)입니다. ALL은 모든 실행기를 의미합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 퍼센트</p>	resource_utilization
glue.ALL.memory.non-heap.[available used]	<p>지표 범주: resource_utilization</p> <p>실행기의 사용 가능한/사용된 힙이 아닌 메모리입니다. ALL은 모든 실행기를 의미합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 바이트</p>	resource_utilization

지표	설명	범주
glue.ALL.memory.non-heap.used.percentage	<p>지표 범주: resource_utilization</p> <p>실행기가 사용한 힙이 아닌 메모리(%)입니다. ALL은 모든 실행기를 의미합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 퍼센트</p>	resource_utilization
glue.ALL.memory.total.[available used]	<p>지표 범주: resource_utilization</p> <p>실행기의 사용 가능한/사용된 총 메모리입니다. ALL은 모든 실행기를 의미합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 바이트</p>	resource_utilization

지표	설명	범주
glue.ALL.memory.total.used.percentage	<p>지표 범주: resource_utilization</p> <p>실행기의 총 메모리 사용률(%)입니다. ALL은 모든 실행기를 의미합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 퍼센트</p>	resource_utilization
glue.driver.disk.[available_GB used_GB]	<p>지표 범주: resource_utilization</p> <p>작업 실행 중 드라이버의 사용 가능한/사용된 디스크 공간입니다. 이는 특히 시간 경과에 따른 디스크 사용량 추세를 파악하는 데 도움이 되며 디스크 공간 부족 관련 오류를 디버깅할 뿐만 아니라 잠재적인 오류를 방지하는 데도 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 기가바이트</p>	resource_utilization

지표	설명	범주
<p>glue.driver.disk.used.percentage]</p>	<p>지표 범주: resource_utilization</p> <p>작업 실행 중 드라이버의 사용 가능한/사용된 디스크 공간입니다. 이는 특히 시간 경과에 따른 디스크 사용량 추세를 파악하는 데 도움이 되며 디스크 공간 부족 관련 오류를 디버깅할 뿐만 아니라 잠재적인 오류를 방지하는 데도 도움이 될 수 있습니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 퍼센트</p>	<p>resource_utilization</p>
<p>glue.ALL.disk.[available_GB used_GB]</p>	<p>지표 범주: resource_utilization</p> <p>실행기의 사용 가능한/사용된 디스크 공간입니다. ALL은 모든 실행기를 의미합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 기가바이트</p>	<p>resource_utilization</p>

지표	설명	범주
<p>glue.ALL.disk.used.percentage</p>	<p>지표 범주: resource_utilization</p> <p>실행기의 사용 가능한/사용된/사용된(%) 디스크 공간입니다. ALL은 모든 실행기를 의미합니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization)</p> <p>Valid Statistics: Average</p> <p>단위: 퍼센트</p>	<p>resource_utilization</p>
<p>glue.driver.bytesRead</p>	<p>지표 범주: 처리량</p> <p>이 작업 실행에서 입력 소스당 및 모든 소스에 대해 읽은 바이트 수입니다. 이를 통해 데이터 양과 시간 경과에 따른 변화를 파악할 수 있으므로 데이터 왜도와 같은 문제를 해결하는 데 도움이 됩니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization), Source(소스 데이터 위치)</p> <p>Valid Statistics: Average</p> <p>단위: 바이트</p>	<p>처리량</p>

지표	설명	범주
glue.driver.[recordsRead filesRead]	<p>지표 범주: 처리량</p> <p>이 작업 실행에서 입력 소스당 및 모든 소스에 대해 읽은 레코드/파일 수입니다. 이를 통해 데이터양과 시간 경과에 따른 변화를 파악할 수 있으므로 데이터 왜도와 같은 문제를 해결하는 데 도움이 됩니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization), Source(소스 데이터 위치)</p> <p>Valid Statistics: Average</p> <p>단위: 수</p>	처리량
glue.driver.partitionsRead	<p>지표 범주: 처리량</p> <p>이 작업 실행에서 Amazon S3 입력 소스당 및 모든 소스에 대해 읽은 파티션 수입니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization), Source(소스 데이터 위치)</p> <p>Valid Statistics: Average</p> <p>단위: 수</p>	처리량

지표	설명	범주
glue.driver.bytesWritten	<p>지표 범주: 처리량</p> <p>이 작업 실행에서 출력 싱크당 및 모든 싱크에 대해 작성된 바이트 수입니다. 이를 통해 데이터양과 시간이 지남에 따라 어떻게 변화하는지 파악할 수 있으므로 처리 왜도와 같은 문제를 해결하는 데 도움이 됩니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization), Sink(싱크 데이터 위치)</p> <p>Valid Statistics: Average</p> <p>단위: 바이트</p>	처리량

지표	설명	범주
glue.driver.[recordsWritten filesWritten]	<p>지표 범주: 처리량</p> <p>이 작업 실행에서 출력 싱크당 및 모든 싱크에 대해 작성된 레코드/파일 수입니다. 이를 통해 데이터양과 시간이 지남에 따라 어떻게 변화하는지 파악할 수 있으므로 처리 왜도와 같은 문제를 해결하는 데 도움이 됩니다.</p> <p>유효 차원: JobName(AWS Glue 작업 이름), JobRunId(JobRun ID 또는 ALL), Type(게이지), ObservabilityGroup(resource_utilization), Sink(싱크 데이터 위치)</p> <p>Valid Statistics: Average</p> <p>단위: 수</p>	처리량

오류 범주

오류 범주	설명
COMPILATION_ERROR	Scala 코드를 컴파일하는 동안 오류가 발생합니다.
CONNECTION_ERROR	서비스/원격 호스트/데이터베이스 서비스 등에 연결하는 동안 오류가 발생합니다.
DISK_NO_SPACE_ERROR	드라이버/실행기의 디스크에 공간이 남아 있지 않은 경우 오류가 발생합니다.

오류 범주	설명
OUT_OF_MEMORY_ERROR	드라이버/실행기의 메모리에 공간이 남아 있지 않은 경우 오류가 발생합니다.
IMPORT_ERROR	종속성을 가져올 때 오류가 발생합니다.
INVALID_ARGUMENT_ERROR	입력 인수가 유효하지 않거나 잘못된 경우 오류가 발생합니다.
PERMISSION_ERROR	서비스, 데이터 등에 대한 권한이 없는 경우 오류가 발생합니다.
RESOURCE_NOT_FOUND_ERROR	데이터, 위치 등이 종료되지 않을 경우 오류가 발생합니다.
QUERY_ERROR	Spark SQL 쿼리 실행으로 인해 오류가 발생합니다.
SYNTAX_ERROR	스크립트에 구문 오류가 있는 경우 오류가 발생합니다.
THROTTLING_ERROR	서비스 동시성 한도에 도달하거나 서비스 할당량 한도를 초과하는 경우 오류가 발생합니다.
DATA_LAKE_FRAMEWORK_ERROR	Hudi, Iceberg 등과 같은 AWS Glue 기본 지원 데이터 레이크 프레임워크에서 오류가 발생합니다.
UNSUPPORTED_OPERATION_ERROR	지원되지 않는 작업을 수행하는 경우 오류가 발생합니다.
RESOURCES_ALREADY_EXISTS_ERROR	생성하거나 추가하려는 리소스가 이미 존재하는 경우 오류가 발생합니다.
GLUE_INTERNAL_SERVICE_ERROR	AWS Glue 내부 서비스 문제가 있는 경우 오류가 발생합니다.
GLUE_OPERATION_TIMEOUT_ERROR	AWS Glue 작업 시간이 초과되는 경우 오류가 발생합니다.

오류 범주	설명
GLUE_VALIDATION_ERROR	AWS Glue 작업에 필요한 값을 검증할 수 없는 경우 오류가 발생합니다.
GLUE_JOB_BOOKMARK_VERSION_MISMATCH_ERROR	동일한 소스 버킷에서 동일한 작업이 실행되고 동일한/다른 대상에 동시에 작성되는 경우 오류가 발생합니다(동시성 > 1).
LAUNCH_ERROR	AWS Glue 작업 시작 단계에서 오류가 발생합니다.
DYNAMODB_ERROR	Amazon DynamoDB 서비스에서 일반 오류가 발생합니다.
GLUE_ERROR	AWS Glue 서비스에서 일반 오류가 발생합니다.
LAKEFORMATION_ERROR	AWS Lake Formation 서비스에서 일반 오류가 발생합니다.
REDSHIFT_ERROR	Amazon Redshift 서비스에서 일반 오류가 발생합니다.
S3_ERROR	Amazon S3 서비스에서 일반 오류가 발생합니다.
SYSTEM_EXIT_ERROR	일반 시스템 종료 오류입니다.
TIMEOUT_ERROR	작업 시간 초과로 인해 작업이 실패하면 일반 오류가 발생합니다.
UNCLASSIFIED_SPARK_ERROR	Spark에서 일반 오류가 발생합니다.
UNCLASSIFIED_ERROR	기본 오류 범주입니다.

제한 사항

Note

`glueContext`를 초기화해야만 지표를 게시할 수 있습니다.

소스 차원에서 값은 소스 유형에 따라 Amazon S3 경로 또는 테이블 이름입니다. 또한 소스가 JDBC이고 쿼리 옵션이 사용되는 경우 쿼리 문자열은 소스 차원에 설정됩니다. 값이 500자보다 길면 500자 이내로 잘립니다. 다음은 값의 제한 사항입니다.

- ASCII가 아닌 문자는 제거됩니다.
- 소스 이름에 ASCII 문자가 포함되어 있지 않으면 <비 ASCII 입력>으로 변환됩니다.

처리량 지표의 제한 사항 및 고려 사항

- DataFrame 및 DataFrame 기반 DynamicFrame(예: JDBC, Amazon S3의 Parquet에서 읽기)은 지원되지만 RDD 기반 DynamicFrame(예: Amazon S3에서 csv, json 읽기 등)은 지원되지 않습니다. 기술적으로는 Spark UI에 표시되는 모든 읽기 및 쓰기가 지원됩니다.
- 데이터 소스가 카탈로그 테이블이고 형식이 JSON, CSV, 텍스트 또는 Iceberg인 경우 `recordsRead` 지표가 내보내집니다.
- `glue.driver.throughput.recordsWritten`, `glue.driver.throughput.bytesWritten`, `glue.driver.throughput.filesWritten` 지표는 JDBC 및 Iceberg 테이블에서 사용할 수 없습니다.
- 지표가 지연될 수 있습니다. 작업이 약 1분 안에 완료되면 Amazon CloudWatch 지표에 처리량 지표가 표시되지 않을 수 있습니다.

작업 모니터링 및 디버깅

AWS Glue 작업에 대한 지표를 수집하고 AWS Glue 및 Amazon CloudWatch 콘솔에서 이러한 지표를 시각화하여 문제를 식별하고 수정할 수 있습니다. AWS Glue 작업을 프로파일링하려면 다음 단계를 수행해야 합니다.

1. 지표를 활성화하려면 다음을 수행합니다.

- a. 작업 정의에서 작업 측정치 옵션을 활성화합니다. AWS Glue 콘솔에서 또는 작업의 파라미터로서 프로파일링을 활성화할 수 있습니다. 자세한 정보는 [Spark 작업에 대한 작업 속성 정의](#) 또는 [AWS Glue 작업 파라미터](#) 섹션을 참조하세요.
 - b. 작업 정의에서 AWS Glue 관찰성 지표 옵션을 활성화합니다. AWS Glue 콘솔에서 또는 작업의 파라미터로서 관찰성을 활성화할 수 있습니다. 자세한 정보는 [AWS Glue 관찰성 메트릭을 사용한 모니터링](#) 섹션을 참조하세요.
2. 작업 스크립트에서 GlueContext를 초기화하는지 확인합니다. 예를 들면 다음 스크립트 조각은 GlueContext를 추가화하고 프로파일링된 코드가 스크립트에 있는 위치를 표시합니다. 이 일반적인 형식은 이어지는 디버깅 시나리오에서 사용됩니다.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

...
...
code-to-profile
...
...

job.commit()
```

3. 작업을 실행합니다.
4. 지표를 시각화하려면 다음을 수행합니다.

- a. AWS Glue 콘솔에서 작업 지표를 시각화하고 드라이버 또는 실행기의 이상 지표를 식별합니다.
 - b. 작업 실행 모니터링 페이지, 작업 실행 세부 정보 페이지 또는 Amazon CloudWatch에서 관찰성 지표를 확인합니다. 자세한 내용은 [AWS Glue 관찰성 메트릭을 사용한 모니터링](#) 단원을 참조하십시오.
5. 식별된 측정치를 사용하여 근본 원인의 범위를 좁힙니다.
 6. 원한다면 식별된 드라이버나 작업 실행기의 로그 스트림을 사용하여 근본 원인을 확인할 수 있습니다.

AWS Glue 관찰성 지표의 사용 사례

- [OOM 예외 사항 및 작업 이상 현상 디버깅](#)
- [까다로운 단계와 스트래글러 작업 디버깅](#)
- [여러 작업의 진행 상황 모니터링](#)
- [DPU 용량 계획 모니터링](#)
- [리소스 사용률 모니터링을 위해 AWS Glue 관찰성을 사용하여 비용 절감](#)

OOM 예외 사항 및 작업 이상 현상 디버깅

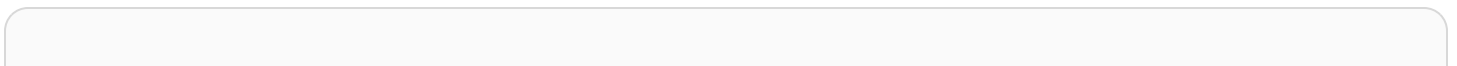
AWS Glue에서 메모리 부족(OOM) 예외 사항 및 작업 이상 현상을 디버깅할 수 있습니다. 다음 단원에서는 Apache Spark 드라이버 또는 Spark 실행기의 메모리 부족 예외 사항을 디버깅하는 시나리오를 설명합니다.

- [드라이버 OOM 예외 사항 디버깅](#)
- [실행기 OOM 예외 사항 디버깅](#)

드라이버 OOM 예외 사항 디버깅

이 시나리오에서는 Spark 작업이 Amazon Simple Storage Service(Amazon S3)에서 여러 작은 파일을 읽습니다. 파일을 Apache Parquet 포맷으로 변환한 다음 Amazon S3에 씁니다. Spark 드라이버에서 메모리 부족이 발생합니다. 입력 Amazon S3 데이터의 파일 수는 서로 다른 Amazon S3 파티션에 걸쳐 100만개가 넘습니다.

프로파일링된 코드는 다음과 같습니다.



```
data = spark.read.format("json").option("inferSchema", False).load("s3://input_path")
data.write.format("parquet").save(output_path)
```

AWS Glue 콘솔에서 프로파일링된 지표 시각화

다음 그래프는 드라이버와 실행기에 대한 메모리 사용률(%)을 표시합니다. 이 사용률은 지난 1분 안에 보고된 값에 대한 평균으로 계산된 데이터 하나로 표시됩니다. [드라이버 메모리](#)가 50% 사용률의 안전 임계값을 넘는 작업의 메모리 프로필을 볼 수 있습니다. 반면에, 모든 실행기의 [평균 메모리 사용률](#)은 여전히 4%보다 낮습니다. 이로써 이 Spark 작업에서 드라이버 실행에 이상이 있음을 명확히 알 수 있습니다.



작업 실행이 곧 실패하고 AWS Glue 콘솔의 [기록(History)] 탭에 [종료 코드 1로 명령 실패(Command Failed with Exit Code 1)] 오류가 나타납니다. 이 오류 문자열은 시스템 전체 오류(이 경우 메모리 부족)로 인해 작업이 실패했음을 의미합니다.

Run ID	Retry attempt	Run status	Error	Logs	Error logs	Execution time	Timeout	Delay	Triggered by	Start time
jr_651bfc34...	-	Failed	!	...	Logs Error logs	2 mins	2880 mins			7 June 2018 7:37 PM UTC-7
jr_5731b225...	-	Failed	Command failed with exit code 1							7 June 2018 7:37 PM UTC-7

콘솔의 [기록(History)] 탭에서 [오류 로그(Error logs)] 링크를 선택하여 CloudWatch Logs에서 드라이버 OOM에 대한 결과를 확인합니다. 작업의 오류 로그에서 "Error"를 검색하여 작업 실패의 원인이 된 OOM 예외 사항이 정말 발생했는지 확인합니다.

```
# java.lang.OutOfMemoryError: Java heap space
# -XX:OnOutOfMemoryError="kill -9 %p"
# Executing /bin/sh -c "kill -9 12039"...
```

작업에 대해 기록 탭에서 로그를 선택합니다. 작업의 시작 부분에서 다음과 같은 CloudWatch Logs의 드라이버 실행 추적 정보를 찾아볼 수 있습니다. Spark 드라이버에서 모든 디렉터리의 모든 파일을 나열하려고 했으며, InMemoryFileIndex를 생성하고 파일마다 작업 하나를 실행합니다. 이로 인해 Spark 드라이버는 모든 작업을 추적하기 위해 메모리에서 매우 많은 상태를 유지해야 합니다. 따라서 메모리 내 인덱스에 대해 다량의 파일을 포함하는 전체 목록을 캐시하므로 드라이버 OOM이 발생하게 됩니다.

그룹화를 사용하여 여러 파일의 처리 수정

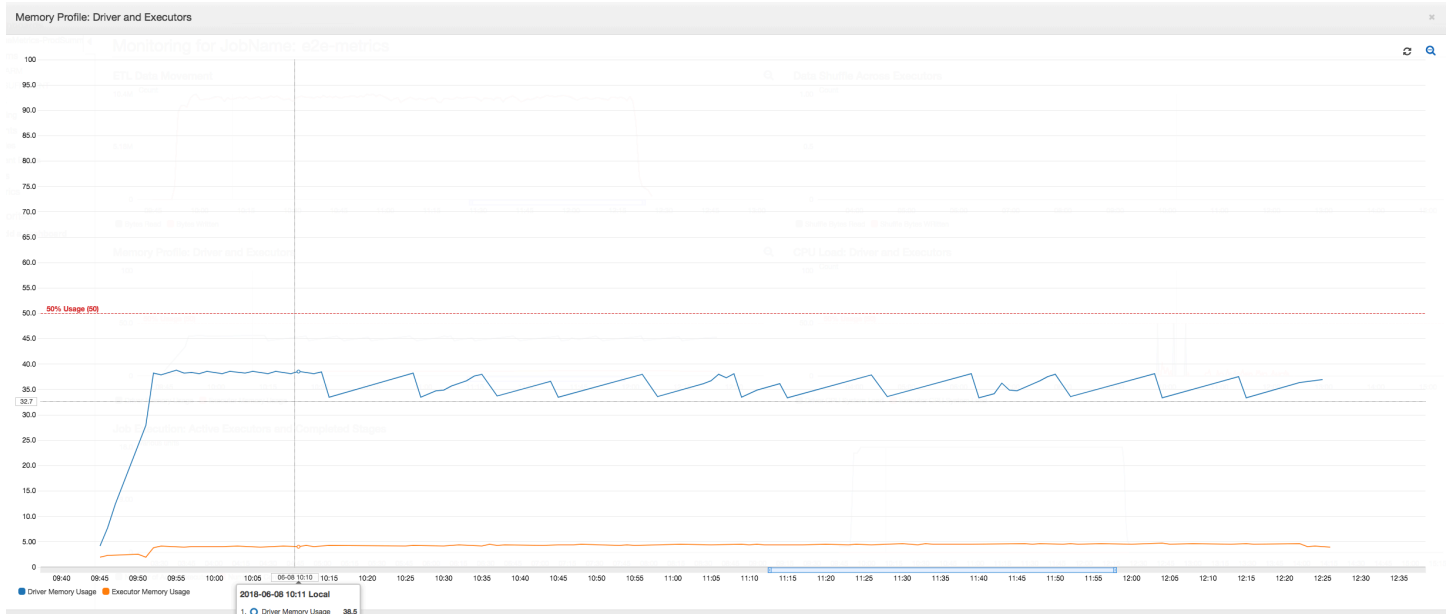
AWS Glue의 그룹화 기능을 사용하여 여러 파일의 처리를 수정할 수 있습니다. 그룹화는 동적 프레임 을 사용할 때와 입력 데이터 세트에 파일 수가 많을 때(50,000개 초과) 자동으로 활성화됩니다. 그룹화 를 통해 여러 파일을 그룹 하나로 합칠 수 있으므로 작업 하나로 단일 파일 대신에 전체 그룹을 처리할 수 있습니다. 결과적으로 Spark 드라이버는 메모리 안에 훨씬 더 적은 상태를 저장하여 몇 개 작업만 추적합니다. 데이터 세트에 대한 그룹화 수동 활성화에 대한 자세한 내용은 [입력 파일을 더 큰 그룹에 서 읽기](#) 단원을 참조하십시오.

AWS Glue 작업의 메모리 프로파일을 확인하려면 그룹화가 활성화된 상태에서 다음과 같은 코드를 프 로파일링합니다.

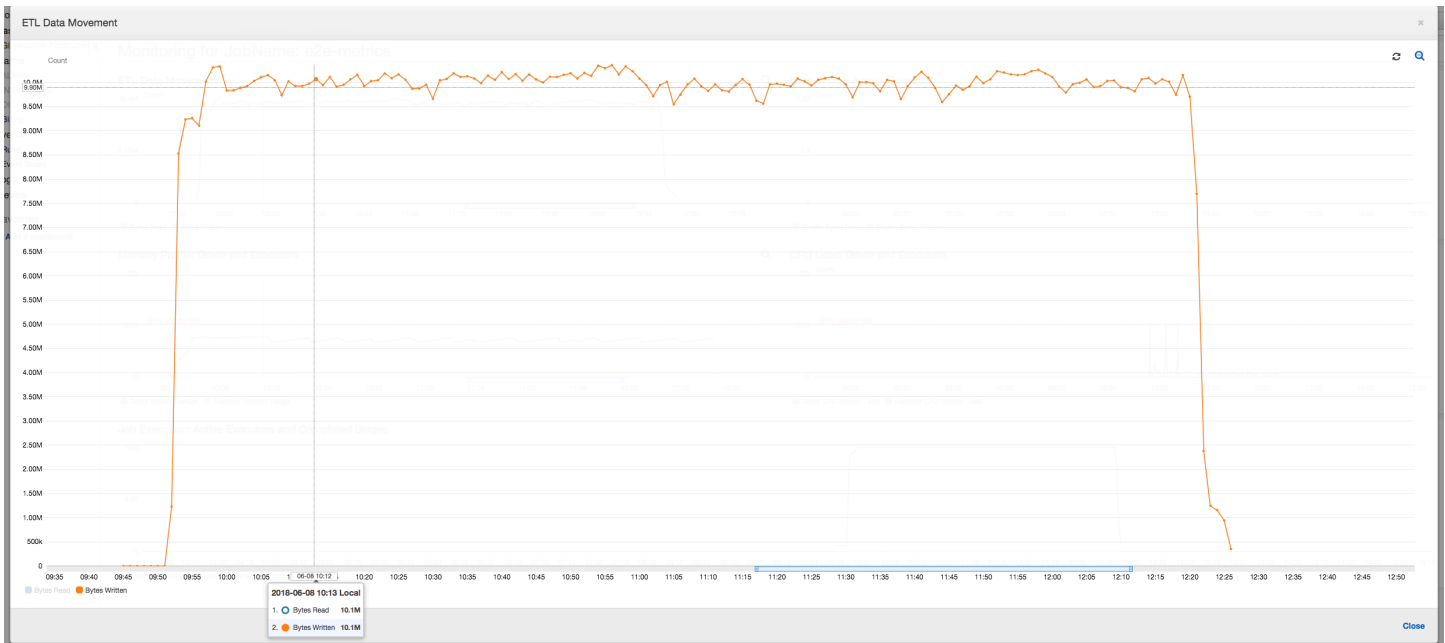
```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"],
  "recurse":True, 'groupFiles': 'inPartition'}, format="json")
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type
  = "s3", connection_options = {"path": output_path}, format = "parquet",
  transformation_ctx = "datasink")
```

AWS Glue 작업 프로파일에서 메모리 프로파일 및 ETL 데이터 이동을 모니터링할 수 있습니다.

이 드라이버는 AWS Glue 작업의 전체 기간 동안 메모리 사용량의 50%인 임계값 미만으로 실행됩니다. 실행기가 Amazon S3에서 데이터를 스트리밍하고 처리한 후 Amazon S3에 씁니다. 결과적으로 모든 시점에서 5%보다 적은 양의 메모리가 사용됩니다.



아래 데이터 이동 프로파일은 작업이 진행됨에 따라 모든 실행기가 마지막 1분 동안 **읽고 쓴 총** Amazon S3 바이트 수를 보여줍니다. 둘 다 모든 실행기에서 데이터가 스트리밍될 때 유사한 패턴을 따릅니다. 이 작업은 세 시간 내에 100만 개에 이르는 모든 파일에 대한 처리를 완료합니다.



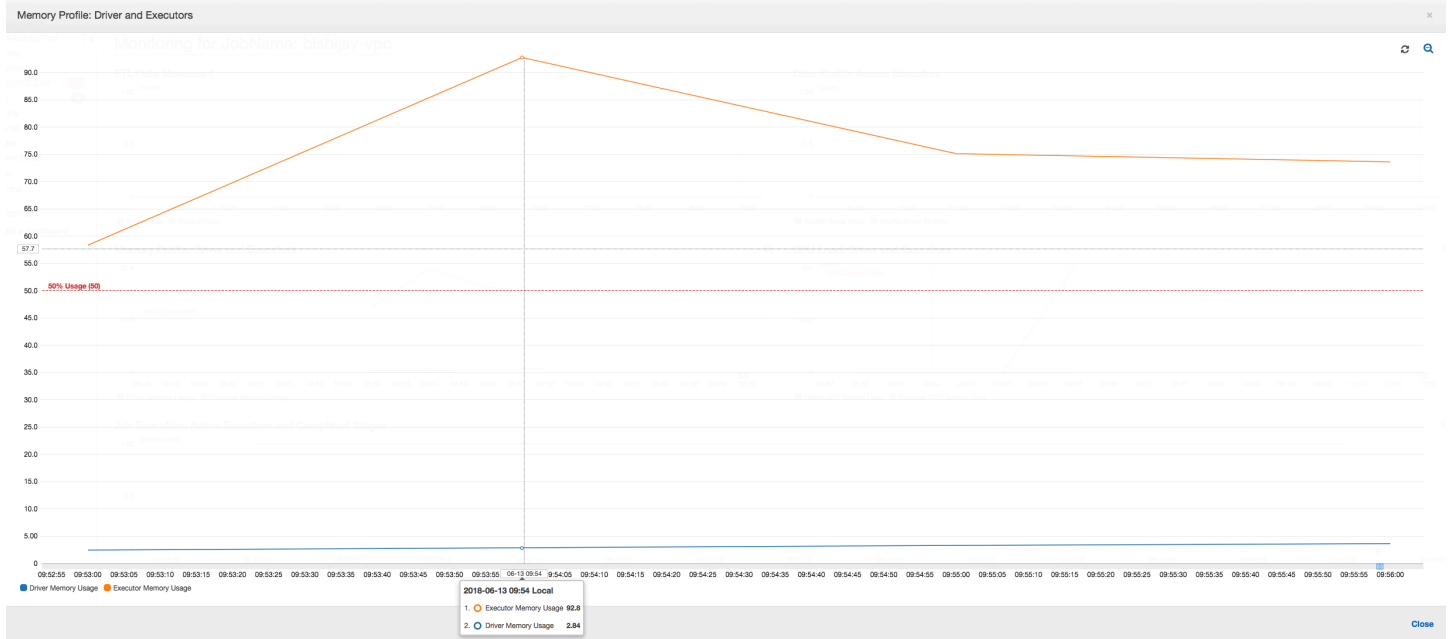
실행기 OOM 예외 사항 디버깅

이 시나리오에서는 Apache Spark 실행기에서 발생할 수 있는 OOM 예외 사항을 디버깅하는 방법을 알아볼 수 있습니다. 다음 코드는 Spark MySQL 리더를 사용하여 약 3400만개 행으로 이루어진 대형 테이블을 Spark 데이터 프레임으로 읽어 들입니다. 그런 다음 Parquet 포맷으로 Amazon S3에 씁니다. 연결 속성을 제공하고 기본 Spark 구성을 사용하여 테이블을 읽을 수 있습니다.

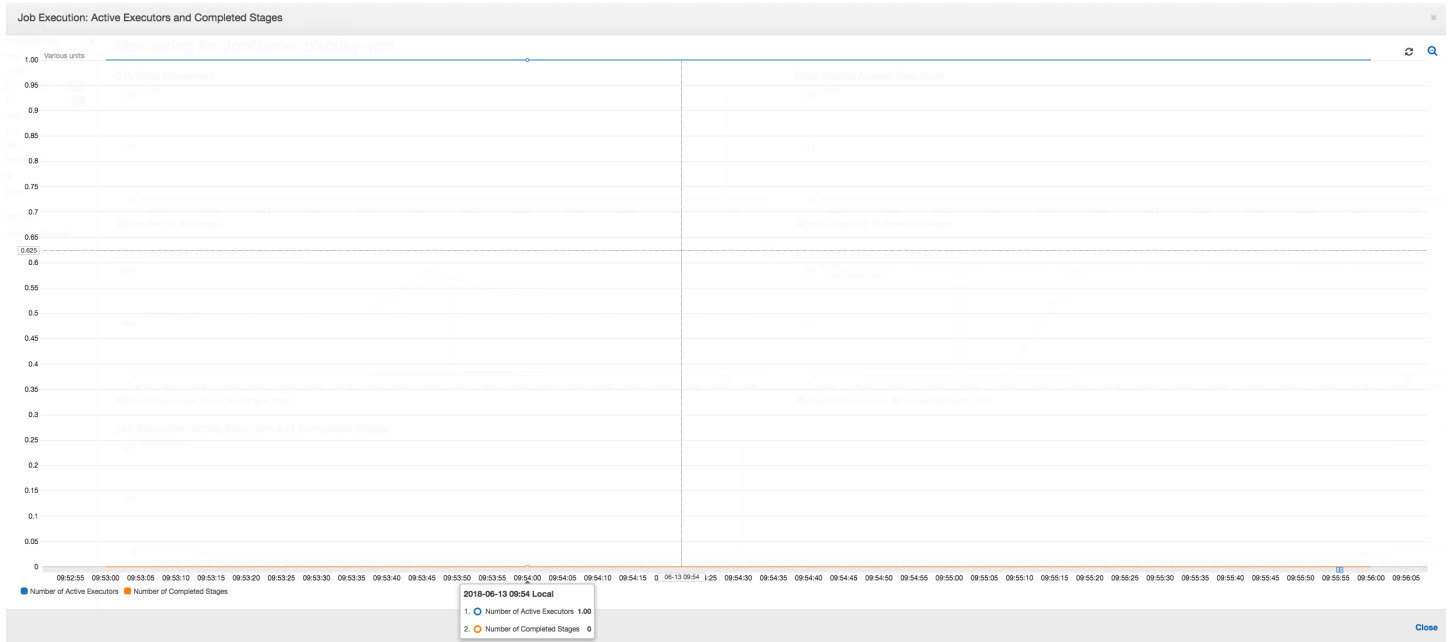
```
val connectionProperties = new Properties()
connectionProperties.put("user", user)
connectionProperties.put("password", password)
connectionProperties.put("Driver", "com.mysql.jdbc.Driver")
val sparkSession = glueContext.sparkSession
val dfSpark = sparkSession.read.jdbc(url, tableName, connectionProperties)
dfSpark.write.format("parquet").save(output_path)
```

AWS Glue 콘솔에서 프로파일링된 지표 시각화

메모리 사용량 그래프의 경사가 양수이고 50%를 넘는 경우 다음 지표를 내보내기 전에 작업이 실패하면 메모리 부족이 실패 원인이 될 수 있습니다. 다음 그래프는 1분 실행 기간 내에 모든 실행기의 **평균 메모리 사용률**이 50%를 급격하게 초과함을 보여줍니다. 이 사용률은 최대 92%에 도달하며 해당 실행기를 실행 중인 컨테이너가 Apache Hadoop YARN에 의해 중지됩니다.



다음 그래프에서처럼, 작업이 실패할 때까지 항상 **단일 실행기**가 실행 중입니다. 이는 새 실행기가 시작되어 중지된 실행기를 대체하기 때문입니다. JDBC 데이터 원본 읽기는 기본적으로 병렬화되지 않습니다. 그 이유는 한 열에서 테이블을 분할하고 여러 연결을 열어야 하기 때문입니다. 따라서 실행기 하나만 전체 테이블에서 순차적으로 읽혀집니다.



다음 그래프에서처럼, Spark는 작업 실패 전에 새 작업 시작을 4회 시도합니다. 세 개 실행기의 **메모리 프로파일**을 볼 수 있습니다. 각 실행기가 모든 메모리를 신속히 소비합니다. 네 번째 실행기에서 메모리 부족이 발생하여 작업이 실패합니다. 따라서 이 작업의 측정치가 즉시 보고되지 않습니다.



다음 이미지와 같이, OOM 예외로 인해 작업이 실패했던 AWS Glue 콘솔에서 오류 문자열을 확인할 수 있습니다.

Run ID	Retry attempt	Run status	Error	Logs	Error logs	Execution time	Timeout	Delay	Triggered by	Start time	End time
j_f_3be21910a910a2e4d411870a1...	-	Failed	org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 0.0 (TID 3, ip-10-1-2-96.ec2.internal, executor 9) ExecutionFailed (executor 4 failed) caused by one of the running tasks) Reason: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead			4 mins	2880 mins			13 June 2018 9:32 AM UT...	13 June 2018 9:32 AM UT...
j_f_4fc7d2723c5d834e90cd0e2f5...	-	Failed	org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 0.0 (TID 3, ip-10-1-2-96.ec2.internal, executor 9) ExecutionFailed (executor 4 failed) caused by one of the running tasks) Reason: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead			0 secs	2880 mins			13 June 2018 9:48 AM UT...	13 June 2018 9:50 AM UT...
j_f_d70a0e828d0e7589a8152d84...	-	Succeeded				2 mins	2880 mins			13 June 2018 9:32 AM UT...	13 June 2018 9:44 AM UT...
j_f_d4c857823082befad919f16a2...	-	Succeeded				2 mins	2880 mins			13 June 2018 8:57 AM UT...	13 June 2018 9:09 AM UT...
j_f_7a0d552d68b36bcd53bbe745...	-	Failed	org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 0.0 (TID 3, ip-10-1-2-96.ec2.internal, executor 9) ExecutionFailed (executor 4 failed) caused by one of the running tasks) Reason: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead			1 hr, 8 mins	2880 mins			12 June 2018 5:15 PM UT...	12 June 2018 6:31 PM UT...

작업 출력 로그: 실행기 OOM 예외 사항의 결과를 추가로 확인하려면 CloudWatch Logs를 살펴봅니다. **Error**를 검색하면 지표 대시보드처럼 거의 동일한 기간 내에 네 개 실행기가 중지됨을 확인할 수 있습니다. 이들 실행기는 메모리 제한을 초과할 때 YARN에 의해 모두 종료됩니다.

실행기 1

```

18/06/13 16:54:29 WARN YarnAllocator: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 ERROR YarnClusterScheduler: Lost executor 1 on ip-10-1-2-175.ec2.internal: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:54:29 WARN TaskSetManager: Lost task 0.0 in stage 0.0 (TID 0, ip-10-1-2-175.ec2.internal, executor 1): ExecutorLostFailure (executor 1

```

```
exited caused by one of the running tasks) Reason: Container killed by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
```

실행기 2

```
18/06/13 16:55:35 WARN YarnAllocator: Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 ERROR YarnClusterScheduler: Lost executor 2 on ip-10-1-2-16.ec2.internal: Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:55:35 WARN TaskSetManager: Lost task 0.1 in stage 0.0 (TID 1, ip-10-1-2-16.ec2.internal, executor 2): ExecutorLostFailure (executor 2 exited caused by one of the running tasks) Reason: Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
```

실행기 3

```
18/06/13 16:56:37 WARN YarnAllocator: Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 ERROR YarnClusterScheduler: Lost executor 3 on ip-10-1-2-189.ec2.internal: Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:56:37 WARN TaskSetManager: Lost task 0.2 in stage 0.0 (TID 2, ip-10-1-2-189.ec2.internal, executor 3): ExecutorLostFailure (executor 3 exited caused by one of the running tasks) Reason: Container killed by YARN for exceeding memory limits. 5.8 GB of 5.5 GB physical memory used. Consider boosting spark.yarn.executor.memoryOverhead.
```

실행기 4

```

18/06/13 16:57:18 WARN YarnAllocator: Container killed by YARN for exceeding
memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 WARN YarnSchedulerBackend$YarnSchedulerEndpoint: Container killed
by YARN for exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider
boosting spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 ERROR YarnClusterScheduler: Lost executor 4 on
ip-10-1-2-96.ec2.internal: Container killed by YARN for exceeding
memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.
18/06/13 16:57:18 WARN TaskSetManager: Lost task 0.3 in stage 0.0 (TID 3,
ip-10-1-2-96.ec2.internal, executor 4): ExecutorLostFailure (executor 4 exited
caused by one of the running tasks) Reason: Container killed by YARN for
exceeding memory limits. 5.5 GB of 5.5 GB physical memory used. Consider boosting
spark.yarn.executor.memoryOverhead.

```

AWS Glue 동적 프레임을 사용하여 가져오기 크기 설정 수정

Spark JDBC 가져오기 크기에 대한 기본 구성이 0이므로 JDBC 테이블을 읽는 동안 실행기에서 메모리 부족이 발생했습니다. 이는 Spark가 한 번에 한 개씩 행을 스트리밍하더라도 Spark 실행기의 JDBC 드라이버가 데이터베이스의 3400만개 행을 함께 가져와서 캐시하려고 함을 의미합니다. Spark에서 가져오기 크기 파라미터를 0 이외의 기본값으로 설정하여 이 시나리오를 방지할 수 있습니다.

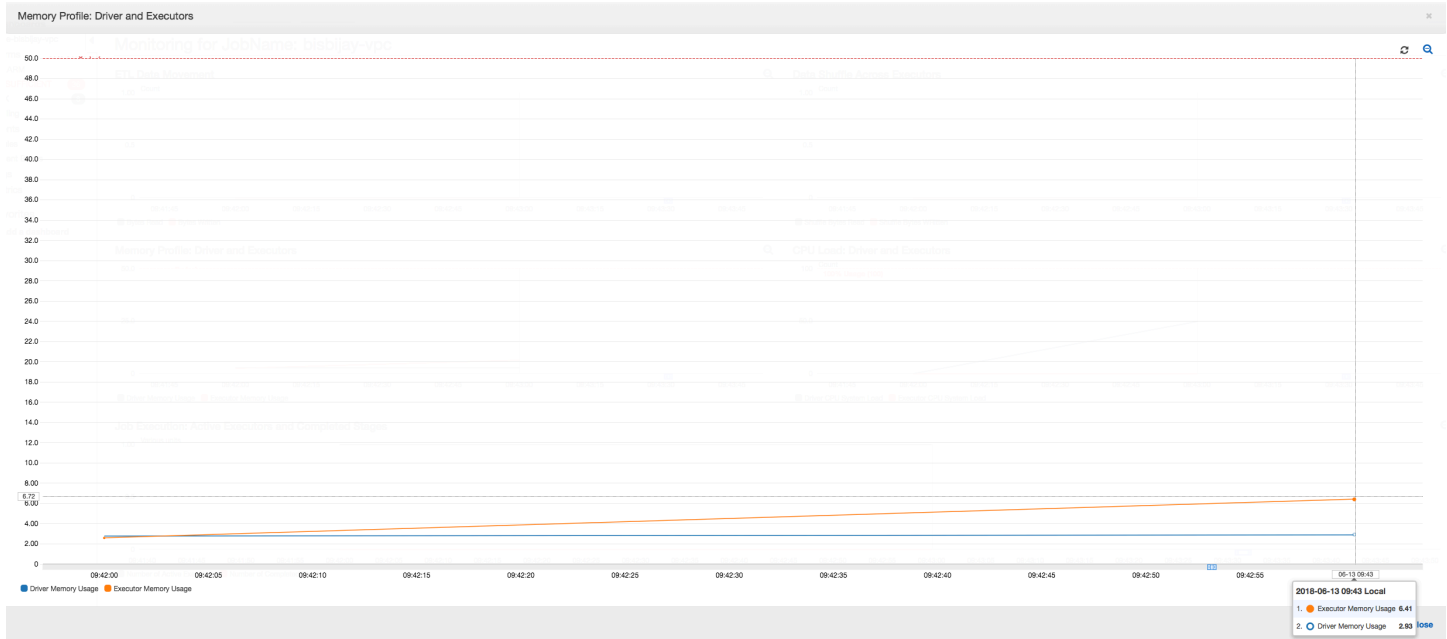
또한 AWS Glue 동적 프레임을 대신 사용하여 이 문제를 해결할 수도 있습니다. 기본적으로 동적 프레임에서는 가져오기 크기로 1,000개 행이 사용되는데, 일반적으로 충분한 값입니다. 결과적으로 실행기에 사용되는 메모리의 양이 총 메모리의 7%를 초과하지 않습니다. AWS Glue 작업은 단일 실행기만으로 2분 안에 완료됩니다. AWS Glue 동적 프레임을 사용하는 것이 권장되는 접근 방식이지만, Apache Spark fetchsize 속성을 사용하여 가져오기 크기를 설정할 수도 있습니다. [Spark SQL, DataFrames 및 데이터 세트 가이드](#)를 참조하십시오.

```

val (url, database, tableName) = {
  ("jdbc_url", "db_name", "table_name")
}
val source = glueContext.getSource(format, sourceJson)
val df = source.getDynamicFrame
glueContext.write_dynamic_frame.from_options(frame = df, connection_type = "s3",
connection_options = {"path": output_path}, format = "parquet", transformation_ctx =
"datasink")

```

정상 프로파일링된 측정치: **실행기 메모리**(AWS Glue 동적 프레임 사용)가 다음 이미지와 같이 안전 임계값을 절대로 초과하지 않습니다. 데이터베이스에서 행을 읽고 어느 시점에든 JDBC 드라이버에 1,000개 행만 캐시합니다. 메모리 부족 예외는 발생하지 않습니다.



까다로운 단계와 스트래글러 작업 디버깅

AWS Glue 작업 프로파일링을 사용하여 추출, 변환 및 로드(ETL) 작업에서 까다로운 단계와 스트래글러 작업을 식별할 수 있습니다. 스트래글러 작업은 AWS Glue 작업의 단계에서 나머지 작업보다 더 오래 걸립니다. 따라서 해당 단계를 완료하는 데 더 오래 걸리므로 작업의 총 실행 시간도 지연됩니다.

작은 입력 파일을 큰 출력 파일로 병합

스트래글러 작업은 여러 작업들 간에 작업량이 균일하지 않게 배포되거나, 데이터 스큐로 인해 한 작업에서 더 많은 데이터를 처리하는 경우에 발생할 수 있습니다.

다음 코드(Apache Spark의 일반 패턴)를 프로파일링하여 다량의 작은 파일을 큰 출력 파일로 병합할 수 있습니다. 예를 들면, 입력 데이터 세트는 JSON Gzip 압축 파일 32GB입니다. 출력 데이터 세트는 거의 190GB에 이르는 압축되지 않은 JSON 파일입니다.

프로파일링된 코드는 다음과 같습니다.

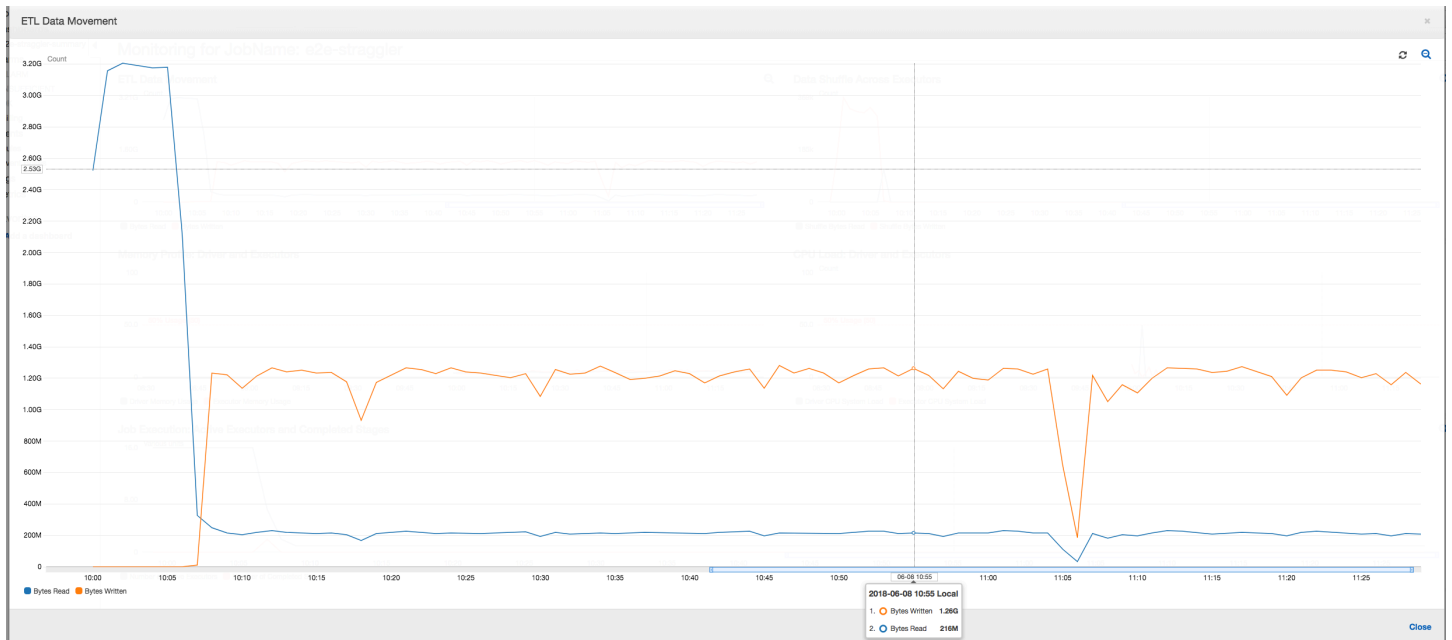
```
datasource0 = spark.read.format("json").load("s3://input_path")
df = datasource0.coalesce(1)
df.write.format("json").save(output_path)
```

AWS Glue 콘솔에서 프로파일링된 지표 시각화

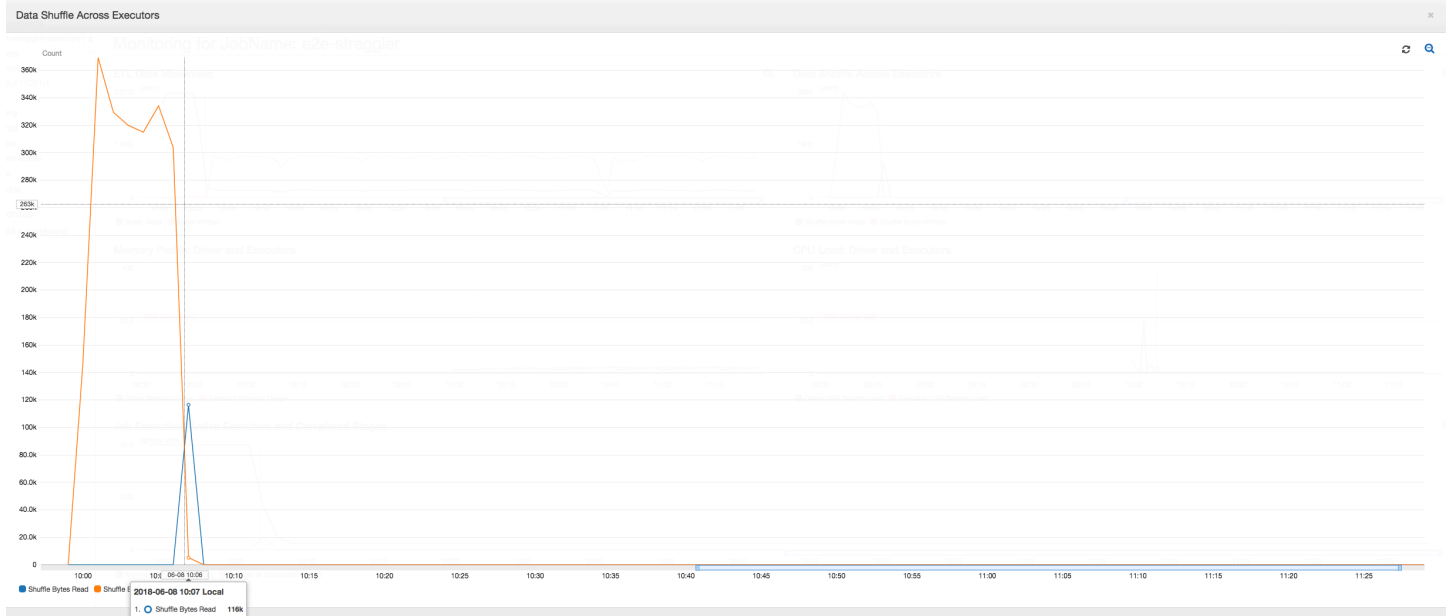
작업을 프로파일링하여 네 가지 지표 세트를 살펴볼 수 있습니다.

- ETL 데이터 이동
- 실행기간의 데이터 셔플
- 작업 실행
- 메모리 프로파일

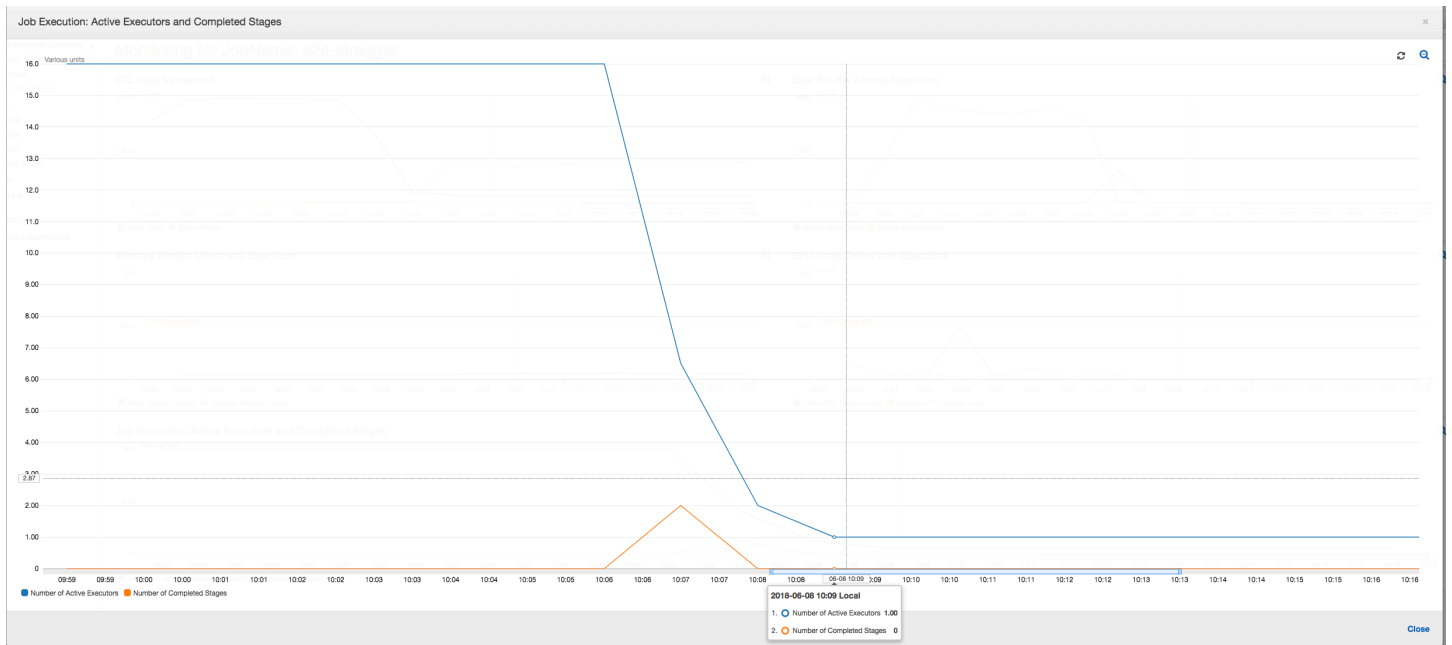
ETL 데이터 이동: ETL 데이터 이동 프로파일에서는 처음 6분 안에 완료되는 첫 번째 단계에서 모든 실행기가 바이트를 매우 빠르게 **읽습니다**. 하지만 총 작업 실행 시간은 거의 1시간으로, 이 시간 중 대부분을 데이터 **쓰기**가 차지합니다.



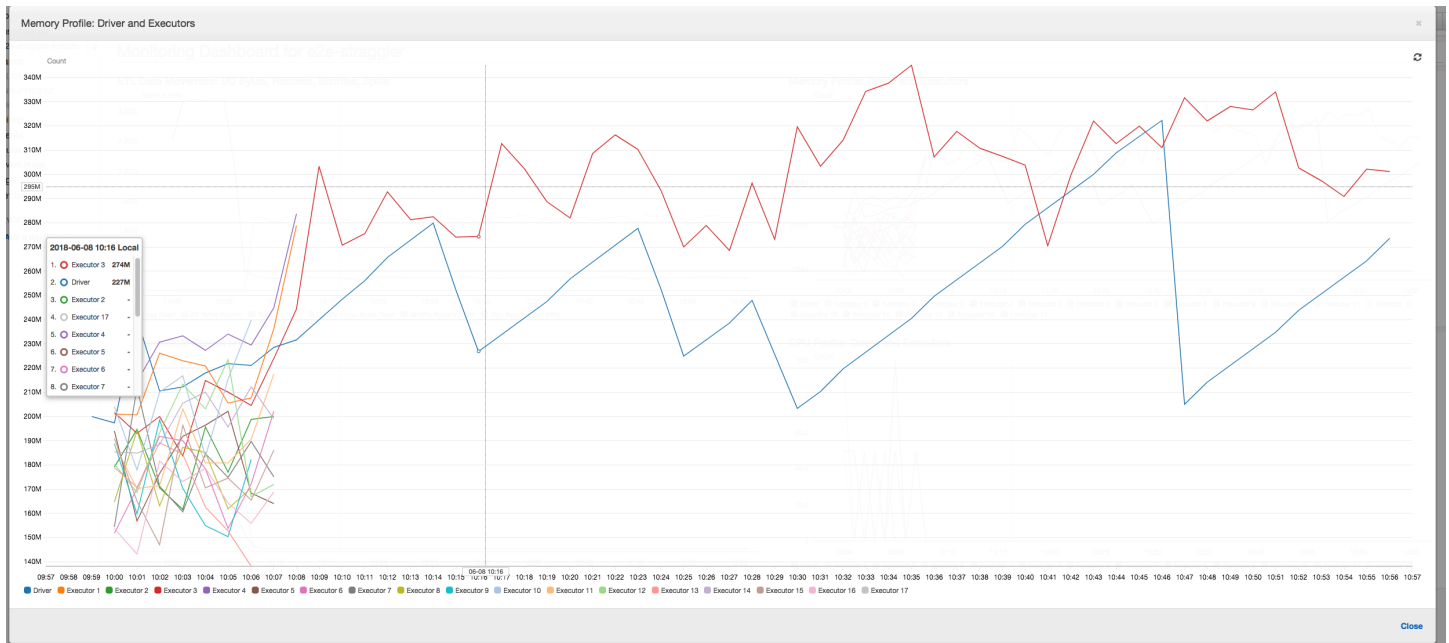
실행기간 데이터 셔플: 셔플링 중에 **읽고 쓴** 바이트 수도 [작업 실행(Job Execution)] 및 [데이터 셔플(Data Shuffle)] 지표에서 알 수 있듯이 2단계가 끝나기 전 스파이크를 나타냅니다. 모든 실행기의 데이터 셔플 후에는 실행기 번호 3에서만 읽기 및 쓰기가 진행됩니다.



작업 실행: 아래 그래프와 같이, 다른 모든 실행기는 유휴 상태이며 최종적으로 10:09에 중지됩니다. 이 시점에는 총 실행기 수가 1로 감소됩니다. 이는 실행기 번호 3이 실행 시간이 가장 오래 걸리고 작업 실행 시간의 대부분을 차지하는 스트래글러 작업으로 이루어짐을 보여줍니다.



메모리 프로필: 처음 두 단계 이후에는 실행기 번호 3에서 메모리를 적극적으로 소비하며 데이터를 처리합니다. 나머지 실행기는 단순 유휴 상태이거나 처음 두 단계가 완료되고 나서 잠시 후에 중지되었습니다.



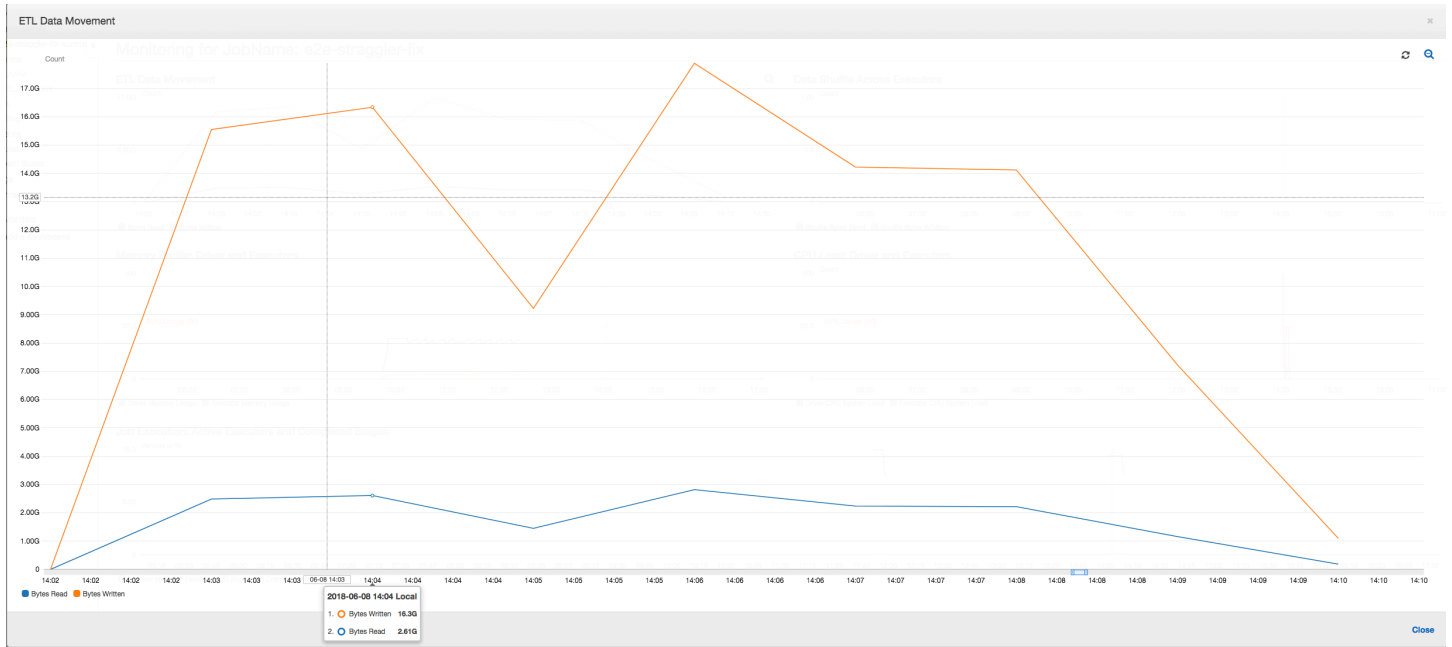
그룹화를 사용하여 스트래글링 실행기 수정

AWS Glue의 그룹화 기능을 사용하면 실행기가 뒤처지는 것을 방지할 수 있습니다. 그룹화를 사용하면 데이터를 모든 실행기에 균일하게 분포하고 클러스터의 모든 사용 가능한 실행기를 사용하여 파일을 더 큰 파일로 병합할 수 있습니다. 자세한 내용은 [입력 파일을 더 큰 그룹에서 읽기](#) 단원을 참조하십시오.

AWS Glue 작업의 ETL 데이터 이동을 확인하려면 그룹화가 활성화된 상태에서 다음과 같은 코드를 프로파일링합니다.

```
df = glueContext.create_dynamic_frame_from_options("s3", {'paths': ["s3://input_path"],
    "recurse":True, 'groupFiles': 'inPartition'}, format="json")
datasink = glueContext.write_dynamic_frame.from_options(frame = df, connection_type =
    "s3", connection_options = {"path": output_path}, format = "json", transformation_ctx
    = "datasink4")
```

ETL 데이터 이동: 이제 작업 실행 시간 전체에 걸쳐 데이터 쓰기과 데이터 읽기를 병렬로 스트리밍합니다. 따라서 작업이 8분(이전보다 훨씬 빠름) 내에 완료됩니다.



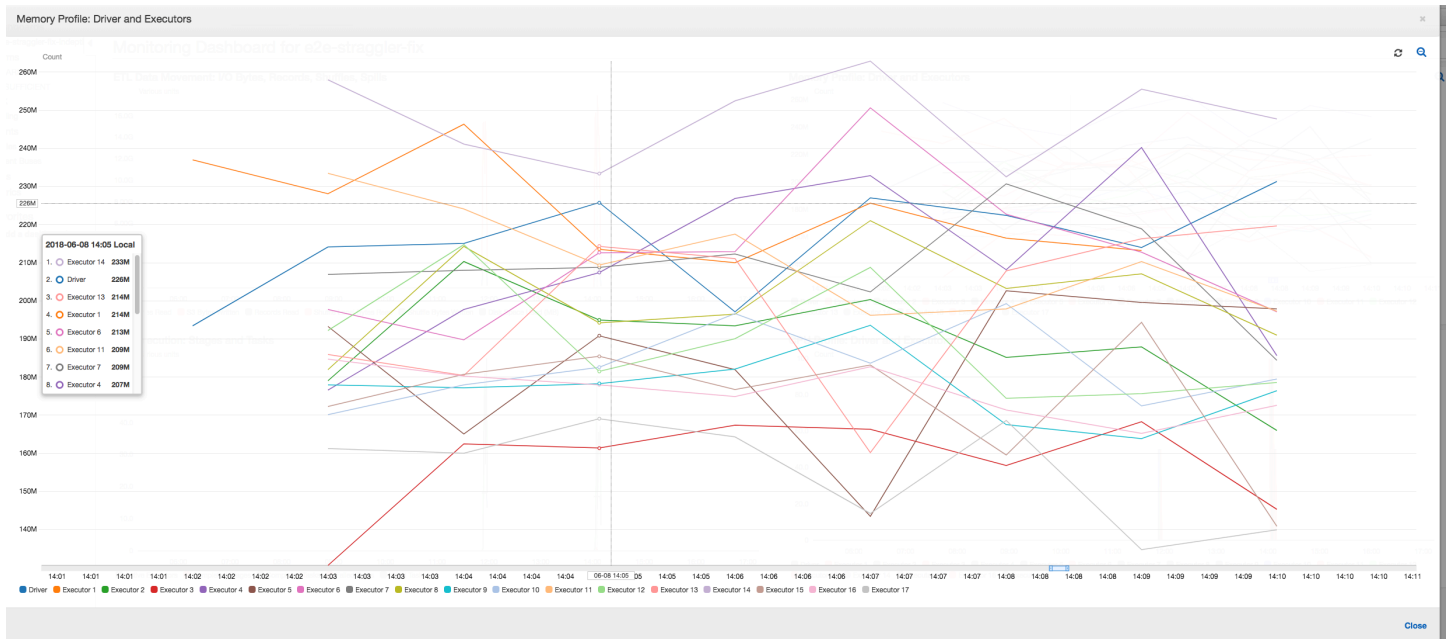
실행기간의 데이터 셔플: 입력 파일이 그룹화 기능을 사용하여 읽는 동안 병합되므로 데이터를 읽은 후에 비용이 많이 들던 데이터 셔플이 발생하지 않습니다.



작업 실행: 작업 실행 측정치는 데이터 실행 및 처리 중인 총 활성 실행기 수가 상당히 일관되게 유지됨을 보여줍니다. 작업에 단일 스트래글러가 없습니다. 모든 실행기가 활성 상태이며 작업이 완료될 때까지 활성 상태로 유지됩니다. 실행기간에 중간 데이터 셔플이 없으므로 작업에 단일 단계만 존재합니다.



메모리 프로파일: 이 지표는 모든 실행기의 **활성 메모리 소비**를 표시하여 모든 실행기에서 활동이 존재함을 재확인합니다. 데이터가 동시에 스트리밍되고 작성되므로, 모든 실행기의 총 메모리 공간이 모든 실행기의 안전 임계값보다 훨씬 낮고 거의 균일합니다.



여러 작업의 진행 상황 모니터링

여러 AWS Glue 작업을 함께 프로파일링하고 작업 간에 데이터 흐름을 모니터링할 수 있습니다. 이 모니터링은 일반적인 워크플로우 패턴으로, 개별 작업 진행 상황, 데이터 처리 백로그, 데이터 재처리 및 작업 북마크를 모니터링해야 합니다,

주제

- [프로파일링된 코드](#)
- [AWS Glue 콘솔에서 프로파일링된 지표 시각화](#)
- [파일 처리 수정](#)

프로파일링된 코드

이 워크플로우에는 두 가지 작업, 즉 입력 작업과 출력 작업이 있습니다. 입력 작업은 정기 트리거를 사용하여 30분마다 실행하도록 예약되어 있습니다. 출력 작업은 입력 작업의 각각의 성공적인 실행 이후에 실행하도록 예약되어 있습니다. 이러한 예약된 작업은 작업 트리거를 사용하여 제어됩니다.

Triggers A trigger starts a job when it fires.

Trigger name	Trigger type	Trigger status	Trigger parameters	Jobs to trigger
<input type="checkbox"/> e2e-bookmark-input	Schedule	ACTIVATED	Every 15 minutes	e2ebookmark-input
<input type="checkbox"/> e2e-bookmark-output	Job events	ACTIVATED	Job events: e2ebookmark-input	e2e-bookmark

입력 작업: 이 작업은 Amazon Simple Storage Service(Amazon S3) 위치에서 데이터를 읽고 ApplyMapping을 사용하여 변환한 다음, 스테이징 Amazon S3 위치에 씁니다. 다음 코드는 입력 작업용으로 프로파일링된 코드입니다.

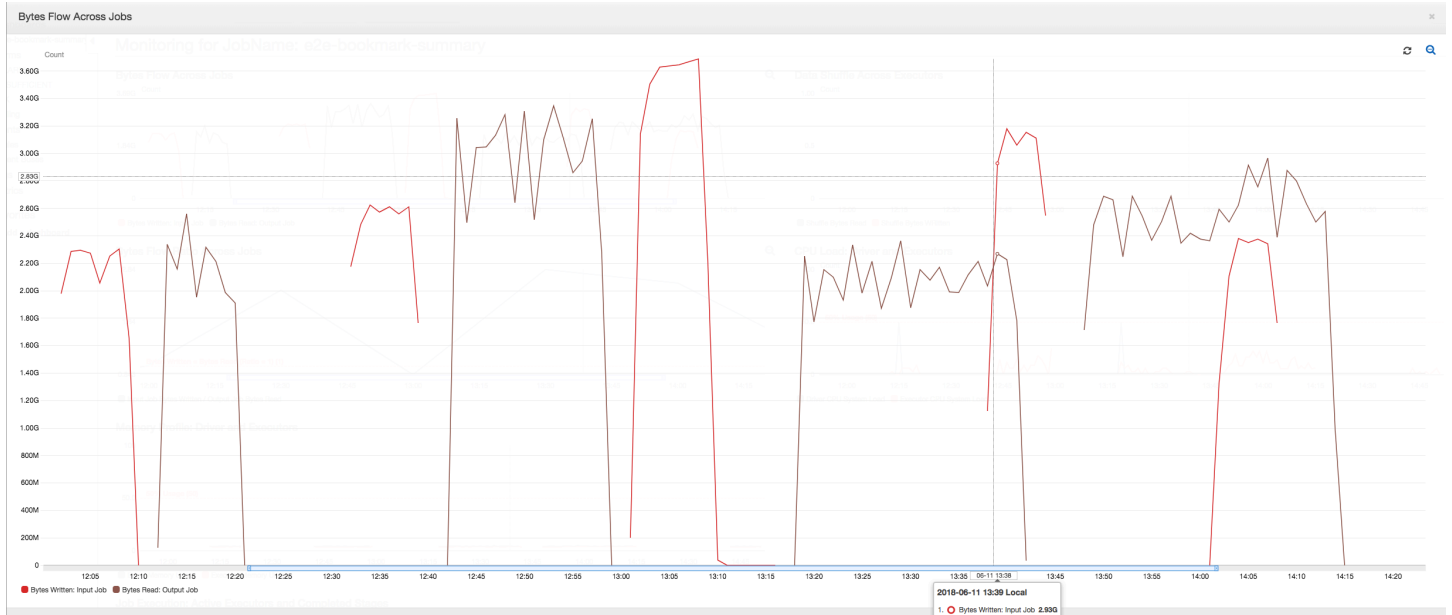
```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
  connection_options = {"paths": ["s3://input_path"],
  "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
  connection_type = "s3", connection_options = {"path": staging_path, "compression":
  "gzip"}, format = "json")
```

출력 작업: 이 작업은 Amazon S3의 스테이징 위치에서 입력 작업의 출력을 읽고, 다시 변환한 다음, 대상에 씁니다.

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
  connection_options = {"paths": [staging_path],
  "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [map_spec])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
  connection_type = "s3", connection_options = {"path": output_path}, format = "json")
```

AWS Glue 콘솔에서 프로파일링된 지표 시각화

다음 대시보드는 출력 작업에 대한 동일한 타임라인에서 입력 작업의 Amazon S3 바이트 쓰기 지표를 Amazon S3 바이트 읽기 지표에 중첩합니다. 이 타임라인은 입력 작업과 출력 작업의 서로 다른 작업 실행을 보여줍니다. 입력 작업(빨간색으로 표시됨)이 30분마다 시작됩니다. 출력 작업(갈색으로 표시됨)은 입력 작업 완료 시 시작되며, 최대 동시성은 1입니다.



이 예에서는, [작업 북마크](#)가 활성화되지 않습니다. 변환 컨텍스트를 사용하여 스크립트 코드에서 작업 북마크를 활성화하지 않습니다.

작업 기록: 입력 및 출력 작업은 오후 12:00부터 시작되는 기록 탭에서처럼 여러 번 실행되었습니다.

AWS Glue 콘솔의 입력 작업은 다음과 같습니다.

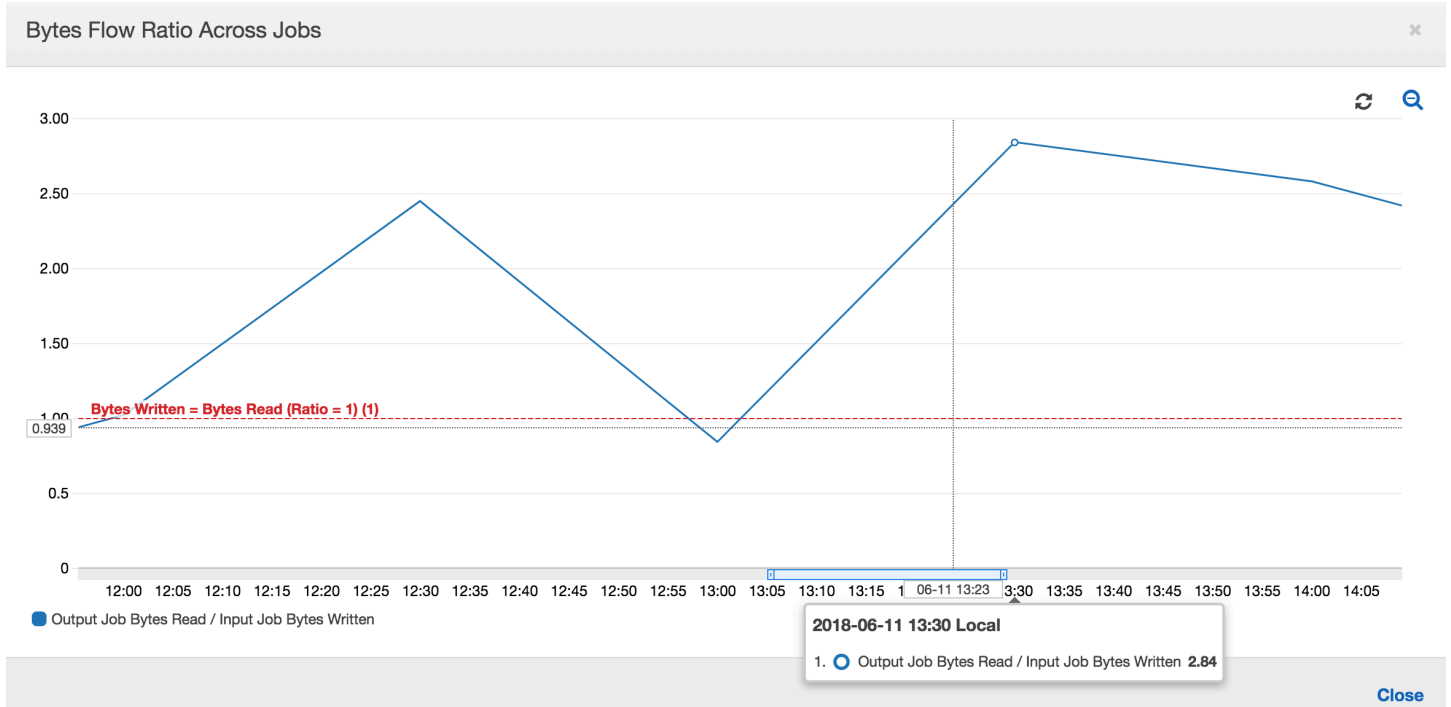
Run ID	Retry attempt	Run status	Error	Logs	Error logs	Execution time	Timeout	Delay	Triggered by	Start time	End time
j_f_0ce47b1a561051f6c9ae96e...	-	Succeeded		Logs		8 mins	2880 mins		e2e-bookmark-input	11 June 2018 2:30 PM UT...	11 June 2018 2:40 PM UT...
j_f_1b49ecdf73dd7614cca2f4274...	-	Succeeded		Logs		8 mins	2880 mins		e2e-bookmark-input	11 June 2018 2:00 PM UT...	11 June 2018 2:10 PM UT...
j_f_07fe4b25350ce516d89086821e...	-	Succeeded		Logs		7 mins	2880 mins		e2e-bookmark-input	11 June 2018 1:30 PM UT...	11 June 2018 1:46 PM UT...
j_f_029349097744be2abf655f6b1...	-	Succeeded		Logs		15 mins	2880 mins		e2e-bookmark-input	11 June 2018 1:00 PM UT...	11 June 2018 1:16 PM UT...

다음 이미지는 출력 작업을 보여줍니다.

Run ID	Retry attempt	Run status	Error	Logs	Error logs	Execution time	Timeout	Delay	Triggered by	Start time	End time
j_f_d2e5ba78770743d373d9dd83...	-	Failed	Max conc...	Logs	Error logs	0 secs	2880 mins		e2e-bookmark-output	11 June 2018 2:11 PM UT...	
j_f_3242babab08a8cb6fcb5df2e3...	-	Succeeded		Logs		27 mins	2880 mins		e2e-bookmark-output	11 June 2018 1:47 PM UT...	11 June 2018 2:15 PM UT...
j_f_c98cccb031be794a2b3a8047b...	-	Succeeded		Logs		24 mins	2880 mins		e2e-bookmark-output	11 June 2018 1:17 PM UT...	11 June 2018 1:43 PM UT...
j_f_0029a3c6f66c6395d59c8f965...	-	Succeeded		Logs		17 mins	2880 mins		e2e-bookmark-output	11 June 2018 12:41 PM U...	11 June 2018 12:59 PM U...

첫 번째 작업 실행: 아래 읽고 쓴 데이터 바이트 그래프에서처럼, 12:00 ~ 12:30의 입력 및 출력 작업의 첫 번째 작업 실행은 거의 동일한 곡선하 면적을 보입니다. 이 면적은 입력 작업에서 쓴 Amazon S3 바이트와 출력 작업에서 읽은 Amazon S3 바이트를 나타냅니다. 또한 이 데이터는 작성된 Amazon S3 바이트의 비율(30분 동안의 합계 - 입력 작업의 경우 작업 트리거 빈도)로 확인됩니다. 오후 12:00에 시작된 입력 작업 비율의 데이터 지점도 1입니다.

다음 그래프는 모든 작업 실행에 대한 데이터 흐름 비율을 보여줍니다.



두 번째 작업 실행: 두 번째 작업 실행에서 입력 작업에서 쓴 바이트 수와 출력 작업에서 읽은 바이트 수 사이에는 명확한 차이가 있습니다. (출력 작업에 대한 두 작업 실행의 곡선하 면적을 비교하거나, 입력 작업과 출력 작업의 두 번째 실행 시 면적을 비교합니다.) 읽은 바이트 수와 쓴 바이트 수의 비율은 12:30 ~ 13:00의 두 번째 기간인 30분 동안 입력 작업에 의해 작성된 데이터의 약 2.5배를 출력 작업에서 읽었음을 보여줍니다. 이는 작업 북마크가 활성화되지 않았기 때문에 출력 작업이 입력 작업의 첫 번째 작업 실행에서 나오는 출력을 재처리하기 때문입니다. 비율이 1보다 크면 출력 작업에 의해 데이터의 추가 백로그가 처리되었음을 의미합니다.

세 번째 작업 실행: 입력 작업은 작성된 바이트 수 면에서 일관성이 거의 유지됩니다(빨간색 곡선하 면적 참조). 하지만, 입력 작업의 세 번째 작업 실행 시간이 예상보다 더 길어졌습니다(빨간색 곡선의 긴 후미부 참조). 결과적으로 출력 작업의 세 번째 작업 실행이 늦게 시작되었습니다. 세 번째 작업은 13:00 ~ 13:30의 나머지 30분 동안 스테이징 위치에 누적된 데이터 부분만 처리했습니다. 바이트 흐름의 비율은 입력 작업의 세 번째 작업 실행에 의해 작성된 데이터의 0.83만 처리되었음을 보여줍니다(13:00의 비율 참조).

입력 작업과 출력 작업의 중첩: 입력 작업의 네 번째 작업 실행이 출력 작업의 세 번째 작업 실행이 완료되기 전에 일정에 따라 13:30에 시작되었습니다. 두 작업 실행 사이에 약간의 중첩 부분이 있습니다. 하지만, 출력 작업의 세 번째 작업 실행은 약 13:17에 시작되었을 때 Amazon S3의 스테이징 위치에 나열되어 있는 파일만 캡처합니다. 캡처 데이터는 입력 작업의 첫 번째 작업 실행에서 얻은 모든 데이터 출력으로 구성됩니다. 13:30에서 실제 비율은 약 2.75입니다. 출력 작업의 세 번째 작업 실행에서는 13:30~14:00의 입력 작업 중 네 번째 작업 실행에서 작성된 데이터의 약 2.75x를 처리했습니다.

이러한 이미지에 표시된 바와 같이, 출력 작업은 입력 작업에 대한 모든 이전 작업 실행의 스테이징 위치부터 데이터를 처리하고 있습니다. 결과적으로, 출력 작업에 대한 네 번째 작업 실행이 가장 길며 입력 작업의 전체 다섯 번째 작업 실행과 겹칩니다.

파일 처리 수정

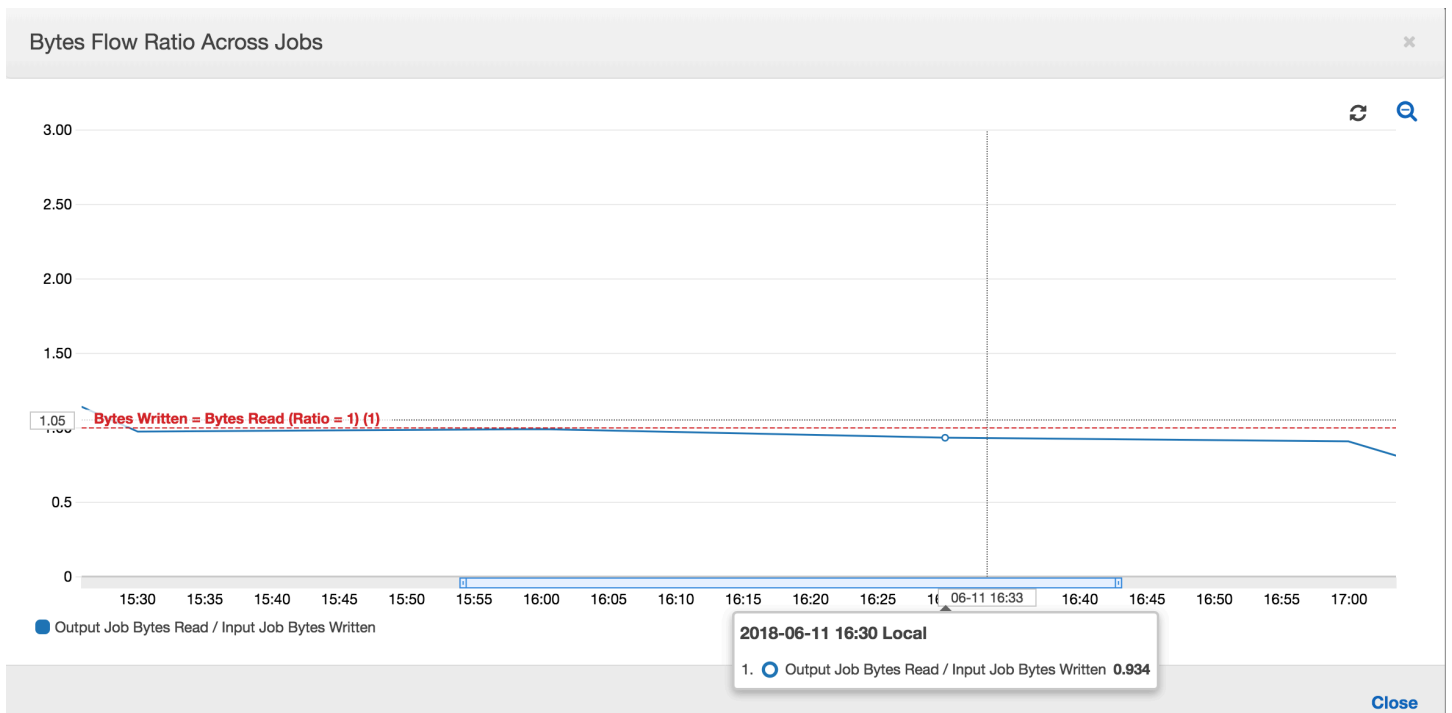
출력 작업에서 출력 작업의 이전 작업 실행에 의해 처리되지 않은 파일만 처리하도록 해야 합니다. 이렇게 하려면 다음과 같이 출력 작업에서 작업 북마크를 활성화하고 변환 컨텍스트를 설정합니다.

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
  connection_options = {"paths": [staging_path],
  "useS3ListImplementation":True,"recurse":True}, format="json", transformation_ctx =
  "bookmark_ctx")
```

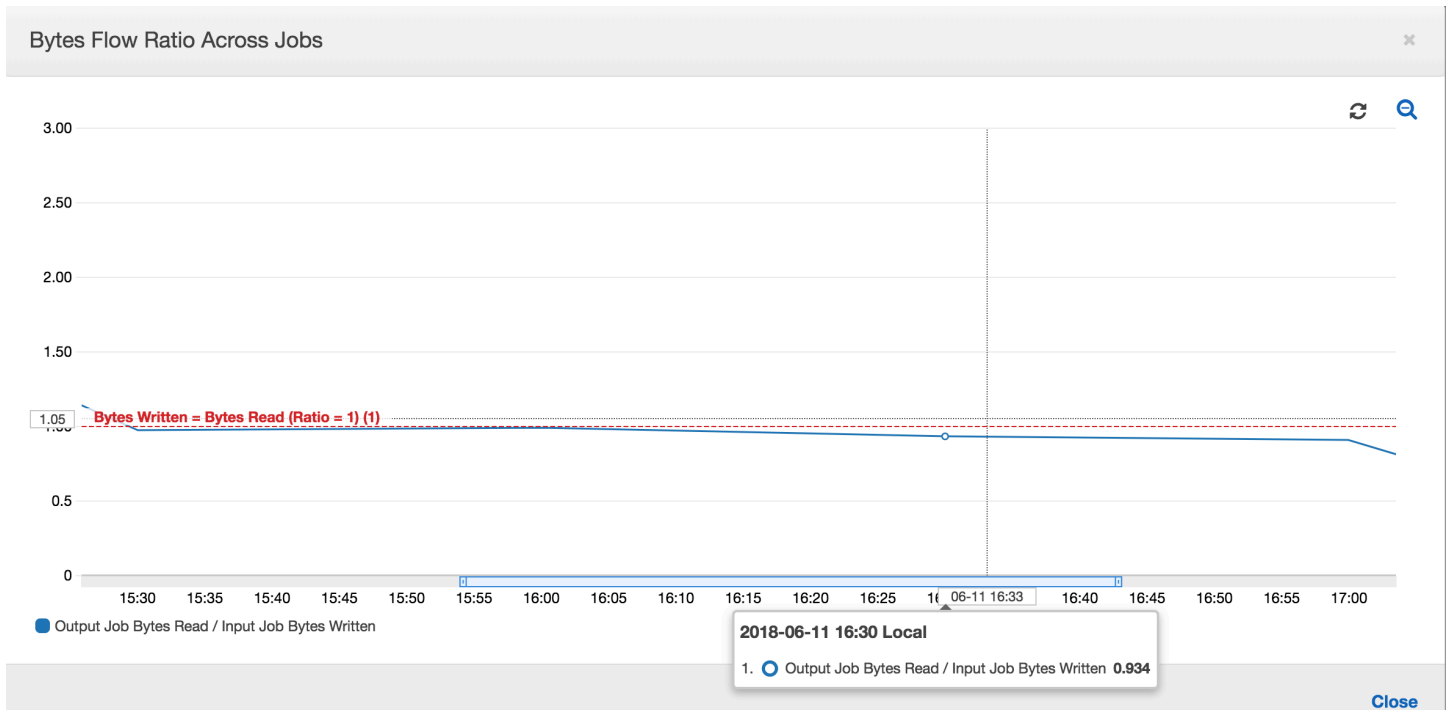
작업 북마크를 활성화한 상태에서는 출력 작업이 입력 작업에 대한 모든 이전 작업 실행의 스테이징 위치에 있는 데이터를 재처리하지 않습니다. 읽고 쓴 데이터를 보여주는 다음 이미지에서는 갈색 곡선 하 면적이 상당히 일관되며 빨간 곡선과 비슷합니다.



추가 데이터 처리가 없으므로 바이트 흐름의 비율도 거의 1에 가깝게 유지됩니다.



다음 입력 작업 실행이 시작되기 전에 출력 작업에 대한 작업 실행이 시작되고 스테이징 위치에 있는 파일을 캡처하여 더 많은 데이터를 스테이징 위치에 넣습니다. 이 작업이 계속되는 동안은 이전 입력 작업 실행에서 캡처한 파일만 처리되며 비율이 계속 1에 가깝게 유지됩니다.



입력 작업이 예상보다 길어지며, 그 결과로서 출력 작업이 두 입력 작업 실행의 스테이징 위치에 있는 파일을 캡처한다고 가정해 보겠습니다. 그러면 해당 출력 작업 실행에 대해 비율이 1보다 높습니다. 하지만, 출력 작업의 다음 작업 실행에서는 출력 작업의 이전 작업 실행에 의해 이미 처리된 파일이 하나도 처리되지 않습니다.

DPU 용량 계획 모니터링

AWS Glue의 작업 측정치를 사용하여 AWS Glue 작업을 확장하는 데 사용할 수 있는 데이터 처리 단위 (DPU) 수를 추정할 수 있습니다.

Note

이 페이지는 AWS Glue 버전 0.9 및 1.0에만 적용됩니다. 최신 버전의 AWS Glue에는 용량 계획 시 추가 고려 사항을 적용하는 비용 절감 기능이 포함되어 있습니다.

주제

- [프로파일링된 코드](#)
- [AWS Glue 콘솔에서 프로파일링된 지표 시각화](#)
- [최적 DPU 용량 결정](#)

프로파일링된 코드

다음 스크립트는 428개의 gzip 압축된 JSON 파일을 포함하는 Amazon Simple Storage Service(Amazon S3) 파티션을 읽습니다. 이 스크립트는 필드 이름에 변경할 매핑을 적용하고 변환한 후 다시 Amazon S3에 Apache Parquet 포맷으로 씁니다. 기본값에 따라 10개 DPU를 프로비저닝하고 이 작업을 실행할 수 있습니다.

```
datasource0 = glueContext.create_dynamic_frame.from_options(connection_type="s3",
  connection_options = {"paths": [input_path],
  "useS3ListImplementation":True,"recurse":True}, format="json")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [(map_spec)])
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1,
  connection_type = "s3", connection_options = {"path": output_path}, format =
  "parquet")
```

AWS Glue 콘솔에서 프로파일링된 지표 시각화

작업 실행 1: 이 작업 실행에서는 클러스터에서 부족 프로비저닝된 DPU가 있는지 알아보는 방법을 보여줍니다. AWS Glue의 작업 실행 기능은 [능동적으로 실행 중인 총 실행기 수](#), [완료된 단계 수](#), 그리고 [최대 필요 실행기 수](#)를 표시합니다.

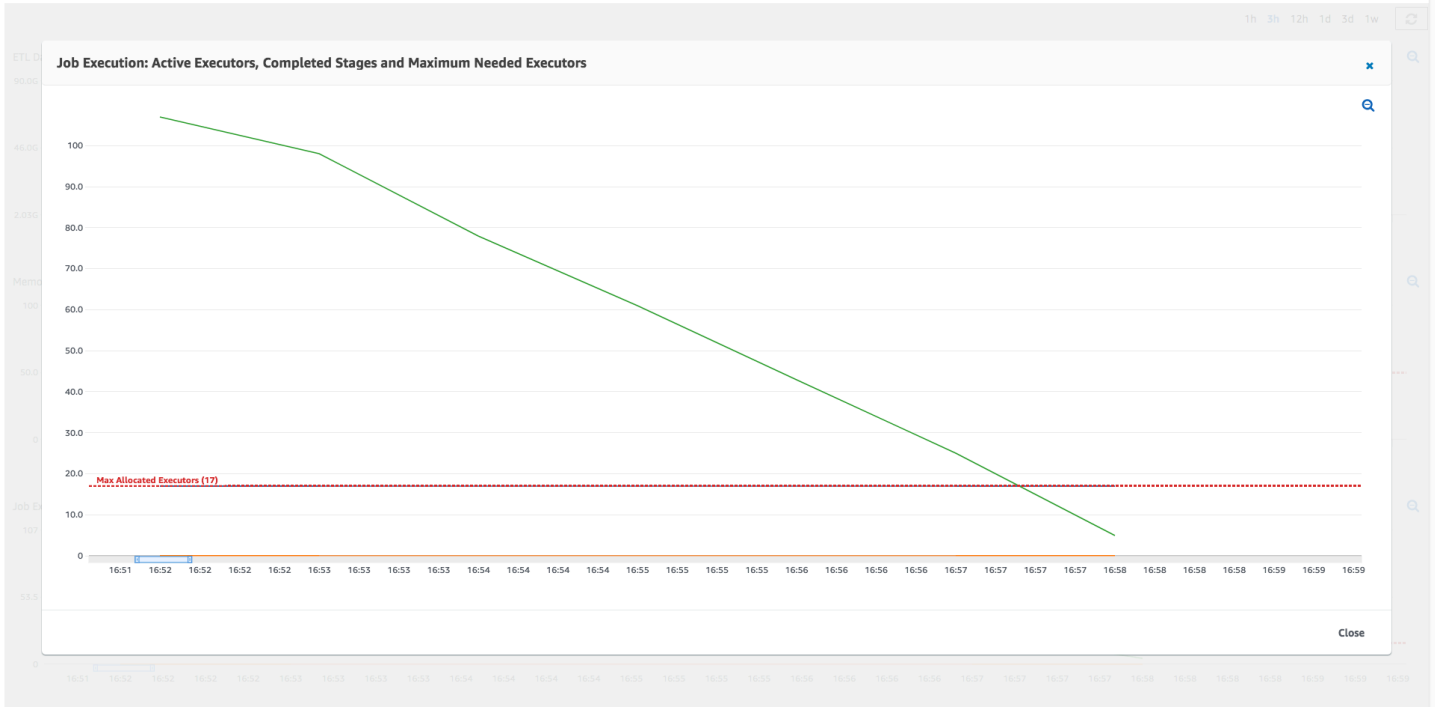
최대 필요 실행기 수는 총 실행 중인 작업 수와 보류 작업 수를 합한 다음 실행기당 작업 수로 나누어서 계산합니다. 이 결과는 현재 부하를 충족하는 데 필요한 총 실행기 수를 나타내는 측정치입니다.

이와는 대조적으로, 능동적으로 실행 중인 실행기 수는 활성 Apache Spark 작업을 실행 중인 실행기 수를 측정합니다. 작업이 진행됨에 따라 최대 필요 실행기가 변경될 수 있으며 일반적으로 보류 작업 대기열이 축소되면서 작업 완료에 가까워질수록 줄어듭니다.

다음 그래프의 가로 빨간색 선은 최대 할당 실행기 수를 표시하며, 이는 작업에 할당하는 DPU 수에 따라 달라집니다. 이 경우, 작업 실행에 대해 10개 DPU를 할당합니다. 하나의 DPU는 관리용으로 예약되어 있습니다. 9개 DPU가 각각 실행기 두 개를 실행하며 실행기 하나가 Spark 드라이버용으로 예약되어 있습니다. Spark 드라이버는 기본 애플리케이션에서 실행됩니다. 그러므로 최대 할당 실행기 수는 $2 \times 9 - 1 = 17$ 개입니다.

Jobs > e2e-dpus

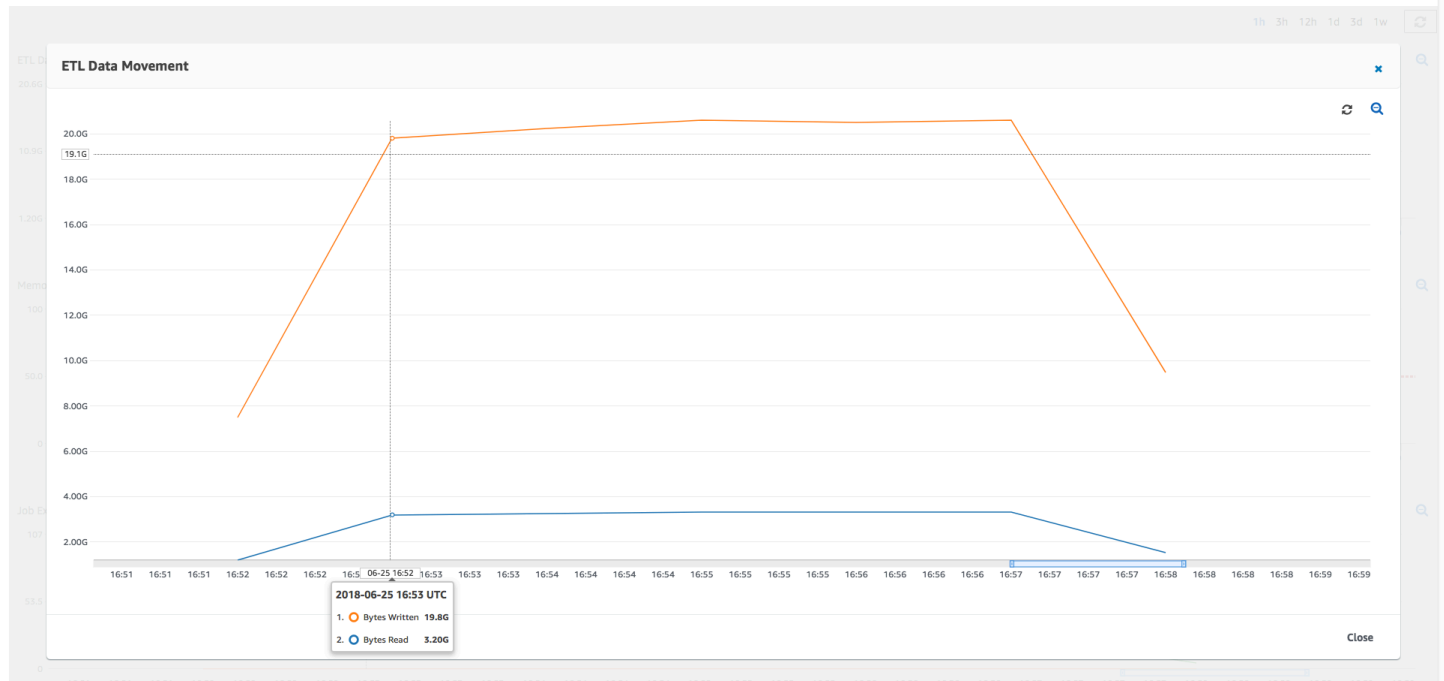
Detailed job metrics



그래프가 표시될 때, 최대 필요 실행기 수는 작업 시작 시 107로 시작하지만, 활성 실행기 수는 17개로 유지됩니다. 이 개수는 DPU 수가 10개인 최대 할당 실행기 수와 동일합니다. 최대 필요 실행기 수와 최대 할당 실행기 수 간의 비율(Spark 드라이버에 대해 두 수치에 모두 1을 더함)은 부족 프로비저닝 계수를 제공합니다. 즉, $108/18 = 6x$. 최대 병렬화로 실행하고 더 빨리 완료하도록 작업을 확장하기 위해 $6(\text{언더프로비저닝 비율}) * 9(\text{현재 DPU 용량} - 1) + \text{DPU } 1\text{개} = \text{DPU } 55\text{개를 프로비저닝할 수 있습니다.}$

AWS Glue 콘솔은 자세한 작업 지표를 원래 할당된 최대 실행기 수를 나타내는 정적 행으로 표시합니다. 이 콘솔은 지표에 대한 작업 정의에서 할당된 최대 실행기를 계산합니다. 반면 자세한 작업 실행 지표의 경우 이 콘솔은 작업 실행 구성에서 할당된 최대 실행기, 특히 작업 실행에 대해 할당된 DPU를 계산합니다. 개별 작업 실행에 대한 지표를 보려면 작업 실행을 선택하고 View run metrics(실행 지표 보기)를 선택합니다.

Jobs > e2e-dpus
Detailed job metrics



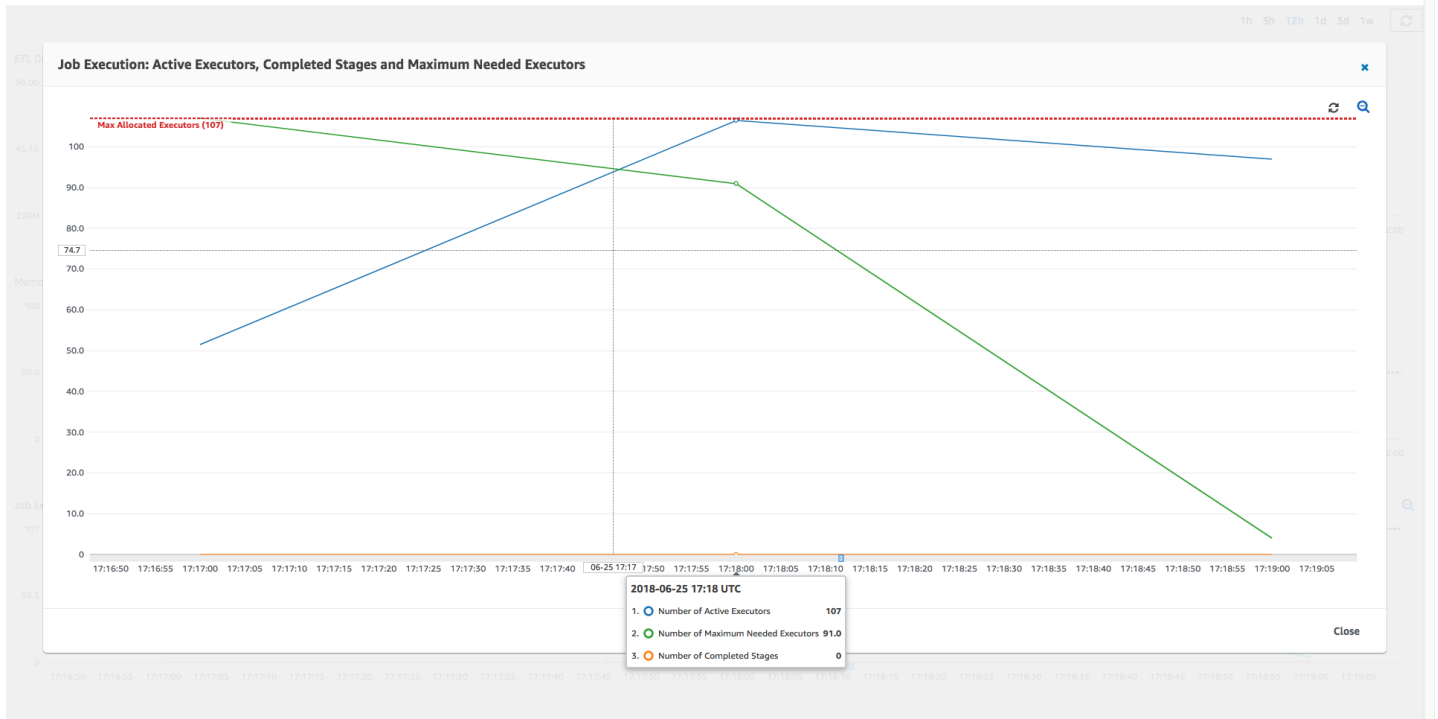
읽고 쓴 Amazon S3 바이트를 살펴보면 작업이 Amazon S3에서 데이터를 스트리밍하고 병렬로 쓰는 데 6분을 모두 소비함을 알 수 있습니다. 할당된 DPU의 모든 코어가 Amazon S3에(서) 읽고 쓰고 있습니다. 최대 필요 실행기 수(107)는 입력 Amazon S3 경로의 파일 수(428)와도 일치합니다. 각 실행기는 Spark 작업 네 개를 시작하여 네 개 입력 파일(JSON gzip 압축됨)을 처리할 수 있습니다.

최적 DPU 용량 결정

이전 작업 실행 결과에 따라, 총 할당된 DPU 수를 55로 늘리고 작업이 어떻게 수행되는지 알아볼 수 있습니다. 작업이 이전 소요 시간의 절반에 해당하는 3분 이내에 완료됩니다. 단기 실행 작업이므로 이 경우에는 작업 확장이 선형적이지 않습니다. 장기 지속 작업이나 다량의 작업을 포함하는 작업(최대 필요 실행기 수가 많음)은 선형에 가까운 DPU 확장 성능 가속화가 도움이 됩니다.

Jobs > e2e-dpus

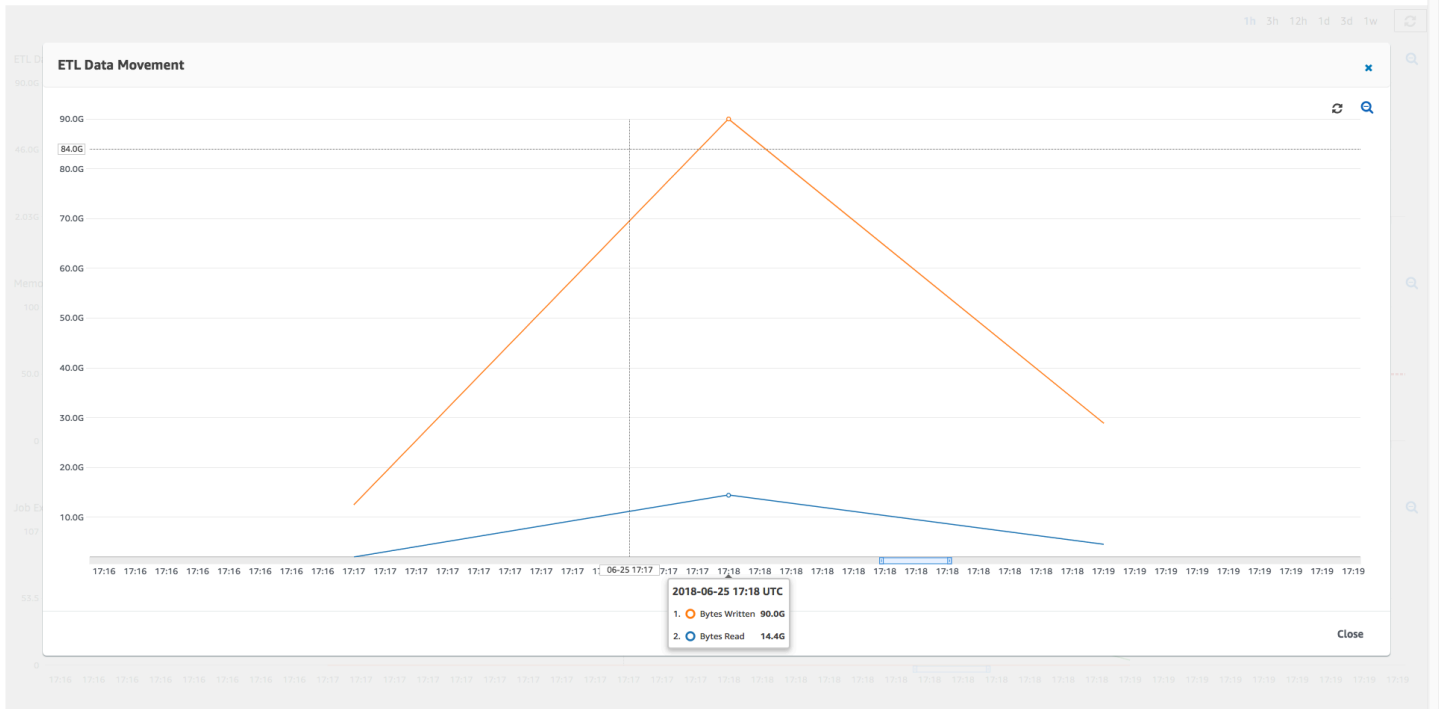
Detailed job metrics



위 이미지에서 볼 수 있듯이 활성 실행기의 총 수는 할당된 최대값인 107개에 도달합니다. 마찬가지로, 최대 필요 실행기는 최대 할당 실행기 수보다 절대 높지 않습니다. 최대 필요 실행기 수는 능동적으로 실행 중인 작업 수와 보류 중인 작업 수에서 계산되므로 활성 실행기 수보다 작을 수도 있습니다. 이는 짧은 시간 동안 부분적으로 또는 전체적으로 유휴 상태이거나 아직 폐기되지 않은 실행기가 있을 수 있기 때문입니다.

Jobs > e2e-dpus

Detailed job metrics



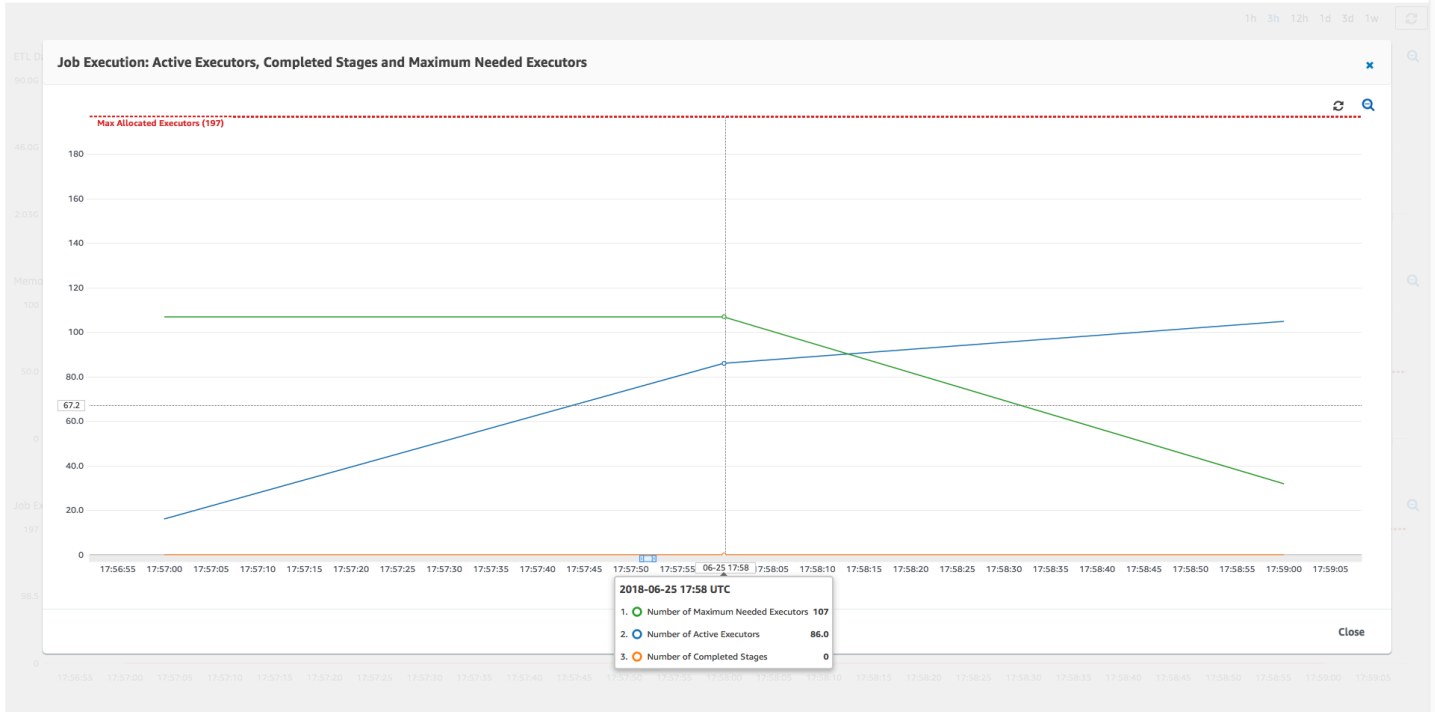
이 작업 실행은 6x 이상의 실행기를 사용하여 Amazon S3에서 데이터를 동시에 읽고 씁니다. 결과적으로 이 작업은 읽기 및 쓰기에 모두 더 많은 Amazon S3 대역폭을 사용하여 더 빨리 완료합니다.

과다 프로비저닝된 DPU 식별

다음에는, DPU가 100개(99 * 2 = 198개 실행기)인 작업을 확장하는 것이 성능을 향상하는 데 도움이 되는지 여부를 결정할 수 있습니다. 다음 그래프와 같이, 작업을 완료하는 데 여전히 3분이 걸립니다. 마찬가지로, 이 작업은 실행기를 107 개 이상(55개 DPU 구성)으로 확장하지 않으며, 남은 91개 실행기가 과다 프로비저닝되고 전혀 사용되지 않습니다. 이는 최대 필요 실행기 수에서 분명히 알 수 있듯이 DPU 수를 늘리는 것이 항상 성능을 향상시키지는 않을 수도 있음을 보여줍니다.

Jobs > e2e-dpus

Detailed job metrics



시간차 비교

다음 표에 나와 있는 세 가지 작업 실행은 10개 DPU, 55개 DPU 및 100개 DPU에 대한 작업 실행 시간을 요약한 것입니다. 첫 번째 작업 실행을 모니터링함으로써 설정한 추정치를 사용하여 작업 실행 시간을 개선할 DPU 용량을 찾아낼 수 있습니다.

작업 ID	DPU 수	실행 시간
jr_c894524c8ef5048a4d9...	10	6분
jr_1a466cf2575e7ffe6856...	55	3분
jr_34fa1ed4c6aa9ff0a814...	100	3분

AWS Glue에서 스트리밍 ETL 작업

지속적으로 실행되고 Amazon Kinesis Data Streams, Apache Kafka 및 Amazon Managed Streaming for Apache Kafka (Amazon MSK)와 같은 스트리밍 소스의 데이터를 사용하는 스트리밍 추출, 변환, 로드 작업을 생성할 수 있습니다. 작업은 데이터를 정리 및 변환한 다음 결과를 Amazon S3 데이터 레이크 또는 JDBC 데이터 스토어에 로드합니다.

또한 Amazon Kinesis Data Streams 스트림용 데이터를 생성할 수 있습니다. 이 기능은 AWS Glue 스크립트를 작성할 때만 사용할 수 있습니다. 자세한 정보는 [the section called “Kinesis 연결”](#)을 참조하세요.

기본적으로 AWS Glue는 100초 기간에 데이터를 처리하고 작성합니다. 이를 통해 데이터를 효율적으로 처리할 수 있으며 예상보다 늦게 도착하는 데이터에 대해 집계를 수행할 수 있습니다. 이 기간 크기를 수정하여 적시성 또는 집계 정확도를 높일 수 있습니다. AWS Glue 스트리밍 작업은 작업 북마크가 아닌 체크포인트를 사용하여 읽은 데이터를 추적합니다.

Note

AWS Glue는 ETL 스트리밍 작업이 실행되는 동안 시간당 요금을 청구합니다.

이 비디오에서는 스트리밍 ETL 비용 문제와 비용 절감 기능에 대해 설명합니다. AWS Glue 스트리밍 ETL 작업을 생성하려면 다음 단계를 수행합니다.

1. Apache Kafka 스트리밍 소스의 경우 Kafka 소스 또는 Amazon MSK 클러스터에 대한 AWS Glue 연결을 생성합니다.
2. 스트리밍 소스에 대한 Data Catalog 테이블을 수동으로 생성합니다.
3. 스트리밍 데이터 원본에 대한 ETL 작업을 생성합니다. 스트리밍 관련 작업 속성을 정의하고 자체 스크립트를 제공하거나 필요에 따라 생성된 스크립트를 수정합니다.

자세한 정보는 [AWS Glue의 스트리밍 ETL](#)을 참조하세요.

Amazon Kinesis Data Streams에 대한 스트리밍 ETL 작업을 생성할 때 AWS Glue 연결을 생성할 필요가 없습니다. 그러나 Kinesis Data Streams를 소스로 포함하는 AWS Glue 스트리밍 ETL 작업에 첨부된 연결이 있는 경우 Kinesis에 대한 Virtual Private Cloud(VPC) 엔드포인트가 필요합니다. 자세한 내용은 VPC 사용 설명서의 [인터페이스 엔드포인트 생성](#)을 참조하세요. 다른 계정에서 Amazon Kinesis Data Streams 스트림을 지정할 때 교차 계정 액세스를 허용하도록 역할과 정책을 설정해야 합니다. 자세한 내용은 [예: 다른 계정의 Kinesis 스트림에서 읽기](#)를 참조하세요.

AWS Glue 스트리밍 ETL 작업은 압축된 데이터를 자동으로 감지하고, 스트리밍 데이터를 투명하게 압축 해제할 수 있으며, 입력 소스에 대해 일반적인 변환을 수행하고, 출력 저장소에 로드할 수 있습니다.

AWS Glue에서는 입력 형식에 따라 다음과 같은 압축 유형에 대해 자동 압축 해제를 지원합니다.

압축 유형	Avro 파일	Avro 데이터	JSON	CSV	Grok
bzip2	예	예	예	예	예
GZIP	아니요	예	예	예	예
Snappy	예(원시 Snappy)	예(프레임 처리된 Snappy)	예(프레임 처리된 Snappy)	예(프레임 처리된 Snappy)	예(프레임 처리된 Snappy)
XZ	예	예	예	예	예
ZSTD	예	아니요	아니요	아니요	아니요
DEFLATE	예	예	예	예	예

주제

- [Apache Kafka 데이터 스트림에 대한 AWS Glue 연결 생성](#)
- [스트리밍 소스에 대한 Data Catalog 테이블 생성](#)
- [Avro 스트리밍 소스에 대한 참고 사항 및 제한 사항](#)
- [스트리밍 소스에 Grok 패턴 적용](#)
- [스트리밍 ETL 작업에 대한 작업 속성 정의](#)
- [스트리밍 ETL 참고 사항 및 제한 사항](#)

Apache Kafka 데이터 스트림에 대한 AWS Glue 연결 생성

Apache Kafka 스트림에서 읽으려면 AWS Glue 연결을 생성해야 합니다.

Kafka 소스에 대한 AWS Glue 연결을 생성하려면(콘솔)

1. <https://console.aws.amazon.com/glue/> 에서 AWS Glue 콘솔을 여십시오.
2. 탐색 창의 데이터 카탈로그에서 연결을 선택합니다.
3. 연결 추가를 선택하고 연결 속성 설정 페이지에서 연결 이름을 입력합니다.

Note

연결 속성 지정에 대한 자세한 내용은 [AWS Glue 연결 속성](#)을 참조하세요.

4. 연결 유형에서 Kafka를 선택합니다.
5. [Kafka 부트스트랩 서버 URL(Kafka bootstrap servers URLs)]에 Amazon MSK 클러스터 또는 Apache Kafka 클러스터용 부트스트랩 브로커의 호스트 및 포트 번호를 입력합니다. Kafka 클러스터에 대한 초기 연결을 설정하려면 전송 계층 보안(TLS) 엔드포인트만 사용합니다. Plaintext 엔드포인트는 지원되지 않습니다.

다음은 Amazon MSK 클러스터의 호스트 이름 및 포트 번호 페어 목록의 예입니다.

```
myserver1.kafka.us-east-1.amazonaws.com:9094,myserver2.kafka.us-
east-1.amazonaws.com:9094,
myserver3.kafka.us-east-1.amazonaws.com:9094
```

부트스트랩 브로커 정보 가져오기에 대한 자세한 내용은 Amazon Managed Streaming for Apache Kafka Developer Guide의 [Getting the Bootstrap Brokers for an Amazon MSK Cluster](#)를 참조하세요.

6. Kafka 데이터 원본에 대한 보안 연결을 원하는 경우 [SSL 연결 필요(Require SSL connection)]를 선택하고 [Kafka 프라이빗 CA 인증서 위치(Kafka private CA certificate location)]에 사용자 정의 SSL 인증서에 대한 유효한 Amazon S3 경로를 입력합니다.

자체 관리형 Kafka에 대한 SSL 연결의 경우 사용자 정의 인증서가 필수입니다. Amazon MSK의 경우 선택 사항입니다.

Kafka에 대한 사용자 정의 인증서 지정에 대한 자세한 내용은 [the section called “SSL 연결 속성”](#) 섹션을 참조하세요.

7. AWS Glue Studio 또는 AWS CLI를 사용하여 Kafka 클라이언트 인증 방법을 지정합니다. 액세스하려면 AWS Glue Studio 왼쪽 AWS Glue탐색 창의 ETL 메뉴에서 선택하십시오.

Kafka 클라이언트 인증 방법에 대한 자세한 내용은 [클라이언트 인증을 위한 AWS Glue Kafka 연결 속성](#) 단원을 참조하세요.

8. 필요에 따라 설명을 입력하고 [다음(Next)]을 선택합니다.
9. Amazon MSK 클러스터의 경우 Virtual Private Cloud (VPC), 서브넷 및 보안 그룹을 지정합니다. 자체 관리형 Kafka의 경우 VPC 정보는 선택 사항입니다.
10. [다음(Next)]을 선택하여 모든 연결 속성을 검토하고 [마침(Finish)]을 선택합니다.

AWS Glue 연결에 대한 자세한 정보는 [데이터에 연결](#) 섹션을 참조하세요.

클라이언트 인증을 위한 AWS Glue Kafka 연결 속성

SASL/GSSAPI(Kerberos) 인증

이 인증 방법을 선택하면 Kerberos 속성을 지정할 수 있습니다.

Kerberos Keytab

keytab 파일의 위치를 선택합니다. keytab은 하나 이상의 보안 주체에 대한 장기 키를 저장합니다. 자세한 내용은 [MIT Kerberos Documentation: Keytab](#)(MIT Kerberos 설명서: Keytab)을 참조하세요.

Kerberos krb5.conf 파일(Kerberos krb5.conf file)

krb5.conf 파일을 선택합니다. 여기에는 KDC 서버의 기본 영역(동일한 KDC 아래에 있는 시스템 그룹을 정의하는 도메인과 유사한 논리 네트워크)과 위치가 포함됩니다. 자세한 내용은 [MIT Kerberos Documentation: krb5.conf](#)(MIT Kerberos 설명서: krb5.conf)를 참조하세요.

Kerberos 보안 주체 및 Kerberos 서비스 이름(Kerberos principal and Kerberos service name)

Kerberos 보안 주체 및 서비스 이름을 입력합니다. 자세한 내용은 [MIT Kerberos 설명서: Kerberos 보안 주체](#)(MIT Kerberos Documentation: Kerberos principal)을 참조하세요.

SASL/SCRAM-SHA-512 인증

이 인증 방법을 선택하면 인증 자격 증명을 지정할 수 있습니다.

AWS 보안 관리자

이름 또는 ARN을 입력하여 검색 상자에서 토큰을 검색합니다.

사용자 이름 및 암호 직접 제공(Provide username and password directly)

이름 또는 ARN을 입력하여 검색 상자에서 토큰을 검색합니다.

SSL 클라이언트 인증

이 인증 방법을 선택하면 Amazon S3를 검색하여 Kafka 클라이언트 키 스토어의 위치를 선택할 수 있습니다. 선택 사항으로 Kafka 클라이언트 키 스토어 암호와 Kafka 클라이언트 키 암호를 입력할 수 있습니다.

IAM 인증.

이 인증 방법은 추가 사양이 필요하지 않으며 스트리밍 소스가 MSK Kafka인 경우에만 적용됩니다.

SASL/PLAIN 인증

이 인증 방법을 선택하면 인증 자격 증명을 지정할 수 있습니다.

스트리밍 소스에 대한 Data Catalog 테이블 생성

데이터 스키마를 포함한 소스 데이터 스트림 속성을 지정하는 데이터 카탈로그 테이블을 스트리밍 소스에 대해 수동으로 생성할 수 있습니다. 이 테이블은 스트리밍 ETL 작업에 대한 데이터 원본으로 사용 됩니다.

원본 데이터 스트림에 있는 데이터의 스키마를 모르는 경우 스키마 없이 테이블을 생성할 수 있습니다. 그런 다음 스트리밍 ETL 작업을 생성할 때 AWS Glue 스키마 감지 기능을 설정할 수 있습니다. AWS Glue는 스트리밍 데이터에서 스키마를 결정합니다.

[AWS Glue 콘솔](#), AWS Command Line Interface (AWS CLI) 또는 AWS Glue API를 사용하여 테이블을 생성합니다. AWS Glue 콘솔을 사용하여 수동으로 테이블을 생성하는 방법에 대한 자세한 내용은 [the section called “테이블 생성”](#) 단원을 참조하십시오.

Note

AWS Lake Formation 콘솔을 사용하여 테이블을 생성할 수 없으며 AWS Glue 콘솔을 사용해야 합니다.

또한 Avro 포맷의 스트리밍 소스 또는 Grok 패턴을 적용할 수 있는 로그 데이터에 대한 다음 정보를 고려합니다.

- [the section called “Avro 스트리밍 소스에 대한 참고 사항 및 제한 사항”](#)
- [the section called “스트리밍 소스에 Grok 패턴 적용”](#)

주제

- [Kinesis 데이터 원본](#)
- [Kafka 데이터 원본](#)
- [AWS Glue Schema Registry 테이블 소스](#)

Kinesis 데이터 원본

테이블 생성 시 다음 스트리밍 ETL 속성(콘솔)을 설정합니다.

소스 유형

Kinesis

동일한 계정에 있는 Kinesis 소스의 경우

지역

Amazon Kinesis 데이터 스트림 서비스가 위치한 AWS 지역. 리전 및 Kinesis 스트림 이름은 함께 스트림 ARN으로 변환됩니다.

예: <https://kinesis.us-east-1.amazonaws.com>

Kinesis 스트림 이름

Amazon Kinesis Data Streams Developer Guide의 [Creating a Stream](#)에 설명된 스트림 이름입니다.

다른 계정의 Kinesis 소스의 경우 [이 예](#)를 참조하여 교차 계정 액세스를 허용하도록 역할과 정책을 설정합니다. 다음 설정을 구성합니다.

스트림 ARN

소비자가 등록된 Kinesis 데이터 스트림의 ARN입니다. 자세한 내용은 [의 Amazon 리소스 이름 \(ARN\) 및 AWS 서비스 네임스페이스](#)를 참조하십시오. AWS 일반 참조

수입된 역할 ARN

수입할 역할의 Amazon 리소스 이름(ARN)입니다.

세션 이름(선택 사항)

수입한 역할 세션의 식별자입니다.

역할 세션 이름을 사용하여 다른 보안 주체가 동일한 역할을 수입할 때 또는 다른 이유로 세션을 고유하게 식별합니다. 교차 계정 시나리오에서는 역할을 소유하는 계정이 역할 세션 이름을 보고 로그할 수 있습니다. 역할 세션 이름은 수입한 역할 보안 주체의 ARN에도 사용됩니다. 즉, 임시 보안 자격 증명을 사용하는 후속 교차 계정 API 요청은 외부 계정의 로그에 역할 세션 이름을 노출합니다. AWS CloudTrail

Amazon Kinesis Data Streams(AWS Glue API 또는 AWS CLI)에 대한 스트리밍 ETL 속성을 설정하려면

- 동일한 계정에서 Kinesis 소스에 대한 스트리밍 ETL 속성을 설정하려면 CreateTable API 작업 또는 create_table CLI 명령의 StorageDescriptor 구조에서 streamName 및 endpointUrl 파라미터를 지정합니다.

```
"StorageDescriptor": {
  "Parameters": {
```

```

"typeOfData": "kinesis",
"streamName": "sample-stream",
"endpointUrl": "https://kinesis.us-east-1.amazonaws.com"
}
...
}

```

또는 streamARN을 지정합니다.

Example

```

"StorageDescriptor": {
  "Parameters": {
    "typeOfData": "kinesis",
    "streamARN": "arn:aws:kinesis:us-east-1:123456789:stream/sample-stream"
  }
  ...
}

```

- 다른 계정에서 Kinesis 소스에 대한 스트리밍 ETL 속성을 설정하려면 CreateTable API 작업 또는 create_table CLI 명령의 StorageDescriptor 구조에서 streamARN, awsSTSRoleARN 및 awsSTSSessionName(선택 사항) 파라미터를 지정합니다.

```

"StorageDescriptor": {
  "Parameters": {
    "typeOfData": "kinesis",
    "streamARN": "arn:aws:kinesis:us-east-1:123456789:stream/sample-stream",
    "awsSTSRoleARN": "arn:aws:iam::123456789:role/sample-assume-role-arn",
    "awsSTSSessionName": "optional-session"
  }
  ...
}

```

Kafka 데이터 원본

테이블 생성 시 다음 스트리밍 ETL 속성(콘솔)을 설정합니다.

소스 유형

Kafka

Kafka 소스의 경우:

주제 이름

Kafka에 지정된 주제 이름입니다.

연결

[the section called “Kafka 데이터 스트림에 대한 연결 생성”](#)에 설명된 대로 Kafka 소스를 참조하는 AWS Glue 연결입니다.

AWS Glue Schema Registry 테이블 소스

AWS Glue Schema Registry를 스트리밍 작업에 사용하려면 [사용 사례: AWS Glue Data Catalog](#)에 있는 지침에 따라 Schema Registry 테이블을 생성하거나 업데이트합니다.

현재 AWS Glue 스트리밍은 스키마 추론이 false로 설정된 Glue Schema Registry Avro 포맷만 지원합니다.

Avro 스트리밍 소스에 대한 참고 사항 및 제한 사항

Avro 포맷의 스트리밍 소스에는 다음 참고 및 제한 사항이 적용됩니다.

- 스키마 감지가 설정되어 있으면 Avro 스키마가 페이로드에 포함되어야 합니다. 해제되어 있으면 데이터만 페이로드에 포함되어야 합니다.
- 일부 Avro 데이터 유형은 동적 프레임에서 지원되지 않습니다. AWS Glue 콘솔의 테이블 생성 마법사에서 [스키마 정의(Define a schema)] 페이지를 사용하여 스키마를 정의할 때 이러한 데이터 유형을 지정할 수 없습니다. 스키마 감지 중에 Avro 스키마에서 지원되지 않는 유형은 다음과 같이 지원되는 유형으로 변환됩니다.
 - EnumType => StringType
 - FixedType => BinaryType
 - UnionType => StructType
- 콘솔의 [스키마 정의(Define a schema)] 페이지를 사용하여 테이블 스키마를 정의하는 경우 스키마에 대한 암시적 루트 요소 유형은 record입니다. record 이외의 루트 요소 유형(예: array 또는 map)을 원하는 경우 [스키마 정의(Define a schema)] 페이지를 사용하여 스키마를 지정할 수 없습니다. 대신 해당 페이지를 건너뛰고 테이블 속성으로 또는 ETL 스크립트 내에서 스키마를 지정해야 합니다.

- 테이블 속성에서 스키마를 지정하려면 테이블 생성 마법사를 완료하고 테이블 세부 정보를 편집한 다음 [테이블 속성(Table properties)]에서 새 키-값 페어를 추가합니다. 다음 스크린샷과 같이 키 avroSchema를 사용하고 값에 대한 스키마 JSON 객체를 입력합니다.

Edit table details

Key

Value

Description

Type description...

Table properties

Key	Value	
classification	avro	✕
avroSchema	{"type": "array", "items": "strin	✕

- ETL 스크립트에서 스키마를 지정하려면 다음 Python 및 Scala 예제와 같이 `datasource0` 할당문을 수정하고 `avroSchema` 키를 `additional_options` 인수에 추가합니다.

Python

```
SCHEMA_STRING = '{"type": "array", "items": "string"}'
datasource0 = glueContext.create_data_frame.from_catalog(database =
    "database", table_name = "table_name", transformation_ctx = "datasource0",
    additional_options = {"startingPosition": "TRIM_HORIZON", "inferSchema":
    "false", "avroSchema": SCHEMA_STRING})
```


Scala

```
val SCHEMA_STRING = """"{"type":"array","items":"string"}""""
val datasource0 = glueContext.getCatalogSource(database = "database", tableName
= "table_name", redshiftTmpDir = "", transformationContext = "datasource0",
additionalOptions = JsonOptions(s""""{"startingPosition": "TRIM_HORIZON",
"inferSchema": "false", "avroSchema": "$SCHEMA_STRING"}"""")).getDataFrame()
```

스트리밍 소스에 Grok 패턴 적용

로그 데이터 원본에 대한 스트리밍 ETL 작업을 생성하고 Grok 패턴을 사용하여 로그를 정형 데이터로 변환할 수 있습니다. 그러면 ETL 작업은 데이터를 정형 데이터 원본으로 처리합니다. 스트리밍 원본에 대한 Data Catalog 테이블을 생성할 때 적용할 Grok 패턴을 지정합니다.

Grok 패턴 및 사용자 정의 패턴 문자열 값에 대한 자세한 내용은 [Grok 사용자 지정 분류자 작성](#) 섹션을 참조하세요.

Data Catalog 테이블에 Grok 패턴 추가(콘솔)

- 테이블 생성 마법사를 사용하여 [the section called “스트리밍 소스에 대한 Data Catalog 테이블 생성”](#)에 지정된 파라미터로 테이블을 생성합니다. 데이터 포맷을 Grok로 지정하고 [Grok 패턴(Grok pattern)] 필드를 채우고 필요에 따라 [사용자 정의 패턴(선택 사항)(Custom patterns (optional))] 아래 사용자 정의 패턴을 추가합니다.

Choose a data format

Classification

CSV
 JSON
 ORC
 Parquet
 Avro
 Grok

Choose the format of the data in your table.

Grok pattern

Built-in and custom named patterns used to parse your data into a structured schema. For more information, see the [list of built-in patterns](#).

Custom patterns

1

Optional custom building blocks for the grok pattern.

각 사용자 정의 패턴 다음에 Enter 키를 누릅니다.

Data Catalog 테이블에 Grok 패턴 추가(AWS Glue API 또는 AWS CLI)

- CreateTable API 작업 또는 create_table CLI 명령에 GrokPattern 파라미터와 CustomPatterns 파라미터(선택 사항)를 추가합니다.

```
"Parameters": {
  ...
  "grokPattern": "string",
  "grokCustomPatterns": "string",
  ...
},
```

grokCustomPatterns를 문자열로 표현하고 패턴 사이의 구분 기호로 "\n"을 사용합니다.

다음은 이러한 파라미터를 지정하는 예입니다.

Example

```
"parameters": {
  ...
  "grokPattern": "%{USERNAME:username} %{DIGIT:digit:int}",
  "grokCustomPatterns": "digit \\d",
  ...
}
```

스트리밍 ETL 작업에 대한 작업 속성 정의

AWS Glue 콘솔에서 스트리밍 ETL 작업을 정의할 때 다음 스트림 관련 속성을 제공합니다. 추가 작업 속성에 대한 설명은 [Spark 작업에 대한 작업 속성 정의](#) 단원을 참조하십시오.

IAM 역할

작업을 실행하고, 스트리밍 소스에 액세스하고, 대상 데이터 스토어에 액세스하는 데 사용되는 리소스에 대한 권한 부여에 사용되는 AWS Identity and Access Management (IAM) 역할을 지정합니다.

Amazon Kinesis Data Streams에 액세스하려면 관리형 정책을 역할에 연결하거나 보다 세분화된 액세스를 허용하는 유사한 IAM 정책을 추가하십시오. AmazonKinesisFullAccess AWS 샘플 정책은 [IAM을 사용하여 Amazon Kinesis Data Streams 리소스에 대한 액세스 제어](#)를 참조하십시오.

AWS Glue 작업을 실행하는 권한에 대한 자세한 내용은 [AWS Glue의 ID 및 액세스 관리](#) 단원을 참조하십시오.

유형

Spark Streaming을 선택합니다.

AWS Glue 버전

AWS Glue 버전에 따라 작업에 사용할 수 있는 Apache Spark와 Python 또는 Scala의 버전이 정해집니다. 작업에 사용 가능한 Python 또는 Scala 버전을 지정하는 선택 항목을 선택합니다. AWS Glue Python 3을 지원하는 버전 2.0은 스트리밍 ETL 작업의 기본값입니다.

유지보수 윈도우

스트리밍 작업을 다시 시작할 수 있는 창을 지정합니다. [the section called “유지 관리 기간”](#)를 참조하세요.

작업 제한 시간

필요에 따라 기간(분)을 입력합니다. 기본값은 공백입니다.

- 스트리밍 작업의 제한 시간 값은 7일 또는 10080분 미만이어야 합니다.
- 값을 비워 두면 유지 관리 기간을 설정하지 않은 경우 작업이 7일 후에 다시 시작됩니다. 유지 관리 기간을 설정한 경우 7일 후 유지 관리 기간 중에 작업이 다시 시작됩니다.

데이터 소스

[the section called “스트리밍 소스에 대한 Data Catalog 테이블 생성”](#)에 생성된 테이블을 지정합니다.

데이터 대상

다음 중 하나를 수행하십시오.

- 데이터 대상의 테이블 생성을 선택하고 다음 데이터 대상 속성을 지정합니다.

데이터 스토어

Amazon S3 또는 JDBC를 선택합니다.

형식

원하는 포맷을 선택합니다. 스트리밍에 모두 지원됩니다.

- [데이터 카탈로그의 테이블 사용 및 데이터 대상 업데이트(Use tables in the data catalog and update your data target)]를 선택하고 JDBC 데이터 스토어에 대한 테이블을 선택합니다.

출력 스키마 정의

다음 중 하나를 수행하십시오.

- [각 레코드의 스키마 자동 감지(Automatically detect schema of each record)]를 선택하여 스키마 감지를 설정합니다. AWS Glue는 스트리밍 데이터에서 스키마를 결정합니다.
- [모든 레코드에 대한 출력 스키마 지정(Specify output schema for all records)]을 선택하여 매핑 적용 변환을 사용하여 출력 스키마를 정의합니다.

Script

필요에 따라 자체 스크립트를 제공하거나 생성된 스크립트를 수정하여 Apache Spark Structured Streaming 엔진이 지원하는 작업을 수행합니다. 사용 가능한 작업에 대한 자세한 내용은 [스트리밍 DataFrames /데이터세트 작업을 참조하십시오](#).

스트리밍 ETL 참고 사항 및 제한 사항

다음 참고 사항 및 제한 사항에 유의하십시오.

- AWS Glue 스트리밍 ETL 작업에 대한 자동 압축 해제는 지원되는 압축 유형에 대해서만 사용할 수 있습니다. 또한 다음을 참조하세요.
 - 프레임 처리된 Snappy는 Snappy용 공식 [프레이밍 형식](#)을 나타냅니다.
 - Deflate는 Glue 버전 2.0이 아닌 Glue 버전 3.0에서 지원됩니다.
- 스키마 감지를 사용하는 경우 스트리밍 데이터의 조인을 수행할 수 없습니다.
- AWS Glue 스트리밍 ETL 작업은 Avro 형식의 AWS Glue 스키마 레지스트리에 대한 Union 데이터 유형을 지원하지 않습니다.
- ETL 스크립트는 AWS Glue의 기본 제공 변환과 Apache Spark Structured Streaming에 대한 기본 변환을 사용할 수 있습니다. 자세한 내용은 Apache Spark [웹 사이트의 스트리밍 DataFrames /데이터 세트 작업](#) 또는 을 참조하십시오. [AWS Glue PySpark 변환 참조](#)
- AWS Glue 스트리밍 ETL 작업은 체크포인트를 사용하여 읽은 데이터를 추적합니다. 따라서 중지되고 다시 시작된 작업은 스트림에서 중단된 부분부터 다시 시작됩니다. 데이터를 다시 처리하려는 경우 스크립트에서 참조된 체크포인트 폴더를 삭제할 수 있습니다.
- 작업 북마크는 지원되지 않습니다.
- Kinesis Data Streams의 향상된 팬아웃 기능을 사용하려면 [the section called “Kinesis 스트리밍 작업에서 향상된 팬아웃 사용”](#) 섹션을 참조하세요.
- AWS Glue Schema Registry에서 생성된 Data Catalog 테이블을 사용하는 경우 새 스키마 버전을 사용할 수 있게 되면 새 스키마를 반영하기 위해 다음을 수행해야 합니다.
 1. 테이블과 연결된 작업을 중지합니다.
 2. Data Catalog 테이블의 스키마를 업데이트합니다.
 3. 테이블과 연결된 작업을 다시 시작합니다.

AWS Lake Formation FindMatches로 레코드 매칭

Note

현재 레코드 일치는 중동(UAE), 유럽(스페인)(eu-south-2) 및 유럽(취리히)(eu-central-2) 리전의 AWS Glue 콘솔에서 사용할 수 없습니다.

AWS Lake Formation에는 기계 학습 기능이 있어서 사용자 지정 변환을 만들어 데이터를 정리할 수 있습니다. 여기에는 FindMatches로 명명된 사용 가능 변환이 하나 있습니다. FindMatches 변환으로는 레코드에 공통된 고유 식별자가 없고 정확히 일치되는 필드 또한 없을 경우에도 데이터 세트에서 중복 레코드나 일치 레코드를 식별할 수 있습니다. 여기에서는 특정 코드 작성이나 기계 학습 진행 방법 속지가 불필요합니다. FindMatches는 다음과 같은 다양한 문제에서 도움이 될 수 있습니다.

- 고객 매칭(Matching customers): 데이터베이스상에서 여러 고객 필드가 서로 정확히 일치하지 않을 때에도(예: 이름의 철자나 주소가 다른 경우, 데이터가 누락되거나 부정확한 경우 등) 고객의 레코드를 다양한 고객 데이터베이스에 연결할 수 있습니다.
- 제품 매칭(Matching products): 항목의 구조가 서로 다를 때에도 사용자 카탈로그의 제품과 다른 출처를 매칭할 수 있습니다(예: 경쟁사의 카탈로그와 매칭).
- 부정 행위 탐지 개선(Improving fraud detection): 중복된 고객 계정을 식별하고 새로 생성된 계정이 이전의 부정 행위 사용자와 일치하거나 일치할 소지가 있을 때 이를 확인할 수 있습니다.
- 기타 매칭 문제(Other matching problems): 주소나 영화, 파트 목록 등을 매칭할 수 있습니다. 일반적으로 사람이 데이터베이스의 행을 확인해서 일치 여부를 확인할 수 있다면 FindMatches 변환으로 업무에 도움이 될 만한 상황이 매우 많습니다.

작업을 생성할 때 이러한 변환을 만들 수 있습니다. 생성하는 변환은 레이블을 지정한 소스 데이터 세트의 예제 데이터 및 소스 데이터 스토어 스키마를 기반으로 합니다(이 프로세스를 변환 '교육'이라고 함). 레이블을 지정하는 레코드가 소스 데이터 세트에 있어야 합니다. 이러한 프로세스에서는 사용자가 레이블 지정하는 파일을 생성한 다음 변환이 어느 정도 학습이 가능한 사항을 다시 업로드합니다. 변환을 학습시킨 다음 Spark 기반 AWS Glue 작업(PySpark 또는 Scala Spark)에서 호출하고 원본 데이터 스토어와 함께 다른 스크립트에 사용할 수 있습니다.

생성된 변환은 AWS Glue에 저장됩니다. AWS Glue 콘솔에서 생성된 변환을 관리할 수 있습니다. 데이터 통합 및 ETL, 데이터 분류 도구 > 레코드 일치 아래의 탐색 창에서 기계 학습 변환을 편집하고 계속 가르칠 수 있습니다. 콘솔에서 변환을 관리하는 방법에 대한 자세한 내용은 [AWS Glue 콘솔에서 기계 학습 변환 작업](#) 단원을 참조하십시오.

Note

AWS Glue 버전 2.0 FindMatches 작업은 변환이 데이터를 처리하는 동안 Amazon S3 버킷 `aws-glue-temp-<accountID>-<region>`을 사용하여 임시 파일을 저장합니다. 실행이 완료된 후 수동으로 또는 Amazon S3 수명 주기 규칙을 설정하여 이 데이터를 삭제할 수 있습니다.

기계 학습 변환의 유형

기계 학습 변환을 만들어 데이터를 정리할 수 있습니다. 이런 변환을 ETL 스크립트에서 호출할 수 있습니다. 데이터가 DynamicFrame이라는 데이터 구조 내의 변환에서 변환으로 전달됩니다. 이는 Apache Spark SQL DataFrame을 확장한 것입니다. DynamicFrame은 데이터를 포함하고 데이터 스키마를 참조하여 데이터를 진행합니다.

다음과 같은 유형의 기계 학습 변환을 사용할 수 있습니다.

일치 항목 찾기

원본 데이터에서 중복 레코드를 찾습니다. 예제 데이터 세트에 일치하는 행을 나타내는 레이블을 지정하여 이 기계 학습 변환을 학습시킵니다. 레이블 지정된 예제 데이터로 학습을 많이 시킬수록 기계 학습 변환이 어떤 행을 매칭해야 하는지 배우게 됩니다. 변환의 구성 방식에 따라 다음 중 하나가 출력됩니다.

- 일치하는 레코드 세트를 나타내는 값이 채워진 match_id 열이 추가된 입력 테이블 사본 match_id 열은 임의 식별자입니다. match_id가 동일한 레코드는 서로 일치한다고 식별되었습니다. match_id가 서로 다른 레코드는 일치하지 않습니다.
- 중복 행이 제거된 입력 테이블 사본 중복 항목이 여러 개 발견되면 기본 키가 가장 낮은 레코드를 유지합니다.

중분 일치 항목 찾기

기존 프레임과 중분 프레임에서 일치 항목을 찾고 일치 그룹당 고유 ID를 포함하는 열을 출력으로 반환하도록 일치 항목 찾기 변환을 구성할 수도 있습니다.

자세한 내용은 [중분 일치 항목 찾기](#) 섹션을 참조하세요.

FindMatches 변환 사용

이 FindMatches 변환을 사용하여 원본 데이터에서 중복 레코드를 찾을 수 있습니다. 변환의 학습을 돕기 위한 레이블 지정 파일이 생성 또는 제공됩니다.

Note

현재 사용자 지정 암호화 키를 사용하는 FindMatches 변환은 다음 리전에서 지원되지 않습니다.

- 아시아 태평양(오사카) - ap-northeast-3

FindMatches 변환을 시작하려면 아래 단계를 수행합니다. 자세한 고급 예제는 AWS 빅 데이터 블로그의 [Harmonize data using AWS Glue and AWS Lake Formation FindMatches ML to build a customer 360 view](#)를 참조하세요.

일치 항목 찾기 변환을 사용하여 시작하기

FindMatches 변환을 시작하려면 다음 단계를 따르십시오.

1. AWS Glue Data Catalog에서 정리할 원본 데이터의 테이블을 만듭니다. 크롤러 생성 방법에 대한 자세한 내용은 [AWS Glue 콘솔에서 크롤러 작업](#)을 참조하세요.

원본 데이터가 CSV(쉼표로 구분된 값) 파일 등 텍스트 기반의 파일이라면 다음을 고려하십시오.

- 입력 레코드 CSV 파일과 레이블 지정 파일을 별도의 파일에 보관합니다. 그렇지 않으면 AWS Glue 크롤러가 이를 동일한 테이블의 각 부분으로 간주하고 Data Catalog에 테이블을 잘못 생성할 수 있습니다.
- CSV 파일에 ASCII 문자만 들어 있는 경우를 제외하고, CSV 파일에는 BOM(바이트 순서 표시) 없는 UTF-8 인코딩을 사용해야 합니다. Microsoft Excel은 흔히 UTF-8 CSV 파일의 시작 부분에 BOM을 추가합니다. 이를 제거하려면 텍스트 편집기에서 CSV 파일을 열고 파일을 UTF-8 without BOM(BOM 없는 UTF-8)으로 다시 저장하십시오.

2. AWS Glue 콘솔에서 작업을 만들고 Find matches(일치 항목 찾기) 변환 유형을 선택합니다.

Important

작업을 위해 선택하는 데이터 원본 테이블의 열 수는 100개를 넘을 수 없습니다.

3. [레이블 지정 파일 생성(Generate labeling file)]을 선택하여 AWS Glue에 레이블 지정 파일을 생성하도록 지시합니다. AWS Glue가 각 labeling_set_id에 비슷한 레코드를 그룹화하기 위한 첫 번째 전달을 맡아 그러한 그룹화를 검토할 수 있게 해줍니다. 사용자의 레이블이 label 열과 일치합니다.
 - 일치하는 행을 나타내는 레코드 예시인 레이블 지정 파일이 이미 있는 경우, 해당 파일을 Amazon Simple Storage Service(Amazon S3)에 업로드합니다. 레이블 지정 파일의 형식에 대한 자세한 내용은 [레이블 지정 파일 형식](#) 단원을 참조하십시오. 4단계로 이동합니다.
4. 레이블 지정 파일을 다운로드하고 [레이블링](#) 단원에 설명된 대로 해당 파일에 레이블을 지정합니다.
5. 수정된 레이블 지정 파일을 업로드합니다. AWS Glue가 일치 항목 찾는 방법을 변환에 학습시키기 위한 작업을 실행합니다.

Machine learning transforms(기계 학습 변환) 목록 페이지에서 기록 탭을 선택합니다. AWS Glue가 다음 작업을 수행하면 이 페이지에 표시됩니다.

- 레이블 가져오기
 - 레이블 내보내기
 - 레이블 생성
 - 품질 예측
6. 더 나은 변환을 만들기 위해 레이블 지정 파일을 반복해서 다운로드하고, 레이블 지정하고, 업로드할 수 있습니다. 처음 실행할 때는 불일치 레코드가 훨씬 더 많을 수 있습니다. 그러나 레이블 지정 파일을 확인하면서 계속 학습시키면 AWS Glue가 학습을 합니다.
 7. 일치 항목 찾기의 성능과 결과를 측정하여 변환을 평가하고 조정하십시오. 자세한 내용은 [AWS Glue에서 기계 학습 변환 튜닝](#) 단원을 참조하십시오.

레이블링

FindMatches가 레이블 지정 파일을 생성할 때 사용자의 소스 테이블에서 레코드가 선택됩니다. 이전의 학습을 토대로 FindMatches는 학습해야 할 가장 중요한 레코드를 파악합니다.

레이블 지정이란 레이블 지정 파일을 편집(Microsoft Excel 같은 스프레드시트 사용을 권장)하고 일치 및 불일치 레코드를 식별하는 label 열에 식별자나 레이블을 추가하는 것을 뜻합니다. 소스 데이터의 일치 항목을 명확하고 일관성 있게 정의해야 합니다. FindMatches는 사용자가 일치(또는 불일치) 항목으로 지정한 레코드에서 학습하고, 사용자의 결정을 토대로 중복 레코드 찾는 방법을 배웁니다.

FindMatches에서 레이블 지정 파일이 생성될 때 약 100개의 레코드가 생성됩니다. 이러한 100개의 레코드는 보통 10개의 레이블 지정 세트로 나뉘며, 각각의 레이블 지정 세트는 FindMatches에서 생성된 고유한 labeling_set_id로 식별됩니다. 각 레이블 지정 세트는 다른 레이블 지정 세트와 독립적인 별도의 레이블 지정 작업으로 간주되어야 합니다. 각각의 레이블 지정 세트 내에서 일치하는 레코드와 일치하지 않는 레코드를 식별하는 작업을 합니다.

스프레드시트에서의 레이블 지정 파일 편집 관련 팁입니다.

스프레드 시트 애플리케이션에서 레이블 지정 파일을 편집할 때는 다음을 고려하십시오.

- 열 필드가 전부 확장된 상태에서는 파일이 열리지 않을 수 있습니다. 원하는 셀의 내용을 보기 위해 labeling_set_id 및 label 열을 확장해야 할 수 있습니다.
- long 데이터 유형 등 기본 키 열이 숫자인 경우, 스프레드시트에서 이를 숫자로 해석하고 값을 변경할 수 있습니다. 이 키 값은 텍스트로 취급해야 합니다. 이 문제를 해결하려면 기본 키 열의 모든 셀을 텍스트 데이터로 서식 지정하십시오.

레이블 지정 파일 형식

FindMatches 변환을 가르치기 위해 AWS Glue에서 생성된 레이블 지정 파일은 다음 형식을 사용합니다. AWS Glue에 대한 자체 파일을 생성하는 경우에는 다음 형식도 따라야 합니다.

- CSV(쉼표로 구분된 값) 파일입니다.
- UTF-8로 인코딩해야 합니다. Microsoft Windows에서 이 파일을 편집하면 cp1252로 인코딩될 수 있습니다.
- Amazon S3 위치에 있어야 AWS Glue로 전달할 수 있습니다.
- 각각의 레이블 지정 태스크마다 적당한 수의 행을 사용하세요. 태스크당 10~20행이 좋지만, 태스크당 2~30행도 허용 가능합니다. 행이 50개 이상인 작업은 권장되지 않으며 결과가 좋지 않거나 시스템 오류가 발생할 수 있습니다.
- “일치”또는 “불일치”로 레이블이 지정된 레코드 쌍으로 구성되고 이미 레이블이 지정된 데이터인 경우에는 괜찮습니다. 이렇게 레이블이 지정된 쌍은 크기가 2인 레이블 지정 세트로 표현될 수 있습니다. 이때 일치하는 경우에는 두 레코드를 모두 문자 “A”로 레이블 지정하고 일치하지 않는 경우에는 하나는 “A”로, 다른 하나는 “B”로 레이블 지정합니다.

Note

레이블 지정 파일에는 추가 열이 있기 때문에 원본 데이터가 들어 있는 파일과는 스키마가 다릅니다. AWS Glue 크롤러가 Data Catalog에서 테이블을 생성할 때 레이블 지정 파일을 고려하는 일이 없도록 이 파일을 변환 입력 CSV 파일과 다른 폴더에 저장합니다. 그렇지 않으면 AWS Glue 크롤러가 만든 테이블에 데이터가 올바르게 표현되지 않을 수 있습니다.

- 처음 두 열(labeling_set_id, label)은 AWS Glue의 필수 열입니다. 나머지 열은 처리할 데이터의 스키마와 일치해야 합니다.
- 각 labeling_set_id에 대해 동일한 레이블을 사용하여 일치하는 레코드를 모두 식별합니다. 레이블이란 label 열에 배치된 고유한 문자열입니다. A, B, C 같이 단순한 문자로 된 레이블을 사용하는 것이 좋습니다. 레이블은 대소문자를 구분하며 label 열에 입력됩니다.
- 동일한 labeling_set_id과 동일한 레이블을 포함하는 행은 일치 항목으로 레이블이 지정되는 것으로 간주됩니다.
- 동일한 labeling_set_id과 다른 레이블을 포함하는 행은 일치 항목이 아닌 것으로 레이블이 지정됩니다.
- 다른 labeling_set_id를 포함하는 행은 일치하거나 일치하지 않은 정보를 전달하지 않는 것으로 간주됩니다.

다음은 데이터 레이블 지정의 예입니다.

labeling_set_id	레이블	first_name	last_name	생일
ABC123	A	존	Doe	04/01/1980
ABC123	B	Jane	Smith	04/03/1980
ABC123	A	Johnny	Doe	04/01/1980
ABC123	A	Jon	Doe	04/01/1980
DEF345	A	Richard	Jones	12/11/1992
DEF345	A	Rich	Jones	11/12/1992
DEF345	B	Sarah	Jones	12/11/1992
DEF345	C	Richie	Jones Jr.	05/06/2017
DEF345	B	Sarah	Jones-Walker	12/11/1992
GHI678	A	Robert	Miller	1/3/1999
GHI678	A	Bob	Miller	1/3/1999
XYZABC	A	William	Robinson	2/5/2001
XYZABC	B	Andrew	Robinson	2/5/1971

- 위의 예에서는 John/Johnny/Jon Doe를 일치 항목으로 식별하고 이러한 레코드들이 Jane Smith와 일치하지 않는다고 시스템을 학습시킵니다. 이와 별도로 Richard와 Rich Jones가 같은 사람이지만 이들 레코드가 Sarah Jones/Jones-Walker와 Richie Jones Jr와 일치하지 않는다고 시스템을 학습시킵니다.
- 보시다시피 레이블의 범위는 labeling_set_id로 제한됩니다. 따라서 레이블은 labeling_set_id 경계를 벗어나지 않습니다. 예를 들어, labeling_set_id 1의 레이블 "A"는 labeling_set_id 2의 레이블 "A"와 아무런 관계가 없습니다.

- 레코드가 레이블 지정 세트 내에서 일치 항목이 없는 경우, 고유한 레이블을 지정합니다. 예를 들어, Jane Smith는 레이블 지정 세트 ABC123의 레코드와 일치하지 않으므로 B라는 레이블이 있는 레이블 지정 세트의 유일한 레코드입니다.
- 레이블 지정 세트 "GHI678"은 레이블 지정 세트가 일치한다는 것을 보여주기 위해 동일한 레이블이 지정된 두 개의 레코드로 구성 될 수 있음을 보여줍니다. 마찬가지로 "XYZABC"는 일치하지 않음을 나타내기 위해 서로 다른 레이블이 지정된 두 개의 레코드를 보여줍니다.
- 때로 레이블 지정 세트에 일치하는 항목이 없거나(예: 레이블 지정 세트의 모든 레코드에 다른 레이블을 지정하는 경우), 레이블 지정 세트가 모두 "동일"할 수 있다는(모두 동일한 레이블을 지정한 경우) 점에 유의하십시오. 레이블 지정 세트가 기준에 따라 "동일"하지 않은 레코드의 예들을 총체적으로 포함하는 경우에는 권장합니다.

Important

AWS Glue에 전달하는 IAM 역할이 레이블 지정 파일이 들어 있는 Amazon S3 버킷에 액세스할 수 있는지 확인합니다. 명명 규칙에 의해 AWS Glue 정책이 Amazon S3 버킷이나 폴더 등 이름에 aws-glue- 접두사가 붙은 곳에 권한을 부여합니다. 레이블 지정 파일이 다른 위치에 있는 경우, IAM 역할에 해당 위치에 대한 권한을 추가합니다.

AWS Glue에서 기계 학습 변환 튜닝

AWS Glue에서 기계 학습 변환을 튜닝하여 데이터 정리 작업의 결과를 개선함으로써 목표를 충족할 수 있습니다. 변환을 개선하려면 레이블 지정 세트를 생성하고, 레이블을 추가한 후, 원하는 결과를 얻을 때까지 이러한 단계를 여러 번 반복하여 교육할 수 있습니다. 일부 기계 학습 파라미터를 변경하여 튜닝할 수도 있습니다.

기계 학습 변환에 대한 자세한 내용은 [AWS Lake Formation FindMatches로 레코드 매칭](#) 단원을 참조하십시오.

주제

- [기계 학습 측정](#)
- [정밀도와 재현율 중에서 결정](#)
- [정확도와 비용 중에서 결정](#)
- [일치 신뢰도 점수를 사용하여 일치 항목의 품질 추정](#)
- [일치 항목 찾기 변환 교육](#)

기계 학습 측정

기계 학습 변환을 튜닝하는 데 사용되는 측정을 이해하려면 다음 용어를 숙지해야 합니다.

참 긍정(TP)

변환이 올바르게 찾은 데이터의 일치로, 적중이라고도 합니다.

참 부정(TN)

변환이 올바르게 거부한 데이터의 불일치입니다.

거짓 긍정(FP)

변환이 일치로 잘못 분류한 데이터의 불일치로, 거짓 경보라고도 합니다.

거짓 부정(FN)

변환이 찾지 못한 데이터의 일치로, 누락이라고도 합니다.

기계 학습에 사용되는 용어에 대한 자세한 내용은 Wikipedia에서 [Confusion matrix](#)를 참조하십시오.

기계 학습 변환을 튜닝하려면 변환의 Advanced properties(고급 속성)에서 다음 측정 값을 변경하면 됩니다.

- [정밀도(Precision)]는 변환이 긍정으로 식별하는 총 레코드 수(참 긍정 및 거짓 긍정) 중에서 참 긍정을 얼마나 잘 찾는지 측정합니다. 자세한 내용은 Wikipedia의 [정밀도 및 재현율](#)을 참조하십시오.
- 재현율은 변환이 소스 데이터의 총 레코드에서 참 긍정을 얼마나 잘 찾는지 측정합니다. 자세한 내용은 Wikipedia의 [정밀도 및 재현율](#)을 참조하십시오.
- 정확도는 변환이 참 긍정과 참 부정을 얼마나 잘 찾는지 측정합니다. 정확도를 증가시키려면 더 많은 기계 리소스와 비용이 필요합니다. 하지만 이렇게 하면 재현율도 증가합니다. 자세한 내용은 Wikipedia의 [Accuracy and precision](#)을 참조하십시오.
- 비용은 변환을 실행하는 데 사용한 컴퓨팅 리소스(따라서 비용)의 양을 측정합니다.

정밀도와 재현율 중에서 결정

각 FindMatches 변환에는 precision-recall 파라미터가 포함되어 있습니다. 이 파라미터를 사용하여 다음 중 하나를 지정합니다.

- 변환이 실제로는 일치하지 않는 두 레코드가 일치한다고 잘못 보고할 것이 더 우려되는 경우, 정밀도를 강조해야 합니다.

- 변환이 정말로 일치하는 레코드를 감지하지 못할 것이 더 우려되는 경우, 재현율을 강조해야 합니다.

AWS Glue 콘솔에서 또는 AWS Glue 기계 학습 API 작업을 사용하여 균형을 이룰 수 있습니다.

정밀도를 추구해야 하는 경우

FindMatches가 실제로는 일치하지 않는 레코드 페어가 일치한다고 할 위험이 더 우려되는 경우 정밀도를 추구합니다. 정밀도를 추구하려면 더 높은 정밀도-재현율 균형 값을 선택합니다. 더 큰 값의 경우 FindMatches 변환에는 레코드 페어가 일치해야 한다고 결정하기 위해 더 많은 증거가 필요합니다. 이 변환은 레코드가 일치하지 않는다고 말하는 경향이 있습니다.

예를 들어 FindMatches를 사용하여 비디오 카탈로그에서 중복 항목을 감지하며, 변환에 더 큰 정밀도-재현율 값을 제공한다고 가정하겠습니다. 변환이 Star Wars: A New Hope가 Star Wars: The Empire Strikes Back과 같다고 잘못 감지하는 경우, A New Hope를 원하는 고객에게 The Empire Strikes Back이 표시될 수 있습니다. 이는 부정적인 고객 경험이 됩니다.

그러나 변환이 Star Wars: A New Hope와 Star Wars: Episode IV—A New Hope가 동일한 항목이라고 감지하지 못하면 고객은 처음에는 혼란스러울 수 있으나 결국에는 둘을 동일하게 인식할 수 있습니다. 이는 실수이지만, 이전 시나리오만큼 나쁘지는 않습니다.

재현율을 추구해야 하는 경우

FindMatches 변환 결과가 실제로는 일치하는 레코드 페어를 감지하지 못할 것이 더 우려되는 경우 재현율을 추구합니다. 재현율을 추구하려면 더 낮은 정밀도-재현율 균형 값을 선택합니다. 더 작은 값의 경우 FindMatches 변환에는 레코드 페어가 일치해야 한다고 결정하기 위해 더 적은 증거가 필요합니다. 이 변환은 레코드가 일치한다고 말하는 경향이 있습니다.

예를 들어 이는 보안 조직의 우선 순위일 수 있습니다. 고객을 알려진 사기꾼 목록과 대조해 보고 있으며, 고객이 사기꾼인지 여부를 확인하는 것이 중요하다고 가정하겠습니다. FindMatches를 사용하여 사기꾼 목록을 고객 목록과 대조하고 있습니다. FindMatches가 두 목록 간에 일치 항목을 감지할 때마다 해당 사람이 실제로 사기꾼인지 확인하기 위해 인간 감사자가 지정됩니다. 조직은 정밀도보다 재현율을 선택하는 것을 선호할 수 있습니다. 다시 말해 감사자가 수동으로 검토하고 고객이 사기꾼이 아닌 일부 사례를 거부하도록 하는 것입니다. 그러면 고객이 실제로 사기꾼 목록에 있음을 식별하지 못할 일이 없습니다.

정밀도와 재현율을 모두 추구하는 방법

정밀도와 재현율을 모두 개선하는 가장 좋은 방법은 더 많은 데이터에 레이블을 지정하는 것입니다. 더 많은 데이터에 레이블을 지정하면 FindMatches 변환의 전반적인 정확도가 개선되므로 정밀도와 재

현율도 개선됩니다. 그럼에도 불구하고 가장 정확한 변환에서조차 정밀도 또는 재현율을 추구하는 실험이 필요하거나 중간 값을 선택해야 하는 애매한 상황이 항상 있습니다.

정확도와 비용 중에서 결정

각 FindMatches 변환에는 accuracy-cost 파라미터가 포함되어 있습니다. 이 파라미터를 사용하여 다음 중 하나를 지정할 수 있습니다.

- 변환이 두 레코드가 일치한다고 정확하게 보고하는 것이 더 우려되는 경우, 정확도를 강조해야 합니다.
- 변환 실행 비용이나 속도가 더 우려되는 경우 더 낮은 비용을 강조해야 합니다.

AWS Glue 콘솔에서 또는 AWS Glue 기계 학습 API 작업을 사용하여 균형을 이룰 수 있습니다.

정확도를 추구해야 하는 경우

find matches 결과에 일치 항목이 포함되지 않을 위험이 더 우려되는 경우 정확도를 추구합니다. 정확도를 추구하려면 더 높은 정확도-비용 균형 값을 선택합니다. 더 높은 값의 경우 FindMatches 변환에는 정확하게 일치하는 레코드를 더 철저히 검색하기 위해 더 많은 시간이 필요합니다. 이 파라미터는 일치하지 않는 레코드 페어를 일치한다고 잘못 호출할 가능성을 낮추지는 않습니다. 이 변환은 일치 항목을 찾는 데 더 많은 시간을 보내는 경향이 있습니다.

비용을 추구해야 하는 경우

find matches 변환을 실행하는 데 드는 비용을 더 우려하고 일치 항목의 수를 찾는 것에 덜 우려하는 경우 비용을 추구합니다. 비용을 추구하려면 더 낮은 정확도-비용 균형 값을 선택합니다. 더 낮은 값의 경우 FindMatches 변환은 실행할 리소스가 더 적게 필요합니다. 이 변환은 일치 항목을 더 적게 찾는 경향이 있습니다. 더 낮은 비용을 추구했을 때 결과가 수용 가능한 수준인 경우 이 설정을 사용합니다.

정확도와 더 낮은 비용을 모두 추구하는 방법

더 많은 레코드 쌍을 검토하여 일치 여부를 확인하려면 더 많은 시간이 걸립니다. 품질을 낮추지 않고 비용을 줄이려면 다음과 같은 단계를 수행하면 됩니다.

- 데이터 원본에서 일치에 대해 우려하지 않는 레코드를 제거합니다.
- 일치/불일치 결정을 내리는 것이 유용하지 않다고 확신하는 열을 데이터 원본에서 제거합니다. 이를 결정하는 좋은 방법은 레코드 세트가 “동일”한지 여부에 대한 자신의 결정에 영향을 미친다고 생각하지 않는 열을 제거하는 것입니다.

일치 신뢰도 점수를 사용하여 일치 항목의 품질 추정

일치 신뢰도 점수는 FindMatches에서 찾은 일치 항목의 품질을 추정하여 기계 학습 모델의 신뢰도가 높거나 불확실하거나 낮은 일치 레코드를 구별합니다. 일치 신뢰도 점수는 0에서 1 사이이며, 점수가 높을수록 유사성이 높아집니다. 일치 신뢰도 점수를 검사하면 신뢰도가 높은 일치 항목 클러스터(병합하기로 결정할 수 있음), 불확실한 클러스터(사람이 검토하기로 결정할 수 있음), 신뢰도가 낮은 클러스터(거부하기로 결정할 수 있음)를 구별할 수 있습니다.

일치 신뢰도 점수가 높는데 일치 항목이 없거나 점수가 낮는데 실제로 일치 항목이 있는 경우 훈련 데이터를 조정하는 것이 좋습니다.

신뢰도 점수는 모든 FindMatches 결정을 검토하는 것이 불가능한 대규모 산업 데이터 집합이 있는 경우에 특히 유용합니다.

일치 신뢰도 점수는 AWS Glue 버전 2.0 이상에서 사용 가능합니다.

일치 신뢰도 점수 생성

FindMatches 또는 FindIncrementalMatches API를 호출할 때 computeMatchConfidenceScores의 부울 값을 True로 설정하면 일치 신뢰도 점수를 생성할 수 있습니다.

AWS Glue는 새 column match_confidence_score를 출력에 추가합니다.

일치 점수 매기기 예

예를 들어 다음 일치 레코드를 고려해 보세요.

점수 >= 0.9

일치 레코드 요약:

primary_id	match_id	match_confidence_score
3281355037663	85899345947	0.9823658302132061
1546188247619	85899345947	0.9823658302132061

세부 정보:

raw_id	phone source	website	poi_id	display_position	primary_name locale_name	street1 street2 street3		
city state country postal_code street_in_one_line	primary_id	match_id match_confidence_score						
aeJq8SD01CbIqHFPPL1j g +43262681160	ye p http://www.commerzbank.at ye p::aeJq8SD01CbIqHFPPL1j g geo:47.711590000,16.344020000 Commerzbank Mattersburg	en_US Hauptstr. 59	null	null Forchtenstein	1	AT	7212	
Hauptstr. 59 1546188247619 85899345947	0.9823658302132061							
uWh0k6v2j 51Z4N81Xm-q0 +43268747266	ye p http://www.commerzbank.at ye p::uWh0k6v2j 51Z4N81Xm-q0 geo:47.787420000,16.455440000 Commerzbank Mattersburg	en_US Hauptstr. 9	null	null	Hirm	1	AT	7824
Hauptstr. 9 3281355037663 85899345947	0.9823658302132061							

이 예제에서는 두 레코드가 매우 유사하고 `display_position`, `primary_name`, `street name`을 공유한다는 것을 확인할 수 있습니다.

점수 ≥ 0.8 및 점수 < 0.9

일치 레코드 요약:

primary_id	match_id	match_confidence_score
309237680432	85899345928	0.8309852373674638
3590592666790	85899345928	0.8309852373674638
343597390617	85899345928	0.8309852373674638
249108124906	85899345928	0.8309852373674638
463856477937	85899345928	0.8309852373674638

세부 정보:

primary_id	row_id	phone	source	website	poi_id	display_position	primary_name	locale_name	street1	street2	street3	city	state	country	postal_code	street_in_one_line		
[NVIDVA35Tm41mnaokyvr_w]	343597390617	[85899345928]	0.8309852373674638	null	yeIp	null	yeIp::NNIMVA35Tm41mnaokyvr_w	geo:50.541800000,7.102920000	Eiscafe Dolomiten	en_US	Ahrhutstr. 49	null	null	Bad Neuenahr-Ahrweiler	RP	DE	53474	Ahrhutstr. 49
[53HnQeSvjKclshT9XQFpeQ]	[85899345928]	[+4956746522]	yeIp	null	yeIp::S3HnQeSvjKclshT9XQFpeQ	geo:51.447337266,9.414379068	Eiscafe Dolomiten	en_US	Markt 5	null	null	Grebenstein	HE	DE	34393	Markt 5		
[o6f-p0Xt3mI9PIKpsjx5CQ]	[85899345928]	[+493691744935]	yeIp	null	yeIp::o6f-p0Xt3mI9PIKpsjx5CQ	geo:50.976200000,10.324000000	Eiscafe Dolomiten	en_US	Alexanderstr. 105	null	null	Eisenach	TH	DE	99817	Alexanderstr. 105		
[D10Q2iYDNXonoG52royfjw]	[85899345928]	[+4926445735]	yeIp	null	yeIp::D10Q2iYDNXonoG52royfjw	geo:50.565900000,7.280050000	Eiscafe Dolomiten	en_US	Rheinstr. 15	null	null	Linz	RP	DE	53545	Rheinstr.		

이 예제에서는 두 레코드가 동일한 `primary_name` 및 `country`를 공유한다는 것을 확인할 수 있습니다.

점수 ≥ 0.6 및 점수 < 0.7

일치 레코드 요약:

primary_id	match_id	match_confidence_score
2164663519676	85899345930	0.6971099896480333
317827595278	85899345930	0.6971099896480333
472446424341	85899345930	0.6971099896480333
3118146262932	85899345930	0.6971099896480333
214748380804	85899345930	0.6971099896480333

세부 정보:

primary_id	raw_id	phone source website	poi_id	display_position	primary_name	locale_name	street1	street2	street3	city	state	country	postal_code	street_in_one_line
[10T_R8tk4ngTFXhpy8Bw]	+33490963451	ye p null	ye p::IQT_R8tk4ngTFXhpy8Bw	geo:43.675559000,4.626792000	Le Vésuve	en_US	15 Rue de la Rotonde	null	null	Arles	13	FR	13200	15 Rue de la Rotonde
[188cCaxbvEcug27QmQYMQ]	0.6971099896480333	ye p null	ye p::b8cCaxbvEcug27QmQYMQ	geo:50.631700000,3.067380000	Le Vésuve	en_US	30 ave du President Kennedy	null	null	Lille	59	FR	59000	30 ave du President
[dJ0C4FZmX0S1wEnFB6vJ5g]	+33442750804	ye p null	ye p::dJ0C4FZmX0S1wEnFB6vJ5g	geo:43.427710000,5.236950000	Le Vésuve	en_US	24 ave Bruxelles	null	null	Vitrolles	13	FR	13127	24 ave
[u859qGa561Cj14wypnkj]	+33297251001	ye p null	ye p::u859qGa561Cj14wypnkj	geo:48.071493200,-2.963742000	Le Vésuve	en_US	49 Rue Gén de Gaulle	null	null	Pontivy	56	FR	56300	49 Rue Gén de Gaulle
[3wH0MEhra3DU0UgF_YcoTA]	+33164069200	ye p null	ye p::3wH0MEhra3DU0UgF_YcoTA	geo:48.610984000,2.888184000	Le Vésuve	en_US	59 Avenue Charles de Gaulle	null	null	Mormant	77	FR	77720	59 Avenue Charles de Gaulle

이 예제에서는 두 레코드가 동일한 primary_name만 공유한다는 것을 확인할 수 있습니다.

자세한 내용은 다음 섹션을 참조하세요.

- [5단계: 기계 학습 변환으로 작업 추가 및 실행](#)
- PySpark: [FindMatches 클래스](#)
- PySpark: [FindIncrementalMatches 클래스](#)
- Scala: [FindMatches 클래스](#)
- Scala: [FindIncrementalMatches 클래스](#)

일치 항목 찾기 변환 교육

각 FindMatches 변환에 일치로 간주해야 하는 사항과 일치로 간주하지 말아야 하는 사항을 교육해야 합니다. 파일에 레이블을 추가하고 AWS Glue에 선택 사항을 업로드하여 변환을 교육합니다.

AWS Glue 콘솔에서 또는 AWS Glue 기계 학습 API 작업을 사용하여 이 레이블 지정을 조정할 수 있습니다.

레이블을 몇 번 추가해야 하나요? 몇 개의 레이블이 필요하나요?

이러한 질문에 대한 답은 거의 사용자에게 달려 있습니다. FindMatches가 필요한 정확도 수준을 제공하는지 여부와 추가 레이블 지정 노력이 가치 있다고 생각하는지 여부를 평가해야 합니다. 이를 결정하는 가장 좋은 방법은 AWS Glue 콘솔에서 [품질 평가(Estimate quality)]를 선택할 때 생성할 수 있는 "정밀도", "재현율" 및 "정밀도 재현율 곡선 아래 영역" 지표를 살펴보는 것입니다. 더 많은 작업 세트에 레이블을 지정한 후 이러한 지표를 다시 실행하고 개선되었는지 확인합니다. 작업 세트 몇 개에 레이블을 지정한 후 초점을 둔 지표에 개선이 보이지 않는 경우, 변환 품질이 정체 상태에 도달했을 수 있습니다.

참 긍정과 참 부정 레이블이 모두 필요한 이유는 무엇인가요?

FindMatches 변환은 사용자가 일치한다고 생각하는 항목을 학습하려면 긍정과 부정 예가 모두 필요합니다. FindMatches에서 생성하는 훈련 데이터에 레이블을 지정하는 경우(예: I do not have labels(레이블이 없음) 옵션 사용), FindMatches는 "레이블 세트 ID" 세트를 생성하려고 시도합니다.

각 작업 내에서 일부 레코드에는 동일한 “레이블”을, 다른 레코드에는 다른 “레이블”을 제공합니다. 다시 말해 이 작업은 일반적으로 모두 동일하지도 모두 다르지도 않습니다(그러나 특정 작업이 모두 “동일”하거나 모두 “동일하지 않은” 경우는 관참음).

[S3에서 레이블 업로드(Upload labels from S3)] 옵션을 사용하여 FindMatches 변환을 가르치는 경우 일치하는 레코드와 일치하지 않는 레코드의 예를 모두 포함합니다. 한 유형만 사용할 수 있습니다. 이러한 레이블을 사용하면 더 정확한 FindMatches 변환을 구축할 수 있지만, Generate labeling file(레이블 지정 파일 생성) 옵션을 사용하여 생성하는 일부 레코드에 레이블을 지정해야 합니다.

변환을 교육할 때 변환이 정확히 일치하도록 강제할 수 있나요?

FindMatches 변환은 사용자가 제공하는 레이블에서 학습하므로, 제공된 레이블을 따르지 않는 레코드 페어를 생성할 수 있습니다. FindMatches 변환이 레이블을 따르도록 하려면 FindMatchesParameter에서 EnforceProvidedLabels를 선택합니다.

ML 변환이 항목을 참이 아닌 일치 항목으로 식별하는 경우 어떤 기술을 사용할 수 있나요?

다음과 같은 기술을 사용할 수 있습니다.

- precisionRecallTradeoff을 더 큰 값으로 높입니다. 그러면 결국 더 적은 일치 항목을 찾게 되지만, 충분히 높은 값에 도달하면 큰 클러스터를 분리해야 합니다.
- 잘못된 결과에 해당하는 출력 행을 가져와서 레이블 지정 세트로 다시 포맷합니다(match_id 열을 제거하고, labeling_set_id 및 label 열 추가). 필요한 경우 여러 레이블 지정 세트로 분리(세분화)하여 레이블러가 레이블을 할당하는 동안 각 레이블 지정 세트를 염두에 둘 수 있도록 합니다. 그런 다음 일치하는 세트에 올바르게 레이블을 지정하고 레이블 파일을 업로드한 후 기존 레이블에 추가합니다. 그러면 패턴을 이해하기 위해 찾고 있는 사항에 대해 변환기를 충분히 교육할 수 있습니다.
- (고급) 마지막으로 데이터를 보고 시스템이 알아채지 못하고 있음을 감지할 수 있는 패턴이 있는지 확인합니다. 표준 AWS Glue 함수를 사용해 해당 데이터를 사전 처리하여 데이터를 정규화합니다. 중요한 데이터를 자체 열로 따로 분리하여 알고리즘이 학습하도록 할 내용을 강조 표시합니다. 또는 관련된 데이터가 있는 열에서 결합된 열을 생성합니다.

AWS Glue 콘솔에서 기계 학습 변환 작업

를 사용하여 데이터를 AWS Glue 정리하는 데 사용할 수 있는 사용자 지정 기계 학습 변환을 만들 수 있습니다. AWS Glue 콘솔에서 작업을 생성할 때 이러한 변환을 사용할 수 있습니다.

기계 학습 변환을 생성하는 방법에 대한 자세한 내용은 [AWS Lake Formation FindMatches로 레코드 매칭](#) 단원을 참조하십시오.

주제

- [변환 속성](#)
- [기계 학습 변환 추가 및 편집](#)
- [변환 세부 정보 보기](#)
- [레이블을 사용하여 변환 학습](#)

변환 속성

기존 기계 학습 변환을 보려면 [에 로그인하고 https://console.aws.amazon.com/glue/ 에서 AWS Glue 콘솔을 여십시오. AWS Management Console](#) 탐색 창의 데이터 통합 및 ETL에서 데이터 분류 도구 > 레코드 일치를 선택합니다.

각 변환의 속성:

변환 이름

생성할 때 변환에 부여한 고유 이름입니다.

ID

변환의 고유 식별자입니다.

레이블 수

변환 학습에 도움이 되도록 지정된 레이블 지정 파일의 레이블 수입니다.

상태 표시기

변환이 [준비(Ready)] 상태인지 또는 [훈련 필요(Needs training)] 상태인지를 나타냅니다. 작업에서 기계 학습 변환을 성공적으로 실행하려면 변환이 [준비(Ready)] 상태여야 합니다.

Created

변환이 생성된 날짜입니다.

수정됨

변환이 마지막으로 업데이트된 날짜입니다.

설명

변환에 대해 제공된 설명(제공된 경우)입니다.

AWS Glue 버전

사용된 AWS Glue 버전입니다.

실행 ID

생성할 때 변환에 부여한 고유 이름입니다.

작업 유형

기계 학습 변환의 유형입니다. 예: Find matching records(일치하는 레코드 찾기)

상태 표시기

작업 실행 상태를 나타냅니다. 가능한 상태는 다음과 같습니다.

- Starting(시작 중)
- 실행 중
- Stopping(중지 중)
- Stopped(중지됨)
- Succeeded(성공)
- Failed(실패)
- 제한 시간

Error

상태가 실패인 경우 실패의 원인을 설명하는 오류 메시지가 표시됩니다.

기계 학습 변환 추가 및 편집

AWS Glue 콘솔에서 변환을 보기, 삭제, 설정 및 교육 또는 튜닝할 수 있습니다. 목록에서 변환 옆의 확인란을 선택하고, Action(작업)을 선택한 다음, 수행할 작업을 선택합니다.

새 ML 변환 생성

새 기계 학습 변환을 추가하려면 변환 생성을 선택합니다. 크롤로 추가 마법사의 지시에 따릅니다. 자세한 정보는 [AWS Lake Formation FindMatches로 레코드 매칭](#)을 참조하세요.

단계 1. 변환 속성을 설정합니다.

1. 이름 및 설명(선택 사항)을 입력합니다.

2. 선택적으로 보안 구성을 설정합니다. [기계 학습 변환에 데이터 암호화 사용](#) 섹션을 참조하십시오.
3. 선택적으로 작업 실행 설정을 설정합니다. 작업 실행 설정을 통해 작업 실행 방식을 사용자 지정할 수 있습니다. 작업자 유형, 작업자 수, 작업 제한 시간(분), 재시도 횟수, AWS Glue 버전을 선택합니다.
4. 선택적으로 태그를 설정합니다. 태그는 AWS 리소스에 할당할 수 있는 레이블입니다. 각 태그는 키와 값(선택 사항)으로 구성됩니다. 태그를 사용하여 리소스를 검색 및 필터링하거나 AWS 비용을 추적할 수 있습니다.

단계 2. 테이블과 프라이머리 키를 선택합니다.

1. AWS Glue 카탈로그 데이터베이스 및 테이블을 선택합니다.
2. 선택한 테이블에서 프라이머리 키를 선택합니다. 프라이머리 키 옆에는 일반적으로 데이터 소스의 모든 레코드에 대한 고유 식별자가 포함됩니다.

단계 3. 튜닝 옵션을 선택합니다.

1. 재현율 대 정밀도의 경우 튜닝 값을 선택하여 재현율 또는 정밀도에 유리하도록 변환을 튜닝합니다. 기본적으로 균형이 선택되지만 재현율에 유리하거나 정밀도에 유리하도록 선택할 수 있습니다. 또는 사용자 지정을 선택하고 0.0에서 1.0(포함) 사이의 값을 입력할 수 있습니다.
2. 낮은 비용 및 정확도의 경우 낮은 비용에 유리하거나 정확도에 유리하도록 튜닝 값을 선택합니다. 또는 사용자 지정을 선택하고 0.0에서 1.0(포함) 사이의 값을 입력합니다.
3. 일치 적용의 경우 사용하는 레이블과 일치하도록 강제 출력함으로서 ML 변환을 학습시키려면 레이블과 일치하도록 강제 출력을 선택합니다.

4단계. 검토 및 생성.

1. 1~3단계의 옵션을 검토합니다.
2. 수정이 필요한 모든 단계에서 편집을 선택합니다. 변환 생성을 선택하여 변환 생성 마법사를 완료합니다.

기계 학습 변환에 데이터 암호화 사용

기계 학습 변환을 AWS Glue에 추가할 때 데이터 원본 또는 데이터 대상과 연결된 보안 구성을 선택적으로 지정할 수 있습니다. 데이터를 저장하는 데 사용된 Amazon S3 버킷이 보안 구성으로 암호화된 경우 변환을 생성할 때 동일한 보안 구성을 지정합니다.

또한 서버 측 암호화 AWS KMS (SSE-KMS) 를 사용하여 모델과 레이블을 암호화하여 승인되지 않은 사람이 검사하지 못하도록 할 수 있습니다. 이 옵션을 선택하면 AWS KMS key 이름별로 선택하라는 메시지가 표시되거나 Enter a key ARN을 선택할 수 있습니다. KMS 키에 대한 ARN을 입력하도록 선택하면 KMS 키 ARN을 입력할 수 있는 두 번째 필드가 나타납니다.

Note

현재 사용자 지정 암호화 키를 사용하는 ML 변환은 다음 리전에서 지원되지 않습니다.

- 아시아 태평양(오사카) - ap-northeast-3

변환 세부 정보 보기

변환 속성 보기

변환 속성 페이지에는 변환 속성이 포함되어 있습니다. 이 탭은 다음을 포함하여 변환 정의에 대한 세부 정보를 보여 줍니다.

- Transform name(변환 이름)은 변환의 이름을 보여 줍니다.
- Type(유형)에는 변환의 유형이 나열됩니다.
- Status(상태)는 변환이 스크립트 또는 작업에 사용할 준비가 되었는지 여부를 표시합니다.
- Force output to match labels(레이블과 일치하도록 출력 강제)는 변환이 사용자가 제공한 레이블과 일치하도록 출력을 강제하는지 여부를 표시합니다.
- [Spark 버전(Spark version)]은 변환을 추가할 때 [태스크 실행 속성(Task run properties)]에서 선택한 AWS Glue 버전과 관련이 있습니다. AWS Glue 1.0 및 Spark 2.4는 대부분의 고객에게 권장됩니다. 자세한 내용은 [AWS Glue 버전](#)을 참조하세요.

기록, 품질 예측 및 태그 탭

변환 세부 정보에는 변환을 생성할 때 정의한 정보가 포함되어 있습니다. 변환에 대한 세부 정보를 보려면 Machine learning transforms(기계 학습 변환) 목록에서 변환을 선택하고 다음 탭에서 정보를 검토합니다.

- 기록
- 품질 예측
- Tags

기록

History(기록) 탭에는 변환 작업 실행 기록이 표시됩니다. 변환을 교육하기 위해 여러 유형의 작업이 실행됩니다. 각 작업에 대한 실행 지표에는 다음이 포함됩니다.

- Run ID(실행 ID)는 이 작업의 각 실행에 대해 AWS Glue에서 생성된 식별자입니다.
- Task type(작업 유형)은 작업 실행의 유형을 보여 줍니다.
- Run status(실행 상태)에는 맨 위에 가장 최근 실행부터 순서대로 나열된 각 작업의 성공 여부를 보여 줍니다.
- 오류는 성공적이지 못한 실행의 결과로서 오류 메시지의 세부 정보를 보여줍니다.
- Start time(시작 시간)은 작업이 시작된 날짜와 시간(로컬 시간)을 보여줍니다.
- 종료 시간은 작업이 종료되는 날짜와 시간(로컬 시간)을 보여줍니다.
- Logs(로그)는 이 작업을 실행할 때 stdout에 작성되는 로그와 연결됩니다.

로그 링크를 클릭하면 Amazon Logs로 CloudWatch 이동합니다. 여기에서 에서 생성된 테이블에 대한 세부 정보와 발생한 모든 오류를 볼 수 있습니다. AWS Glue Data Catalog CloudWatch 콘솔에서 로그 보존 기간을 관리할 수 있습니다. 기본 로그 보관은 Never Expire. 보존 기간을 변경하는 방법에 대한 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 [CloudWatch 로그 데이터 보존 변경을 참조하십시오](#).

- 레이블 파일은 Amazon S3에 대한 생성된 레이블 지정 파일의 링크를 보여줍니다.

품질 예측

Estimate quality(예상 품질) 탭은 변환의 품질을 측정하는 데 사용할 수 있는 지표를 보여 줍니다. 추정치는 레이블 지정 데이터의 하위 세트를 사용하는 변환 일치 예측과 사용자가 제공한 레이블을 비교해서 계산합니다. 이 추정치는 근사값입니다. 이 탭에서 Estimate quality(예상 품질) 작업 실행을 호출할 수 있습니다.

Estimate quality(예상 품질) 탭에는 다음 속성이 포함된 마지막 Estimate quality(예상 품질) 실행의 지표가 표시됩니다.

- Area under the Precision-Recall curve(정밀도-재현율 곡선 아래 면적)는 전체적 변환 품질의 상한값을 예측하는 단일 숫자입니다. 이 항목은 정밀도-재현율 파라미터에 대해 수행한 선택과 무관합니다. 값이 높을수록 더 매력적인 정밀도-재현율 트레이드오프가 있음을 나타냅니다.
- Precision(정밀도)은 변환이 일치를 정확하게 예측하는 빈도를 예측치로 표시합니다.
- Recall upper limit(재현율 상한)는 실제 일치에 대해 변환이 일치를 예측하는 빈도를 예측치로 표시합니다.

- F1은 0~1의 범위에서 변환의 정확도를 예측합니다. 여기서 1은 최상의 정확도입니다. 자세한 내용은 Wikipedia의 [F1 점수](#)를 참조하십시오.
- [열 중요도(Column importance)] 테이블에는 각 열의 열 이름과 중요도 점수가 표시됩니다. 열 중요도는 일치 작업을 수행하는 데 가장 많이 사용되는 레코드의 열을 식별하여 열이 모델에 기여하는 방식을 이해하는 데 도움이 됩니다. 이 데이터는 열 중요도를 높이거나 낮추기 위해 레이블 집합을 추가하거나 변경하라는 메시지를 표시할 수 있습니다.

중요도 열은 1.0 이하의 십진수로 각 열에 대한 숫자 점수를 제공합니다.

품질 예측과 실제 품질 비교 이해에 대한 자세한 내용은 [품질 추정치와 end-to-end \(실제\) 품질 비교](#) 단원을 참조하십시오.

변환 튜닝에 대한 자세한 내용은 [AWS Glue에서 기계 학습 변환 튜닝](#) 단원을 참조하십시오.

품질 추정치와 end-to-end (실제) 품질 비교

AWS Glue에서는 사용자가 일치하는 레이블을 제공했지만 이전에 모델에서는 본 적이 없는 수많은 레코드 페어로 기계 학습된 내부 모델을 제시하여 변환의 품질을 예측합니다. 이러한 품질 예측은 기계 학습된 모델의 품질 함수입니다(변환을 “교육”하기 위해 레이블 지정하는 수많은 레코드의 영향을 받음). 실제 재현율 (에 의해 자동으로 계산되지 않음 ML transform) 은 기계 학습된 모델과 가능한 다양한 일치 항목을 제안하는 ML transform 필터링 메커니즘의 영향을 받기도 합니다. end-to-end

주로 낮은 비용 대비 정확도 튜닝 값을 사용하여 이 필터링 방법을 튜닝할 수 있습니다. 튜닝 값이 정확도 쪽으로 이동할수록 시스템은 일치 항목일 수 있는 레코드 페어에 대해 더 철저하고 비용이 많이 드는 검색을 수행합니다. 기계 학습 모델에는 더 많은 레코드 쌍이 제공되므로 사용자 end-to-end 또는 실제 리콜이 ML transform 추정 리콜 지표에 가까워집니다. 따라서 경기 비용/정확도 트레이드오프 변경으로 인한 경기 end-to-end 품질 변화는 일반적으로 품질 추정치에 반영되지 않습니다.

Tags

태그는 리소스에 할당할 수 있는 레이블입니다. AWS 각 태그는 키와 값(선택 사항)으로 구성됩니다. 태그를 사용하여 리소스를 검색 및 필터링하거나 AWS 비용을 추적할 수 있습니다.

레이블을 사용하여 변환 학습

ML 변환 세부 정보 페이지에서 변환 학습을 선택하여 레이블(예제)을 사용한 ML 변환을 학습할 수 있습니다. 예제(레이블이라고도 함)를 제공하여 기계 학습 알고리즘을 학습할 때 사용할 기존 레이블을 선택하거나 레이블 지정 파일을 생성할 수 있습니다.

Teach the transform using labels

Labeling

Teach your machine learning algorithms by providing examples, called labels. For your transform, provide examples of matching and nonmatching records.

I do not have labels

I have labels

▶ How to label

Generate labeling file

AWS Glue extracts records from your source data and suggests potential matching records. The file will contain approximately 100 data samples for you to work with. You can download the file once it has been generated.

S3 path to store the generated label file

[View](#)

[Browse S3](#)

[Generate labeling file](#)

[Download labeling file](#)

Upload labels from S3

The completed labeling file must be in the correct format and in Amazon S3.

S3 path where the label file is stored

[View](#)

[Browse S3](#)

Existing labels

Append to my existing labels

Overwrite my existing labels

[Upload labeling file from S3](#)

- 레이블 지정 - 레이블이 있는 경우 레이블 있음을 선택합니다. 레이블이 없으면 레이블 지정 파일을 생성하는 다음 단계를 계속할 수 있습니다.
- 레이블 지정 파일 생성 - AWS Glue는 소스 데이터에서 레코드를 추출하고 일치 가능성이 있는 레코드를 제안합니다. 생성된 레이블 파일을 저장할 Amazon S3 버킷을 선택합니다. 레이블 지정 파일 생성을 선택하여 프로세스를 시작합니다. 완료되면 레이블 지정 파일 다운로드를 선택합니다. 다운로드한 파일에는 레이블에 입력할 수 있는 레이블 열이 있습니다.
- Amazon S3에서 레이블 업로드 - 레이블 파일이 저장되어 있는 Amazon S3 버킷에서 완성된 레이블 지정 파일을 선택합니다. 그런 다음 기존 레이블에 레이블을 추가하거나 기존 레이블을 덮어쓸지 선택합니다. Amazon S3에서 레이블 지정 파일 업로드를 선택합니다.

자습서: AWS Glue로 기계 학습 변환 생성

이 자습서에서는 AWS Glue를 사용하여 ML(기계 학습) 변환을 생성하고 관리하는 작업을 안내합니다. 이 자습서를 사용하기 전에, AWS Glue 콘솔을 사용하여 크롤러 및 작업을 추가하고 스크립트를 편집

하는 방법을 숙지해야 합니다. 또한 Amazon Simple Storage Service(Amazon S3) 콘솔에서 파일을 찾고 다운로드하는 방법도 잘 알아야 합니다.

이 예제에서는 FindMatches 변환을 만들어 일치하는 레코드를 찾고, 일치 레코드 및 불일치 레코드를 식별하는 방법을 변환에 학습시키고, AWS Glue 작업에 이 변환을 사용해 봅니다. AWS Glue 작업은 match_id라는 추가 열을 포함하는 새 Amazon S3 파일을 작성합니다.

이 자습서에서 사용하는 원본 데이터는 dblp_acm_records.csv라는 파일입니다. 이 파일은 원래의 [DBLP ACM 데이터 세트](#)에서 사용 가능한 교육용 게시물(DBLP 및 ACM)을 수정한 버전입니다. dblp_acm_records.csv 파일은 BOM(바이트 순서 표시) 없는 UTF-8 형식의 CSV(쉼표로 구분된 값) 파일입니다.

두 번째 파일인 dblp_acm_labels.csv는 이 자습서에 따라 변환을 학습시키는 데 사용할 일치 레코드와 불일치 레코드가 둘 다 들어 있는 레이블 지정 예제 파일입니다.

주제

- [1단계: 소스 데이터 크롤링](#)
- [2단계: 기계 학습 변환 추가](#)
- [3단계: 기계 학습 변환 학습](#)
- [4단계: 기계 학습 변환의 품질 예측](#)
- [5단계: 기계 학습 변환으로 작업 추가 및 실행](#)
- [6단계: Amazon S3의 출력 데이터 확인](#)

1단계: 소스 데이터 크롤링

우선, 원본 Amazon S3 CSV 파일을 크롤링하여 Data Catalog에 그와 상응하는 메타데이터 테이블을 생성합니다.

Important

크롤러가 이 CSV 파일 전용의 테이블을 만들게 하기 위해 CSV 원본 데이터를 나머지 파일과 다른 Amazon S3 폴더에 저장합니다.

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 크롤러, 크롤러 추가를 선택합니다.

3. 마법사를 따라 `demo-db-dblp-acm` 데이터베이스로 출력되는 `demo-crawl-dblp-acm`이라는 크롤러를 만들고 실행합니다. `demo-db-dblp-acm` 데이터베이스가 아직 없으면 이 마법사를 실행하는 동안 만듭니다. 현재 AWS 리전에 있는 샘플 데이터로 이동하는 Amazon S3 포함 경로를 선택합니다. 예를 들어 `us-east-1`의 경우, Amazon S3에서 소스 파일로 이동하는 포함 경로는 `s3://ml-transforms-public-datasets-us-east-1/dblp-acm/records/dblp_acm_records.csv`입니다.

작업이 성공하면 크롤러가 `id`, `title`, `authors`, `venue`, `year` 및 `source` 열이 있는 `dblp_acm_records_csv` 테이블을 만듭니다.

2단계: 기계 학습 변환 추가

다음으로, `demo-crawl-dblp-acm`이라는 크롤러에서 만든 데이터 원본 테이블의 스키마를 토대로 한 기계 학습 변환을 추가합니다.

1. AWS Glue Console의 탐색 창의 데이터 통합 및 ETL 아래에서 데이터 분류 도구 > 레코드 일치 선택한 다음 변환 추가를 선택합니다. 마법사를 따라 다음 속성으로 Find matches 변환을 생성합니다.
 - a. 변환 이름으로 **demo-xform-dblp-acm**을 입력합니다. 이것은 원본 데이터에서 일치 항목을 찾을 때 사용하는 변환의 이름입니다.
 - b. IAM 역할(IAM role)로는 Amazon S3 원본 데이터, 레이블 지정 파일 및 AWS Glue API 작업에 대한 권한이 있는 IAM 역할을 선택합니다. 자세한 내용은 AWS Glue Developer Guide의 [Create an IAM Role for AWS Glue](#)를 참조하세요.
 - c. [데이터 원본(Data source)]의 경우, 데이터베이스 [`demo-db-dblp-acm`]의 [`dblp_acm_records_csv`]라는 이름의 테이블을 선택합니다.
 - d. 기본 키로는 테이블의 기본 키 열(`id`)을 선택합니다.
2. 마법사에서 완료 선택하고 ML transforms(ML 변환) 목록으로 돌아갑니다.

3단계: 기계 학습 변환 학습

이제 자습서의 레이블 지정 샘플 파일을 사용하여 기계 학습 변환을 학습시켜야 합니다.

Ready for use(사용 준비 완료) 상태가 되기 전에는 기계어 변환을 ETL(추출, 변환, 로드) 작업에 사용할 수 없습니다. 변환을 준비시키려면 일치 레코드와 불일치 레코드의 예제를 제공하여 일치 및 불일치 레코드를 식별하는 방법을 가르쳐야 합니다. 변환을 학습시키기 위해 레이블 파일을 생성하고, 레이블을 추가하고, 그 레이블 파일을 업로드할 수 있습니다. 이 자습서에서는 `dblp_acm_labels.csv`라는

레이블 지정 예제 파일을 사용합니다. 레이블 지정 프로세스에 대한 자세한 내용은 [레이블링](#) 단원을 참조하십시오.

1. AWS Glue Console의 탐색 창에서 레코드 일치를 선택합니다.
2. demo-xform-dblp-acm 변환을 선택한 다음 Action(작업)과 Teach(학습)를 선택합니다. 마법사를 따라 Find matches 변환을 학습시킵니다.
3. 변환 속성 페이지에서 I have labels(레이블 있음)를 선택합니다. 현재 AWS 리전에 있는 샘플 레이블 지정 파일로 이동하는 Amazon S3 경로를 선택합니다. 예를 들어 us-east-1의 경우, Amazon S3 경로 s3://ml-transforms-public-datasets-us-east-1/dblp-acm/labels/dblp_acm_labels.csv에서 제공한 레이블 지정 파일을 업로드하되 기존 레이블 [덮어쓰기 (overwrite)] 옵션을 포함합니다. 레이블 지정 파일이 AWS Glue 콘솔과 같은 리전의 Amazon S3에 있어야 합니다.

레이블 지정 파일을 업로드하면 해당 변환에 데이터 원본 처리 방법을 학습시키는 데 사용할 레이블을 추가하거나 덮어쓰는 작업이 AWS Glue에서 시작됩니다.

4. 마법사의 마지막 페이지에서 완료를 선택하고 ML transforms(ML 변환) 목록으로 돌아옵니다.

4단계: 기계 학습 변환의 품질 예측

이제 기계 학습 변환의 품질을 예측할 수 있습니다. 품질은 레이블 지정 작업을 얼마나 많이 했는지에 따라 달라집니다. 품질 예측에 대한 자세한 내용은 [품질 예측](#) 단원을 참조하십시오.

1. AWS Glue Console의 탐색 창의 데이터 통합 및 ETL에서 데이터 분류 도구 > 레코드 일치를 선택합니다.
2. demo-xform-dblp-acm 변환을 선택하고 Estimate quality(품질 예측) 탭을 선택합니다. 해당 변환의 현재 품질이 예측되어 있으면 이 탭에 표시됩니다.
3. Estimate quality(품질 예측)를 선택하여 변환의 품질 예측 작업을 시작합니다. 품질 예측의 정확성은 원본 데이터의 레이블 지정에 달려 있습니다.
4. History(기록) 탭으로 이동합니다. 이 창에는 Estimating quality(품질 예측) 작업을 비롯하여 해당 변환에 실행한 작업이 나열되어 있습니다. 작업 실행에 대한 자세한 내용은 로그를 참조하십시오. 완료된 작업의 실행 상태가 성공인지 확인합니다.

5단계: 기계 학습 변환으로 작업 추가 및 실행

이 단계에서는 기계 학습 변환을 사용하여 AWS Glue에서 작업을 추가하고 실행해 봅니다. demo-xform-dblp-acm 변환이 Ready for use(사용 준비 완료) 상태이면 ETL 작업에 사용할 수 있습니다.

1. AWS Glue 콘솔의 탐색 창에서 작업을 선택합니다.
2. 작업 추가를 선택하고, 마법사의 단계에 따라 생성된 스크립트로 ETL Spark 작업을 만듭니다. 변환에 대해 다음 속성 값을 선택하십시오.
 - a. 이름으로 이 자습서의 예제 작업인 demo-etl-dblp-acm을 선택합니다.
 - b. [IAM 역할(IAM role)]로는 Amazon S3 원본 데이터, 레이블 지정 파일 및 AWS Glue API 작업에 대한 권한이 있는 IAM 역할을 선택합니다. 자세한 내용은 AWS Glue Developer Guide의 [Create an IAM Role for AWS Glue](#)를 참조하세요.
 - c. ETL 언어로 Scala를 선택합니다. 이것은 ETL 스크립트의 프로그래밍 언어입니다.
 - d. 스크립트 파일 이름으로 demo-etl-dblp-acm을 선택합니다. 이것은 Scala 스크립트의 파일 이름입니다(작업 이름과 동일).
 - e. 데이터 원본으로 dblp_acm_records_csv를 선택합니다. 선택한 데이터 원본이 기계 학습 변환의 데이터 원본 스키마와 일치해야 합니다.
 - f. Transform type(변환 유형)으로 Find matching records(일치 레코드 찾기)를 선택하여 기계 학습 변환을 사용하는 작업을 만듭니다.
 - g. Remove duplicate records(중복 레코드 제거)를 선택 취소합니다. 작성되는 출력 레코드에 match_id 필드가 하나 더 추가되므로 중복 레코드를 제거하지 않아도 됩니다.
 - h. 변환으로 이 작업에 사용할 기계 학습 변환인 demo-xform-dblp-acm을 선택합니다.
 - i. 데이터 대상에서 테이블 생성에서 다음 속성의 테이블을 만들도록 선택합니다.
 - [데이터 스토어 유형(Data store type)] - **Amazon S3**
 - [포맷(Format)] - **CSV**
 - [압축 유형(Compression type)] - **None**
 - [대상 경로(Target path)] - 작업의 출력이 쓰이는 Amazon S3 경로(현재 콘솔 AWS 리전에서)
3. 작업 저장 및 스크립트 편집을 선택하여 스크립트 편집기 페이지를 표시합니다.
4. 대상 경로에 대한 작업 출력을 파티션 파일 하나에 쓰도록 하는 문을 추가하여 스크립트를 편집합니다. FindMatches 변환을 실행하는 문 바로 뒤에 이 문을 추가합니다. 이 문은 다음과 비슷합니다.

```
val single_partition = findmatches1.repartition(1)
```

출력을 `.writeDynamicFrame(single_partition)`으로 쓰도록 `.writeDynamicFrame(findmatches1)` 문을 수정해야 합니다.

5. 스크립트를 편집한 다음 저장을 선택합니다. 수정된 스크립트는 다음 코드와 비슷한 모양이지만 사용자 환경에 맞게 조정되어 있습니다.

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.ml.FindMatches
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    // @type: DataSource
    // @args: [database = "demo-db-dblp-acm", table_name = "dblp_acm_records_csv",
transformation_ctx = "datasource0"]
    // @return: datasource0
    // @inputs: []
    val datasource0 = glueContext.getCatalogSource(database = "demo-db-dblp-acm",
tableName = "dblp_acm_records_csv", redshiftTmpDir = "", transformationContext =
"datasource0").getDynamicFrame()
    // @type: FindMatches
    // @args: [transformId = "tfm-123456789012", emitFusion = false,
survivorComparisonField = "<primary_id>", transformation_ctx = "findmatches1"]
    // @return: findmatches1
    // @inputs: [frame = datasource0]
    val findmatches1 = FindMatches.apply(frame = datasource0, transformId
= "tfm-123456789012", transformationContext = "findmatches1",
computeMatchConfidenceScores = true)

    // Repartition the previous DynamicFrame into a single partition.
    val single_partition = findmatches1.repartition(1)

    // @type: DataSink
```

```

// @args: [connection_type = "s3", connection_options = {"path": "s3://aws-
glue-ml-transforms-data/sal"}, format = "csv", transformation_ctx = "datasink2"]
// @return: datasink2
// @inputs: [frame = findmatches1]
val datasink2 = glueContext.getSinkWithFormat(connectionType =
"s3", options = JsonOptions("""{"path": "s3://aws-glue-ml-transforms-
data/sal"}"""), transformationContext = "datasink2", format =
"csv").writeDynamicFrame(single_partition)
    Job.commit()
}
}

```

6. 작업 실행을 선택하여 작업을 실행하기 시작합니다. 작업 목록에서 작업의 상태를 확인합니다. 작업이 완료되면 ML transform(ML 변환)의 History(기록) 탭에 새로운 실행 ID 행이 ETL 작업 유형으로 추가됩니다.
7. 작업, History(기록) 탭으로 이동합니다. 이 창에는 작업 실행이 나열되어 있습니다. 작업 실행에 대한 자세한 내용은 로그를 참조하십시오. 완료된 작업의 실행 상태가 성공인지 확인합니다.

6단계: Amazon S3의 출력 데이터 확인

이 단계에서는 작업을 추가할 때 선택한 Amazon S3 버킷에서 작업 실행의 출력을 확인합니다. 출력 파일을 로컬 시스템에 다운로드하고, 일치하는 레코드가 식별되었는지 확인할 수 있습니다.

1. <https://console.aws.amazon.com/s3/>에서 Amazon S3 콘솔을 엽니다.
2. demo-etl-dblp-acm 작업의 대상 출력 파일을 다운로드합니다. 해당 파일을 스프레드시트 애플리케이션에서 엽니다. (파일을 제대로 열려면 파일 확장명 .csv를 추가해야 할 수 있음)

아래 그림은 Microsoft Excel의 출력을 발췌한 것입니다.

	B	C	D	E	F	G	H	I
1	title	authors	venue	year	source	primary_id	match_id	match_confidence_score
2	Semantic Integration of Environmental Models for Application to Global Information S.D. Scott Mackay		SIGMOD Record	1999	DBLP	3092	0	0.830985237
3	Semantic integration of environmental models for application to global information s.d. Scott Mackay		ACM SIGMOD Record	1999	ACM	3590	0	0.830985237
4	Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balan Viswanath Poozala, Yannis E. J.VLDB			1996	DBLP	3435	1	0.801848258
5	Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balan Viswanath Poozala, Yannis E. J.Very Large Data Bas			1996	ACM	2491	1	0.801848258
6	Incremental Maintenance for Non-Distributive Aggregate Functions	Themistoklis Palpanas, Richar	VLDB	2002	DBLP	4638	2	0.697109993
7	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Da Zhao-Hui Tang, Georges Gard		VLDB	1996	DBLP	3768	3	0.791241276
8	Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Da Georges Gardarin, Jean-Rober		Very Large Data Bas	1996	ACM	5926	3	0.791241276
9	Benchmarking Spatial Join Operations with Spatial Output	Erik G. Hoel, Hanan Samet	Very Large Data Bas	1995	ACM	9739	4	0.723535024
10	Benchmarking Spatial Join Operations with Spatial Output	Erik G. Hoel, Hanan Samet	VLDB	1995	DBLP	8124	4	0.723535024
11	Efficient geometry-based similarity search of 3D spatial databases	Daniel A. Keim	International Confe	1999	ACM	5647	5	0.786350237
12	Efficient Geometry-based Similarity Search of 3D Spatial Databases	Daniel A. Keim	SIGMOD Conference	1999	DBLP	3432	5	0.786350237
13	Mining the World Wide Web: An Information Search Approach - Book Review	Aris M. Oukel		2002	DBLP	6790	6	0.697109993
14	Enhanced Abstract Data Types in Object-Relational Databases	Praveen Seshadri	VLDB J.	1998	DBLP	2617	7	0.827350237
15	Enhanced abstract data types in object-relational databases	Praveen Seshadri	The VLDB Journal &	1998	ACM	4906	7	0.827350237
16	Report on DART '96: Databases: Active and Real-Time (Concepts meet Practice)	Nandit Soparkar, Krithi Ramani	SIGMOD Record	1997	DBLP	7937	8	0.708350237
17	Report on DART '96: databases: active and real-time (concepts meet practice)	Krithi Ramamritham, Nandit S	ACM SIGMOD Record	1997	ACM	8193	8	0.708350237
18	UniSQL's next-generation object-relational database management system	Albert D'Andrea, Phil Janus	ACM SIGMOD Record	1996	ACM	8491	9	0.818340237
19	UniSQL's Next-Generation Object-Relational Database Management System	Phil Janus, Albert D'Andrea	SIGMOD Record	1996	DBLP	4869	9	0.818340237

데이터 원본과 대상 파일 모두 레코드가 4,911개 있습니다. 그러나 Find matches 변환은 출력에서 일치하는 레코드를 식별하기 위해 match_id라는 열을 하나 더 추가합니다. match_id가 동

일한 행은 일치하는 레코드로 간주됩니다. `match_confidence_score`는 Find matches에서 찾은 일치 항목의 품질을 추정하는, 0에서 1 사이의 숫자입니다.

3. 일치하는 레코드를 보기 쉽도록 출력 파일을 `match_id`로 정렬합니다. 나머지 열의 값을 비교하여 Find matches 변환의 결과에 동의할 만한지 알아봅니다. 동의할 수 없으면 계속해서 레이블을 더 추가하여 변환을 학습시킬 수 있습니다.

또한 `title` 등 다른 필드를 기준으로 파일을 정렬하여 제목이 비슷한 레코드의 `match_id`가 동일한지 확인할 수도 있습니다.

중분 일치 항목 찾기

일치 항목 찾기 기능을 사용하면 레코드에 공통된 고유 식별자가 없고 정확히 일치하는 필드가 없는 경우에도 데이터 집합에서 중복 레코드나 일치 레코드를 식별할 수 있습니다. 일치 항목 찾기 변환의 초기 릴리스에서는 단일 데이터 집합 내의 일치 레코드를 식별했습니다. 데이터 집합에 새 데이터를 추가하는 경우 정리된 기존 데이터 집합과 병합하고 병합된 전체 데이터 집합에 대해 일치를 다시 실행해야 했습니다.

중분 일치 기능을 사용하면 기존 일치 데이터 집합과 중분 레코드를 더 간단하게 일치시킬 수 있습니다. 기존 고객 데이터 집합과 잠재 고객 데이터를 일치시키려는 경우를 가정합니다. 중분 일치 기능을 사용하면 결과를 단일 데이터베이스나 테이블에 병합하여 잠재 고객 및 고객의 기존 데이터베이스와 수십만 명의 신규 잠재 고객을 유연성 있게 일치시킬 수 있습니다. 중분 일치 항목 찾기 최적화는 새 데이터 집합과 기존 데이터 집합 간에만 일치시켜 계산 시간을 단축하므로 비용도 절감됩니다.

중분 일치 사용법은 [자습서: AWS Glue로 기계 학습 변환 생성](#)에 설명된 일치 항목 찾기와 유사합니다. 이 주제에서는 중분 일치와의 차이점만 설명합니다.

자세한 내용은 [중분 데이터 일치](#)에 대한 블로그 게시물을 참조하세요.

중분 일치 작업 실행

다음 절차에서는 다음과 같이 가정합니다.

- 기존 데이터 세트를 `first_records` 테이블로 크롤링했습니다. `first_records` 데이터 세트는 일치하는 데이터 세트이거나 일치하는 작업의 출력이어야 합니다.
- AWS Glue 버전 2.0을 사용하여 일치 항목 찾기 변환을 생성하고 훈련시켰습니다. 중분 일치는 이 버전의 AWS Glue에서만 지원됩니다.
- ETL 언어는 Scala입니다. Python도 지원됩니다.
- `demo-xform`이라는 모델이 이미 생성되어 있습니다.

1. 중분 데이터 집합을 `second_records` 테이블로 크롤링합니다.
2. AWS Glue 콘솔의 탐색 창에서 작업을 선택합니다.
3. 작업 추가를 선택하고, 마법사의 단계에 따라 생성된 스크립트로 ETL Spark 작업을 만듭니다. 변환에 대해 다음 속성 값을 선택하십시오.
 - a. 이름(Name)에서 `demo-etl`을 선택합니다.
 - b. IAM 역할(IAM role)에서 Amazon S3 소스 데이터, 레이블 지정 파일, [AWS Glue API 작업](#)에 대한 권한이 있는 IAM 역할을 선택합니다.
 - c. ETL 언어로 Scala를 선택합니다.
 - d. 스크립트 파일 이름(Script file name)에서 `demo-etl`을 선택합니다. Scala 스크립트의 파일 이름입니다.
 - e. 데이터 원본(Data source)에서 `first_records`를 선택합니다. 선택한 데이터 원본이 기계 학습 변환의 데이터 원본 스키마와 일치해야 합니다.
 - f. Transform type(변환 유형)으로 Find matching records(일치 레코드 찾기)를 선택하여 기계 학습 변환을 사용하는 작업을 만듭니다.
 - g. 중분 일치 옵션을 선택하고 데이터 원본(Data Source)에서 `second_records`라는 테이블을 선택합니다.
 - h. 변환(Transform)에서 이 작업에 사용할 기계 학습 변환인 `demo-xform`을 선택합니다.
 - i. 데이터 대상에 테이블 생성(Create tables in your data target) 또는 데이터 카탈로그 내 테이블 사용 및 데이터 대상 업데이트(Use tables in the data catalog and update your data target)를 선택합니다.
4. 작업 저장 및 스크립트 편집을 선택하여 스크립트 편집기 페이지를 표시합니다.
5. 작업 실행을 선택하여 작업을 실행하기 시작합니다.

시각적 작업에서 FindMatches 사용

AWS Glue Studio에서 FindMatches 변환을 사용하려면 FindMatches API를 간접 호출하는 사용자 지정 변환 노드를 사용하면 됩니다. 사용자 지정 변환을 사용하는 방법에 대한 자세한 내용은 [사용자 지정 변환 생성](#)을 참조하세요.

Note

현재 FindMatches API는 Glue 2.0에서만 작동합니다. FindMatches API를 간접 호출하는 사용자 지정 변환을 포함하는 작업을 실행하려면 AWS Glue 버전이 작업 세부 정보 탭에서 Glue

2.0인지 확인합니다. AWS Glue의 버전이 Glue 2.0이 아닌 경우 작업이 런타임에 실패하고 'cannot import name 'FindMatches' from 'awsglueml.transforms' 오류 메시지가 표시됩니다.

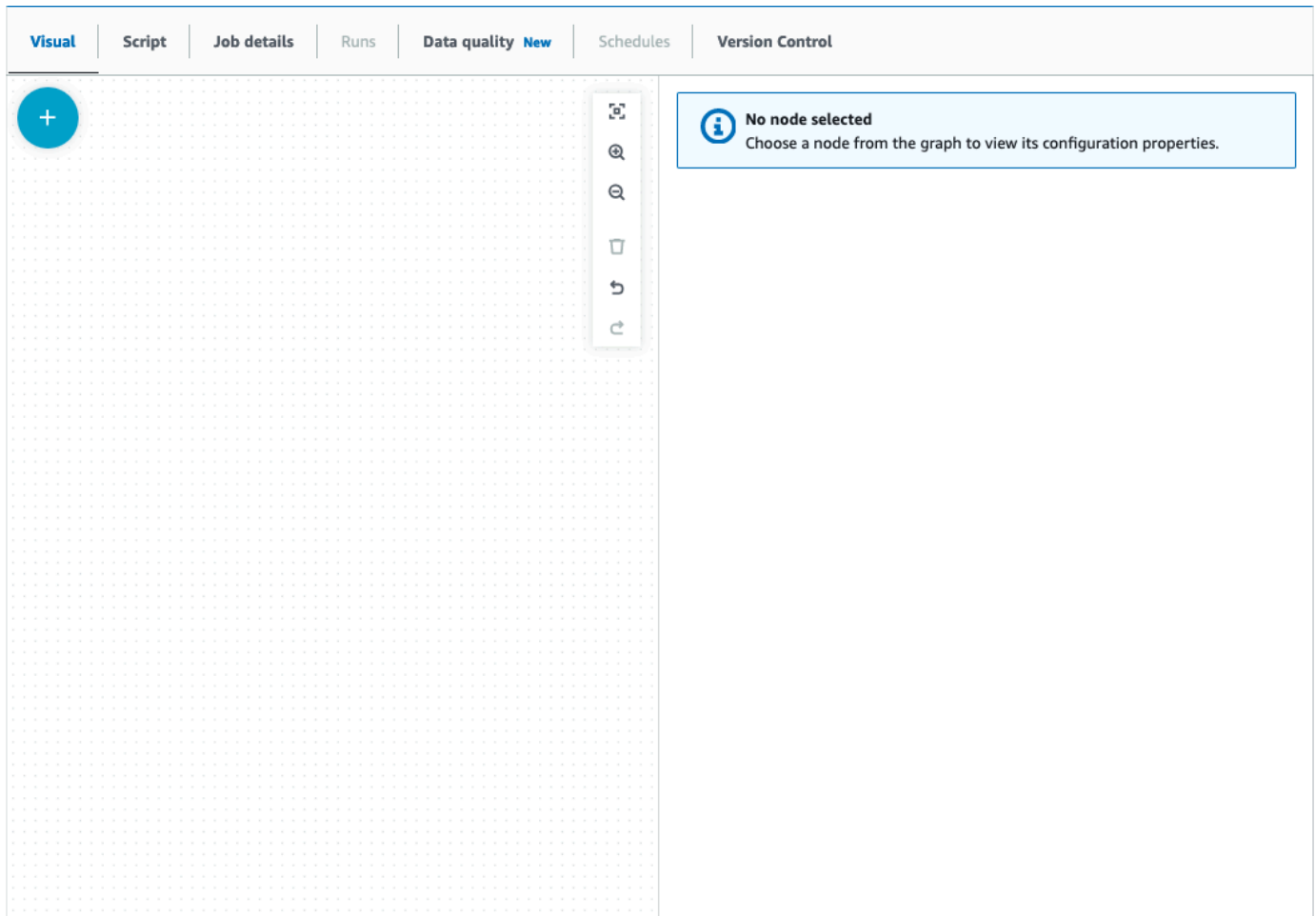
필수 조건

- 일치 항목 찾기 변환을 사용하려면 <https://console.aws.amazon.com/gluestudio/>에서 AWS Glue Studio 콘솔을 엽니다.
- 기계 학습 변환을 생성합니다. 생성되면 transformId가 생성됩니다. 아래 단계를 수행하려면 이 ID가 필요합니다. 기계 학습 변환을 생성하는 방법에 대한 자세한 내용은 [기계 학습 변환 추가 및 편집](#)을 참조하세요.

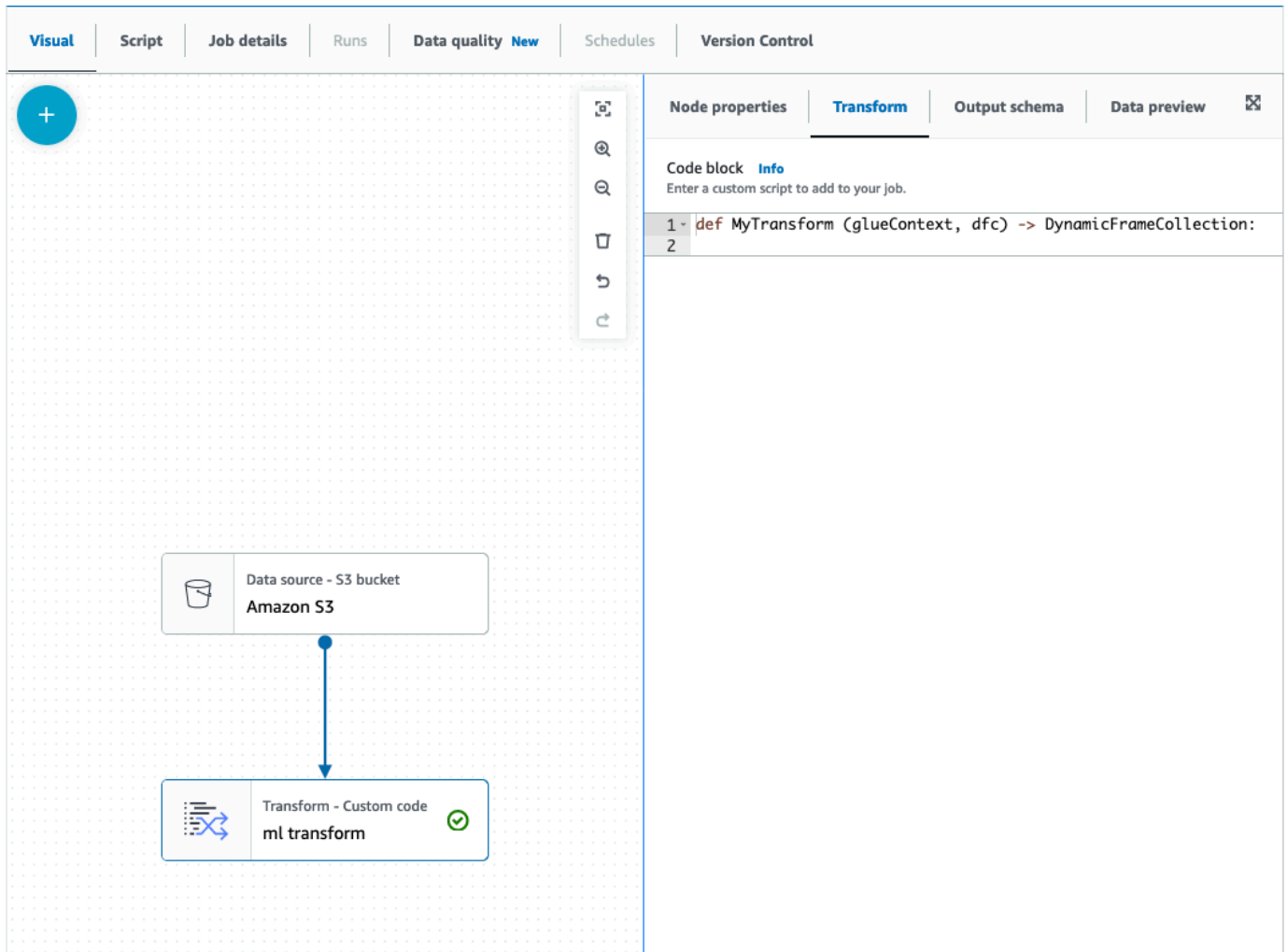
FindMatches 변환 추가

FindMatches 변환을 추가하려면:

1. AWS Glue Studio 작업 편집기에서 시각적 작업 그래프의 왼쪽 상단에 있는 십자 기호를 클릭하여 리소스 패널을 열고 데이터 탭을 선택하여 데이터 소스를 선택합니다. 이는 일치하는지 확인하려는 데이터 소스입니다.



2. 데이터 소스 노드를 선택한 다음 시각적 작업 그래프의 왼쪽 상단에 있는 십자 기호를 클릭하여 리소스 패널을 열고 '사용자 지정 변환'을 검색합니다. 사용자 지정 변환 노드를 선택하여 그래프에 추가합니다. 사용자 지정 변환이 데이터 소스 노드에 링크됩니다. 링크되지 않은 경우 사용자 지정 변환 노드를 클릭하고 노드 속성 탭을 선택한 다음 노드 상위 아래에서 데이터 소스를 선택할 수 있습니다.
3. 시각적 그래프에서 사용자 지정 변환 노드를 클릭한 다음 노드 속성 탭을 선택하고 사용자 지정 변환의 이름을 지정합니다. 시각적 그래프에서 변환 이름을 쉽게 식별할 수 있도록 변환의 이름을 바꾸는 것이 좋습니다.
4. 변환 탭을 선택합니다. 여기에서 코드 블록을 편집할 수 있습니다. 여기에서 FindMatches API를 간접 호출하는 코드를 추가할 수 있습니다.



코드 블록에는 미리 채워진 코드가 포함되어 있어 시작하는 데 도움이 됩니다. 미리 채워진 코드를 아래 템플릿으로 덮어씁니다. 템플릿에는 사용자가 제공할 수 있는 transformId의 자리 표시자가 있습니다.

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    dynf = dfc.select(list(dfc.keys())[0])
    from awsglueml.transforms import FindMatches
    findmatches = FindMatches.apply(frame = dynf, transformId = "<your id>")
    return(DynamicFrameCollection({"FindMatches": findmatches}, glueContext))
```

- 시각적 그래프에서 사용자 지정 변환 노드를 클릭하고 시각적 작업 그래프의 왼쪽 상단에 있는 십자 기호를 클릭하여 리소스 패널을 연 후 '컬렉션에서 선택'을 검색합니다. 컬렉션에는 DynamicFrame 하나만 있으므로 기본 선택을 변경할 필요가 없습니다.

- 계속해서 변환을 추가하거나 결과를 저장할 수 있습니다. 이제 일치 항목 찾기와 같은 추가된 열로 보강되었습니다. 다운스트림 변환에서 이러한 새 열을 참조하려면 변환 출력 스키마에 해당 열을 추가해야 합니다. 가장 쉬운 방법은 데이터 미리 보기 탭을 선택한 다음 스키마 탭에서 '데이터 미리보기 스키마 사용'을 선택하는 것입니다.
- FindMatches를 사용자 지정하려면 'apply' 메서드에 전달할 추가 파라미터를 추가할 수 있습니다. [FindMatches 클래스](#)를 참조하세요.

FindMatches 증분 변환 추가

증분 일치의 경우 프로세스는 다음 차이점을 제외하고 FindMatches 변환 추가와 동일합니다.

- 사용자 지정 변환의 상위 노드 대신, 두 개의 상위 노드가 필요합니다.
- 첫 번째 상위 노드는 데이터 세트여야 합니다.
- 두 번째 상위 노드는 증분 데이터 세트여야 합니다.

템플릿 코드 블록에서 transformId를 사용자의 transformId로 바꿉니다.

```
def MyTransform (glueContext, dfc) -> DynamicFrameCollection:
    dfs = list(dfc.values())
    dynf = dfs[0]
    inc_dynf = dfs[1]
    from awsglueml.transforms import FindIncrementalMatches
    findmatches = FindIncrementalMatches.apply(existingFrame = dynf, incrementalFrame
    = inc_dynf,
                                           transformId = "<your id>")
    return(DynamicFrameCollection({"FindMatches": findmatches}, glueContext))
```

- 선택적 파라미터는 [FindIncrementalMatches 클래스](#)를 참조하세요.

Apache Spark 프로그램을 AWS Glue로 마이그레이션

Apache Spark는 대규모 데이터 세트에서 수행되는 분산 컴퓨팅 워크로드를 위한 오픈 소스 플랫폼입니다. AWS Glue에서는 Spark의 기능을 활용하여 ETL에 최적화된 환경을 제공합니다. Spark 프로그램을 AWS Glue로 마이그레이션하여 기능을 활용할 수 있습니다. AWS Glue에서는 Amazon EMR의 Apache Spark에서 기대하는 것과 동일한 성능 향상을 제공합니다.

Spark 코드 실행

기본 Spark 코드는 기본 제공 AWS Glue 환경에서 실행될 수 있습니다. 스크립트는 대화형 세션에 적합한 워크플로우인 코드를 반복적으로 변경하여 개발되는 경우가 많습니다. 그러나 기존 코드는 AWS Glue 작업에서 실행되기에 더 적합하며 이를 통해 각 스크립트 실행에 대한 로그와 지표를 예약하고 일관되게 가져올 수 있습니다. 콘솔을 통해 기존 스크립트를 업로드하고 편집할 수 있습니다.

1. 스크립트의 소스를 확보합니다. 이 예에서는 Apache Spark 리포지토리의 스크립트 예를 사용합니다. [이진화 메서드 예](#)
2. AWS Glue 콘솔에서 왼쪽 탐색 창을 확장하고 ETL > 작업(Jobs)을 선택합니다.

작업 생성(Create job) 패널에서 Spark 스크립트 편집기(Spark script editor)를 선택합니다. 옵션(Options) 섹션이 나타납니다. 옵션(Options)에서 기존 스크립트 업로드 및 편집(Upload and edit an existing script)을 선택합니다.

파일 업로드(File upload) 섹션이 나타납니다. 파일 업로드(File upload)에서 파일 선택(Choose file)을 클릭합니다. 시스템 파일 선택기가 나타납니다. `binarizer_example.py`를 저장한 위치로 이동하여 선택하고 선택을 확인합니다.

생성(Create) 버튼이 작업 생성(Create job) 패널의 머리글에 나타납니다. 이 버튼을 클릭합니다.

The screenshot shows the AWS Glue Studio interface. At the top, there's a search bar and navigation icons. Below that, the 'Jobs' section is active. The 'Create job' panel is displayed, featuring several options for creating a job:

- Visual with a source and target: Start with a source, ApplyMapping transform, and target.
- Visual with a blank canvas: Author using an interactive visual interface.
- Spark script editor: Write or upload your own Spark code.
- Python Shell script editor: Write or upload your own Python shell script.
- Jupyter Notebook: Write your own code in a Jupyter Notebook for interactive development.

Below these options, the 'Options' section is visible:

- Create a new script with boilerplate code
- Upload and edit an existing script: Choose a local file.

The 'File upload' section shows a 'Choose file' button and a list of files:

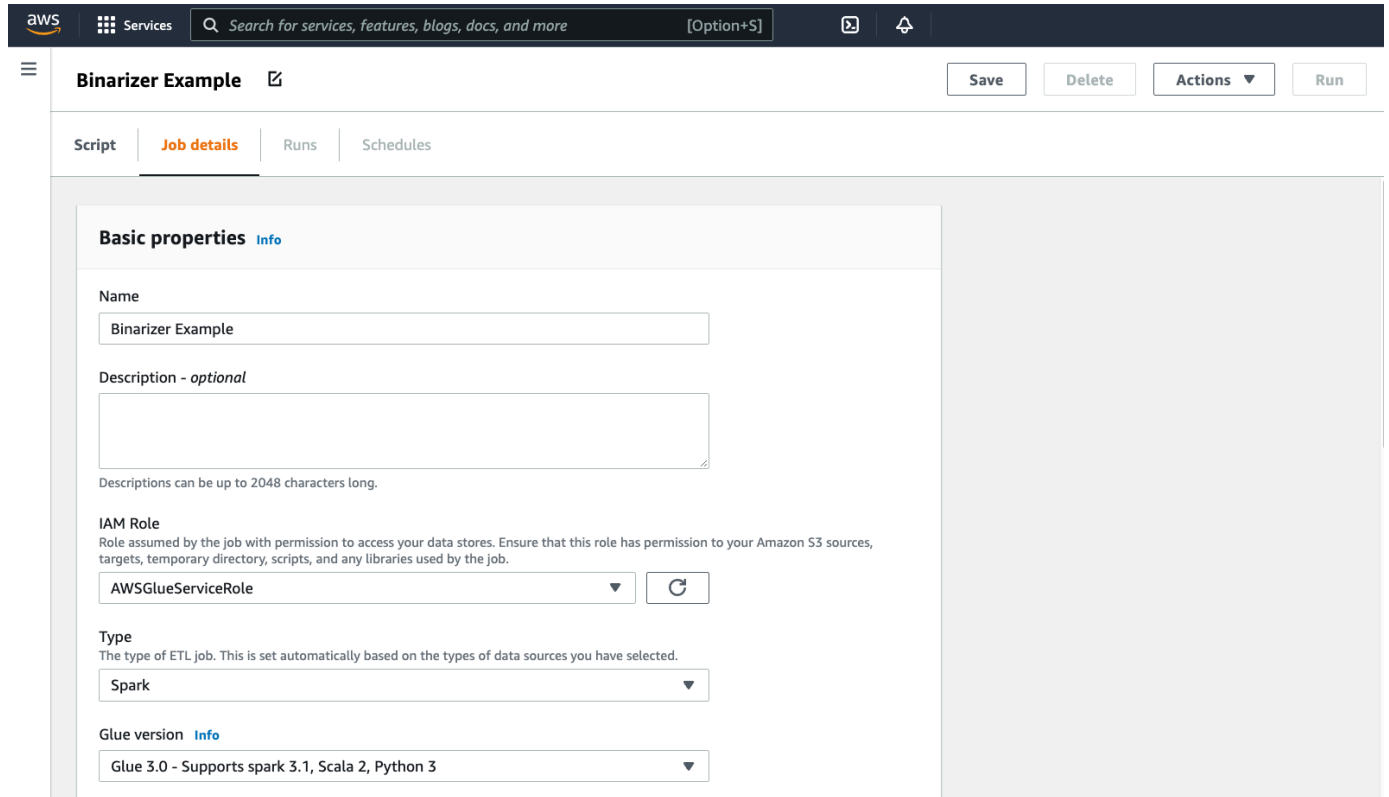
- binarizer_example.py (1.45 KB, June 21, 2022)

A note indicates: 'Limited to Python (*.py, *.py3) and Scala (*.scala) files only.'

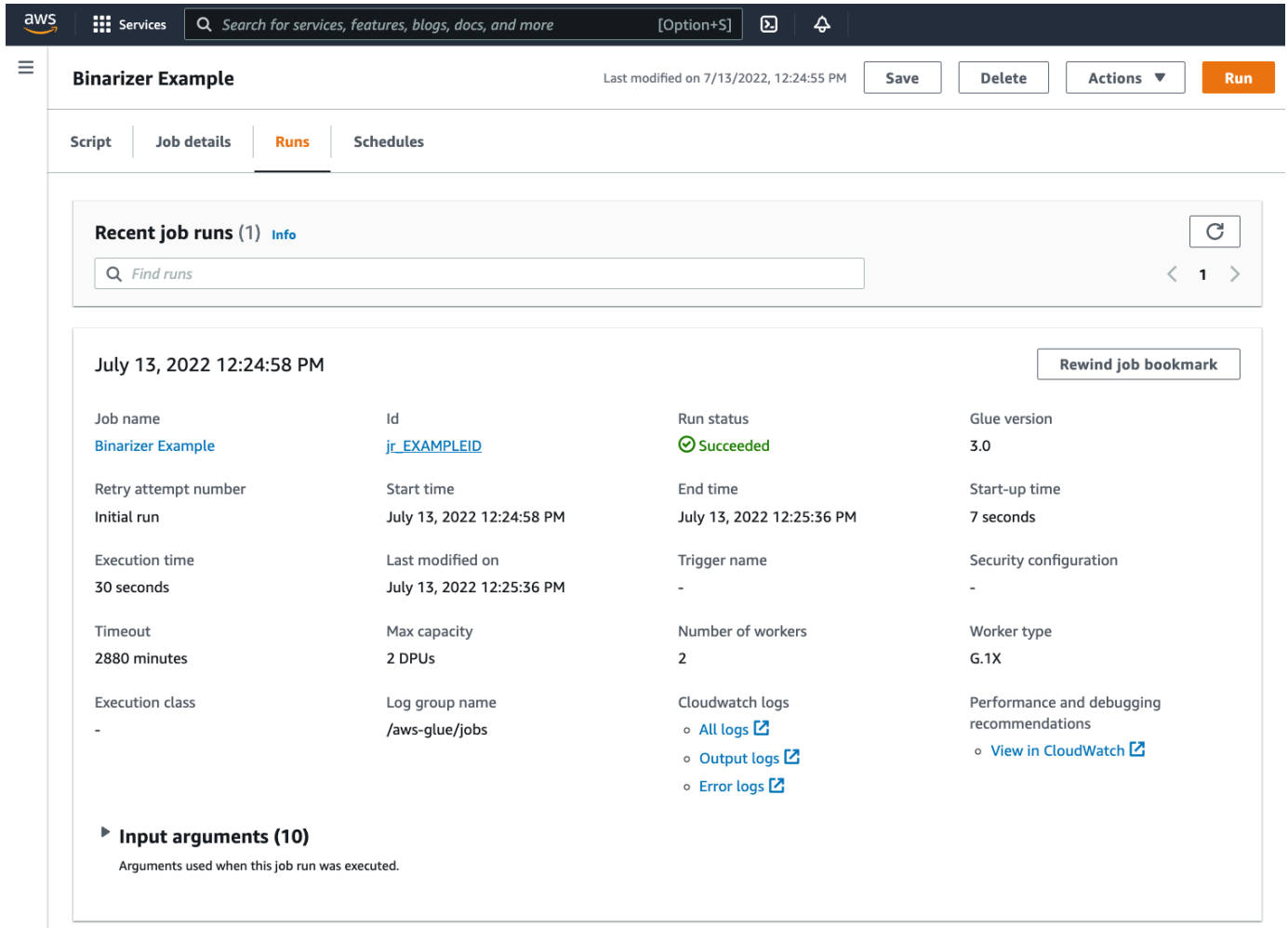
3. 브라우저가 스크립트 편집기로 이동합니다. 머리글에서 Job 세부 정보(Job details) 탭을 클릭합니다. 이름 및 IAM 역할을 설정합니다. AWS Glue IAM 역할에 대한 지침은 [the section called “IAM 권한 설정”](#) 섹션을 참조하세요.

옵션 사항 - 요청된 작업자 수(Requested number of workers)를 2로, 재시도 횟수(Number of retries)를 1로 설정합니다. 이러한 옵션은 프로덕션 작업을 실행할 때 유용하지만 이 옵션을 끄면 기능을 테스트하는 동안 사용 환경이 간소화됩니다.

제목 표시줄에서 저장(Save)을 클릭한 다음 실행(Run)을 클릭합니다.



4. 실행(Runs) 탭으로 이동합니다. 작업 실행에 해당하는 패널이 표시됩니다. 몇 분간 기다리면 페이지가 자동으로 새로 고침되어 실행 상태(Run status) 아래에 성공(Succeeded)이라고 표시됩니다.



- 출력을 검사하여 Spark 스크립트가 의도한 대로 실행되었는지 확인하려고 합니다. 이 Apache Spark 샘플 스크립트는 출력 스트림에 문자열을 기록해야 합니다. 성공적인 작업 실행에 대한 패널의 Cloudwatch 로그(Cloudwatch logs)에서 출력 로그(Output logs)로 이동하면 해당 문자열이 표시됩니다. jr_로 시작하는 ID 레이블에서 생성된 ID인 작업 실행 ID를 기록해 둡니다.

그러면 CloudWatch 콘솔이 열리고 작업 실행 ID에 대한 로그 스트림의 내용으로 필터링된 기본 AWS Glue 로그 그룹 /aws-glue/jobs/output의 내용을 시각화하도록 설정됩니다. 각 작업자는 로그 스트림(Log streams) 아래 행으로 표시된 로그 스트림을 생성하게 됩니다. 한 작업자가 요청된 코드를 실행했어야 합니다. 올바른 작업자를 식별하려면 모든 로그 스트림을 열어야 합니다. 적합한 작업자를 찾으면 다음 이미지에 표시된 것과 같이 스크립트의 출력이 표시되어야 합니다.

The screenshot displays the AWS CloudWatch console interface. At the top, there's a search bar and navigation tabs for 'Log groups'. The main content area shows 'Log events' for a specific job. It includes a filter bar, a 'View as text' checkbox, and a 'Create Metric Filter' button. Below this, a table of log events is shown with columns for 'Timestamp' and 'Message'. One event is expanded to show a table of binarized feature outputs.

Timestamp	Message
2022-07-13T13:27:33.060-07:00	No older events at this moment. Retry
2022-07-13T13:27:33.062-07:00	2022-07-13 20:27:33,058 main WARN JNDI lookup class is not available because...
2022-07-13T13:27:54.066-07:00	2022-07-13 20:27:33,062 main INFO Log4j appears to be running in a Servlet e... Binarizer output with Threshold = 0.500000
2022-07-13T13:28:02.833-07:00	+++++-----+-----+-----+ id feature binarized_feature +-----+ +-----+-----+-----+ +-----+-----+-----+ 0 0.1 0.0 1 0.8 1.0 2 0.2 0.0 +-----+-----+-----+

Spark 프로그램을 마이그레이션하는 데 필요한 일반적인 절차

Spark 버전 지원 평가

AWS Glue 릴리즈 버전에 따라 AWS Glue 작업에 사용할 수 있는 Apache Spark와 Python의 버전이 정해집니다. AWS Glue 버전 및 [the section called “AWS Glue 버전”](#)의 지원 대상을 찾을 수 있습니다. 특정 AWS Glue 기능에 액세스하려면 최신 버전의 Spark와 호환되도록 Spark 프로그램을 업데이트해야 할 수 있습니다.

타사 라이브러리 포함

기존의 많은 Spark 프로그램에는 프라이빗 및 퍼블릭 아티팩트 모두에 대한 종속성이 있습니다. AWS Glue에서는 Scala 작업에 대한 JAR 스타일 종속성뿐만 아니라 Python 작업에 대한 휠 및 소스 퓨어-Python 종속성을 지원합니다.

Python - Python 종속성에 대한 자세한 정보는 [the section called “Python 라이브러리”](#) 섹션을 참조하세요.

공통 Python 종속성은 일반적으로 요청되는 [Pandas](#) 라이브러리를 포함한 AWS Glue 환경에서 제공됩니다. 종속성은 AWS Glue 버전 2.0+에 포함되어 있습니다. 제공된 모듈에 대한 자세한 내용을 알아보려면 [the section called “AWS Glue에서 이미 제공되는 Python 모듈”](#) 섹션을 참조하세요. 기본적으로

포함된 다른 버전의 종속성을 가진 작업을 제공해야 하는 경우 `--additional-python-modules`를 사용할 수 있습니다. 작업 인수에 대한 자세한 정보는 [the section called “작업 파라미터”](#) 섹션을 참조하세요.

`--extra-py-files` 작업 인수를 추가 Python 종속성에 제공할 수 있습니다. Spark 프로그램에서 작업을 마이그레이션하는 경우 이 파라미터는 PySpark의 `--py-files` 플래그와 기능적으로 동일하고 동일한 제한이 적용되므로 좋은 옵션입니다. `--extra-py-files` 파라미터에 대한 자세한 내용을 알아보려면 [the section called “PySpark 기본 기능을 갖춘 Python 파일 포함”](#) 섹션을 참조하세요.

새 작업의 경우 `--additional-python-modules` 작업 인수를 사용하여 Python 종속성을 관리할 수 있습니다. 이 인수를 사용하면 종속성 관리를 보다 철저하게 수행할 수 있습니다. 이 파라미터는 Amazon Linux 2와 호환되는 기본 코드 바인딩이 있는 종속성을 포함하여 Wheel 스타일 종속성을 지원합니다.

Scala

`--extra-jars` 작업 인수를 추가 Scala 종속성에 제공할 수 있습니다. 종속성은 Amazon S3에서 호스팅되어야 하며, 인수 값은 공백 없이 쉼표로 구분된 Amazon S3 경로 목록이어야 합니다. 종속성을 호스팅하고 구성하기 전에 다시 번들링하여 구성을 관리하는 것이 더 쉬울 수 있습니다. AWS Glue JAR 종속성에는 모든 JVM 언어에서 생성될 수 있는 Java 바이트 코드가 포함됩니다. Java와 같은 다른 JVM 언어를 사용하여 사용자 정의 종속성을 작성할 수 있습니다.

데이터 소스 자격 증명 관리

기존 Spark 프로그램에는 데이터 소스에서 데이터를 가져오기 위한 복잡한 구성 또는 사용자 정의 구성이 제공될 수 있습니다. 일반적인 데이터 소스 인증 흐름은 AWS Glue 연결에서 지원됩니다. AWS Glue 연결에 대한 자세한 정보는 [데이터에 연결](#) 섹션을 참조하세요.

AWS Glue 연결은 라이브러리에 대한 메서드 호출 및 AWS 콘솔에서 추가 네트워크 연결(Additional network connection) 설정과 같은 두 가지 기본 방법으로 작업을 다양한 유형의 데이터 저장소에 쉽게 연결할 수 있도록 합니다. 작업 내에서 AWS SDK를 호출하여 연결에서 정보를 검색할 수도 있습니다.

메서드 호출(Method calls) – AWS Glue 연결은 데이터 세트에 대한 정보를 큐레이팅할 수 있는 서비스인 AWS Glue Data Catalog와 밀접히 통합되며, AWS Glue 연결과 상호 작용할 수 있는 메서드가 그것을 반영합니다. 재사용하려는 기존 인증 구성이 있으면 JDBC 연결의 경우 GlueContext의 `extract_jdbc_conf` 메서드를 통해 AWS Glue 연결 구성에 액세스할 수 있습니다. 자세한 정보는 [the section called “extract_jdbc_conf”](#) 섹션을 참조하세요.

콘솔 구성(Console configuration) – AWS Glue 작업은 관련된 AWS Glue 연결을 사용하여 Amazon VPC 서브넷에 대한 연결을 구성할 수 있습니다. 보안 자료를 직접 관리하는 경우 라우팅을 구성하는

AWS 콘솔에서 NETWORK 유형의 추가 네트워크 연결(Additional network connection)을 제공해야 할 수 있습니다. AWS Glue 연결에 대한 자세한 정보는 [the section called “연결”](#) 섹션을 참조하세요.

Spark 프로그램에 사용자 지정 또는 일반적이지 않은 인증 흐름이 있는 경우 보안 자료를 직접 관리해야 할 수 있습니다. AWS Glue 연결이 적합하지 않은 것 같은 경우 Secrets Manager에서 보안 자료를 안전하게 호스팅하고 작업에서 제공되는 boto3 또는 AWS SDK를 통해 해당 자료에 액세스 할 수 있습니다.

Apache Spark 구성

복잡한 마이그레이션은 Spark 구성을 변경하여 워크로드를 수용하는 경우가 많습니다. 최신 버전의 Apache Spark를 사용하면 런타임 구성을 SparkSession으로 설정할 수 있습니다. AWS Glue 3.0+ 작업에 런타임 구성을 설정하기 위해 수정할 수 있는 SparkSession이 제공됩니다. [Apache Spark 구성](#). Spark 튜닝은 복잡하고, AWS Glue에서 모든 Spark 구성 설정에 대한 지원을 보장하지는 않습니다. 마이그레이션에 상당한 Spark 수준의 구성이 필요한 경우 지원 팀에 문의하세요.

사용자 지정 구성 설정

마이그레이션된 Spark 프로그램은 사용자 지정 구성을 사용하도록 설계될 수 있습니다. AWS Glue에서는 작업 인수를 통해 작업 및 작업 실행 수준에서 구성을 설정할 수 있습니다. 작업 인수에 대한 자세한 정보는 [the section called “작업 파라미터”](#) 섹션을 참조하세요. 라이브러리를 통해 작업 컨텍스트 내에서 작업 인수에 액세스할 수 있습니다. AWS Glue에서는 유틸리티 함수를 제공하여 작업에 설정된 인수와 작업 실행 시 설정된 인수 간에 일관된 보기를 유지합니다. Python의 [the section called “getResolvedOptions 옵션”](#) 섹션 및 Scala의 [the section called “GlueArgParser”](#) 섹션을 참조하세요.

Java 코드 마이그레이션

[the section called “타사 라이브러리”](#)에서 설명한 대로 종속성에는 Java 또는 Scala와 같은 JVM 언어로 생성된 클래스가 포함될 수 있습니다. 종속성에는 main 메서드가 포함될 수 있습니다. 종속성의 main 메서드를 AWS Glue Scala 작업에 대한 진입점으로 사용할 수 있습니다. 이를 통해 Java의 main 메서드를 작성하거나 자체 라이브러리 표준에 맞게 패키징된 main 메서드를 재사용할 수 있습니다.

종속성에서 main 메서드를 사용하려면 다음을 수행합니다. 기본 GlueApp 객체를 제공하는 편집 창의 내용을 삭제합니다. 종속성에 있는 클래스의 정규화된 이름을 키 --class를 통해 작업 인수로 제공합니다. 그러면 작업 실행을 트리거할 수 있습니다.

인수 AWS Glue의 순서나 구조가 main 메서드에 전달되도록 구성할 수 없습니다. 기존 코드에서 AWS Glue에 설정된 구성을 읽어야 하는 경우 이전 코드와 호환되지 않을 수 있습니다. getResolvedOptions를 사용하는 경우 이 메서드를 호출하기에 적합한 장소도 없을 것입니다. AWS Glue에 의해 생성된 기본 메서드에서 직접 종속성을 호출하는 것이 좋습니다. 이에 대한 예는 다음 AWS Glue ETL 스트립트에 표시됩니다.

```
import com.amazonaws.services.glue.util.GlueArgParser

object GlueApp {
  def main(sysArgs: Array[String]) {
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)

    // Invoke static method from JAR. Pass some sample arguments as a String[], one
    // defined inline and one taken from the job arguments, using getResolvedOptions
    com.mycompany.myproject.MyClass.myStaticPublicMethod(Array("string parameter1",
    args("JOB_NAME")))

    // Alternatively, invoke a non-static public method.
    (new com.mycompany.myproject.MyClass).someMethod()
  }
}
```

AWS Glue에서 Ray 작업 사용

이 섹션에서는 AWS Glue for Ray 작업 사용에 대한 정보를 제공합니다. AWS Glue for Ray 스크립트 작성에 대한 자세한 내용은 [the section called “AWS Glue for Ray”](#) 섹션을 참조하세요.

주제

- [AWS Glue for Ray 시작하기](#)
- [지원되는 Ray 런타임 환경](#)
- [Ray 작업에서 작업자 고려](#)
- [Ray 작업에서 작업 파라미터 사용](#)
- [지표를 통한 Ray 작업 모니터링](#)

AWS Glue for Ray 시작하기

AWS Glue for Ray를 사용하려면 AWS Glue for Spark를 사용할 때와 동일한 AWS Glue 작업 및 대화형 세션을 사용합니다. AWS Glue 작업은 동일한 스크립트를 반복적으로 실행하도록 설계된 반면, 대화형 세션은 프로비저닝된 동일한 리소스에 대해 순차적으로 코드 조각을 실행할 수 있도록 설계되었습니다.

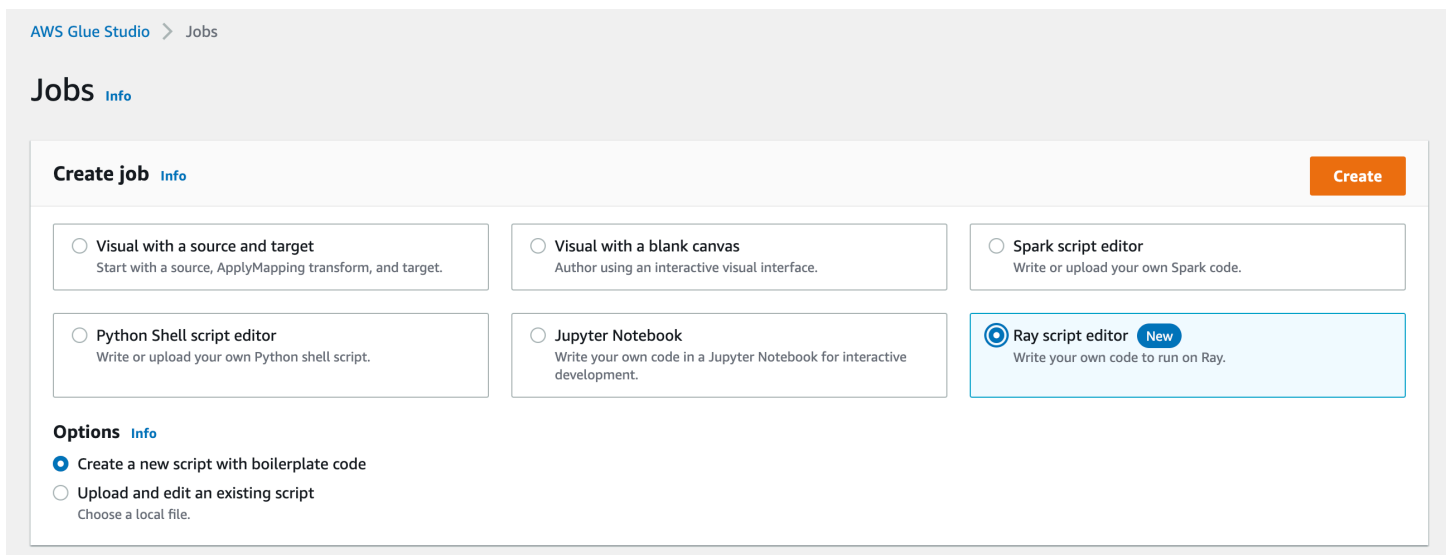
AWS Glue ETL과 Ray는 기본적으로 다르므로 스크립트에서 다른 도구, 기능 및 구성에 액세스할 수 있습니다. AWS Glue에서 관리하는 새로운 계산 프레임워크인 Ray는 아키텍처가 다르며 해당 기능을 설명하기 위해 다양한 어휘를 사용합니다. 자세한 내용은 Ray 설명서의 [아키텍처 백서](#)를 참조하세요.

Note

AWS Glue for Ray는 미국 동부(버지니아 북부), 미국 동부(오하이오), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드)에서 사용 가능합니다.

AWS Glue Studio 콘솔에서 Ray 작업

AWS Glue Studio에서 작업을 생성할 때 AWS Glue Studio 콘솔의 작업 페이지에서 새 옵션(Ray 스크립트 편집기)을 선택할 수 있습니다. 콘솔에서 Ray 작업을 생성하려면 이 옵션을 선택합니다. 작업에 대한 자세한 내용과 사용 방법은 [AWS Glue Studio를 사용하여 시각적 ETL 작업 구축](#) 섹션을 참조하세요.



AWS CLI 및 SDK에서 Ray 작업

AWS CLI에서 Ray 작업은 다른 작업과 동일한 SDK 작업 및 파라미터를 사용합니다. AWS Glue for Ray에는 특정 파라미터에 대한 새로운 값이 도입되었습니다. 작업 API에 대한 자세한 내용은 [the section called “작업”](#) 섹션을 참조하세요.

지원되는 Ray 런타임 환경

Spark 작업에서는 GlueVersion에서 AWS Glue for Spark 작업에 대해 사용할 수 있는 Apache Spark 및 Python 버전을 결정합니다. Python의 버전으로 Spark 유형의 작업에 대해 지원되는 버전을 확인할 수 있습니다. Ray 런타임 환경은 이렇게 구성되지 않습니다.

Ray 작업의 경우 GlueVersion을 4.0 이상으로 설정해야 합니다. 하지만 Ray 작업에서 사용할 수 있는 Ray, Python 및 추가 라이브러리 버전은 작업 정의의 Runtime 필드에 의해 결정됩니다.

Ray 2.4 런타임 환경은 출시 후 최소 6개월 동안 사용할 수 있습니다. Ray가 빠르게 진화함에 따라 향후 런타임 환경 릴리스를 통해 Ray 업데이트 및 개선 사항을 통합할 수 있습니다.

유효값: Ray 2.4

런타임 값	Ray 및 Python 버전
Ray 2.4(AWS Glue 4.0 이상)	Ray 2.4.0 Python 3.9

추가 정보

- Ray 릴리스에서 AWS Glue를 함께 제공하는 릴리스 정보는 [the section called “AWS Glue 버전”](#) 섹션을 참조하세요.
- 런타임 환경에서 제공되는 Python 라이브러리는 [the section called “Ray 작업과 함께 제공되는 모듈”](#) 섹션을 참조하세요.

Ray 작업에서 작업자 고려

AWS Glue에서는 새로운 Graviton 기반 EC2 작업자 유형(Ray 작업에만 사용 가능)에서 Ray 작업을 실행합니다. Ray의 설계 목표였던 워크로드에 맞게 이러한 작업자를 적절하게 프로비저닝하기 위해 대부분의 작업자와는 다른 비율로, 메모리 리소스에 대한 컴퓨팅 리소스를 제공합니다. 이러한 리소스를 고려하기 위해 표준 데이터 처리 장치(DPU) 대신, 메모리에 최적화된 데이터 처리 장치(M-DPU)를 사용합니다.

- 1개의 M-DPU는 vCPU 4개와 32GB의 메모리에 해당합니다.
- 1개의 DPU가 vCPU 4개와 16GB 메모리에 해당합니다. AWS Glue에서 Spark 작업 및 해당 작업자로 리소스를 처리하려는 경우 DPU가 사용됩니다.

Ray 작업은 현재 하나의 작업자 유형(Z.2X)에 액세스할 수 있습니다. Z.2X 작업자는 2개의 M-DPU(vCPU 8개, 메모리 64GB)에 매핑되며 128GB의 디스크 공간을 보유하고 있습니다. Z.2X 시스템은 8개의 Ray 작업자(vCPU당 한 개)를 제공합니다.

계정에서 동시에 사용할 수 있는 M-DPU 수에는 서비스 할당량이 적용됩니다. AWS Glue 계정 제한에 대한 자세한 내용은 [AWS Glue 엔드포인트 및 할당량](#)을 참조하세요.

작업 정의에서 `--number-of-workers` (`NumberOfWorkers`)를 사용하여 Ray 작업에 사용할 수 있는 워커 노드 수를 지정합니다. 작업 API에서 Ray 값에 대한 자세한 내용은 [the section called “작업”](#) 섹션을 참조하세요.

`--min-workers` 작업 파라미터를 사용하여 Ray 작업에서 할당해야 하는 최소 작업자 수를 추가로 지정할 수 있습니다. 작업 파라미터에 대한 자세한 내용을 알아보려면 [the section called “레퍼런스”](#) 섹션을 참조하세요.

Ray 작업에서 작업 파라미터 사용

AWS Glue for Spark 작업과 동일한 방식으로 AWS Glue Ray 작업에 대한 인수를 설정합니다. AWS Glue API에 대한 자세한 내용은 [the section called “작업”](#)을 참조하십시오. 이 참조에 나열된 다양한 인수로 AWS Glue Ray 작업을 구성할 수 있습니다. 내 인수를 제공할 수도 있습니다.

Job Parameters(작업 파라미터) 제목 아래의 Job details(작업 세부 정보) 탭에서 콘솔을 통해 작업을 구성할 수 있습니다. 작업에서 `DefaultArguments`를 설정하거나 작업 실행에서 `Arguments`를 설정하여 AWS CLI를 통해 작업을 구성할 수도 있습니다. 기본 인수 및 작업 파라미터는 여러 번 실행해도 작업과 함께 유지됩니다.

예를 들어 다음은 특수 파라미터를 설정하기 위해 `--arguments`를 사용하여 작업을 실행하는 구문입니다.

```
$ aws glue start-job-run --job-name "CSV to CSV" --arguments='--scriptLocation="s3://my_glue/libraries/test_lib.py",--test-environment="true"'
```

인수를 설정한 후에는 환경 변수를 통해 Ray 작업 내에서 작업 파라미터에 액세스할 수 있습니다. 그러면 각 실행에 대한 작업을 구성할 수 있습니다. 환경 변수의 이름은 `--` 접두사가 없는 작업 인수 이름입니다.

예를 들어 이전 예제에서 변수 이름은 `scriptLocation` 및 `test-environment`입니다. 그런 다음 표준 라이브러리(`test_environment = os.environ.get('test-environment')`)에서 사용할 수 있는 메서드를 통해 인수를 검색합니다. Python으로 환경 변수에 액세스하는 방법에 대한 자세한 내용은 Python 설명서의 [os 모듈](#)을 참조하세요.

Ray 작업이 로그를 생성하는 방법 구성

기본적으로 Ray 작업은 CloudWatch와 Amazon S3로 전송되는 로그와 지표를 생성합니다. `--logging-configuration` 파라미터를 사용하여 로그 생성 방식을 변경할 수 있으며, 현재 이를 사용하여 Ray 작업에서 다양한 유형의 로그를 생성하지 못하게 할 수 있습니다. 이 파라미터는 변경하려는 로그/동작에 해당하는 키를 갖는 JSON 객체를 사용하며, 다음 키를 지원합니다.

- CLOUDWATCH_METRICS - 작업 상태를 시각화하는 데 사용할 수 있는 CloudWatch 지표 시리즈를 구성합니다. 지표에 대한 자세한 정보는 [the section called “Ray 작업 지표”](#) 섹션을 참조하세요.
- CLOUDWATCH_LOGS - 작업 실행 상태에 대한 Ray 애플리케이션 수준 세부 정보를 제공하는 CloudWatch 로그를 구성합니다. 로그에 대한 자세한 내용은 [the section called “Ray 오류 해결”](#) 섹션을 참조하세요.
- S3 - AWS Glue가 Amazon S3에 작성하는 내용을 구성합니다. 주로 CloudWatch 로그와 유사한 정보이지만 로그 스트림이 아닌 파일로 작성합니다.

Ray 로깅 동작을 비활성화하려면 `{"IS_ENABLED\":"False\"}` 값을 제공합니다. 예를 들어, CloudWatch 지표와 CloudWatch 로그를 비활성화하려면 다음 구성을 제공합니다.

```
--logging_configuration": "{\"CLOUDWATCH_METRICS\": {\"IS_ENABLED\": \"False\"},
  \"CLOUDWATCH_LOGS\": {\"IS_ENABLED\": \"False\"}}"
```

레퍼런스

Ray 작업은 Ray 작업 및 작업 실행에 대한 스크립트 환경을 설정하는 데 사용할 수 있는 다음과 같은 인수 이름을 인식합니다.

- `--logging_configuration` - Ray 작업에서 생성되는 다양한 로그의 생성을 중지하는 데 사용됩니다. 이러한 로그는 모든 Ray 작업에서 기본적으로 생성됩니다. 형식은 문자열 이스케이프 JSON 객체입니다. 자세한 내용은 [the section called “Ray 작업이 로그를 생성하는 방법 구성”](#) 단원을 참조하십시오.
- `--min-workers` - Ray 작업에 할당되는 작업자 노드의 최소 수입니다. 작업자 노드는 가상 CPU당 하나씩 여러 복제본을 실행할 수 있습니다. 형식: 정수 최소값: 0 최대값: 작업 정의의 `--number-of-workers` (NumberOfWorkers)에 지정된 값입니다. 워커 노드 고려 사항에 대한 자세한 내용은 [the section called “Ray 작업에서 작업자 고려”](#) 섹션을 참조하세요.
- `--object_spilling_config` - AWS Glue for Ray는 Ray의 객체 스토어로 사용 가능한 공간을 확장하는 방법으로 Amazon S3를 사용할 수 있도록 지원합니다. 이 동작을 활성화하려면 이 파라미터와 함께 객체 유출 JSON 구성 객체를 Ray에 제공할 수 있습니다. Ray 객체 유출 구성에 대한 자세한 내용은 Ray 설명서의 [Object Spilling](#)을 참조하세요. 형식: JSON 객체.

AWS Glue for Ray는 디스크로 유출 또는 Amazon S3로 한 번에 유출하는 것만 지원합니다. 이러한 제한을 준수하는 유출에 대해 여러 위치를 제공할 수 있습니다. Amazon S3로 유출하는 경우 이 버킷의 작업에 IAM 권한도 추가해야 합니다.

CLI에서 JSON 객체를 구성으로 제공할 때는 JSON 객체를 문자열로 이스케이프 처리하여 문자열로 제공해야 합니다. 예를 들어 하나의 Amazon S3 경로로 유출하기 위한 문자열 값은 `{"type\": \"smart_open\", \"params\": {\"uri\": \"s3path\"}}`과 같습니다. AWS Glue Studio에서는 이 파라미터를 추가 형식 지정 없이 JSON 객체로 제공합니다.

- `--object_store_memory_head` - Ray 헤드 노드의 Plasma 객체 스토어에 할당된 메모리입니다. 이 인스턴스는 작업자 복제본뿐만 아니라 클러스터 관리 서비스를 실행합니다. 값은 부팅 후 인스턴스에서 사용 가능한 메모리의 백분율을 나타냅니다. 메모리를 많이 사용하는 워크로드를 조정하는데 이 파라미터를 사용합니다. 기본값은 대부분의 사용 사례에 적합합니다. 형식: 양의 정수 최소값: 1 최대값: 100

Plasma에 대한 자세한 내용은 Ray 설명서의 [Plasma 메모리 내 객체 스토어](#)를 참조하세요.

- `--object_store_memory_worker` - Ray 작업자 노드의 Plasma 객체 스토어에 할당된 메모리입니다. 이러한 인스턴스는 작업자 복제본만 실행합니다. 값은 부팅 후 인스턴스에서 사용 가능한 메모리의 백분율을 나타냅니다. 이 파라미터는 메모리를 많이 사용하는 워크로드를 조정하는데 사용됩니다. 기본값은 대부분의 사용 사례에 허용됩니다. 형식: 양의 정수 최소값: 1 최대값: 100

Plasma에 대한 자세한 내용은 Ray 설명서의 [Plasma 메모리 내 객체 스토어](#)를 참조하세요.

- `--pip-install` - 설치할 Python 패키지 세트입니다. 이 인수를 사용하여 PyPI에서 패키지를 설치할 수 있습니다. 형식: 쉼표로 구분된 목록.

PyPI 패키지 항목은 `package==version` 형식(대상 패키지의 PyPI 이름 및 버전)입니다. 항목은 단일 등호 `=`가 아닌 `==`와 같은 Python 버전 매칭을 사용하여 패키지와 버전을 매칭합니다. 다른 버전 일치 연산자도 있습니다. 자세한 내용은 Python 웹사이트의 [PEP 440](#)을 참조하세요. `--s3-py-modules`에서 사용자 지정 모듈을 제공할 수도 있습니다.

- `--s3-py-modules` - Python 모듈 배포를 호스팅하는 Amazon S3 경로 세트입니다. 형식: 쉼표로 구분된 목록.

이를 사용하여 Ray 작업에 자체 모듈을 배포할 수 있습니다. `--pip-install`에서 PyPI의 모듈을 제공할 수도 있습니다. AWS Glue ETL과 달리, 사용자 지정 배포는 pip를 통해 설정되지 않고 배포를 위해 Ray로 전달됩니다. 자세한 내용은 [the section called “Ray 작업을 위한 추가 Python 모듈”](#) 단원을 참조하십시오.

- `--working-dir` - Amazon S3에 호스팅된.zip 파일의 경로로, Ray 작업을 실행하는 모든 노드에 배포할 파일을 포함합니다. 형식: 문자열. 자세한 내용은 [the section called “Ray 작업에 파일 제공”](#) 단원을 참조하십시오.

지표를 통한 Ray 작업 모니터링

AWS Glue Studio 및 Amazon CloudWatch를 사용하여 Ray 작업을 모니터링할 수 있습니다. CloudWatch는 Ray를 통해 AWS Glue에서 원시 데이터를 수집하고 처리합니다. 이러한 데이터는 분석에 사용할 수 있습니다. 이러한 지표는 AWS Glue Studio 콘솔에서 시각화되므로 작업을 실행하면서 모니터링할 수 있습니다.

AWS Glue 모니터링 방법에 대한 일반적인 개요는 [the section called “CloudWatch 지표 사용”](#) 섹션을 참조하세요. AWS Glue에서 게시한 CloudWatch 지표를 사용하는 방법에 대한 일반적인 개요는 [the section called “CloudWatch를 사용하여 모니터링”](#) 섹션을 참조하세요.

AWS Glue 콘솔에서 Ray 작업 모니터링

작업 실행에 대한 세부 정보 페이지의 실행 세부 정보 섹션 아래에서 사용 가능한 작업 지표를 시각화하는 사전 구축된 집계된 그래프를 볼 수 있습니다. AWS Glue Studio에서는 모든 작업 실행 시 작업 지표를 Cloud Watch에 전송합니다. 이를 통해 클러스터 및 작업의 프로파일을 구축하고 각 노드에 대한 세부 정보에 액세스할 수 있습니다.

사용 가능한 지표에 대한 자세한 내용은 [the section called “Ray 작업 실행에 대한 Amazon CloudWatch 지표 보기”](#) 섹션을 참조하십시오.

CloudWatch의 Ray 작업 지표 개요

CloudWatch에서 세부 모니터링이 활성화되면 Ray 지표를 게시합니다. 지표는 Glue/Ray CloudWatch 네임스페이스에 게시됩니다.

- 인스턴스 지표

작업에 할당된 인스턴스의 CPU, 메모리 및 디스크 사용률에 대한 지표를 게시합니다. 이러한 지표는 ExecutorId, ExecutorType, host와 같은 기능으로 식별됩니다. 이러한 지표는 표준 Linux CloudWatch 에이전트 지표의 하위 세트입니다. CloudWatch 설명서에서 지표 이름 및 기능에 대한 정보를 확인할 수 있습니다. 자세한 내용은 [CloudWatch 에이전트가 수집하는 지표](#)를 참조하세요.

- Ray 클러스터 지표

스크립트를 실행하는 Ray 프로세스에서 이 네임스페이스로 지표를 전달한 후 사용자에게 가장 중요한 지표를 제공합니다. 사용 가능한 지표는 Ray 버전에 따라 다를 수 있습니다. 작업이 실행 중인 Ray 버전에 대한 자세한 내용은 [the section called “AWS Glue 버전”](#) 섹션을 참조하세요.

Ray는 인스턴스 수준에서 지표를 수집합니다. 또한 작업 및 클러스터에 대한 지표도 제공합니다. Ray의 기본 지표 전략에 대한 자세한 내용은 Ray 설명서의 [Metrics](#)를 참조하세요.

Note

AWS Glue ETL 작업에만 사용되는 Glue/Job Metrics/ 네임스페이스에는 Ray 지표를 게시하지 않습니다.

AWS Glue에서 Python 셸 작업에 대한 작업 속성 구성

AWS Glue에서 Python 셸 작업을 사용하여 Python 스크립트를 셸로 실행할 수 있습니다. Python 셸 작업을 사용하면 Python 3.6 또는 Python 3.9와 호환되는 스크립트를 실행할 수 있습니다.

주제

- [제한 사항](#)
- [Python 셸 작업에 대한 작업 속성 정의](#)
- [Python 셸 작업에 지원되는 라이브러리](#)
- [자체 Python 라이브러리 제공](#)
- [AWS Glue에서 Python 셸 작업과 함께 AWS CloudFormation 사용](#)

제한 사항

Python 셸 작업의 다음 제한 사항에 유의합니다.

- Python 셸 작업에는 작업 북마크를 사용할 수 없습니다.
- Python 3.9 이상에서는 어떤 Python 라이브러리도 .egg 파일로 패키징할 수 없습니다. 대신 .whl를 사용합니다.
- S3 데이터의 임시 복사본에 대한 제한 사항 때문에 이 `--extra-files` 옵션을 사용할 수 없습니다.

Python 셸 작업에 대한 작업 속성 정의

이 섹션에서는 AWS Glue Studio에서 또는 AWS CLI를 사용하여 작업 속성을 정의하는 것을 설명합니다.

AWS Glue Studio

AWS Glue Studio에서 Python 셸 작업을 정의할 때 다음 속성 중 일부를 입력합니다.

IAM 역할

사용된 리소스에 대한 권한 부여에 사용되는 AWS Identity and Access Management(IAM) 역할을 지정하여 작업을 실행하고 데이터 스토어에 액세스합니다. AWS Glue 작업을 실행하는 권한에 대한 자세한 내용은 [AWS Glue의 ID 및 액세스 관리](#) 단원을 참조하십시오.

유형

Python shell(Python 셸)을 선택하여 `pythonshell` 작업 명령으로 Python 스크립트를 실행합니다.

Python 버전

Python 버전을 선택합니다. 기본값은 Python 3.9입니다. 유효한 버전은 Python 3.6과 Python 3.9입니다.

공통 분석 라이브러리 로드(권장)

Python 셸에 Python 3.9용 공통 라이브러리를 포함하려면 이 옵션을 선택합니다.

라이브러리가 사용자 지정 라이브러리가거나 사전 설치된 라이브러리와 충돌하는 경우 공통 라이브러리를 설치하지 않을 수 있습니다. 하지만 공통 라이브러리 외에 추가 라이브러리를 설치할 수 있습니다.

이 옵션을 선택하는 경우 `library-set` 옵션이 `analytics`로 설정됩니다. 이 옵션을 선택 취소하는 경우 `library-set` 옵션이 `none`으로 설정됩니다.

스크립트 파일 이름 및 스크립트 경로

스크립트의 코드는 작업 절차 논리를 정의합니다. Amazon Simple Storage Service(Amazon S3)에서 스크립트 이름과 위치를 제공합니다. 경로의 스크립트 디렉터리와 동일한 이름의 파일이 없다는 것을 확인합니다. 스크립트 사용에 대한 자세한 내용은 [AWS Glue 프로그래밍 안내서](#) 단원을 참조하십시오.

Script

스크립트의 코드는 작업 절차 논리를 정의합니다. Python 3.6 또는 Python 3.9에서 스크립트를 코딩할 수 있습니다. AWS Glue Studio에서 스크립트를 편집할 수 있습니다.

데이터 처리 단위

이 작업이 실행될 때 할당할 수 있는 AWS Glue 데이터 처리 단위(DPU)의 최대 수입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#)을 참조하십시오.

값을 0.0625 또는 1로 설정할 수 있습니다. 기본값은 0.0625입니다. 어느 경우든 인스턴스의 로컬 디스크는 20GB가 됩니다.

CLI

다음 예와 같이 AWS CLI를 사용하여 [Python 셸(Python shell)] 작업을 생성할 수도 있습니다.

```
aws glue create-job --name python-job-cli --role Glue_DefaultRole
  --command '{"Name" : "pythonshell", "PythonVersion": "3.9", "ScriptLocation" :
"s3://DOC-EXAMPLE-BUCKET/scriptname.py"}'
  --max-capacity 0.0625
```

Note

--glue-version 파라미터는 AWS Glue 셸 작업에 적용되지 않으므로 AWS Glue 버전을 지정할 필요가 없습니다. 지정된 버전은 모두 무시됩니다.

AWS CLI를 사용하여 생성된 작업은 Python 3으로 기본 설정됩니다. 유효한 Python 버전은 3(3.6에 해당)과 3.9입니다. Python 3.6을 지정하려면 이 튜플을 --command 파라미터 "PythonVersion": "3"에 추가합니다.

Python 3.9를 지정하려면 이 튜플을 --command 파라미터 "PythonVersion": "3.9"에 추가합니다.

Python 셸 작업에서 사용하는 최대 용량을 설정하려면 --max-capacity 파라미터를 제공합니다. Python 셸 작업에서 --allocated-capacity 파라미터는 사용할 수 없습니다.

Python 셸 작업에 지원되는 라이브러리

Python 3.9를 사용하는 Python 셸에서 필요에 따라 사전 패키징된 라이브러리 세트를 사용하도록 라이브러리 세트를 선택할 수 있습니다. library-set 옵션을 사용하여 라이브러리 세트를 선택할 수 있습니다. 유효한 값은 analytics 및 none입니다.

Python 셸 작업의 실행 환경은 다음 라이브러리를 지원합니다.

Python 버전	Python 3.6	Python 3.9	
라이브러리 세트	해당 사항 없음	분석	none
avro		1.11.0	
awscli	116.242	1.23.5	1.23.5

Python 버전	Python 3.6	Python 3.9	
awswrangler		2.15.1	
botocore	1.12.232	1.24.21	1.23.5
boto3	1.9.203	1.21.21	
elasticsearch		8.2.0	
numpy	1.16.2	1.22.3	
pandas	0.24.2	1.4.2	
psycopg2		2.9.3	
pyathena		2.5.3	
PyGreSQL	5.0.6		
PyMySQL		1.0.2	
pyodbc		4.0.32	
pyorc		0.6.0	
redshift-connector		2.0.907	
요청	2.22.0	2.27.1	
scikit-learn	0.20.3	1.0.2	
scipy	1.2.1	1.8.0	
SQLAlchemy		1.4.36	
s3fs		2022.3.0	

NumPy 라이브러리는 공학용 컴퓨팅의 Python 셸 작업에서 사용할 수 있습니다. 자세한 내용은 [NumPy](#)를 참조하십시오. 다음 예제는 Python 셸 작업에 사용할 수 있는 NumPy 스크립트를 보여줍니다. 이 스크립트는 "Hello world" 및 몇 가지 수학적 계산 결과를 프린트합니다.

```
import numpy as np
print("Hello world")

a = np.array([20,30,40,50])
print(a)

b = np.arange( 4 )

print(b)

c = a-b

print(c)

d = b**2

print(d)
```

자체 Python 라이브러리 제공

PIP 사용

Python 3.9를 사용하는 Python 셸을 사용하면 작업 수준에서 추가 Python 모듈 또는 다른 버전을 제공할 수도 있습니다. `--additional-python-modules` 옵션을 쉼표로 구분된 Python 모듈 목록과 함께 사용하여 새 모듈을 추가하거나 기존 모듈의 버전을 변경할 수 있습니다. Python 셸 작업을 사용할 때는 이 파라미터로 Amazon S3에 호스팅된 사용자 지정 Python 모듈을 제공할 수 없습니다.

예를 들어 새 `scikit-learn` 모듈을 업데이트하거나 추가하려면 `--additional-python-modules`, `"scikit-learn==0.21.3"` 키 및 값을 사용합니다.

AWS Glue는 Python 패키지 설치 프로그램(`pip3`)을 사용하여 추가 모듈을 설치합니다. `--additional-python-modules` 값 내에 추가 `pip3` 옵션을 전달할 수 있습니다. 예를 들면 `"scikit-learn==0.21.3 -i https://pypi.python.org/simple/"`입니다. `pip3`의 비호환성 또는 제한 사항이 적용됩니다.

Note

향후에 비호환성을 방지하려면 Python 3.9용으로 빌드된 라이브러리를 사용하는 것이 좋습니다.

Egg 또는 Whl 파일 사용

이미 하나 이상의 .egg 또는 .whl 파일로 패키징된 하나 이상의 Python 라이브러리가 있을 수 있습니다. 그렇다면 다음 예와 같이 AWS Command Line Interface(AWS CLI)를 사용하여 "--extra-py-files" 플래그 하에서 작업에 이를 지정할 수 있습니다.

```
aws glue create-job --name python-redshift-test-cli --role role --command '{"Name" :
"pythonshell", "ScriptLocation" : "s3://MyBucket/python/library/redshift_test.py"}'
--connections Connections=connection-name --default-arguments '{"--extra-py-
files" : ["s3://DOC-EXAMPLE-BUCKET/EGG-FILE", "s3://DOC-EXAMPLE-BUCKET/WHEEL-FILE"]}'
```

Python 라이브러리에서 .egg 파일 또는 .whl 파일을 생성하는 방법을 모르는 경우 다음 단계를 따르십시오. 이 예제는 macOS, Linux 및 Windows Subsystem for Linux(WSL)에 적용됩니다.

Python .egg 또는 .whl 파일을 만들려면

1. Virtual Private Cloud(VPC)에서 Amazon Redshift 클러스터를 생성하고 일부 데이터를 테이블에 추가합니다.
2. 클러스터를 만드는 데 사용한 VPC-SecurityGroup-Subnet 조합에 대해 AWS Glue 연결을 만듭니다. 연결이 성공인지 테스트합니다.
3. redshift_example이라는 디렉터리를 생성하고 setup.py라는 파일을 생성합니다. setup.py에 다음 코드를 붙여넣습니다.

```
from setuptools import setup

setup(
    name="redshift_module",
    version="0.1",
    packages=['redshift_module']
)
```

4. redshift_example 디렉터리에 redshift_module 디렉터리를 만듭니다. redshift_module 디렉터리에서 __init__.py 및 pygresql_redshift_common.py라는 파일을 만듭니다.
5. __init__.py 파일은 비워둡니다. pygresql_redshift_common.py에 다음 코드를 붙여넣습니다. *port*, *db_name*, *user* 및 *password_for_user*를 Amazon Redshift 클러스터에 해당하는 세부 정보로 바꿉니다. *table_name*을 Amazon Redshift의 테이블 이름으로 바꿉니다.

```
import pg
```

```
def get_connection(host):
    rs_conn_string = "host=%s port=%s dbname=%s user=%s password=%s" % (
        host, port, db_name, user, password_for_user)

    rs_conn = pg.connect(dbname=rs_conn_string)
    rs_conn.query("set statement_timeout = 1200000")
    return rs_conn

def query(con):
    statement = "Select * from table_name;"
    res = con.query(statement)
    return res
```

6. 아직 이동하지 않은 경우 `redshift_example` 디렉터리로 변경합니다.
7. 다음 중 하나를 수행하십시오.

- `.egg` 파일을 생성하려면 다음 명령을 실행합니다.

```
python setup.py bdist_egg
```

- `.whl` 파일을 생성하려면 다음 명령을 실행합니다.

```
python setup.py bdist_wheel
```

8. 이전 명령에 필요한 종속성을 설치합니다.
9. 해당 명령은 `dist` 디렉터리에서 파일을 생성합니다.

- `egg` 파일을 만든 경우, 이름은 `redshift_module-0.1-py2.7.egg`입니다.
- `wheel` 파일을 만든 경우, 이름은 `redshift_module-0.1-py2.7-none-any.whl`입니다.

이 파일을 Amazon S3에 업로드합니다.

이 예제에서 업로드된 파일 경로는 `s3://DOC-EXAMPLE-BUCKET/EGG-FILE` 또는 `s3://DOC-EXAMPLE-BUCKET/WHEEL-FILE`입니다.

10. AWS Glue 작업의 스크립트로 사용할 Python 파일을 생성하고 다음 코드를 파일에 추가합니다.

```

from redshift_module import pygresql_redshift_common as rs_common

con1 = rs_common.get_connection(redshift_endpoint)
res = rs_common.query(con1)

print "Rows in the table cities are: "

print res

```

11. 이전 파일을 Amazon S3에 업로드합니다. 이 예제에서 업로드된 파일 경로는 `s3://DOC-EXAMPLE-BUCKET/scriptname.py`입니다.
12. 이 스크립트를 사용하여 Python 셸 작업을 생성합니다. AWS Glue 콘솔의 [작업 속성(Job properties)] 페이지에서 Python 라이브러리 경로(Python library path) 상자에 `.egg/.whl` 파일의 경로를 지정합니다. 여러 `.egg/.whl` 파일 및 Python 파일이 있으면 이 상자에 쉼표로 구분된 목록을 제공합니다.

.egg 파일을 수정하거나 이름을 바꿀 때 파일 이름은 “python setup.py bdist_egg” 명령으로 생성된 기본 이름을 사용하거나 Python 모듈 명명 규칙을 준수해야 합니다. 자세한 내용은 [Python 코드 스타일 가이드](#)를 참조하십시오.

AWS CLI를 사용하여 다음 예와 같이 명령으로 작업을 생성하십시오.

```

aws glue create-job --name python-redshift-test-cli --role Role --command
'{"Name" : "pythonshell", "ScriptLocation" : "s3://DOC-EXAMPLE-BUCKET/
scriptname.py"}'
    --connections Connections="connection-name" --default-arguments '{"--extra-
py-files" : ["s3://DOC-EXAMPLE-BUCKET/EGG-FILE", "s3://DOC-EXAMPLE-BUCKET/WHEEL-
FILE"]}'

```

작업이 실행되면 스크립트는 Amazon Redshift 클러스터의 `table_name` 테이블에 생성된 행을 인쇄합니다.

AWS Glue에서 Python 셸 작업과 함께 AWS CloudFormation 사용

AWS Glue에서 Python 셸 작업과 함께 AWS CloudFormation을 사용할 수 있습니다. 다음은 그 예제입니다.

```
AWSTemplateFormatVersion: 2010-09-09
```

```
Resources:
  Python39Job:
    Type: 'AWS::Glue::Job'
    Properties:
      Command:
        Name: pythonshell
        PythonVersion: '3.9'
        ScriptLocation: 's3://bucket/location'
      MaxRetries: 0
      Name: python-39-job
      Role: RoleName
```

Python 셸 작업 출력에 대한 Amazon CloudWatch Logs 그룹은 `/aws-glue/python-jobs/output`입니다. 오류의 경우 로그 그룹 `/aws-glue/python-jobs/error`를 확인합니다.

AWS Glue 모니터링

모니터링은 AWS Glue 및 사용자의 다른 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 있어서 중요한 부분입니다. AWS는 AWS Glue를 보면서, 이상이 있을 때 이를 보고하고, 적절할 때 자동으로 조치를 취할 수 있게 해주는 모니터링 도구를 제공합니다.

다음과 같은 자동 모니터링 도구를 사용하여 AWS Glue를 관찰하고 문제 발생 시 보고할 수 있습니다.

- Amazon CloudWatch Events는 AWS 리소스의 변경 사항을 설명하는 시스템 이벤트의 스트림을 거의 실시간으로 제공합니다. CloudWatch Events는 자동화된 이벤트 중심 컴퓨팅을 지원합니다. 특정 이벤트를 감시하는 규칙을 작성하고 이러한 이벤트가 발생할 때 다른 AWS 서비스에서 자동화된 작업을 트리거할 수 있습니다. 자세한 내용은 [Amazon CloudWatch Events 사용 설명서](#)를 참조하세요.
- Amazon CloudWatch Logs로 Amazon EC2 인스턴스, AWS CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. CloudWatch Logs는 로그 파일의 정보를 모니터링하고 특정 임계값에 도달하면 사용자에게 알릴 수 있습니다. 또한 매우 내구력 있는 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 [Amazon CloudWatch Logs 사용 설명서](#)를 참조하십시오.
- AWS CloudTrail은 직접 수행하거나 AWS 계정을 대신하여 수행한 API 직접 호출 및 관련 이벤트를 캡처하고 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 어떤 사용자 및 계정이 AWS를 호출하는지, 어떤 소스 IP 주소에 호출이 이루어지는지, 언제 호출이 발생했는지 확인할 수 있습니다. 자세한 내용은 [AWS CloudTrail 사용 설명서](#)를 참조하십시오.

또한 AWS Glue 콘솔에서 다음과 같은 인사이트에 액세스하여 작업을 디버깅하고 프로파일링하는 데 도움을 받을 수 있습니다.

- Spark 작업 - 선택된 CloudWatch 지표 시리즈의 시각화를 볼 수 있으며, 새로운 작업은 Spark UI에 액세스할 수 있습니다. 자세한 내용은 [the section called “Spark 작업 모니터링”](#) 단원을 참조하십시오.
- Ray 작업 - 선택된 CloudWatch 지표 시리즈의 시각화를 볼 수 있습니다. 자세한 내용은 [the section called “Ray 작업 지표”](#) 단원을 참조하십시오.

주제

- [AWS Glue의 AWS 태그](#)
- [CloudWatch Events로 AWS Glue 자동화](#)
- [AWS Glue 리소스 모니터링](#)
- [AWS CloudTrail을 사용하여 AWS Glue API 호출 로깅](#)

AWS Glue의 AWS 태그

AWS Glue 리소스를 쉽게 관리할 수 있도록 선택에 따라 일부 AWS Glue 리소스 유형에 자체 태그를 할당할 수 있습니다. 태그는 AWS 리소스에 할당하는 레이블입니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. AWS Glue의 태그를 사용하여 리소스를 정리하고 식별할 수 있습니다. 태그는 비용 회계 보고서를 생성하고 리소스에 대한 액세스를 제한하는 데 사용할 수 있습니다. AWS Identity and Access Management를 사용하는 경우 AWS 계정에서 태그를 생성, 편집 또는 삭제할 수 있는 권한이 있는 사용자를 제어할 수 있습니다. 태그 관련 API를 직접 호출하는 권한 외에도 연결에서 태그 지정 API를 직접 호출할 수 있는 `glue:GetConnection` 권한과 데이터베이스에서 태그 지정 API를 직접 호출할 수 있는 `glue:GetDatabase` 권한도 필요합니다. 자세한 내용은 [ABAC \(접착제 포함\) AWS](#) 단원을 참조하십시오.

AWS Glue에서 다음 리소스에 태그를 지정할 수 있습니다.

- 연결
- 데이터베이스
- 크롤러
- 대화형 세션
- 개발 엔드포인트
- 작업
- 트리거
- 워크플로

- 청사진
- 기계 학습 변환
- 데이터 품질 규칙 세트
- 스트림 스키마
- 스트림 스키마 레지스트리

Note

이러한 AWS Glue 리소스의 태그 지정이 가능하도록 모범 사례로서 정책에 항상 `glue:TagResource` 작업을 포함시킵니다.

AWS Glue에서 태그를 사용할 때 다음을 고려하십시오.

- 엔터티당 최대 50개의 태그가 지원됩니다.
- AWS Glue에서 {"string": "string" ...} 포맷의 키-값 페어 목록으로 태그를 지정합니다.
- 객체에서 태그를 생성할 때 태그 키는 필수이고 태그 값은 선택 사항입니다.
- 태그 키와 태그 값은 대/소문자를 구분합니다.
- 태그 키와 태그 값은 접두사 `aws`를 포함하지 않아야 합니다. 이러한 태그에서 어떤 작업도 허용되지 않습니다.
- 태그 키의 최대 길이는 UTF-8 형식의 유니코드 문자 128자입니다. 태그 키는 비어 있거나 null이면 안 됩니다.
- 태그 값의 최대 길이는 UTF-8 형식의 유니코드 문자 256자입니다. 태그 값은 비어 있거나 null일 수 있습니다.

AWS Glue 연결에 대한 태깅 지원

리소스 태그를 기준으로 `CreateConnection`, `UpdateConnection`, `GetConnection` 및 `DeleteConnection` 작업 권한을 제한할 수 있습니다. 이렇게 하면 Data Catalog에서 JDBC 연결 정보를 가져와야 하는 JDBC 데이터 원본을 사용하는 AWS Glue 작업에 최소 권한 액세스 제어를 구현할 수 있습니다.

사용 예

`["connection-category", "dev-test"]` 태그를 사용하여 AWS Glue 연결을 생성합니다.

IAM 정책에서 GetConnection 작업에 대한 태그 조건을 지정합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "glue:GetConnection"
  ],
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {
      "aws:ResourceTag/tagKey": "dev-test"
    }
  }
}
```

예제

다음 예는 태그가 할당된 작업을 생성합니다.

AWS CLI

```
aws glue create-job --name job-test-tags --role MyJobRole --command
  Name=glueet1,ScriptLocation=S3://aws-glue-scripts//prod-job1
  --tags key1=value1,key2=value2
```

AWS CloudFormation JSON

```
{
  "Description": "AWS Glue Job Test Tags",
  "Resources": {
    "MyJobRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "glue.amazonaws.com"
                ]
              }
            }
          ]
        }
      }
    }
  }
}
```

```
        "Action": [
            "sts:AssumeRole"
        ]
    },
    "Path": "/",
    "Policies": [
        {
            "PolicyName": "root",
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Action": "*",
                        "Resource": "*"
                    }
                ]
            }
        }
    ]
},
"MyJob": {
    "Type": "AWS::Glue::Job",
    "Properties": {
        "Command": {
            "Name": "glueetl",
            "ScriptLocation": "s3://aws-glue-scripts//prod-job1"
        },
        "DefaultArguments": {
            "--job-bookmark-option": "job-bookmark-enable"
        },
        "ExecutionProperty": {
            "MaxConcurrentRuns": 2
        },
        "MaxRetries": 0,
        "Name": "cf-job1",
        "Role": {
            "Ref": "MyJobRole",
            "Tags": {
                "key1": "value1",
                "key2": "value2"
            }
        }
    }
}
```



```

    }
  }
}
}
}
}
}

```

AWS CloudFormation YAML

```

Description: AWS Glue Job Test Tags
Resources:
  MyJobRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - glue.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: "/"
      Policies:
        - PolicyName: root
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action: "*"
                Resource: "*"
  MyJob:
    Type: AWS::Glue::Job
    Properties:
      Command:
        Name: glueetl
        ScriptLocation: s3://aws-glue-scripts//prod-job1
      DefaultArguments:
        "--job-bookmark-option": job-bookmark-enable
      ExecutionProperty:
        MaxConcurrentRuns: 2
      MaxRetries: 0

```

```

Name: cf-job1
Role:
  Ref: MyJobRole
Tags:
  key1: value1
  key2: value2

```

자세한 내용은 [AWS 태그 지정 전략](#)을 참조하세요.

태그를 이용한 액세스 제어 방법에 대한 자세한 내용은 [ABAC \(접착제 포함\) AWS](#) 단원을 참조하십시오.

CloudWatch Events로 AWS Glue 자동화

Amazon CloudWatch Events를 사용하여 AWS 서비스를 자동화하고 애플리케이션 가용성 문제나 리소스 변경 같은 시스템 이벤트에 자동으로 응답합니다. AWS 서비스의 이벤트는 거의 실시간으로 CloudWatch Events로 전송됩니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 작업을 지정할 수 있습니다. 자동으로 트리거할 수 있는 작업은 다음과 같습니다.

- AWS Lambda 함수 호출
- Amazon EC2 Run Command 호출
- Amazon Kinesis Data Streams로 이벤트 릴레이
- AWS Step Functions 상태 머신 활성화
- Amazon SNS 주제 또는 Amazon SQS 대기열 알림

다음은 CloudWatch Events를 AWS Glue에 사용하는 몇 가지 예입니다.

- ETL 작업이 성공하면 Lambda 함수를 활성화합니다.
- ETL 작업이 실패하면 Amazon SNS 주제를 알립니다.

다음 CloudWatch Events는 AWS Glue에 의해 생성됩니다.

- "detail-type": "Glue Job State Change" 이벤트는 SUCCEEDED, FAILED, TIMEOUT 및 STOPPED에 대해 생성됩니다.
- 작업 지연 알림 임계값을 초과한다면 RUNNING, STARTING 및 STOPPING 작업 실행에 "detail-type": "Glue Job Run Status"용 이벤트가 생성됩니다. 이러한 이벤트를 수신하려면 작업 지연 알림 임계값 속성을 설정해야 합니다.

작업 지연 알림 임계값을 초과한 경우 작업 실행 상태당 하나의 이벤트만 생성됩니다.

- "detail-type": "Glue Crawler State Change" 이벤트는 Started, Succeeded 및 Failed을 위해 발생합니다.
- "detail-type": "Glue Data Catalog Database State Change" 이벤트는 CreateDatabase, DeleteDatabase, CreateTable, DeleteTable 및 BatchDeleteTable에 대해 생성됩니다. 예를 들어 테이블이 생성되거나 삭제되면 CloudWatch Events에 알림이 전송됩니다. 순서가 잘못되었거나 누락되는 등 알림 이벤트의 순서 또는 존재 여부에 따라 프로그램을 작성할 수 없을 수 있습니다. 이벤트는 최선의 작업을 기반으로 발생합니다. 알림 세부 정보:
 - typeOfChange에는 API 작업의 이름이 포함되어 있습니다.
 - databaseName에는 영향을 받는 데이터베이스 이름이 포함되어 있습니다.
 - changedTables는 알림당 영향을 받는 테이블의 이름을 최대 100개까지 포함합니다. 테이블 이름이 길면 여러 알림이 생성될 수 있습니다.
- "detail-type": "Glue Data Catalog Table State Change"에 대한 이벤트는 UpdateTable, CreatePartition, BatchCreatePartition, UpdatePartition, DeletePartition, BatchUpdatePartition 및 BatchDeletePartition에 대해 생성됩니다. 예를 들어 테이블이 생성되거나 파티션이 업데이트되면 CloudWatch Events에 알림이 전송됩니다. 순서가 잘못되었거나 누락되는 등 알림 이벤트의 순서 또는 존재 여부에 따라 프로그램을 작성할 수 없을 수 있습니다. 이벤트는 최선의 작업을 기반으로 발생합니다. 알림 세부 정보:
 - typeOfChange에는 API 작업의 이름이 포함되어 있습니다.
 - databaseName은 영향을 받는 리소스가 포함된 데이터베이스의 이름을 포함합니다.
 - tableName에는 영향을 받는 테이블 이름이 포함되어 있습니다.
 - changedPartitions는 하나의 알림에서 최대 100개의 영향을 받는 파티션을 지정합니다. 파티션 이름이 길면 여러 알림이 생성될 수 있습니다.

예를 들어 Year 및 Month와 같은 두 개의 파티션 키가 있는 경우 "2018,01", "2018,02"는 "Year=2018" and "Month=01" 및 "Year=2018" and "Month=02"가 있는 파티션을 수정합니다.

```
{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
  "detail-type": "Glue Data Catalog Table State Change",
  "source": "aws.glue",
  "account": "123456789012",
  "time": "2017-09-07T18:57:21Z",
```

```
"region": "us-west-2",
"resources": ["arn:aws:glue:us-west-2:123456789012:database/default/foo"],
"detail": {
  "changedPartitions": [
    "2018,01",
    "2018,02"
  ],
  "databaseName": "default",
  "tableName": "foo",
  "typeOfChange": "BatchCreatePartition"
}
}
```

자세한 내용은 [Amazon CloudWatch Events 사용 설명서](#)를 참조하세요. AWS Glue에 한정되는 이벤트는 [AWS Glue 이벤트](#)를 참조하십시오.

AWS Glue 리소스 모니터링

AWS Glue에는 예상치 못한 과도한 프로비저닝과 요금 증가를 의도한 악의적인 행동으로부터 고객을 보호하기 위한 서비스 한도가 있습니다. 또한 이러한 한도는 서비스도 보호합니다. AWS Service Quota 콘솔에 로그인하면 고객은 현재 리소스 한도를 확인하고 필요한 경우 리소스 증가를 요청할 수 있습니다.

AWS Glue에서는 Amazon CloudWatch에서 서비스의 리소스 사용량을 백분율로 보고 사용량을 모니터링하도록 CloudWatch 경보를 구성할 수 있습니다. Amazon CloudWatch는 Amazon 인프라에서 실행되는 AWS 리소스 및 고객 애플리케이션에 대한 모니터링을 제공합니다. 지표는 무료로 제공됩니다. 다음과 같은 지표가 지원됩니다.

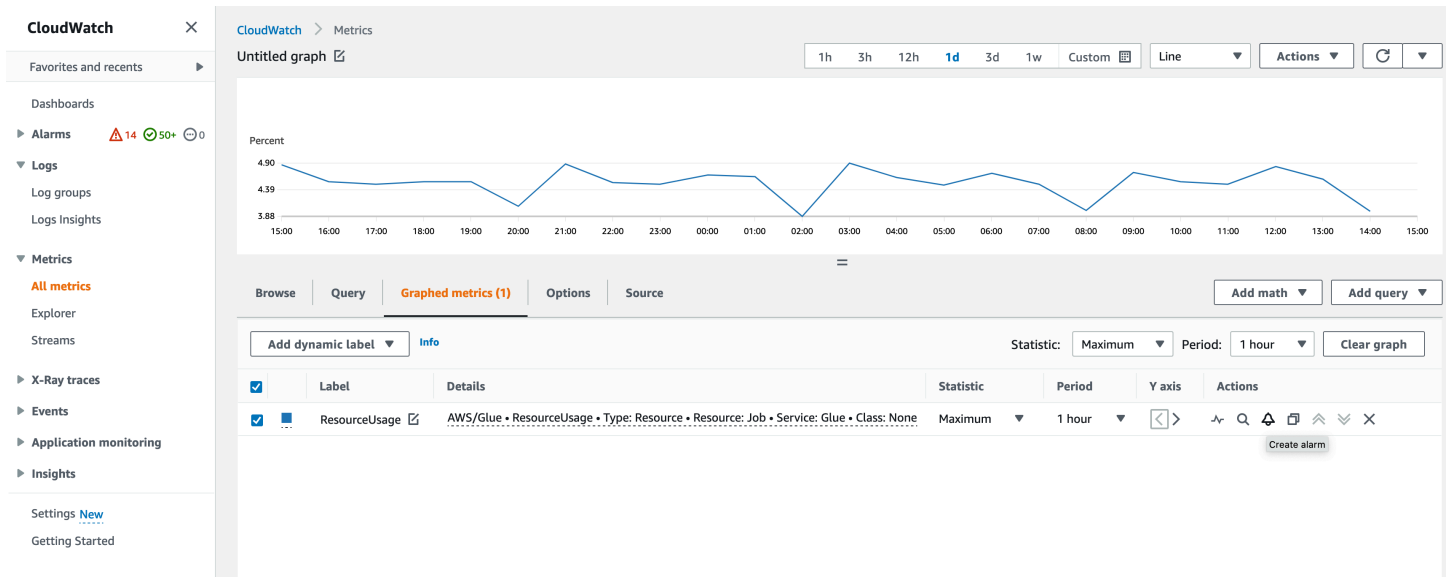
- 계정당 워크플로우 수
- 계정당 트리거 수
- 계정당 작업 수
- 계정당 동시 작업 실행 개수
- 계정당 청사진 수
- 계정당 대화형 세션 수

리소스 지표 구성 및 사용

이 기능을 사용하려면 Amazon CloudWatch 콘솔로 이동하여 지표를 보고 경보를 구성하면 됩니다. 지표는 AWS/Glue 네임스페이스 아래에 있으며 실제 리소스 사용량 수치를 리소스 할당량으로 나눈 백분율입니다. CloudWatch 지표는 사용자 계정으로 전송되므로 비용이 청구되지 않습니다. 예를 들어, 10개의 워크플로를 생성하고 서비스 할당량으로 최대 200개의 워크플로를 사용할 수 있는 경우 사용량은 $10/200 = 5\%$ 이고, 그래프에는 5의 데이터 포인트가 백분율로 표시됩니다. 구체적으로 설명하면 다음과 같습니다.

```

Namespace: AWS/Glue
Metric name: ResourceUsage
Type: Resource
Resource: Workflow (or Trigger, Job, JobRun, Blueprint, InteractiveSession)
Service: Glue
Class: None
  
```



CloudWatch 콘솔에서 지표에 대한 경보를 생성하려면:

1. 지표를 찾은 후에 그래프로 표시된 지표로 이동합니다.
2. 작업에서 경보 생성을 클릭합니다.
3. 필요에 따라 경보를 구성합니다.

리소스 사용량이 변경될 때마다(예: 증가 또는 감소) 지표를 생성합니다. 그러나 리소스 사용량이 변하지 않는 경우에는 매시간 지표를 생성하므로 지속적인 CloudWatch 그래프가 생성됩니다. 데이터 포인트가 누락되는 것을 방지하기 위해 1시간 미만의 기간을 구성하는 것은 권장하지 않습니다.

또한 다음 예제에서처럼 AWS CloudFormation을 사용하여 경보를 구성할 수 있습니다. 이 예제에서 워크플로 리소스 사용량이 80%에 도달하면 기존 SNS 주제에 메시지를 보내는 경보가 트리거되며, 이 주제를 구독하면 알림을 받을 수 있습니다.

```
{
  "Type": "AWS::CloudWatch::Alarm",
  "Properties": {
    "AlarmName": "WorkflowUsageAlarm",
    "ActionsEnabled": true,
    "OKActions": [],
    "AlarmActions": [
      "arn:aws:sns:af-south-1:085425700061:Default_CloudWatch_Alarms_Topic"
    ],
    "InsufficientDataActions": [],
    "MetricName": "ResourceUsage",
    "Namespace": "AWS/Glue",
    "Statistic": "Maximum",
    "Dimensions": [{
      "Name": "Type",
      "Value": "Resource"
    },
    {
      "Name": "Resource",
      "Value": "Workflow"
    },
    {
      "Name": "Service",
      "Value": "Glue"
    },
    {
      "Name": "Class",
      "Value": "None"
    }
  ],
  "Period": 3600,
  "EvaluationPeriods": 1,
  "DatapointsToAlarm": 1,
  "Threshold": 80,
  "ComparisonOperator": "GreaterThanThreshold",
  "TreatMissingData": "notBreaching"
}
}
```

AWS CloudTrail을 사용하여 AWS Glue API 호출 로깅

AWS Glue는 AWS Glue에서 사용자, 역할, 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 AWS Glue에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 AWS Glue 콘솔로부터의 직접 호출과 AWS Glue API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 AWS Glue 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 AWS Glue에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 설명은 [AWS CloudTrail 사용자 가이드](#)를 참조하세요.

CloudTrail의 AWS Glue 정보

CloudTrail은 계정 생성 시 AWS 계정에서 사용되도록 설정됩니다. AWS Glue에서 활동이 발생하면 해당 활동이 [이벤트 기록]의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 설명은 [CloudTrail 이벤트를 사용하여 이벤트 보기](#)를 참조하십시오.

AWS Glue에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 지역에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 지역의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 설명은 다음 자료를 참조하십시오:

- [AWS 계정에 대한 추적 생성](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에서 Amazon SNS 알림 구성](#)
- [여러 리전으로부터 CloudTrail 로그 파일 받기 및 여러 계정으로부터 CloudTrail 로그 파일 받기](#)

모든 AWS Glue 작업이 CloudTrail에서 로깅되고 [AWS Glue API](#)에서 문서화됩니다. 예를 들어 CreateDatabase, CreateTable, CreateScript 작업을 직접적으로 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 보안 인증 정보로 했는지 여부.
- 역할 또는 페더레이션 사용자의 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

자세한 설명은 [CloudTrail userIdentity 요소](#)를 참조하십시오.

그러나 CloudTrail은 호출에 따른 모든 정보를 로그하지 않습니다. 예를 들어, 연결에 사용된 ConnectionProperties와 같은 특정 민감한 정보는 로그하지 않고 다음 API에 의해 반환된 응답 대신 null을 로그합니다.

BatchGetPartition	GetCrawlers	GetJobs	GetTable
CreateScript	GetCrawlerMetrics	GetJobRun	GetTables
GetCatalogImportStatus	GetDatabase	GetJobRuns	GetTableVersions
GetClassifier	GetDatabases	GetMapping	GetTrigger
GetClassifiers	GetDataflowGraph	GetObjects	GetTriggers
GetConnection	GetDevEndpoint	GetPartition	GetUserDefinedFunction
GetConnections	GetDevEndpoints	GetPartitions	GetUserDefinedFunctions
GetCrawler	GetJob	GetPlan	

AWS Glue 로그 파일 항목 이해

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 전송할 수 있게 하는 구성입니다.

CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공용 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음은 DeleteCrawler 작업을 보여주는 CloudTrail 로그 항목이 나타낸 예입니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/johndoe",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "johndoe"
  },
  "eventTime": "2017-10-11T22:29:49Z",
  "eventSource": "glue.amazonaws.com",
```



```

"eventName": "DeleteCrawler",
"awsRegion": "us-east-1",
"sourceIPAddress": "72.21.198.64",
"userAgent": "aws-cli/1.11.148 Python/3.6.1 Darwin/16.7.0 botocore/1.7.6",
"requestParameters": {
  "name": "tes-alpha"
},
"responseElements": null,
"requestID": "b16f4050-aed3-11e7-b0b3-75564a46954f",
"eventID": "e73dd117-cfd1-47d1-9e2f-d1271cad838c",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}

```

다음은 CreateConnection 작업을 설명하는 CloudTrail 로그 항목을 보여주는 예제입니다.

```

{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/johndoe",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "johndoe"
  },
  "eventTime": "2017-10-13T00:19:19Z",
  "eventSource": "glue.amazonaws.com",
  "eventName": "CreateConnection",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "72.21.198.66",
  "userAgent": "aws-cli/1.11.148 Python/3.6.1 Darwin/16.7.0 botocore/1.7.6",
  "requestParameters": {
    "connectionInput": {
      "name": "test-connection-alpha",
      "connectionType": "JDBC",
      "physicalConnectionRequirements": {
        "subnetId": "subnet-323232",
        "availabilityZone": "us-east-1a",
        "securityGroupIdList": [
          "sg-12121212"
        ]
      }
    }
  }
}

```

```

    }
  },
  "responseElements": null,
  "requestID": "27136ebc-afac-11e7-a7d6-ab217e5c3f19",
  "eventID": "e8b3baeb-c511-4597-880f-c16210c60a4a",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}

```

AWS Glue 작업 실행 상태

AWS Glue 추출, 변환, 로드 작업이 실행되는 동안 또는 중지된 후 작업 상태를 볼 수 있습니다. AWS Glue 콘솔, AWS Command Line Interface(AWS CLI) 또는 AWS Glue API의 [GetJobRun 작업](#)을 사용하여 상태를 볼 수 있습니다.

가능한 작업 실행 상태는 STARTING, RUNNING, STOPPING, STOPPED, SUCCEEDED, FAILED, ERROR, WAITING 및 TIMEOUT입니다.

다음 표에는 비정상적인 작업 종료를 나타내는 상태가 나열되어 있습니다.

작업 실행 상태	설명
FAILED	작업이 허용되는 최대 동시 실행을 초과했거나 알 수 없는 종료 코드로 종료되었습니다.
ERROR	워크플로, 일정 트리거 또는 이벤트 트리거가 삭제된 작업을 실행하려고 했습니다.
TIMEOUT	작업 런타임이 지정된 시간 제한 값을 초과했습니다.

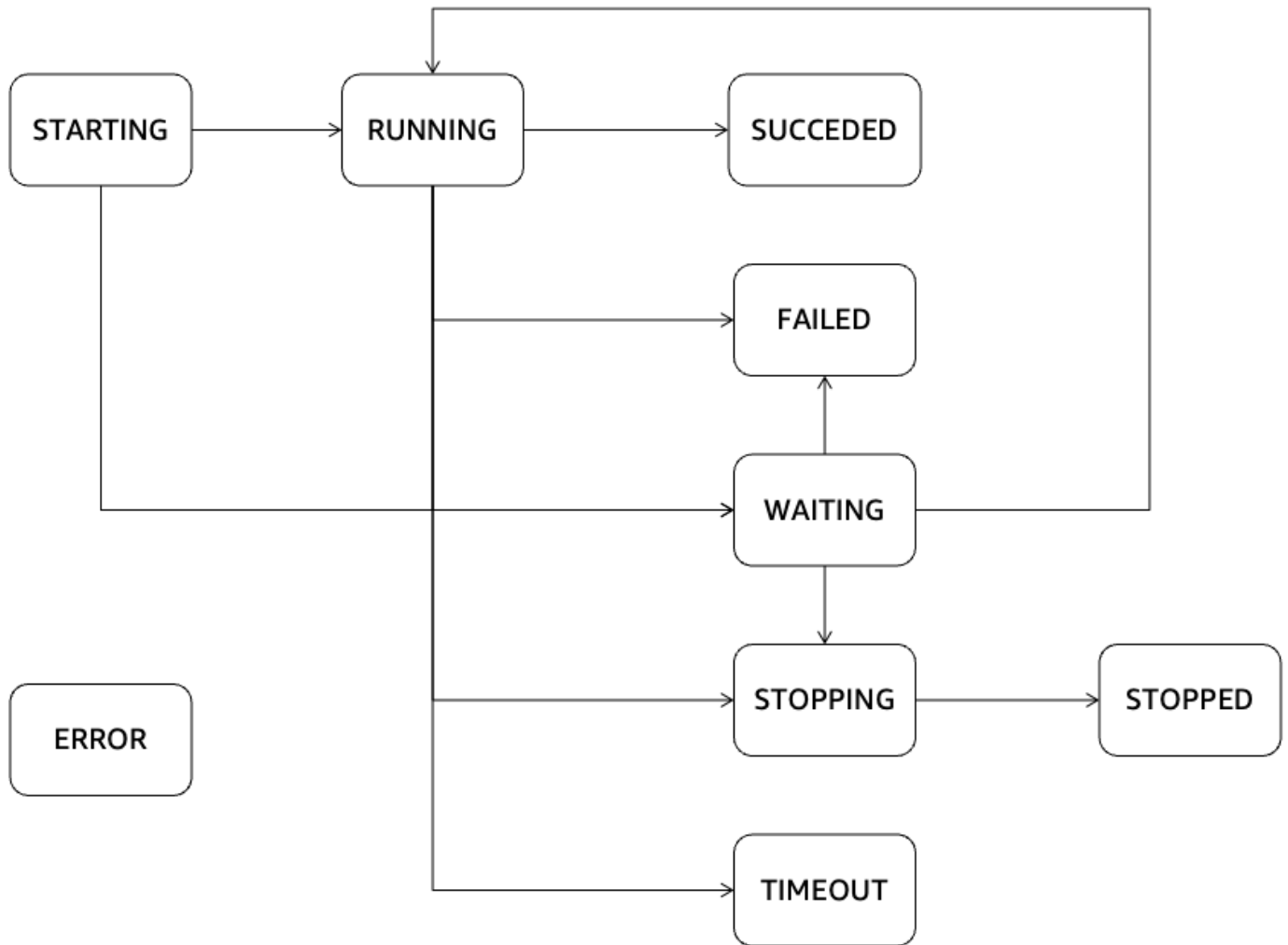
WAITING 상태는 작업 실행에서 리소스를 기다리고 있음을 나타냅니다. 다음 표에서는 여러 작업 클래스의 대기 동작을 설명합니다.

작업 유형	동작
Spark 작업(표준)	<code>maxRetries</code> 구성에 따라 재시도하도록 구성되지 않은 작업은 WAITING 상태가 될 수 있습니다.

작업 유형	동작
	<p>습니다. 서비스에서 실행을 시작하기에 충분한 리소스를 얻을 수 없는 경우 새 작업 실행은 WAITING 상태가 됩니다. 이 문제는 계정의 서비스 할당량 또는 해당 지역의 용량 제한으로 인해 다음과 같은 오류 사례 중 하나가 발생할 수 있습니다.</p> <ul style="list-style-type: none"> • 계정당 최대 동시 작업 실행 수를 초과했습니다 • 작업당 최대 동시 작업 실행 수를 초과했습니다(계정 수준 서비스 할당량 및 작업에 MaxConcurrentRuns 로 지정한 한도 포함) • 최대 동시 컴퓨팅(DPU 사용량) 초과 • 사용 가능하지 않는 리소스 <p>AWS Glue 서비스 할당량에 대한 자세한 내용은 AWS Glue 엔드포인트 및 할당량을 참조하십시오. AWS Glue가 리소스를 기다리는 시간은 상황에 따라 다를 수 있습니다. 리소스를 확보하려고 할 때 작업이 터미널이 아닌 상태로 전환될 수 있습니다. 결국 리소스를 확보할 수 없는 경우 작업이 FAILED로 전환됩니다. AWS Glue는 최대 15분 또는 10회 시도(둘 중 먼저 도래하는 시점 기준) 동안 재시도합니다.</p>
<p>Spark 잡(플렉스)</p>	<p>서비스에서 실행을 시작하기에 충분한 리소스를 얻을 수 없는 경우 새 작업 실행은 WAITING 상태가 되며, 이로 인해 실행 시작이 지연됩니다. 실행은 최대 20분 동안 WAITING 상태가 됩니다(서비스에 의해 시간 초과가 제어됨). 15분 후 서비스가 강제 시작을 시도하며 사용 가능한 용량에 따라 적절한 오류 메시지와 함께 실행이 시작되거나 실패할 수 있습니다.</p>

작업 유형	동작
Python 셸 작업	Spark를 사용하는 표준 작업과 동일한 동작입니다.

다음 상태 다이어그램은 AWS Glue 작업의 수명 주기 동안 예상되는 상태 전환을 간략하게 설명합니다. 이 정보는 모든 작업 유형에 적용됩니다.



AWS Glue 스트리밍

AWS Glue의 AWS Glue구성 요소인 스트리밍을 사용하면 거의 실시간으로 스트리밍 데이터를 효율적으로 처리하여 데이터 수집, 처리, 기계 학습과 같은 중요한 작업을 수행할 수 있습니다. Apache Spark Streaming 프레임워크를 사용하는 스트리밍은 AWS Glue 스트리밍 데이터를 대규모로 처리할 수 있는 서버리스 서비스를 제공합니다. AWS Glue 서버리스 인프라, 자동 크기 조정, 시각적 작업 개발, 스트리밍 작업을 위한 인스턴트 온 노트북 및 기타 성능 향상과 같은 Apache Spark를 기반으로 다양한 최적화를 제공합니다.

스트리밍 사용 사례

AWS Glue 스트리밍의 몇 가지 일반적인 사용 사례는 다음과 같습니다.

Near-real-time 데이터 처리: AWS Glue 스트리밍을 통해 조직은 스트리밍 데이터를 거의 실시간으로 처리하여 최신 정보를 기반으로 통찰력을 얻고 시기적절한 결정을 내릴 수 있습니다.

사기 탐지: 스트리밍을 활용하여 AWS Glue 스트리밍 데이터를 실시간으로 분석할 수 있으므로 신용 카드 사기, 네트워크 침입 또는 온라인 사기와 같은 사기 행위를 탐지하는 데 유용합니다. 수신 데이터를 지속적으로 처리하고 분석하여 의심스러운 패턴이나 이상 징후를 신속하게 식별할 수 있습니다.

소셜 미디어 분석: AWS Glue 스트리밍은 트윗, 게시물 또는 댓글과 같은 실시간 소셜 미디어 데이터를 처리할 수 있으므로 조직은 트렌드, 감정 분석을 모니터링하고 브랜드 평판을 실시간으로 관리할 수 있습니다.

사물 인터넷 (IoT) 분석: AWS Glue 스트리밍은 IoT 장치, 센서 및 연결된 기계에서 생성되는 고속 데이터 스트림을 처리하고 분석하는 데 적합합니다. 이는 실시간 모니터링, 이상 탐지, 예측 유지 보수 및 기타 IoT 분석 사용 사례를 지원합니다.

클릭스트림 분석: AWS Glue 스트리밍은 웹사이트 또는 모바일 애플리케이션의 실시간 클릭스트림 데이터를 처리하고 분석할 수 있습니다. 이를 통해 기업은 사용자 행동에 대한 인사이트를 얻고, 사용자 경험을 개인화하고, 실시간 클릭스트림 데이터를 기반으로 마케팅 캠페인을 최적화할 수 있습니다.

로그 모니터링 및 분석: AWS Glue 스트리밍은 서버, 애플리케이션 또는 네트워크 장치의 로그 데이터를 실시간으로 지속적으로 처리하고 분석할 수 있습니다. 이는 이상을 탐지하고, 문제를 해결하고, 시스템 상태와 성능을 모니터링하는 데 도움이 됩니다.

추천 시스템: AWS Glue 스트리밍은 사용자 활동 데이터를 실시간으로 처리하고 추천 모델을 동적으로 업데이트할 수 있습니다. 이를 통해 사용자 행동 및 기본 설정을 기반으로 개인화된 실시간 추천이 가능합니다.

다음은 AWS Glue 스트리밍을 적용할 수 있는 다양한 사용 사례의 몇 가지 예입니다. AWS 에코시스템 및 관리형 서비스와의 통합 덕분에 클라우드에서의 실시간 스트림 처리 및 분석을 위한 편리한 선택이 됩니다.

AWS Glue 스트리밍을 사용하면 어떤 이점이 있나요?

AWS Glue 스트리밍 사용의 이점은 다음과 같습니다.

- **서버리스:** AWS Glue 스트리밍은 서버리스이므로 인프라를 관리할 필요가 없습니다. 이를 통해 운영 오버헤드가 줄어들고 사용자는 인프라 관리보다는 데이터 처리 및 분석 작업에 집중할 수 있습니다.
- **자동 크기 조정:** AWS Glue 스트리밍은 워크로드에 따라 처리 용량을 동적으로 조정하는 자동 크기 조정 기능을 제공합니다. 데이터 볼륨의 변동을 처리하기 위해 자동으로 스케일 아웃 또는 스케일 인하여 최적의 성능과 리소스 활용도를 보장합니다.
- **시각적 개발:** 스트리밍 작업 개발은 복잡할 수 있습니다. AWS Glue 스트리밍은 시각적 저작 도구인 AWS Glue Studio를 제공하여 이러한 문제를 해결합니다. AWS Glue Studio는 스트리밍 워크플로를 만드는 프로세스를 간소화하고 개발자가 스트리밍 애플리케이션을 시각적으로 설계 및 관리할 수 있도록 지원하여 학습 시간을 줄이고 생산성을 높일 수 있습니다.
- **비용 효율성:** AWS Glue 스트리밍은 서버리스 서비스로서 인프라를 프로비저닝하고 유지 관리할 필요가 없으므로 비용 효율성을 제공합니다. 스트리밍 작업을 실행하는 동안 소비된 리소스를 기준으로 사용자에게 요금이 청구되므로 실제 사용량에 따라 비용을 최적화하고 조정할 수 있습니다.
- **복잡한 워크로드 처리:** AWS Glue 스트리밍은 복잡한 스트리밍 워크로드를 처리하도록 설계되었습니다. 대량의 실시간 데이터를 처리 및 분석하고, 고급 변환을 지원하고, 다른 AWS 서비스와 통합하여 정교한 스트리밍 데이터 파이프라인 및 분석 워크플로를 지원할 수 있습니다.
- **종속 없음:** AWS Glue 스트리밍은 유연성을 제공하고 벤더 종속을 방지합니다. 사용자는 AWS Glue 스트리밍을 보다 광범위한 AWS 에코시스템의 일부로 활용하여 다른 서비스와 원활하게 통합할 수 있습니다. AWS 이를 통해 특정 기술이나 플랫폼에 얽매이지 않고도 기존 데이터 소스, 애플리케이션 및 서비스와 쉽게 통합할 수 있습니다.

스트리밍은 언제 사용하나요? AWS Glue

스트리밍 사용 사례에는 여러 가지 옵션이 있습니다. 다음 시나리오에서 AWS Glue 스트리밍하는 것이 좋습니다.

1. 이미 일괄 처리에 AWS Glue 또는 Spark를 사용하고 있다면 AWS Glue 스트리밍이 가장 적합합니다. 새로운 언어나 프레임워크를 배울 필요 없이 스트리밍 작업 구축으로 원활하게 전환할 수 있습니다.

니다. 스트리밍은 기존 지식과 인프라를 활용하여 작업 개발 프로세스를 단순화하고 데이터 처리 기능을 실시간 AWS Glue 스트리밍 시나리오로 쉽게 확장할 수 있도록 합니다.

2. 일괄 처리, 스트리밍 및 이벤트 기반 워크로드를 처리하기 위한 통합 서비스 또는 제품이 필요한 경우 AWS Glue 스트리밍이 적합한 솔루션입니다. AWS Glue 스트리밍을 사용하면 데이터 처리 요구 사항을 단일 프레임워크로 통합하여 여러 시스템을 관리하는 복잡성을 없앨 수 있습니다. 이를 통해 다양한 데이터 워크플로를 효율적으로 개발하고 유지 보수하는 동시에 다양한 워크로드 유형에서 일관성과 호환성을 보장할 수 있습니다.
3. AWS Glue 스트리밍은 매우 큰 스트리밍 데이터 볼륨과 복잡한 변환 (예: 스트림 간 결합 또는 관계형 데이터베이스) 과 관련된 시나리오에 적합합니다. 대량의 데이터 스트림을 효율적으로 처리하고 분석할 수 있으므로 까다로운 워크로드를 쉽게 처리할 수 있습니다. 고속 데이터 수집이든 복잡한 데이터 조작이든 AWS Glue 스트리밍의 확장성과 고급 처리 기능은 최적의 성능과 정확한 결과를 보장합니다.
4. 스트리밍 작업을 구축하는 데 시각적인 접근 방식을 선호하는 경우 스트리밍 애플리케이션을 시각적으로 설계하고 관리할 수 있는 AWS Glue Studio를 AWS Glue 제공하여 개발 프로세스를 단순화할 수 있습니다. 이 직관적인 인터페이스를 통해 개발자는 시각적 인터페이스를 사용하여 스트리밍 워크플로를 생성, 구성 및 모니터링할 수 있으므로 학습 곡선을 줄이고 생산성을 높일 수 있습니다.
5. AWS Glue 스트리밍은 10초 이상의 엄격한 SLA (서비스 수준 계약) 가 적용되는 near-real-time 사용 사례에 적합합니다.
6. Apache Iceberg, Apache Hudi 또는 Delta Lake를 사용하여 트랜잭션 데이터 레이크를 구축하는 경우 AWS Glue 스트리밍은 이러한 오픈 테이블 형식을 기본적으로 지원합니다. 이러한 원활한 통합을 통해 이러한 트랜잭션 데이터 레이크에서 직접 스트리밍 데이터를 처리하여 데이터 일관성, 무결성 및 호환성을 보장할 수 있습니다.
7. 다양한 데이터 대상에 대한 스트리밍 데이터를 수집해야 하는 경우: AWS Glue 스트리밍은 Amazon Redshift, Amazon RDS, Amazon Aurora, Oracle, SQL Server 및 기타 대상과 같은 다양한 데이터 대상에 대한 기본 대상을 제공합니다.

지원되는 데이터 원본

AWS Glue 스트리밍은 다음과 같은 데이터 소스를 지원합니다.

- Amazon Kinesis
- Amazon MSK(Managed Streaming for Apache Kafka)
- 자체 관리형 Apache Kafka

지원되는 데이터 대상

AWS Glue 스트리밍은 다음과 같은 다양한 데이터 대상을 지원합니다.

- 데이터 카탈로그에서 지원하는 AWS Glue 데이터 대상
- Amazon S3
- Amazon Redshift
- MySQL
- PostgreSQL
- Oracle
- Microsoft SQL Server
- Snowflake
- JDBC를 사용하여 연결할 수 있는 모든 데이터베이스
- Apache Iceberg, Delta 및 Apache Hudi
- AWS Glue Marketplace 커넥터

자습서: AWS Glue Studio를 사용하여 첫 번째 스트리밍 워크로드 구축

이 자습서에서는 AWS Glue Studio를 사용하여 스트리밍 작업을 생성하는 방법을 알아봅니다. AWS Glue Studio는 AWS Glue 작업을 생성하는 시각적 인터페이스입니다.

지속적으로 실행되고 Amazon Kinesis Data Streams, Apache Kafka 및 Amazon Managed Streaming for Apache Kafka(Amazon MSK)에서 스트리밍 소스의 데이터를 소비하는 스트리밍 추출, 전환, 적재(ETL) 작업을 생성할 수 있습니다.

필수 조건

이 자습서를 따르려면 AWS Glue, Amazon Kinesis, Amazon S3, Amazon Athena, AWS CloudFormation, AWS Lambda 및 Amazon Cognito를 사용할 수 있는 AWS 콘솔 권한이 있는 사용자가 필요합니다.

Amazon Kinesis의 스트리밍 데이터 소비

주제

- [Kinesis 데이터 생성기로 모의 데이터 생성](#)
- [AWS Glue Studio로 AWS Glue 스트리밍 작업 생성](#)
- [변환 수행 및 Amazon S3에 변환된 결과 저장](#)

Kinesis 데이터 생성기로 모의 데이터 생성

Kinesis 데이터 생성기(KDG)를 사용하여 JSON 형식의 샘플 데이터를 종합적으로 생성할 수 있습니다. [도구 설명서](#)에서 전체 지침과 세부 정보를 찾아볼 수 있습니다.

1. 시작하려면



클

클릭하여 AWS 환경에서 AWS CloudFormation 템플릿을 실행합니다.

Note

Kinesis 데이터 생성기용 Amazon Cognito 사용자와 같은 일부 리소스가 AWS 계정에 이미 존재하기 때문에 CloudFormation 템플릿 오류가 발생할 수 있습니다. 다른 자습서나 블로그에서 이미 설정했기 때문일 수 있습니다. 이 문제를 해결하려면 새 AWS 계정에서 템플릿을 사용해 새로 시작하거나 다른 AWS 리전을 탐색해 봅니다. 이러한 옵션을 사용하면 기존 리소스와 충돌하지 않고 자습서를 실행할 수 있습니다.

템플릿은 Kinesis 데이터 스트림과 Kinesis Data Generator 계정을 프로비저닝합니다. 또한 데이터를 보관할 Amazon S3 버킷과 이 자습서에 필요한 권한이 있는 Glue 서비스 역할을 생성합니다.

2. KDG가 인증에 사용할 사용자 이름과 암호를 입력합니다. 나중에 사용할 수 있도록 사용자 이름과 암호를 기록해 둡니다.
3. 마지막 단계까지 다음을 선택합니다. IAM 리소스 생성을 확인합니다. 화면 상단에 최소 요구 사항을 충족하지 않는 암호와 같은 오류가 있는지 확인하고 템플릿을 배포합니다.
4. 스택의 출력 탭으로 이동합니다. 템플릿이 배포되면 생성된 속성 KinesisDataGeneratorUrl이 표시됩니다. 해당 URL을 클릭합니다.
5. 기록해 둔 사용자 이름과 암호를 입력합니다.
6. 사용 중인 리전을 선택하고 Kinesis 스트림 GlueStreamTest-{AWS::AccountId}를 선택합니다.
7. 다음 템플릿을 입력합니다.

```

{
  "ventilatorid": {{random.number(100)}},
  "eventtime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
  "serialnumber": "{{random.uuid}}",
  "pressurecontrol": {{random.number(
    {
      "min":5,
      "max":30
    }
  )}},
  "o2stats": {{random.number(
    {
      "min":92,
      "max":98
    }
  )}},
  "minutevolume": {{random.number(
    {
      "min":5,
      "max":8
    }
  )}},
  "manufacturer": "{{random.arrayElement(
    ["3M", "GE", "Vyair", "Getinge"]
  )}}"
}

```

이제 테스트 템플릿을 사용하여 모의 데이터를 보고 데이터 전송을 사용하여 모의 데이터를 Kinesis 로 모을 수 있습니다.

8. 데이터 전송을 클릭하고 Kinesis에 5~10,000개의 레코드를 생성합니다.

AWS Glue Studio로 AWS Glue 스트리밍 작업 생성

1. 동일한 리전의 콘솔에서 AWS Glue로 이동합니다.
2. 데이터 통합 및 ETL 아래의 왼쪽 탐색 표시줄에서 ETL 작업을 선택합니다.
3. 빈 캔버스가 있는 시각적 객체를 통해 AWS Glue 작업을 생성합니다.

AWS Glue > Jobs

AWS Glue Studio [Info](#)

Create job [Info](#) Create

Visual with a source and target
 Start with a source, ApplyMapping transform, and target.

Visual with a blank canvas
 Author using an interactive visual interface.

Spark script editor
 Write or upload your own Spark code.

Python Shell script editor
 Write or upload your own Python shell script.

Jupyter Notebook
 Write your own code in a Jupyter Notebook for interactive development.

Ray script editor New
 Write your own code to run on Ray.

4. 작업 세부 정보 탭으로 이동합니다.
5. AWS Glue 작업 이름으로 DemoStreamingJob을 입력합니다.
6. IAM 역할로 CloudFormation 템플릿에서 프로비저닝한 역할인 glue-tutorial-role-\${AWS::AccountId}를 선택합니다.
7. Glue 버전으로 Glue 3.0을 선택합니다. 다른 모든 옵션은 기본값으로 둡니다.

Basic properties [Info](#)

Name

DemoStreamingJob

Description - optional

Descriptions can be up to 2048 characters long.

IAM Role

Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job.

glue-tutorial-role- XXXXXXXXXX ▼ ↻

Type

The type of ETL job. This is set automatically based on the types of data sources you have selected.

Spark Streaming

Glue version [Info](#)

Glue 3.0 - Supports spark 3.1, Scala 2, Python 3 ▼

Language

Python 3 ▼

Worker type

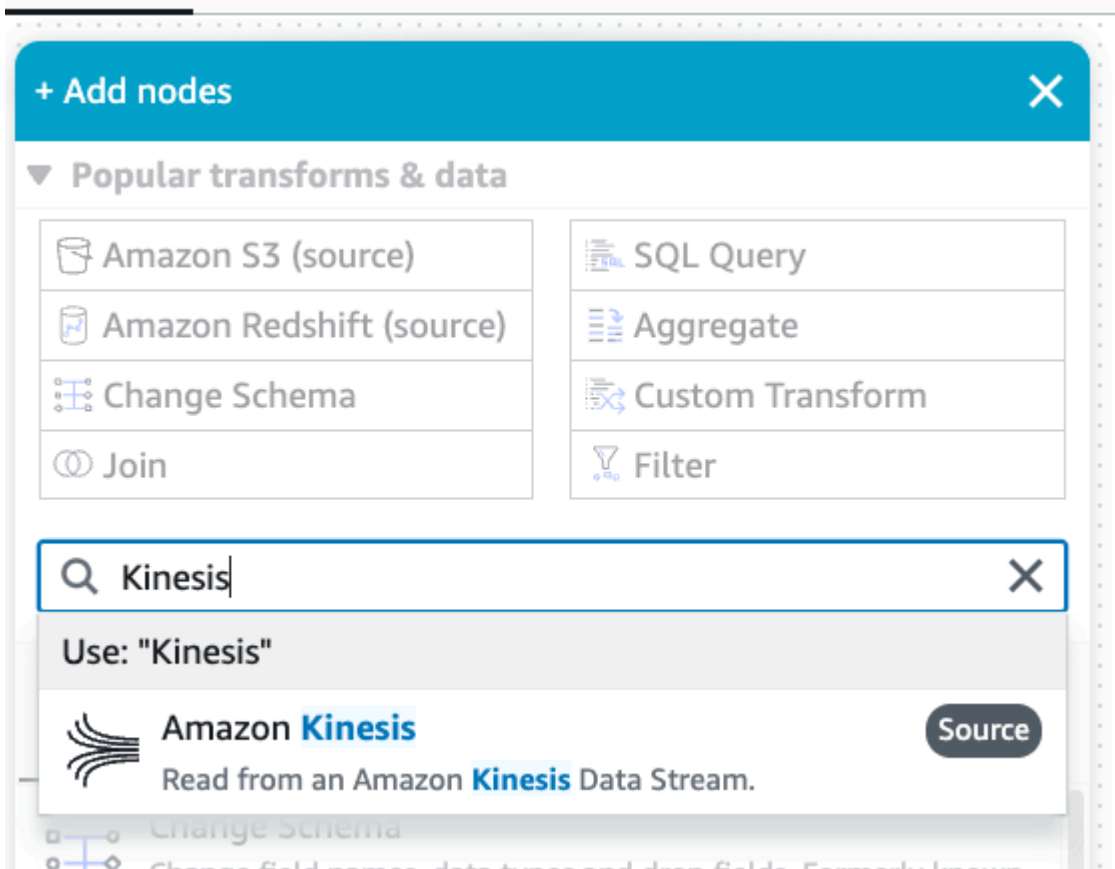
Set the type of predefined worker that is allowed when a job runs.

G 1X
(4vCPU and 16GB RAM) ▼

Automatically scale the number of workers

AWS Glue will optimize costs and resource usage by dynamically scaling the number of workers up and down throughout the job run. Requires Glue 3.0 or later.

8. 시각적 객체 탭으로 이동합니다.
9. 더하기 아이콘을 클릭합니다. 검색 창에 Kinesis를 입력합니다. Amazon Kinesis 데이터 소스를 선택합니다.



10. 데이터 소스 속성 - Kinesis Stream 탭에서 Amazon Kinesis 소스에 대한 스트림 세부 정보를 선택합니다.
11. 데이터 스트림의 위치에서 스트림이 내 계정에 있음을 선택합니다.
12. 사용 중인 리전을 선택합니다.
13. GlueStreamTest-{AWS::AccountId} 스트림을 선택합니다.
14. 다른 모든 설정은 기본값으로 유지합니다.

Data source properties - Kinesis Stream
Output schema
Data preview

Name

Amazon Kinesis

Amazon Kinesis Source [Info](#)

Stream details

Data Catalog table

Location of data stream

Stream is located in my account

Stream is located in another account

Region

US East (Ohio) us-east-2 ▼

Stream name [Info](#)

GlueStreamTest-

C

Data format

JSON ▼

Starting position

Select the position where the job will start reading from the input stream.

Earliest
Start reading from the oldest available record in the stream.
▼

Window size [Info](#)

Enter the time in seconds spent between batch calls.

100

15데이터 미리 보기 탭으로 이동합니다.

16KDG에서 생성된 모의 데이터를 미리 보는 데이터 미리 보기 세션 시작을 클릭합니다. AWS Glue 스트리밍 작업에 대해 이전에 생성한 Glue 서비스 역할을 선택합니다.

미리 보기 데이터가 표시되는 데 30~60초 걸립니다. 표시할 데이터 없음이 표시되면 기어 아이콘을 클릭하고 샘플링할 행 수를 100으로 변경합니다.

다음과 같이 샘플 데이터를 확인할 수 있습니다.

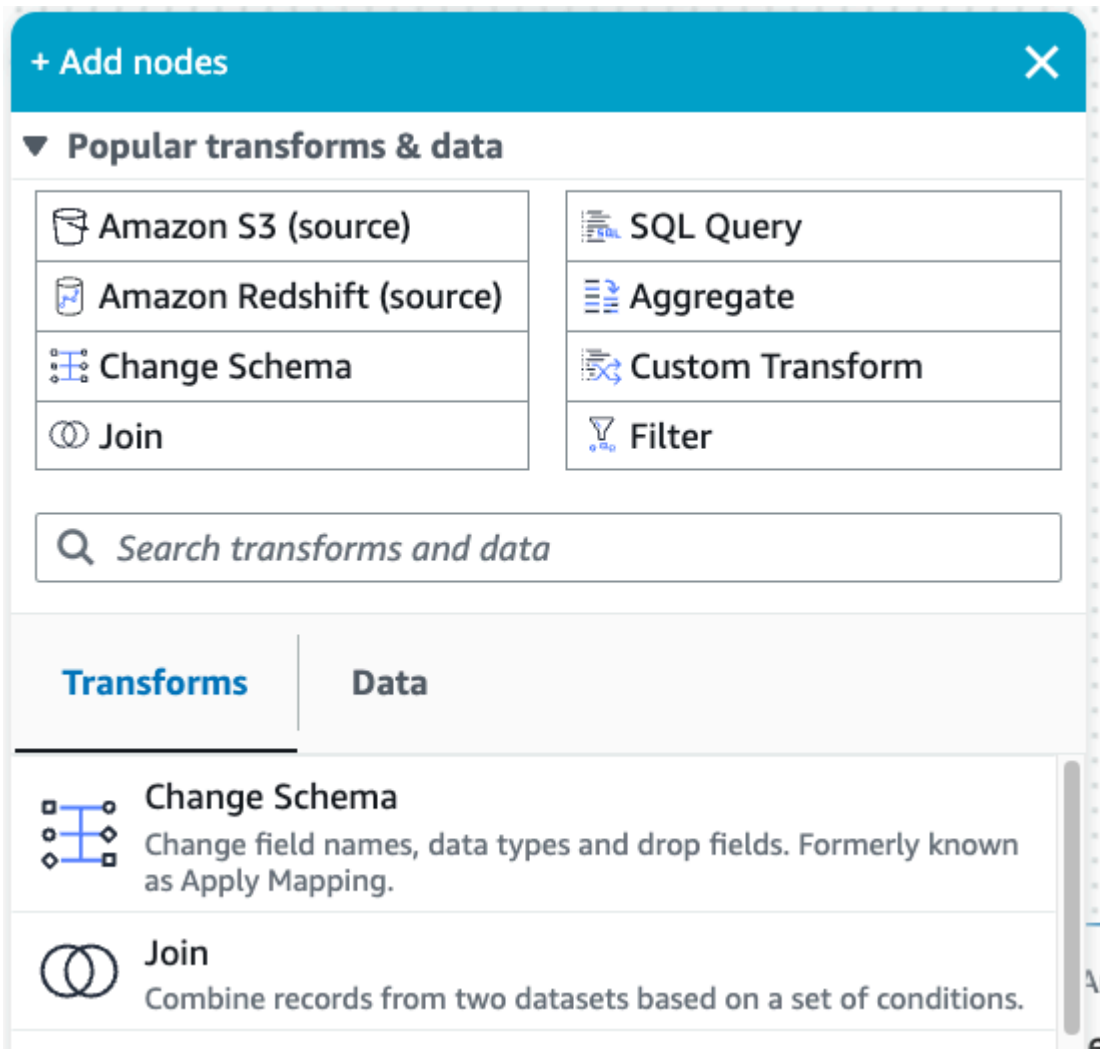
Data source properties - Kinesis Stream		Output schema	Data preview				
Data preview (100) Info			Previewing 7 of 7 fields				
<input type="text" value="Filter sample dataset"/>							
eventtime	manufacturer	minutevolume	o2stats	pressurecontrol	serialnumber	ventilatorid	
2023-06-26 14:25:37	Vyaire	5	95	7	9e79ae66-33a7-48e5-ab78-a61271199d5d	92	
2023-06-26 14:25:37	3M	5	98	17	cfb845ca-b513-4c27-9543-74dd222fc537	10	
2023-06-26 14:25:37	GE	8	98	23	90ba966c-6676-4567-a584-e267e714e57d	37	
2023-06-26 14:25:37	Vyaire	8	92	16	77f78f41-be24-47dc-b25c-05428bd76a0b	56	
2023-06-26 14:25:37	Getinge	6	92	23	ddf7b9e1-d0f7-4381-8aea-06a934583f5c	28	
2023-06-26 14:25:37	Getinge	5	92	6	c3ca9991-9b97-43e7-a866-59acbc6c5b17	84	
2023-06-26 14:25:37	3M	8	98	21	93c49e41-868b-4b5b-b725-06b4b1fb0a09	68	
2023-06-26 14:25:37	Vyaire	8	92	18	e46abe8d-b02f-43e6-91bf-c4700719f846	10	
2023-06-26 14:25:37	Vyaire	8	93	16	b3946e38-6292-4afd-8695-ada5cc09d0dd	15	
2023-06-26 14:25:37	GE	8	93	10	e3f7390d-1e68-4def-9dae-5c98b1d85d9d	3	
2023-06-26 14:25:37	Vyaire	8	98	17	a3917233-fe7f-4105-8728-779bd7ab1379	8	
2023-06-26 14:25:37	Getinge	8	98	16	06a8e8ff-cae4-4438-9714-33324f1524c9	93	
2023-06-26 14:25:37	Getinge	6	96	14	7af06237-bbdf-4615-b9ac-05d05d484ba0	13	
2023-06-26 14:25:37	3M	8	93	8	bf9985f6-04b8-442b-b7f9-24b1db6b5a37	81	
2023-06-26 14:25:37	Getinge	6	97	28	e67f4220-3070-4951-b4e0-c86b7489de10	19	
2023-06-26 14:25:37	3M	6	92	15	77954206-535e-4ef8-a1fe-0da5ece049a6	31	
2023-06-26 14:25:37	Vyaire	7	94	25	81303a43-6206-46cb-851f-fc3986491bf9	32	

출력 스키마 탭에서 추론된 스키마를 확인할 수도 있습니다.

Data source properties - Kinesis Stream		Output schema	Data preview
Schema Info			
Key			Data type
eventtime			string
manufacturer			string
minutevolume			long
o2stats			long
pressurecontrol			long
serialnumber			string
ventilatorid			long

변환 수행 및 Amazon S3에 변환된 결과 저장

1. 소스 노드를 선택한 상태에서 왼쪽 상단의 더하기 아이콘을 클릭하여 변환 단계를 추가합니다.
2. 스키마 변경 단계를 선택합니다.



3. 이 단계에서 필드 이름을 바꾸고 필드의 데이터 형식을 변환할 수 있습니다. o2stats 열의 이름을 OxygenSaturation으로 바꾸고 모든 long 데이터 형식을 int로 변환합니다.

Transform
Output schema
Data preview

Name

Node parents
Choose which nodes will provide inputs for this one.

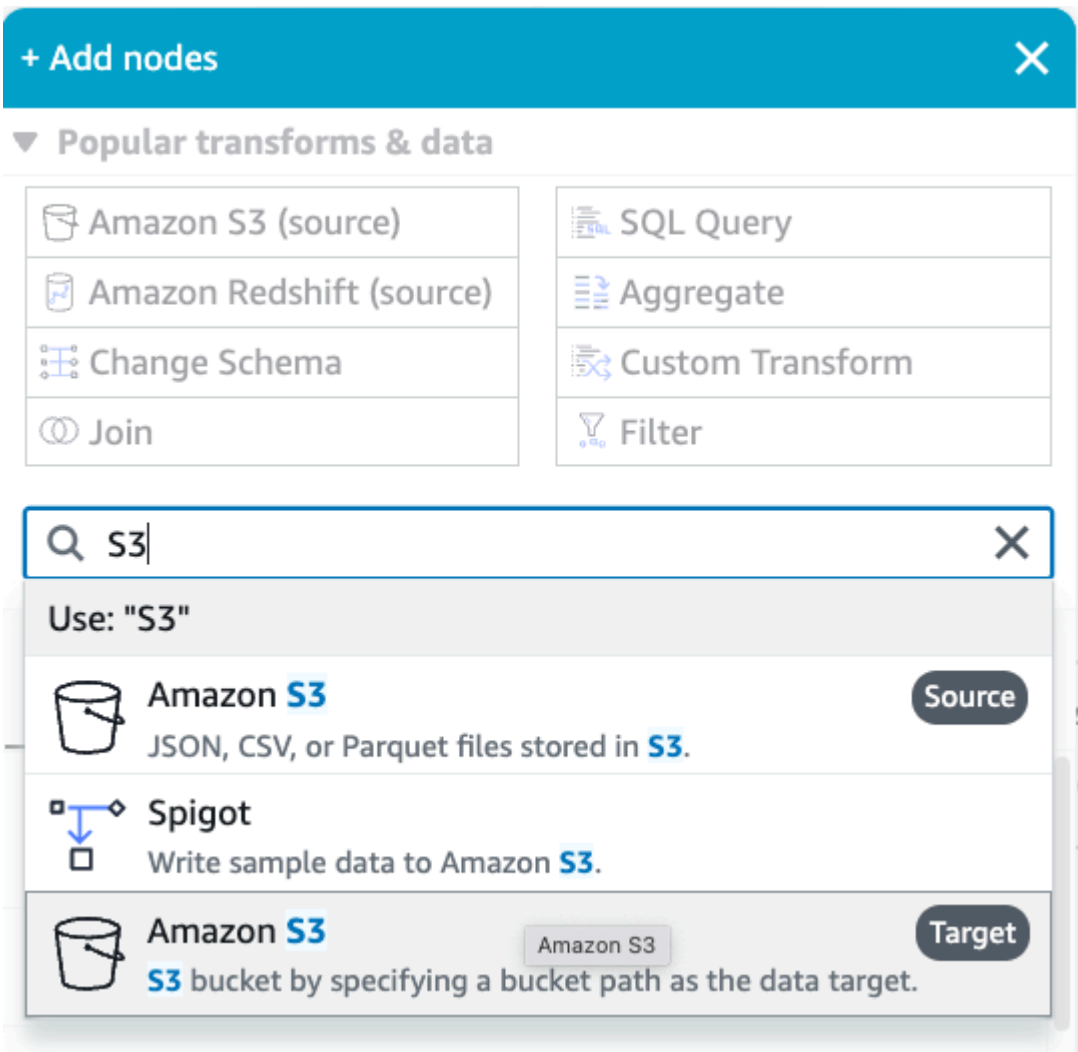
Amazon Kinesis ✕

Kinesis - DataSource

Change Schema (Apply mapping)

Source key	Target key	Data type	Drop
eventtime	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="eventtime"/>	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="string"/> ▼	<input type="checkbox"/>
manufacturer	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="manufacturer"/>	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="string"/> ▼	<input type="checkbox"/>
minutevolume	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="minutevolume"/>	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="int"/> ▼	<input type="checkbox"/>
o2stats	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="OxygenSaturation"/>	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="int"/> ▼	<input type="checkbox"/>
pressurecontrol	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="pressurecontrol"/>	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="int"/> ▼	<input type="checkbox"/>
serialnumber	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="serialnumber"/>	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="string"/> ▼	<input type="checkbox"/>
ventilatorid	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="ventilatorid"/>	<input style="width: 100%; border: 1px solid #ccc;" type="text" value="int"/> ▼	<input type="checkbox"/>

4. 더하기 아이콘을 클릭하여 Amazon S3 대상을 추가합니다. 검색 상자에 S3를 입력하고 Amazon S3 - 대상 변환 단계를 선택합니다.



5. 대상 파일 형식으로 Parquet을 선택합니다.
6. 압축 유형으로 Snappy를 선택합니다.
7. CloudFormation 템플릿 streaming-tutorial-s3-target-{AWS::AccountId}로 생성된 S3 대상 위치를 입력합니다.
8. 데이터 카탈로그에 테이블 생성, 후속 실행 시 기존 스키마 유지 및 새 파티션 추가를 선택합니다.
9. Amazon S3 대상 테이블의 스키마를 저장할 대상 데이터베이스와 테이블 이름을 입력합니다.

Name

Amazon S3

Node parents

Choose which nodes will provide inputs for this one.

Choose one or more parent node

Change Schema ✕
ApplyMapping - Transform

Format

Parquet

Compression Type

Snappy

S3 Target Location

Choose an S3 location in the format s3://bucket/prefix/object/ with a trailing slash (/).



View

Browse S3

Data Catalog update options [Info](#)

Choose how you want to update the Data Catalog table's schema and partitions. These options will only apply if the Data Catalog table is an S3 backed source.

- Do not update the Data Catalog
- Create a table in the Data Catalog and on subsequent runs, update the schema and add new partitions
- Create a table in the Data Catalog and on subsequent runs, keep existing schema and add new partitions

Database

Choose the database from the AWS Glue Data Catalog.

demo

**▶ Use runtime parameters**

Table name

Enter a table name for the AWS Glue Data Catalog.

demo_stream_transform_result

10스크립트 탭을 클릭하여 생성된 코드를 봅니다.

11오른쪽 상단의 저장을 클릭하여 ETL 코드를 저장한 다음, 실행을 클릭하여 AWS Glue 스트리밍 작업을 시작합니다.

실행 탭에서 실행 상태를 확인할 수 있습니다. 작업을 3~5분간 실행한 후 중지합니다.

Visual	Script	Job details	Runs	Data quality <small>New</small>	Schedules	Version Control
Job runs (1/1) Info						
<input type="text" value="Filter job runs by property"/>						
Run status	Retry	Start time	End time	Duration		
● ● Running	0	06/26/2023 15:58:05	-	35 s		

12 Amazon Athena에서 생성된 새 테이블을 확인합니다.

● Query 9 +

```
1 select * from "demo_stream_transform_result"
```

SQL Ln 1, Col 45 ≡ ☒ ⚙

Run again
Explain
Cancel
Clear
Create

 Reuse query results
 up to 60 minutes ago

Query results Query stats

Completed
 Time in queue: 137 ms Run time: 894 ms Data scanned: 91.59 KB

Results (2,200)
Copy
Download results

< 1 ... > ⚙

#	eventtime	manufacturer	minutevolume	oxygensaturation	pressurecontrol	serialnumber	ventilatorid	ingest_year	ingest_month	ingest_day
13	2023-06-26 16:03:24	Vyair	6	98	10	8e438321-3bee-423f-9bcd-c693ee475868	91	2023	06	26
17	2023-06-26 16:03:24	3M	5	98	17	a7bcb332-6c52-489e-9a55-c923f3f650d2	64	2023	06	26
19	2023-06-26 16:03:24	Getinge	7	98	24	871a5ed3-4912-4b51-8428-5cb3e1d0034a	30	2023	06	26
27	2023-06-26 16:04:24	Vyair	8	98	8	5e4eeeba-29bb-4add-9013-2307c640b09e	94	2023	06	26
29	2023-06-26 16:04:24	3M	7	98	26	69443bbd-f347-419a-97d0-912cb88b36eb	3	2023	06	26
31	2023-06-26 16:04:24	3M	7	98	16	9d6242e6-7f57-48a4-bbb6-3e1b954454be	8	2023	06	26

자습서: AWS Glue Studio 노트북을 사용하여 첫 번째 스트리밍 워크로드 구축

이 자습서에서는 AWS Glue Studio 노트북을 활용하여 실시간에 가까운 데이터 처리를 위해 ETL 작업을 대화형으로 구축하고 개선하는 방법을 살펴보겠습니다. AWS Glue를 처음 사용하거나 기술 세트를 향상시키고자 할 때 이 가이드는 AWS Glue 대화형 세션 노트북의 잠재력을 최대한 활용할 수 있도록 프로세스를 안내합니다.

AWS Glue 스트리밍을 사용하면 지속적으로 실행되고 Amazon Kinesis Data Streams, Apache Kafka, Amazon Managed Streaming for Apache Kafka(Amazon MSK) 등의 스트리밍 소스의 데이터를 소비하는 스트리밍 추출, 전환, 적재(ETL) 작업을 생성할 수 있습니다.

필수 조건

이 자습서를 따르려면 AWS Glue, Amazon Kinesis, Amazon S3, Amazon Athena, AWS CloudFormation, AWS Lambda 및 Amazon Cognito를 사용할 수 있는 AWS 콘솔 권한이 있는 사용자가 필요합니다.

Amazon Kinesis의 스트리밍 데이터 소비

주제

- [Kinesis 데이터 생성기로 모의 데이터 생성](#)
- [AWS Glue Studio로 AWS Glue 스트리밍 작업 생성](#)
- [정리](#)
- [결론](#)

Kinesis 데이터 생성기로 모의 데이터 생성

Note

이전 [자습서: AWS Glue Studio를 사용하여 첫 번째 스트리밍 워크로드 구축](#)을 완료한 경우 계정에 Kinesis Data Generator가 이미 설치되어 있으므로 아래의 1~8단계를 건너뛰고 [AWS Glue Studio로 AWS Glue 스트리밍 작업 생성](#) 섹션으로 이동할 수 있습니다.

Kinesis 데이터 생성기(KDG)를 사용하여 JSON 형식의 샘플 데이터를 종합적으로 생성할 수 있습니다. [도구 설명서](#)에서 전체 지침과 세부 정보를 찾아볼 수 있습니다.

1. 시작하려면



클릭하여 AWS 환경에서 AWS CloudFormation 템플릿을 실행합니다.

Note

Kinesis 데이터 생성기용 Amazon Cognito 사용자와 같은 일부 리소스가 AWS 계정에 이미 존재하기 때문에 CloudFormation 템플릿 오류가 발생할 수 있습니다. 다른 자습서나 블로그에서 이미 설정했기 때문일 수 있습니다. 이 문제를 해결하려면 새 AWS 계정에서 템플릿을 사용해 새로 시작하거나 다른 AWS 리전을 탐색해 봅니다. 이러한 옵션을 사용하면 기존 리소스와 충돌하지 않고 자습서를 실행할 수 있습니다.

템플릿은 Kinesis 데이터 스트림과 Kinesis Data Generator 계정을 프로비저닝합니다.

2. KDG가 인증에 사용할 사용자 이름과 암호를 입력합니다. 나중에 사용할 수 있도록 사용자 이름과 암호를 기록해 둡니다.
3. 마지막 단계까지 다음을 선택합니다. IAM 리소스 생성을 확인합니다. 화면 상단에 최소 요구 사항을 충족하지 않는 암호와 같은 오류가 있는지 확인하고 템플릿을 배포합니다.
4. 스택의 출력 탭으로 이동합니다. 템플릿이 배포되면 생성된 속성 `KinesisDataGeneratorUrl`이 표시됩니다. 해당 URL을 클릭합니다.
5. 기록해 둔 사용자 이름과 암호를 입력합니다.
6. 사용 중인 리전을 선택하고 Kinesis 스트림 `GlueStreamTest-{AWS::AccountId}`를 선택합니다.
7. 다음 템플릿을 입력합니다.

```
{
  "ventilatorid": "{{random.number(100)}}",
  "eventtime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
  "serialnumber": "{{random.uuid}}",
  "pressurecontrol": {{random.number(
    {
      "min":5,
      "max":30
    }
  )}},
  "o2stats": {{random.number(
    {
      "min":92,
      "max":98
    }
  )}},
}
```

```

"minutevolume": {{random.number(
  {
    "min":5,
    "max":8
  }
)}}},
"manufacturer": "{{random.arrayElement(
  ["3M", "GE", "Vyair", "Getinge"]
)}}}"
}
    
```

이제 테스트 템플릿을 사용하여 모의 데이터를 보고 데이터 전송을 사용하여 모의 데이터를 Kinesis 로 모을 수 있습니다.

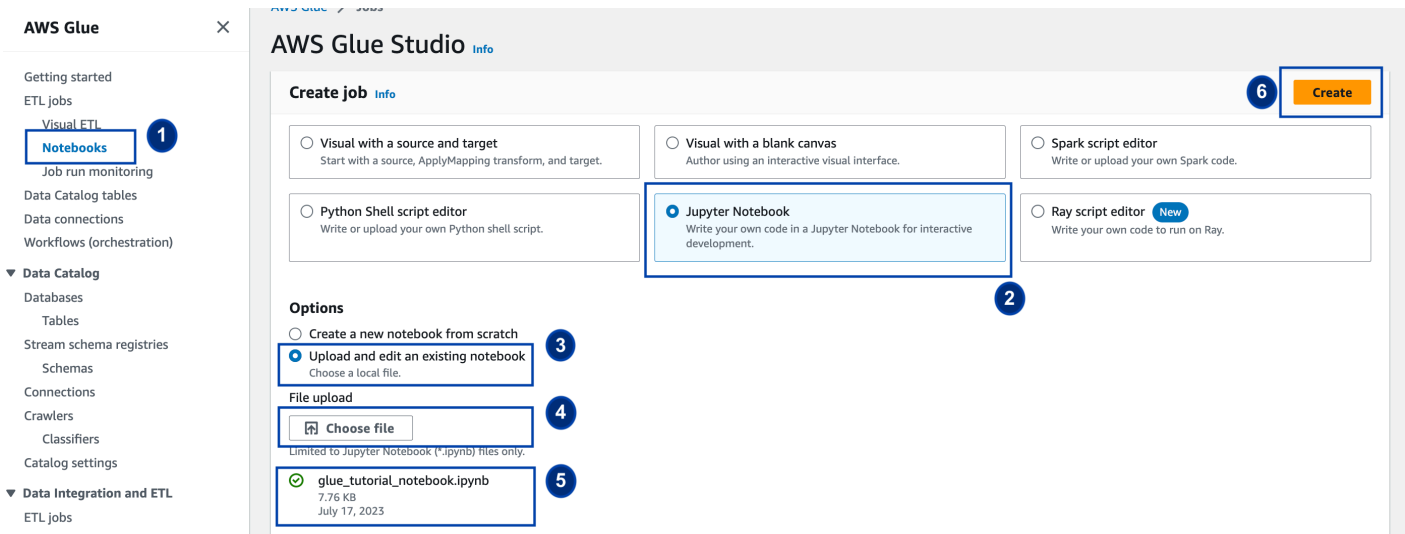
8. 데이터 전송을 클릭하고 Kinesis에 5~10,000개의 레코드를 생성합니다.

AWS Glue Studio로 AWS Glue 스트리밍 작업 생성

AWS Glue Studio는 데이터 통합 파이프라인의 설계, 오케스트레이션 및 모니터링 프로세스를 간소화 하는 시각적 인터페이스입니다. 이를 통해 사용자는 광범위한 코드를 작성하지 않고도 데이터 변환 파이프라인을 구축할 수 있습니다. AWS Glue Studio에는 시각적 작업 작성 환경 외에도 이 자습서의 나머지 부분에서 사용할 AWS Glue 대화형 세션에서 지원하는 Jupyter Notebook도 포함되어 있습니다.

AWS Glue 스트리밍 대화형 세션 작업 설정

1. 제공된 [노트북 파일](#)을 다운로드하여 로컬 디렉토리에 저장합니다.
2. AWS Glue 콘솔을 열고 왼쪽 창에서 노트북 > Jupyter Notebook > 기존 노트북 업로드 및 편집을 클릭합니다. 이전 단계의 노트북을 업로드하고 생성을 클릭합니다.



3. 작업 이름, 역할을 제공하고 기본 Spark 커널을 선택합니다. 그런 다음, 노트북 시작을 클릭합니다. IAM 역할로 CloudFormation 템플릿에서 프로비저닝한 역할을 선택합니다. CloudFormation의 출력 탭에서 이를 확인할 수 있습니다.

AWS Glue > Notebook setup

Notebook setup [Info](#)

Initial configuration

Job name
Enter a name for the job. This name will be used for the script and the notebook file.

IAM Role
Role assumed by the job with permission to access your data stores. Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job.

Kernel
The kernel with which the notebook will be created.

노트북에는 자습서를 계속하는 데 필요한 모든 지침이 있습니다. 노트북의 지침을 실행하거나 이 자습서를 따라 작업 개발을 계속할 수 있습니다.

노트북 셀을 실행합니다.

1. (선택 사항) 첫 번째 코드 셀인 %help는 사용 가능한 모든 노트북 매직을 나열합니다. 지금은 이 셀을 건너뛰어도 되지만 자유롭게 탐색해 봅니다.
2. 다음 코드 블록 %streaming으로 시작합니다. 이 매직은 작업 유형을 스트리밍으로 설정하여 AWS Glue 스트리밍 ETL 작업을 개발, 디버깅 및 배포할 수 있게 합니다.
3. 다음 셀을 실행하여 AWS Glue 대화형 세션을 생성합니다. 출력 셀에는 세션 생성을 확인하는 메시지가 있습니다.

Run this cell to set up and start your interactive session.

```
[1]: %glue_version 3.0

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue import DynamicFrame
from datetime import datetime
from pyspark.sql.types import StructType, StructField, StringType, LongType
from pyspark.sql.functions import lit,col,from_json
import boto3

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)

Setting Glue version to: 3.0
Authenticating with environment variables and user-defined glue_role_arn: arn:aws:iam::6[redacted]:role/glue-tutorial-role[redacted]
Trying to create a Glue session for the kernel.
Worker Type: G.1X
Number of Workers: 5
Session ID: [redacted]af
Job Type: gluestreaming
Applying the following default arguments:
--glue_kernel_version 0.37.3
--enable-glue-datacatalog true
Waiting for session 4[redacted] to get into ready status...
Session 48 [redacted] has been created.
```

4. 다음 셀은 변수를 정의합니다. 값을 작업에 적합한 값으로 바꾸고 셀을 실행합니다. 예:

```
output_database_name="default"
output_table_name="test_stream_001"

account_id = boto3.client("sts").get_caller_identity()["Account"]
region_name=boto3.client('s3').meta.region_name
stream_arn_name = "arn:aws:kinesis:{{region_name}}:{{account_id}}:stream/GlueStreamTest-{{account_id}}"
s3_bucket_name = "streaming-tutorial-s3-target-{{account_id}}"

output_location = "s3://{{s3_bucket_name}}/streaming_output/"
checkpoint_location = "s3://{{s3_bucket_name}}/checkpoint_location/"
```

5. 데이터가 이미 Kinesis Data Streams로 스트리밍되고 있으므로 다음 셀은 스트림의 결과를 소비합니다. 다음 셀을 실행합니다. 인쇄 문이 없으므로 이 셀에서 예상되는 출력이 없습니다.
6. 다음 셀에서는 샘플 세트를 가져와서 수신 스트림을 탐색하고 해당 스키마와 실제 데이터를 인쇄합니다. 예:

Sample and print the incoming records

the sampling is for debugging purpose. You may comment off the entire code cell below, before deploying the actual code

```
[4]: options = {
  --- "pollingTimeInMs": "20000",
  --- "windowSize": "5 seconds"
}
sampled_dynamic_frame = glueContext.getSampleStreamingDynamicFrame(data_frame, options, None)

count_of_sampled_records = sampled_dynamic_frame.count()

print(count_of_sampled_records)

sampled_dynamic_frame.printSchema()

sampled_dynamic_frame.toDF().show(10, False)
```

```
100
root
```

```
|-- eventtime: string
|-- manufacturer: string
|-- minutevolume: long
|-- o2stats: long
|-- pressurecontrol: long
|-- serialnumber: string
|-- ventilatorid: long
```

eventtime	manufacturer	minutevolume	o2stats	pressurecontrol	serialnumber	ventilatorid
2023-07-18 10:20:11	3M	6	92	24	a3e860ba-24b9-41c4-bc10-91c6b35e1406	6
2023-07-18 10:20:11	Vyair	6	95	6	96101dca-3e88-457f-b390-e3291df48a81	26
2023-07-18 10:20:12	Getinge	8	96	24	18f3d448-1dee-4c80-835b-1a0daa818915	22
2023-07-18 10:20:12	Getinge	7	98	30	25f425cd-b978-4953-9a03-4d607a639364	91
2023-07-18 10:20:12	GE	5	93	25	2cd7cdc2-f5f5-4ff2-ae32-45e5a8922d53	93

7. 다음으로 실제 데이터 변환 로직을 정의합니다. 셀은 모든 마이크로 배치 중 트리거되는 processBatch 메서드로 구성됩니다. 셀을 실행합니다. 높은 수준에서 수신 스트림에 대해 다음을 수행합니다.
 - a. 입력 열의 하위 집합을 선택합니다.
 - b. 열 이름을 o2stats에서 oxygen_stats로 바꿉니다.
 - c. 새 열(serial_identifer, ingest_year, ingest_month 및 ingest_day)을 파생합니다.
 - d. 결과를 Amazon S3 버킷에 저장하고 분할된 AWS Glue 카탈로그 테이블도 생성합니다.
8. 마지막 셀에서는 10초마다 프로세스 배치를 트리거합니다. 셀을 실행하고 Amazon S3 버킷과 AWS Glue 카탈로그 테이블이 채워질 때까지 약 30초간 기다립니다.
9. 마지막으로 Amazon Athena 쿼리 편집기를 사용하여 저장된 데이터를 찾아봅니다. 이름이 바뀐 열과 새 파티션도 볼 수 있습니다.

```
1 select * from test_stream_001 limit 10
```

SQL Ln 1, Col 39

Run again Explain Cancel Clear Create

Completed Time in queue: 164 ms Run time: 1.22 sec Data scanned: 11.76 KB

Results (10) Copy Download results

Search rows

Time	Manufacturer	Oxygen Stats	Serial Number	Ventilator ID	Serial Identifier	Ingest Year	Ingest Month	Ingest Day
7-18 14:08:12	GE	96	a28895a3-0d57-4d0e-9d5e-86fdc92a5ba8	54	a28895a3	2023	7	18
7-18 14:08:12	Getinge	93	1e7b6e7e-e248-4cc7-971c-7cc7f4bb53e9	94	1e7b6e7e	2023	7	18
7-18 14:08:12	GE	97	52f8b540-4baa-4b90-bc65-986d668e8174	42	52f8b540	2023	7	18
7-18 14:08:12	Vyaire	93	e4ebdf4a-ca96-4465-ba03-681b438d9589	14	e4ebdf4a	2023	7	18
7-18 14:08:12	GE	92	52ba9e2b-748f-4226-9ac0-3767ce900233	33	52ba9e2b	2023	7	18
7-18 14:08:12	Getinge	96	74922910-ddcd-4e03-899b-acdf7487bb6c	8	74922910	2023	7	18

노트북에는 자습서를 계속하는 데 필요한 모든 지침이 있습니다. 노트북의 지침을 실행하거나 이 자습서를 따라 작업 개발을 계속할 수 있습니다.

AWS Glue 작업을 저장하고 실행합니다.

대화형 세션 노트북을 사용하여 애플리케이션 개발 및 테스트가 완료되면 노트북 인터페이스 상단에 있는 저장을 클릭합니다. 저장한 후에는 애플리케이션을 작업으로 실행할 수도 있습니다.

glue_tutorial_notebook

Stop notebook Download Notebook Actions Save Run

Notebook Script Job details Runs Data quality New Schedules Version Control

Code Download

Glue PySpark

AWS Glue Streaming Tutorials - Working with Studio Notebook

정리

계정에 추가 요금이 발생하지 않도록 하려면 지침의 일부로 시작한 스트리밍 작업을 중지합니다. 노트북을 중지하면 됩니다. 그러면 세션이 종료됩니다. Amazon S3 버킷을 비우고 이전에 프로비저닝한 AWS CloudFormation 스택을 삭제합니다.

결론

이 자습서에서는 AWS Glue Studio Notebook을 사용하여 다음을 수행하는 방법을 시연했습니다.

- 노트북을 사용하여 스트리밍 ETL 작업 작성
- 수신 데이터 스트림 미리 보기
- AWS Glue 작업을 게시할 필요 없이 코딩 및 문제 수정
- 엔드 투 엔드 작업 코드를 검토하고, 디버깅을 제거하고, 노트북에서 명령문이나 셀을 인쇄합니다.
- 코드를 AWS Glue 작업으로 게시합니다.

이 자습서의 목표는 AWS Glue 스트리밍 및 대화형 세션을 사용한 실습 경험을 제공하는 것입니다. 이를 개별 AWS Glue 스트리밍 사용 사례에 대한 참조로 사용하는 것이 좋습니다. 자세한 내용은 [AWS Glue 대화형 세션 시작하기](#) 단원을 참조하십시오.

AWS Glue 스트리밍 개념

다음 섹션에서는 AWS Glue 스트리밍의 개념에 대한 정보를 제공합니다.

주제

- [AWS Glue 스트리밍 작업의 구조](#)
- [Kafka 연결](#)
- [Kinesis 연결](#)
- [AWS Glue 스트리밍 옵션](#)

AWS Glue 스트리밍 작업의 구조

AWS Glue 스트리밍 작업은 Spark 스트리밍 패러다임에서 작동하며 Spark 프레임워크의 구조화된 스트리밍을 활용합니다. 스트리밍 작업은 특정 시간 간격으로 스트리밍 데이터 소스를 지속적으로 폴링

하여 레코드를 마이크로 배치로 가져옵니다. 다음 섹션에서는 AWS Glue 스트리밍 작업의 여러 부분을 살펴봅니다.

```
def processBatch(data_frame, batchId):
    if data_frame.count() > 0:
        AmazonKinesis_node1696872487972 = DynamicFrame.fromDF(
            glueContext.add_ingestion_time_columns(data_frame, "hour"),
            glueContext,
            "from_data_frame",
        )
        # Script generated for node Change Schema
        ChangeSchema_node1696872679326 = ApplyMapping.apply(
            frame=AmazonKinesis_node1696872487972,
            mappings=[
                ("eventtime", "string", "eventtime", "string"),
                ("manufacturer", "string", "manufacturer", "string"),
                ("minutevolume", "long", "minutevolume", "int"),
                ("o2stats", "long", "OxygenSaturation", "int"),
                ("pressurecontrol", "long", "pressurecontrol", "int"),
                ("serialnumber", "string", "serialnumber", "string"),
                ("ventilatorid", "long", "ventilatorid", "long"),
                ("ingest_year", "string", "ingest_year", "string"),
                ("ingest_month", "string", "ingest_month", "string"),
                ("ingest_day", "string", "ingest_day", "string"),
                ("ingest_hour", "string", "ingest_hour", "string"),
            ],
            transformation_ctx="ChangeSchema_node1696872679326",
        )
        # Script generated for node Amazon S3
        AmazonS3_node1696872743449_path = (
            "s3://streaming-tutorial-s3-target-"
        )
        AmazonS3_node1696872743449 = glueContext.getSink(
            path=AmazonS3_node1696872743449_path,
            connection_type="s3",
            update_behavior="UPDATE_IN_DATABASE",
            partition_keys=["ingest_year", "ingest_month", "ingest_day", "ingest_hour"],
            compression="snappy",
            enable_update_catalog=True,
            transformation_ctx="AmazonS3_node1696872743449",
        )
        AmazonS3_node1696872743449.setCatalogInfo(
            catalog_database="demo", catalog_table_name="demo_stream_transform_result"
        )
        AmazonS3_node1696872743449.setFormat("glueparquet")
        AmazonS3_node1696872743449.writeFrame(ChangeSchema_node1696872679326)

    glueContext.forEachBatch(
        frame=dataFrame_AmazonKinesis_node1696872487972,
        batch_function=processBatch,
        options={
            "windowSize": "100 seconds",
            "checkpointLocation": args["TempDir"] + "/" + args["JOB_NAME"] + "/checkpoint/",
        },
    ),
}
job.commit()
```

2

3

4

5

6

1 ← Entry Point

forEachBatch

forEachBatch 메서드는 AWS Glue 스트리밍 작업 실행의 진입점입니다. AWS Glue 스트리밍 작업은 forEachBatch 메서드를 사용하여 스트리밍 작업의 수명 주기 동안 활성 상태로 유지되는 반복자처럼 작동하는 데이터를 풀링하고 정기적으로 새 데이터에 대한 스트리밍 소스를 풀링하고 최신 데이터를 마이크로 배치로 처리합니다.

```
glueContext.forEachBatch(
    frame=dataFrame_AmazonKinesis_node1696872487972,
    batch_function=processBatch,
    options={
        "windowSize": "100 seconds",
        "checkpointLocation": args["TempDir"] + "/" + args["JOB_NAME"] + "/
checkpoint/",
    },
)
```

forEachBatch의 frame 속성을 구성하여 스트리밍 소스를 지정합니다. 이 예제에서는 작업을 생성하는 동안 빈 캔버스에서 생성한 소스 노드가 작업의 기본 DataFrame으로 채워집니다. 각 마이크로 배치 작업에 대해 간접적으로 호출하기로 결정한 function으로 batch_function 속성을 설정합니다. 수신 데이터에 대한 배치 변환을 처리할 함수를 정의해야 합니다.

소스

processBatch 함수의 첫 번째 단계에서 프로그램은 forEachBatch의 프레임 속성으로 정의한 DataFrame의 레코드 수를 확인합니다. 프로그램은 비어 있지 않은 DataFrame에 모으기 타임스탬프를 추가합니다. data_frame.count()>0 절은 최신 마이크로 배치가 비어 있지 않고 추가 처리를 위한 준비가 되었는지 여부를 결정합니다.

```
def processBatch(data_frame, batchId):
    if data_frame.count() >0:
        AmazonKinesis_node1696872487972 = DynamicFrame.fromDF(
            glueContext.add_ingestion_time_columns(data_frame, "hour"),
            glueContext,
            "from_data_frame",
        )
```

Mapping

프로그램의 다음 섹션은 매핑을 적용하는 것입니다. Spark DataFrame의 Mapping.apply 메서드를 사용하면 데이터 요소에 대한 변환 규칙을 정의할 수 있습니다. 일반적으로 이름을 바꾸거나, 데이터 형식을 변경하거나, 소스 데이터 열에 사용자 지정 함수를 적용하고 이를 대상 열에 매핑할 수 있습니다.

```
#Script generated for node ChangeSchema
ChangeSchema_node16986872679326 = ApplyMapping.apply(
    frame = AmazonKinesis_node1696872487972,
    mappings = [
        ("eventtime", "string", "eventtime", "string"),
        ("manufacturer", "string", "manufacturer", "string"),
        ("minutevolume", "long", "minutevolume", "int"),
        ("o2stats", "long", "OxygenSaturation", "int"),
        ("pressurecontrol", "long", "pressurecontrol", "int"),
```

```

        ("serialnumber", "string", "serialnumber", "string"),
        ("ventilatorid", "long", "ventilatorid", "long"),
        ("ingest_year", "string", "ingest_year", "string"),
        ("ingest_month", "string", "ingest_month", "string"),
        ("ingest_day", "string", "ingest_day", "string"),
        ("ingest_hour", "string", "ingest_hour", "string"),
    ],
    transformation_ctx="ChangeSchema_node16986872679326",
)
)

```

Sink

이 섹션에서는 스트리밍 소스에서 수신 데이터 세트가 대상 위치에 저장됩니다. 이 예제에서는 Amazon S3 위치에 데이터를 씁니다. 캔버스에서 작업을 생성할 때 사용한 설정에 따라 AmazonS3_node_path 속성 세부 정보가 미리 채워집니다. 사용 사례에 따라 updateBehavior를 설정할 수 있습니다. 그런 다음, 데이터 카탈로그 테이블을 업데이트하지 않거나, 데이터 카탈로그를 생성하고 후속 실행 시 데이터 카탈로그 스키마를 업데이트하거나 카탈로그 테이블을 생성하고 후속 실행 시 스키마 정의를 업데이트하지 않도록 결정할 수 있습니다.

partitionKeys 속성은 스토리지 파티션 옵션을 정의합니다. 기본 동작은 소스 섹션에서 사용할 수 있게 된 ingestion_time_columns 단위로 데이터를 분할하는 것입니다. compression 속성을 사용하면 대상 쓰기 중 적용할 압축 알고리즘을 설정할 수 있습니다. Snappy, LZO 또는 GZIP을 압축 기법으로 설정할 수 있는 옵션이 있습니다. enableUpdateCatalog 속성은 AWS Glue 카탈로그 테이블을 업데이트해야 하는지 여부를 제어합니다. 이 속성에 사용할 수 있는 옵션은 True 또는 False입니다.

```

#Script generated for node Amazon S3
AmazonS3_node1696872743449 = glueContext.getSink(
    path = AmazonS3_node1696872743449_path,
    connection_type = "s3",
    updateBehavior = "UPDATE_IN_DATABASE",
    partitionKeys = ["ingest_year", "ingest_month", "ingest_day", "ingest_hour"],
    compression = "snappy",
    enableUpdateCatalog = True,
    transformation_ctx = "AmazonS3_node1696872743449",
)

```

AWS Glue 카탈로그 싱크

이 작업 섹션은 AWS Glue 카탈로그 테이블 업데이트 동작을 제어합니다. AWS Glue 카탈로그 데이터베이스 이름과 설계 중인 AWS Glue 작업과 연결된 테이블 이름에 따라 `catalogDatabase` 및 `catalogTableName` 속성을 설정합니다. `setFormat` 속성을 통해 대상 데이터의 파일 형식을 정의할 수 있습니다. 이 예제에서는 데이터를 Parquet 형식으로 저장합니다.

이 자습서를 참조하여 AWS Glue 스트리밍 작업을 설정하고 실행하면 Amazon Kinesis Data Streams에서 생성된 스트리밍 데이터가 빠른 압축을 통해 Parquet 형식으로 Amazon S3 위치에 저장됩니다. 스트리밍 작업이 성공적으로 실행되면 Amazon Athena를 통해 데이터를 쿼리할 수 있습니다.

```
AmazonS3_node1696872743449 = setCatalogInfo(
    catalogDatabase = "demo", catalogTableName = "demo_stream_transform_result"
)
AmazonS3_node1696872743449.setFormat("glueparquet")
AmazonS3_node1696872743449.writeFormat("ChangeSchema_node16986872679326")
)
```

Kafka 연결

Kafka 클러스터 또는 Amazon Managed Streaming for Apache Kafka 클러스터에 대한 연결을 지정합니다.

데이터 카탈로그 테이블에 저장된 정보를 사용하거나 데이터 스트림에 직접 액세스할 수 있는 정보를 제공하여 Kafka Data Streams에서 읽고 쓸 수 있습니다. Kafka에서 DataFrame Spark로 정보를 읽은 다음 Glue로 변환할 수 있습니다. AWS DynamicFrame JSON 형식으로 DynamicFrames Kafka에 글을 쓸 수 있습니다. 데이터 스트림에 직접 액세스하는 경우 이러한 옵션을 사용하여 데이터 스트림에 액세스하는 방법에 대한 정보를 제공합니다.

`getCatalogSource` 또는 `create_data_frame_from_catalog`를 사용하여 Kafka 스트리밍 소스에서 레코드를 소비하거나 `getCatalogSink` 또는 `write_dynamic_frame_from_catalog`를 사용하여 Kafka에 레코드를 쓰는 경우 작업에는 데이터 카탈로그 데이터베이스와 테이블 이름 정보가 있으며 이를 사용하여 Kafka 스트리밍 소스에서 읽기 위한 몇 가지 기본 파라미터를 얻을 수 있습니다. `getSource`, `getCatalogSink`, `getSourceWithFormat`, `getSinkWithFormat`, `createDataFrameFromOptions`, `create_data_frame_from_options` 또는 `write_dynamic_frame_from_catalog`를 사용하는 경우 여기에 설명된 연결 옵션을 통해 이러한 기본 파라미터를 지정해야 합니다.

GlueContext 클래스에 지정된 메서드에 대해 다음 인수를 사용하여 Kafka에 대한 연결 옵션을 지정할 수 있습니다.

- Scala
 - connectionOptions: getSource, createDataFrameFromOptions, getSink와(과) 함께 사용
 - additionalOptions: getCatalogSource, getCatalogSink와 함께 사용
 - options: getSourceWithFormat, getSinkWithFormat와 함께 사용
- Python
 - connection_options: create_data_frame_from_options, write_dynamic_frame_from_options와 함께 사용
 - additional_options: create_data_frame_from_catalog, write_dynamic_frame_from_catalog와 함께 사용
 - options: getSource, getSink와 함께 사용

스트리밍 ETL 작업에 대한 참고 및 제한 사항은 [the section called “스트리밍 ETL 참고 사항 및 제한 사항”](#) 섹션을 참조하세요.

Kafka 구성

인터넷을 통해 사용할 수 있는 AWS Kafka 스트림에 연결하기 위한 전제 조건은 없습니다.

AWS Glue Kafka 연결을 생성하여 연결 자격 증명을 관리할 수 있습니다. 자세한 정보는 [the section called “Kafka 데이터 스트림에 대한 연결 생성”](#)을 참조하세요. AWS Glue 작업 구성에서 *ConnectionName#* 추가 네트워크 연결로 제공한 다음, 메서드 호출에서 *ConnectionName# #### #* 제공하십시오. `connectionName`

경우에 따라 추가 필수 조건을 구성해야 합니다.

- IAM 인증과 함께 Amazon Managed Streaming for Apache Kafka를 사용하는 경우 적절한 IAM 구성 이 필요합니다.
- Amazon VPC 내에서 Amazon Managed Streaming for Apache Kafka를 사용하는 경우 적절한 Amazon VPC 구성 이 필요합니다. Amazon VPC 연결 정보를 제공하는 AWS Glue 연결을 생성해야 합니다. AWS Glue 연결을 추가 네트워크 연결로 포함하려면 작업 구성 이 필요합니다.

스트리밍 ETL 작업 필수 조건에 대한 자세한 내용은 [the section called “스트리밍 ETL 작업”](#) 섹션을 참조하세요.

예제: Kafka 스트림에서 읽기

[the section called “forEachBatch”](#)과(와) 함께 사용합니다.

Kafka 스트리밍 소스의 예:

```
kafka_options =
  { "connectionName": "ConfluentKafka",
    "topicName": "kafka-auth-topic",
    "startingOffsets": "earliest",
    "inferSchema": "true",
    "classification": "json"
  }
data_frame_datasource0 =
  glueContext.create_data_frame.from_options(connection_type="kafka",
  connection_options=kafka_options)
```

예: Kafka 스트림에 쓰기

Kafka에 쓰기의 예제:

getSink 메서드를 사용한 예제:

```
data_frame_datasource0 =
  glueContext.getSink(
    connectionType="kafka",
    connectionOptions={
      JsonOptions("""{
        "connectionName": "ConfluentKafka",
        "classification": "json",
        "topic": "kafka-auth-topic",
        "typeOfData": "kafka"}
      """))},
    transformationContext="dataframe_ApacheKafka_node1711729173428")
  .getDataFrame()
```

write_dynamic_frame.from_options 메서드를 사용한 예제:

```
kafka_options =
```

```

    { "connectionName": "ConfluentKafka",
      "topicName": "kafka-auth-topic",
      "classification": "json"
    }
  data_frame_datasource0 =
    glueContext.write_dynamic_frame.from_options(connection_type="kafka",
    connection_options=kafka_options)

```

Kafka 연결 옵션 참조

읽을 때는 "connectionType": "kafka"를 사용한 다음 연결 옵션을 사용합니다.

- "bootstrap.servers" (필수) 부트스트랩 서버 URL의 목록입니다(예: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094). 이 옵션은 API 호출에 지정하거나 데이터 카탈로그의 테이블 메타데이터에 정의해야 합니다.
- "security.protocol" (필수) 브로커와 통신하는 데 사용되는 프로토콜입니다. 가능한 값은 "SSL" 또는 "PLAINTEXT"입니다.
- "topicName": (필수) 선택으로 구분된 구독할 주제의 목록입니다. "topicName", "assign" 또는 "subscribePattern" 중에서 하나만 지정해야 합니다.
- "assign": (필수) 사용할 특정 TopicPartitions을 지정하는 JSON 문자열입니다. "topicName", "assign" 또는 "subscribePattern" 중에서 하나만 지정해야 합니다.

예: '{"topicA":[0,1],"topicB":[2,4]}'

- "subscribePattern": (필수) 구독할 주제 목록을 식별하는 Java 정규식 문자열입니다. "topicName", "assign" 또는 "subscribePattern" 중에서 하나만 지정해야 합니다.

예: 'topic.*'

- "classification"(필수) 레코드의 데이터에서 사용하는 파일 형식입니다. 데이터 카탈로그를 통해 제공되지 않는 한 필수입니다.
- "delimiter"(선택 사항) classification이 CSV일 때 사용되는 값 구분 기호입니다. 기본값은 ','입니다.
- "startingOffsets": (선택 사항) 데이터를 읽을 Kafka 주제의 시작 위치입니다. 가능한 값은 "earliest" 또는 "latest"입니다. 기본값은 "latest"입니다.
- "startingTimestamp": (선택 사항, AWS Glue 버전 4.0 이상에서만 지원됨) Kafka 주제에서 데이터를 읽을 레코드의 타임스탬프입니다. 가능한 값은 yyyy-mm-ddTHH:MM:SSZ 패턴에서 UTC 형식의 타임스탬프 문자열입니다(여기서, Z는 UTC 시간대 오프셋(+/-)임, 예: '2023-04-04T08:00:00-04:00').

참고: AWS Glue 스트리밍 스크립트의 연결 옵션 목록에는 'StartingOffsets' 또는 'StartingTimestamp' 중 하나만 표시될 수 있습니다. 이 두 속성을 모두 포함하면 작업이 실패합니다.

- "endingOffsets": (선택 사항) 일괄 쿼리가 종료되는 엔드포인트입니다. 가능한 값은 "latest" 또는 각 TopicPartition의 끝 오프셋을 지정하는 JSON 문자열입니다.

JSON 문자열의 경우 포맷은 {"topicA":{"0":23,"1":-1},"topicB":{"0":-1}}입니다. 오프셋으로 값 -1은 "latest"를 나타냅니다.

- "pollTimeoutMs": (선택 사항) Spark 작업 실행기의 Kafka에서 데이터를 폴링하는 시간 제한(밀리초)입니다. 기본 값은 512입니다.
- "numRetries": (선택 사항) Kafka 오프셋 가져오기에 실패하기 전에 재시도할 횟수입니다. 기본 값은 3입니다.
- "retryIntervalMs": (선택 사항) Kafka 오프셋을 가져오기를 다시 시도하기 전에 대기하는 시간(밀리초)입니다. 기본 값은 10입니다.
- "maxOffsetsPerTrigger": (선택 사항) 트리거 간격당 처리되는 최대 오프셋 수에 대한 속도 제한입니다. 지정된 총 오프셋 수는 서로 다른 볼륨의 topicPartitions에 비례하여 분할됩니다. 기본값은 null입니다. 즉, 소비자가 알려진 최신 오프셋까지 모든 오프셋을 읽습니다.
- "minPartitions": (선택 사항) Kafka에서 읽을 원하는 최소 파티션 수입니다. 기본값은 null이며 이는 Spark 파티션의 수가 Kafka 파티션의 수와 동일함을 의미합니다.
- "includeHeaders": (선택 사항) Kafka 헤더를 포함할지 여부입니다. 옵션이 "true"로 설정되면 데이터 출력에는 유형이 Array[Struct(key: String, value: String)]인 "glue_streaming_kafka_headers"라는 추가 열이 포함됩니다. 기본값은 "false"입니다. 이 옵션은 AWS Glue 버전 3.0 이상에서만 사용할 수 있습니다.
- "schema": (InferSchema가 false로 설정된 경우 필수) 페이로드를 처리하는 데 사용할 스키마입니다. 분류가 avro인 경우 제공된 스키마는 Avro 스키마 형식이어야 합니다. 분류가 avro가 아닌 경우 제공된 스키마는 DDL 스키마 형식이어야 합니다.

다음은 스키마의 예입니다.

Example in DDL schema format

```
'column1' INT, 'column2' STRING , 'column3' FLOAT
```

Example in Avro schema format

```
{
  "type":"array",
```

```

"items":
{
  "type":"record",
  "name":"test",
  "fields":
  [
    {
      "name": "_id",
      "type": "string"
    },
    {
      "name": "index",
      "type":
      [
        "int",
        "string",
        "float"
      ]
    }
  ]
}
}
}

```

- "inferSchema": (선택 사항) 기본값은 'false'입니다. 'true'로 설정하면 스키마가 런타임 시 foreachbatch 내의 페이로드에서 감지됩니다.
- "avroSchema": (더 이상 사용되지 않음) Avro 형식을 사용할 때 Avro 데이터의 스키마를 지정하는 데 사용되는 파라미터입니다. 이 파라미터는 이제 사용 중단되었습니다. schema 파라미터를 사용합니다.
- "addRecordTimestamp": (선택 사항) 이 옵션이 'true'로 설정되면 데이터 출력에는 이름이 '_src_timestamp'라는 추가 열이 포함됩니다. 이 열은 주제에서 해당 레코드를 수신한 시간을 나타냅니다. 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.
- "emitConsumerLagMetrics": (선택 사항) 옵션을 'true'로 설정하면 각 배치에 대해 주제가 수신한 가장 오래된 레코드와 해당 레코드가 도착한 시간 사이의 기간 동안 메트릭이 내보내집니다. AWS Glue CloudWatch 메트릭의 이름은 "glue.driver.streaming입니다. maxConsumerLagInMs". 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.

쓸 때는 "connectionType": "kafka"를 사용한 다음 연결 옵션을 사용합니다.

- "connectionName"(필수) Kafka 클러스터에 연결하는 데 사용되는 AWS Glue 연결 이름 (Kafka 소스와 유사).

- "topic"(필수) 주제 열이 존재하는 경우 주제 구성 옵션이 설정되지 않은 한 Kafka에 지정된 행을 쓸 때 해당 값이 주제로 사용됩니다. 즉, topic 구성 옵션이 주제 열을 재정의합니다.
- "partition"(선택 사항) 유효한 파티션 번호가 지정되면 레코드를 보낼 때 해당 partition이 사용됩니다.

파티션을 지정하지 않았지만 key가 있는 경우 키의 해시를 사용하여 파티션이 선택됩니다.

key와 partition이 모두 존재하지 않으면 파티션에 최소 batch.size 바이트가 생성될 때 변경 사항을 고정 파티셔닝하여 파티션이 선택됩니다.

- "key"(선택 사항) partition이 null인 경우 파티셔닝에 사용됩니다.
- "classification"(선택 사항) 레코드의 데이터에서 사용하는 파일 형식입니다. JSON, CSV, Avro만 지원합니다.

Avro 형식을 사용하면 직렬화할 사용자 지정 avroSchema를 제공할 수 있지만 역직렬화를 위해서는 소스에서도 이를 제공해야 한다는 점에 유의하세요. 그렇지 않으면 기본적으로 직렬화에 Apache를 사용합니다. AvroSchema

또한 필요에 따라 [Kafka 프로듀서 구성 파라미터](#)를 업데이트하여 Kafka 싱크를 미세 튜닝할 수 있습니다. 연결 옵션에는 허용 목록이 없으며 모든 키-값 쌍은 싱크에 그대로 유지됩니다.

그러나 적용되지 않는 일부 거부 옵션의 목록이 있습니다. 자세한 내용은 [Kafka 특정 구성](#)을 참조하세요.

Kinesis 연결

데이터 카탈로그 테이블에 저장된 정보를 사용하거나 데이터 스트림에 직접 액세스할 수 있는 정보를 제공하여 Amazon Kinesis Data Streams에서 읽고 쓸 수 있습니다. Kinesis에서 DataFrame Spark로 정보를 읽은 다음 이를 Glue로 변환할 수 있습니다. AWS DynamicFrame DynamicFrames Kinesis에 JSON 형식으로 쓸 수 있습니다. 데이터 스트림에 직접 액세스하는 경우 이러한 옵션을 사용하여 데이터 스트림에 액세스하는 방법에 대한 정보를 제공합니다.

getCatalogSource 또는 create_data_frame_from_catalog를 통해 Kinesis 스트리밍 소스의 레코드를 사용하는 경우 작업에 데이터 카탈로그 데이터베이스 및 테이블 이름 정보가 있으며, 해당 정보를 사용하여 Kinesis 스트리밍 소스에서 읽기 위한 몇 가지 기본 파라미터를 얻을 수 있습니다. getSource, getSourceWithFormat, createDataFrameFromOptions 또는 create_data_frame_from_options를 사용하는 경우 여기에 설명된 연결 옵션을 통해 이러한 기본 파라미터를 지정해야 합니다.

GlueContext 클래스에 지정된 메서드에 대해 다음 인수를 사용하여 Kinesis에 대한 연결 옵션을 지정할 수 있습니다.

- Scala
 - connectionOptions: getSource, createDataFrameFromOptions, getSink와(과) 함께 사용
 - additionalOptions: getCatalogSource, getCatalogSink와 함께 사용
 - options: getSourceWithFormat, getSinkWithFormat와 함께 사용
- Python
 - connection_options: create_data_frame_from_options, write_dynamic_frame_from_options와 함께 사용
 - additional_options: create_data_frame_from_catalog, write_dynamic_frame_from_catalog와 함께 사용
 - options: getSource, getSink와 함께 사용

스트리밍 ETL 작업에 대한 참고 및 제한 사항은 [the section called “스트리밍 ETL 참고 사항 및 제한 사항”](#) 섹션을 참조하세요.

Kinesis 구성

AWS Glue Spark 작업에서 Kinesis 데이터 스트림에 연결하려면 몇 가지 사전 요구 사항이 필요합니다.

- 읽는 경우 AWS Glue 작업에는 Kinesis 데이터 스트림에 대한 읽기 액세스 수준 IAM 권한이 있어야 합니다.
- 작성하는 경우 AWS Glue 작업에는 Kinesis 데이터 스트림에 대한 쓰기 액세스 수준 IAM 권한이 있어야 합니다.

경우에 따라 추가 필수 조건을 구성해야 합니다.

- AWS Glue 작업이 추가 네트워크 연결 (일반적으로 다른 데이터세트에 연결) 으로 구성되어 있고 이러한 연결 중 하나가 Amazon VPC 네트워크 옵션을 제공하는 경우, 이 작업은 Amazon VPC를 통해 통신하도록 지시합니다. 이 경우 Amazon VPC를 통해 통신하도록 Kinesis 데이터 스트림도 구성해야 합니다. Amazon VPC와 Kinesis 데이터 스트림 사이에서 인터페이스 VPC 엔드포인트를 생성하면 됩니다. 자세한 내용은 [Using Kinesis Data Streams with Interface VPC Endpoints](#)를 참조하세요.

- 다른 계정에서 Amazon Kinesis Data Streams를 지정할 때 크로스 계정 액세스를 허용하도록 역할과 정책을 설정해야 합니다. 자세한 내용은 [예: 다른 계정의 Kinesis 스트림에서 읽기](#)를 참조하세요.

스트리밍 ETL 작업 필수 조건에 대한 자세한 내용은 [the section called “스트리밍 ETL 작업”](#) 섹션을 참조하세요.

Kinesis에서 읽기

예제: Kinesis 스트림에서 읽기

[the section called “forEachBatch”](#)과(와) 함께 사용합니다.

Amazon Kinesis 스트리밍 소스의 예:

```
kinesis_options =
  { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
    "startingPosition": "TRIM_HORIZON",
    "inferSchema": "true",
    "classification": "json"
  }
data_frame_datasource0 =
  glueContext.create_data_frame.from_options(connection_type="kinesis",
  connection_options=kinesis_options)
```

Kinesis에 쓰기

예: Kinesis 스트림에 쓰기

[the section called “forEachBatch”](#)과(와) 함께 사용합니다. JSON 형식으로 스트림에 DynamicFrame 기록됩니다. 작업을 여러 번 시도해도 쓸 수 없다면 작업이 실패합니다. 기본적으로 각 DynamicFrame 레코드는 Kinesis 스트림으로 개별적으로 전송됩니다. aggregationEnabled 및 관련 파라미터를 사용하여 이 동작을 구성할 수 있습니다.

스트리밍 작업에서 Amazon Kinesis로 작성하는 예:

Python

```
glueContext.write_dynamic_frame.from_options(
  frame=frameToWrite
  connection_type="kinesis",
  connection_options={
```

```

        "partitionKey": "part1",
        "streamARN": "arn:aws:kinesis:us-east-1:111122223333:stream/streamName",
    }
)

```

Scala

```

glueContext.getSinkWithFormat(
    connectionType="kinesis",
    options=JsonOptions("""{
        "streamARN": "arn:aws:kinesis:us-
east-1:111122223333:stream/streamName",
        "partitionKey": "part1"
    }"""),
)
    .writeDynamicFrame(frameToWrite)

```

Kinesis 연결 파라미터

Amazon Kinesis Data Streams에 대한 연결 옵션을 지정합니다.

Kinesis 스트리밍 데이터 원본에 대한 다음 연결 옵션을 사용합니다.

- "streamARN" (필수) 읽기/쓰기에 사용됩니다. Kinesis 데이터 스트림의 ARN.
- "classification" (읽기 필수) 읽기에 사용됩니다. 레코드의 데이터에서 사용하는 파일 형식. 데이터 카탈로그를 통해 제공되지 않는 한 필수입니다.
- "streamName" - (선택 사항) 읽기에 사용됩니다. 읽을 Kinesis 데이터 스트림의 이름. `endpointUrl`와(과) 함께 사용됩니다.
- "endpointUrl" - (선택 사항) 읽기에 사용됩니다. 기본값: "https://kinesis.us-east-1.amazonaws.com". Kinesis 스트림의 AWS 엔드포인트입니다. 특정 지역에 연결하는 경우가 아니면 변경하지 않아도 됩니다.
- "partitionKey" - (선택 사항) 쓰기에 사용됩니다. 레코드를 생성할 때 사용되는 Kinesis 파티션 키.
- "delimiter" (선택 사항) 읽기에 사용됩니다. `classification`이(가) CSV일 때 사용되는 값 구분 기호입니다. 기본값은 ','입니다.
- "startingPosition": (선택 사항) 읽기에 사용됩니다. 데이터를 읽을 Kinesis 데이터 스트림의 시작 위치입니다. 가능한 값은 yyyy-mm-ddTHH:MM:SSZ 패턴에서 UTC 형식의 타임스탬프 문자열이나 "latest", "trim_horizon" 또는 "earliest"입니다(여기서, Z는 UTC 시간대 오프셋(+/-)

임, 예: '2023-04-04T08:00:00-04:00'). 기본 값은 "latest"입니다. 참고: UTC 형식의 타임스탬프 문자열은 AWS Glue 버전 4.0 이상에서만 지원됩니다. "startingPosition"

- "failOnDataLoss": (선택 사항) 활성 샤드가 누락되거나 만료된 경우 작업이 실패합니다. 기본 값은 "false"입니다.
- "awsSTSRoleARN": (선택 사항) 읽기/쓰기에 사용됩니다. () 를 사용하여 위임할 역할의 Amazon 리소스 이름 AWS Security Token Service (AWS STS ARN) 입니다. 이 역할에는 Kinesis 데이터 스트림에 대한 레코드 작업을 설명하거나 읽을 수 있는 권한이 있어야 합니다. 다른 계정의 데이터 스트림에 액세스할 때 이 파라미터를 사용해야 합니다. "awsSTSSessionName"과(와) 함께 사용됩니다.
- "awsSTSSessionName": (선택 사항) 읽기/쓰기에 사용됩니다. AWS STS을(를) 사용하여 역할을 맡는 세션의 식별자입니다. 다른 계정의 데이터 스트림에 액세스할 때 이 파라미터를 사용해야 합니다. "awsSTSRoleARN"과(와) 함께 사용됩니다.
- "awsSTSEndpoint": (선택 사항) 위임된 역할을 사용하여 Kinesis에 연결할 때 사용할 AWS STS 엔드포인트입니다. 이를 통해 VPC에서 리전 AWS STS 엔드포인트를 사용할 수 있지만, 기본 글로벌 엔드포인트로는 불가능합니다.
- "maxFetchTimeInMs": (선택 사항) 읽기에 사용됩니다. 작업 실행기가 Kinesis 데이터 스트림에서 현재 배치에 대한 레코드를 읽는 데 걸리는 최대 시간(밀리초(ms) 단위로 지정)입니다. 이 시간 내에 여러 개의 GetRecords API 호출을 할 수 있습니다. 기본 값은 1000입니다.
- "maxFetchRecordsPerShard": (선택 사항) 읽기에 사용됩니다. 마이크로 배치에 따라 Kinesis 데이터 스트림에서 샤드당 가져올 최대 레코드 수입니다. 참고: 스트리밍 작업이 이미 Kinesis의 동일한 get-records 호출에서 추가 레코드를 읽은 경우 클라이언트가 이 제한을 초과할 수 있습니다. maxFetchRecordsPerShard가 엄격해야 한다면 maxRecordPerRead의 배수여야 합니다. 기본 값은 100000입니다.
- "maxRecordPerRead": (선택 사항) 읽기에 사용됩니다. 각 getRecords 작업에서 Kinesis 데이터 스트림에서 가져올 최대 레코드 수. 기본 값은 10000입니다.
- "addIdleTimeBetweenReads": (선택 사항) 읽기에 사용됩니다. 연속 두 getRecords 작업 사이에 시간 지연을 추가합니다. 기본 값은 "False"입니다. 이 옵션은 Glue 버전 2.0 이상에서만 구성할 수 있습니다.
- "idleTimeBetweenReadsInMs": (선택 사항) 읽기에 사용됩니다. 연속 두 getRecords 작업 사이의 최소 시간 지연으로, ms 단위로 지정됩니다. 기본 값은 1000입니다. 이 옵션은 Glue 버전 2.0 이상에서만 구성할 수 있습니다.
- "describeShardInterval": (선택 사항) 읽기에 사용됩니다. 스크립트가 리샤딩을 고려하기 위한 두 ListShards API 호출 사이의 최소 시간 간격. 자세한 내용은 Amazon Kinesis Data Streams Developer Guide의 [Strategies for Resharding](#)을 참조하세요. 기본 값은 1s입니다.

- "numRetries": (선택 사항) 읽기에 사용됩니다. Kinesis Data Streams API 요청의 최대 재시도 횟수입니다. 기본 값은 3입니다.
- "retryIntervalMs": (선택 사항) 읽기에 사용됩니다. Kinesis Data Streams API 호출을 재시도하기 전의 휴지 기간(ms 단위로 지정)입니다. 기본 값은 1000입니다.
- "maxRetryIntervalMs": (선택 사항) 읽기에 사용됩니다. Kinesis Data Streams API 호출을 두 번 재시도하는 사이의 최대 휴지 시간(ms 단위로 지정)입니다. 기본 값은 10000입니다.
- "avoidEmptyBatches": (선택 사항) 읽기에 사용됩니다. 배치가 시작되기 전에 Kinesis 데이터 스트림에서 읽지 않은 데이터를 확인하여 빈 마이크로 배치 작업 생성을 방지합니다. 기본 값은 "False"입니다.
- "schema": (inferSchema가 거짓으로 설정된 경우 필수 사항) 읽기에 사용됩니다. 페이로드를 처리하는 데 사용하는 스키마. 분류가 avro인 경우 제공된 스키마는 Avro 스키마 형식이어야 합니다. 분류가 avro가 아닌 경우 제공된 스키마는 DDL 스키마 형식이어야 합니다.

다음은 스키마의 예입니다.

Example in DDL schema format

```
`column1` INT, `column2` STRING , `column3` FLOAT
```

Example in Avro schema format

```
{
  "type": "array",
  "items":
  {
    "type": "record",
    "name": "test",
    "fields":
    [
      {
        "name": "_id",
        "type": "string"
      },
      {
        "name": "index",
        "type":
        [
          "int",
          "string",
          "float"
        ]
      }
    ]
  }
}
```

```

    ]
  }
]
}
}

```

- "inferSchema": (선택 사항) 읽기에 사용됩니다. 기본값은 'false'입니다. 'true'로 설정하면 스키마가 런타임 시 foreachbatch 내의 페이로드에서 감지됩니다.
- "avroSchema": (더 이상 사용되지 않음) 읽기에 사용됩니다. Avro 형식을 사용할 때 Avro 데이터의 스키마를 지정하는 데 사용되는 파라미터입니다. 이 파라미터는 이제 사용 중단되었습니다. schema 파라미터를 사용합니다.
- "addRecordTimestamp": (선택 사항) 읽기에 사용됩니다. 이 옵션이 'true'로 설정되면 데이터 출력에는 이름이 '_src_timestamp'라는 추가 열이 포함됩니다. 이 열은 스트림에서 해당 레코드를 수신한 시간을 나타냅니다. 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.
- "emitConsumerLagMetrics": (선택 사항) 읽기에 사용됩니다. 옵션을 'true'로 설정하면 각 배치에 대해 스트림이 수신한 가장 오래된 레코드와 스트림이 수신된 시간 사이의 기간 동안 측정치를 내보냅니다. AWS Glue CloudWatch 메트릭의 이름은 "glue.driver.streaming.maxConsumerLagInMs". 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.
- "fanoutConsumerARN": (선택 사항) 읽기에 사용됩니다. streamARN에서 지정된 스트림에 대한 Kinesis 스트림 소비자의 ARN. Kinesis 연결에 대해 향상된 팬아웃 모드를 활성화하는 데 사용됩니다. 향상된 팬아웃과 함께 Kinesis 스트림을 사용하는 방법에 대한 자세한 내용은 [the section called "Kinesis 스트리밍 작업에서 향상된 팬아웃 사용"](#) 섹션을 참조하세요.
- "recordMaxBufferedTime" - (선택 사항) 쓰기에 사용됩니다. 기본값: 1000(ms). 레코드 쓰기를 대기하는 중 레코드가 버퍼링되는 최대 시간.
- "aggregationEnabled" - (선택 사항) 쓰기에 사용됩니다. 기본값: true. Kinesis로 보내기 전에 레코드를 집계해야 하는지 여부를 지정합니다.
- "aggregationMaxSize" - (선택 사항) 쓰기에 사용됩니다. 기본값: 51200(바이트). 레코드가 이 한도보다 크면 애그리게이터를 우회합니다. 참고 Kinesis는 레코드 크기를 50KB로 제한합니다. 이 값이 50KB를 초과하도록 설정하면 Kinesis에서 크기가 초과된 레코드를 거부합니다.
- "aggregationMaxCount" - (선택 사항) 쓰기에 사용됩니다. 기본값: 4294967295. 집계된 레코드에 포함할 최대 항목 수.
- "producerRateLimit" - (선택 사항) 쓰기에 사용됩니다. 기본값: 150(%). 단일 생성자(예: 작업)에서 보내는 샤드당 처리량을 백엔드 한도의 백분율로 제한합니다.

- "collectionMaxCount" – (선택 사항) 쓰기에 사용됩니다. 기본값: 500. PutRecords 요청 시 포함할 최대 품목 수.
- "collectionMaxSize" – (선택 사항) 쓰기에 사용됩니다. 기본값: 5242880(바이트). PutRecords 요청과 함께 전송할 수 있는 최대 데이터 양.

AWS Glue 스트리밍 옵션

Kafka 클러스터 또는 Amazon Managed Streaming for Apache Kafka 클러스터에 대한 연결을 지정합니다.

데이터 카탈로그 테이블에 저장된 정보를 사용하거나 데이터 스트림에 직접 액세스할 수 있는 정보를 제공하여 Kafka Data Streams에서 읽고 쓸 수 있습니다. Kafka에서 DataFrame Spark로 정보를 읽은 다음 Glue로 변환할 수 있습니다. AWS DynamicFrame JSON 형식으로 DynamicFrames Kafka에 글을 쓸 수 있습니다. 데이터 스트림에 직접 액세스하는 경우 이러한 옵션을 사용하여 데이터 스트림에 액세스하는 방법에 대한 정보를 제공합니다.

getCatalogSource 또는 create_data_frame_from_catalog를 사용하여 Kafka 스트리밍 소스에서 레코드를 소비하거나 getCatalogSink 또는 write_dynamic_frame_from_catalog를 사용하여 Kafka에 레코드를 쓰는 경우 작업에는 데이터 카탈로그 데이터베이스와 테이블 이름 정보가 있으며 이를 사용하여 Kafka 스트리밍 소스에서 읽기 위한 몇 가지 기본 파라미터를 얻을 수 있습니다. getSource, getCatalogSink, getSourceWithFormat, getSinkWithFormat, createDataFrameFromOptions, create_data_frame_from_options 또는 write_dynamic_frame_from_catalog를 사용하는 경우 여기에 설명된 연결 옵션을 통해 이러한 기본 파라미터를 지정해야 합니다.

GlueContext 클래스에 지정된 메서드에 대해 다음 인수를 사용하여 Kafka에 대한 연결 옵션을 지정할 수 있습니다.

- Scala
 - connectionOptions: getSource, createDataFrameFromOptions, getSink와(과) 함께 사용
 - additionalOptions: getCatalogSource, getCatalogSink와 함께 사용
 - options: getSourceWithFormat, getSinkWithFormat와 함께 사용
- Python
 - connection_options: create_data_frame_from_options, write_dynamic_frame_from_options와 함께 사용

- `additional_options`: `create_data_frame_from_catalog`, `write_dynamic_frame_from_catalog`와 함께 사용
- `options`: `getSource`, `getSink`와 함께 사용

스트리밍 ETL 작업에 대한 참고 및 제한 사항은 [the section called “스트리밍 ETL 참고 사항 및 제한 사항”](#) 섹션을 참조하세요.

AWS Glue 스트리밍 자동 크기 조정

다음 섹션에서는 AWS Glue 스트리밍 자동 크기 조정에 대한 정보를 제공합니다.

AWS Glue Studio에서 Auto Scaling 사용

AWS Glue Studio의 Job details(작업 세부 정보) 탭에서 유형을 Spark 또는 Spark Streaming으로, Glue version(Glue 버전)을 **Glue 3.0** 또는 **Glue 4.0**으로 선택합니다. 그러면 작업자 유형(Worker type) 아래에 확인란이 표시됩니다.

- 작업자 수 자동 크기 조정(Automatically scale the number of workers) 옵션을 선택합니다.
- 최대 작업자 수(Maximum number of workers)를 설정하여 작업 실행에 판매할 수 있는 최대 작업자 수를 정의합니다.

Visual
Script
Job details
Runs
Data quality
Schedules

Version Control

Type
The type of ETL job. This is set automatically based on the types of data sources you have selected.

Spark

Glue version [Info](#)

Glue 4.0 - Supports spark 3.3, Scala 2, Python 3 ▼

Language

Python 3 ▼

Worker type
Set the type of predefined worker that is allowed when a job runs.

G 1X
(4vCPU and 16GB RAM) ▼

Automatically scale the number of workers
AWS Glue will optimize costs and resource usage by dynamically scaling the number of workers up and down throughout the job run. Requires Glue 3.0 or later.

Maximum number of workers
The number of workers you want AWS Glue to allocate to this job.

10

AWS CLI 또는 SDK를 사용하여 Auto Scaling 사용 설정

AWS CLI에서 Auto Scaling을 사용 설정하여 작업을 실행하려면, 다음 구성으로 `start-job-run`을 실행합니다.

```
{
  "JobName": "<your job name>",
  "Arguments": {
    "--enable-auto-scaling": "true"
  },
  "WorkerType": "G.2X", // G.1X and G.2X are allowed for Auto Scaling Jobs
}
```

```
"NumberOfWorkers": 20, // represents Maximum number of workers
...other job run configurations...
}
```

ETL 작업 실행이 완료되면 `get-job-run`을 호출하여 DPU 초 단위로 작업 실행의 실제 리소스 사용량을 확인할 수도 있습니다. 참고: 새 필드 `DPUSeconds`는 Auto Scaling이 사용 설정된 AWS Glue 3.0 이상의 배치 작업에만 표시됩니다. 이 필드는 스트리밍 작업에는 지원되지 않습니다.

```
$ aws glue get-job-run --job-name your-job-name --run-id jr_xx --endpoint https://
glue.us-east-1.amazonaws.com --region us-east-1
{
  "JobRun": {
    ...
    "GlueVersion": "3.0",
    "DPUSeconds": 386.0
  }
}
```

동일한 구성의 [AWS Glue SDK](#)를 사용하여 Auto Scaling으로 작업 실행을 구성할 수도 있습니다.

작동 방식

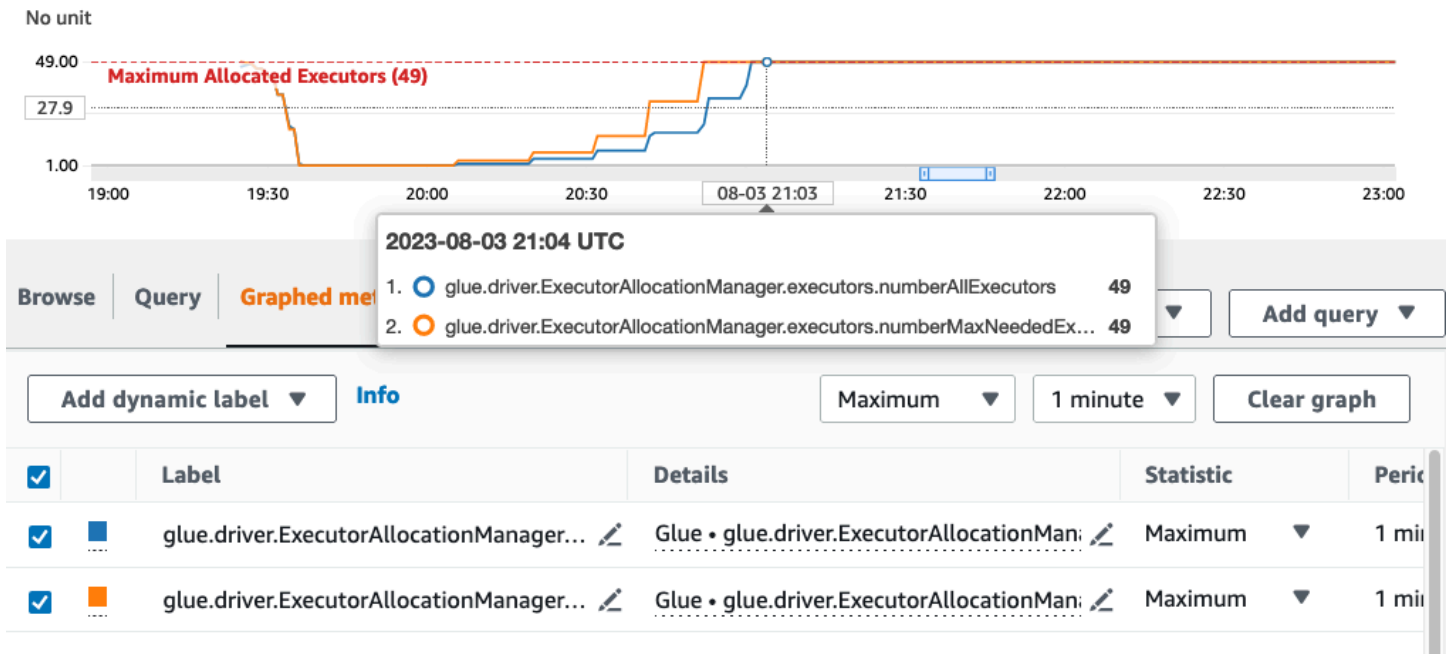
마이크로 배치 전반에 걸쳐 크기 조정

다음 예제는 자동 크기 조정의 작동 방식을 설명하는 데 사용됩니다.

- 50개의 DPU로 시작하는 AWS Glue 작업이 있습니다.
- 자동 크기 조정이 활성화되었습니다.

이 예제에서 AWS Glue는 몇 개의 마이크로 배치에 대한 'batchProcessingTimeInMs' 지표를 살펴보고 사용자가 설정한 기간 내에서 작업이 완료되는지 확인합니다. 작업이 더 빨리 완료되는 경우 작업이 얼마나 빨리 완료되는지에 따라 AWS Glue가 스케일 다운될 수 있습니다. 'numberAllExecutors'로 그린 이 지표를 Amazon CloudWatch에서 모니터링하여 자동 크기 조정이 어떻게 작동하는지 확인할 수 있습니다.

실행기 수는 각 마이크로 배치가 완료된 후에만 기하급수적으로 스케일 업 또는 다운됩니다. Amazon CloudWatch 모니터링 로그에서 볼 수 있듯이, AWS Glue는 필요한 실행기 수(주황색 선)를 살펴보고 그에 맞춰 자동으로 실행기 크기(파란색 선)를 조정합니다.



AWS Glue가 실행기 수를 스케일 다운하고 데이터 볼륨 증가에 따른 마이크로 배치 처리 시간 증가가 관찰되면 AWS Glue는 지정된 상한인 50 DPU까지 스케일 업합니다.

마이크로 배치 내에서 크기 조정

위 예제에서 시스템은 완료된 마이크로 배치 몇 개를 모니터링하여 스케일 업할지 스케일 다운할지 결정합니다. 기간이 길어질수록 몇 개의 마이크로 배치를 기다리는 대신 마이크로 배치 내에서 더 빠르게 응답하기 위해 Auto Scaling이 필요합니다. 이러한 경우 추가 구성 `--auto-scale-within-microbatch`를 `true`로 사용할 수 있습니다. 아래와 같이 AWS Glue Studio의 AWS Glue 작업 속성에 이를 추가할 수 있습니다.

Job parameters [Info](#)

Key Value - optional

`--auto-scale-within-microbatch` `true` Remove

Add new parameter

You can add 49 more parameters.

AWS Glue 스트리밍 유지 관리 기간

AWS Glue 정기적으로 유지 관리 활동을 수행합니다. 이러한 유지 관리 기간 AWS Glue 동안에는 스트리밍 작업을 다시 시작해야 합니다. 유지 관리 기간을 지정하여 작업 재시작 시기를 제어할 수 있습니다. 이 섹션에서는 유지 관리 기간을 설정할 수 있는 위치와 고려해야 할 특정 동작에 대해 설명합니다.

주제

- [유지 관리 기간 설정](#)
- [유지 관리 기간 동작](#)
- [Job 모니터링](#)
- [데이터 손실 처리](#)

유지 관리 기간 설정

AWS Glue Studio 또는 API를 사용하여 유지 관리 기간을 설정할 수 있습니다.

Studio에서 AWS Glue 유지 관리 기간 설정

AWS Glue 스트리밍 작업의 Job Details 페이지에서 유지 관리 기간을 지정할 수 있습니다. 날짜 및 시간을 GMT로 지정할 수 있습니다. AWS Glue 지정된 시간 내에 작업을 다시 시작합니다.

Maintenance window

Restart on

at hours (GMT)

For maintenance reasons, AWS Glue will restart streaming jobs within 3 hours of the specified maintenance window. You have the option to designate the start time in GMT for this maintenance. For more information, refer to documentation.

API에서 유지 관리 기간 설정

Create Job API에서 유지 관리 기간을 설정할 수도 있습니다. 다음은 API를 통해 유지 관리 기간을 구성하는 예시입니다.

```
aws glue create-job --name jobName --role roleArnForTheJob --command
Name=gluestreaming,ScriptLocation=s3-path-to-the-script --maintenance-window="Sun:10"
```

예제 명령은 다음과 같습니다.

```
aws glue create-job --name testMaintenance --role arn:aws:iam::012345678901:role/
Glue_DefaultRole --command Name=gluestreaming,ScriptLocation=s3://glue-example-test/
example.py --maintenance-window="Sun:10"
```

유지 관리 기간 동작

AWS Glue 일련의 단계를 거쳐 작업 재시작 시기를 결정합니다.

1. 새 스트리밍 작업이 시작되면 AWS Glue 먼저 작업 실행과 관련된 제한 시간이 있는지 확인합니다. 타임아웃을 통해 작업 종료 시간을 구성할 수 있습니다. 제한 시간이 7일 미만이면 작업이 다시 시작되지 않습니다.
2. 제한 시간이 7일을 초과하는 경우 해당 작업에 대해 유지 관리 기간이 구성되어 AWS Glue 있는지 확인합니다. 그럴 경우 해당 창이 선택되고 해당 창이 작업 실행에 할당됩니다. AWS Glue 지정된 유지 관리 기간으로부터 3시간 이내에 작업을 다시 시작합니다. 예를 들어, 유지 관리 기간을 월요일 오전 10시 (GMT) 로 설정하면 작업이 그리니치 표준시 오전 10시에서 오후 1시 (GMT) 사이에 다시 시작됩니다.
3. 유지 관리 기간을 구성하지 않은 경우 재시작 시간을 작업 실행 시작 시간으로부터 7일 경과한 시간으로 AWS Glue 자동 설정합니다. 예를 들어, 2024년 7월 1일 오전 12:00 GMT에 작업을 시작하고 유지 관리 기간을 지정하지 않은 경우 작업은 2024년 7월 8일 오전 12:00 GMT에 다시 시작되도록 설정됩니다.

Note

이미 스트리밍 작업을 실행 중인 경우, 이 변경 사항은 2024년 7월 1일부터 시작되는 작업에 적용됩니다. 6월 30일까지 유지 관리 기간을 구성할 수 있습니다. 7월 1일 이후에는 시작한 모든 스트리밍 작업이 이 설명서에 따라 다시 시작됩니다. 추가 지원이 필요한 경우 Support에 문의할 AWS 수 있습니다.

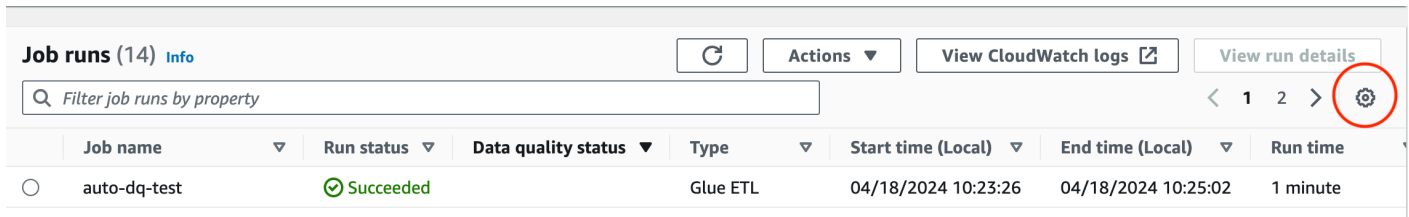
4. 경우에 AWS Glue 따라, 특히 진행 중인 마이크로 배치 처리되지 않는 경우 작업을 다시 시작하지 못할 수 있습니다. 이러한 경우에는 작업이 중단되지 않습니다. 이러한 경우에는 14일 후에 작업이 다시 AWS Glue 시작되며, 이 경우 유지 관리 기간은 유효하지 않습니다.

Job 모니터링

AWS Glue Studio 모니터링 페이지에서 작업을 모니터링할 수 있습니다.

스트리밍 작업의 다음 예상 재시작 시간을 보려면 모니터링 페이지의 Job run 테이블에 열을 표시하십시오.

1. 테이블 오른쪽 상단에 있는 기어 아이콘을 클릭합니다.



Job name	Run status	Data quality status	Type	Start time (Local)	End time (Local)	Run time
auto-dq-test	Succeeded		Glue ETL	04/18/2024 10:23:26	04/18/2024 10:25:02	1 minute

2. 아래로 스크롤하여 예상 재시작 시간 열을 켜십시오. UTC 및 현지 시간 옵션을 모두 사용할 수 있습니다.

worker type	
DPU hours	
Last modified (Local)	
Worker utilization	
Data skewness	
Start time (UTC)	
End time (UTC)	
Last modified (UTC)	
Data quality	
Expected restart time (UTC)	
Expected restart time (Local)	

Cancel
Confirm

3. 그러면 테이블의 열을 볼 수 있습니다.

Job runs (14) [Info](#)

	Job name	Run status	Type	Start time (Local)	End time (Local)	Expected restart time (Local)
<input type="radio"/>	auto-dq-test	✔ Succeeded	Glue ETL	04/18/2024 10:23:26	04/18/2024 10:25:02	-
<input type="radio"/>	StreamingTest	🔄 Running	Glue Streaming	04/16/2024 16:32:49	-	04/23/2024 02:00:00
<input type="radio"/>	StreamingProd	🔄 Running	Glue Streaming	04/16/2024 13:45:10	-	04/25/2024 05:00:00

원래 작업은 “EXPIRED” 상태이고 새 작업 인스턴스는 “RUNNING” 상태가 됩니다. 다시 시작된 새 작업 실행에는 초기 작업 실행 ID와 재시작 횟수를 나타내는 접두사 “restart_”가 결합된 작업 실행 ID가 있습니다. 예를 들어, 초기 작업 실행 ID가 인 경우 재시작된 작업 실행은 jr_1234 첫 번째 재시작의 ID를 갖게 됩니다. jr1234_restart_1 두 번째 재시작은 jr1234_restart_2 두 번째 재시작이고, 이런 식으로 계속됩니다.

재시동으로 인해 재시도는 영향을 받지 않습니다. 실행이 실패하고 자동 재시도로 인해 새 실행이 시작되는 경우 재시작 카운터가 1부터 다시 시작됩니다. 예를 들어, 실행이 실패하면 자동 재시도는 ID를 사용하여 새 실행을 시작하고 7일 후에 이 시도를 다시 시작하면 새 실행 ID는 다음과 같습니다 jr_id1_attempt_4. jr_1234_attempt_3_restart_5 jr_id1_attempt_4_restart_1

데이터 손실 처리

유지 관리 재시작 시 AWS Glue 스트리밍은 이전 작업 실행과 재시작된 작업 실행 간의 데이터 무결성 및 일관성을 보장하는 프로세스를 따릅니다. 단, 작업 재시작 간에 데이터 무결성과 일관성이 보장되는 AWS Glue 것은 아니므로 스트리밍 작업 내에서 중복된 데이터를 처리할 때는 아키텍처를 고려하는 것이 좋습니다.

1. 유지 관리 재시작 조건 감지: AWS Glue 스트리밍은 유지 관리 재시작을 트리거해야 하는 시점 (예: 7일 후 유지 관리 기간에 도달하거나 14일 후에 강제 재시작이 필요한 경우) 을 나타내는 조건을 모니터링합니다.
2. 정상 종료 호출: 유지 관리 재시작 조건이 충족되면 AWS Glue 스트리밍은 현재 실행 중인 작업에 대한 정상 종료 프로세스를 시작합니다. 이 프로세스에는 다음 단계가 포함됩니다.
 - a. 새 데이터 수집 중지: 스트리밍 작업은 입력 소스 (예: Kafka 주제, Kinesis 스트림 또는 파일) 의 새 데이터 사용을 중지합니다.
 - b. 보류 중인 데이터 처리: 작업은 내부 버퍼 또는 대기열에 이미 있는 모든 데이터를 계속 처리합니다.
 - c. 오프셋 및 체크포인트 커밋: 작업은 최신 오프셋 또는 체크포인트를 외부 시스템 (예: Kafka, Kinesis 또는 Amazon S3) 에 커밋하여 재시작된 작업이 이전 작업이 중단된 지점부터 다시 시작할 수 있도록 합니다.
3. 작업 재시작: 정상 종료 프로세스가 완료되면 Streaming은 보존된 상태 및 체크포인트를 사용하여 작업을 다시 시작합니다. AWS Glue 다시 시작된 작업은 마지막으로 커밋된 오프셋 또는 체크포인트에서 처리를 선택하여 데이터가 손실되거나 중복되지 않도록 합니다.
4. 데이터 처리 재개: 다시 시작된 작업은 이전 작업이 중단된 시점부터 데이터 처리를 재개합니다. 마지막으로 커밋된 오프셋 또는 체크포인트에서 시작하여 입력 소스에서 새 데이터를 계속 수집하고 정의된 ETL 로직에 따라 데이터를 처리합니다.

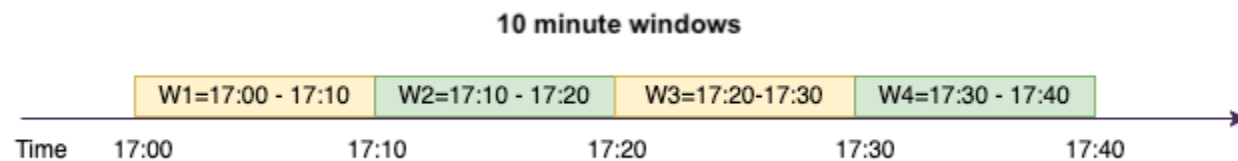
고급 AWS Glue 스트리밍 개념

현대의 데이터 기반 애플리케이션에서 데이터의 중요성은 시간 경과에 따라 감소하고 그 가치는 예측에서 반응으로 전환됩니다. 따라서 고객은 더 빠른 의사 결정을 내리기 위해 실시간으로 데이터를 처리하기를 원합니다. IoT 센서와 같은 실시간 데이터 피드를 처리할 때 데이터가 정렬되지 않은 상태로 도착하거나 네트워크 지연 시간과 수집 중 기타 소스 관련 오류로 인해 처리가 지연될 수 있습니다. AWS Glue 플랫폼의 일부인 AWS Glue 스트리밍은 이러한 기능을 기반으로 Apache Spark 정형 스트리밍으로 구동되는 확장 가능한 서버리스 스트리밍 ETL을 제공하여 사용자에게 실시간 데이터 처리 기능을 제공합니다.

이 주제에서는 AWS Glue 스트리밍의 고급 스트리밍 개념과 기능을 살펴보겠습니다.

스트림 처리 시 시간 고려 사항

스트림을 처리할 때 시간에 대한 네 가지 개념이 있습니다.



- 이벤트 타임 - 이벤트가 발생한 시간입니다. 대부분의 경우 이 필드는 소스의 이벤트 데이터 자체에 포함됩니다.
- Event-time-window — 두 이벤트 시간 사이의 시간 프레임. 위 다이어그램에서 볼 수 있듯이 W1은 17:00 event-time-window 부터 17:10 사이입니다. event-time-window 각각은 여러 이벤트의 그룹입니다.
- 트리거 시간 - 트리거 시간은 데이터 처리 및 결과 업데이트 빈도를 제어합니다. 마이크로 배치 처리가 시작된 시간입니다.
- 모으기 시간 - 스트림 데이터가 스트리밍 서비스에 수집된 시간입니다. 이벤트 시간이 이벤트 자체에 포함되지 않은 경우 경우에 따라 이 시간을 기간 설정에 사용할 수 있습니다.

기간 설정

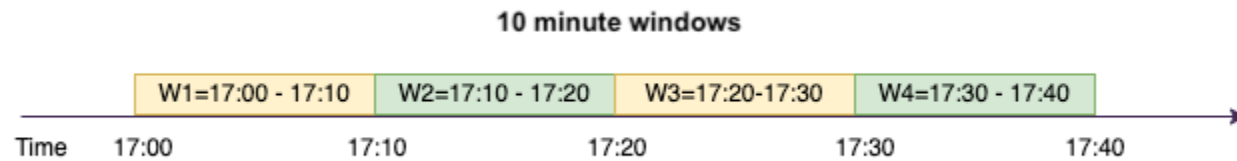
윈도잉은 여러 이벤트를 그룹화하고 집계하는 기법입니다. event-time-window 다음 예제에서는 기간 설정의 이점과 이를 사용하는 경우를 살펴보겠습니다.

비즈니스 사용 사례에 따라 Spark에서 지원하는 세 가지 유형의 기간이 있습니다.

- 텀블링 윈도우 — 일련의 겹치지 않는 고정 크기로 집계합니다. event-time-windows
- 슬라이딩 기간 - '고정된 크기'라는 점에서 연속 기간과 유사하지만, 슬라이드 지속 시간이 기간 자체의 지속 시간보다 짧으면 기간이 겹치거나 지나갈 수 있습니다.
- 세션 기간 - 입력 데이터 이벤트로 시작하여 공백 또는 비활성 기간 내에 입력을 수신하는 한 자체적으로 계속 확장됩니다. 세션 기간은 입력에 따라 기간 길이의 크기가 정적이거나 동적일 수 있습니다.

텀블링 윈도우

텀블링 윈도우는 일련의 겹치지 않는 고정 크기를 집계하는 것을 말합니다. event-time-windows 실제 사례를 들어 설명해 보겠습니다.



ABC Auto는 새로운 브랜드의 스포츠카에 대한 마케팅 캠페인을 진행하고자 합니다. 이를 위해 스포츠카 팬이 가장 많은 도시를 선택하려고 합니다. 이 목표를 달성하기 위해 이 회사는 웹 사이트에 자동차를 소개하는 15초짜리 짧은 광고를 선보입니다. 모든 “클릭”과 해당 “도시”가 기록되고 스트리밍됩니다. Amazon Kinesis Data Streams 10분 동안의 클릭 수를 세고 이를 도시별로 그룹화하여 수요가 가장 높은 도시를 확인하려고 합니다. 다음은 집계된 출력입니다.

window_start_time	window_end_time	구/군/시	total_clicks
2023-07-10 17:00:00	2023-07-10 17:10:00	댈러스	75
2023-07-10 17:00:00	2023-07-10 17:10:00	시카고	10
2023-07-10 17:20:00	2023-07-10 17:30:00	댈러스	20
2023-07-10 17:20:00	2023-07-10 17:30:00	시카고	50

위에서 설명한 것처럼 이는 트리거 시간 간격과 다릅니다. event-time-windows 예를 들어, 트리거 시간이 1분마다인 경우에도 출력 결과에는 겹치지 않는 집계 기간이 10분만 표시됩니다. 최적화를 위해서는 트리거 간격을 에 맞춰 정렬하는 것이 좋습니다. event-time-window

위 표에서 17:00~17:10 기간 동안 Dallas는 75번의 클릭이 있었고 Chicago는 10번의 클릭이 있었습니다. 또한 17:10~17:20 기간은 어느 도시의 데이터도 없으므로 생략됩니다.

이제 다운스트림 분석 애플리케이션에서 이 데이터에 대한 추가 분석을 실행하여 마케팅 캠페인을 실행할 가장 독점적인 도시를 결정할 수 있습니다.

AWS Glue에서 연속 기간 사용

1. a를 Amazon Kinesis Data Streams DataFrame 만들고 거기서 읽어보세요. 예제

```
parsed_df = kinesis_raw_df \
    .selectExpr('CAST(data AS STRING)') \
    .select(from_json("data", ticker_schema).alias("data")) \
    .select('data.event_time', 'data.ticker', 'data.trade', 'data.volume',
'data.price')
```

2. 연속 기간에서 데이터를 처리합니다. 아래 예제에서 데이터는 10분 연속 기간의 입력 필드 'event_time'을 기준으로 그룹화되고 출력을 Amazon S3 데이터 레이크에 씁니다.

```
grouped_df = parsed_df \
    .groupBy(window("event_time", "10 minutes"), "city") \
    .agg(sum("clicks").alias("total_clicks"))

summary_df = grouped_df \
    .withColumn("window_start_time", col("window.start")) \
    .withColumn("window_end_time", col("window.end")) \
    .withColumn("year", year("window_start_time")) \
    .withColumn("month", month("window_start_time")) \
    .withColumn("day", dayofmonth("window_start_time")) \
    .withColumn("hour", hour("window_start_time")) \
    .withColumn("minute", minute("window_start_time")) \
    .drop("window")

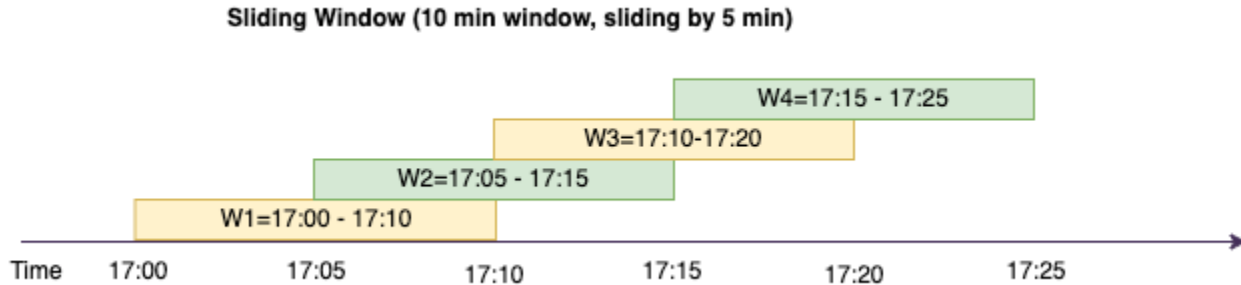
write_result = summary_df \
    .writeStream \
    .format("parquet") \
    .trigger(processingTime="10 seconds") \
    .option("checkpointLocation", "s3a://bucket-stock-stream/stock-
stream-catalog-job/checkpoint/") \
    .option("path", "s3a://bucket-stock-stream/stock-stream-catalog-
job/summary_output/") \
    .partitionBy("year", "month", "day") \
```



```
.start()
```

슬라이딩 윈도우

슬라이딩 기간은 '고정된 크기'라는 점에서 연속 기간과 유사하지만, 슬라이드 지속 시간이 기간 자체의 지속 시간보다 짧으면 기간이 겹치거나 지나갈 수 있습니다. 슬라이딩의 특성으로 인해 입력이 여러 기간에 바인딩될 수 있습니다.



이해를 돕기 위해 신용 카드 사기 가능성을 탐지하려는 은행의 예를 들어 보겠습니다. 스트리밍 애플리케이션은 신용 카드 거래의 지속적인 스트림을 모니터링할 수 있습니다. 이러한 트랜잭션은 10분 기간으로 집계될 수 있으며 5분마다 기간이 앞으로 이동하여 가장 오래된 5분의 데이터를 제거하고 최신 5분의 새 데이터를 추가합니다. 각 기간 내에서 미국에서의 트랜잭션 직후 호주에서의 트랜잭션과 같은 의심스러운 패턴을 확인하면서 국가별로 트랜잭션을 그룹화할 수 있습니다. 간소화를 위해 총 트랜잭션 금액이 100 USD를 초과하는 경우 이러한 트랜잭션을 사기로 분류하겠습니다. 이러한 패턴이 탐지되면 사기 가능성이 있다는 신호이며 카드가 동결될 수 있습니다.

신용 카드 처리 시스템은 국가와 함께 각 카드 ID에 대한 일련의 거래 이벤트를 Kinesis로 전송하고 있습니다. AWS Glue 작업은 분석을 실행하고 다음과 같은 집계된 결과를 생성합니다.

window_start_time	window_end_time	card_last_four	country	total_amount
2023-07-10 17:00:00	2023-07-10 17:10:00	6544	미국	85
2023-07-10 17:00:00	2023-07-10 17:10:00	6544	호주	10

window_start_time	window_end_time	card_last_four	country	total_amount
2023-07-10 17:05:45	2023-07-10 17:15:45	6544	미국	50
2023-07-10 17:10:45	2023-07-10 17:20:45	6544	미국	50
2023-07-10 17:10:45	2023-07-10 17:20:45	6544	호주	150

위 집계를 기준으로 5분마다 이동하는 10분 기간을 트랜잭션 금액별로 합산하여 볼 수 있습니다. 이상치(호주에서 150 USD 트랜잭션)가 있는 17:10~17:20 기간에서 이상이 탐지되었습니다. AWS Glue는 이러한 이상을 탐지하고 boto3를 사용하여 잘못된 키가 포함된 경보 이벤트를 SNS 주제에 푸시할 수 있습니다. 또한 Lambda 함수는 이 주제를 구독하고 조치를 취할 수 있습니다.

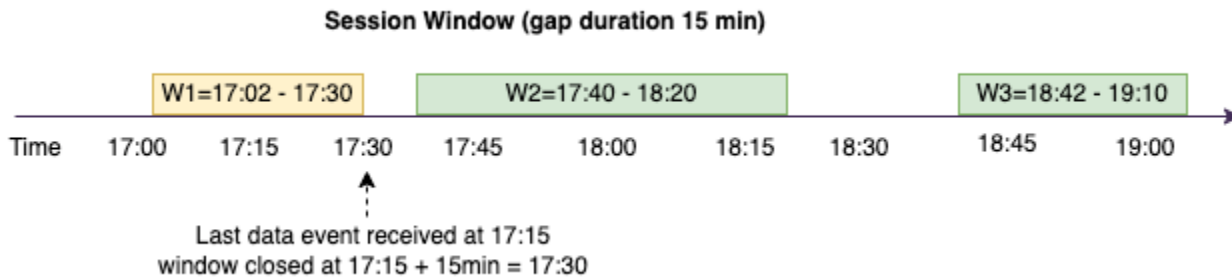
연속 기간에서 데이터 처리

group-by 절과 기간 함수는 아래와 같이 슬라이딩 기간을 구현하는 데 사용됩니다.

```
grouped_df = parsed_df \
    .groupBy(window(col("event_time"), "10 minute", "5 min"), "country",
"card_last_four") \
    .agg(sum("tx_amount").alias("total_amount"))
```

세션 기간

크기가 고정된 위의 두 기간과 달리 세션 기간은 입력에 따라 기간 길이의 크기가 정적이거나 동적일 수 있습니다. 세션 기간은 입력 데이터 이벤트로 시작하여 공백 또는 비활성 기간 내에 입력을 수신하는 한 자체적으로 계속 확장됩니다.



예를 들어 보겠습니다. ABC 호텔은 일주일 중 가장 바쁜 시간이 언제인지 알아보고 고객에게 더 나은 가격을 제공하고자 합니다. 게스트가 체크인하는 즉시 세션 창이 시작되고 스파크는 이에 대한 집계를 포함한 상태를 유지합니다. event-time-window 게스트가 체크인할 때마다 이벤트가 생성되어 에게 전송됩니다. Amazon Kinesis Data Streams 호텔은 15분 동안 체크인이 없을 경우 문을 닫을 event-time-window 수 있다는 결정을 내립니다. 새 체크인이 event-time-window 되면 다음 체크인이 다시 시작됩니다. 출력은 다음과 같습니다.

window_start_time	window_end_time	구/군/시	total_checkins
2023-07-10 17:02:00	2023-07-10 17:30:00	댈러스	50
2023-07-10 17:02:00	2023-07-10 17:30:00	시카고	25
2023-07-10 17:40:00	2023-07-10 18:20:00	댈러스	75
2023-07-10 18:50:45	2023-07-10 19:15:45	댈러스	20

첫 번째 체크인은 event_time=17:02에 발생했습니다. 집계는 17:02 에 시작됩니다 event-time-window . 이 집계는 15분 이내에 이벤트를 수신하는 한 계속됩니다. 위 예제에서 수신한 마지막 이벤트는 17:15였고 이후 15분 동안 이벤트가 없었습니다. 그 결과 스파크는 이를 17:15 +15분 = 17:30 으로 event-time-window 마감하고 17:02 - 17:30 으로 설정했습니다. 17:47 에 새로운 체크인 데이터 이벤트를 수신하면서 새로운 event-time-window 시작을 알렸습니다.

세션 기간에서 데이터 처리

group-by 절과 기간 함수는 슬라이딩 기간을 구현하는 데 사용됩니다.

```
grouped_df = parsed_df \
    .groupBy(session_window(col("event_time"), "10 minute"), "city") \
    .agg(count("check_in").alias("total_checkins"))
```

출력 모드

출력 모드는 무제한 테이블의 결과를 외부 싱크에 쓰는 모드입니다. 세 가지 모드를 사용할 수 있습니다. 다음 예제에서는 각 마이크로 배치에서 데이터 라인이 스트리밍되고 처리될 때 단어 발생 횟수를 계산합니다.

- 전체 모드 — 현재 단어 수가 업데이트되지 않았더라도 모든 마이크로 배치 처리가 끝나면 전체 결과 테이블이 싱크에 기록됩니다. event-time-window
- 추가 모드 - 이 모드는 기본 모드로, 마지막 트리거 이후 결과 테이블에 추가된 새 단어 및/또는 행만 싱크에 씁니다. 이 모드는 맵, flatMap, 필터와 같은 쿼리의 상태 비저장 스트리밍에 적합합니다.
- 업데이트 모드 - 마지막 트리거 이후 업데이트되거나 추가된 결과 테이블의 단어 및/또는 행만 싱크에 씁니다.

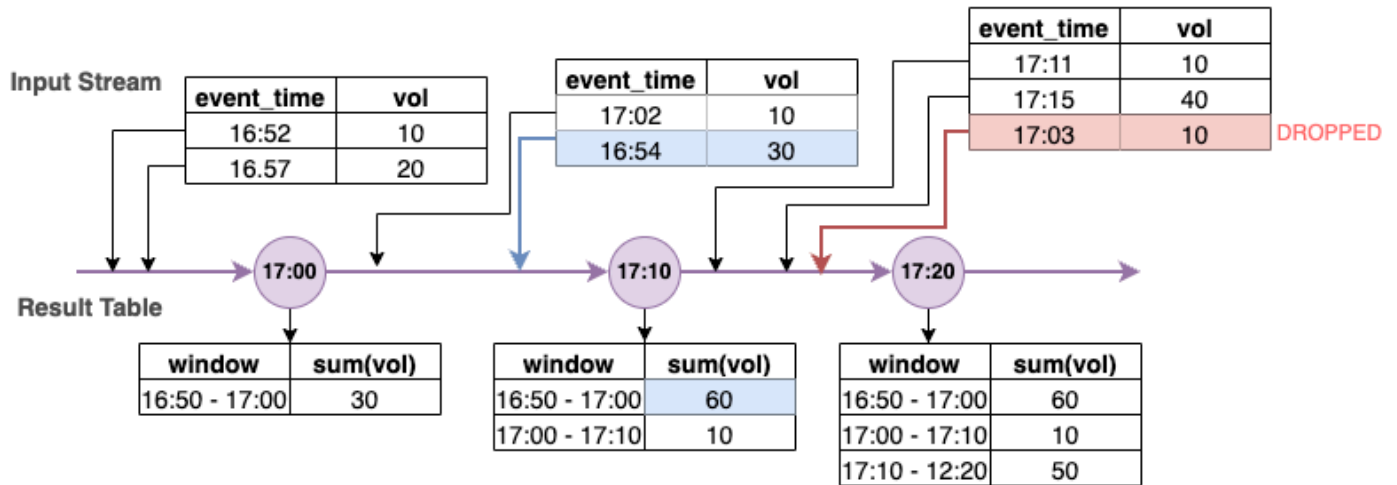
Note

출력 모드 = '업데이트'는 세션 기간에서 지원되지 않습니다.

최신 데이터 및 워터마크 처리

실시간 데이터로 작업할 때 네트워크 지연 및 업스트림 장애로 인해 데이터 도착이 지연될 수 있으므로 누락된 데이터를 다시 집계할 수 있는 메커니즘이 필요합니다. event-time-window 하지만 이렇게 하려면 상태를 유지해야 합니다. 이와 동시에 상태 크기를 제한하려면 오래된 데이터를 정리해야 합니다. Spark 버전 2.1에는 상태를 유지하고 사용자가 지연 데이터에 대한 임계값을 지정할 수 있는 워터마킹이라는 기능에 대한 지원이 추가되었습니다.

위의 주식 시세 표시기 예제를 참조하여 지연 데이터에 대한 허용 임계값을 10분 이내로 가정해 보겠습니다. 간단하게 하기 위해 연속 기간을 가정하겠습니다(티커는 AMZ, 거래는 BUY).



위 다이어그램에서는 10분의 연속 기간 동안 총 볼륨을 계산하고 있습니다. 17:00, 17:10, 17:20에 트리거가 있습니다. 타임라인 화살표 위에는 입력 데이터 스트림이 있고 아래에는 무제한 결과 테이블이 있습니다.

첫 번째 10분의 연속 기간에서는 event_time을 기준으로 집계했고, total_volume은 30으로 계산되었습니다. 두 번째 event-time-window 버전에서는 spark가 event_time= 17:02 인 첫 번째 데이터 이벤트를 수신했습니다. 이는 지금까지 Spark에서 본 최대 event_time이므로 워터마크 임계값은 10분 뒤로 설정됩니다(즉, watermark_event_time=16:52). 16:52 이후의 event_time이 있는 모든 데이터 이벤트는 시간 제한 집계로 간주되며 그 이전의 모든 데이터 이벤트는 삭제됩니다. 이를 통해 Spark는 추가 10분 동안 중간 상태를 유지하여 지연 데이터를 수용할 수 있습니다. 17:08 즈음에 Spark는 임계값 내에 있는 event_time=16:54의 이벤트를 수신했습니다. 따라서 스파크는 “16:50 - 17:00 “을 다시 계산했고 총 볼륨은 30에서 60으로 업데이트되었습니다. event-time-window

그러나 트리거 시간 17:20에 Spark가 event_time=17:15의 이벤트를 수신하면 watermark_event_time=17:05로 설정됩니다. 따라서 event_time=17:03의 지연 데이터 이벤트는 '너무 늦음'으로 간주되어 무시되었습니다.

$$\text{Watermark Boundary} = \text{Max(Event Time)} - \text{Watermark Threshold}$$

AWS Glue에서 워터마크 사용

Spark는 워터마크 경계를 통과할 때까지 외부 싱크에 데이터를 내보내거나 쓰지 않습니다. AWS Glue에서 워터마크를 구현하려면 아래 예제를 참조하세요.

```
grouped_df = parsed_df \
    .withWatermark("event_time", "10 minutes") \
```

```
.groupBy(window("event_time", "5 minutes"), "ticker") \
.agg(sum("volume").alias("total_volume"))
```

AWS Glue 스트리밍 작업 모니터링

스트리밍 작업을 모니터링하는 것은 ETL 파이프라인 구축의 중요한 부분입니다. Spark UI 외에도 Amazon CloudWatch를 사용하여 지표를 모니터링할 수도 있습니다. 다음은 AWS Glue 프레임워크에서 내보내는 스트리밍 지표 목록입니다. 모든 AWS Glue 지표의 전체 목록은 [Amazon CloudWatch 지표를 사용한 AWS Glue 모니터링](#)을 참조하세요.

AWS Glue는 구조화된 스트리밍 프레임워크를 사용하여 입력 이벤트를 처리합니다. 코드에서 Spark API를 직접 사용하거나 이러한 지표를 게시하는 GlueContext에서 제공하는 ForEachBatch를 활용할 수 있습니다. 이러한 지표를 이해하려면 먼저 windowSize를 파악해야 합니다.

windowSize: windowSize는 사용자가 제공하는 마이크로 배치 간격입니다. 기간을 60초로 지정하면 AWS Glue 스트리밍 작업이 60초 또는 그 이상(이전 배치가 완료되지 않은 경우) 대기한 후 스트리밍 소스에서 배치의 데이터를 읽고 ForEachBatch에 제공된 변환을 적용합니다. 이를 트리거 간격이라고도 합니다.

상태 및 성능 특성을 이해하기 위해 지표를 더 자세히 검토해 보겠습니다.

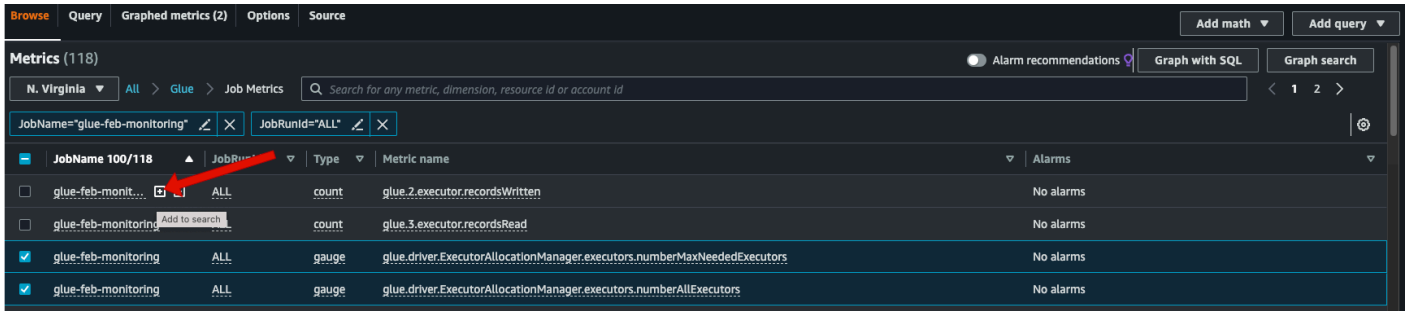
Note

지표는 30초마다 내보내집니다. windowSize가 30초 미만인 경우 보고된 지표는 집계입니다. 예를 들어, windowSize가 10초이고 마이크로 배치당 20개의 레코드를 꾸준히 처리하고 있다고 가정해 보겠습니다. 이 시나리오에서 numRecords에 대해 내보낸 지표 값은 60입니다. 사용 가능한 데이터가 없으면 지표가 내보내지지 않습니다. 또한 소비자 지연 지표의 경우 해당 지표를 가져오는 기능을 활성화해야 합니다.

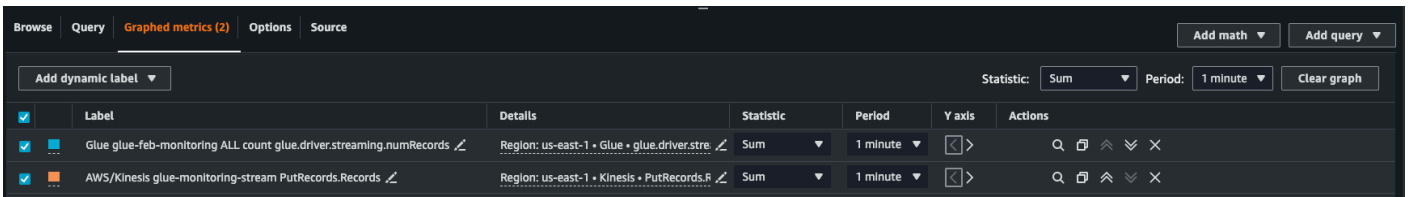
지표 시각화

시각적 지표 도표화

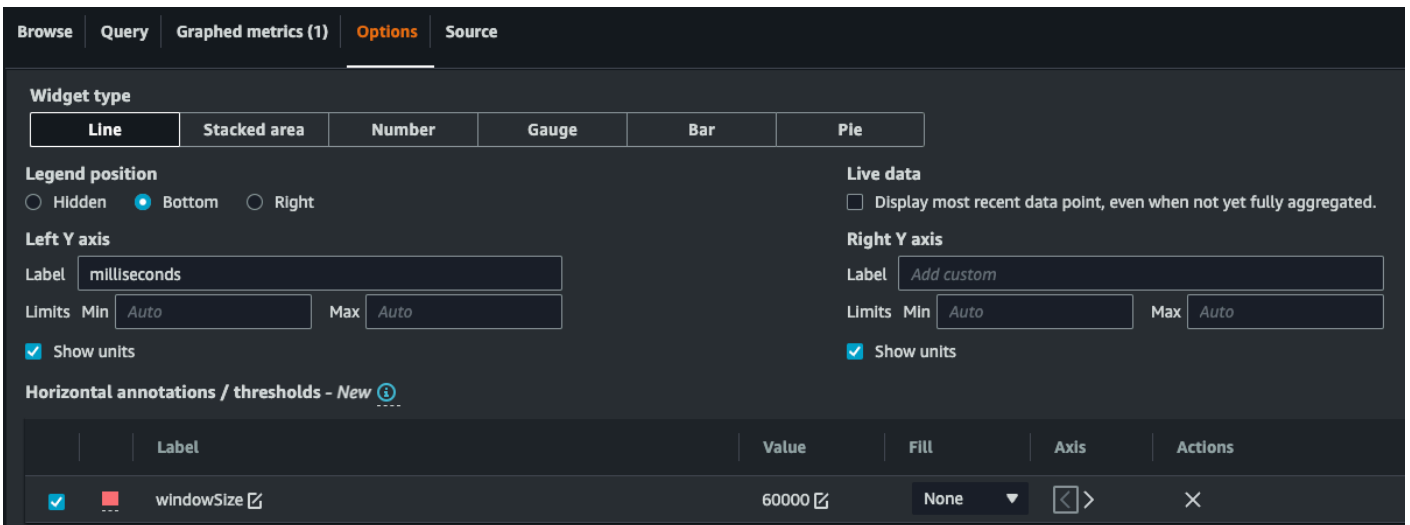
1. Amazon CloudWatch 콘솔에서 지표로 이동한 다음 찾아보기 탭을 선택합니다. 그런 다음 '사용자 지정 네임스페이스'에서 Glue를 선택합니다.



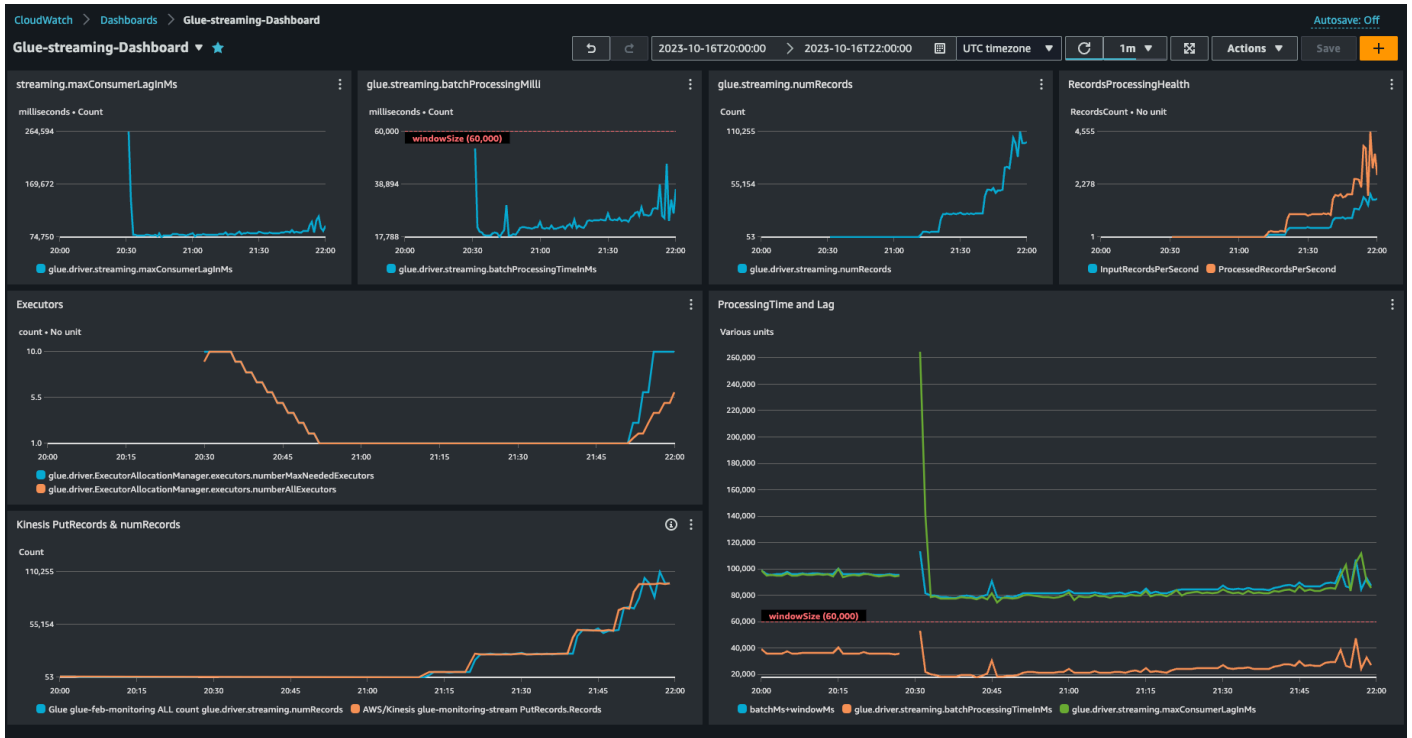
2. 작업 지표를 선택하여 모든 작업에 대한 지표를 표시합니다.
3. JobName=glue-feb-monitoring과 JobRunId=ALL을 기준으로 지표를 필터링합니다. 아래 그림에 표시된 대로 '+' 기호를 클릭하여 검색 필터에 지표를 추가할 수 있습니다.
4. 관심 있는 지표의 확인란을 선택합니다. 아래 그림에서는 numberAllExecutors와 numberMaxNeededExecutors를 선택했습니다.



5. 이러한 지표를 선택한 후에는 그래프로 표시된 지표 탭으로 이동하여 통계를 적용할 수 있습니다.
6. 지표가 1분마다 보내지기 때문에 batchProcessingTimeInMs와 maxConsumerLagInMs에 대해 1분 동안의 '평균'을 적용할 수 있습니다. numRecords에 대해 1분마다 '합계'를 적용할 수 있습니다.
7. 옵션 탭을 사용하여 그래프에 가로 windowSize 주석을 추가할 수 있습니다.



8. 지표를 선택했다면 대시보드를 생성하고 지표를 추가합니다. 다음은 샘플 대시보드입니다.

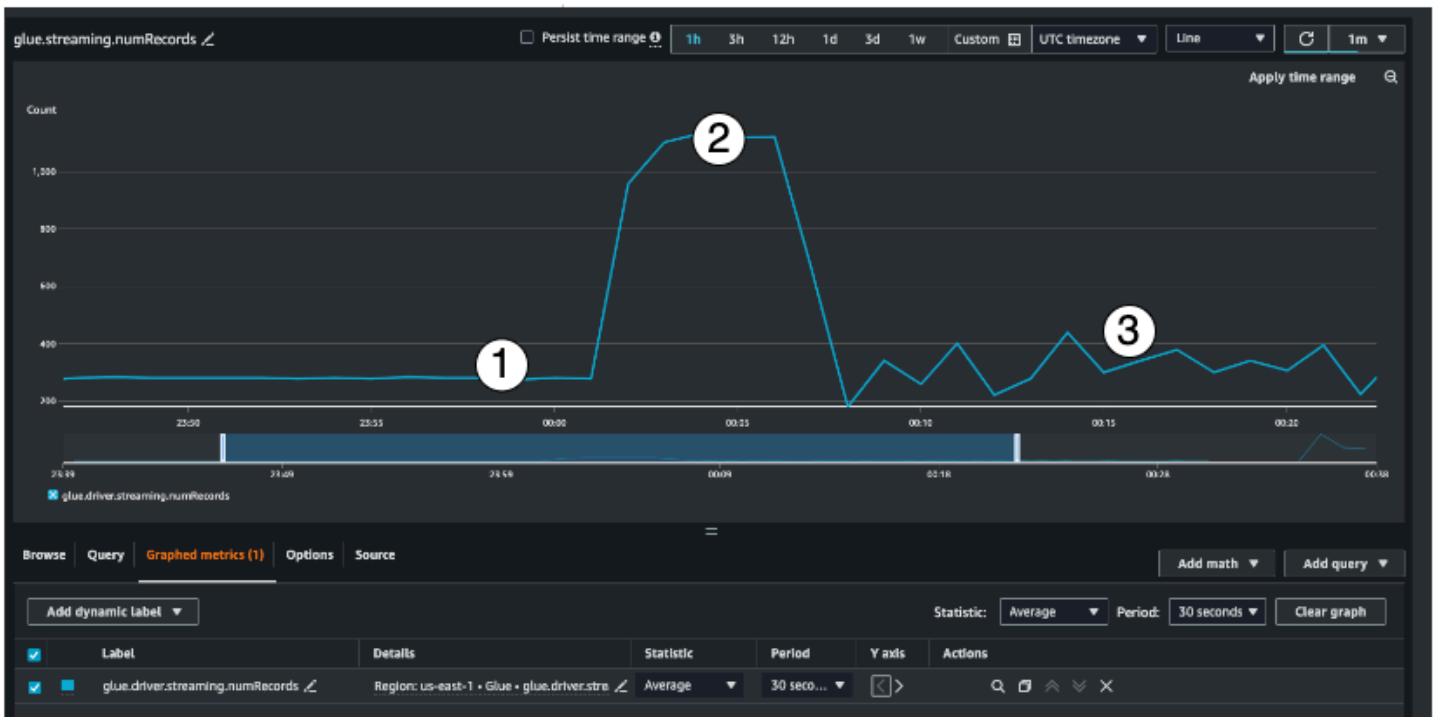


지표 심층 분석

이 섹션에서는 각 지표와 이러한 지표가 서로 어떻게 상호 연관되는지 설명합니다.

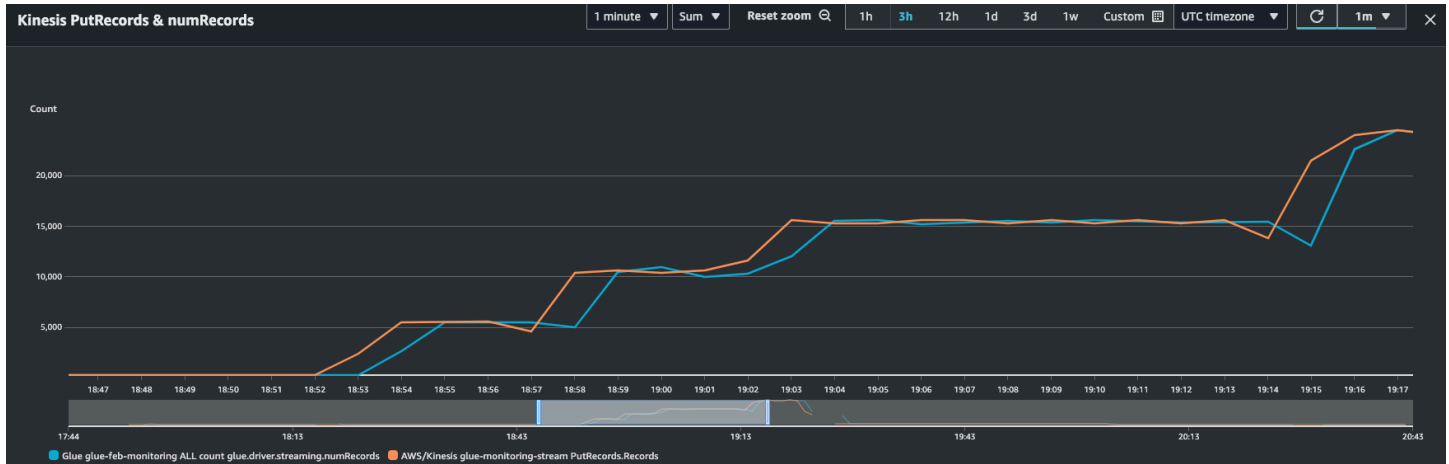
레코드 수(지표: streaming.numRecords)

이 지표는 처리 중인 레코드 수를 나타냅니다.



이 스트리밍 지표는 기간 중 처리하고 있는 레코드 수에 대한 가시성을 제공합니다. 이는 처리 중인 레코드 수와 함께 입력 트래픽의 동작을 이해하는 데도 도움이 됩니다.

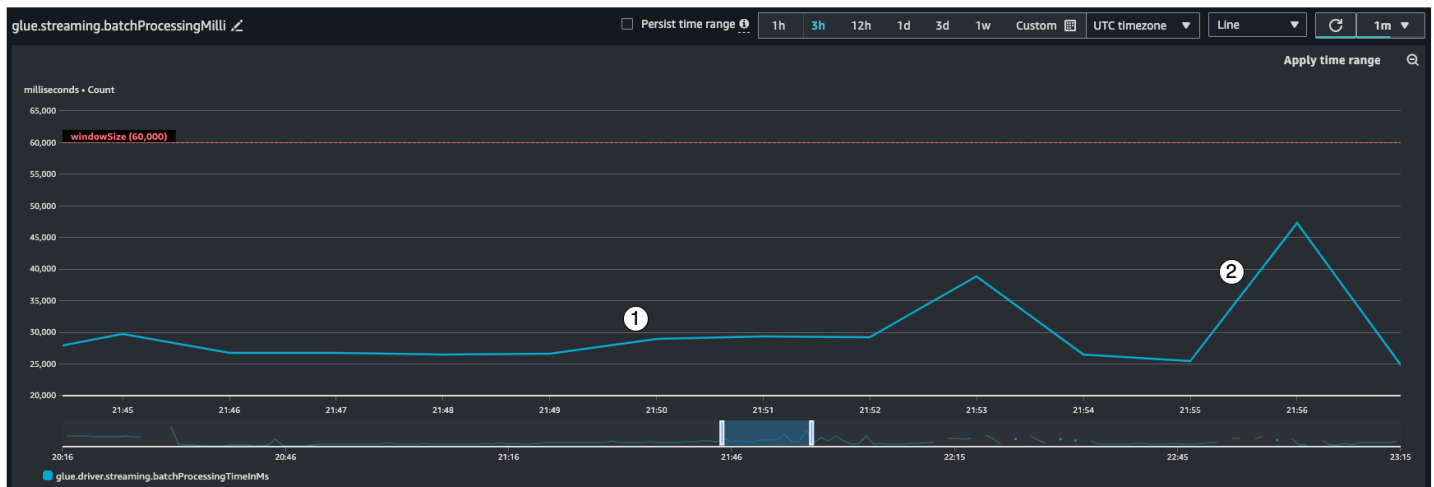
- 표시기 #1은 버스트가 없는 안정적인 트래픽의 예제를 보여줍니다. 일반적으로 이는 일정 간격으로 데이터를 수집하여 스트리밍 소스로 전송하는 IoT 센서와 같은 애플리케이션입니다.
- 표시기 #2는 안정된 부하에서 트래픽이 갑자기 급증하는 예제를 보여줍니다. 이는 블랙 프라이데이와 같은 마케팅 이벤트가 있고 클릭 수가 급증할 때 클릭스트림 애플리케이션에서 발생할 수 있습니다.
- 표시기 #3은 예측할 수 없는 트래픽의 예제를 보여줍니다. 트래픽을 예측할 수 없으면 문제가 있는 것입니다. 이는 입력 데이터의 특성일 뿐입니다. IoT 센서 예제로 돌아가서 수백 개의 센서가 스트리밍 소스로 날씨 변화 이벤트를 전송하고 있다고 가정해 보겠습니다. 날씨 변화를 예측할 수 없으므로 데이터도 예측할 수 없습니다. 트래픽 패턴을 파악하는 것이 실행기 크기 조정의 핵심입니다. 입력이 급증하는 경우 자동 크기 조정 사용을 고려할 수 있습니다(나중에 자세히 설명).



이 지표를 Kinesis PutRecords 지표와 결합하여 수집하고 있는 이벤트 수와 읽고 있는 레코드 수가 거의 동일한지 확인할 수 있습니다. 이는 지연을 이해하려고 할 때 특히 유용합니다. 수집 속도가 증가함에 따라 AWS Glue에서 읽는 numRecords도 증가합니다.

배치 처리 시간(지표: streaming.batchProcessingTimeInMs)

배치 처리 시간 지표는 클러스터가 과소 프로비저닝되었는지 과잉 프로비저닝되었는지 확인하는 데 도움이 됩니다.

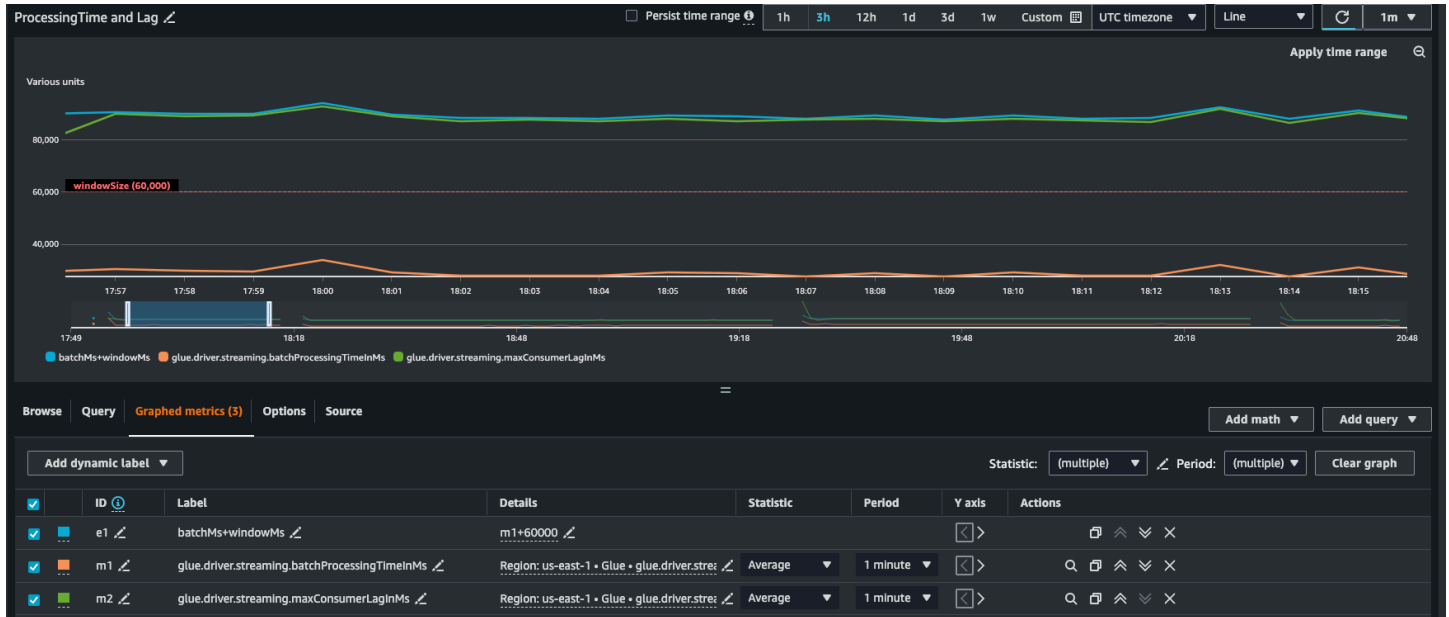


이 지표는 레코드의 각 마이크로 배치를 처리하는 데 걸린 시간(밀리초)을 나타냅니다. 여기서 주요 목표는 이 시간을 모니터링하여 windowSize 간격보다 작은지 확인하는 것입니다. 다음 기간 간격에서 복구되는 한 batchProcessingTimeInMs가 일시적으로 초과되어도 괜찮습니다. 표시기 #1은 작업을 처리하는 데 걸리는 시간이 다소 안정적임을 보여줍니다. 그러나 입력 레코드 수가 늘어나면 표시기 #2와 같이 작업을 처리하는 데 걸리는 시간이 늘어납니다. numRecords가 증가하지 않지만 처리 시간이 증가하는 경우 실행기의 작업 처리를 더 자세히 살펴봐야 합니다. batchProcessingTimeInMs가

10분 이상 120%를 넘지 않도록 임계값과 경보를 설정하는 것이 좋습니다. 경보 설정에 대한 자세한 내용은 [Amazon CloudWatch 경보 사용](#)을 참조하세요.

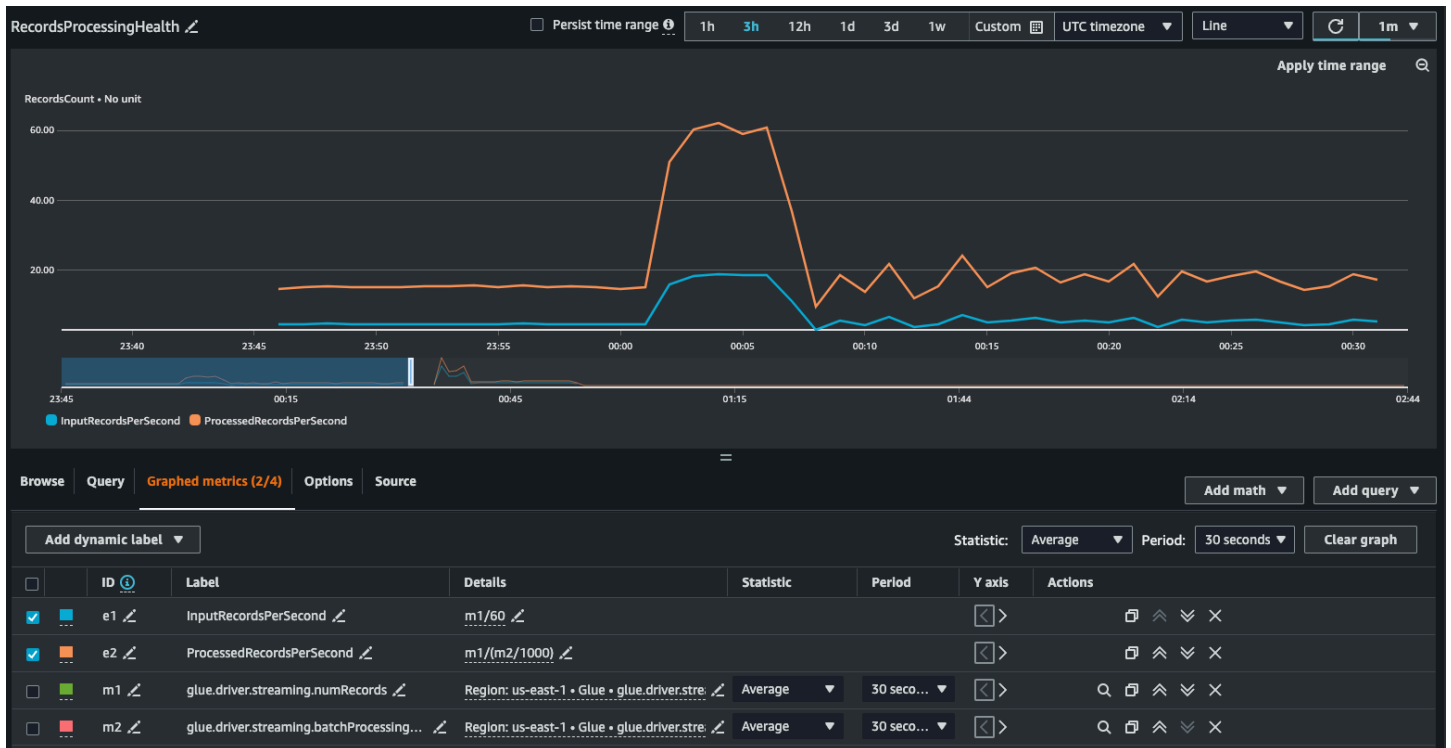
소비자 지연(지표: streaming.maxConsumerLagInMs)

소비자 지연 지표는 이벤트 처리에 지연이 있는지 파악하는 데 도움이 됩니다. 지연이 너무 길면 올바른 windowSize가 있더라도 비즈니스가 의존하는 처리 SLA를 놓칠 수 있습니다. emitConsumerLagMetrics 연결 옵션을 사용하여 이 지표를 명시적으로 활성화해야 합니다. 자세한 내용은 [KinesisStreamingSourceOptions](#)를 참조하세요.



파생된 지표

더 심층적인 인사이트를 얻으려면 Amazon CloudWatch에서 스트리밍 작업에 대해 더 자세히 이해하기 위해 파생된 지표를 생성할 수 있습니다.



파생된 지표로 그래프를 작성하여 더 많은 DPU를 사용해야 하는지 결정할 수 있습니다. 자동 크기 조정은 이를 자동으로 수행하는 데 도움이 되지만 파생된 지표를 사용하여 자동 크기 조정이 효과적으로 작동하는지 확인할 수 있습니다.

- InputRecordsPerSecond는 입력 레코드를 가져오는 속도를 나타냅니다. 입력 레코드 수 ($\text{glue.driver.streaming.numRecords}$)/Window Size로 파생됩니다.
- ProcessingRecordsPerSecond는 레코드가 처리되는 속도를 나타냅니다. 입력 레코드 수 ($\text{glue.driver.streaming.numRecords}$)/batchProcessingTimeInMs로 파생됩니다.

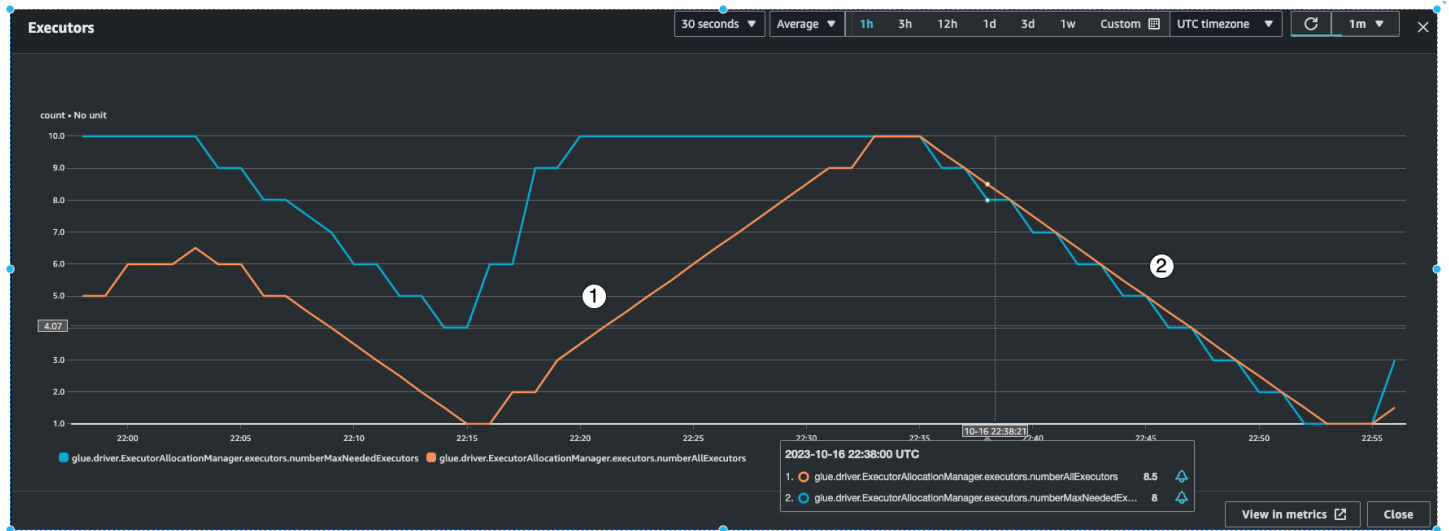
입력 속도가 처리 속도보다 높으면 작업을 처리하기 위해 용량을 더 추가하거나 병렬 처리를 늘려야 할 수 있습니다.

자동 크기 조정 지표

입력 트래픽이 급증하는 경우 자동 크기 조정 활성화를 고려하고 최대 워커 수를 지정해야 합니다. 이를 통해 numberAllExecutors와 numberMaxNeededExecutors의 두 가지 추가 지표를 얻을 수 있습니다.

- numberAllExecutors는 능동적으로 실행 중인 작업 실행기 수입니다.
- numberMaxNeededExecutors는 현재 로드를 충족하는 데 필요한 최대 작업 (능동 실행 및 보류) 실행기 수입니다.

이 두 지표는 자동 크기 조절이 제대로 작동하는지 파악하는 데 도움이 됩니다.



AWS Glue는 몇 가지 마이크로 배치를 통해 `batchProcessingTimeInMs` 지표를 모니터링하고 두 가지 중 하나를 수행합니다. `batchProcessingTimeInMs`가 `windowSize`에 더 가까울 경우 실행기를 스케일 아웃하고, `batchProcessingTimeInMs`가 `windowSize`보다 비교적 낮을 경우 실행기를 스케일 인합니다. 또한 실행기의 단계적 크기 조절을 위한 알고리즘을 사용합니다.

- 표시기 #1은 로드 처리에 필요한 최대 실행기를 따라잡기 위해 활성 실행기가 어떻게 스케일 업되었는지 보여줍니다.
- 표시기 #2는 `batchProcessingTimeInMs`가 낮았기 때문에 활성 실행기가 어떻게 스케일 인되었는지 보여줍니다.

이러한 지표를 사용하여 현재 실행기 수준의 병렬 처리를 모니터링하고 이에 따라 자동 크기 조절 구성의 최대 워커 수를 조정할 수 있습니다.

최고의 성능을 얻는 방법

Spark는 Amazon Kinesis 스트림에서 읽기 위해 샤드당 하나의 작업을 생성하려고 합니다. 각 샤드의 데이터가 파티션이 됩니다. 그런 다음 각 워커의 코어 수에 따라 이러한 작업을 실행기/워커 전체에 분배합니다(워커당 코어 수는 선택하는 워커 유형(G.025X, G.1X 등)에 따라 다름). 그러나 작업이 분배되는 방식은 비결정적입니다. 모든 작업은 해당 코어에서 병렬로 실행됩니다. 샤드가 사용 가능한 실행기 코어 수보다 많으면 작업이 대기열에 추가됩니다.

위의 지표와 샤드 수를 조합해서 사용하여 버스트를 위한 여유 공간이 있는 안정적인 로드를 위해 실행기를 프로비저닝할 수 있습니다. 대략적인 워커 수 확인을 위해 작업을 몇 번 반복해서 실행하는 것이

좋습니다. 불안정하거나 급증하는 워크로드의 경우 자동 크기 조정과 최대 워커 수를 설정하여 동일한 작업을 수행할 수 있습니다.

비즈니스의 SLA 요구 사항에 따라 `windowSize`를 설정합니다. 예를 들어, 비즈니스상 처리된 데이터가 120초를 초과해서는 안 되는 경우, 평균 소비자 지연이 120초 미만이 되도록 `windowSize`를 60초 이상으로 설정합니다(위의 소비자 지연 섹션 참조). 여기에서 샤드의 `numRecords`와 수에 따라 DPU 단위로 용량을 계획하여 `batchProcessingTimeInMs`가 대부분의 경우 `windowSize`의 70% 미만이 되도록 합니다.

Note

핫 샤드는 데이터 스큐를 일으킬 수 있으며, 이는 일부 샤드/파티션이 다른 샤드/파티션보다 훨씬 크다는 것을 의미합니다. 이로 인해 병렬로 실행되는 일부 작업의 경우 시간이 오래 걸려 작업이 지연될 수 있습니다. 결과적으로 이전 배치의 모든 작업이 완료될 때까지 다음 배치를 시작할 수 없으며 이는 `batchProcessingTimeInMillis`와 최대 지연에 영향을 미칩니다.

AWS Glue 데이터 품질

AWS Glue 데이터 품질을 사용하면 데이터 품질을 측정하고 모니터링하여 현명한 비즈니스 결정을 내릴 수 있습니다. 오픈 소스 DeeQu 프레임워크를 기반으로 구축된 AWS Glue Data Quality는 관리형 서버리스 환경을 제공합니다. AWS Glue 데이터 품질은 데이터 품질 규칙을 정의하는 데 사용하는 도메인별 언어인 데이터 품질 정의 언어 (DQDL) 와 함께 작동합니다. DQDL 및 지원되는 규칙 유형에 대한 자세한 내용은 [데이터 품질 정의 언어\(DQDL\) 참조](#) 섹션을 참조하세요.

추가 제품 세부 정보 및 요금은 [AWS Glue Data Quality](#)의 서비스 페이지를 참조하세요.

이점 및 주요 특징

AWS Glue 데이터 품질의 이점 및 주요 기능은 다음과 같습니다.

- 서버리스 - 설치, 패치 또는 유지 관리가 필요하지 않습니다.
- 빠르게 시작하기 — AWS Glue Data Quality는 데이터를 빠르게 분석하고 사용자를 위한 데이터 품질 규칙을 생성합니다. '데이터 품질 규칙 생성 → 권장 규칙'을 두 번만 클릭하여 시작할 수 있습니다.
- 데이터 품질 문제 감지 — 기계 학습 (ML) 을 사용하여 이상 현상과 hard-to-detect 데이터 품질 문제를 감지합니다.
- 규칙을 즉흥적으로 개선하세요. 25개 이상의 out-of-the-box DQ 규칙부터 시작하여 특정 요구 사항에 맞는 규칙을 만들 수 있습니다.
- 품질 평가 및 신뢰할 수 있는 비즈니스 의사 결정 지원 - 규칙을 평가한 후에는 데이터 상태에 대한 개요를 제공하는 Data Quality 점수를 제공합니다. Data Quality 점수를 사용하여 신뢰할 수 있는 비즈니스 의사 결정을 지원합니다.
- 잘못된 데이터에 집중 - AWS Glue 데이터 품질을 사용하면 품질 평가 점수를 떨어뜨린 정확한 기록을 식별할 수 있습니다. 데이터를 쉽게 식별하고 격리한 후 수정할 수 있습니다.
- 사용한 만큼 지불 — AWS Glue Data Quality를 사용하는 데 필요한 연간 라이선스는 없습니다.
- 종속 없음 — AWS Glue Data Quality는 오픈 소스를 DeeQu 기반으로 구축되므로 작성 중인 규칙을 공개 언어로 유지할 수 있습니다.
- 데이터 품질 검사 — AWS Glue 데이터 품질 Data Catalog 및 AWS Glue ETL 파이프라인을 대상으로 데이터 품질 검사를 시행하여 저장 중인 데이터 및 전송 중인 데이터 품질을 관리할 수 있습니다.
- ML 기반 데이터 품질 탐지 — 기계 학습 (ML) 을 사용하여 이상 현상과 데이터 품질 문제를 탐지합니다. hard-to-detect

작동 방식

AWS Glue 데이터 품질에는 두 가지 진입점이 있습니다. 바로 ET 및 ETL 작업입니다. AWS Glue Data Catalog AWS Glue 이 섹션에서는 각 진입점이 지원하는 사용 사례 및 AWS Glue 기능에 대한 개요를 제공합니다.

의 데이터 품질 AWS Glue Data Catalog

AWS Glue Data Quality는 코더가 아닌 사용자도 데이터 품질 규칙을 쉽게 설정할 수 있는 방법에 저장되어 있는 개체를 평가합니다. AWS Glue Data Catalog 이러한 페르소나로, 데이터 관리자 및 비즈니스 분석가가 해당됩니다.

다음과 같은 사용 사례에서 이 옵션을 선택할 수 있습니다.

- AWS Glue Data Catalog에서 이미 카탈로그화한 데이터 세트에 대해 데이터 품질 작업을 수행하려고 합니다.
- 데이터 거버넌스 관련 업무를 수행하면서 데이터 레이크의 데이터 품질 문제를 지속적으로 식별하거나 평가해야 합니다.

다음 인터페이스를 사용하여 데이터 카탈로그의 데이터 품질을 관리할 수 있습니다.

- 관리 콘솔 AWS Glue
- AWS Glue API

AWS Glue 데이터 품질을 시작하려면 [AWS Glue Data Catalog 참조하십시오](#) [Data Catalog에서 AWS Glue Data Quality 시작하기](#).

AWS Glue ETL 작업의 데이터 품질

AWS Glue AWS Glue ETL 작업의 데이터 품질을 통해 사전 예방적인 데이터 품질 작업을 수행할 수 있습니다. 사전 예방적 작업을 사용하면 데이터 세트를 데이터 레이크에 로드하기 전에 잘못된 데이터를 식별하고 필터링할 수 있습니다.

[비디오: ETL AWS Glue 파이프라인의 데이터 품질 소개](#)

다음 사용 사례에서 ETL 작업에 대한 데이터 품질을 선택할 수 있습니다.

- 데이터 품질 작업을 ETL 작업에 통합하려고 합니다.

- ETL 스크립트에서 데이터 품질 작업을 정의하는 코드를 작성하려고 합니다.
- 시각적 데이터 파이프라인에서 이동하는 데이터의 품질을 관리하려고 합니다.

다음 인터페이스를 사용하여 ETL 작업에 대한 데이터 품질을 관리할 수 있습니다.

- AWS Glue Studio, AWS Glue Studio 노트북, 대화형 세션 AWS Glue
- AWS Glue ETL 스크립팅용 라이브러리
- AWS Glue API

ETL 작업에 대한 Data Quality를 시작하려면 AWS Glue Studio 사용 설명서의 [자습서: Data Quality 시작하기](#)를 참조하세요.

데이터 카탈로그에 대한 데이터 품질과 ETL 작업에 대한 데이터 품질 비교

이 표는 AWS Glue 데이터 품질의 각 진입점이 지원하는 기능에 대한 개요를 제공합니다.

기능	데이터 카탈로그에 대한 데이터 품질	ETL 작업에 대한 데이터 품질
데이터 소스	Amazon S3, Amazon Redshift, 데이터 카탈로그와 호환되는 JDBC 소스, 그리고 Apache Iceberg, Apache Hudi 및 Delta Lake와 같은 트랜잭션 데이터 레이크 형식. 테이블을 AWS Lake Formation 관리하는 경우 Iceberg, Delta 및 HUDI 테이블은 지원되지 않는다는 점에 유의하십시오. Amazon Athena에 AWS Glue Data Catalog 카탈로그가 있는 뷰는 지원되지 않습니다.	에서 AWS Glue 지원하는 모든 데이터 소스 (사용자 지정 커넥터 및 타사 커넥터 포함)
Data Quality 규칙 권장 사항	지원	지원되지 않음
DQDL 규칙 작성 및 실행	지원	지원

기능	데이터 카탈로그에 대한 데이터 품질	ETL 작업에 대한 데이터 품질
Auto Scaling	지원되지 않음	지원
AWS Glue Flex 지원	지원되지 않음	지원
일정 예약	Step Functions를 통해 데이터 품질 규칙을 평가할 때 지원됩니다.	Step Functions와 워크플로를 사용할 때 지원됩니다.
데이터 품질 검사에 실패한 레코드 식별	지원되지 않음	지원
Amazon EventBridge와 통합	지원	지원
AWS 클라우드워치와의 통합	지원	지원
Amazon S3에 데이터 품질 결과 작성	지원	지원
중분 데이터 품질	푸시다운 조건자를 통해 지원됨	북마크를 통해 지원 AWS Glue
AWS CloudFormation 지원	지원	지원
ML 기반 이상 탐지	지원되지 않음	미리 보기
동적 규칙	지원되지 않음	지원

고려 사항

AWS Glue 데이터 품질을 사용하기 전에 다음 항목을 고려하십시오.

- 데이터 품질 규칙은 중첩된 데이터 소스 또는 목록 유형 데이터 소스를 평가할 수 없습니다. [중첩된 구조체 평면화](#) 섹션을 참조하십시오.

용어

다음 목록은 AWS Glue 데이터 품질과 관련된 용어를 정의합니다.

DQDL(데이터 품질 정의 언어)

AWS Glue 데이터 품질 규칙을 작성하는 데 사용할 수 있는 도메인별 언어입니다.

DQDL에 대한 자세한 내용은 [데이터 품질 정의 언어\(DQDL\) 참조](#) 안내서를 참조하세요.

데이터 품질

데이터세트가 특정 용도에 얼마나 잘 부합하는지 설명합니다. AWS Glue 데이터 품질은 데이터세트를 기준으로 규칙을 평가하여 데이터 품질을 측정합니다. 각 규칙은 데이터 최신성 또는 무결성과 같은 특정 특성을 확인합니다. 데이터 품질을 정량화하려면 데이터 품질 점수를 사용할 수 있습니다.

데이터 품질 점수

데이터 품질로 규칙 세트를 평가할 때 통과한 데이터 품질 규칙 (결과 충족) 의 백분율입니다. AWS Glue

규칙

데이터에 특정 특성이 있는지 검사하고 부울 값을 반환하는 DQDL 표현식입니다. 자세한 정보는 [규칙 구조](#)을 참조하세요.

분석기

데이터 통계를 수집하는 DQDL 표현식입니다. 분석기는 ML 알고리즘이 시간 경과에 따른 이상 현상과 데이터 품질 문제를 탐지하는 데 사용할 수 있는 데이터 통계를 수집합니다. hard-to-detect

규칙 세트

일련의 데이터 AWS Glue 품질 규칙으로 구성된 리소스입니다. 규칙 세트는 AWS Glue Data Catalog의 테이블과 연결되어야 합니다. 규칙 세트를 저장할 때 AWS Glue에서는 Amazon 리소스 이름(ARN)을 규칙 세트에 할당합니다.

데이터 품질 점수

AWS Glue 데이터 품질로 규칙 세트를 평가할 때 통과한 데이터 품질 규칙(true 결과)의 비율입니다.

관찰

AWS Glue에서 시간이 지남에 따라 규칙과 분석기로부터 수집된 데이터 통계를 분석하여 생성되는 확인되지 않은 인사이트입니다.

Limits

AWS Glue 데이터 품질 서비스 제한:

- 규칙 세트에 2000개의 규칙을 포함할 수 있습니다. 규칙 세트가 더 크면 여러 규칙 세트로 분할하는 것이 좋습니다.
- 규칙 세트의 크기는 65KB입니다. 규칙 세트가 더 크면 여러 규칙 세트로 분할하는 것이 좋습니다.

데이터 품질에 대한 릴리스 노트 AWS Glue

이 항목에서는 AWS Glue 데이터 품질에 도입된 기능에 대해 설명합니다.

정식 출시: 새 기능

AWS Glue Data Quality의 일반 버전과 함께 다음과 같은 새 기능을 사용할 수 있습니다.

- 데이터 품질 검사에 실패한 레코드를 식별하는 기능이 이제 에서 지원됩니다. AWS Glue Studio
- 두 데이터 세트 간 데이터의 참조 무결성 검증, 두 데이터 세트 간 데이터 비교, 데이터 형식 검사와 같은 새로운 데이터 품질 규칙 유형
- 의 사용자 경험 개선 AWS Glue Data Catalog
- Apache Iceberg, Apache Hudi 및 Delta Lake에 대한 지원
- Amazon Redshift에 대한 지원
- Amazon을 통한 간소화된 알림 EventBridge
- AWS CloudFormation 규칙 세트 생성 지원
- 성능 개선: ETL의 캐싱 옵션 및 데이터 품질 평가 AWS Glue Studio 시 더 빠른 성능 제공

2023년 11월 27일(미리 보기)

- ML 기반 이상 탐지 기능은 이제 AWS Glue ETL 및 AWS Glue Studio에서 사용할 수 있습니다. 이제 이를 통해 이상 현상과 데이터 품질 문제를 감지할 수 있습니다. [hard-to-detect](#)
- [동적 규칙을 사용하면 동적 임계값\(예: RowCount > avg\(last\(10\)\)\)을 제공할 수 있습니다.](#)

2024년 3월 12일

- DQDL 개선 사항

- [NULL, BLANKS, WHITESPACES_ONLY와 같은 키워드에 대한 지원](#)
- [AWS Glue Data Quality가 복합 규칙을 처리하는 방법을 지정하는 옵션](#)
- [ColumnValues 규칙 유형은 비교 중에 NULL 값을 전달하는 것을 허용하지 않습니다.](#)
- [DQDL의 NOT 연산자에 대한 지원](#)

2024년 6월 26일

- DQDL 개선 사항
 - DQDL은 이제 [where 절](#)을 지원하므로 DQ 규칙을 적용하기 전에 데이터를 필터링할 수 있습니다.

AWS Glue Data Quality의 이상 탐지

Note

AWS Glue Data Quality는 다음 리전에서 평가판으로 사용 가능합니다.

- 미국 동부(오하이오, 버지니아 북부)
- 미국 서부(오레곤)
- 아시아 태평양(도쿄)
- 유럽(아일랜드)

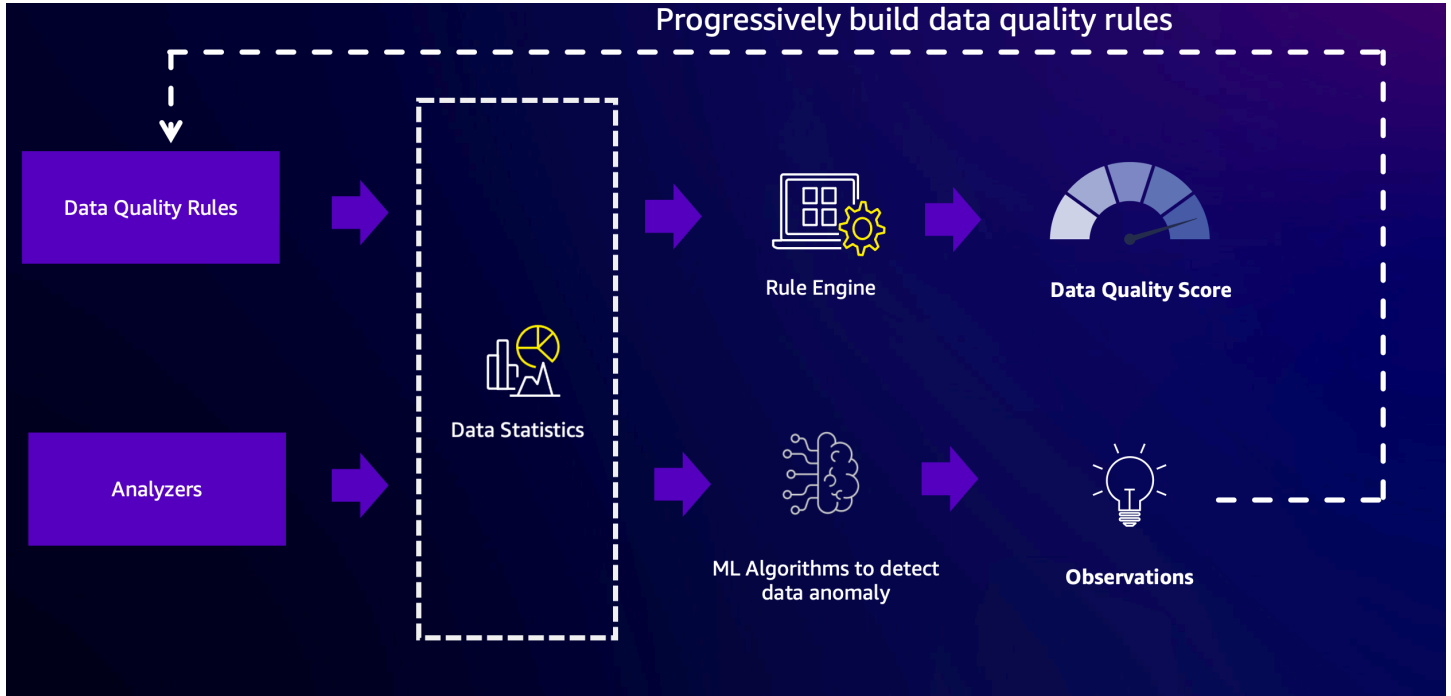
AWS Glue Data Quality 이상 탐지는 시간 경과에 따른 데이터 통계에 기계 학습(ML) 알고리즘을 적용하여 규칙으로 탐지하기 어려운 비정상적인 패턴과 숨겨진 데이터 품질 문제를 탐지합니다. 현재 이상 탐지 기능은 AWS Glue 4.0에서만 사용할 수 있습니다. 이 기능은 현재 AWS Glue Studio Visual ETL 및 AWS Glue ETL에서만 사용할 수 있습니다. 이 기능은 AWS Glue Studio 노트북, AWS Glue 데이터 카탈로그, AWS Glue 대화형 세션, AWS Glue 데이터 미리 보기에서는 작동하지 않습니다.

작동 방식

Data Quality 규칙을 평가할 때 AWS Glue는 데이터가 규칙을 준수하는지 여부를 결정하는 데 필요한 데이터 통계를 캡처합니다. 예를 들어 Data Quality는 데이터 세트의 고유값 수를 계산한 다음 해당 값을 예상 값과 비교합니다.

Data Quality 규칙 엔진은 통계 값을 정의된 임계값과 비교하여 품질 요구 사항을 평가합니다. 이러한 통계는 시간이 지남에 따라 수집되므로 ETL 파이프라인에서 이상 탐지를 활성화하여 AWS Glue가 과

거 통계에서 학습하고 숨겨진 패턴을 관찰 결과로 보고하도록 할 수 있습니다. 관찰 결과는 AWS Glue의 ML 알고리즘이 식별한 확인되지 않은 인사이트입니다. 발견된 패턴을 모니터링하기 위해 규칙 세트에 적용할 수 있는 권장 Data Quality 규칙이 함께 제공됩니다. 정기적인 일정(예: 시간별, 일별)에 따라 작업을 실행하는 것이 좋습니다. 불규칙적으로 실행되면 인사이트가 제대로 생성되지 않을 수 있습니다.



분석기를 사용하여 데이터 검사

때로는 데이터 품질 규칙을 작성할 시간이 없을 수도 있습니다. 바로 이런 경우에 분석기가 유용합니다. 분석기는 규칙 세트의 일부이며 구성이 매우 간단합니다. 예를 들어 규칙 세트에 다음과 같이 작성할 수 있습니다.

```

Analyzers = [
    RowCount,
    Completeness "AllColumns"
]

```

그러면 다음과 같은 통계가 수집됩니다.

- 전체 데이터 세트의 행 수
- 데이터 세트에 있는 모든 열의 완전성

임꺽값에 대해 걱정할 필요가 없으므로 분석기를 사용하는 것이 좋습니다. 데이터 파이프라인을 실행할 수 있으며 3회 실행 후 AWS Glue Data Quality에서 이상이 탐지되면 관찰 결과 및 규칙 권장 사항을 생성하기 시작합니다. 관찰 결과와 관련 통계를 검토하고 규칙 세트에 규칙 권장 사항을 쉽게 통합할 수 있습니다. 시작하려면 [이상 탐지 구성 및 인사이트 생성](#) 섹션을 참조하세요. 분석기는 데이터 품질 점수에 영향을 주지 않음에 유의하세요. 시간 경과에 따라 분석할 수 있는 통계를 생성하여 관찰 결과를 생성합니다.

DetectAnomaly 규칙 사용

때때로 이상을 탐지했을 때 작업이 실패하기를 원할 때가 있습니다. 제약 조건을 적용하려면 규칙을 구성해야 합니다. 분석기는 작업을 중단하지 않습니다. 대신 통계를 수집하고 데이터를 분석합니다. 규칙 세트의 규칙 섹션에서 DetectAnomaly 규칙을 구성하면 DQ 스캔에서 작업이 스캔의 일부 규칙을 통과하지 못했다고 보고하는 것을 확인할 수 있습니다.

이상 탐지의 이점 및 사용 사례

엔지니어는 언제든지 수백 개의 데이터 파이프라인을 관리할 수 있습니다. 각 파이프라인은 서로 다른 소스에서 데이터를 추출하여 데이터 레이크에 로드할 수 있습니다. 각 파이프라인이 서로 다른 소스에서 데이터를 추출하여 데이터 레이크에 로드할 수 있으므로 데이터의 형태가 크게 바뀌었는지, 기존 트랜드에서 벗어났는지 등 데이터에 대한 즉각적인 피드백을 얻기가 어렵습니다.

과거에는 업스트림 데이터 원본이 데이터 엔지니어링 팀에 아무런 경고 없이 변경되어 이 프로세스에 hard-to-track “데이터 버그”가 발생했습니다. 작업에 Data Quality 노드를 추가하면 문제가 발견될 때 작업이 실패하기 때문에 훨씬 더 간편하게 작업을 수행할 수 있습니다. 하지만 이렇게 한다고 해서 데이터 팀이 걱정하는 모든 장애 모드가 제거되는 것은 아니며, 다른 데이터 버그가 발생할 가능성이 있습니다.

한 가지 장애 모드는 데이터 볼륨과 관련이 있습니다. 시간이 지나면서 회사의 데이터 스토어가 커짐에 따라 데이터 파이프라인에서 생성되는 기록량이 기하급수적으로 증가할 수 있습니다. 매주 데이터 팀은 수집되는 행 수에 제한을 설정하는 각 Data Quality 규칙을 늘리기 위해 ETL 작업을 수동으로 업데이트해야 할 수 있습니다.

또 다른 장애 모드는 광범위한 일부 데이터 품질 규칙 제한이며 이는 요일별로 거래량이 다르다는 사실을 수용하기 위한 결과입니다. 주말에는 거래가 거의 없고 월요일에는 다른 평일보다 약 3배 정도 거래가 많습니다. 데이터 팀에는 두 가지 옵션이 있습니다. 해당 날짜의 상황에 따라 즉시 규칙 세트를 변경하는 로직을 구현하거나 기대치를 매우 넓게 설정하는 것입니다.

마지막으로 데이터 팀은 잘 정의되지 않은 데이터 버그에 대해서도 우려하고 있습니다. 모델은 특정 특성이 있는 데이터로 학습되므로, 팀에서는 데이터가 예상하지 못한 방식으로 왜곡되기 시작할 때 이를 파악하기를 원합니다. 예를 들어 2월에 회사가 몬태나주로 확장하면 ‘MT’ 코드를 포함한 거래가 더 자

주 발생하기 시작합니다. 이로 인해 ML 추론이 깨질 수 있으며 그 결과 모델은 몬테나의 모든 거래를 허위라고 잘못 예측하게 됩니다.

Data Quality 이상 탐지가 이러한 문제를 해결하는 데 도움이 될 수 있습니다. Data Quality 이상 탐지의 몇 가지 이점은 다음과 같습니다.

- 예약, 이벤트 기반 또는 수동으로 데이터를 스캔합니다.
- 의도하지 않은 이벤트, 계절성 또는 통계적 이상을 나타낼 수 있는 이상을 탐지합니다.
- Data Quality 이상 탐지로 발견된 관찰 결과에 조치를 취하기 위한 규칙 권장 사항을 제공합니다.

이는 다음과 같은 경우에 유용합니다.

- 데이터 품질 규칙을 작성할 필요 없이 데이터 이상을 자동으로 탐지하고자 합니다.
- 데이터 품질 규칙만으로는 발견할 수 없는 데이터의 잠재적 문제를 포착하고자 합니다.
- 데이터 품질 모니터링을 위해 수집되는 행 수를 제한하는 등 시간이 지남에 따라 진화하는 일부 작업을 자동화하려고 합니다.

AWS Glue Data Quality에 대한 IAM 권한 구성

이 주제에서는 IAM 관리자가 AWS Glue Data Quality에 대한 AWS Identity and Access Management(IAM) 정책에서 사용할 수 있는 작업과 리소스를 이해하는 데 도움이 되는 정보를 제공합니다. 또한 AWS Glue 데이터 카탈로그와 함께 AWS Glue Data Quality를 사용하는 데 필요한 최소 권한이 포함된 샘플 IAM 정책도 포함되어 있습니다.

AWS Glue의 보안에 대한 자세한 내용은 [AWS Glue의 보안](#)을(를) 참조하세요.

AWS Glue Data Quality에 대한 IAM 권한

다음 테이블에는 특정 AWS Glue Data Quality 작업을 수행하는 데 필요한 권한이 나와 있습니다. AWS Glue Data Quality에 대한 세분화된 권한 부여를 설정하려면 IAM 정책 명령문의 Action 요소에서 이러한 작업을 지정할 수 있습니다.

AWS Glue 데이터 품질 작업

작업	설명	리소스 유형
glue:CreateDataQualityRuleset	데이터 품질 규칙 세트를 생성할 수 있는 권한을 부여합니다.	::dataQualityRuleset/<name>

작업	설명	리소스 유형
<code>glue:DeleteDataQualityRuleset</code>	데이터 품질 규칙 세트를 삭제할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/<name></code>
<code>glue:GetDataQualityRuleset</code>	데이터 품질 규칙 세트를 검색할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/<name></code>
<code>glue:ListDataQualityRulesets</code>	모든 데이터 품질 규칙 세트를 검색할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>
<code>glue:UpdateDataQualityRuleset</code>	데이터 품질 규칙 세트를 업데이트할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/<name></code>
<code>glue:GetDataQualityResult</code>	데이터 품질 작업 실행 결과를 검색할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/<name></code>
<code>glue:ListDataQualityResults</code>	모든 데이터 품질 작업 실행 결과를 검색할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>
<code>glue:CancelDataQualityRuleRecommendationRun</code>	진행 중인 데이터 품질 권장 작업 실행을 중지할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>
<code>glue:GetDataQualityRuleRecommendationRun</code>	데이터 품질 권장 작업 실행을 검색할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>
<code>glue:ListDataQualityRuleRecommendationRuns</code>	모든 데이터 품질 권장 작업 실행을 검색할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>
<code>glue:StartDataQualityRuleRecommendationRun</code>	데이터 품질 권장 작업 실행을 시작할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>

작업	설명	리소스 유형
<code>glue:CancelDataQualityRulesetEvaluationRun</code>	진행 중인 데이터 품질 작업 실행을 중지할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>
<code>glue:GetDataQualityRulesetEvaluationRun</code>	데이터 품질 작업 실행을 검색할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>
<code>glue:ListDataQualityRulesetEvaluationRuns</code>	모든 데이터 품질 작업 실행을 검색할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/*</code>
<code>glue:StartDataQualityRulesetEvaluationRun</code>	데이터 품질 작업 실행을 시작할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/<name></code>
<code>glue:PublishDataQuality</code>	데이터 품질 결과를 게시할 수 있는 권한을 부여합니다.	<code>::dataQualityRuleset/<name></code>

평가 실행을 예약하는 데 필요한 IAM 설정

IAM 권한

예약된 데이터 품질 평가를 실행하려면 권한 정책에 IAM:PassRole 작업을 추가해야 합니다.

AWS EventBridge 스케줄러 필수 권한

작업	설명	리소스 유형
<code>iam:PassRole</code>	사용자가 승인된 역할을 전달할 수 있도록 IAM에 권한을 부여합니다.	<code>StartDataQualityRulesetEvaluationRun</code> 직접 호출에 사용된 역할의 ARN

이 권한이 없으면 다음과 같은 오류가 발생합니다.

```
"errorCode": "AccessDenied"
"errorMessage": "User: arn:aws:sts::account_id:assumed-role/AWSGlueServiceRole is not
authorized to perform: iam:PassRole on resource: arn:aws:iam::account_id:role/service-
role/AWSGlueServiceRole
because no identity-based policy allows the iam:PassRole action"
```

IAM 신뢰할 수 있는 엔티티

예약된 StartDataQualityEvaluationRun을 생성하고 실행하려면 AWS Glue 및 AWS EventBridge 스케줄러 서비스가 신뢰할 수 있는 엔티티에 나열되어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "glue.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "scheduler.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

예제 IAM 정책

AWS Glue Data Quality를 위한 IAM 역할에는 다음과 같은 유형의 권한이 필요합니다.

- AWS Glue Data Quality 작업에 대한 권한을 통해 권장 데이터 품질 규칙을 가져오고 AWS Glue 데이터 카탈로그의 테이블에 대해 데이터 품질 작업을 실행할 수 있습니다. 이 섹션의 예제 IAM 정책에는 AWS Glue Data Quality 작업에 필요한 최소 권한이 포함되어 있습니다.
- 데이터 카탈로그 테이블 및 기본 데이터에 대한 액세스 권한을 부여하는 권한입니다. 이러한 권한은 사용 사례에 따라 다릅니다. 예를 들어 Amazon S3에서 분류하는 데이터의 경우 권한에 Amazon S3에 대한 액세스가 포함되어야 합니다.

Note

이 섹션에 설명된 권한과 함께 Amazon S3 권한을 구성해야 합니다.

권장 데이터 품질 규칙에 필요한 최소 권한

이 예제 정책에는 권장 데이터 품질 규칙을 생성하는 데 필요한 권한이 포함되어 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGlueRuleRecommendationRunActions",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRuleRecommendationRun",
        "glue:PublishDataQuality",
        "glue:CreateDataQualityRuleset"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*"
    },
    {
      "Sid": "AllowCatalogPermissions",
      "Effect": "Allow",
      "Action": [
        "glue:GetPartitions",
        "glue:GetTable"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowS3GetObjectToRunRuleRecommendationTask",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::aws-glue-*"
    },
    { // Optional for Logs
```

```

    "Sid": "AllowPublishingCloudwatchLogs",
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogStream",
        "logs:CreateLogGroup",
        "logs:PutLogEvents"
    ],
    "Resource": "*"
  },
]
}

```

데이터 품질 작업을 실행하기 위한 최소한의 권한

이 예제 정책에는 데이터 품질 평가 작업을 실행하는 데 필요한 권한이 포함되어 있습니다.

다음 정책 설명은 사용 사례에 따라 선택 사항입니다.

- AllowCloudWatchPutMetricDataToPublishTaskMetrics - 데이터 품질 실행 지표를 Amazon CloudWatch에 게시하려는 경우 필요합니다.
- AllowS3PutObjectToWriteTaskResults - 데이터 품질 실행 결과를 Amazon S3에 기록하려는 경우 필요합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGlueGetDataQualityRuleset",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRuleset"
      ],
      "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/<YOUR-RULESET-NAME>"
    },
    {
      "Sid": "AllowGlueRulesetEvaluationRunActions",
      "Effect": "Allow",
      "Action": [
        "glue:GetDataQualityRulesetEvaluationRun",
        "glue:PublishDataQuality"
      ],
    }
  ]
}

```

```

    "Resource": "arn:aws:glue:us-east-1:111122223333:dataQualityRuleset/*"
  },
  {
    "Sid": "AllowCatalogPermissions",
    "Effect": "Allow",
    "Action": [
      "glue:GetPartitions",
      "glue:GetTable"
    ],
    "Resource": [
      "*"
    ]
  },
  {
    "Sid": "AllowS3GetObjectForRulesetEvaluationRun",
    "Effect": "Allow",
    "Action": [
      "s3:GetObject"
    ],
    "Resource": "arn:aws:s3::aws-glue-*"
  },
  {
    "Sid": "AllowCloudWatchPutMetricDataToPublishTaskMetrics",
    "Effect": "Allow",
    "Action": [
      "cloudwatch:PutMetricData"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "cloudwatch:namespace": "Glue Data Quality"
      }
    }
  },
  {
    "Sid": "AllowS3PutObjectToWriteTaskResults",
    "Effect": "Allow",
    "Action": [
      "s3:PutObject*"
    ],
    "Resource": "arn:aws:s3:::<YOUR-BUCKET-NAME>/*"
  }
]

```

}

Data Catalog에서 AWS Glue Data Quality 시작하기

이 시작하기 섹션에서는 AWS Glue 콘솔에서 AWS Glue Data Quality를 시작하는 데 도움이 되는 지침을 제공합니다. 데이터 품질 규칙 권장 사항 생성, 데이터에 대한 규칙 세트 평가와 같은 필수 작업을 완료하는 방법을 알아봅니다.

주제

- [필수 조건](#)
- [단계별 예제](#)
- [규칙 권장 사항 생성](#)
- [모니터링 규칙 권장 사항](#)
- [권장 규칙 세트 편집](#)
- [새 규칙 세트 생성](#)
- [규칙 세트를 실행하여 데이터 품질 평가](#)
- [데이터 품질 점수 및 결과 보기](#)
- [관련 주제](#)

필수 조건

AWS Glue Data Quality를 사용하기 전에 AWS Glue에서 Data Catalog 및 크롤러를 사용하는 방법을 숙지해야 합니다. AWS Glue Data Quality를 사용하여 Data Catalog 데이터베이스의 테이블 품질을 평가할 수 있습니다. 또한 다음 항목이 필요합니다.

- 데이터 품질 규칙 세트를 평가하기 위한 Data Catalog의 테이블.
- 규칙 권장 사항을 생성하거나 데이터 품질 작업을 실행할 때 제공하는 AWS Glue의 IAM 역할. 이 역할에는 다양한 AWS Glue Data Quality 프로세스를 대신 실행하는 데 필요한 리소스에 대한 액세스 권한이 있어야 합니다. 이러한 리소스에는 AWS Glue, Amazon S3, IAM, CloudWatch가 포함됩니다. AWS Glue Data Quality에 대한 최소 권한이 포함된 정책 예제를 보려면 [예제 IAM 정책](#) 섹션을 참조하세요.

AWS Glue의 IAM 역할에 대한 자세한 내용은 [AWS Glue 서비스를 위한 IAM 정책 생성 및 AWS Glue 서비스를 위한 IAM 역할 생성](#)을 참조하세요. [AWS Glue Data Quality 작업에 대한 권한](#)에서 데이터 품질과 관련된 모든 AWS Glue 권한 목록을 볼 수도 있습니다.

- 다양한 데이터를 포함하는 테이블이 하나 이상 있는 데이터베이스. 이 자습서에서 사용되는 테이블의 이름은 yyz-tickets이며, tickets 테이블도 있습니다. 이 데이터는 토론토시가 주차 위반 고지서에서 공개적으로 이용할 수 있는 정보의 컬렉션입니다. 테이블을 직접 생성하는 경우 가장 적합한 권장 규칙 세트를 가져올 수 있도록 테이블에 다양한 유효한 데이터를 입력해야 합니다.

단계별 예제

샘플 데이터 세트가 포함된 단계별 예제는 [AWS Glue Data Quality 블로그 게시물](#)을 참조하세요.

규칙 권장 사항 생성

규칙 권장을 사용하면 코드를 작성하지 않고도 데이터 품질을 쉽게 시작할 수 있습니다. AWS Glue Data Quality를 사용하면 데이터를 분석하고, 규칙을 식별하며, 데이터 품질 작업에서 평가할 수 있는 규칙 세트를 생성할 수 있습니다. 권장 사항 실행은 90일 후에 자동으로 삭제됩니다.

데이터 품질 규칙 권장 사항을 생성하려면

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 Tables(테이블)을 선택합니다. 그런 다음 데이터 품질 규칙 권장 사항을 생성할 테이블을 선택합니다.
3. 테이블 세부 정보 페이지에서 데이터 품질 탭을 선택하여 테이블의 AWS Glue Data Quality 규칙 및 설정에 액세스합니다.
4. 데이터 품질 탭에서 규칙 추가 및 데이터 품질 모니터링을 선택합니다.
5. 규칙 권장 사항이 실행되지 않는 경우 규칙 세트 빌더 페이지의 페이지 상단에 권장 작업을 시작하라는 경보가 표시됩니다.
6. 권장 규칙을 선택하여 모달을 열고 권장 작업의 파라미터를 입력합니다.
7. AWS Glue에 액세스할 수 있는 IAM 역할을 선택합니다. 이 역할에는 다양한 AWS Glue 데이터 품질 프로세스를 대신 실행하는 데 필요한 리소스에 대한 액세스 권한을 부여해야 합니다.
8. 기본 설정에 따라 필드를 작성한 후 권장 규칙을 선택하여 권장 작업 실행을 시작합니다. 권장 실행이 진행 중이거나 완료된 경우 이 알림에서 실행을 관리할 수 있습니다. 상태 변경을 보려면 알림을 새로 고쳐야 할 수 있습니다. 완료된 권장 작업 실행과 진행 중인 권장 작업 실행은 지난 90일 동안의 권장 사항 실행을 모두 나열하는 실행 기록 페이지에 표시됩니다.

권장 규칙의 의미

AWS Glue Data Quality는 입력 테이블의 각 열에 있는 데이터를 기반으로 규칙을 생성합니다. 그리고 규칙을 사용하여 품질 요구 사항을 유지하기 위해 데이터를 필터링할 수 있는 잠재적 경계를 식별합니다. 생성된 규칙의 다음 목록에는 규칙의 의미와 데이터에 적용할 때 규칙이 어떤 역할을 할 수 있는지 이해하는 데 유용한 예제가 포함되어 있습니다.

생성된 데이터 품질 정의 언어(DQDL) 규칙 유형의 전체 목록은 [DQDL 규칙 유형 참조](#)를 참조하세요.

- IsComplete "SET_FINE_AMOUNT" - IsComplete 규칙은 해당 행에 열이 채워졌는지 확인합니다. 이 규칙을 사용하여 데이터에서 열에 선택 사항이 아닌 항목으로 태그를 지정합니다.
- Uniqueness "TICKET_NUMBER" > 0.95 - Uniqueness 규칙은 열 내 데이터가 고유성 임계값을 충족하는지 확인합니다. 이 예제에서는 "TICKET_NUMBER"에 대해 해당 행을 채우는 데이터가 나머지 모든 행의 콘텐츠와 최대 95% 동일하다고 확인되었으며, 이에 따라 이 규칙을 제안합니다.
- ColumnValues "PROVINCE" in ["ON", "QC", "AB", "NY", ...] - ColumnValues 규칙은 기존 열 콘텐츠를 기반으로 열의 유효한 값을 정의합니다. 이 예제에서 각 행의 데이터는 state 또는 province에 대한 2자리 라이선스 코드입니다.
- ColumnLength "INFRACTION_DESCRIPTION" between 15 and 31 - 이 ColumnLength 규칙은 열 데이터에 길이 제한을 적용합니다. 이 규칙은 문자열 열에 기록된 최소 및 최대 길이를 기반으로 샘플 데이터에서 생성됩니다.

모니터링 규칙 권장 사항

데이터 품질 규칙 권장 작업이 실행 중이면 규칙 추가 및 데이터 품질 모니터링 페이지에 상단 표시줄에서 사용할 수 있는 정보와 추가 작업이 표시됩니다.

규칙 권장 작업이 진행 중인 경우 권장 작업이 완료되기 전에 실행 중지를 선택할 수 있습니다. 작업이 진행 중인 동안에는 진행 중 상태가 나타나고, 실행이 시작된 날짜 및 시간이 표시됩니다.

규칙 권장 작업이 완료되면 규칙 권장 사항 표시줄에 권장된 규칙 수, 마지막 권장 실행 상태, 완료 날짜 및 타임스탬프가 표시됩니다.

규칙 권장 사항 삽입을 선택하여 권장 규칙을 추가할 수 있습니다. 이전의 권장 규칙을 보려면 특정 날짜를 선택합니다. 새 권장 작업을 실행하려면 추가 작업을 선택하고 권장 규칙을 선택합니다.

사용자 설정 관리를 선택하여 기본 설정을 지정합니다. Amazon S3의 기본 경로를 설정하여 규칙 세트를 저장하거나 데이터 카탈로그를 실행하기 위한 기본 역할을 설정할 수 있습니다.

권장 규칙 세트 편집

AWS Glue Data Quality는 사용 가능한 기존 데이터를 기반으로 규칙을 생성하므로 자동 제안에서 예상치 못한 규칙 또는 바람직하지 않은 규칙이 표시될 수 있습니다. 권장 규칙 세트를 최대한 활용하려면 규칙 세트를 평가하고 수정해야 합니다. 자습서의 이 단계에서는 이전 단계에서 생성한 규칙을 가져와 이를 조정하여 일부 데이터에 더 제한적인 품질을 적용합니다. 또한 나중에 정확하고 고유한 데이터를 추가할 수 있도록 다른 규칙을 완화할 수도 있습니다.

제안된 규칙 세트 편집

1. AWS Glue 콘솔에서 데이터 카탈로그를 선택하고 탐색 창에서 데이터베이스 테이블을 선택합니다. tickets 테이블을 선택합니다.
2. 테이블 세부 정보 페이지에서 데이터 품질 탭을 선택하여 테이블의 AWS Glue Data Quality 규칙 및 설정에 액세스합니다.
3. 규칙 세트 섹션에서는 [규칙 권장 사항 생성](#)에서 생성된 규칙 세트를 선택합니다.
4. 작업을 선택한 다음 콘솔 창에서 편집을 선택합니다. 규칙 세트 편집기가 콘솔에 로드됩니다. 여기에는 규칙을 위한 편집 창과 DQDL에 대한 빠른 참조가 포함되어 있습니다.
5. 스크립트의 2번째 줄을 제거합니다. 이렇게 하면 데이터베이스 크기를 특정 행 수로 제한해야 하는 요구 사항이 완화됩니다. 편집 후에는 파일의 1~3번째 줄에 다음을 포함해야 합니다.

```
Rules = [
  IsComplete "TAG_NUMBER_MASKED",
  ColumnLength "TAG_NUMBER_MASKED" between 6 and 9,
```

6. 스크립트의 25번째 줄을 제거합니다. 이렇게 하면 기록된 province의 96%가 ON이어야 한다는 요구 사항이 완화됩니다. 편집 후에는 파일의 24번째 줄부터 규칙 세트 끝까지 다음을 포함해야 합니다.

```
ColumnValues "PROVINCE" in ["ON", "QC", "AB", "NY", "AZ", "NS", "BC", "MI", "PQ",
  "MB", "PA", "FL", "SK", "NJ", "OH", "NB", "IL", "MA", "CA",
  "VA", "TX", "NF", "MD", "PE", "CT", "NC", "GA", "IN", "OR", "MN", "TN", "WI",
  "KY", "MO", "WA", "NH", "SC", "CO", "OK", "VT", "RI", "ME", "AL",
  "YT", "IA", "DE", "AR", "LA", "XX", "WV", "MT", "KS", "NT", "DC", "NV", "NE",
  "UT", "MS", "NM", "ID", "SD", "ND", "AK", "NU", "GO", "WY", "HI"],
ColumnLength "PROVINCE" = 2
]
```

7. 14번째 줄을 다음과 같이 변경합니다.

```
IsComplete "TIME_OF_INFRACTION",
```

이렇게 하면 데이터베이스를 기록된 위반 시간이 포함된 티켓으로만 제한함으로써 열의 요구 사항이 강화됩니다. 기록된 위반 시간이 없는 티켓은 항상 이 데이터 세트에서 유효하지 않은 데이터로 간주해야 합니다. 추가 데이터를 사용하거나 검사하여 품질 규칙을 결정하려고 할 때 파티셔닝 또는 변환이 더 적합한 경우와는 상황이 다릅니다.

8. 콘솔 페이지 하단에서 규칙 세트 업데이트를 선택합니다.

새 규칙 세트 생성

규칙 세트는 데이터를 기준으로 평가하는 데이터 품질 규칙의 그룹입니다. AWS Glue 콘솔에서 데이터 품질 정의 언어(DQDL)를 사용하여 사용자 지정 규칙 세트를 작성할 수 있습니다.

데이터 품질 규칙 세트를 생성하려면

1. AWS Glue 콘솔에서 데이터 카탈로그를 선택하고 데이터베이스를 선택한 다음 탐색 창에서 테이블을 선택합니다. tickets 테이블을 선택합니다.
2. Data quality(데이터 품질) 탭을 엽니다.
3. 규칙 섹션에서 규칙 생성을 선택합니다. DQDL 편집기가 콘솔에서 실행됩니다. 여기에는 직접 편집할 수 있는 텍스트 영역과 DQDL 규칙 및 테이블 스키마에 대한 빠른 참조가 포함되어 있습니다.
4. DQDL 편집기의 텍스트 영역에 규칙을 추가하기 시작합니다. 이 자습서에서 직접 규칙을 작성하거나 데이터 품질 규칙 편집기의 DQDL 규칙 빌더 기능을 사용할 수 있습니다.

Note

DQDL 규칙 빌더를 사용하는 방법

1. 목록에서 규칙 유형을 선택하고 더하기 기호를 선택하여 편집기 창에 예제 구문을 삽입합니다.
2. 자리 표시자 열 이름을 자신의 열 이름으로 변경합니다. 테이블의 열 이름은 스키마 탭에서 사용할 수 있습니다.
3. 상황에 맞게 표현식 파라미터를 업데이트합니다. DQDL이 지원하는 전체 표현식 목록은 [Expressions](#) 섹션을 참조하세요.

예를 들어 다음 규칙은 tickets 테이블의 ticket_number 열에 대한 데이터 검증과 관련된 제약 조건입니다. 다음 규칙을 추가하려면 DQDL 규칙 빌더를 사용하거나 규칙 세트를 직접 편집합니다.

```
IsComplete "ticket_number",
IsUnique "ticket_number",
ColumnValues "ticket_number" > 9000000000
```

5. 규칙 세트 이름 필드에 새 규칙 세트의 이름을 제공합니다.
6. 규칙 세트 저장을 선택합니다.

여러 데이터 세트에서 데이터 품질 평가

ReferentialIntegrity 및 DatasetMatch 규칙 세트를 사용하여 여러 데이터 세트에서 데이터 품질 규칙을 설정할 수 있습니다. ReferentialIntegrity는 기본 데이터 세트의 데이터가 다른 데이터 세트에 있는지 검사합니다.

참조 데이터 세트를 추가하려면 스키마 탭을 선택한 다음 참조 테이블 업데이트를 선택합니다. 데이터베이스와 테이블을 선택하라는 메시지가 표시됩니다. 테이블을 추가한 다음 데이터 품질 규칙을 설정할 수 있습니다. AggregateMatch, RowCountMatch, ReferentialIntegrity, SchemaMatch 및 DatasetMatch와 같은 규칙 유형은 여러 데이터 세트에서 데이터 품질 검사를 수행하는 기능을 지원합니다.

규칙 세트를 실행하여 데이터 품질 평가

데이터 품질 작업을 실행할 때 AWS Glue 데이터 품질은 데이터를 기준으로 규칙 세트를 평가하고 데이터 품질 점수를 계산합니다. 이 점수는 입력에 대해 통과한 데이터 품질 규칙의 백분율을 나타냅니다.

데이터 품질 작업을 실행하려면

1. AWS Glue 콘솔에서 데이터 카탈로그를 선택하고 데이터베이스를 선택한 다음 탐색 창에서 테이블을 선택합니다. tickets 테이블을 선택합니다.
2. 데이터 품질 탭을 선택합니다.
3. 규칙 세트 목록에서 테이블을 기준으로 평가하려는 규칙 세트를 선택합니다. 이 단계에서는 생성된 규칙 대신 이미 작성했거나 수정한 규칙 세트를 사용하는 것이 좋습니다. Run(실행)을 선택합니다.

4. 모달에서 IAM 역할을 선택합니다. 이 역할에는 다양한 AWS Glue 데이터 품질 프로세스를 대신 실행하는 데 필요한 리소스에 대한 액세스 권한을 부여해야 합니다. IAM 역할을 기본값으로 저장하거나 기본 설정 페이지로 이동하여 수정할 수 있습니다.
5. Data quality actions(데이터 품질 작업)에서 Publish metrics to Amazon CloudWatch(Amazon CloudWatch에 지표 게시) 여부를 선택합니다. 이 옵션을 선택하면 AWS Glue Data Quality는 통과한 규칙 수와 실패한 규칙 수를 나타내는 지표를 게시합니다. 이러한 방식으로 저장된 지표에 대해 작업을 수행하기 위해 CloudWatch 경보를 사용할 수 있습니다. 알림을 설정할 수 있도록 주요 지표도 Amazon EventBridge에 게시됩니다. 자세한 내용은 [알림, 배포 및 예약 설정](#)을 참조하세요.
6. 실행 빈도에서 온디맨드로 실행을 선택하거나 규칙 세트를 예약합니다. 규칙 세트를 예약하면 작업 이름을 입력하라는 메시지가 표시됩니다. 예약은 Amazon EventBridge에서 생성됩니다. Amazon EventBridge에서 예약을 편집할 수 있습니다.
7. Amazon S3에 데이터 품질 결과를 저장하려면 데이터 품질 결과 위치를 선택합니다. 이 작업을 위해 이전에 선택한 IAM 역할에는 선택한 위치에 대한 쓰기 권한이 있어야 합니다.
8. 추가 구성 아래에 AWS Glue에서 데이터 품질 작업에 할당할 요청된 작업자 수를 입력합니다.
9. 선택적으로 데이터 소스에서 필터를 설정할 수 있습니다. 이렇게 하면 읽는 데이터가 줄어듭니다. 필터를 사용하여 파티션 정보를 선택하고 API 직접 호출을 통해 해당 정보를 파라미터로 전달하여 증분 검증을 실행할 수도 있습니다. 성능 개선을 위해 파티션 조건자를 제공할 수 있습니다.
10. 실행을 선택합니다. Data quality task runs(데이터 품질 작업 실행) 목록에 새 작업이 표시됩니다. 작업의 실행 상태 열이 완료됨으로 표시되면 품질 점수 결과를 볼 수 있습니다. 상태를 올바르게 업데이트하려면 콘솔 창을 새로 고쳐야 할 수도 있습니다.
11. 데이터 품질 결과 세부 정보 열을 보려면 '+' 아이콘을 선택하여 규칙 세트를 확장합니다. 결과에는 평가에서 통과한 규칙과 실패한 규칙, 그리고 규칙 실패의 원인이 표시됩니다.

데이터 품질 점수 및 결과 보기

생성된 모든 규칙 세트의 최신 실행을 보려면

1. AWS Glue 콘솔의 탐색 창에서 Tables(테이블)을 선택합니다. 그런 다음 데이터 품질 작업을 실행할 테이블을 선택합니다.
2. 데이터 품질 탭을 선택합니다.
3. 데이터 품질 스냅샷에서는 시간 경과에 따른 일반적인 실행 추세를 보여줍니다. 기본적으로 모든 규칙 세트에 대한 마지막 10개 실행이 표시됩니다. 규칙 세트별로 필터링하려면 드롭다운 목록에서 원하는 항목을 선택합니다. 실행이 10개 미만인 경우 사용 가능한 모든 완료된 실행이 표시됩니다.

4. 데이터 품질 테이블에는 최근 실행 기록이 있는 각 규칙 세트(있는 경우)가 점수와 함께 표시됩니다. 규칙 세트를 확장하면 해당 규칙 세트에 있는 규칙과 해당 실행의 규칙 결과가 함께 표시됩니다.

특정 규칙 세트의 최신 실행을 보려면

1. AWS Glue 콘솔의 탐색 창에서 Tables(테이블)을 선택합니다. 그런 다음 데이터 품질 작업을 실행할 테이블을 선택합니다.
2. 데이터 품질 탭을 선택합니다.
3. 데이터 품질 테이블에서 특정 규칙 세트를 선택합니다.
4. 규칙 세트 세부정보 페이지에서 실행 기록 탭을 선택합니다.

이 특정 규칙 세트에 대한 모든 평가 실행이 이 탭의 테이블에 나열됩니다. 점수 기록과 실행 상태를 볼 수 있습니다.

5. 특정 실행에 대한 자세한 내용을 보려면 실행 ID를 선택하여 평가 실행 세부 정보 페이지로 이동합니다. 이 페이지에서는 실행에 대한 세부 정보와 개별 규칙 결과의 상태에 대한 자세한 내용을 볼 수 있습니다.

관련 주제

- [DQDL 규칙 유형 참조](#)
- [데이터 품질 정의 언어\(DQDL\) 참조](#)

AWS Glue Studio에서 데이터 품질 평가

AWS Glue 데이터 품질은 정의된 규칙에 따라 데이터의 품질을 평가하고 모니터링합니다. 그러면 조치가 필요한 데이터를 쉽게 식별할 수 있습니다. AWS Glue Studio에서는 시각적 작업에 데이터 품질 노드를 추가하여 데이터 카탈로그의 테이블에서 데이터 품질 규칙을 생성할 수 있습니다. 그런 다음 시간 경과에 따라 진화하는 데이터 세트의 변경 사항을 모니터링하고 평가할 수 있습니다. AWS Glue Studio에서 AWS Glue Data Quality를 사용하는 방법에 대한 개요는 다음 비디오를 참조하세요.

다음은 AWS Glue 데이터 품질을 사용하는 방법에 대한 주요 단계입니다.

1. Create data quality rules(데이터 품질 규칙 생성) - 구성된 기본 제공 규칙 세트를 선택하여 DQDL 빌더를 통해 데이터 품질 규칙 세트를 구축합니다.

2. Configure a data quality job(데이터 품질 작업 구성) - 데이터 품질 결과 및 출력 옵션을 기반으로 작업을 정의합니다.
3. 데이터 품질 작업 저장 및 실행 - 작업을 생성하고 실행합니다. 작업을 저장하면 해당 작업에 대해 생성한 규칙 세트가 저장됩니다.
4. Monitor and review the data quality results(데이터 품질 결과 모니터링 및 검토) - 작업 실행이 완료된 후 데이터 품질 결과를 검토합니다. 원하는 경우 작업을 미래 날짜로 예약할 수 있습니다.

이점

데이터 분석가, 데이터 엔지니어 및 데이터 사이언티스트는 AWS Glue Studio의 데이터 품질 평가 노드를 사용하여 시각적 작업 편집기에서 데이터 품질을 분석, 구성, 모니터링 및 개선할 수 있습니다. 데이터 품질 노드를 사용하면 다음과 같은 이점이 있습니다.

- 데이터 품질 문제 감지 가능 - 데이터 세트의 특성을 확인하는 규칙을 생성하여 문제를 확인할 수 있습니다.
- 간편한 시작 - 미리 구축된 규칙 및 작업으로 시작할 수 있습니다.
- 긴밀한 통합 - AWS Glue Data Quality는 AWS Glue 데이터 카탈로그를 기반으로 실행되므로 AWS Glue Studio에서 데이터 품질 노드를 사용할 수 있습니다.

AWS Glue Studio에서 ETL 작업에 대한 데이터 품질 평가

이 자습서에서는 AWS Glue Studio에서 AWS Glue Data Quality를 시작합니다. 다음 작업을 수행하는 방법에 대해 알아보십시오.

- 데이터 품질 정의 언어(DQDL) 규칙 작성기를 사용하여 규칙을 생성합니다.
- 데이터 품질 조치, 출력할 데이터 및 데이터 품질 결과의 출력 위치를 지정하는 방법
- 데이터 품질 결과를 검토하는 방법

예제를 사용해 연습하려면 블로그 게시물, [ETL 파이프라인용 AWS Glue Data Quality 시작하기](#)를 검토하세요.

1단계: 시각적 작업에 데이터 품질 평가 변환 노드 추가

이 단계에서는 시각적 작업 편집기에 데이터 품질 평가 노드를 추가합니다.

데이터 품질 노드를 추가하려면

1. AWS Glue Studio 콘솔의 작업 생성 섹션에서 소스 및 대상이 있는 시각적 객체를 선택하고 생성을 선택합니다.
2. 데이터 품질 변환을 적용할 노드를 선택합니다. 일반적으로 변환 노드 또는 데이터 소스입니다.
3. '+' 아이콘을 선택하여 왼쪽의 리소스 패널을 엽니다. 검색 표시줄에서 데이터 품질 평가를 검색하고 검색 결과에서 데이터 품질 평가를 선택합니다.
4. 시각적 작업 편집기에는 선택한 노드에서 분기되는 데이터 품질 평가 변환 노드가 표시됩니다. 콘솔 오른쪽에 Transform(변환) 탭이 자동으로 열립니다. 상위 노드를 변경해야 하는 경우 노드 속성 탭을 선택한 다음 드롭다운 메뉴에서 노드 상위 항목을 선택합니다.

새 노드 상위 항목을 선택하면 상위 노드와 Evaluate Data Quality(데이터 품질 평가) 노드 사이에 새 연결이 설정됩니다. 원치 않는 상위 노드를 제거합니다. 각 Evaluate Data Quality(데이터 품질 평가) 노드에는 상위 노드를 하나만 연결할 수 있습니다.

5. 데이터 품질 평가 변환은 여러 상위 항목을 지원하므로 여러 데이터 세트에서 데이터 품질 규칙을 검증할 수 있습니다. 여러 데이터 세트를 지원하는 규칙으로는 ReferentialIntegrity, DatasetMatch, SchemaMatch, RowCountMatch, AggregateMatch가 있습니다.

데이터 품질 평가 변환에 여러 입력을 추가할 때는 '기본' 입력을 선택해야 합니다. 기본 입력은 데이터 품질을 검증하려는 데이터 세트입니다. 다른 모든 노드 또는 입력은 참조로 처리됩니다.

데이터 품질 평가 변환을 사용하여 데이터 품질 검사에 실패한 특정 레코드를 식별할 수 있습니다. 잘못된 레코드에 플래그를 지정하는 새 열이 기본 데이터 세트에 추가되므로 기본 데이터 세트를 선택하는 것이 좋습니다.

6. 입력 데이터 소스의 별칭을 지정할 수 있습니다. 별칭은 ReferentialIntegrity 규칙을 사용할 때 입력 소스를 참조하는 또 다른 방법을 제공합니다. 하나의 데이터 소스만 기본 소스로 지정할 수 있으므로 사용자가 추가하는 각 추가 데이터 소스에는 별칭이 필요합니다.

다음 예제에서 ReferentialIntegrity 규칙은 별칭 이름으로 입력 데이터 소스를 지정하고 기본 데이터 소스에 대한 일대일 비교를 수행합니다.

```
Rules = [
  ReferentialIntegrity "Aliasname.name" = 1
]
```


2단계: DQDL을 사용하여 규칙 생성

이 단계에서는 DQDL을 사용하여 규칙을 생성합니다. 이 자습서에서는 완전성 규칙 유형을 사용하여 단일 규칙을 생성합니다. 이 규칙 유형은 지정된 표현식을 기준으로 열에서 전체(null이 아닌) 값의 백분율을 검사합니다. DQDL 사용에 대한 자세한 내용은 [DQDL](#)을 참조하세요.

1. 변환 탭에서 삽입 버튼을 선택하여 규칙 유형을 추가합니다. 그러면 규칙 편집기에 규칙 유형이 추가되어 규칙에 대한 파라미터를 입력할 수 있습니다.

Note

규칙을 편집할 경우 규칙이 대괄호 안에 있고 쉼표로 구분되었는지 확인합니다. 예를 들어 전체 규칙 표현식은 다음과 비슷합니다.

```
Rules= [
    Completeness "year">0.8, Completeness "month">0.8
]
```

이 예제에서는 이름이 'year' 및 'month'인 열의 완전성 파라미터를 지정합니다. 규칙이 통과하려면 이러한 열의 'complete' 비율이 80%를 초과하거나 각 열에 대해 80%가 넘는 인스턴스에 데이터가 있어야 합니다.

이 예에서는 Completeness(완전성) 규칙 유형을 검색하여 삽입합니다. 그러면 규칙 유형이 규칙 편집기에 추가됩니다. 이 규칙 유형의 구문은 다음과 같습니다. Completeness <COL_NAME> <EXPRESSION>.

대부분의 규칙 유형에서는 부울 응답을 생성하려면 표현식을 파라미터로 제공해야 합니다. 지원되는 DQDL 표현식에 대한 자세한 내용은 [DQDL 표현식](#)을 참조하세요. 이제 열 이름을 추가합니다.

2. DQDL 규칙 작성기에서 스키마 탭을 선택합니다. 검색 표시줄을 사용하여 입력 스키마에서 열 이름을 찾습니다. 입력 스키마는 열 이름과 데이터 유형을 표시합니다.
3. 규칙 편집기에서 규칙 유형의 오른쪽을 클릭하여 열을 삽입할 위치에 커서를 삽입합니다. 또는 규칙에 열 이름을 입력할 수 있습니다.

예를 들어 입력 스키마 목록의 열 목록에서 열(이 예제의 경우 year) 옆에 있는 삽입 버튼을 클릭합니다. 그러면 해당 열이 규칙에 추가됩니다.

4. 그런 다음 규칙 편집기에서 규칙을 평가하는 표현식을 추가합니다. 완전성 규칙 유형은 지정된 표현식을 기준으로 열에서 전체(null이 아닌) 값의 백분율을 검사하므로 > 0.8과 같은 표현식을 입력합니다. 이 규칙은 전체(null이 아닌) 값이 80%를 초과하는 경우에 열을 검사합니다.

3단계: 데이터 품질 출력 구성

데이터 품질 규칙을 생성한 후 추가 옵션을 선택하여 데이터 품질 노드 출력을 지정할 수 있습니다.

1. 데이터 품질 변환 출력에서 다음 옵션 중 하나를 선택합니다.

- 원래 데이터 - 원래 입력 데이터를 출력하려면 선택합니다. 이 옵션을 선택하면 새 하위 노드, 'rowLevelOutcomes'가 작업에 추가됩니다. 스키마는 변환에 대한 입력으로 전달된 기본 데이터 세트의 스키마와 일치합니다. 이 옵션은 데이터를 전달할 때 품질 문제가 있으면 작업을 실패시키려는 경우에 유용합니다.

또 다른 사용 사례로, 데이터 품질 검사에 실패한 잘못된 레코드를 감지하려는 경우가 있습니다. 잘못된 레코드를 감지하려면 데이터 품질 오류를 표시하도록 새 열 추가 옵션을 선택합니다. 이 작업을 수행하면 'rowLevelOutcomes' 변환의 스키마에 4개의 새 열이 추가됩니다.

- DataQualityRulesPass(문자열 배열) - 데이터 품질 검사에 통과한 규칙 배열을 제공합니다.
- DataQualityRulesFail(문자열 배열) - 데이터 품질 검사에 실패한 규칙 배열을 제공합니다.
- DataQualityRulesSkip(문자열 배열) - 건너뛴 규칙 배열을 제공합니다. 다음 규칙은 데이터 세트 수준에서 적용되므로 오류 레코드를 식별할 수 없습니다.
 - AggregateMatch
 - ColumnCount
 - ColumnExists
 - ColumnNamesMatchPattern
 - CustomSql
 - RowCount
 - RowCountMatch
 - StandardDeviation
 - 평균
 - ColumnCorrelation
- DataQualityEvaluationResult - 행 수준에서 '통과' 또는 '실패' 상태를 제공합니다. 전체 결과는 실패일 수 있지만 특정 레코드는 통과할 수 있습니다. 예를 들어 RowCount 규칙은 실패해도 다른 모든 규칙이 성공했을 수 있습니다. 이 경우 이 필드 상태는 '통과'입니다.

2. 데이터 품질 결과 - 구성된 규칙과 규칙의 통과 또는 실패 상태를 출력하려면 선택합니다. 이 옵션은 결과를 Amazon S3 또는 다른 데이터베이스에 기록하려는 경우 유용합니다.
3. 데이터 품질 출력 설정(선택 사항) - 데이터 품질 출력 설정을 클릭하여 데이터 품질 결과 위치 필드를 표시합니다. 그런 다음 찾아보기를 클릭하여 데이터 품질 출력 대상으로 설정할 Amazon S3 위치를 검색합니다.

4단계. 데이터 품질 작업 구성

작업을 사용하여 지표를 CloudWatch에 게시하거나 특정 기준에 따라 작업을 중지할 수 있습니다. 작업은 규칙을 생성한 경우에만 사용할 수 있습니다. 이 옵션을 선택하면 동일한 지표도 Amazon EventBridge에 게시됩니다. 이러한 옵션을 사용하여 [알림용 경보를 생성](#)할 수 있습니다.

- 규칙 세트 실패 시 - 작업 실행 중에 규칙 세트에서 실패한 경우 수행할 조치를 선택할 수 있습니다. 데이터 품질에서 실패한 경우 작업이 실패하게 하려면 다음 옵션 중 하나를 선택하여 작업이 실패해야 하는 시점을 선택합니다. 기본적으로 이 작업은 선택되지 않으며 데이터 품질 규칙에서 실패하더라도 작업 실행은 완료됩니다.
 - 없음 - 없음(기본값)을 선택하면 규칙 세트에서 실패했어도 작업은 실패하지 않고 계속 실행됩니다.
 - 대상에 데이터를 로드한 후 작업 실패 - 작업이 실패하고 데이터는 저장되지 않습니다. 결과를 저장하려면 데이터 품질 결과를 저장할 Amazon S3 위치를 선택합니다.
 - 대상 데이터를 로드하지 않고 작업 실패 - 이 옵션은 데이터 품질 오류가 발생하는 경우 즉시 작업이 실패하게 합니다. 데이터 품질 변환의 결과를 포함하여 어떠한 데이터 대상도 로드되지 않습니다.

5단계: 데이터 품질 결과 보기

작업을 실행한 후 데이터 품질 탭을 클릭하여 데이터 품질 결과를 확인합니다.

1. 각 작업 실행에 대해 데이터 품질 결과를 확인합니다. 각 노드에는 데이터 품질 상태와 상태 세부 정보가 표시됩니다. 노드를 선택하면 모든 규칙과 각 규칙의 상태가 표시됩니다.
2. 결과 다운로드를 선택하여 작업 실행 및 데이터 품질 결과에 대한 정보가 포함된 CSV 파일을 다운로드합니다.
3. 데이터 품질 결과를 포함하는 작업 실행이 두 개 이상인 경우 날짜 및 시간 범위별로 결과를 필터링할 수 있습니다. 날짜 및 시간 범위별 필터링을 선택하여 필터 기간을 확장합니다.

4. Relative range(상대 범위) 또는 Absolute range(절대 범위)를 선택합니다. 절대 범위의 경우 달력을 사용하여 날짜를 선택하고 시작 시간 및 종료 시간 값을 입력합니다. 완료했으면 적용을 선택합니다.

데이터 품질 규칙 작성기

데이터 품질 정의 언어(DQDL) 규칙 작성기를 사용하면 데이터 품질 규칙을 생성하여 데이터를 평가할 수 있습니다. 먼저 규칙 유형을 선택하고 규칙 편집기에서 파라미터를 지정합니다. 규칙 편집기에는 규칙을 생성할 때 발생하는 오류 및 경고도 표시됩니다.

[DQDL 안내서](#)에서는 DQDL 구문, 기본 제공 규칙 유형 및 예제를 사용하여 규칙을 구성하는 방법에 대한 포괄적인 문서를 제공합니다.

데이터 품질 평가 노드

데이터 품질 평가 변환 노드 및 DQDL 규칙 작성기를 사용하여 작업 공간을 확장할 수 있습니다.

- 변환 탭을 확장하여 전체 화면을 채우려면 노드 세부 정보 패널의 오른쪽 상단에 있는 확장 아이콘을 선택합니다.
- DQDL 규칙 편집기를 확장하려면 << 아이콘을 선택하여 규칙 편집기를 확장하고 규칙 유형 및 스키마 탭을 축소합니다.

The screenshot displays the AWS Glue console interface. On the left, a workflow diagram shows a central 'Transform - Evaluate Data Quality...' node receiving input from two 'Data source - Data Catalog' nodes: 'employees' and 'customers'. This central node outputs to two 'Transform - SelectFrom...' nodes: 'rowLevelOutcomes' and 'ruleOutcomes'. 'rowLevelOutcomes' is connected to a 'Data target - S3 bucket Amazon S3', and 'ruleOutcomes' is connected to a 'Data target - Data Catalog AWS Glue Data Catalog'.

On the right, the configuration panel for the 'Evaluate Data Quality (Multiframe)' node is shown. The 'Name' field contains 'Evaluate Data Quality (Multiframe)'. Under 'Node parents', 'employees' and 'customers' are selected. The 'Aliases for referenced data sources' section shows 'employees' as the primary source and 'customers' as a secondary source. The 'Rules' section contains a DQDL rule:

```

Rules = [
  1 ReferentialIntegrity "employeenumber" "customers
  2 salesRepEmployeeNumber" between 0.6 and 0.7,
  3 RowCount > 1000,
  4 CustomSql "select count(*) from primary" between 10 and 200
  5 ]
    
```

Below the rules, there are tabs for 'Rule types' and 'Schema'. The 'Rule types' list includes: AggregateMatch, ColumnCorrelation, ColumnCount, ColumnDataType, and ColumnExists. At the bottom, the 'Data quality transform output info' section has 'Original data' selected as the output option.

구성 요소

26개의 규칙 유형이 AWS Glue Studio에 기본 제공됩니다. 각 규칙 유형에는 사용 방법에 대한 설명과 예제가 있습니다.

데이터 품질 규칙 유형

AWS Glue Studio에서는 규칙을 쉽게 만들 수 있도록 기본 제공 규칙 유형을 제공합니다. 규칙 유형에 대한 자세한 내용은 [DQDL 규칙 유형 참조](#)를 참조하세요.

스키마

Schema(스키마) 탭에는 상위 노드의 열 이름과 데이터 유형이 표시됩니다. 여러 노드의 스키마가 표시됩니다. 입력 스키마를 보고, 열 이름을 기준으로 검색하고, 해당 열을 규칙 편집기에 삽입할 수 있습니다.

Node properties | **Transform** | Output schema | Data preview

Evaluate data quality [Info](#)
Evaluate data quality by defining your data quality rules and actions

Data quality rules [Info](#)
Add rules using DQDL (Data Quality Definition Language)

DQDL rule builder <<

Rule types (18) **Schema**

Search

▼ **Input schema**

- year int
- month int
- day int
- fl_date string

```

1 Rules= [
2   Completeness"year">0.8
3 ]

```

Ln 1, Col 1 Errors: 0 Warnings: 0

규칙 편집기

규칙 편집기는 규칙을 작성하고 편집할 수 있는 텍스트 편집기입니다. DQDL 규칙 작성기에서 규칙 유형을 선택하면 규칙 유형이 규칙 편집기에 추가됩니다. 그런 다음 텍스트를 수정하여 필요에 따라 파라미터를 지정하고 규칙을 추가하며 규칙을 편집할 수 있습니다. AWS Glue Studio에서는 규칙 편집기에서 규칙을 검증하고 오류 및 경고가 있을 경우 이를 표시합니다.

Errors and warnings(오류 및 경고)

규칙이 DQDL 규칙 구문을 따르지 않는 경우 규칙 편집기에 오류가 있음을 나타내는 몇 가지 시각적 표시기가 나타납니다.

- 규칙 편집기는 오류가 있는 행을 오류 아이콘과 함께 빨간색으로 표시합니다.
- 규칙 편집기의 빨간색 오류 아이콘 옆에 오류 수가 표시됩니다.
- 오류가 있는 행을 선택하면 오류 설명과 위치(행 및 열)가 규칙 편집기 아래쪽에 표시됩니다.

Node properties | **Transform** | Output schema | Data preview

Evaluate data quality [Info](#)
Evaluate data quality by defining your data quality rules and actions

Data quality rules [Info](#)
Add rules using DQDL (Data Quality Definition Language)

DQDL rule builder <<

Rule types (18) | Schema

Search

ColumnCorrelation column rule +
▶ Description, examples

ColumnExists column rule +
▶ Description, examples

ColumnLength column rule +
▶ Description, examples

Ln 1, Col 18 **x 1** ⚠ 0

Ln 1, Col 1 h is null

데이터 품질 작업

기본적으로 이 작업은 선택되지 않으며 데이터 품질 규칙이 실패하더라도 작업 실행이 완료됩니다.

다음 작업 중에서 선택합니다. 작업을 사용하여 결과를 CloudWatch에 게시하거나 특정 기준에 따라 작업을 중지할 수 있습니다. 작업은 규칙을 생성한 경우에만 사용할 수 있습니다.

- CloudWatch에 결과 게시 - 작업을 실행할 때 결과를 CloudWatch에 추가합니다.
- 데이터 품질이 실패하면 작업 실패 - 데이터 품질 규칙이 실패하면 결과적으로 작업도 실패합니다.

데이터 품질 변환 출력

- 원래 데이터 - 원래 입력 데이터를 출력하려면 선택합니다. 이 옵션은 품질 문제가 감지되었을 때 작업을 중지하려는 경우에 적합합니다.
- 데이터 품질 지표 - 구성된 규칙과 규칙의 통과 또는 실패 상태를 출력하려면 선택합니다. 이 옵션은 사용자 지정 작업을 수행하려는 경우에 유용합니다.

데이터 품질 출력 설정

Amazon S3 위치를 데이터 품질 출력 대상으로 지정하여 데이터 품질 결과 위치를 설정합니다.

이상 탐지 구성 및 인사이트 생성

AWS Glue Data Quality(DQ)는 사용자가 작성한 데이터 품질 규칙에 따라 데이터를 평가하고 시간 경과에 따른 데이터에 대한 인사이트와 관찰 결과를 제공하여 즉각적인 조치를 취할 수 있도록 합니다. DQ는 데이터를 스캔한 다음 행 수, 최댓값 또는 최솟값과 같은 통계 지표를 계산한 다음 임계값 표현식과 비교합니다.

Data Quality 이상 탐지의 몇 가지 이점은 다음과 같습니다.

- 데이터의 지속적인 자동 스캔
- 의도하지 않은 이벤트 또는 통계적 이상을 나타낼 수 있는 이상 탐지
- Data Quality 이상 탐지로 발견된 관찰에 대한 조치를 취하기 위한 규칙 권장 사항 제공

이는 다음과 같은 경우에 유용합니다.

- 데이터 품질을 기록할 필요 없이 데이터의 이상을 자동으로 탐지하려는 경우
- 데이터를 프로파일링하고 데이터의 시각적 표현을 확인하려는 경우
- 시간이 지남에 따라 데이터가 어떻게 변화하는지 추적하고 싶은 경우

내 데이터에 대해 어떤 관찰 결과를 볼 수 있나요?

DQ는 수집된 데이터 통계에서 이상치, 데이터 형식의 변경, 데이터 드리프트, 스키마 변경을 식별합니다. DQ는 관찰 결과를 바탕으로 사용자가 쉽게 운영할 수 있는 데이터 품질 규칙을 추천합니다. 통계

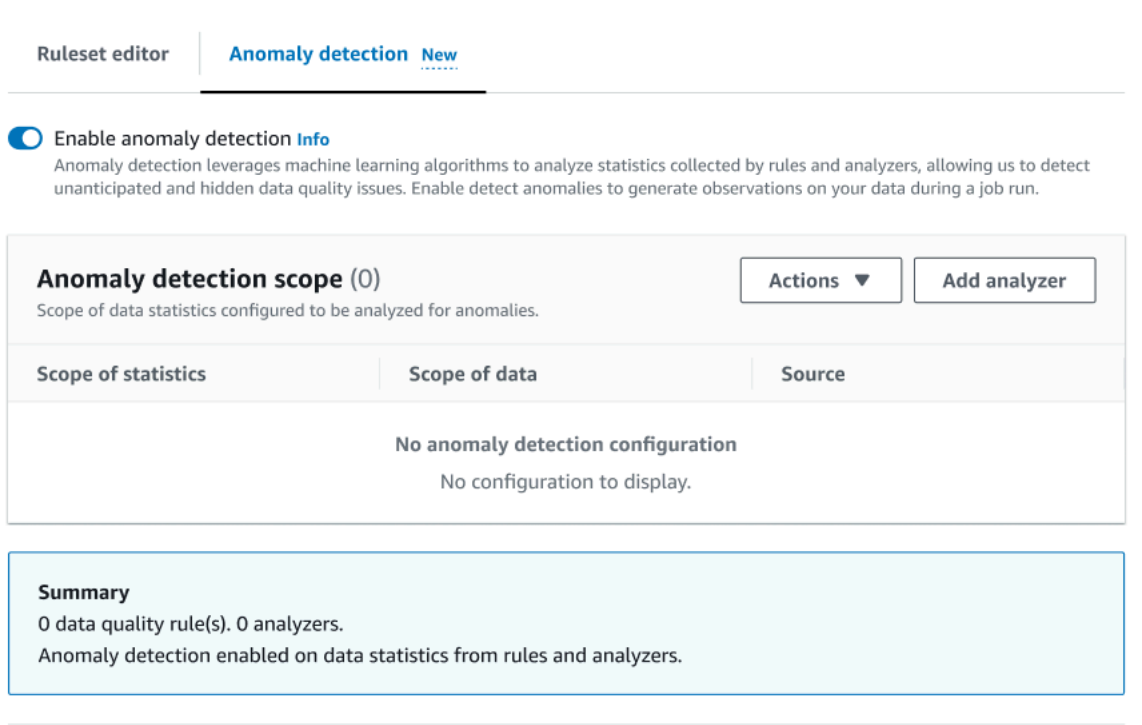
에는 완전성, 고유성, 평균, 합계 StandardDeviation, 엔트로피 등이 포함됩니다. DistinctValuesCount UniqueValueRatio

AWS Glue Studio에서 이상 탐지 활성화

이상 탐지를 활성화하려면 AWS Glue Studio 작업을 열고 '이상 탐지 활성화'를 켜면 됩니다. 이 기능을 켜면 시간 경과에 따른 데이터를 분석하고 데이터에 대한 데이터 통계와 조치를 취할 수 있는 관찰 결과를 제공하여 데이터에 대한 변칙 검색을 할 수 있습니다.

AWS Glue Studio에서 이상 탐지를 활성화하려면 다음을 수행합니다.

1. 작업에서 Data Quality 노드를 선택한 다음 이상 탐지 탭을 선택합니다. '이상 탐지 활성화'를 설정합니다.



Ruleset editor | **Anomaly detection** New

Enable anomaly detection [Info](#)
 Anomaly detection leverages machine learning algorithms to analyze statistics collected by rules and analyzers, allowing us to detect unanticipated and hidden data quality issues. Enable detect anomalies to generate observations on your data during a job run.

Anomaly detection scope (0) Actions ▼ Add analyzer
 Scope of data statistics configured to be analyzed for anomalies.

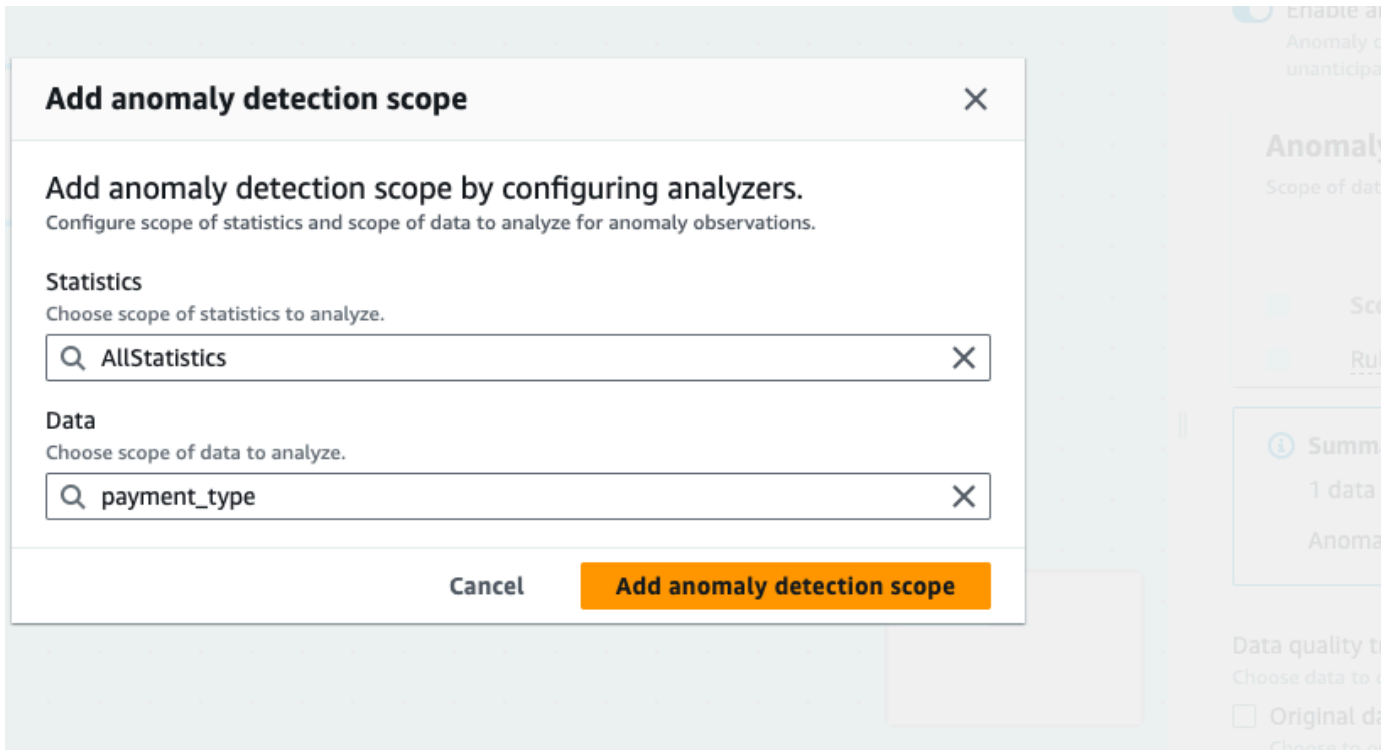
Scope of statistics	Scope of data	Source
No anomaly detection configuration No configuration to display.		

Summary
 0 data quality rule(s). 0 analyzers.
 Anomaly detection enabled on data statistics from rules and analyzers.

2. 분석기 추가를 선택하여 이상을 모니터링할 데이터를 정의합니다. 입력할 수 있는 두 가지 필드는 통계와 데이터입니다.

통계는 데이터의 형태 및 기타 속성에 대한 정보입니다. 한 번에 하나 이상의 통계를 선택하거나 모든 통계를 선택할 수 있습니다. 통계에는 완전성, 고유성, 평균, 합계, 엔트로피 등이 포함됩니다. StandardDeviation DistinctValuesCount UniqueValueRatio

데이터는 데이터 세트의 열입니다. 모든 열 또는 개별 열을 선택할 수 있습니다.



3. 이상 탐지 범위 추가를 선택하여 변경 사항을 저장합니다. 분석기를 생성하면 이상 탐지 범위 섹션에서 해당 분석기를 확인할 수 있습니다.

작업 메뉴를 사용하여 분석기를 편집하거나 규칙 세트 편집기 탭을 선택하고 규칙 세트 편집기 메모장에서 직접 분석기를 편집할 수도 있습니다. 생성한 규칙 바로 아래에 저장한 분석기가 표시됩니다.

```
Rules = [
]

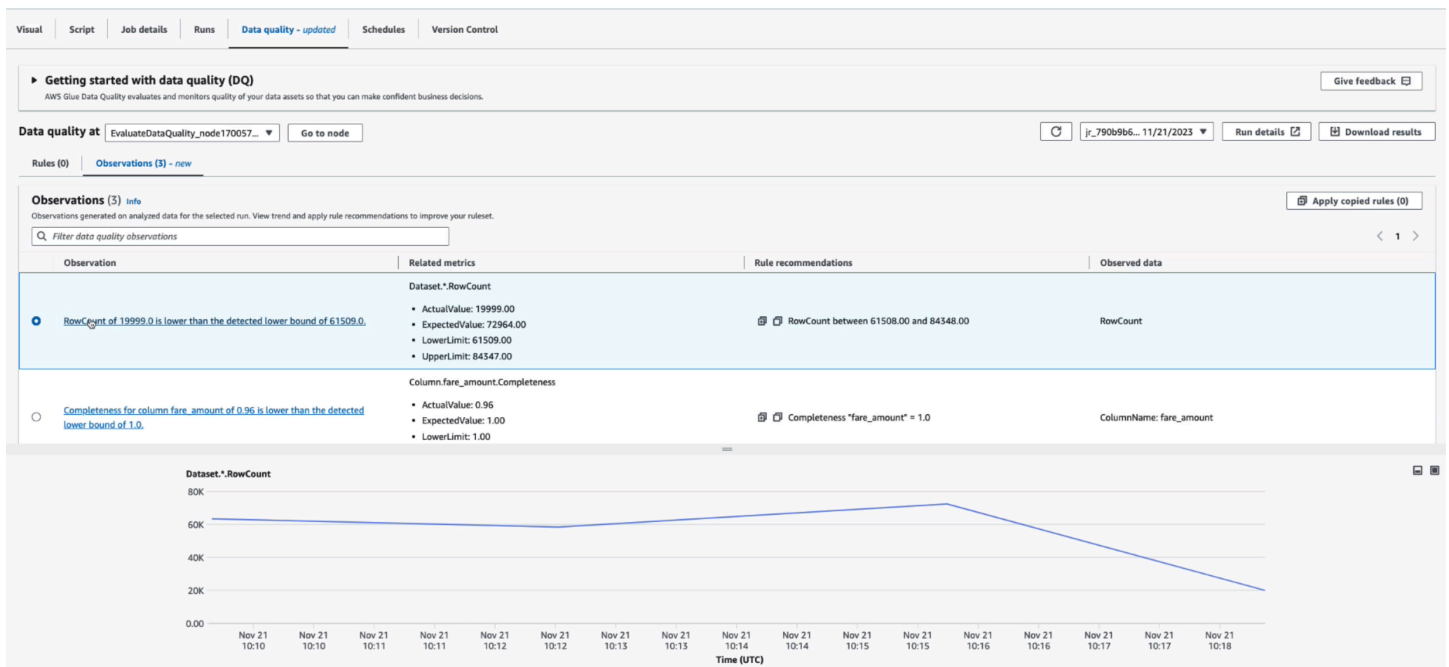
Analyzers = [
  Completeness "id"
]
```

Data Quality는 분석기와 함께 업데이트된 규칙 세트를 통해 들어오는 데이터를 지속적으로 모니터링하고, 설정에 따라 경고 또는 작업 중지를 통해 이상을 알립니다.

Note

관찰 결과는 데이터 세트에서 데이터 통계당 최소 3개 이상의 값이 관찰될 때 생성됩니다. 관찰 결과가 보이지 않으면 Data Quality에서 관찰 결과를 생성할 데이터가 충분하지 않은 것입니다. 여러 번의 작업 실행 후 Data Quality는 데이터에 대한 인사이트를 제공할 수 있으며 관찰 결과 섹션에 표시됩니다.

분석기는 데이터에서 이상을 탐지하여 관찰을 생성하고 점진적으로 규칙을 구축할 수 있는 권장 사항을 제공합니다. Data Quality 탭을 선택하여 관찰 결과를 볼 수 있습니다. 관찰 결과는 각 작업 실행에 따라 다릅니다. 관찰 결과 섹션 상단에서 특정 Data Quality 노드와 실행 중인 작업을 볼 수 있습니다. 새 노드 또는 작업 실행을 선택하면 해당 노드 및 작업과 관련된 관찰 결과를 볼 수 있습니다.



관찰 - 각 인사이트는 사용자가 지정한 규칙 세트와 분석기로 구성된 특정 작업 실행을 기반으로 합니다.

관련 지표 - 관찰 결과가 생성되면 관련 지표 옆에 규칙, 실제 및 예상 값, 하한값과 상한값이 표시됩니다.

규칙 권장 사항 - AWS Glue 그런 다음 이 문제를 해결하기 위한 규칙도 권장합니다. 권장되는 각 규칙은 복사 아이콘을 클릭하여 복사할 수 있습니다. 각 규칙 옆의 복사 아이콘을 클릭한 다음 복사된 규칙 적용을 클릭하여 모든 권장 규칙을 복사할 수 있습니다.

모니터링된 데이터 - 모니터링된 데이터 열에는 모니터링되고 관찰을 트리거한 열 또는 행이 표시됩니다.

Data Quality 노드에 권장 규칙 적용

관찰 결과가 생성되고 권장 규칙이 제공되면 해당 규칙을 Data Quality 노드에 적용할 수 있습니다. 방법:

1. 각 규칙 권장 사항 옆에 있는 복사 아이콘을 클릭합니다. 이렇게 하면 나중에 검색할 수 있도록 메모장에 규칙 권장 사항이 추가됩니다.
2. 규칙 권장 사항 적용을 클릭합니다. 그러면 이전에 복사한 규칙을 볼 수 있는 메모장이 열립니다.
3. 규칙 복사를 선택합니다.
4. 규칙 세트 편집기에 적용을 선택합니다. 그러면 복사한 규칙을 붙여넣을 수 있는 규칙 세트 편집기가 열립니다.
5. 복사한 규칙을 규칙 세트 편집기에 붙여넣습니다.

AWS Glue Studio 노트북의 ETL 작업에 대한 데이터 품질

이 자습서에서는 AWS Glue Studio 노트북에서 추출, 변환, 적재(ETL) 작업에 대해 AWS Glue Data Quality를 사용하는 방법을 알아봅니다.

AWS Glue Studio에서 노트북을 사용하여 전체 작업을 실행하지 않고도 작업 스크립트를 편집하고 출력을 볼 수 있습니다. 마크다운을 추가하고 노트북을 .ipynb 파일 및 작업 스크립트로 저장할 수도 있습니다. 소프트웨어를 로컬로 설치하거나 서버를 관리하지 않고도 노트북을 시작할 수 있습니다. 코드가 만족스러우면 AWS Glue Studio를 사용하여 노트북을 AWS Glue 작업으로 쉽게 전환할 수 있습니다.

이 예제에서 사용된 데이터 세트는 두 Data.CMS.gov 데이터 세트('Inpatient Prospective Payment System Provider Summary for the Top 100 Diagnosis-Related Groups - FY2011' 및 'Inpatient Charge Data FY 2011')에서 다운로드한 Medicare Provider 지불 데이터로 구성됩니다.

데이터 다운로드 후 데이터 집합을 수정하여 파일 끝에 몇 가지 잘못된 기록을 소개합니다. 이 수정된 파일은 `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv`의 퍼블릭 Amazon S3 버킷에 있습니다.

사전 조건

- 대상 Amazon S3 버킷에 대한 Amazon S3 쓰기 권한이 있는 AWS Glue 역할

- 새 노트북([AWS Glue Studio에서 노트북 시작하기 참조](#))

AWS Glue Studio에서 ETL 작업 생성

ETL 작업을 생성하려면

1. 세션 버전을 AWS Glue 3.0으로 변경합니다.

이렇게 하려면 다음과 같은 매직으로 모든 표준 문안 코드 셀을 제거하고 셀을 실행합니다. 이 표준 문안 코드는 새 노트북이 생성되면 첫 번째 셀에 자동으로 제공됩니다.

```
%glue_version 3.0
```

2. 다음 코드를 복사하여 셀에서 실행합니다.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
```

3. 다음 셀에서는 AWS Glue Data Quality를 평가하는 EvaluateDataQuality 클래스를 가져옵니다.

```
from awsgluedq.transforms import EvaluateDataQuality
```

4. 다음 셀에서는 퍼블릭 Amazon S3 버킷에 저장된.csv 파일을 사용하여 소스 데이터를 읽습니다.

```
medicare = spark.read.format(
    "csv").option(
    "header", "true").option(
    "inferSchema", "true").load(
    's3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv')
medicare.printSchema()
```

5. 데이터를 로 AWS Glue DynamicFrame 변환합니다.

```
from awsglue.dynamicframe import DynamicFrame
medicare_dyf = DynamicFrame.fromDF(medicare,glueContext,"medicare_dyf")
```

6. 데이터 품질 정의 언어(DQDL)를 사용하여 규칙 세트를 생성합니다.

```
EvaluateDataQuality_ruleset = """
    Rules = [
        ColumnExists "Provider Id",
        IsComplete "Provider Id",
        ColumnValues " Total Discharges " > 15
    ]
    """
```

7. 규칙 세트를 기준으로 데이터 세트를 검증합니다.

```
EvaluateDataQualityMultiframe = EvaluateDataQuality().process_rows(
    frame=medicare_dyf,
    ruleset=EvaluateDataQuality_ruleset,
    publishing_options={
        "dataQualityEvaluationContext": "EvaluateDataQualityMultiframe",
        "enableDataQualityCloudWatchMetrics": False,
        "enableDataQualityResultsPublishing": False,
    },
    additional_options={"performanceTuning.caching": "CACHE_NOTHING"},
)
```

8. 결과를 검토합니다.

```
ruleOutcomes = SelectFromCollection.apply(
    dfc=EvaluateDataQualityMultiframe,
    key="ruleOutcomes",
    transformation_ctx="ruleOutcomes",
)
```

```
ruleOutcomes.toDF().show(truncate=False)
```

출력:

```
-----+-----
+-----+
+-----+
|Rule                                |Outcome|FailureReason
          |EvaluatedMetrics          |
+-----+-----+
+-----+
+-----+
|ColumnExists "Provider Id"         |Passed |null
          |{}                               |
|IsComplete "Provider Id"           |Passed |null
          |{Column.Provider Id.Completeness -> 1.0} |
|ColumnValues " Total Discharges " > 15|Failed |Value: 11.0 does not meet the
  constraint requirement!|{Column. Total Discharges .Minimum -> 11.0}|
+-----+-----+
+-----+
+-----+
+-----+
```

9. Data Quality 행 수준 결과에서 통과된 행을 필터링하고 실패한 행을 검토합니다.

```
rowLevelOutcomes = SelectFromCollection.apply(
dfc=EvaluateDataQualityMultiframe,
key="rowLevelOutcomes",
transformation_ctx="rowLevelOutcomes",
)

rowLevelOutcomes_df = rowLevelOutcomes.toDF() # Convert Glue DynamicFrame to
SparkSQL DataFrame
rowLevelOutcomes_df_passed =
rowLevelOutcomes_df.filter(rowLevelOutcomes_df.DataQualityEvaluationResult ==
"Passed") # Filter only the Passed records.
rowLevelOutcomes_df.filter(rowLevelOutcomes_df.DataQualityEvaluationResult ==
"Failed").show(5, truncate=False) # Review the Failed records
```


출력:

```

+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
|DRG Definition          |Provider Id|Provider Name
      |Provider Street Address |Provider City|Provider State|Provider Zip
Code|Hospital Referral Region Description| Total Discharges | Average Covered
Charges | Average Total Payments |Average Medicare Payments|DataQualityRulesPass
      |DataQualityRulesFail      |DataQualityRulesSkip      |
DataQualityEvaluationResult|
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10005      |MARSHALL MEDICAL CENTER SOUTH
      |2505 U S HIGHWAY 431 NORTH|BOAZ          |AL          |35957
|AL - Birmingham          |14          |$15131.85
|$5787.57                |$4976.71   |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
|
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10046      |RIVERVIEW REGIONAL MEDICAL
CENTER |600 SOUTH THIRD STREET |GADSDEN      |AL          |35901
|AL - Birmingham          |14          |$67327.92
|$5461.57                |$4493.57   |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
|
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|10083      |SOUTH BALDWIN REGIONAL
MEDICAL CENTER|1613 NORTH MCKENZIE STREET|FOLEY        |AL          |36535
|AL - Mobile              |15          |$25411.33
|$5282.93                |$4383.73   |[IsComplete "Provider
Id"]|[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
|
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|30002      |BANNER GOOD SAMARITAN MEDICAL
CENTER |1111 EAST MCDOWELL ROAD |PHOENIX      |AZ          |85006
|AZ - Phoenix             |11          |$34803.81

```

```

|$7768.90          |$6951.45          |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
|
|039 - EXTRACRANIAL PROCEDURES W/O CC/MCC|30010          |CARONDELET ST  MARYS HOSPITAL
|1601 WEST ST MARY'S ROAD |TUCSON          |AZ          |85745
|AZ - Tucson          |12          |$35968.50
|$6506.50          |$5379.83          |[IsComplete "Provider Id"]|
[ColumnValues " Total Discharges " > 15]|[ColumnExists "Provider Id"]|Failed
|
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+
only showing top 5 rows

```

참고로 AWS Glue Data Quality에는 4개의 새 열 (DataQualityRulesPass, DataQualityRulesFail, DataQualityRulesSkip, 및 DataQualityEvaluationResult) 이 추가되었습니다. 이는 통과한 레코드, 실패한 레코드, 행 수준 평가에서 건너뛴 규칙, 전체 행 수준 결과를 나타냅니다.

- 출력을 Amazon S3 버킷에 기록하여 데이터를 분석하고 결과를 시각화합니다.

```

#Write the Passed records to the destination.

glueContext.write_dynamic_frame.from_options(
    frame = rowLevelOutcomes_df_passed,
    connection_type = "s3",
    connection_options = {"path": "s3://glue-sample-target/output-dir/
medicare_parquet"},
    format = "parquet")

```

데이터 품질 정의 언어(DQDL) 참조

데이터 품질 정의 언어 (DQDL) 는 AWS Glue 데이터 품질에 대한 규칙을 정의하는 데 사용하는 도메인 별 언어입니다.

이 안내서에서는 언어를 이해하는 데 도움이 되는 주요 DQDL 개념을 소개합니다. 또한 구문 및 예제와 함께 DQDL 규칙 유형에 대한 참조를 제공합니다. 이 가이드를 사용하기 전에 AWS Glue Data Quality에 익숙해지는 것이 좋습니다. 자세한 정보는 [AWS Glue 데이터 품질](#)을 참조하세요.

Note

DynamicRules AWS GlueETL에서만 지원됩니다.

목차

- [DQDL 구문](#)
 - [규칙 구조](#)
 - [복합 규칙](#)
 - [복합 규칙의 작동 방식](#)
 - [Expressions](#)
 - [NULL, EMPTY, WHITESPACES_ONLY에 대한 키워드](#)
 - [Where 절을 사용한 필터링](#)
 - [동적 규칙](#)
 - [분석기](#)
 - [설명](#)
- [DQDL 규칙 유형 참조](#)
 - [AggregateMatch](#)
 - [ColumnCorrelation](#)
 - [ColumnCount](#)
 - [ColumnDataType](#)
 - [ColumnExists](#)
 - [ColumnLength](#)
 - [ColumnNamesMatchPattern](#)
 - [ColumnValues](#)
 - [완전성](#)
 - [CustomSQL](#)
- [데이터 품질 정의 언어\(DQDL\) 참조](#)
 - [DataFreshness](#)

- [DatasetMatch](#)
- [DistinctValuesCount](#)
- [Entropy](#)
- [IsComplete](#)
- [IsPrimaryKey](#)
- [IsUnique](#)
- [Mean](#)
- [ReferentialIntegrity](#)
- [RowCount](#)
- [RowCountMatch](#)
- [StandardDeviation](#)
- [Sum](#)
- [SchemaMatch](#)
- [Uniqueness](#)
- [UniqueValueRatio](#)
- [DetectAnomalies](#)

DQDL 구문

DQDL 문서는 대/소문자를 구분하며 개별 데이터 품질 규칙을 그룹화하는 규칙 세트를 포함합니다. 규칙 세트를 구성하려면 한 쌍의 대괄호로 구분된 Rules(대문자) 목록을 생성해야 합니다. 목록에는 다음 예와 같이 쉼표로 구분된 DQDL 규칙이 하나 이상 포함되어야 합니다.

```
Rules = [
  IsComplete "order-id",
  IsUnique "order-id"
]
```

규칙 구조

DQDL 규칙의 구조는 규칙 유형에 따라 달라집니다. 하지만 DQDL 규칙은 일반적으로 다음 형식에 적합합니다.

```
<RuleType> <Parameter> <Parameter> <Expression>
```

RuleType은 구성하려는 규칙 유형의 이름(대/소문자 구분)입니다. 예: IsComplete, IsUnique 또는 CustomSql. 규칙 파라미터는 규칙 유형마다 다릅니다. DQDL 규칙 유형 및 파라미터에 대한 전체 참조는 [DQDL 규칙 유형 참조](#) 섹션을 참조하세요.

복합 규칙

DQDL은 규칙을 결합하는 데 사용될 수 있는 다음과 같은 논리 연산자를 지원합니다. 이러한 규칙을 복합 규칙이라고 합니다.

그리고

논리 and 연산자는 연결하는 규칙이 true인 경우에만 true 결과를 얻습니다. 그렇지 않으면 결합된 규칙에 따라 false 결과를 얻습니다. and 연산자와 연결하는 각 규칙은 괄호로 묶어야 합니다.

다음 예제에서는 and 연산자를 사용하여 두 DQDL 규칙을 결합합니다.

```
(IsComplete "id") and (IsUnique "id")
```

또는

논리 or 연산자는 연결하는 규칙 중 하나 이상이 true인 경우에만 true 결과를 얻습니다. or 연산자와 연결하는 각 규칙은 괄호로 묶어야 합니다.

다음 예제에서는 or 연산자를 사용하여 두 DQDL 규칙을 결합합니다.

```
(RowCount "id" > 100) or (IsPrimaryKey "id")
```

동일한 연산자를 사용하여 여러 규칙을 연결할 수 있으므로 다음과 같은 규칙 조합이 허용됩니다.

```
(Mean "Star_Rating" > 3) and (Mean "Order_Total" > 500) and (IsComplete "Order_Id")
```

하지만 논리 연산자를 단일 표현식으로 결합할 수 없습니다. 예를 들어 다음 결합은 허용되지 않습니다.

```
(Mean "Star_Rating" > 3) and (Mean "Order_Total" > 500) or (IsComplete "Order_Id")
```

복합 규칙의 작동 방식

기본적으로 복합 규칙은 전체 데이터세트 또는 테이블에서 개별 규칙으로 평가된 다음 결과가 결합됩니다. 즉, 전체 열을 먼저 평가한 다음 연산자를 적용합니다. 이 기본 동작은 예제와 함께 아래에 설명되어 있습니다.

```
# Dataset

+-----+-----+
|myCol1|myCol2|
+-----+-----+
|      2|      1|
|      0|      3|
+-----+-----+

# Overall outcome

+-----+-----+
|Rule                                     |Outcome|
+-----+-----+
|(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)|Failed |
+-----+-----+
```

위의 예제에서 AWS Glue Data Quality는 먼저 `(ColumnValues "myCol1" > 1)`을 평가하여 실패하게 됩니다. 그런 다음 `(ColumnValues "myCol2" > 2)`를 평가하며 역시 실패하게 됩니다. 두 결과의 조합은 FAILED로 표시됩니다.

그러나 전체 행을 평가해야 하는 SQL과 같은 동작을 선호하는 경우 아래 코드 스니펫의 `additionalOptions`에 표시된 것처럼 `ruleEvaluation.scope` 파라미터를 명시적으로 설정해야 합니다.

```
object GlueApp {
  val datasource = glueContext.getCatalogSource(
    database="<db>",
    tableName="<table>",
    transformationContext="datasource"
  ).getDynamicFrame()

  val ruleset = ""
  Rules = [
    (ColumnValues "age" >= 26) OR (ColumnLength "name" >= 4)
  ]
}
```

```

]
"""

val dq_results = EvaluateDataQuality.processRows(
  frame=datasource,
  ruleset=ruleset,
  additionalOptions=JsonOptions("""
    {
      "compositeRuleEvaluation.method":"ROW"
    }
    """)
)
}

```

AWS Glue Studio 및 AWS Glue 데이터 카탈로그에서는 아래와 같이 사용자 인터페이스에서 이 옵션을 쉽게 설정할 수 있습니다.

▼ Composite rule settings - *new*

Rule evaluation configuration [Info](#)

Configure how composite rules should work. [Learn more](#) 

Row

The composite rules will behave as single rule evaluating entire row.

Column

The composite rules will evaluate individual rules across the entire dataset and combine the results.

일단 설정되면 복합 규칙은 전체 행을 평가하는 단일 규칙으로 작동합니다. 다음 예제는 이 동작을 보여 줍니다.

```
# Row Level outcome
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```

|myCol1|myCol2|DataQualityRulesPass|
DataQualityEvaluationResult|
+-----+-----+-----+
+-----+
|2      |1      |[(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)]|Passed
|      |      |      |
|0      |3      |[(ColumnValues "myCol1" > 1) OR (ColumnValues "myCol2" > 2)]|Passed
|      |      |      |
+-----+-----+-----+
+-----+

```

일부 규칙은 임계값이나 비율에 따라 전체 결과가 달라지므로 이 기능에서 지원되지 않습니다. 이는 아래에 나열되어 있습니다.

비율에 따른 규칙:

- 완전성
- DatasetMatch
- ReferentialIntegrity
- Uniqueness

임계값에 따른 규칙:

다음 규칙에 with threshold가 포함된 경우 해당 규칙은 지원되지 않습니다. 그러나 with threshold를 포함하지 않는 규칙은 계속 지원됩니다.

- ColumnDataType
- ColumnValues
- CustomSQL

Expressions

규칙 유형이 부울 응답을 생성하지 않는 경우 부울 응답을 생성하려면 표현식을 파라미터로 제공해야 합니다. 예를 들어 다음 규칙은 열에 있는 모든 값의 중앙값(평균)을 표현식과 비교하여 true 또는 false 결과를 반환합니다.

```
Mean "colA" between 80 and 100
```


IsUnique 및 IsComplete와 같은 일부 규칙 유형은 이미 부울 응답을 반환합니다.

다음 표는 DQDL 규칙에서 사용할 수 있는 표현식을 나열합니다.

지원되는 DQDL 표현식

표현식	설명	예
<code>=x</code>	규칙 유형 응답이 <code>x</code> 와 같은 경우 <code>true</code> 로 확인됩니다.	<pre>Completeness "colA" = "1.0", ColumnValues "colA" = "2022-06-30"</pre>
<code>!=x</code>	규칙 유형 응답이 <code>x</code> 와 같지 않으면 <code>x</code> 는 <code>true</code> 로 확인됩니다.	<pre>ColumnValues "colA" != "a", ColumnValues "colA" != "2022-06-30"</pre>
<code>> x</code>	규칙 유형 응답이 <code>x</code> 보다 큰 경우 <code>true</code> 로 확인됩니다.	<pre>ColumnValues "colA" > 10</pre>
<code>< x</code>	규칙 유형 응답이 <code>x</code> 보다 작은 경우 <code>true</code> 로 확인됩니다.	<pre>ColumnValues "colA" < 1000, ColumnValues "colA" < "2022-06-30"</pre>
<code>>= x</code>	규칙 유형 응답이 <code>x</code> 보다 크거나 같은 경우 <code>true</code> 로 확인됩니다.	<pre>ColumnValues "colA" >= 10</pre>
<code><= x</code>	규칙 유형 응답이 <code>x</code> 보다 작거나 같은 경우 <code>true</code> 로 확인됩니다.	<pre>ColumnValues "colA" <= 1000</pre>
<code>between x and y</code>	규칙 유형 응답이 지정된 범위 (제외)에 속하는 경우 <code>true</code> 로 확인됩니다. 이 표현식 유형은 숫자 및 날짜 유형에만 사용해야 합니다.	<pre>Mean "colA" between 8 and 100, ColumnValues "colA" between "2022-05-31" and "2022-06-30"</pre>

표현식	설명	예
x 및 y 사이가 아님	규칙 유형 응답이 지정된 범위 (포함)에 속하지 않으면 true로 확인됩니다. 이 표현식 유형은 숫자 및 날짜 유형에만 사용해야 합니다.	ColumnValues "colA" not between "2022-05-31" and "2022-06-30"
in [a , b , c , ...]	규칙 유형 응답이 지정된 세트 내에 있는 경우 true로 확인됩니다.	ColumnValues "colA" in [1, 2, 3], ColumnValues "colA" in ["a", "b", "c"]
not in [a , b , c , ...]	규칙 유형 응답이 지정된 세트에 없으면 true로 확인됩니다.	ColumnValues "colA" not in [1, 2, 3], ColumnValues "colA" not in ["a", "b", "c"]
matches $/ab+c/i$	규칙 유형 응답이 정규 표현식과 일치하는 경우 true로 확인됩니다.	ColumnValues "colA" matches "[a-zA-Z]*"
$/ab+c/i$ 와 일치하지 않음	규칙 유형 응답이 정규 표현식과 일치하지 않는 true 경우로 해석됩니다.	ColumnValues "colA" not matches "[a-zA-Z]*"
now()	ColumnValues 규칙 유형에서만 작동하여 날짜 표현식을 생성합니다.	ColumnValues "load_date" > (now() - 3 days)

표현식	설명	예
<code>matches/in [...]/not matches/not in [...] with threshold</code>	<p>규칙 조건과 일치하는 값의 백분율을 지정합니다. <code>ColumnValues</code>, <code>ColumnDataType</code>, <code>CustomSQL</code> 규칙 유형에만 적용됩니다.</p>	<pre>ColumnValues "colA" in ["A", "B"] with threshold > 0.8, ColumnValues "colA" matches "[a-zA-Z]*" with threshold between 0.2 and 0.9 ColumnDataType "colA" = "Timestamp" with threshold > 0.9</pre>

NULL, EMPTY, WHITESPACES_ONLY에 대한 키워드

문자열 열에 null, 비어 있음 또는 공백만 있는 문자열이 있는지 확인하려면 다음 키워드를 사용할 수 있습니다.

- `NULL/null` - 이 키워드는 문자열 열의 null 값에 대해 true로 확인됩니다.

`ColumnValues "colA" != NULL with threshold > 0.5`는 데이터의 50% 이상에 null 값이 없는 경우 true를 반환합니다.

`(ColumnValues "colA" = NULL) or (ColumnLength "colA" > 5)`는 null 값이 있거나 길이가 5를 초과하는 모든 행에 대해 true를 반환합니다. 단, 이 경우 `"compositeRuleEvaluation.method" = "ROW"` 옵션을 사용해야 합니다.

- `EMPTY/empty` - 이 키워드는 문자열 열의 빈 문자열("") 값에 대해 true로 확인됩니다. 일부 데이터 형식은 문자열 열의 null을 빈 문자열로 변환합니다. 이 키워드는 데이터에서 빈 문자열을 필터링하는 데 도움이 됩니다.

`(ColumnValues "colA" = EMPTY) or (ColumnValues "colA" in ["a", "b"])`는 행이 비어 있거나 "a" 또는 "b"인 경우 true를 반환합니다. 참고로 이 경우 `"compositeRuleEvaluation.method" = "ROW"` 옵션을 사용해야 합니다.

- `WHITESPACES_ONLY/whitespaces_only` - 이 키워드는 문자열 열에 공백(" ") 값만 있는 문자열에 대해 true로 확인됩니다.

`ColumnValues "colA" not in ["a", "b", WHITESPACES_ONLY]`는 'a', 'b' 또는 공백만 있는 행이 아닌 경우 true를 반환합니다.

지원되는 규칙:

- [ColumnValues](#)

숫자 또는 날짜 기반 표현식에서 열에 null이 있는지 확인하려는 경우 다음 키워드를 사용할 수 있습니다.

- NULL/null - 이 키워드는 문자열 열의 null 값에 대해 true로 확인됩니다.

ColumnValues "colA" in [NULL, "2023-01-01"]은 열의 날짜가 2023-01-01이거나 null인 경우 true를 반환합니다.

(ColumnValues "colA" = NULL) or (ColumnValues "colA" between 1 and 9)는 null 값이 있거나 1에서 9 사이의 값을 갖는 모든 행에 대해 true를 반환합니다. 참고로 이 경우 "compositeRuleEvaluation.method" = "ROW" 옵션을 사용해야 합니다.

지원되는 규칙:

- [ColumnValues](#)

Where 절을 사용한 필터링

규칙을 작성할 때 데이터를 필터링할 수 있습니다. 이는 조건부 규칙을 적용하려는 경우에 유용합니다.

```
<DQDL Rule> where "<valid SparkSQL where clause> "
```

필터는 where 키워드와 따옴표로 묶인 유효한 SparkSQL 문을 사용하여 지정해야 합니다. ("")

임계값이 있는 규칙에 where 절을 추가하려면 임계값 조건보다 먼저 where 절을 지정해야 합니다.

```
<DQDL Rule> where "valid SparkSQL statement" with threshold <threshold condition>
```

이 구문을 사용하면 다음과 같은 규칙을 작성할 수 있습니다.

```
Completeness "colA" > 0.5 where "colB = 10"
ColumnValues "colB" in ["A", "B"] where "colC is not null" with threshold > 0.9
ColumnLength "colC" > 10 where "colD != Concat(colE, colF)"
```

제공된 SparkSQL 문이 유효한지 검증할 것입니다. 유효하지 않은 경우 규칙 평가가 실패하고 다음 형식의 `IllegalArgumentException`이 실행됩니다.

```
Rule <DQDL Rule> where "<invalid SparkSQL>" has provided an invalid where clause :
<SparkSQL Error>
```

행 수준 오류 레코드 식별이 켜져 있을 때의 Where 절 동작

AWS Glue Data Quality를 사용하면 실패한 특정 레코드를 식별할 수 있습니다. 행 수준 결과를 지원하는 규칙에 where 절을 적용할 때는 where 절로 필터링된 행에 a라는 레이블을 지정합니다. Passed 필터링된 행에 별도로 레이블을 지정하려는 경우 ETL `additionalOptions` 작업에 대해 다음과 같이 SKIPPED 설정할 수 있습니다.

```
object GlueApp {
  val datasource = glueContext.getCatalogSource(
    database="<db>",
    tableName="<table>",
    transformationContext="datasource"
  ).getDynamicFrame()

  val ruleset = """
    Rules = [
      IsComplete "att2" where "att1 = 'a'"
    ]
  """

  val dq_results = EvaluateDataQuality.processRows(
    frame=datasource,
    ruleset=ruleset,
    additionalOptions=JsonOptions("""
      {
        "rowLevelConfiguration.filteredRowLabel":"SKIPPED"
      }
    """)
  )
}
```

예를 들어, 다음 규칙 및 데이터 프레임을 참조하십시오.

```
IsComplete att2 where "att1 = 'a'"
```

id	att1	att2	행 수준 결과 (기본값)	행 수준 결과 (옵션 건너뛰기)	설명
1	a	f	통과	통과	
2	b	d	통과	SKIPPED	행은 필터링 되지 않았으 므로 att1 필 터링됩니다. "a"
3	a	null	FAILED	FAILED	
4	a	f	통과	통과	
5	b	null	통과	SKIPPED	행은 필터링 되지 않았으 므로 att1 필 터링됩니다. "a"
6	a	f	통과	통과	

동적 규칙

이제 동적 규칙을 작성하여 규칙에 의해 생성된 현재 지표를 기록 값과 비교할 수 있습니다. 이러한 기록 비교는 표현식에 `last()` 연산자를 사용하여 사용할 수 있습니다. 예를 들어 현재 실행 중인 행 수가 동일한 데이터 세트에 대한 가장 최근의 이전 행 수보다 많을 때 규칙 `RowCount > last()`가 성공합니다. `last()`는 고려할 기록 지표의 개수를 나타내는 선택적 자연수 인수를 취하고, `last(k)`는 마지막 `k` 지표를 참조하는 `k >= 1`을 취합니다.

- 사용 가능한 데이터 포인트가 없는 경우 `last(k)`는 기본값 0.0을 반환합니다.
- 사용할 수 있는 지표가 `k`개 미만인 경우 `last(k)`는 이전 지표를 모두 반환합니다.

유효한 표현식을 만들려면 `last(k)`를 사용합니다. 여기서 `k > 1`에는 여러 개의 기록 결과를 하나의 숫자로 줄이는 집계 함수가 필요합니다. 예를 들어 `RowCount > avg(last(5))`는 현재 데이터 세트

의 행 수가 동일한 데이터 세트의 최근 5개 행 수의 평균보다 확실하게 큰지 확인합니다. `RowCount > last(5)`는 현재 데이터 세트 행 수를 목록과 유의미하게 비교할 수 없기 때문에 오류를 생성합니다.

지원되는 집계 함수:

- avg
- median
- max
- min
- sum
- std(표준 편차)
- abs(절대값)
- `index(last(k), i)`를 사용하면 최근 k개 중 가장 최근인 i번째 값을 선택할 수 있습니다. i는 0으로 인덱싱되므로 `index(last(3), 0)`은 가장 최근의 데이터 포인트를 반환하고, `index(last(3), 3)`은 데이터 포인트가 3개뿐이지만 4번째 가장 최근의 데이터 포인트를 인덱싱하려고 시도하기 때문에 오류가 발생합니다.

샘플 표현식

ColumnCorrelation

- `ColumnCorrelation "colA" "colB" < avg(last(10))`

DistinctValuesCount

- `DistinctValuesCount "colA" between min(last(10))-1 and max(last(10))+1`

숫자 조건 또는 임계값이 있는 대부분의 규칙 유형은 동적 규칙을 지원합니다. 제공된 테이블인 [분석기 및 규칙](#)을 참조하여 해당 규칙 유형에 동적 규칙이 지원되는지 확인하세요.

분석기

Note

분석기는 AWS Glue 데이터 카탈로그에서 지원되지 않습니다.

DQDL 규칙은 분석기라는 함수를 사용하여 데이터에 대한 정보를 수집합니다. 이 정보는 규칙의 부울 표현식에 의해 사용되어 규칙의 성공 또는 실패 여부를 결정합니다. 예를 들어 RowCount 규칙은 $\text{RowCount} > 5$ 행 개수 분석기를 사용하여 데이터셋의 행 수를 검색하고 해당 개수를 > 5 표현식과 비교하여 현재 데이터세트에 5개 이상의 행이 있는지 확인합니다.

때로는 규칙을 작성하는 대신 분석기를 생성한 다음 이상을 탐지하는 데 사용할 수 있는 통계를 생성하는 것을 권장합니다. 이러한 경우에는 분석기를 생성할 수 있습니다. 분석기는 다음과 같은 점에서 규칙과 다릅니다.

기능	분석기	규칙
규칙 세트의 일부	예	예
통계 생성	예	예
관찰 결과 생성	예	예
조건 평가 및 어설션 가능	아니요	예
실패 시 작업 중지, 작업 계속 처리 등의 작업을 구성할 수 있음	아니요	예

분석기는 규칙 없이 독립적으로 존재할 수 있으므로 신속하게 구성하고 데이터 품질 규칙을 점진적으로 구축할 수 있습니다.

일부 규칙 유형은 규칙 세트의 Analyzers 블록에 입력하여 분석기에 필요한 규칙을 실행하고 조건에 대한 검사를 적용하지 않고 정보를 수집할 수 있습니다. 일부 분석기는 규칙과 연결되지 않으며 Analyzers 블록에만 입력할 수 있습니다. 다음 테이블에는 각 항목이 규칙 또는 독립 실행형 분석기로 지원되는지 여부와 각 규칙 유형에 대한 추가 세부 정보가 나와 있습니다.

분석기를 사용한 규칙 세트 예제

다음 규칙 세트는 다음을 사용합니다.

- 데이터 세트가 지난 세 번의 작업 실행에 대한 후행 평균보다 증가하고 있는지 확인하는 동적 규칙
- 데이터 세트의 Name 열에 고유한 값의 수를 기록하는 DistinctValuesCount 분석기
- 시간 경과에 따른 최소 및 최대 Name 크기를 추적하는 ColumnLength 분석기

분석기 메트릭 결과는 작업 실행의 Data Quality 탭에서 확인할 수 있습니다.

```
Rules = [
  RowCount > avg(last(3))
]
Analyzers = [
  DistinctValuesCount "Name",
  ColumnLength "Name"
]
```

설명

문자를 사용하여 DQDL 문서에 설명을 추가할 수 있습니다. # 문자 이후부터 줄 끝까지의 모든 내용은 DQDL에서 무시됩니다.

```
Rules = [
  # More items should generally mean a higher price, so correlation should be
  # positive
  ColumnCorrelation "price" "num_items" > 0
]
```

DQDL 규칙 유형 참조

이 섹션에서는 AWS Glue Data Quality가 지원하는 각 규칙 유형에 대한 참조를 제공합니다.

Note

- DQDL은 현재 중첩형 또는 목록형 열 데이터를 지원하지 않습니다.
- 아래 테이블의 괄호로 묶인 값은 규칙 인수에 제공된 정보로 대체됩니다.
- 규칙에는 일반적으로 표현식에 대한 추가 인수가 필요합니다.

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
Aggregat Match	총 판 매액과 같은 요약 지표를 비교하여 두 데이터 세트가 일치하는지 확인합니다. 금융 기관에서 모든 데이터가 소스 시스템에서 수집되었는지 비교할 때 유용합니다.	하나 이상의 집계	첫 번째 및 두 번째 집계 열 이름이 일치하는 경우: Column.[Column].aggregate 첫 번째 및 두 번째 집계 열 이름이 다른 경우: Column.[Column1, Column2].aggregate	예	아니요	아니요	아니요	아니요	아니요

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
AllStatistics	제공된 열 또는 데이터 세트의 모든 열에 대한 여러 지표를 수집하는 독립 실행형 분석기.	단일 열 이름 또는 "" AllColumns	모든 유형의 열에 적용됩니다. Dataset .RowColumns Column.[Column].completeness Column.[Column].liqueness 문자열 값 열에 대한 추가 지표: Columnlength metrics	아니요	예	아니요	아니요	아니요	아니요

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
			숫자 값 열에 대한 추가 지표: Column\ues metrics						
ColumnC relation	두 열의 상관 관계 적 합도를 확인합니다.	정확히 두 개의 열 이름	Multicolumn. [Column1], [Column2].ColumnCorrection	예	예	아니요	예	아니요	예
ColumnC nt	삭제된 열이 있는지 확인합니다.	None	Dataset.Columnrunt	예	아니요	아니요	예	예	아니요

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
ColumnConstraintType	열이 데이터 형식을 준수하는지 확인합니다.	정확히 하나의 열 이름	ColumnConstraintType.Con	예	아니요	아니요	예, 행 수준 임계값 표현식에서	아니요	예

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
ColumnEs	데이터 세트에 열이 있는지 확인합니다. 이를 통해 고객은 특정 열을 사용할 수 있도록 셀프 서비스 데이터 플랫폼을 구축할 수 있습니다.	정확히 하나의 열 이름	N/A	예	아니요	아니요	아니요	아니요	아니요

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
ColumnLength	데이터 길이가 일관되는지 확인합니다.	정확히 하나의 열 이름	Column.[Column].maximumLength Column.[Column].minimumLength 행수준 임계값이 제공된 경우 추가지표: Column.[Column].ColumnValues.Compance	예	예	예, 행수준 임계값이 제공된 경우	아니요	예, 최소 및 최대 길이를 분석하여 관찰 결과만 생성합니다.	예

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
ColumnNamesMatchPattern	열 이름이 정의된 패턴과 일치하는지 확인합니다. 거버넌스 팀이 열 이름의 일관성을 적용하는데 유용합니다.	열 이름에 대한 정규식	Dataset .ColumnNamesMatchFunction	예	아니요	아니요	아니요	아니요	아니요

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
ColumnValues	데이터가 정의된 값에 따라 일관되는지 확인합니다. 이 규칙은 정규식을 지원합니다.	정확히 하나의 열 이름	Column.[Column].maximum Column.[Column].minimum 행수준 임계값이 제공된 경우 추가지표: Column.[Column].ColumnValues.Compliance	예	예	예, 행수준 임계값이 제공된 경우	아니요	예 최솟값 및 최댓값을 분석하여 관찰 결과만 생성합니다.	예

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
완전성	데이터에 공백 또는 NULL이 있는지 확인합니다.	정확히 하나의 열 이름	Column.[Column].completers	예	예	예	예	예	예
CustomSQL	고객은 거의 모든 유형의 데이터 품질 검사를 SQL로 구현할 수 있습니다.	SQL 문 (선택 사항) 행수준 임계값	Dataset.CUSTOML 행수준 임계값이 제공된 경우 추가지표: Dataset.CUSTOML.Compliance	예	아니요	예, 행수준 임계값이 제공된 경우	예	아니요	아니요

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
DataFreshness	데이터가 최신인지 확인합니다.	정확히 하나의 열 이름	Column.[Column].DataFreshness.Compliance	예	아니요	예	아니요	아니요	예
DatasetMatch	두 데이터 세트를 비교하여 두 데이터 세트가 동기화되었는지 확인합니다.	참조 데이터 세트 이름 열 매핑 (선택 사항) 일치하는 항목이 있는지 확인하는 열	Dataset[DatasetReferences].DatasetMatch	예	아니요	예	예	아니요	아니요
DistinctValuesCount	중복된 값이 있는지 확인합니다.	정확히 하나의 열 이름	Column.[Column].DistinctValuesCount	예	예	예	예	예	예

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
DetectArmalies	다른 규칙 유형의 보고된 지표에 서 이상 있는지 확인합니다.	규칙 유형	규칙 유형 인수로 보고된 지표	예	아니요	아니요	아니요	아니요	아니요
Entropy	데이터의 엔트로피를 확인합니다.	정확히 하나의 열 이름	Column [Column]. tropy	예	예	아니요	예	아니요	예
IsComplete	데이터가 100% 완전한 지 확인합니다.	정확히 하나의 열 이름	Column [Column]. mpleters	예	아니요	예	아니요	아니요	예

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
IsPrimaryKey	열이 프라미리 키 (NULL이 아니고 고유함)인지 확인합니다.	정확히 하나의 열 이름	단일 열의 경우: <code>Column.[Column]. uniqueness</code> 여러 열의 경우: <code>Multicolumn[Column delimited columns]. uniqueness</code>	예	아니요	예	아니요	아니요	예
IsUnique	데이터가 100% 고유한지 확인합니다.	정확히 하나의 열 이름	<code>Column.[Column]. uniqueness</code>	예	아니요	예	아니요	아니요	예

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
평균	평균이 설정된 임계값과 일치하는지 확인합니다.	정확히 하나의 열 이름	Column [Column].an	예	예	예	예	아니요	예
ReferentialIntegrity	두 데이터 세트에 참조 관계성이 있는지 확인합니다.	데이터 세트에서 하나 이상의 열 이름 참조 데이터 세트에서 하나 이상의 열 이름	Column [Referencataset/as].ReferentialIntegrity	예	아니요	예	예	아니요	아니요

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과를 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
RowCount	레코드 수가 임계값과 일치하는지 확인합니다.	None	Dataset.RowCount	예	예	아니요	예	예	예
RowCountMatch	두 데이터 세트 사이에 레코드 수가 일치하는지 확인합니다.	참조 데이터 세트 별칭	Dataset [RefererDataset].RowCountMat	예	아니요	아니요	예	아니요	아니요
StandardDeviation	표준 편차가 임계값과 일치하는지 확인합니다.	정확히 하나의 열 이름	Column [Column].StandardDeviation	예	예	예	예	아니요	예

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
Schema	두 데이터 세트 사이에 서스키마가 일치하는지 확인합니다.	참조 데이터 세트 별칭	Dataset [Referer Datasetias].SchemaMatch	예	아니요	아니요	예	아니요	아니요
Sum	합계가 설정된 임계값과 일치하는지 확인합니다.	정확히 하나의 열 이름	Column [Column].m	예	예	아니요	예	아니요	예
Uniques	데이터 세트의 고유성이 임계값과 일치하는지 확인합니다.	정확히 하나의 열 이름	Column [Column].uniques	예	예	예	예	아니요	예

Ruletype	설명	인수	보고된 지표	규칙으로 지원되나요?	분석기로 지원되나요?	행수준 결과 반환하나요?	동적 규칙이 지원되나요?	관찰 결과 생성	Where 절 구문을 지원하나요?
UniqueValueRatio	고유 값 비율이 임계값과 일치하는지 확인합니다.	정확히 하나의 열 이름	Column.[Column].UniqueValueRatio	예	예	예	예	아니요	예

주제

- [AggregateMatch](#)
- [ColumnCorrelation](#)
- [ColumnCount](#)
- [ColumnDataType](#)
- [ColumnExists](#)
- [ColumnLength](#)
- [ColumnNamesMatchPattern](#)
- [ColumnValues](#)
- [완전성](#)
- [CustomSQL](#)
- [DataFreshness](#)
- [DatasetMatch](#)
- [DistinctValuesCount](#)
- [Entropy](#)
- [IsComplete](#)

- [IsPrimaryKey](#)
- [IsUnique](#)
- [Mean](#)
- [ReferentialIntegrity](#)
- [RowCount](#)
- [RowCountMatch](#)
- [StandardDeviation](#)
- [Sum](#)
- [SchemaMatch](#)
- [Uniqueness](#)
- [UniqueValueRatio](#)
- [DetectAnomalies](#)

AggregateMatch

지정된 표현식에서 두 열의 집계 비율을 확인합니다. 이 규칙 유형은 여러 데이터 세트에 적용됩니다. 두 열의 집계를 평가하고 첫 번째 열 집계 결과를 두 번째 열 집계 결과로 나누어 비율을 생성합니다. 제공된 표현식에서 비율을 확인하여 부울 응답을 생성합니다.

구문

열 집계

```
ColumnExists <AGG_OPERATION> (<OPTIONAL_REFERENCE_ALIAS>.<COL_NAME>)
```

- AGG_OPERATION – 집계에 사용할 연산입니다. 현재 sum 및 avg가 지원됩니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- OPTIONAL_REFERENCE_ALIAS – 기본 데이터 세트가 아닌 참조 데이터 세트에서 열을 가져온 경우 이 파라미터를 제공해야 합니다. AWS <database_name>Glue Data Catalog에서 이 규칙을 사용하는 경우 참조 별칭은 "형식을 따라야 합니다. <table_name>. <column_name>

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- COL_NAME - 집계할 열의 이름입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

예: 평균

```
"avg(rating)"
```

예: 합계

```
"sum(amount)"
```

예: 참조 데이터 세트의 열 평균

```
"avg(reference.rating)"
```

규칙

```
AggregateMatch <AGG_EXP_1> <AGG_EXP_2> <EXPRESSION>
```

- AGG_EXP_1 - 첫 번째 열 집계입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- AGG_EXP_2 - 두 번째 열 집계입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 합계를 사용한 일치 항목 집계

다음 예제 규칙은 amount 열에서 값의 합계가 total_amount 열에서 값의 합계와 정확히 같은지 여부를 확인합니다.

```
AggregateMatch "sum(amount)" "sum(total_amount)" = 1.0
```

예: 평균을 사용한 일치 항목 집계

다음 예제 규칙은 ratings 열에서 값의 평균이 reference 데이터 세트 내 ratings 열에서 값의 평균 중 90% 이상인지를 확인합니다. 참조 데이터 세트는 ETL 또는 데이터 카탈로그 환경에서 추가 데이터 소스로 제공됩니다.

AWS Glue ETL에서는 다음을 사용할 수 있습니다.

```
AggregateMatch "avg(ratings)" "avg(reference.ratings)" >= 0.9
```

AWS Glue 데이터 카탈로그에서는 다음을 사용할 수 있습니다.

```
AggregateMatch "avg(ratings)" "avg(database_name.tablename.ratings)" >= 0.9
```

Null 동작

AggregateMatch 규칙은 집계 방법(합계/평균)의 계산에서 NULL 값이 있는 행을 무시합니다. 예:

```
+---+-----+
|id |units   |
+---+-----+
|100|0       |
|101|null  |
|102|20     |
|103|null  |
|104|40     |
+---+-----+
```

units 열의 평균은 $(0 + 20 + 40)/3 = 20$ 이 됩니다. 101행과 103행은 이 계산에서 고려되지 않습니다.

ColumnCorrelation

주어진 표현식과 비교하여 두 열 간의 상관 관계를 확인합니다. AWS Glue Data Quality는 피어슨 상관 계수를 사용하여 두 열 간의 선형 상관 관계를 측정합니다. 결과는 관계의 강도와 방향을 측정하는 -1 과 1 사이의 숫자입니다.

구문

```
ColumnCorrelation <COL_1_NAME> <COL_2_NAME> <EXPRESSION>
```

- COL_1_NAME - 데이터 품질 규칙을 평가할 첫 번째 열의 이름입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- COL_2_NAME - 데이터 품질 규칙을 평가할 첫 번째 열의 이름입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 열 상관 관계

다음 예제 규칙은 열 height 및 weight 간의 상관 계수가 강한 양의 상관 관계(0.8보다 큰 계수 값)를 갖는지 여부를 확인합니다.

```
ColumnCorrelation "height" "weight" > 0.8
```

```
ColumnCorrelation "weightinkgs" "Salary" > 0.8 where "weightinkgs > 40"
```

샘플 동적 규칙

- ColumnCorrelation "colA" "colB" between min(last(10)) and max(last(10))
- ColumnCorrelation "colA" "colB" < avg(last(5)) + std(last(5))

Null 동작

ColumnCorrelation 규칙은 상관 관계 계산에서 NULL 값이 있는 행을 무시합니다. 예:

```
+---+-----+
|id |units  |
+---+-----+
|100|0      |
|101|null  |
|102|20     |
|103|null  |
|104|40     |
+---+-----+
```

101행과 103행은 무시되고 ColumnCorrelation이 1.0이 됩니다.

ColumnCount

지정된 표현식을 기준으로 기본 데이터 세트의 열 수를 확인합니다. 표현식에서 > 및 < 같은 연산자를 사용하여 열 수 또는 열 범위를 지정할 수 있습니다.

구문

```
ColumnCount <EXPRESSION>
```

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 열 수 숫자 검사

다음 예제 규칙은 열 수가 지정된 범위 내에 있는지 확인합니다.

```
ColumnCount between 10 and 20
```

샘플 동적 규칙

- ColumnCount >= avg(last(10))
- ColumnCount between min(last(10))-1 and max(last(10))+1

ColumnDataType

제공된 예상 유형과 비교하여 지정된 열에 있는 값의 고유한 데이터 형식을 확인합니다. with threshold 식을 수락하여 열에 있는 값의 하위 세트를 확인합니다.

구문

```
ColumnDataType <COL_NAME> = <EXPECTED_TYPE>
ColumnDataType <COL_NAME> = <EXPECTED_TYPE> with threshold <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 문자열 유형

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPECTED_TYPE - 열에 있는 값의 예상 유형입니다.

지원되는 값: Boolean, Date, Timestamp, Integer, Double, Float, Long

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPRESSION - 예상 유형이어야 하는 값의 백분율을 지정하는 선택적 표현식입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

예: 열 데이터 형식 정수 역할의 문자열

다음 예제 규칙은 문자열 유형인 지정된 열의 값이 실제로 정수인지 여부를 확인합니다.

```
ColumnDataType "colA" = "INTEGER"
```

예: 열 데이터 형식 정수 역할의 문자열에서 값의 하위 세트 확인

다음 예제 규칙은 문자열 유형인 지정된 열의 값 중 90%가 실제로 정수인지 여부를 확인합니다.

```
ColumnDataType "colA" = "INTEGER" with threshold > 0.9
```

ColumnExists

열이 존재하는지 확인합니다.

구문

```
ColumnExists <COL_NAME>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

예: 열이 존재함

다음 예제 규칙은 Middle_Name 열이 존재하는지 여부를 확인합니다.

```
ColumnExists "Middle_Name"
```

ColumnLength

열에 있는 각 행의 길이가 지정된 표현식을 준수하는지 확인합니다.

구문

```
ColumnLength <COL_NAME><EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 문자열

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 열 행 길이

다음 예제 규칙은 Postal_Code 열의 각 행에 있는 값의 길이가 5자인지 여부를 확인합니다.

```
ColumnLength "Postal_Code" = 5
ColumnLength "weightinkgs" = 2 where "weightinkgs" > 10"
```

Null 동작

ColumnLength 규칙은 NULL을 0 길이 문자열로 취급합니다. NULL 행의 경우:

```
ColumnLength "Postal_Code" > 4 # this will fail
```

```
ColumnLength "Postal_Code" < 6 # this will succeed
```

다음 예시 복합 규칙은 NULL 값을 명시적으로 실패하도록 하는 방법을 제공합니다.

```
(ColumnLength "Postal_Code" > 4) AND (ColumnValues != NULL)
```

ColumnNamesMatchPattern

기본 데이터 세트의 모든 열 이름이 지정된 정규식과 일치하는지 확인합니다.

구문


```
ColumnNamesMatchPattern <PATTERN>
```

- PATTERN - 데이터 품질 규칙을 평가하는 데 사용할 패턴입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

예: 열 이름 일치 패턴

다음 예제 규칙은 모든 열이 접두사 'aws_'로 시작하는지 여부를 확인합니다.

```
ColumnNamesMatchPattern "aws_.*"
ColumnNamesMatchPattern "aws_.*" where "weightinkgs > 10"
```

ColumnValues

열의 값에 대해 표현식을 실행합니다.

구문

```
ColumnValues <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 허용된 값

다음 예제 규칙은 지정된 열의 각 값이 허용되는 값 세트(null, 비어 있음, 공백만 있는 문자열 포함)에 있는지 확인합니다.

```
ColumnValues "Country" in [ "US", "CA", "UK", NULL, EMPTY, WHITESPACES_ONLY ]
ColumnValues "gender" in ["F", "M"] where "weightinkgs < 10"
```

예: 정규 표현식

다음 예제 규칙은 열의 값을 정규 표현식과 비교하여 검사합니다.

```
ColumnValues "First_Name" matches "[a-zA-Z]*"
```

예: 날짜 값

다음 예제 규칙은 날짜 열의 값을 날짜 표현식과 비교하여 검사합니다.

```
ColumnValues "Load_Date" > (now() - 3 days)
```

예: 숫자 값

다음 예제 규칙은 열 값이 특정 숫자 제약 조건과 일치하는지 여부를 확인합니다.

```
ColumnValues "Customer_ID" between 1 and 2000
```

Null 동작

모든 ColumnValues 규칙(!= 및 NOT IN 제외)의 경우 NULL 행이 규칙에 실패합니다. null 값으로 인해 규칙이 실패하면 실패 이유가 다음과 같이 표시됩니다.

```
Value: NULL does not meet the constraint requirement!
```

다음 예제 복합 규칙은 NULL 값을 명시적으로 허용하는 방법을 제공합니다.

```
(ColumnValues "Age" > 21) OR (ColumnValues "Age" = NULL)
```

and 구문을 사용하는 부정 ColumnValues 규칙은 행에 != not in 전달됩니다. NULL 예:

```
ColumnValues "Age" != 21
```

```
ColumnValues "Age" not in [21, 22, 23]
```

다음 예제는 NULL 값을 명시적으로 실패하도록 하는 방법을 제공합니다.

```
(ColumnValues "Age" != 21) AND (ColumnValues "Age" != NULL)
```

```
ColumnValues "Age" not in [21, 22, 23, NULL]
```

완전성

지정된 표현식을 기준으로 열에서 전체(null이 아닌) 값의 백분율을 검사합니다.

구문

```
Completeness <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: Null 값 백분율

다음 예제 규칙은 열에 있는 값의 95% 이상이 완전한지를 확인합니다.

```
Completeness "First_Name" > 0.95
Completeness "First_Name" > 0.95 where "weightinkgs" > 10
```

샘플 동적 규칙

- Completeness "colA" between min(last(5)) - 1 and max(last(5)) + 1
- Completeness "colA" <= avg(last(10))

Null 동작

CSV 데이터 형식에 대한 참고 사항: CSV 열의 빈 행은 여러 동작을 표시할 수 있습니다.

- 열의 유형이 String인 경우 빈 행은 빈 문자열로 인식되어 Completeness 규칙에 실패하지 않습니다.
- 열이 Int와 같은 다른 데이터 유형인 경우 빈 행이 NULL로 인식되어 Completeness 규칙에 실패합니다.

CustomSQL

이 규칙 유형은 다음과 같은 두 가지 사용 사례를 지원하도록 확장되었습니다.

- 데이터 세트에 대해 사용자 지정 SQL 문을 실행하고 지정된 표현식을 기준으로 반환 값을 확인합니다.
- 일부 조건과 비교하여 행 수준 결과를 얻는 SELECT 문에 열 이름을 지정하는 사용자 지정 SQL 문을 실행합니다.

구문

```
CustomSql <SQL_STATEMENT> <EXPRESSION>
```

- SQL_STATEMENT - 큰따옴표로 묶인 단일 숫자 값을 반환하는 SQL 문입니다.
- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 전체 규칙 결과를 검색하기 위한 사용자 지정 SQL

이 예제 규칙에서는 SQL 문을 사용하여 데이터 세트의 레코드 수를 검색합니다. 그런 다음 레코드 수가 10~20 사이인지 확인합니다.

```
CustomSql "select count(*) from primary" between 10 and 20
```

예: 행 수준 결과를 검색하기 위한 사용자 지정 SQL

이 예제 규칙은 일부 조건과 비교하여 행 수준 결과를 얻는 SELECT 문에 열 이름을 지정하는 SQL 문을 사용합니다. 임계값 조건 표현식에서는 전체 규칙이 실패하기 위해 실패해야 하는 레코드 수의 임계값을 정의합니다. 규칙에는 조건과 키워드가 모두 포함되지 않을 수도 있습니다.

```
CustomSql "select Name from primary where Age > 18"
```

또는

```
CustomSql "select Name from primary where Age > 18" with threshold > 3
```

Important

`primary` 별칭은 평가하려는 데이터 세트의 이름을 나타냅니다. 콘솔에서 시각적 ETL 작업을 수행할 때 `primary`에서는 항상 `EvaluateDataQuality.apply()` 변환에 전달 중인

DynamicFrame을 나타냅니다. AWS Glue Data Catalog를 사용하여 테이블에 대해 데이터 품질 작업을 실행하는 경우 primary 테이블이 표시됩니다.

AWS Glue 데이터 카탈로그에 있는 경우 실제 테이블 이름을 사용할 수도 있습니다.

```
CustomSql "select count(*) from database.table" between 10 and 20
```

여러 테이블을 조인하여 서로 다른 데이터 요소를 비교할 수도 있습니다.

```
CustomSql "select count(*) from database.table inner join database.table2 on id1 = id2"
between 10 and 20
```

AWS Glue ETL에서 CustomSQL은 데이터 품질 검사에 실패한 레코드를 식별할 수 있습니다. 이 작업을 수행하려면 데이터 품질을 평가하는 기본 테이블의 일부인 레코드를 반환해야 합니다. 쿼리의 일부로 반환된 레코드는 성공으로 간주되고 반환되지 않은 레코드는 실패로 간주됩니다.

다음 규칙에서는 age가 100 미만인 레코드는 성공으로, 그 이상인 레코드는 실패로 표시됩니다.

```
CustomSql "select id from primary where age < 100"
```

레코드 중 50%에서 age가 10을 초과하는 경우 이 CustomSQL 규칙은 통과되며 실패한 레코드도 식별합니다. 이 CustomSQL에서 반환된 레코드는 통과로 간주되고 반환되지 않은 레코드는 실패로 간주됩니다.

```
CustomSQL "select ID, CustomerID from primary where age > 10" with threshold > 0.5
```

참고: 데이터 세트에서 사용할 수 없는 레코드를 반환하면 CustomSQL 규칙이 실패합니다.

DataFreshness

현재 시간과 날짜 열 값 간의 차이를 평가하여 열에 있는 데이터의 최신성을 검사합니다. 이 규칙 유형에 시간 기반 표현식을 지정하여 열 값이 최신 상태인지 확인할 수 있습니다.

구문

```
DataFreshness <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 날짜

- 표현식 - 시간 또는 일 단위의 숫자 표현식입니다. 표현식에 시간 단위를 지정해야 합니다.

예: 데이터 최신성

다음 예제 규칙은 데이터 최신성을 확인합니다.

```
DataFreshness "Order_Date" <= 24 hours
DataFreshness "Order_Date" between 2 days and 5 days
```

Null 동작

DataFreshness 규칙은 NULL 값이 있는 행에 대해 실패합니다. null 값으로 인해 규칙이 실패하면 실패 이유가 다음과 같이 표시됩니다.

```
80.00 % of rows passed the threshold
```

여기서 실패한 행의 20%에는 NULL이 있는 행이 포함됩니다.

다음 예제 복합 규칙은 NULL 값을 명시적으로 허용하는 방법을 제공합니다.

```
(DataFreshness "Order_Date" <= 24 hours) OR (ColumnValues "Order_Date" = NULL)
```

Amazon S3 객체의 데이터 최신성

Amazon S3 파일 생성 시간을 기준으로 데이터의 최신성을 검증해야 하는 경우가 있습니다. 이렇게 하려면 다음 코드를 사용하여 타임스탬프를 가져와 데이터 프레임에 추가한 다음 데이터 최신성 검사를 적용할 수 있습니다.

```
df = glueContext.create_data_frame.from_catalog(database = "default", table_name =
  "mytable")
df = df.withColumn("file_ts", df["_metadata.file_modification_time"])

Rules = [
  DataFreshness "file_ts" < 24 hours
]
```

DatasetMatch

기본 데이터 세트의 데이터가 참조 데이터 세트의 데이터와 일치하는지 확인합니다. 제공된 키 열 매핑을 사용하여 두 데이터 세트를 조인합니다. 해당 열에서만 데이터의 관계(equality)를 확인하려는 경우 추가 열 매핑을 제공할 수 있습니다. 참고로 DataSetMatch 작동하려면 조인 키가 고유해야 하고 NULL이 아니어야 합니다 (기본 키여야 함). 이러한 조건을 충족하지 않으면 'Provided key map not suitable for given data frames' 오류 메시지가 표시됩니다. 고유한 조인 키를 사용할 수 없는 경우에는 요약 데이터를 기준으로 일치시키는 것과 같은 다른 규칙 유형을 사용하는 AggregateMatch 것이 좋습니다.

구문

```
DatasetMatch <REFERENCE_DATASET_ALIAS> <JOIN_CONDITION_WITH_MAPPING> <OPTIONAL_MATCH_COLUMN_MAPPINGS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS – 기본 데이터 세트의 데이터를 비교하는 참조 데이터 세트의 별칭입니다.
- KEY_COLUMN_MAPPINGS – 데이터 세트의 키를 구성하는 쉼표로 구분된 열 이름 목록입니다. 두 데이터 세트의 열 이름이 동일하지 않은 경우 ->로 구분해야 합니다.
- OPTIONAL_MATCH_COLUMN_MAPPINGS - 특정 열의 데이터만 일치하는지 확인하려는 경우 이 파라미터를 제공할 수 있습니다. 키 열 매핑과 동일한 구문을 사용합니다. 이 파라미터를 제공하지 않으면 나머지 모든 열의 데이터를 일치시킵니다. 키가 아닌 나머지 열은 두 데이터 세트에서 이름이 같아야 합니다.
- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: ID 열을 사용하여 설정된 데이터 세트 일치

다음 예제 규칙은 두 데이터 세트를 조인하기 위해 'ID' 열을 사용하여 기본 데이터 세트 중 90%가 넘는 항목이 참조 데이터 세트와 일치하는지 확인합니다. 이 경우 모든 열을 비교합니다.

```
DatasetMatch "reference" "ID" >= 0.9
```

예: 여러 키 열을 사용하는 설정된 데이터 세트 일치

다음 예제에서는 기본 데이터 세트와 참조 데이터 세트의 키 열 이름이 다릅니다. ID_1 및 ID_2는 기본 데이터 세트에서 복합 키를 함께 형성합니다. ID_ref1 및 ID_ref2는 참조 데이터 세트에 복합 키를 함께 형성합니다. 이 시나리오에서는 특수 구문을 사용하여 열 이름을 제공할 수 있습니다.

```
DatasetMatch "reference" "ID_1->ID_ref1,ID_ref2->ID_ref2" >= 0.9
```

예: 여러 키 열을 사용하여 설정된 데이터 세트 일치 및 특정 열이 일치하는지 확인

이 예는 이전 예를 기반으로 구축되었습니다. 금액이 포함된 열만 일치하는지 확인하려고 합니다. 이 열의 이름은 기본 데이터 세트에서 Amount1이고, 참조 데이터 세트에서 Amount2입니다. 정확한 일치시키려고 합니다.

```
DatasetMatch "reference" "ID_1->ID_ref1,ID_ref2->ID_ref2" "Amount1->Amount2" >= 0.9
```

DistinctValuesCount

지정된 표현식을 기준으로 열의 고유 값 수를 확인합니다.

구문

```
DistinctValuesCount <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 고유 열 값 개수

다음 예제 규칙은 State 열에 3개 이상의 고유 값이 포함되어 있는지 확인합니다.

```
DistinctValuesCount "State" > 3
DistinctValuesCount "Customer_ID" < 6 where "Customer_ID < 10"
```

샘플 동적 규칙

- DistinctValuesCount "colA" between avg(last(10))-1 and avg(last(10))+1
- DistinctValuesCount "colA" <= index(last(10),2) + std(last(5))

Entropy

열의 엔트로피 값이 지정된 표현식과 일치하는지 확인합니다. 엔트로피는 메시지에 포함된 정보의 수준을 측정합니다. 열의 값에 대한 확률 분포를 고려할 때 엔트로피는 값을 식별하는 데 필요한 비트 수를 설명합니다.

구문

```
Entropy <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 열 엔트로피

다음 예제 규칙은 Feedback 열의 엔트로피 값이 1보다 큰지 확인합니다.

```
Entropy "Star_Rating" > 1
Entropy "First_Name" > 1 where "Customer_ID < 10"
```

샘플 동적 규칙

- Entropy "colA" < max(last(10))
- Entropy "colA" between min(last(10)) and max(last(10))

IsComplete

열의 모든 값이 완전한지(null이 아님) 확인합니다.

구문

```
IsComplete <COL_NAME>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

예: Null 값

다음 예에서는 email 열의 모든 값이 null이 아닌지 확인합니다.

```
IsComplete "email"
IsComplete "Email" where "Customer_ID between 1 and 50"
IsComplete "Customer_ID" where "Customer_ID < 16 and Customer_ID != 12"
IsComplete "passenger_count" where "payment_type<>0"
```

Null 동작

CSV 데이터 형식에 대한 참고 사항: CSV 열의 빈 행은 여러 동작을 표시할 수 있습니다.

- 열의 유형이 String인 경우 빈 행은 빈 문자열로 인식되어 Completeness 규칙에 실패하지 않습니다.
- 열이 Int와 같은 다른 데이터 유형인 경우 빈 행이 NULL로 인식되어 Completeness 규칙에 실패합니다.

IsPrimaryKey

열에 기본 키가 포함되어 있는지 확인합니다. 열의 모든 값이 고유하고 완전한 경우(null이 아닌 경우) 열에 기본 키가 포함됩니다.

구문

```
IsPrimaryKey <COL_NAME>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

예: 기본 키

다음 예제 규칙은 Customer_ID 열에 기본 키가 있는지 여부를 확인합니다.

```
IsPrimaryKey "Customer_ID"
```

```
IsPrimaryKey "Customer_ID" where "Customer_ID < 10"
```

예: 여러 열이 있는 프라이머리 키입니다. 다음 예제는 유효합니다.

```
IsPrimaryKey "colA" "colB"
IsPrimaryKey "colA" "colB" "colC"
IsPrimaryKey colA "colB" "colC"
```

IsUnique

열의 모든 값이 고유한지 확인하고 부울 값을 반환합니다.

구문

```
IsUnique <COL_NAME>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

예: 고유한 열 값

다음 예제 규칙은 email 열의 모든 값이 고유한지 확인합니다.

```
IsUnique "email"
IsUnique "Customer_ID" where "Customer_ID < 10"]
```

Mean

열에 있는 모든 값의 평균이 지정된 표현식과 일치하는지 확인합니다.

구문

```
Mean <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- **EXPRESSION** - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 평균값

다음 예제 규칙은 열에 있는 모든 값의 평균이 임계값을 초과하는지 여부를 확인합니다.

```
Mean "Star_Rating" > 3
Mean "Salary" < 6200 where "Customer_ID < 10"
```

샘플 동적 규칙

- Mean "colA" > avg(last(10)) + std(last(2))
- Mean "colA" between min(last(5)) - 1 and max(last(5)) + 1

Null 동작

Mean 규칙은 평균을 계산할 때 NULL 값이 있는 행을 무시합니다. 예:

```
+---+-----+
|id|units  |
+---+-----+
|100|0      |
|101|null  |
|102|20    |
|103|null  |
|104|40    |
+---+-----+
```

units 열의 평균은 $(0 + 20 + 40)/3 = 20$ 이 됩니다. 101행과 103행은 이 계산에서 고려되지 않습니다.

ReferentialIntegrity

기본 데이터 세트의 열 세트 값이 어느 범위까지 참조 데이터 세트의 열 세트 값에 대한 하위 세트인지를 확인합니다.

구문

```
ReferentialIntegrity <PRIMARY_COLS> <REFERENCE_DATASET_COLS> <EXPRESSION>
```

- PRIMARY_COLS - 기본 데이터 세트의 심표로 구분된 열 이름 목록입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- REFERENCE_DATASET_COLS - 이 파라미터는 마침표로 구분된 두 부분을 포함합니다. 첫 번째 부분은 참조 데이터 세트의 별칭입니다. 두 번째 부분은 참조 데이터 세트에서 괄호로 묶인 심표로 구분된 열 이름 목록입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: zip code 열의 참조 무결성 확인

다음 예제 규칙은 기본 데이터 세트의 zipcode 열에 있는 값 중 90%가 넘는 항목이 reference 데이터 세트의 zipcode 열에 있는지 확인합니다.

```
ReferentialIntegrity "zipcode" "reference.zipcode" >= 0.9
```

예: city 및 state 열의 참조 무결성 확인

다음 예제에서는 city 및 state 정보가 포함된 열이 기본 데이터 세트와 참조 데이터 세트에 있습니다. 두 데이터 세트의 열 이름은 서로 다릅니다. 이 규칙은 기본 데이터 세트의 열 값 세트가 참조 데이터 세트의 열 값 세트와 정확히 같은지 확인합니다.

```
ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" = 1.0
```

샘플 동적 규칙

- `ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" > avg(last(10))`
- `ReferentialIntegrity "city,state" "reference.{ref_city,ref_state}" between min(last(10)) - 1 and max(last(10)) + 1`

RowCount

지정된 표현식을 기준으로 데이터 세트의 행 수를 확인합니다. 표현식에서 > 및 < 같은 연산자를 사용하여 행 수 또는 행 범위를 지정할 수 있습니다.

구문

```
RowCount <EXPRESSION>
```

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 행 수 수치 검사

다음 예제 규칙은 행 수가 지정된 범위 내에 있는지 확인합니다.

```
RowCount between 10 and 100
RowCount between 1 and 50 where "Customer_ID < 10"
```

샘플 동적 규칙

```
RowCount > avg(lats(10)) *0.8
```

RowCountMatch

주어진 표현식과 비교하여 기본 데이터 세트의 행 수와 참조 데이터 세트의 행 수 비율을 확인합니다.

구문

```
RowCountMatch <REFERENCE_DATASET_ALIAS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS - 행 수를 비교할 참조 데이터 세트의 별칭입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 참조 데이터 세트를 기준으로 행 수 확인

다음 예제 규칙은 기본 데이터 세트의 행 수가 참조 데이터 세트 행 수의 90% 이상인지 확인합니다.

```
RowCountMatch "reference" >= 0.9
```

StandardDeviation

지정된 표현식을 기준으로 열에 있는 모든 값의 표준 편차를 검사합니다.

구문

```
StandardDeviation <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 표준 편차

다음 예제 규칙은 colA 열에 있는 값의 표준 편차가 지정된 값보다 작은지 여부를 확인합니다.

```
StandardDeviation "Star_Rating" < 1.5
StandardDeviation "Salary" < 3500 where "Customer_ID < 10"
```

샘플 동적 규칙

- StandardDeviation "colA" > avg(last(10)) + 0.1
- StandardDeviation "colA" between min(last(10)) - 1 and max(last(10)) + 1

Null 동작

StandardDeviation 규칙은 표준 편차를 계산할 때 NULL 값이 있는 행을 무시합니다. 예:

```
+---+-----+-----+
|id |units1      |units2      |
+---+-----+-----+
|100|0           |0           |
|101|null      |0           |
|102|20          |20          |
|103|null      |0           |
|104|40          |40          |
+---+-----+-----+
```

units1 열의 표준 편차는 101열과 103열을 고려하지 않고 16.33으로 계산됩니다. units2 열의 표준 편차는 16이 됩니다.

Sum

지정된 표현식을 기준으로 열에 있는 모든 값의 합계를 확인합니다.

구문

```
Sum <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 합계

다음 예제 규칙은 열에 있는 모든 값의 합계가 지정된 임계값을 초과하는지 여부를 확인합니다.

```
Sum "transaction_total" > 500000
Sum "Salary" < 55600 where "Customer_ID < 10"
```

샘플 동적 규칙

- Sum "ColA" > avg(last(10))
- Sum "colA" between min(last(10)) - 1 and max(last(10)) + 1

Null 동작

Sum 규칙은 합계를 계산할 때 NULL 값이 있는 행을 무시합니다. 예:

```
+---+-----+
|id |units   |
+---+-----+
|100|0       |
|101|null  |
|102|20     |
```



```
|103|null      |
|104|40        |
+---+-----+

```

units 열의 합계는 101열과 103열을 고려하지 않고 $(0 + 20 + 40) = 60$ 으로 계산됩니다.

SchemaMatch

기본 데이터 세트의 스키마가 참조 데이터 세트의 스키마와 일치하는지 확인합니다. 스키마 검사는 열 단위로 수행됩니다. 이름이 동일하고 유형이 동일하면 두 열의 스키마가 일치합니다. 열 순서는 중요하지 않습니다.

구문

```
SchemaMatch <REFERENCE_DATASET_ALIAS> <EXPRESSION>
```

- REFERENCE_DATASET_ALIAS - 스키마를 비교할 때 사용할 참조 데이터 세트의 별칭입니다.

지원되는 열 유형: Byte, Decimal, Double, Float, Integer, Long, Short

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: SchemaMatch

다음 예제 규칙은 기본 데이터 세트의 스키마가 참조 데이터 세트의 스키마와 정확히 일치하는지 확인합니다.

```
SchemaMatch "reference" = 1.0
```

Uniqueness

지정된 표현식을 기준으로 열에서 고유한 값의 백분율을 검사합니다. 고유한 값은 정확히 한 번 발생합니다.

구문

```
Uniqueness <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 고유성 백분율

다음 예제 규칙은 열의 고유 값 백분율이 특정 숫자 기준과 일치하는지 여부를 확인합니다.

```
Uniqueness "email" = 1.0
Uniqueness "Customer_ID" != 1.0 where "Customer_ID" < 10"
```

샘플 동적 규칙

- Uniqueness "colA" between min(last(10)) and max(last(10))
- Uniqueness "colA" >= avg(last(10))

UniqueValueRatio

지정된 표현식을 기준으로 열의 고유 값 비율을 검사합니다. 고유 값 비율은 고유 값의 비율을 열에 있는 모든 고유 값의 수로 나눈 값입니다. 고유 값은 정확히 한 번 나타나는 반면 개별 값은 한 번 이상 나타납니다.

예를 들어 [a, a, b] 세트에는 고유 값 1개(b)와 개별 값 2개(a 및 b)가 포함됩니다. 따라서 세트의 고유 값 비율은 $\frac{1}{2} = 0.5$ 입니다.

구문

```
UniqueValueRatio <COL_NAME> <EXPRESSION>
```

- COL_NAME - 데이터 품질 규칙을 평가할 열의 이름입니다.

지원되는 열 유형: 모든 열 유형

- EXPRESSION - 부울 값을 생성하기 위해 규칙 유형 응답에 대해 실행할 표현식입니다. 자세한 정보는 [Expressions](#)을 참조하세요.

예: 고유 값 비율

이 예에서는 열의 고유 값 비율을 값 범위와 비교하여 확인합니다.

```
UniqueValueRatio "test_score" between 0 and 0.5
UniqueValueRatio "Customer_ID" between 0 and 0.9 where "Customer_ID < 10"
```

샘플 동적 규칙

- UniqueValueRatio "colA" > avg(last(10))
- UniqueValueRatio "colA" <= index(last(10),2) + std(last(5))

DetectAnomalies

지정된 데이터 품질 규칙의 이상을 탐지합니다. DetectAnomalies 규칙을 실행할 때마다 지정된 규칙에 대한 평가 값이 저장됩니다. 수집된 데이터가 충분하면 예외 항목 탐지 알고리즘은 해당 규칙에 대한 모든 과거 데이터를 가져와 예외 항목 탐지를 실행합니다. DetectAnomalies 예외 항목이 탐지되면 규칙이 실패합니다. 어떤 이상이 탐지되었는지에 대한 자세한 정보는 관찰에서 확인할 수 있습니다.

구문

```
DetectAnomalies <RULE_NAME> <RULE_PARAMETERS>
```

RULE_NAME - 이상을 평가하고 탐지하려는 규칙의 이름입니다. 지원되는 규칙:

- "RowCount"
- "Completeness"
- "Uniqueness"
- "Mean"
- "Sum"
- "StandardDeviation"
- "Entropy"
- "DistinctValuesCount"
- "UniqueValueRatio"
- "ColumnLength"
- "ColumnValues"

- "ColumnCorrelation"

RULE_PARAMETERS - 일부 규칙을 실행하려면 추가 파라미터가 필요합니다. 필수 파라미터를 확인하려면 해당 규칙 설명서를 참조하세요.

예: 에 대한 예외 RowCount

예를 들어, RowCount 이상 징후를 탐지하려는 경우 규칙 이름으로 제공합니다. RowCount

```
DetectAnomalies "RowCount"
```

예: 에 대한 예외 항목 ColumnLength

예를 들어, ColumnLength 이상 징후를 탐지하려는 경우 규칙 이름과 열 ColumnLength 이름을 제공합니다.

```
DetectAnomalies "ColumnLength" "id"
```

API를 사용하여 데이터 품질 측정 및 관리

이 주제에서는 API를 사용하여 데이터 품질을 측정하고 관리하는 방법에 대해 설명합니다.

목차

- [필수 조건](#)
- [AWS Glue Data Quality 권장 사용](#)
- [AWS Glue Data Quality 규칙 세트 사용](#)
- [AWS Glue Data Quality 실행 사용](#)
- [AWS Glue Data Quality 결과 사용](#)

필수 조건

- 최신 AWS Glue Data Quality API가 포함되도록 boto3 버전이 최신 버전인지 확인합니다.
- 최신 CLI를 포함하도록 AWS CLI 버전이 최신인지 확인합니다.

AWS Glue 작업을 사용하여 이러한 API를 실행하는 경우 다음 옵션을 사용하여 boto3 라이브러리를 최신 버전으로 업데이트할 수 있습니다.

```
-additional-python-modules boto3==<version>
```

AWS Glue Data Quality 권장 사용

AWS Glue Data Quality 권장 실행을 시작하려면:

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def start_data_quality_rule_recommendation_run(self, database_name, table_name,
    role_arn):
        """
        Starts a recommendation run that is used to generate rules when you don't
        know what rules to write. AWS Glue Data Quality analyzes the data and comes up with
        recommendations for a potential ruleset. You can then triage the ruleset and modify
        the generated ruleset to your liking.

        :param database_name: The name of the AWS Glue database which contains the
        dataset.
        :param table_name: The name of the AWS Glue table against which we want a
        recommendation
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and Access
        Management (IAM) role that grants permission to let AWS Glue access the resources it
        needs.

        """
        try:
            response = self.client.start_data_quality_rule_recommendation_run(
                DataSource={
                    'GlueTable': {
                        'DatabaseName': database_name,
                        'TableName': table_name
                    }
                },
                Role=role_arn
            )
        except ClientError as err:
            logger.error(
```

```

        "Couldn't start data quality recommendation run %s. Here's why: %s:
%s", name,
        err.response['Error']['Code'], err.response['Error']['Message'])
    raise
else:
    return response['RunId']

```

권장 실행의 경우 `pushDownPredicates` 또는 `catalogPartitionPredicates`를 사용하여 성능을 개선하고 카탈로그 소스의 특정 파티션에서만 권장을 실행할 수 있습니다.

```

client.start_data_quality_rule_recommendation_run(
    DataSource={
        'GlueTable': {
            'DatabaseName': database_name,
            'TableName': table_name,
            'AdditionalOptions': {
                'pushDownPredicate': "year=2022"
            }
        }
    },
    Role=role_arn,
    NumberOfWorkers=2,
    CreatedRulesetName='<rule_set_name>'
)

```

AWS Glue Data Quality 권장 실행 결과를 얻으려면:

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_data_quality_rule_recommendation_run(self, run_id):
        """
        Gets the specified recommendation run that was used to generate rules.

        :param run_id: The id of the data quality recommendation run

        """
        try:

```

```

        response =
self.client.get_data_quality_rule_recommendation_run(RunId=run_id)
    except ClientError as err:
        logger.error(
            "Couldn't get data quality recommendation run %. Here's why: %s: %s",
run_id,
            err.response['Error']['Code'], err.response['Error']['Message'])
        raise
    else:
        return response

```

위의 응답 객체에서 실행에서 권장한 규칙 세트를 추출하여 이후 단계에서 사용할 수 있습니다.

```

print(response['RecommendedRuleset'])

Rules = [
    RowCount between 2000 and 8000,
    IsComplete "col1",
    IsComplete "col2",
    StandardDeviation "col3" between 58138330.8 and 64258155.09,
    ColumnValues "col4" between 1000042965 and 1214474826,
    IsComplete "col5"
]

```

필터링 및 나열할 수 있는 모든 권장 실행 목록을 가져오려면:

```

response = client.list_data_quality_rule_recommendation_runs(
    Filter={
        'DataSource': {
            'GlueTable': {
                'DatabaseName': '<database_name>',
                'TableName': '<table_name>'
            }
        }
    }
)

```

기존 AWS Glue Data Quality 권장 작업을 취소하려면:

```

response = client.cancel_data_quality_rule_recommendation_run(
    RunId='dqrun-d4b6b01957fdd79e59866365bf9cb0e40fxxxxxxx'
)

```

AWS Glue Data Quality 규칙 세트 사용

AWS Glue Data Quality 규칙 세트를 생성하려면:

```
response = client.create_data_quality_ruleset(
    Name='<ruleset_name>',
    Ruleset='Rules = [IsComplete "col1", IsPrimaryKey "col2", RowCount between 2000 and
8000]',
    TargetTable={
        'TableName': '<table_name>',
        'DatabaseName': '<database_name>'
    }
)
```

데이터 품질 규칙 세트를 가져오려면:

```
response = client.get_data_quality_ruleset(
    Name='<ruleset_name>'
)
print(response)
```

이 API를 사용하여 규칙 세트를 추출할 수 있습니다.

```
print(response['Ruleset'])
```

테이블의 모든 데이터 품질 규칙 세트를 나열하려면:

```
response = client.list_data_quality_rulesets()
```

API 내의 필터 조건을 사용하여 특정 데이터베이스 또는 테이블에 연결된 모든 규칙 세트를 필터링할 수 있습니다.

```
response = client.list_data_quality_rulesets(
    Filter={
        'TargetTable': {
            'TableName': '<table_name>',
            'DatabaseName': '<database_name>'
        }
    },
)
```


데이터 품질 규칙 세트를 업데이트하려면:

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def update_data_quality_ruleset(self, ruleset_name, ruleset_string):
        """
        Update an AWS Glue Data Quality Ruleset

        :param ruleset_name: The name of the AWS Glue Data Quality ruleset to update
        :param ruleset_string: The DQDL ruleset string to update the ruleset with

        """
        try:
            response = self.client.update_data_quality_ruleset(
                Name=ruleset_name,
                Ruleset=ruleset_string
            )
        except ClientError as err:
            logger.error(
                "Couldn't update the AWS Glue Data Quality ruleset. Here's why: %s:
%s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response
```

데이터 품질 규칙 세트를 삭제하려면:

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def delete_data_quality_ruleset(self, ruleset_name):
```

```

"""
Delete a AWS Glue Data Quality Ruleset

:param ruleset_name: The name of the AWS Glue Data Quality ruleset to delete

"""
try:
    response = self.client.delete_data_quality_ruleset(
        Name=ruleset_name
    )
except ClientError as err:
    logger.error(
        "Couldn't delete the AWS Glue Data Quality ruleset. Here's why: %s:
%s",
        err.response['Error']['Code'], err.response['Error']['Message'])
    raise
else:
    return response

```

AWS Glue Data Quality 실행 사용

AWS Glue Data Quality 실행을 시작하려면:

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def start_data_quality_ruleset_evaluation_run(self, database_name, table_name,
role_name, ruleset_list):
        """
        Start an AWS Glue Data Quality evaluation run

        :param database_name: The name of the AWS Glue database which contains the
dataset.
        :param table_name: The name of the AWS Glue table against which we want to
evaluate.
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and Access
Management (IAM) role that grants permission to let AWS Glue access the resources it
needs.

```

:param ruleset_list: The list of AWS Glue Data Quality ruleset names to evaluate.

```

"""
try:
    response = client.start_data_quality_ruleset_evaluation_run(
        DataSource={
            'GlueTable': {
                'DatabaseName': database_name,
                'TableName': table_name
            }
        },
        Role=role_name,
        RulesetNames=ruleset_list
    )
except ClientError as err:
    logger.error(
        "Couldn't start the AWS Glue Data Quality Run. Here's why: %s: %s",
        err.response['Error']['Code'], err.response['Error']['Message'])
    raise
else:
    return response['RunId']

```

pushDownPredicate 또는 catalogPartitionPredicate 파라미터를 전달하여 카탈로그 테이블 내의 특정 파티션 세트만 대상으로 데이터 품질 실행을 수행할 수 있습니다. 예:

```

response = client.start_data_quality_ruleset_evaluation_run(
    DataSource={
        'GlueTable': {
            'DatabaseName': '<database_name>',
            'TableName': '<table_name>',
            'AdditionalOptions': {
                'pushDownPredicate': 'year=2023'
            }
        }
    },
    Role='<role_name>',
    NumberOfWorkers=5,
    Timeout=123,
    AdditionalRunOptions={
        'CloudWatchMetricsEnabled': False
    },
    RulesetNames=[

```

```

        '<ruleset_name>',
    ]
)

```

AWS Glue Data Quality 실행에 대한 정보를 가져오려면:

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def get_data_quality_ruleset_evaluation_run(self, run_id):
        """
        Get details about an AWS Glue Data Quality Run

        :param run_id: The AWS Glue Data Quality run ID to look up

        """
        try:
            response = self.client.get_data_quality_ruleset_evaluation_run(
                RunId=run_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't look up the AWS Glue Data Quality run ID. Here's why: %s:
%s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response

```

AWS Glue Data Quality 실행 결과를 가져오려면:

해당 AWS Glue Data Quality 실행에 대해 다음 방법을 사용하여 실행 평가 결과를 추출할 수 있습니다.

```

response = client.get_data_quality_ruleset_evaluation_run(
    RunId='d4b6b01957fdd79e59866365bf9cb0e40fxxxxxxx'
)

```

```

resultID = response['ResultIds'][0]

response = client.get_data_quality_result(
    ResultId=resultID
)

print(response['RuleResults'])

```

AWS Glue Data Quality 실행을 모두 나열하려면:

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def list_data_quality_ruleset_evaluation_runs(self, database_name, table_name):
        """
        Lists all the AWS Glue Data Quality runs against a given table

        :param database_name: The name of the database where the data quality runs
        :param table_name: The name of the table against which the data quality runs
        were created

        """
        try:
            response = self.client.list_data_quality_ruleset_evaluation_runs(
                Filter={
                    'DataSource': {
                        'GlueTable': {
                            'DatabaseName': database_name,
                            'TableName': table_name
                        }
                    }
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't list the AWS Glue Quality runs. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise

```

```

else:
    return response

```

특정 시간 사이의 결과 또는 특정 테이블에 대해 실행된 결과만 표시하도록 filter 절을 수정할 수 있습니다.

진행 중인 AWS Glue Data Quality 실행을 중지하려면:

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 AWS Glue client.
        """
        self.glue_client = glue_client

    def cancel_data_quality_ruleset_evaluation_run(self, result_id):
        """
        Cancels a given AWS Glue Data Quality run

        :param result_id: The result id of a AWS Glue Data Quality run to cancel

        """
        try:
            response = self.client.cancel_data_quality_ruleset_evaluation_run(
                ResultId=result_id
            )
        except ClientError as err:
            logger.error(
                "Couldn't cancel the AWS Glue Data Quality run. Here's why: %s: %s",
                err.response['Error']['Code'], err.response['Error']['Message'])
            raise
        else:
            return response

```

AWS Glue Data Quality 결과 사용

AWS Glue Data Quality 실행 결과를 가져오려면:

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""
    def __init__(self, glue_client):

```

```

"""
:param glue_client: A Boto3 AWS Glue client.
"""
self.glue_client = glue_client

def get_data_quality_result(self, result_id):
    """
    Outputs the result of an AWS Glue Data Quality Result

    :param result_id: The result id of an AWS Glue Data Quality run

    """
    try:
        response = self.client.get_data_quality_result(
            ResultId=result_id
        )
    except ClientError as err:
        logger.error(
            "Couldn't get the AWS Glue Data Quality result. Here's why: %s: %s",
            err.response['Error']['Code'], err.response['Error']['Message'])
        raise
    else:
        return response

```

기존 AWS Glue Data Quality 권장 작업을 취소하려면:

AWS Glue Data Quality 실행 ID가 제공되면 아래와 같이 결과 ID를 추출하여 실제 결과를 얻을 수 있습니다.

```

response = client.get_data_quality_ruleset_evaluation_run(
    RunId='dqrn-abca77ee126abe1378c1da1ae0750xxxxxxx'
)

resultID = response['ResultIds'][0]

response = client.get_data_quality_result(
    ResultId=resultID
)

print(resp['RuleResults'])

```

알림, 배포 및 예약 설정

이 주제에서는 AWS Glue Data Quality에 대한 알림, 배포 및 일정을 설정하는 방법을 설명합니다.

목차

- [Amazon EventBridge 통합에서 알림 및 알림 설정](#)
 - [이벤트 패턴의 추가 구성 옵션](#)
 - [알림을 이메일 형식으로 지정](#)
- [통합 시 알림 및 알림 설정 CloudWatch](#)
- [데이터 품질 결과를 쿼리하여 대시보드 구축](#)
- [다음을 사용하여 데이터 품질 규칙을 배포합니다. AWS CloudFormation](#)
- [데이터 품질 규칙 예약](#)

Amazon EventBridge 통합에서 알림 및 알림 설정

AWS Glue Data Quality는 데이터 품질 규칙 세트 평가 실행 완료 시 발생하는 EventBridge 이벤트 게시를 지원합니다. 이를 통해 데이터 품질 규칙 실패 시 알림을 쉽게 설정할 수 있습니다.

다음은 데이터 카탈로그에서 데이터 품질 규칙 세트를 평가할 때 발생하는 샘플 이벤트입니다. 이 정보를 사용하여 Amazon에서 제공하는 데이터를 검토할 수 EventBridge 있습니다. 추가 API 직접 호출을 통해 자세한 내용을 확인할 수 있습니다. 예를 들어 결과 ID로 `get_data_quality_result` API를 직접 호출하여 특정 실행의 세부 정보를 가져옵니다.

```
{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
  "detail-type": "Data Quality Evaluation Results Available",
  "source": "aws.glue-dataquality",
  "account": "123456789012",
  "time": "2017-09-07T18:57:21Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "context": {
      "contextType": "GLUE_DATA_CATALOG",
      "runId": "dqrn-12334567890",
      "databaseName": "db-123",
      "tableName": "table-123",
    }
  }
}
```



```

        "catalogId": "123456789012"
      },
      "resultID": "dqresult-12334567890",
      "rulesetNames": ["rulset1"],
      "state": "SUCCEEDED",
      "score": 1.00,
      "rulesSucceeded": 100,
      "rulesFailed": 0,
      "rulesSkipped": 0
    }
  }
}

```

다음은 Glue ETL 또는 AWS Glue Studio 노트북에서 데이터 품질 규칙 세트를 평가할 때 게시되는 샘플 이벤트입니다. AWS

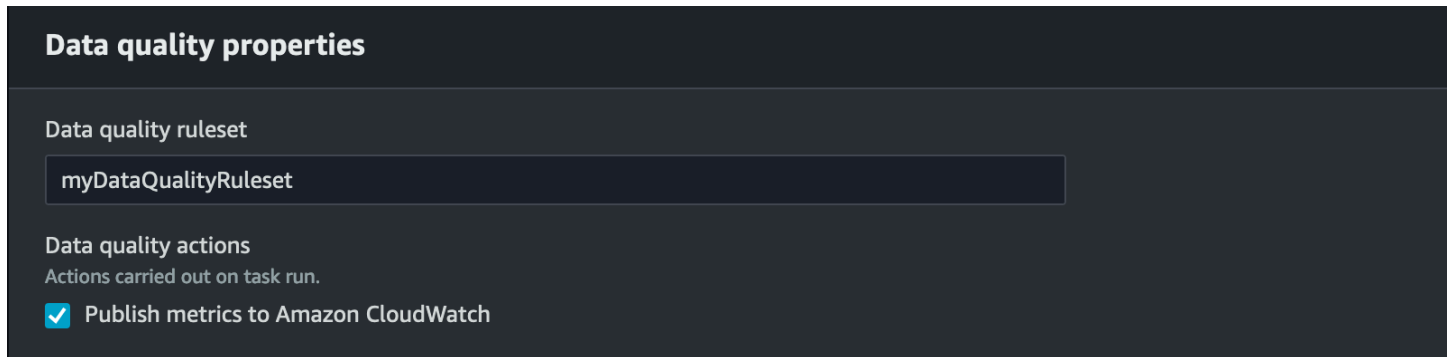
```

{
  "version": "0",
  "id": "abcdef00-1234-5678-9abc-def012345678",
  "detail-type": "Data Quality Evaluation Results Available",
  "source": "aws.glue-dataquality",
  "account": "123456789012",
  "time": "2017-09-07T18:57:21Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "context": {
      "contextType": "GLUE_JOB",
      "jobId": "jr-12334567890",
      "jobName": "dq-eval-job-1234",
      "evaluationContext": ""
    }
    "resultID": "dqresult-12334567890",
    "rulesetNames": ["rulset1"],
    "state": "SUCCEEDED",
    "score": 1.00
    "rulesSucceeded": 100,
    "rulesFailed": 0,
    "rulesSkipped": 0
  }
}

```

데이터 카탈로그와 ETL 작업 모두에서 데이터 품질 평가를 실행하려면 기본적으로 선택되는 메트릭 게시 대상 Amazon CloudWatch 옵션을 선택한 상태로 유지해야 게시가 제대로 작동합니다. EventBridge

알림 설정 EventBridge



생성된 이벤트를 수신하고 대상을 정의하려면 Amazon EventBridge 규칙을 구성해야 합니다. 규칙을 생성하려면:

1. Amazon EventBridge 콘솔을 엽니다.
2. 탐색 표시줄의 버스 섹션에서 규칙을 선택합니다.
3. [Create Rule]을 선택합니다.
4. 규칙 세부 정보 정의에서:
 - a. 이름에 myDQRu1e을 입력합니다.
 - b. 설명을 입력합니다(선택 사항).
 - c. 이벤트 버스의 경우 이벤트 버스를 선택합니다. 이벤트 버스가 아직 없는 경우 기본값으로 둡니다.
 - d. 규칙 유형에서 이벤트 패턴이 있는 규칙을 선택하고 다음을 선택합니다.
5. 이벤트 패턴 작성에서:
 - a. 이벤트 소스의 경우 AWS 이벤트 또는 EventBridge 파트너 이벤트를 선택하십시오.
 - b. 샘플 이벤트 섹션은 건너뜁니다.
 - c. 생성 방법으로 패턴 양식 사용을 선택합니다.
 - d. 이벤트 패턴의 경우:
 - i. 이벤트 소스에 대한 AWS 서비스를 선택합니다.
 - ii. AWS 서비스를 받으려면 Glue 데이터 품질을 선택합니다.
 - iii. 이벤트 유형에 대해 사용할 수 있는 데이터 품질 평가 결과를 선택합니다.
 - iv. 특정 상태에 대해 실패를 선택합니다. 그러면 다음과 비슷한 이벤트 패턴이 나타납니다.

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "state": ["FAILED"]
  }
}
```

v. 더 많은 구성 옵션은 [이벤트 패턴의 추가 구성 옵션](#) 섹션을 참조하세요.

6. 대상 선택에서:

- a. 대상 유형에 대해 AWS 서비스를 선택합니다.
- b. 대상 선택 드롭다운을 사용하여 연결하려는 AWS 서비스 (SNS, Lambda, SQS 등) 를 선택하고 다음을 선택합니다.

7. 태그 구성에서 새 태그 추가를 클릭하여 선택적 태그를 추가하고 다음을 선택합니다.

8. 모든 선택 항목의 요약 페이지가 표시됩니다. 하단에서 규칙 생성을 선택합니다.

이벤트 패턴의 추가 구성 옵션

성공 또는 실패 시 이벤트 필터링 외에도 다양한 파라미터에서 이벤트를 추가로 필터링할 수 있습니다.

이렇게 하려면 이벤트 패턴 섹션으로 이동한 다음 패턴 편집을 선택하여 추가 파라미터를 지정합니다. 이벤트 패턴의 필드는 대소문자를 구분합니다. 다음은 이벤트 패턴 구성에 대한 예제입니다.

특정 규칙 세트를 평가하는 특정 테이블의 이벤트를 캡처하려면 다음 유형의 패턴을 사용합니다.

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "context": {
      "contextType": ["GLUE_DATA_CATALOG"],
      "databaseName": "db-123",
      "tableName": "table-123",
    },
    "rulesetNames": ["ruleset1", "ruleset2"]
  },
  "state": ["FAILED"]
}
```

ETL 환경에서 특정 작업의 이벤트를 캡처하려면 다음 유형의 패턴을 사용합니다.

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "context": {
      "contextType": ["GLUE_JOB"],
      "jobName": ["dq_evaluation_job1", "dq_evaluation_job2"]
    },
    "state": ["FAILED"]
  }
}
```

점수가 특정 임계값(예: 70%) 미만인 이벤트를 캡처하려면:

```
{
  "source": ["aws.glue-dataquality"],
  "detail-type": ["Data Quality Evaluation Results Available"],
  "detail": {
    "score": [{
      "numeric": ["<=", 0.7]
    }]
  }
}
```

알림을 이메일 형식으로 지정

비즈니스 팀에 올바른 형식의 이메일 알림을 보내야 하는 경우가 있습니다. EventBridge Amazon과 AWS Lambda를 사용하여 이를 달성할 수 있습니다.

Glue Data Quality rulesets **Glue_DQ_RULESET_CUSTOM_20de29c13537** run details



AWS Notifications <no-reply@sns.amazonaws.com>

Thursday, 11. May 2023 at 15:01

To: [REDACTED]

Glue Data Quality run details:

```
ruleset_name:  Glue_DQ_RULESET_CUSTOM_20de29c13537
glue_table_name:  devprod_tbl_nxc_taxi_data
glue_database_name:  devprod_db_nyc_taxi_data
run_id:  dqrun-066b41002a56921f9163a4e9156a4f6e20ce47a8
result_id:  dqresult-cd03a2e91c9114b611f6f79363b2288133fc96c0
state:  FAILED
score:  0.5
numRulesSucceeded: 1
numRulesFailed: 1
numRulesSkipped: 0
```

The subject of the email contains the name of the ruleset

Body of email with statistics from the Glue Data Quality Ruleset.

Here are the results of the ruleset evaluation steps

ruleset details evaluation steps results:

Name: Rule_1	Result: PASS	Description: IsComplete "vendorid"	
Name: Rule_2	Result: FAIL	EvaluationMessage: Value: 0.0 does not meet the constraint requirement!	Description: IsPrimaryKey "vendorid"

--

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:

[REDACTED] >[https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:728060703200:SNStandardGlueDataQualityBlogAlertNotification:9d82097d-06f6-4c11-951a-3c0e1d9748f2&Endpoint=\[REDACTED\]](https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:728060703200:SNStandardGlueDataQualityBlogAlertNotification:9d82097d-06f6-4c11-951a-3c0e1d9748f2&Endpoint=[REDACTED])

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

다음 샘플 코드를 사용하여 데이터 품질 알림 형식을 지정하여 이메일을 생성할 수 있습니다.

```
import boto3
import json
from datetime import datetime

sns_client = boto3.client('sns')
glue_client = boto3.client('glue')

sns_topic_arn = 'arn:aws:sns:<region-code>:<account-id>:<sns-topic-name>'

def lambda_handler(event, context):
    log_metadata = {}
    message_text = ""
```

```

subject_text = ""

if event['detail']['context']['contextType'] == 'GLUE_DATA_CATALOG':
    log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
    log_metadata['tableName'] = str(event['detail']['context']['tableName'])
    log_metadata['databaseName'] = str(event['detail']['context']['databaseName'])
    log_metadata['runId'] = str(event['detail']['context']['runId'])
    log_metadata['resultId'] = str(event['detail']['resultId'])
    log_metadata['state'] = str(event['detail']['state'])
    log_metadata['score'] = str(event['detail']['score'])
    log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
    log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
    log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

    message_text += "Glue Data Quality run details:\n"
    message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
    message_text += "glue_table_name: {}\n".format(log_metadata['tableName'])
    message_text += "glue_database_name: {}\n".format(log_metadata['databaseName'])
    message_text += "run_id: {}\n".format(log_metadata['runId'])
    message_text += "result_id: {}\n".format(log_metadata['resultId'])
    message_text += "state: {}\n".format(log_metadata['state'])
    message_text += "score: {}\n".format(log_metadata['score'])
    message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
    message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
    message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

    subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

else:
    log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
    log_metadata['jobName'] = str(event['detail']['context']['jobName'])
    log_metadata['jobId'] = str(event['detail']['context']['jobId'])
    log_metadata['resultId'] = str(event['detail']['resultId'])
    log_metadata['state'] = str(event['detail']['state'])
    log_metadata['score'] = str(event['detail']['score'])

    log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
    log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
    log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

    message_text += "Glue Data Quality run details:\n"
    message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])

```

```

message_text += "glue_job_name: {}\n".format(log_metadata['jobName'])
message_text += "job_id: {}\n".format(log_metadata['jobId'])
message_text += "result_id: {}\n".format(log_metadata['resultId'])
message_text += "state: {}\n".format(log_metadata['state'])
message_text += "score: {}\n".format(log_metadata['score'])
message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
message_text += "numRulesFailed: {}\n".format(log_metadata['numRulesFailed'])
message_text += "numRulesSkipped: {}\n".format(log_metadata['numRulesSkipped'])

subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

resultID = str(event['detail']['resultId'])
response = glue_client.get_data_quality_result(ResultId=resultID)
RuleResults = response['RuleResults']
message_text += "\n\nruleset details evaluation steps results:\n\n"
subresult_info = []

for dic in RuleResults:
    subresult = "Name: {}\t\tResult: {}\t\tDescription: \t{}".format(dic['Name'],
dic['Result'], dic['Description'])
    if 'EvaluationMessage' in dic:
        subresult += "\t\tEvaluationMessage: {}".format(dic['EvaluationMessage'])
    subresult_info.append({
        'Name': dic['Name'],
        'Result': dic['Result'],
        'Description': dic['Description'],
        'EvaluationMessage': dic.get('EvaluationMessage', '')
    })
    message_text += "\n" + subresult

log_metadata['resultrun'] = subresult_info

sns_client.publish(
    TopicArn=sns_topic_arn,
    Message=message_text,
    Subject=subject_text
)

return {
    'statusCode': 200,

```

```
'body': json.dumps('Message published to SNS topic')
}
```

통합 시 알림 및 알림 설정 CloudWatch

EventBridgeAmazon에서 고객에게 알림을 보내려면 일회성 설정이 EventBridge 필요하므로 Amazon을 사용하여 데이터 품질 알림을 설정하는 것이 좋습니다. 그러나 일부 고객은 익숙함 CloudWatch 때문에 Amazon을 선호합니다. 이러한 고객을 위해 Amazon과의 통합 서비스를 제공합니 다 CloudWatch.

각 AWS Glue Data Quality 평가는 데이터 품질 실행당 `glue.data.quality.rules.passed` (통과 한 규칙의 수를 나타냄) 및 `glue.data.quality.rules.failed` (실패한 규칙의 수를 나타냄) 라는 이름의 지표 쌍을 내보냅니다. 이 생성된 지표를 사용하여 해당 데이터 품질 실행이 임계값 아래로 떨어질 경우 사용자에게 알리는 경보를 생성할 수 있습니다. Amazon SNS 알림을 통해 이메일을 보내는 경보 설정을 시작하려면 아래 단계를 수행합니다.

Amazon SNS 알림을 통해 이메일을 보내는 경보 설정을 시작하려면 아래 단계를 수행합니다.

1. Amazon CloudWatch 콘솔을 엽니다.
2. 지표 아래에서 모든 지표를 선택합니다. Glue Data Quality라는 사용자 지정 네임스페이스 아래에 추가 네임스페이스가 표시됩니다.

Note

AWS Glue Data Quality 실행을 시작할 때 Amazon에 지표 게시 CloudWatch 확인란이 활성화되어 있는지 확인하십시오. 그렇지 않으면 특정 실행에 대한 지표가 Amazon에 게시되지 않습니다 CloudWatch.

Glue Data Quality 네임스페이스 아래에서 테이블 및 규칙 세트별로 지표가 생성되었음을 알 수 있습니다. 이 주제의 목적에 따라 이 값이 1을 초과하는 경우 `glue.data.quality.rules.failed` 규칙 및 경보를 사용합니다(즉, 실패한 규칙 평가 수가 1보다 크면 알림을 제공하려고 함).

3. 경보를 생성하려면 경보 아래에서 모든 경보를 선택합니다.
4. 경보 생성을 선택하세요.
5. 지표 선택을 선택하세요.

6. 생성한 테이블에 해당하는 `glue.data.quality.rules.failed` 지표를 선택한 다음 지표 선택을 선택합니다.

7. 지표 및 조건 지정 탭의 지표 섹션 아래에서:

- a. Statistic(통계)에서 Sum(합계)를 선택합니다.
- b. 기간에서 1분을 선택합니다.

8. 조건 섹션에서:

- a. 임계값 유형(Threshold type)에서 정적(Static)을 선택합니다.
- b. `glue.data.quality.rules.failed` 조건이 다음과 같을 때마다...에서 이상을 선택합니다.
- c. 기준...에서 임계값으로 1을 입력합니다.

이렇게 선택하면 `glue.data.quality.rules.failed` 지표가 1 이상의 값을 생성하는 경우 경보가 트리거됩니다. 하지만 데이터가 없으면 허용 가능한 값으로 간주합니다.

9. 다음을 선택합니다.

10.작업 구성에서:

- a. 경보 상태 트리거에서 경보 내를 선택합니다.
- b. 다음 SNS 주제로 알림 보내기 섹션에서 새 주제를 생성하여 새 SNS 주제를 통해 알림 보내기를 선택합니다.
- c. 알림을 수신할 이메일 엔드포인트에 이메일 주소를 입력합니다. 그리고 주제 생성을 클릭합니다.
- d. 다음을 선택합니다.

11.경보 이름에 `myFirstDQAlarm`을 입력한 후 다음을 선택합니다.

12.모든 선택 항목의 요약 페이지가 표시됩니다. 하단에서 경보 생성을 선택합니다.

이제 Amazon CloudWatch 경보 대시보드에서 경보가 생성되는 것을 볼 수 있습니다.

데이터 품질 결과를 쿼리하여 대시보드 구축

대시보드를 구축하여 데이터 품질 결과를 표시할 수 있습니다. 이렇게 하는 방법은 두 가지입니다.

다음 코드를 EventBridge 사용하여 Amazon을 설정하여 Amazon S3에 데이터를 기록합니다.

```
import boto3
import json
from datetime import datetime

s3_client = boto3.client('s3')
```

```
glue_client = boto3.client('glue')

s3_bucket = 's3-bucket-name'

def write_logs(log_metadata):
    try:
        filename = datetime.now().strftime("%m%d%Y%H%M%S") + ".json"
        key_opts = {
            'year': datetime.now().year,
            'month': "{:02d}".format(datetime.now().month),
            'day': "{:02d}".format(datetime.now().day),
            'filename': filename
        }
        s3key = "gluedataqualitylogs/year={year}/month={month}/day={day}/
{filename}".format(**key_opts)
        s3_client.put_object(Bucket=s3_bucket, Key=s3key,
Body=json.dumps(log_metadata))
    except Exception as e:
        print(f'Error writing logs to S3: {e}')

def lambda_handler(event, context):
    log_metadata = {}
    message_text = ""
    subject_text = ""

    if event['detail']['context']['contextType'] == 'GLUE_DATA_CATALOG':
        log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
        log_metadata['tableName'] = str(event['detail']['context']['tableName'])
        log_metadata['databaseName'] = str(event['detail']['context']['databaseName'])
        log_metadata['runId'] = str(event['detail']['context']['runId'])
        log_metadata['resultId'] = str(event['detail']['resultId'])
        log_metadata['state'] = str(event['detail']['state'])
        log_metadata['score'] = str(event['detail']['score'])
        log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
        log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
        log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

    message_text += "Glue Data Quality run details:\n"
    message_text += "ruleset_name: {}\n".format(log_metadata['ruleset_name'])
    message_text += "glue_table_name: {}\n".format(log_metadata['tableName'])
    message_text += "glue_database_name: {}\n".format(log_metadata['databaseName'])
    message_text += "run_id: {}\n".format(log_metadata['runId'])
```

```

message_text += "result_id: {}".format(log_metadata['resultId'])
message_text += "state: {}".format(log_metadata['state'])
message_text += "score: {}".format(log_metadata['score'])
message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
message_text += "numRulesFailed: {}".format(log_metadata['numRulesFailed'])
message_text += "numRulesSkipped: {}".format(log_metadata['numRulesSkipped'])

subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

else:
    log_metadata['ruleset_name'] = str(event['detail']['rulesetNames'][0])
    log_metadata['jobName'] = str(event['detail']['context']['jobName'])
    log_metadata['jobId'] = str(event['detail']['context']['jobId'])
    log_metadata['resultId'] = str(event['detail']['resultId'])
    log_metadata['state'] = str(event['detail']['state'])
    log_metadata['score'] = str(event['detail']['score'])

    log_metadata['numRulesSucceeded'] = str(event['detail']['numRulesSucceeded'])
    log_metadata['numRulesFailed'] = str(event['detail']['numRulesFailed'])
    log_metadata['numRulesSkipped'] = str(event['detail']['numRulesSkipped'])

    message_text += "Glue Data Quality run details:\n"
    message_text += "ruleset_name: {}".format(log_metadata['ruleset_name'])
    message_text += "glue_job_name: {}".format(log_metadata['jobName'])
    message_text += "job_id: {}".format(log_metadata['jobId'])
    message_text += "result_id: {}".format(log_metadata['resultId'])
    message_text += "state: {}".format(log_metadata['state'])
    message_text += "score: {}".format(log_metadata['score'])
    message_text += "numRulesSucceeded:
{}\n".format(log_metadata['numRulesSucceeded'])
    message_text += "numRulesFailed: {}".format(log_metadata['numRulesFailed'])
    message_text += "numRulesSkipped: {}".format(log_metadata['numRulesSkipped'])

    subject_text = "Glue Data Quality ruleset {} run
details".format(log_metadata['ruleset_name'])

    resultID = str(event['detail']['resultId'])
    response = glue_client.get_data_quality_result(ResultId=resultID)
    RuleResults = response['RuleResults']
    message_text += "\n\nruleset details evaluation steps results:\n\n"
    subresult_info = []

```

```

for dic in RuleResults:
    subresult = "Name: {}\t\tResult: {}\t\tDescription: \t{}".format(dic['Name'],
dic['Result'], dic['Description'])
    if 'EvaluationMessage' in dic:
        subresult += "\t\tEvaluationMessage: {}".format(dic['EvaluationMessage'])
    subresult_info.append({
        'Name': dic['Name'],
        'Result': dic['Result'],
        'Description': dic['Description'],
        'EvaluationMessage': dic.get('EvaluationMessage', '')
    })
    message_text += "\n" + subresult

log_metadata['resultrun'] = subresult_info

write_logs(log_metadata)

return {
    'statusCode': 200,
    'body': json.dumps('Message published to SNS topic')
}

```

Amazon S3에 작성한 후 AWS Glue 크롤러를 사용하여 Athena에 등록하고 테이블을 쿼리할 수 있습니다.

데이터 품질 평가 중에 Amazon S3 위치 구성:

Glue 데이터 카탈로그 또는 AWS Glue ETL에서 데이터 품질 작업을 실행할 때 Amazon S3 위치를 제공하여 데이터 품질 결과를 Amazon S3에 기록할 수 있습니다. 아래 구문을 통해 데이터 품질 결과를 읽기 위해 대상을 참조하여 테이블을 생성할 수 있습니다.

CREATE EXTERNAL TABLE 및 MSCK REPAIR TABLE 쿼리는 별도로 실행해야 합니다.

```

CREATE EXTERNAL TABLE <my_table_name>(
    catalogid string,
    databasename string,
    tablename string,
    dqrunid string,
    evaluationstartedon timestamp,
    evaluationcompletedon timestamp,
    rule string,

```

```

    outcome string,
    failurereason string,
    evaluatedmetrics string)
PARTITIONED BY (
    `year` string,
    `month` string,
    `day` string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (

    'paths'='catalogId,databaseName,dqRunId,evaluatedMetrics,evaluationCompletedOn,evaluationStart
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://glue-s3-dq-bucket-us-east-2-results/'
TBLPROPERTIES (
    'classification'='json',
    'compressionType'='none',
    'typeOfData'='file');

```

```
MSCK REPAIR TABLE <my_table_name>;
```

위 테이블을 생성한 후에 Amazon Athena를 사용하여 분석 쿼리를 실행할 수 있습니다.

다음을 사용하여 데이터 품질 규칙을 배포합니다. AWS CloudFormation

를 AWS CloudFormation 사용하여 데이터 품질 규칙을 만들 수 있습니다. 자세한 내용은 [AWS CloudFormation AWS Glue](#)를 참조하십시오.

데이터 품질 규칙 예약

다음 방법을 사용하여 데이터 품질 규칙을 예약할 수 있습니다.

- 데이터 카탈로그에서 데이터 품질 규칙을 예약하세요. 어떤 코드에서도 사용자가 이 옵션을 사용하여 데이터 품질 검사를 쉽게 예약할 수 없습니다. AWS Glue Data Quality는 아마존에서 일정을 생성합니다 EventBridge. 데이터 품질 규칙을 예약하려면:
 - 규칙 세트로 이동한 다음 실행을 클릭합니다.
 - 실행 빈도에서 원하는 예약을 선택하고 작업 이름을 제공합니다. 이 작업 이름은 내 일정의 이름입니다 EventBridge.
- Amazon EventBridge 및 AWS Step Functions를 사용하여 데이터 품질 규칙에 대한 평가 및 권장 사항을 조정하십시오.

AWS Glue 데이터 품질 오류 문제 해결

AWS Glue Data Quality에서 오류가 발생하는 경우 다음 해결 방법을 사용하여 문제의 원인을 찾아 해결하십시오.

목차

- [오류: AWS Glue 데이터 품질 모듈이 없습니다.](#)
- [오류: AWS Lake Formation 권한이 충분하지 않습니다.](#)
- [오류: 규칙 세트의 이름이 고유하지 않음](#)
- [오류: 특수 문자가 있는 테이블](#)
- [오류: 큰 규칙 세트로 인한 오버플로 오류](#)
- [오류: 전체 규칙 상태에서 실패임](#)
- [AnalysisException: 기본 데이터베이스가 있는지 확인할 수 없습니다.](#)
- [오류 메시지: Provided key map not suitable for given data frames](#)
- [사용자 클래스에서의 예외: java.lang RuntimeException : 데이터를 가져오지 못했습니다. 로그인을 확인하여 자세한 내용을 확인하세요. CloudWatch](#)
- [시작 오류: 버킷에 대해 S3에서 다운로드하는 중 오류 발생](#)
- [InvalidInputException \(상태: 400\): DataQuality 규칙을 파싱할 수 없습니다.](#)
- [오류: Eventbridge에서 사용자가 설정한 일정에 따라 Glue DQ 작업을 트리거하지 않습니다.](#)
- [CustomSQL 오류](#)
- [동적 규칙](#)
- [사용자 클래스에서의 예외: org.apache.spark.sql AnalysisException: org.apache.hadoop.hive.ql.metadata. HiveException](#)
- [UNCLASSIFIED_ERROR; IllegalArgumentException: 파싱 오류: 규칙이나 분석기가 제공되지 않았습니다. 입력 시 실행 가능한 대안이 없습니다.](#)

오류: AWS Glue 데이터 품질 모듈이 없습니다.

오류 메시지: No module named 'awsgluedq'.

해결: 이 오류는 지원되지 않는 버전에서 AWS Glue Data Quality를 실행할 때 발생합니다. AWS Glue 데이터 품질은 Glue 버전 3.0 이상에서만 지원됩니다.

오류: AWS Lake Formation 권한이 충분하지 않습니다.

오류 메시지: 사용자 클래스의 예외:: impact_sdg_incluvement에 대한 Lake Formation 권한이 충분하지 않습니다 (서비스com.amazonaws.services.glue.model.AccessDeniedException:AWS Glue; 상태 코드: 400; 오류 코드: AccessDeniedException; 요청 ID: 465ae693-b7ba-4df0-a4e4-6b17xxxxxx, 프록시: null).

해결 방법: AWS Lake Formation에서 충분한 권한을 제공해야 합니다.

오류: 규칙 세트의 이름이 고유하지 않음

오류 메시지: 사용자 클래스의 예외:... services.glue.model.AlreadyExistsException: 같은 이름을 가진 다른 규칙 세트가 이미 있습니다.

해결 방법: 규칙 세트는 글로벌 항목이며 고유해야 합니다.

오류: 특수 문자가 있는 테이블

오류 메시지: 사용자 클래스의 예외: org.apache.spark.sql AnalysisException: 주어진 입력 열에 “C”를 해결할 수 없음: [기본.데이터_종료시간, 기본.데이터_시작_시간, 기본.종료_시간, 기본.마지막_업데이트, 기본.메시지, 기본.process_date, 기본.rowhash, 기본.run_id, 기본.시작_시간, 기본.status], 1줄 pos 44;.

해결 방법: 현재 “.” 같은 특수 문자가 있는 테이블에서는 AWS Glue Data Quality를 실행할 수 없다는 제한이 있습니다.

오류: 큰 규칙 세트에 의한 오버플로 오류

오류 메시지: 사용자 클래스의 예외: java.lang StackOverflowError.

해결 방법: 규칙 개수가 2,000개가 넘는 큰 규칙 세트가 있는 경우 이 문제가 발생할 수 있습니다. 규칙을 여러 규칙 세트로 구분합니다.

오류: 전체 규칙 상태에서 실패임

오류 조건: 내 규칙 세트는 성공했지만 전체 규칙 상태는 실패입니다.

해결 방법: 이 오류는 게시하는 CloudWatch 동안 Amazon에 지표를 게시하는 옵션을 선택했기 때문에 발생했을 가능성이 큼니다. 데이터세트가 VPC에 있는 경우, VPC는 AWS Glue가 Amazon에 지표를 게시하는 것을 허용하지 않을 수 있습니다. CloudWatch 이 경우 VPC가 Amazon에 액세스할 수 있도록 엔드포인트를 설정해야 합니다. CloudWatch

AnalysisException: 기본 데이터베이스가 있는지 확인할 수 없습니다.

오류 상태: AnalysisException: 기본 데이터베이스의 존재를 확인할 수 없습니다:

com.amazonaws.services.glue.model.AccessDeniedException: 기본적으로 Lake Formation 권한이 충분하지 않음 (서비스:AWS Glue; 상태 코드: 400, 오류 코드: AccessDeniedException; 요청 ID: XXXXXX-XXXX-XXXX-XXXX-XXXXXXXXX, 프록시: null)

해결 방법: AWS Glue 작업의 카탈로그 통합에서 AWS Glue는 항상 기본 데이터베이스의 존재 여부와 AWS Glue GetDatabase API의 사용 여부를 확인하려고 합니다. DESCRIBE Lake Formation 권한이 부여되지 않거나 GetDatabase IAM 권한이 부여되면 기본 데이터베이스의 존재를 확인할 때 작업에 실패합니다.

이 문제를 해결하려면:

1. Lake Formation에서 기본 데이터베이스에 대한 DESCRIBE 권한을 추가합니다.
2. Lake Formation에서 데이터베이스 생성자로 AWS Glue 작업에 연결된 IAM 역할을 구성합니다. 그러면 자동으로 기본 데이터베이스가 생성되고 역할에 필요한 Lake Formation 권한이 부여됩니다.
3. --enable-data-catalog 옵션을 비활성화합니다. (이는 AWS Glue Studio에서 Hive 메타스토어로 Data Catalog 사용으로 표시됩니다.)

작업에 Spark SQL Data Catalog 통합이 필요하지 않은 경우 이를 비활성화할 수 있습니다.

오류 메시지: Provided key map not suitable for given data frames

오류 조건: 제공된 키 맵이 해당 데이터 프레임에 적합하지 않습니다.

해결 방법: 규칙 유형을 사용하고 있는데 조인 키가 중복되었습니다. DataSetMatch 조인 키는 고유해야 하며 NULL이 아니어야 합니다. 고유한 조인 키를 사용할 수 없는 경우에는 요약 데이터를 기준으로 매칭하는 것과 같은 다른 규칙 유형을 사용하는 AggregateMatch 것이 좋습니다.

사용자 클래스에서의 예외: java.lang RuntimeException : 데이터를 가져오지 못했습니다. 로그인을 확인하여 자세한 내용을 확인하세요. CloudWatch

오류 상태: 사용자 클래스의 예외: java.lang. RuntimeException : 데이터를 가져오지 못했습니다. 로그인을 확인하여 자세한 내용을 확인하세요. CloudWatch

해결 방법: 이 문제는 Amazon S3 기반 테이블에서 Amazon RDS 또는 와 비교되는 DQ 규칙을 생성할 때 발생합니다. Amazon Redshift이 경우에는 AWS Glue에서 연결을 로드할 수 없습니다. 대신,

Amazon Redshift 또는 Amazon RDS 데이터 세트에 DQ 규칙을 설정해 보십시오. 이는 알려진 버그입니다.

시작 오류: 버킷에 대해 S3에서 다운로드하는 중 오류 발생

오류 조건: 시작 오류: 버킷에 대해 S3에서 다운로드하는 중 오류 발생: aws-glue-ml-data-quality-assets-us-east-1, key: jars/aws-glue-ml-data-quality-etl.jar.Access Denied (Service: Amazon S3; Status Code: 403; Please refer logs for details) .

해결 방법: AWS Glue Data Quality에 전달된 역할의 권한은 이전 Amazon S3 위치에서의 읽기를 허용해야 합니다. 이 IAM 정책은 다음 역할에 연결되어야 합니다.

```
{
  "Sid": "allowS3",
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::aws-glue-ml-data-quality-assets-<region>/*"
}
```

자세한 권한은 [데이터 품질 권한 부여](#)를 참조하세요. 이 라이브러리는 데이터 세트의 데이터 품질을 평가하는 데 필요합니다.

InvalidInputException (상태: 400): DataQuality 규칙을 파싱할 수 없습니다.

오류 조건: InvalidInputException (상태: 400): DataQuality 규칙을 파싱할 수 없습니다.

해결 방법: 이 오류에는 여러 가지 가능성이 있습니다. 한 가지 가능성으로, 규칙에 작은따옴표가 포함되었을 수 있습니다. 큰따옴표로 묶여 있는지 확인합니다. 예:

```
Rules = [
  ColumnValues "tipo_vinculo" in ["COD0", "DOC0", "COC0", "DOD0"] AND "categoria" = 'ES'
  AND "cod_bandera" = 'CEP'
```

이를 다음과 같이 변경합니다.

```
Rules = [
  (ColumnValues "tipovinculo" in [ "COD0", "DOC0", "COC0", "DOD0"]) AND (ColumnValues
  "categoria" = "ES")
```

```
AND (ColumnValues "codbandera" = "CEP")
]
```

오류: Eventbridge에서 사용자가 설정한 일정에 따라 Glue DQ 작업을 트리거하지 않습니다.

오류 조건: Eventbridge에서 사용자가 설정한 일정에 따라 AWS Glue Data Quality 작업을 트리거하지 않습니다.

해결 방법: 작업을 트리거하는 역할에 적절한 권한이 없을 수 있습니다. 작업을 시작하는 데 사용하는 역할에 [평가 실행 예약에 필요한 IAM 설정](#)에서 언급한 권한이 있는지 확인합니다.

CustomSQL 오류

오류 조건: The output from CustomSQL must contain at least one column that matches the input dataset for AWS Glue Data Quality to provide row level results. The SQL query is a valid query but no columns from the SQL result are present in the Input Dataset. Ensure that matching columns are returned from the SQL.

해결 방법: SQL 쿼리가 유효한 경우 기본 테이블에서 열만 선택했는지 확인합니다. 기본 테이블의 열에서 sum, count 등의 집계 함수를 선택하면 이 오류가 발생할 수 있습니다.

오류 조건: There was a problem when executing your SQL statement: cannot resolve "Col".

해결 방법: 이 열은 기본 테이블에 없습니다.

오류 조건: The columns that are returned from the SQL statement should only belong to the primary table. "In this case, some columns (Col) belong to reference table".

해결 방법: SQL 쿼리에서 기본 테이블을 다른 참조 테이블과 조인할 때 기본 테이블에 대해 행 수준 결과를 생성하려면 select 문에 기본 테이블의 열 이름만 있는지 확인합니다.

동적 규칙

오류 조건: Dynamic rules require job context, and cannot be evaluated in interactive session or data preview..

원인: 해당 오류 메시지는 규칙 세트에 동적 DQ 규칙이 있는 경우 데이터 미리 보기 결과 또는 다른 대화형 세션에 나타날 수 있습니다. 동적 규칙은 특정 작업 이름 및 평가 컨텍스트와 관련된 기록 지표를 참조하므로 대화형 세션에서는 평가할 수 없습니다.

해결 방법: AWS Glue 작업을 실행하면 이전 지표가 생성되며 이후 동일한 작업에 대한 작업 실행 시 참조할 수 있습니다.

오류 조건:

- [RuleType] rule only supports simple atomic operands in thresholds..
- Function last not yet implemented for [RuleType] rule.

해결 방법: 동적 규칙은 일반적으로 숫자 표현식의 모든 DQDL 규칙 유형에 대해 지원됩니다([데이터 품질 정의 언어\(DQDL\) 참조](#) 참조). 하지만 여러 메트릭을 생성하는 일부 규칙 (ColumnValues 및 ColumnLength) 은 아직 지원되지 않습니다.

오류 조건: Binary expression operands must resolve to a single number..

원인: 동적 규칙은 $\text{RowCount} > \text{avg}(\text{last}(5)) * 0.9$ 와 같은 이진 표현식을 지원합니다. 여기서 이진 표현식은 $\text{avg}(\text{last}(5)) * 0.9$ 입니다. 이 규칙은 피연산자 $\text{avg}(\text{last}(5))$ 및 0.9 가 모두 단일 숫자로 해석되므로 유효합니다. $\text{RowCount} > \text{last}(5) * 0.9$ 는 잘못된 예입니다. $\text{last}(5)$ 는 현재 행 수와 유의미하게 비교할 수 없는 목록을 생성하기 때문입니다.

해결 방법: 집계 함수를 사용하여 목록 값 피연산자를 단일 숫자로 줄입니다.

오류 조건:

- Rule threshold results in list, and a single value is expected.
Use aggregation functions to produce a single value. Valid example:
`sum(last(10)), avg(last(10))`.
- Rule threshold results in empty list, and a single value is expected.

원인: 동적 규칙을 사용하여 데이터 세트의 일부 기능을 기록 값과 비교할 수 있습니다. 마지막 함수를 사용하는 경우 양의 정수 인수가 제공되면 여러 기록 값을 검색할 수 있습니다. 예를 들어 $\text{last}(5)$ 는 규칙에 대한 작업 실행에서 관찰된 가장 최근의 값 5개를 검색합니다.

해결 방법: 현재 작업 실행에서 관찰된 값과 유의미한 비교를 수행하려면 집계 함수를 사용하여 이러한 값을 단일 숫자로 줄여야 합니다.

유효한 예:

- `RowCount >= avg(last(5))`
- `RowCount > last(1)`
- `RowCount < last()`

잘못된 예: `RowCount > last(5)`.

오류 조건:

- Function `index` used in threshold requires positive integer argument.
- Index argument must be an integer. Valid syntax example: `RowCount > index(last(10, 2))`, which means `RowCount` must be greater than third most recent execution from last 10 job runs.

해결 방법: 동적 규칙을 작성할 때 `index` 집계 함수를 사용하여 목록에서 하나의 기록 값을 선택할 수 있습니다. 예를 들어 `RowCount > index(last(5), 1)`은 현재 작업에서 관찰된 행 수가 해당 작업에서 관찰된 두 번째로 최근 행 수보다 확실하게 큰지 여부를 확인합니다. `index`는 0으로 인덱싱됩니다.

오류 조건: `IllegalArgumentException: Parsing Error: Rule Type: DetectAnomalies is not valid.`

해결 방법: 이상 탐지는 AWS Glue 4.0에서만 사용할 수 있습니다.

오류 조건: `IllegalArgumentException: Parsing Error: Unexpected condition for rule of type ... no viable alternative at input`

참고: ...은 동적입니다. 예: `IllegalArgumentException: Parsing Error: Unexpected condition for rule of type RowCount with number return type, line 4:19 no viable alternative at input '>last'.`

해결 방법: 이상 탐지는 AWS Glue 4.0에서만 사용할 수 있습니다.

사용자 클래스에서의 예외: `org.apache.spark.sql AnalysisException: org.apache.hadoop.hive.ql.metadata. HiveException`

오류 조건: `Exception in User Class: org.apache.spark.sql.AnalysisException: org.apache.hadoop.hive.ql.metadata.HiveException: Unable to fetch table`

mailpiece_submitted. StorageDescriptor#InputFormat cannot be null for table: mailpiece_submitted (Service: null; Status Code: 0; Error Code: null; Request ID: null; Proxy: null)

원인: Glue 데이터 카탈로그에서 Apache Iceberg를 사용하고 있는데 AWS Glue AWS 데이터 카탈로그의 입력 형식 속성이 비어 있습니다.

해결 방법: 이 문제는 DQ 규칙에서 CustomSQL 레이블 유형을 사용할 때 발생합니다. 이 문제를 해결하는 한 가지 방법은 '기본'을 사용하거나 카탈로그 이름 glue_catalog.를 <database>.<table> in Custom ruletype으로 추가하는 것입니다.

UNCLASSIFIED_ERROR; IllegalArgumentException: 파싱 오류: 규칙이나 분석기가 제공되지 않았습니다. 입력 시 실행 가능한 대안이 없습니다.

오류 조건: UNCLASSIFIED_ERROR; IllegalArgumentException: Parsing Error: No rules or analyzers provided., no viable alternative at input

해결: DQDL은 파싱할 수 없습니다. 이런 일이 발생할 수 있는 경우가 몇 가지 있습니다. 복합 규칙을 사용하는 경우 올바른 괄호가 있는지 확인하세요.

```
(RowCount >= avg(last(10)) * 0.6) and (RowCount <= avg(last(10)) * 1.4) instead of  
RowCount >= avg(last(10)) * 0.6 and RowCount <= avg(last(10)) * 1.4
```

Amazon Q 데이터 통합 AWS Glue

Amazon Q 데이터 AWS Glue 통합은 데이터 엔지니어와 ETL 개발자가 자연어를 사용하여 데이터 통합 작업을 구축할 수 있게 AWS Glue 해주는 새로운 생성 AI 기능입니다. 엔지니어와 개발자는 Amazon Q에 작업을 작성하고, 문제를 해결하고, 데이터 통합에 대한 AWS Glue 질문에 답하도록 요청할 수 있습니다.

Amazon Q란 무엇인가요?

Note

Amazon Bedrock 제공: [자동](#) 악용 탐지 AWS 기능을 구현합니다. Amazon Q 데이터 통합은 Amazon Bedrock을 기반으로 구축되었으므로 사용자는 Amazon Bedrock에 구현된 제어 기능을 최대한 활용하여 안전, 보안, 인공 지능(AI)의 책임 있는 사용을 강화할 수 있습니다.

Amazon Q는 애플리케이션을 이해, 구축, 확장 및 운영하는 데 도움이 되는 생성적 인공 지능 (AI) 기반 대화형 도우미입니다. AWS Amazon Q를 지원하는 모델은 고품질 AWS 콘텐츠로 보강되어 더욱 완전하고 실행 가능하며 참조할 수 있는 답변을 제공하여 구축을 가속화합니다. AWS자세한 내용은 [What is Amazon Q?](#)를 참조하세요.

AWS Glue의 Amazon Q 데이터 통합이란 무엇인가요?

Amazon Q 데이터 통합에는 다음과 같은 기능이 AWS Glue 포함됩니다.

- 채팅 — Amazon Q Data Integration은 AWS Glue 소스 AWS Glue 및 대상 커넥터, AWS Glue ETL 작업, 데이터 카탈로그, 크롤러 및 기타 기능 설명서, 모범 사례와 AWS Lake Formation같은 데이터 통합 도메인에 대한 자연어 질문에 영어로 답변할 AWS Glue 수 있습니다. Amazon Q 데이터 통합은 AWS Glue step-by-step 지침에 따라 처리되며 해당 정보 소스에 대한 참조를 포함합니다.
- 데이터 통합 코드 생성 — Amazon Q 데이터 통합을 통해 AWS Glue ETL 스크립트에 대한 질문에 답하고 영어로 된 자연어 질문이 주어지면 새 코드를 생성할 AWS Glue 수 있습니다.
- 문제 해결 — Amazon Q 데이터 AWS Glue 통합은 AWS Glue 작업 오류를 이해하는 데 도움이 되도록 특별히 설계되었으며 문제의 근본 원인 및 해결을 위한 step-by-step 지침을 제공합니다.

Note

Amazon Q 데이터 통합은 대화가 진행되는 동안 향후 응답을 알리기 위해 대화의 컨텍스트를 사용하지 AWS Glue 않습니다. Amazon Q 데이터 통합과의 각 AWS Glue 대화는 이전 또는 향후 대화와 무관합니다.

AWS Glue의 Amazon Q 데이터 통합을 사용 중이신가요?

Amazon Q 패널에서 Amazon Q에 AWS Glue ETL 스크립트용 코드 생성을 요청하거나 AWS Glue 기능에 대한 질문에 답변하거나 오류 해결을 요청할 수 있습니다. 응답은 스크립트를 사용자 지정하고 검토 및 실행하기 위한 step-by-step 지침이 PySpark 포함된 ETL 스크립트입니다. 질문의 경우 데이터 통합 지식을 기반으로 답변이 생성되며, 답변에는 참조할 수 있는 요약 및 소스 URL이 포함되어 있습니다.

예를 들어 Amazon Q에 "Snowflake에서 읽고, 필드 이름을 바꾸고, Redshift에 쓰는 Glue 스크립트를 제공해 주십시오"라고 AWS Glue 요청하면 이에 대한 응답으로 Amazon Q 데이터 통합이 요청된 작업을 수행할 수 있는 AWS Glue 작업 스크립트를 반환합니다. 생성된 코드를 검토하여 요청된 의도를 충족하는지 확인할 수 있습니다. 만족하면 프로덕션 환경에서 작업으로 배포할 수 있습니다. AWS Glue 통합 기능에 오류와 실패를 설명하고 해결책을 제안하도록 요청하여 작업 문제를 해결할 수 있습니다. Amazon Q는 데이터 통합 모범 사례에 대한 AWS Glue 질문에 답변해 드립니다.

The screenshot shows the AWS Glue Studio interface. The left sidebar contains navigation options like 'Getting started', 'ETL jobs', 'Data Catalog', and 'Legacy pages'. The main content area is titled 'AWS Glue Studio' and includes a 'Create job' section with three options: 'Visual ETL', 'Notebook', and 'Script editor'. Below this is an 'Example jobs' section with a 'Create example job' button. The 'Your jobs' section shows a search for 'demo' with 2 matches, displaying a table of jobs.

Job name	Type	Last modified	AWS Glue version
q-demo-taxi	Glue ETL	4/26/2024, 1:19:07 PM	4.0
q-demo	Glue ETL	4/25/2024, 3:41:38 PM	4.0

다음은 Amazon Q 데이터 통합이 기반을 구축하는 데 어떻게 AWS Glue 도움이 되는지 보여주는 예제 질문입니다 AWS Glue.

AWS Glue ETL 코드 생성:

- S3에서 JSON을 읽고, 매핑 적용을 사용하여 필드를 변환하고, Amazon Redshift에 쓰는 AWS Glue 스크립트를 작성합니다.
- DynamoDB에서 읽기 위한 AWS Glue 스크립트를 작성하고 변환을 적용하고 S3에 DropNullFields Parquet으로 쓰려면 어떻게 해야 합니까?
- MySQL에서 읽고, 비즈니스 로직을 기반으로 일부 필드를 삭제하고, Snowflake에 쓰는 AWS Glue 스크립트를 주세요.
- DynamoDB에서 읽고 S3에 AWS Glue JSON으로 쓸 작업을 작성합니다.
- S3에 대한 AWS Glue 데이터 카탈로그용 AWS Glue 스크립트를 개발하는 데 도움을 주세요.
- S3에서 JSON을 읽고, 널을 삭제하고, Redshift에 쓰는 AWS Glue 작업을 작성합니다.

AWS Glue 기능 설명:

- AWS Glue 데이터 품질을 사용하려면 어떻게 해야 하나요?
- AWS Glue 구직 북마크는 어떻게 사용하나요?
- AWS Glue 자동 크기 조정을 활성화하려면 어떻게 해야 하나요?
- AWS Glue 동적 프레임과 Spark 데이터 프레임의 차이점은 무엇입니까?
- 지원되는 연결 유형에는 어떤 것이 있습니까? AWS Glue

AWS Glue 문제 해결:

- AWS Glue 작업 시 발생하는 메모리 부족 (OOM) 오류를 해결하는 방법은 무엇입니까?
- AWS Glue 데이터 품질을 설정할 때 표시될 수 있는 오류 메시지는 어떤 것이 있으며 어떻게 해결할 수 있습니까?
- Amazon S3 액세스 거부 오류가 발생한 AWS Glue 작업을 수정하려면 어떻게 해야 합니까?
- AWS Glue 작업 시 데이터 셔플 관련 문제를 해결하려면 어떻게 해야 합니까?

Amazon Q 데이터 통합과의 상호 작용을 위한 모범 사례

다음은 Amazon Q 데이터 통합과 상호 작용하는 모범 사례입니다.

- Amazon Q 데이터 통합과 상호 작용할 때는 구체적인 질문을 하고, 복잡한 요청이 있을 때는 반복하고, 답변이 정확한지 확인하십시오.
- 자연어로 데이터 통합 프롬프트를 제공할 때는 어시스턴트가 필요한 내용을 정확히 이해할 수 있도록 최대한 구체적으로 작성하십시오. “S3에서 데이터 추출”을 요청하는 대신 “S3에서 JSON 파일을 추출하는 AWS Glue 스크립트 작성”과 같은 세부 정보를 제공하십시오.
- 생성된 스크립트를 실행하기 전에 검토하여 정확성을 확인하십시오. 생성된 스크립트에 오류가 있거나 의도와 일치하지 않는 경우 도우미에게 수정 방법에 대한 지침을 제공하십시오.
- 생성형 AI 기술은 새로운 기술이며 응답에 할루시네이션이라고 하는 실수가 있을 수 있습니다. 환경 또는 워크로드에서 사용하기 전에 오류와 취약성이 있는지 모든 코드를 테스트하고 검토하세요.

AWS Glue 서비스 개선을 위한 Amazon Q 데이터 통합

Amazon Q 데이터 통합이 AWS 서비스에 대해 가장 관련성이 높은 정보를 AWS Glue 제공할 수 있도록 아마존은 서비스 개선을 위해 사용자가 Amazon Q에 묻는 질문과 이에 대한 답변과 같은 Amazon Q의 특정 콘텐츠를 사용할 수 있습니다.

당사에서 사용하는 콘텐츠 및 옵트아웃 방법에 대한 자세한 내용은 [Amazon Q Developer 사용 설명서의 Amazon Q Developer 서비스 개선을](#) 참조하십시오.

고려 사항

AWS Glue의 Amazon Q 데이터 통합을 사용하기 전에 다음 항목을 고려하세요.

- 현재 코드 생성은 PySpark 커널에서만 작동합니다. 생성된 코드는 Python Spark 기반 AWS Glue 작업용입니다.
- Amazon Q 데이터 통합에서 AWS Glue지원되는 코드 생성 기능 조합에 대한 자세한 내용은 [을 참조하십시오](#) [지원되는 코드 생성 기능](#).

에서 Amazon Q 데이터 통합 설정 AWS Glue

다음 섹션에서는 AWS Glue의 Amazon Q 데이터 통합을 설정하는 방법에 대해 설명합니다.

주제

- [IAM 권한 구성](#)

IAM 권한 구성

이 주제에서는 Amazon Q 채팅 환경 및 AWS Glue Studio 노트북 환경에 대해 구성하는 IAM 권한을 설명합니다.

주제

- [Amazon Q 채팅을 위한 IAM 권한 구성](#)
- [스튜디오 노트북의 IAM 권한 구성 AWS Glue](#)

Amazon Q 채팅을 위한 IAM 권한 구성

Amazon Q 데이터 통합에 사용되는 API에 권한을 AWS Glue 부여하려면 적절한 AWS ID 및 액세스 관리 (IAM) 권한이 필요합니다. 다음 사용자 지정 AWS 정책을 IAM ID (예: 사용자, 역할 또는 그룹)에 연결하여 권한을 얻을 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:StartCompletion",
        "glue:GetCompletion"
      ],
      "Resource": [
        "arn:aws:glue:*:*:completion/*"
      ]
    }
  ]
}
```

스튜디오 노트북의 IAM 권한 구성 AWS Glue

AWS Glue 스튜디오 노트북에서 Amazon Q 데이터 통합을 활성화하려면 노트북 IAM 역할에 다음 권한이 연결되어 있어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

    {
      "Effect": "Allow",
      "Action": [
        "glue:StartCompletion",
        "glue:GetCompletion"
      ],
      "Resource": [
        "arn:aws:glue:*:*:completion/*"
      ]
    },
    {
      "Sid": "CodeWhispererPermissions",
      "Effect": "Allow",
      "Action": [
        "codewhisperer:GenerateRecommendations"
      ],
      "Resource": "*"
    }
  ]
}

```

Note

Amazon Q 데이터 AWS Glue 통합에는 프로그래밍 방식으로 사용할 수 있는 AWS SDK를 통해 사용할 수 있는 API가 없습니다. Amazon Q 채팅 패널 또는 AWS Glue 스튜디오 노트 북을 통해 이러한 경험을 활성화하기 위해 IAM 정책에서 사용되는 API는 다음과 같습니다. StartCompletion GetCompletion

권한 할당

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- AWS IAM ID 센터의 사용자 및 그룹: 권한 세트를 생성하십시오. AWS IAM Identity Center 사용 설명서의 [권한 세트 생성](#)의 지침을 따르세요.
- ID 제공업체를 통해 IAM에서 관리되는 사용자: ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 [서드 파티 자격 증명 공급자의 역할 만들기\(연합\)](#)의 지침을 따르세요.
- IAM 사용자:
 - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 [IAM 사용자의 역할 생성](#)의 지침을 따르세요.

- (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 [사용자\(콘솔\)에 권한 추가](#)의 지침을 따르세요.

지원되는 코드 생성 기능

다음은 AWS Glue의 Amazon Q 데이터 통합에 있는 코드 생성 기능 조합입니다.

소스	변환	대상
다음과 같은 형식 유형을 사용하는 S3: json, csv, parquet, hudi, delta	ApplyMapping	다음과 같은 형식 유형을 사용하는 S3: json, csv, avro, orc, parquet, hudi, delta
Glue Data 카탈로그	RenameField	Glue Data 카탈로그
Amazon Redshift	DropFields	Amazon Redshift
MySQL	SelectFields	MySQL
Postgres	DropNullFields	Postgres
Oracle	필터	Oracle
SQL Server	Spigot	SQL Server
DynamoDB	사용자 지정 SQL 코드	DynamoDB
Snowflake	Aggregate	Snowflake
MongoDB	DropDuplicates	MongoDB
사용자 지정 JDBC 커넥터	조인	커스텀 JDBC 커넥터
커스텀 스파크 커넥터	결합	커스텀 스파크 커넥터
구글 BigQuery		구글 BigQuery
Teradata		Teradata
아마존 OpenSearch 서비스		아마존 OpenSearch 서비스

소스	변환	대상
Vertica		Vertica
애저 SQL		Azure SQL
SAP HANA		SAP HANA
애저 코스모스		아주르 코스모스

상호 작용 예제

Amazon Q 데이터 통합을 AWS Glue 통해 Amazon Q 패널에 질문을 입력할 수 있습니다. AWS Glue 에서 제공하는 데이터 통합 기능에 관한 질문을 입력할 수 있습니다. 참조 문서와 함께 자세한 답변이 반환됩니다.

또 다른 사용 사례는 AWS Glue ETL 작업 스크립트 생성입니다. 데이터 추출, 변환, 로드 작업을 수행 하는 방법에 대해 질문할 수 있습니다. 생성된 PySpark 스크립트가 반환됩니다.

주제

- [Amazon Q 채팅 상호 작용](#)
- [AWS Glue 스튜디오 노트북 상호 작용](#)

Amazon Q 채팅 상호 작용

AWS Glue 콘솔에서 새 작업 작성을 시작하고 Amazon Q에 “Snowflake에서 읽고, 필드 이름을 바꾸고, Redshift에 쓰는 Glue 스크립트를 제공해 주세요.” 라고 요청하세요.

The screenshot shows the AWS Glue Studio interface. The left sidebar contains navigation options like 'Getting started', 'ETL jobs', 'Data Catalog', and 'Legacy pages'. The main content area is titled 'AWS Glue Studio' and features a 'Create job' section with three options: 'Visual ETL', 'Notebook', and 'Script editor'. Below this is an 'Example jobs' section with a 'Create example job' button. The 'Your jobs (2)' section displays a table of existing jobs:

Job name	Type	Last modified	AWS Glue version
q-demo-taxi	Glue ETL	4/26/2024, 1:19:07 PM	4.0
q-demo	Glue ETL	4/25/2024, 3:41:38 PM	4.0

코드가 생성된 것을 확인할 수 있을 것입니다. 이 응답을 통해 목적에 맞는 AWS Glue 코드를 작성하는 방법을 배우고 이해할 수 있습니다. 생성된 코드를 스크립트 편집기에 복사하여 붙여넣고 플레이스홀더를 구성할 수 있습니다. 작업에 AWS ID 및 액세스 관리 (IAM) 역할과 AWS Glue 연결을 구성한 후 작업을 저장하고 실행합니다. 작업이 완료되면 Amazon Redshift에서 Snowflake에서 내보낸 테이블에 대한 쿼리를 시작할 수 있습니다.

다음 프롬프트는 서로 다른 두 소스에서 데이터를 읽고, 개별적으로 필터링 및 프로젝션하고, 공통 키로 결합하고, 출력을 세 번째 대상에 씁니다. Amazon Q에 질문하십시오. “S3에서 Parquet 형식으로 데이터를 읽고 일부 필드를 선택하고 싶습니다. 또한 DynamoDB에서 데이터를 읽고, 일부 필드를 선택하고, 일부 행을 필터링하고 싶습니다. 이 두 데이터셋을 통합하고 결과를 여기에 쓰고 싶습니다. OpenSearch

The screenshot shows the AWS Glue Studio interface. On the left is a navigation sidebar with categories like 'Getting started', 'ETL jobs', 'Data Catalog', and 'Data Integration and ETL'. The main content area is titled 'AWS Glue Studio' and 'Jobs'. It features a 'Create job' section with three options: 'Visual ETL' (highlighted in orange), 'Notebook', and 'Script editor'. Below this is an 'Example jobs' section with a 'Create example job' button. The 'Your jobs (5)' section shows a search bar with 'demo' and 3 matches. A table lists the jobs:

Job name	Type	Last modified	AWS Glue version
q-demo-snowflake-to-redshift	Glue ETL	4/26/2024, 1:28:55 PM	4.0
q-demo-taxi	Glue ETL	4/26/2024, 1:19:07 PM	4.0
q-demo	Glue ETL	4/25/2024, 3:41:38 PM	4.0

코드가 생성됩니다. 작업이 완료되면 다운스트림 워크로드에서 인덱스를 사용할 수 OpenSearch 있고 인덱스를 사용할 수 있습니다.

AWS Glue 스튜디오 노트북 상호 작용

새 셀을 추가하고 코멘트를 입력하여 달성하고자 하는 목표를 설명하세요. Tab 키와 Enter를 누르면 권장 코드가 표시됩니다.

첫 번째 목적은 데이터를 추출하는 것입니다. “Glue Data Catalog 테이블을 읽는 코드를 주세요.”, “star_rating>3인 필터 변환을 적용하는 코드를 주세요”, “S3에 프레임을 Parquet으로 쓰는 코드를 주세요.”

q-nodes [↗](#) Stop notebook Download Notebook Actions ▾ Save Run

Notebook | Script | Job details | Runs | **Data quality - updated** | Schedules | Version Control

Glue PySpark

```

Worker Type: G.1X
Number of Workers: 5
Session ID: a6846a9a-6489-4599-bf8d-066b59d887da
Applying the following default arguments:
--glue_kernel_version 1.0.4
--enable-glue-datacatalog true
Waiting for session a6846a9a-6489-4599-bf8d-066b59d887da to get into ready status...
Session a6846a9a-6489-4599-bf8d-066b59d887da has been created.

```

[]:

Mode: Edit Ln 1, Col 1 Untitled.ipynb

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie pref

q-nodes [↗](#) Stop notebook Download Notebook Actions ▾ Save Run

Notebook | Script | Job details | Runs | **Data quality - updated** | Schedules | Version Control

Glue PySpark

```

|      US| 171306|R00GN0TEQS4ISDM| 90211|white and yellow ...| 3| 5| 5| N|PAP, and regular ...|Words themselves
...|2013-01-23 00:00:00| 2013| Jewelry|

```

only showing top 20 rows

```

/opt/amazon/spark/python/lib/pyspark.zip/pyspark/sql/dataframe.py:127: UserWarning: DataFrame constructor is internal. Do not directly use it.

```

[]:

Mode: Edit Ln 1, Col 1 Untitled.ipynb

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie pref

The screenshot shows the AWS Glue Studio notebook interface. At the top, there are buttons for 'Stop notebook', 'Download Notebook', 'Actions', 'Save', and 'Run'. Below these are tabs for 'Notebook', 'Script', 'Job details', 'Runs', 'Data quality - updated', 'Schedules', and 'Version Control'. The main area displays a data table with the following content:

Marine end-to-e... Lake maracaibo in...	2013	25	US 2013-01-27 00:00:00	207443 white gold anklet...	Jewelry	5 Re
23008 RDYS06F9U5EZLSG	N	6				
gular supply al... Amongst other neg...	2013					
89950 RBJPZWSA0NG285W	N	2	US 2013-01-11 00:00:00	352961 silver-plated hai...	Jewelry	5 Gr
oundwater recha... Not exactly were ...	2013					

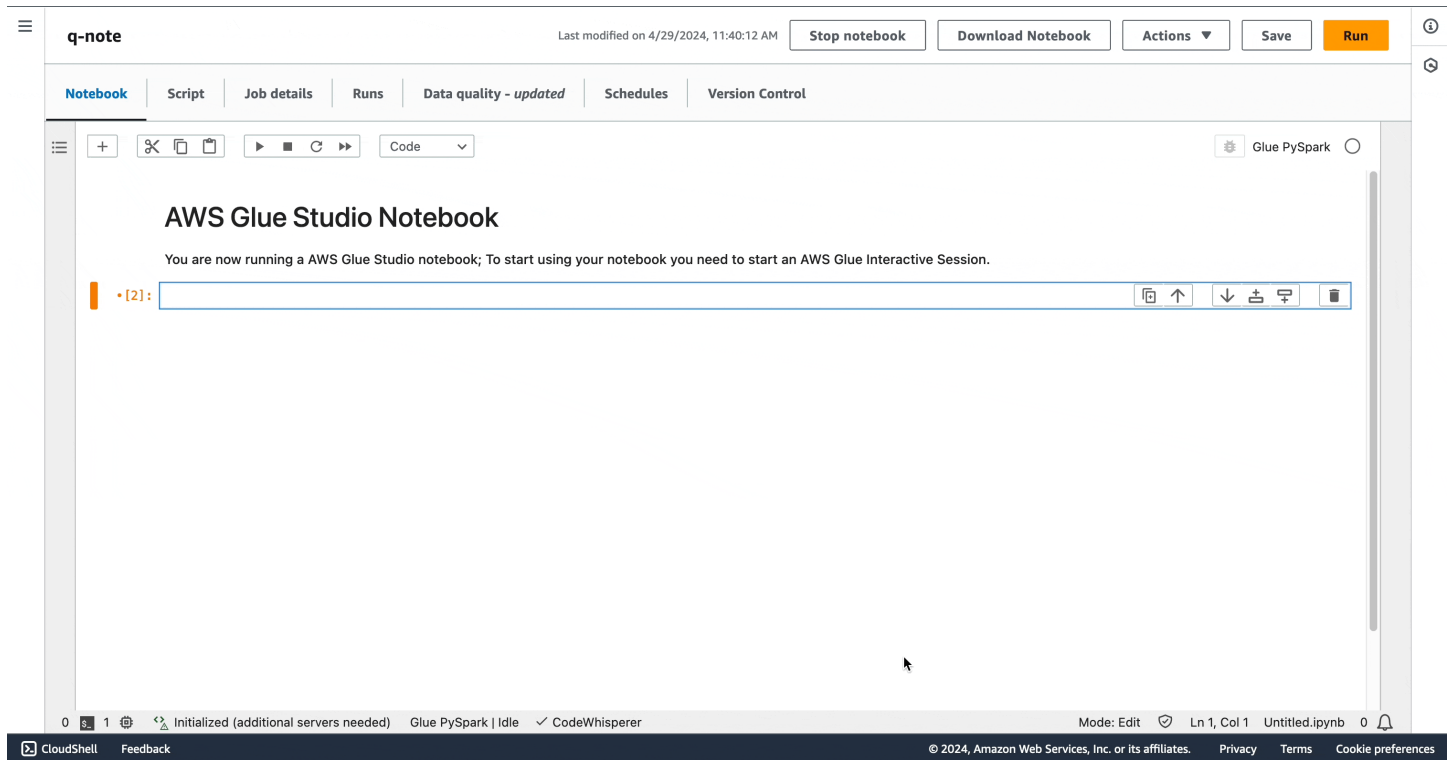
Below the table, it says 'only showing top 20 rows'. At the bottom of the notebook, there is a status bar with '0', '1', 'Initialized (additional servers needed)', 'Glue PySpark | Idle', 'CodeWhisperer', 'Mode: Command', 'Ln 1, Col 1', 'Untitled.ipynb', and '0'. The footer contains '© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie pref'.

Amazon Q 채팅 환경과 마찬가지로 코드를 사용하는 것이 좋습니다. Tab을 누르면 권장 코드가 선택됩니다.

생성된 코드의 소스에 적합한 옵션을 입력하여 각 셀을 실행할 수 있습니다. 실행 중 언제든지 `show()` 메서드를 사용하여 데이터세트 샘플을 미리 볼 수도 있습니다.

복잡한 프롬프트

하나의 복잡한 프롬프트로 전체 스크립트를 생성할 수 있습니다. “S3에는 JSON 데이터가 있고 Oracle에는 결합이 필요한 데이터가 있습니다. 두 소스에서 모두 읽고 조인한 다음 결과를 Redshift에 기록하는 Glue 스크립트를 제공하십시오.”



The screenshot displays the AWS Glue Studio Notebook interface. At the top, there's a header with the notebook name 'q-note', a timestamp 'Last modified on 4/29/2024, 11:40:12 AM', and several action buttons: 'Stop notebook', 'Download Notebook', 'Actions', 'Save', and 'Run'. Below this is a navigation bar with tabs for 'Notebook', 'Script', 'Job details', 'Runs', 'Data quality - updated', 'Schedules', and 'Version Control'. The main content area shows a message: 'AWS Glue Studio Notebook. You are now running a AWS Glue Studio notebook; To start using your notebook you need to start an AWS Glue Interactive Session.' Below the message is a code editor area with a toolbar and a status bar at the bottom showing 'Glue PySpark | Idle' and 'CodeWhisperer'.

노트북에서 Amazon Q 데이터를 통합하면 Amazon Q 채팅에서 AWS Glue 생성된 것과 동일한 코드 스니펫이 생성되었음을 알 수 있습니다.

실행을 선택하거나 프로그래밍 방식으로 노트북을 작업으로 실행할 수 있습니다.

AWS Glue에서 오케스트레이션

다음 섹션에서는 AWS Glue에서 작업 오케스트레이션에 대한 정보를 제공합니다.

주제

- [트리거를 사용하여 작업 및 크롤러 시작](#)
- [AWS Glue에서 블루프린트 및 워크플로를 사용하여 복잡한 ETL 활동 수행](#)
- [AWS Glue에서 블루프린트 개발](#)

트리거를 사용하여 작업 및 크롤러 시작

AWS Glue에서는 트리거라는 Data Catalog 객체를 생성할 수 있습니다. 이 객체를 사용하면 하나 이상의 크롤러 또는 추출, 변환, 로드 작업을 수동 또는 자동으로 시작할 수 있습니다. 트리거를 사용하여 종속 작업 및 크롤러 체인을 설계할 수 있습니다.

Note

워크플로를 정의하여 동일한 작업을 수행할 수 있습니다. 워크플로는 복잡한 다중 작업 ETL 작업을 생성하는 데 권장됩니다. 자세한 내용은 [the section called “블루프린트 및 워크플로를 사용하여 복잡한 ETL 활동 수행”](#) 단원을 참조하십시오.

주제

- [AWS Glue 트리거](#)
- [트리거 추가](#)
- [트리거 활성화 및 비활성화](#)

AWS Glue 트리거

트리거가 실행되면 지정된 작업 및 크롤러를 시작할 수 있습니다. 트리거는 일정에 따라 또는 이벤트 조합을 기반으로 요청 시 실행됩니다.

Note

단일 트리거로 2개의 크롤러만 활성화할 수 있습니다. 여러 개의 데이터 스토어를 크롤링하려면 여러 크롤러를 동시에 실행하지 말고 크롤러마다 각각 여러 개의 소스를 사용합니다.

트리거는 여러 상태 중 하나에 존재할 수 있습니다. 트리거는 CREATED, ACTIVATED 또는 DEACTIVATED입니다. 일시적 상태(예: ACTIVATING)도 있습니다. 트리거 실행을 일시적으로 중지하려면 비활성화할 수 있습니다. 그런 다음 나중에 다시 활성화할 수 있습니다.

트리거에는 다음 세 가지 유형이 있습니다.

예약됨**cron 기반의 시간 기반 트리거**

일정에 따라 일련의 작업 또는 크롤러에 대한 트리거를 생성할 수 있습니다. 작업 또는 크롤러가 실행되는 빈도, 실행되는 요일 및 실행되는 시간과 같은 제약 조건을 지정할 수 있습니다. 이 제약 조건은 cron을 기반으로 합니다. 트리거를 위한 일정을 설정하고자 한다면 cron의 기능 및 제한을 고려해야 합니다. 예를 들어, 매월 31일에 크롤러를 실행하고자 한다면 매월 31일이 없다는 점을 유의하기 바랍니다. Cron에 대한 자세한 내용은 [작업 및 크롤러를 위한 시간 기반 일정](#) 섹션을 참조하세요.

조건

이전 작업이나 크롤러 또는 여러 작업이나 크롤러가 조건 목록을 만족할 때 실행되는 트리거입니다.

조건부 트리거를 생성할 때 감시할 작업 목록과 크롤러 목록을 지정합니다. 감시한 각 작업 또는 크롤러에 대해 감시할 상태(예: 성공, 실패, 시간 초과 등)를 지정합니다. 감시한 작업 또는 크롤러가 지정된 상태로 종료되면 트리거가 실행됩니다. 감시한 이벤트 중 일부 또는 전부가 발생할 때 트리거가 실행되도록 구성할 수 있습니다.

예를 들어 작업 J1과 작업 J2가 모두 성공적으로 완료되면 작업 J3을 시작하도록 트리거 T1을 구성하고 작업 J1 또는 작업 J2 중 하나가 실패할 경우 작업 J4를 시작하도록 또 다른 트리거 T2를 구성할 수 있습니다.

다음 표에는 트리거가 감시하는 작업 및 크롤러 완료 상태(이벤트)가 나와 있습니다.

작업 완료 상태	크롤러 완료 상태
<ul style="list-style-type: none"> • SUCCEEDED • STOPPED • FAILED • TIMEOUT 	<ul style="list-style-type: none"> • SUCCEEDED • FAILED • CANCELLED

온디맨드

활성화할 때 실행되는 트리거입니다. 온디맨드 트리거는 ACTIVATED 또는 DEACTIVATED 상태로 전환되지 않습니다. 항상 CREATED 상태로 유지됩니다.

트리거가 존재하는 즉시 실행할 수 있도록 플래그를 설정하여 생성할 때 예약된 트리거와 조건부 트리거를 활성화할 수 있습니다.

Important

다른 작업 또는 크롤러 완료 시 실행되는 작업 또는 크롤러를 종속이라고 합니다. 종속 작업 또는 크롤러는 완료된 작업 또는 크롤러가 트리거에 의해 시작된 경우에만 시작됩니다. 종속 체인 내의 모든 작업 또는 크롤러는 단일 예약됨 또는 온디맨드 트리거의 하위 요소여야 합니다.

트리거를 사용하여 작업 파라미터 전달

트리거는 시작하는 작업에 파라미터를 전달할 수 있습니다. 매개 변수에는 작업 인수, 시간 초과 값, 보안 구성 등이 포함됩니다. 트리거가 여러 작업을 시작하면 파라미터가 각 작업에 전달됩니다.

다음은 트리거에 의해 전달된 작업 인수에 대한 규칙입니다.

- 키 - 값 쌍의 키가 기본 작업 인수와 일치하면 전달된 인수는 기본 인수를 재정의합니다. 키가 기본 인수와 일치하지 않으면 인수는 추가 인수로 작업에 전달됩니다.
- 키 - 값 쌍의 키가 재정의할 수 없는 인수와 일치하면 전달된 인수는 무시됩니다.

자세한 내용은 AWS Glue API의 [the section called “트리거”](#) 단원을 참조하십시오.

트리거 추가

AWS Glue 콘솔, AWS Command Line Interface(AWS CLI) 또는 AWS Glue API를 사용하여 트리거를 추가할 수 있습니다.

Note

현재 AWS Glue 콘솔은 트리거를 사용하여 작업할 때 크롤러가 아닌 작업만 지원합니다. AWS CLI 또는 AWS Glue API를 사용하여 작업과 크롤러로 트리거를 구성할 수 있습니다.

트리거를 추가하려면(콘솔)

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 ETL 아래에서 트리거를 선택합니다. 트리거 추가를 선택합니다.
3. 다음 속성을 제공합니다.

명칭

트리거에 고유한 이름을 지정합니다.

트리거 유형

다음 중 하나를 지정하세요.

- 일정: 트리거는 특정 빈도와 시간에 실행됩니다.
 - 작업 이벤트: 조건부 트리거입니다. 목록의 작업 중 일부 또는 전부가 지정된 상태와 일치하면 트리거가 실행됩니다. 트리거가 실행되기 위해서는 감시한 작업이 트리거에 의해 시작되었어야 합니다. 어떤 작업을 선택하든지 하나의 작업 이벤트(완료 상태)만 감시할 수 있습니다.
 - 온디맨드: 트리거가 활성화되면 실행됩니다.
4. 트리거 마법사를 완료합니다. [검토(Review)] 페이지에서 [생성 시 트리거 사용(Enable trigger on creation)]을 선택하여 [일정(Schedule)] 및 [작업 이벤트(Job events)](조건부) 트리거를 즉시 활성화할 수 있습니다.

트리거를 추가하려면(AWS CLI)

- 다음과 유사한 명령을 입력합니다.

```
aws glue create-trigger --name MyTrigger --type SCHEDULED --schedule "cron(0 12 * * ? *)" --actions CrawlerName=MyCrawler --start-on-creation
```

이 명령은 라는 MyTrigger라는 스케줄 트리거를 생성합니다. 이 트리거는 매일 오후 12시(UTC)에 실행되며 MyCrawler라는 크롤러를 시작합니다. 트리거는 활성화된 상태로 생성됩니다.

자세한 내용은 [the section called “AWS Glue 트리거”](#) 단원을 참조하십시오.

작업 및 크롤러를 위한 시간 기반 일정

AWS Glue에서 작업 및 크롤러를 위한 시간 기반 일정을 정의합니다. 일정 정의는 Unix식 [cron](#) 구문을 사용합니다. [협정시계시\(UTC\)](#)의 시간을 지정하고 일정을 위한 최소한의 정확도는 5분입니다.

일정을 사용하여 실행되도록 작업 및 크롤러 구성에 대한 자세한 내용은 [트리거를 사용하여 작업 및 크롤러 시작](#) 섹션을 참조하세요.

cron 표현식

cron 표현식에는 각각 공백으로 구분되는 필수 필드 6개가 있습니다.

구문

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

필드	값	와일드카드
Minutes	0~59	, - * /
Hours	0~23	, - * /
Day-of-month	1~31	, - * ? / L W
월	1~12 또는 JAN-DEC	, - * /
요일	1~7 또는 SUN~SAT	, - * ? / L
연도	1970~2199	, - * /

와일드카드

- ,(쉼표) 와일드카드는 추가 값을 포함합니다. Month 필드에서 JAN, FEB, MAR는 1월, 2월, 3월을 포함한다는 의미입니다.
- -(대시) 와일드카드는 범위를 지정합니다. Day 필드에서 1~15는 지정된 달의 1일에서 15일까지 포함한다는 의미입니다.
- *(별표) 와일드카드는 필드의 모든 값을 포함합니다. Hours 필드에서 *는 모든 시간을 포함한다는 의미입니다.
- /(슬래시) 와일드카드로 증분을 지정합니다. Minutes 필드에서 **1/10**을 입력하면 지정한 시간의 1분부터 시작해서 매 10분 간격(예를 들어, 11분, 21분, 31분 등)을 지정할 수 있습니다.
- ?(물음표) 와일드카드는 어떤 한 가지나 다른 것을 지정합니다. Day-of-month 필드에 7을 입력하고 Day-of-week 필드에는 ?을 입력하면 매월 7일이 무슨 요일이든 상관없이 7번째 되는 날을 지정한다는 의미입니다.
- Day-of-month 또는 Day-of-week 필드에서 L 와일드카드는 해당 월 또는 주의 마지막 날을 지정합니다.
-] 필드에서는 W 와일드카드로 어떤 한 평일을 지정할 수 있습니다. Day-of-month Day-of-month 필드에서 3W를 해당 월의 세 번째 평일에 가장 가까운 날을 지정할 수 있습니다.

Limits

- 동일한 cron 표현식에 Day-of-month와 Day-of-week 필드를 지정할 수 없습니다. 이 필드 중 하나에 값을 지정하는 경우에는 다른 필드에서 반드시 ?(물음표)를 사용해야 합니다.
- 5분보다 빠른 속도로 이어지는 cron 식은 지원되지 않습니다.

예제

일정을 생성할 때는 다음과 같은 Cron 문자열을 사용할 수 있습니다.

분	시간	일	월	요일	연도	의미
0	10	*	*	?	*	매일 오전 10시(UTC)에 실행
15	12	*	*	?	*	매일 오후 12시 15분

분	시간	일	월	요일	연도	의미
						(UTC)에 실행
0	18	?	*	월-금	*	매주 월요일부터 금요일까지 오후 6시 (UTC)에 실행
0	8	1	*	?	*	매월 1일 오전 8시 (UTC)에 실행
0/15	*	*	*	?	*	15분마다 실행
0/10	*	?	*	월-금	*	월요일부터 금요일까지 10분마다 실행
0/5	8~17	?	*	월-금	*	월요일부터 금요일까지 오전 8시부터 오후 5시 55분(UTC) 사이에 5분마다 실행

예를 들어, 매일 12시 15분(UTC)에 실행하려면 다음과 같이 지정합니다.

```
cron(15 12 * * ? *)
```

트리거 활성화 및 비활성화

AWS Glue 콘솔, AWS Command Line Interface (AWS CLI) 또는 AWS Glue API를 사용하여 트리거를 활성화하거나 비활성화할 수 있습니다.

트리거를 활성화 또는 비활성화하려면(콘솔)

1. <https://console.aws.amazon.com/glue/>에서 AWS Management Console 로그인하고 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 ETL 아래에서 트리거를 선택합니다.
3. 원하는 트리거 옆의 확인란을 선택하고 작업 메뉴에서 트리거 활성화를 선택하여 트리거를 활성화하거나 트리거를 선택하여 트리거를 비활성화합니다.

트리거를 활성화 또는 비활성화하려면(AWS CLI)

- 다음 명령 중 하나를 입력합니다.

```
aws glue start-trigger --name MyTrigger
```

```
aws glue stop-trigger --name MyTrigger
```

트리거를 시작하면 트리거가 활성화되고 트리거를 중지하면 트리거가 비활성화됩니다. 온디맨드 트리거를 활성화하면 즉시 실행됩니다.

자세한 내용은 [the section called “AWS Glue 트리거”](#)을(를) 참조하세요.

AWS Glue에서 블루프린트 및 워크플로를 사용하여 복잡한 ETL 활동 수행

조직의 복잡한 추출, 변환, 로드 프로세스 중 일부는 여러 종속 AWS Glue 작업 및 크롤러를 사용하여 가장 잘 구현될 수 있습니다. AWS Glue 워크플로를 사용하면 AWS Glue에서 단일 엔터티로 실행하고 추적할 수 있는 복잡한 다중 작업, 다중 크롤러 ETL 프로세스를 설계할 수 있습니다. 워크플로를 생성하고 워크플로에서 작업, 크롤러 및 트리거를 지정한 후 온디맨드로 또는 일정에 따라 워크플로를 실행할 수 있습니다.

주제

- [AWS Glue의 워크플로 개요](#)

- [AWS Glue에서 워크플로 수동 생성 및 구축](#)
- [Amazon EventBridge 이벤트로 AWS Glue 워크플로 시작](#)
- [워크플로를 시작한 EventBridge 이벤트 보기](#)
- [AWS Glue에서 워크플로 실행 및 모니터링](#)
- [워크플로 실행 중지](#)
- [워크플로 실행 복구 및 재개](#)
- [AWS Glue에서 워크플로 실행 속성 가져오기 및 설정](#)
- [AWS Glue API를 사용하여 워크플로 쿼리](#)
- [AWS Glue의 블루프린트 및 워크플로 제한 사항](#)
- [AWS Glue의 블루프린트 오류 해결](#)
- [AWS Glue 블루프린트에 대한 페르소나 및 역할의 권한](#)

AWS Glue의 워크플로 개요

AWS Glue에서 여러 크롤러, 작업 및 트리거가 포함된 복잡한 ETL(추출, 변환 및 로드) 활동을 생성 및 시각화할 수 있습니다. 각 워크플로는 모든 구성 요소의 실행과 모니터링을 관리합니다. 워크플로는 각 구성 요소를 실행할 때 실행 진행률과 상태를 기록합니다. 이는 더 큰 태스크의 개요와 각 단계의 세부 정보를 제공합니다. AWS Glue 콘솔은 워크플로우의 시각적 표현을 그래프로 제공합니다.

AWS Glue 블루프린트에서 워크플로를 생성하거나 AWS Management Console 또는 AWS Glue API를 사용하여 구성 요소를 한 번에 수동으로 워크플로를 구축할 수 있습니다. 청사진에 대한 자세한 내용은 [the section called “블루프린트 개요”](#)을 참조하십시오.

워크플로 내의 트리거는 작업과 크롤러를 모두 시작할 수 있으며 작업 또는 크롤러가 완료되면 실행될 수 있습니다. 트리거를 사용하여 상호 종속된 작업과 크롤러로 이루어진 대형 체인을 생성할 수 있습니다. 작업 및 크롤러 종속성을 정의하는 워크플로 내의 트리거 외에도 각 워크플로에는 시작 트리거가 있습니다. 시작 트리거에는 다음 세 가지 유형이 있습니다.

- 일정 - 정의한 일정에 따라 워크플로가 시작됩니다. 일정은 매일, 매주, 매월 등이 될 수도 있고 cron 표현식을 기반으로 한 사용자 정의 일정이 될 수도 있습니다.
- 온디맨드 - 워크플로가 AWS Glue 콘솔, API 또는 AWS CLI에서 수동으로 시작됩니다.
- EventBridge 이벤트 - 단일 Amazon EventBridge 이벤트 또는 Amazon EventBridge 이벤트 배치가 발생하면 워크플로가 시작됩니다. 이 트리거 유형을 사용하면 AWS Glue는 이벤트 중심 아키텍처에서 이벤트 소비자가 될 수 있습니다. 모든 EventBridge 이벤트 유형은 워크플로를 시작할 수 있습니다. 일반적인 사용 사례는 Amazon S3 버킷에 새 객체가 도착하는 것입니다(S3 PutObject 작업).

이벤트 배치로 워크플로를 시작한다는 것은 지정된 수의 이벤트가 수신되거나 지정된 시간이 경과할 때까지 기다리는 것을 의미합니다. EventBridge 이벤트 트리거를 생성할 때 선택적으로 배치 조건을 지정할 수 있습니다. 배치 조건을 지정하는 경우 배치 크기(이벤트 수)를 지정해야 하며 선택적으로 배치 기간(초 수)을 지정할 수 있습니다. 기본 및 최대 배치 기간은 900초(15분)입니다. 먼저 충족되는 배치 조건이 워크플로를 시작합니다. 첫 번째 이벤트가 도착하면 배치 기간이 시작됩니다. 트리거를 생성할 때 배치 조건을 지정하지 않으면 배치 크기는 기본적으로 1로 설정됩니다.

워크플로가 시작되면 배치 조건이 재설정되고 이벤트 트리거는 워크플로를 다시 시작하기 위해 충족되는 다음 배치 조건을 감시하기 시작합니다.

다음 표는 배치 크기와 배치 기간이 함께 작동하여 워크플로를 트리거하는 방법을 보여줍니다.

배치 크기	배치 기간	결과 트리거 조건
10		10개의 EventBridge 이벤트가 도착하거나 첫 번째 이벤트가 도착한 후 15분 중 먼저 발생하는 시점에 워크플로가 트리거됩니다. (기간 크기를 지정하지 않으면 기본적으로 15분으로 설정됩니다.)
10	2분	10개의 EventBridge 이벤트가 도착하거나 첫 번째 이벤트가 도착한 후 2분 중 먼저 발생하는 시점에 워크플로가 트리거됩니다.
1		워크플로는 첫 번째 이벤트가 도착하면 트리거됩니다. 기간 크기는 관련이 없습니다. EventBridge 이벤트 트리거를 생성할 때 배치 조건을 지정하지 않으면 배치 크기의 기본값은 1입니다.

GetWorkflowRun API 작업은 워크플로를 트리거한 배치 조건을 반환합니다.

워크플로 시작 방법에 관계 없이 워크플로를 생성할 때 동시 워크플로 실행의 최대 수를 지정할 수 있습니다.

이벤트 또는 이벤트 배치가 결국 실패하는 워크플로 실행을 시작하는 경우 해당 이벤트 또는 이벤트 배치는 더 이상 워크플로 실행 시작에 대해 고려되지 않습니다. 새 워크플로 실행은 다음 이벤트 또는 이벤트 배치가 도착할 때만 시작됩니다.

⚠ Important

워크플로 내 작업, 크롤러, 트리거의 총수를 100개 이하로 제한합니다. 100개가 넘게 포함될 경우 워크플로 실행을 재개하거나 중지하려고 할 때 오류가 발생할 수 있습니다.

이벤트 조건이 충족되더라도 워크플로에 대해 설정된 동시성 제한을 초과하면 워크플로 실행이 시작되지 않습니다. 예상 이벤트 볼륨에 따라 워크플로 동시성 제한을 조정하는 것이 좋습니다. AWS Glue는 초과 동시성 제한으로 인해 실패한 워크플로 실행을 다시 시도하지 않습니다. 마찬가지로, 예상 이벤트 볼륨을 기반으로 워크플로 내에서 작업 및 크롤러에 대한 동시성 제한을 조정하는 것이 좋습니다.

워크플로 실행 속성

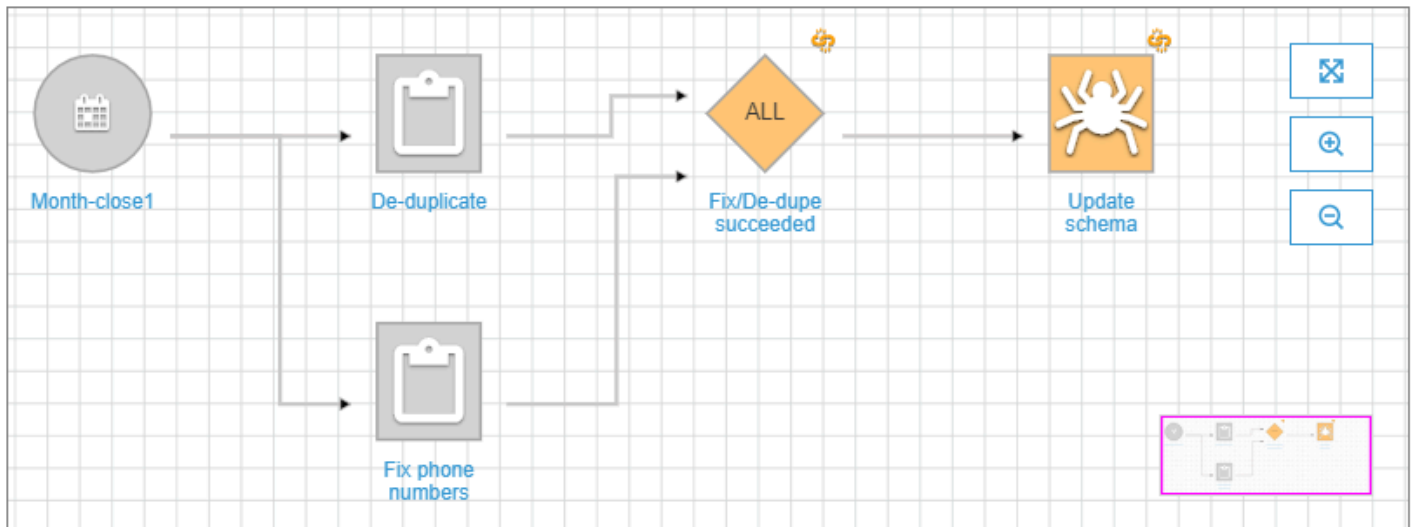
워크플로우 실행 전체의 상태를 공유 및 관리하려면 기본 워크플로우 실행 속성을 정의하면 됩니다. 이름/값 페어인 이러한 속성은 워크플로우의 모든 작업에서 사용할 수 있습니다. AWS Glue API를 사용하여 작업은 워크플로우 실행 속성을 검색하고 워크플로우에 나중에 오는 작업을 위해 수정할 수 있습니다.

워크플로 그래프

다음 이미지에서는 AWS Glue 콘솔에서 매우 기본적인 워크플로의 그래프를 보여줍니다. 워크플로우에는 구성 요소가 수십 개 있을 수 있습니다.

Graph Details History

Workflow : De-dupe and fix Remove Action ▾



이 워크플로는 2개의 작업 De-duplicate 및 Fix phone numbers를 시작하는 일정 트리거 Month-close1에 의해 시작됩니다. 두 작업이 성공적으로 완료되면 이벤트 트리거 Fix/De-dupe succeeded가 크롤러 Update schema를 시작합니다.

정적 및 동적 워크플로 보기

워크플로우별로 정적 보기와 동적 보기라는 개념이 있습니다. 정적 보기는 워크플로우의 설계를 나타냅니다. 동적 보기는 각 작업 및 크롤러에 대한 최신 실행 정보를 포함하는 런타임 보기입니다. 실행 정보에는 성공 상태와 오류 세부 정보가 포함됩니다.

워크플로우가 실행 중이면 콘솔에 완료된 작업과 아직 실행될 작업을 그래픽으로 나타내는 동적 보기가 표시됩니다. AWS Glue API를 사용하여 실행 중인 워크플로의 동적 보기를 검색할 수도 있습니다. 자세한 내용은 [AWS Glue API를 사용하여 워크플로 쿼리](#) 단원을 참조하십시오.

i 다음 사항도 참조하십시오.

- [the section called “블루프린트에서 워크플로 생성”](#)
- [the section called “워크플로 수동 생성 및 구축”](#)
- [워크플로](#)(워크플로 API의 경우)

AWS Glue에서 워크플로 수동 생성 및 구축

AWS Glue 콘솔을 사용하여 한 번에 한 노드씩 워크플로를 수동으로 생성하고 구축할 수 있습니다.

워크플로우에는 작업, 크롤러 및 트리거가 포함됩니다. 수동으로 워크플로를 생성하기 전에 워크플로에 포함시킬 작업과 크롤러를 생성합니다. 워크플로에 요청 시 실행되는 크롤러를 지정하는 것이 좋습니다. 워크플로우를 구축하는 동안 새 트리거를 생성할 수도 있고, 기존 트리거를 워크플로우에 복제할 수도 있습니다. 트리거를 복제하면 트리거와 연결된 모든 카탈로그 객체(이를 실행하는 작업 또는 크롤러 및 시작하는 작업 또는 크롤러)가 워크플로에 추가됩니다.

Important

워크플로 내 작업, 크롤러, 트리거의 총수를 100개 이하로 제한합니다. 100개가 넘게 포함될 경우 워크플로 실행을 재개하거나 중지하려고 할 때 오류가 발생할 수 있습니다.

워크플로우 그래프에 트리거를 추가하고 각 트리거에 대해 감시되는 이벤트와 작업을 정의하여 워크플로우를 구축합니다. 온디맨드 트리거 또는 일정 트리거인 시작 트리거를 시작하고, 이벤트(조건부) 트리거를 추가하여 그래프를 완성합니다.

1단계: 워크플로우 생성

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 ETL 아래에서 Workflows(워크플로우)를 선택합니다.
3. Add workflow(워크플로우 추가)를 선택하고 Add a new ETL workflow(새 ETL 워크플로우 추가) 양식을 작성합니다.

추가하는 선택적 기본 실행 속성은 워크플로우의 모든 작업에 대한 인수로 사용할 수 있습니다. 자세한 내용은 [AWS Glue에서 워크플로 실행 속성 가져오기 및 설정](#) 단원을 참조하십시오.

4. Add workflow(워크플로우 추가)를 선택합니다.

새 워크플로우가 Workflows(워크플로우) 페이지의 목록에 나타납니다.

2단계: 시작 트리거 추가

1. Workflows(워크플로우) 페이지에서 새 워크플로우를 선택합니다. 그런 다음 페이지 하단에서 [그래프(Graph)] 탭이 선택되어 있는지 확인합니다.

2. Add trigger(트리거 추가)를 선택하고 Add trigger(트리거 추가) 대화 상자에서 다음 중 하나를 수행합니다.

- Clone existing(기존 복제)을 선택하고 복제할 트리거를 선택합니다. 그런 다음 추가를 선택합니다.

트리거가 감시하는 작업 및 크롤러와 트리거가 시작하는 작업 및 크롤러와 함께 트리거가 그래프에 나타납니다.

실수로 잘못된 트리거를 선택한 경우 그래프에서 해당 트리거를 선택한 후 Remove(제거)를 선택합니다.

- Add new(새로 추가)를 선택하고 Add trigger(트리거 추가) 양식을 작성합니다.

1. [트리거 유형(Trigger type)]에 대해 [일정(Schedule)], [온디맨드(On demand)] 또는 [EventBridge 이벤트(EventBridge event)]를 선택합니다.

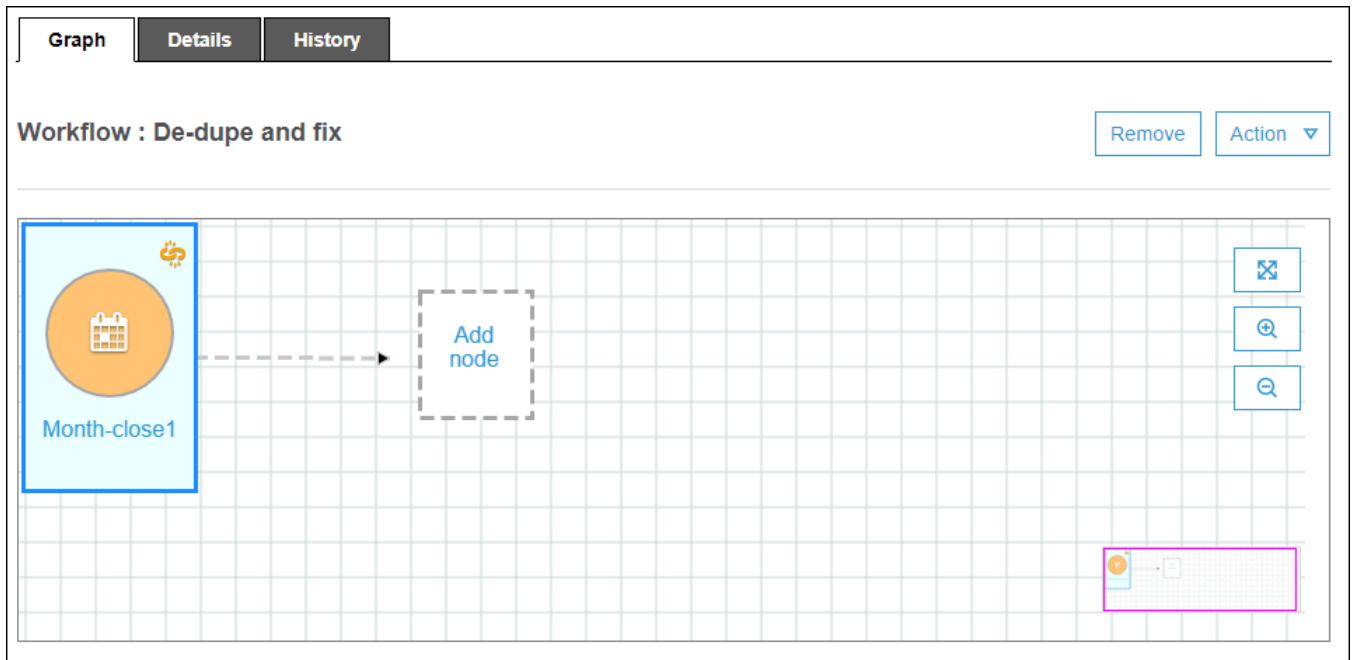
트리거 유형 [일정(Schedule)]에서 [주파수(Frequency)] 옵션 중 하나를 선택합니다. [사용자 정의(Custom)]를 선택하여 cron 표현식을 입력합니다.

트리거 유형 [EventBridge 이벤트(EventBridge event)]의 경우 [이벤트 수(Number of events)] (배치 크기)를 입력하고 선택적으로 [시간 지연(Time delay)](배치 기간)을 입력합니다. [시간 지연(Time delay)]을 생략하면 배치 기간은 기본적으로 15분으로 설정됩니다. 자세한 내용은 [AWS Glue의 워크플로 개요](#) 단원을 참조하십시오.

2. 추가(Add)를 선택합니다.

자리 표시자 노드(Add node(노드 추가)라고 레이블이 지정됨)와 함께 트리거가 그래프에 나타납니다. 아래 예에서 시작 트리거는 Month-close1이라는 일정 트리거입니다.

이때 트리거는 아직 저장되어 있지 않습니다.



3. 새 트리거를 추가한 경우 다음 단계를 완료합니다.

a. 다음 중 하나를 수행하십시오.

- 자리 표시자 노드(Add node(노드 추가))를 선택합니다.
- 시작 트리거가 선택되어 있는지 확인하고, 그래프 위의 Action(작업) 메뉴에서 Add jobs/crawlers to trigger(트리거할 작업/크롤러 추가)를 선택합니다.

b. Add jobs(s) and crawler(s) to trigger(트리거할 작업 및 크롤러 추가) 대화 상자에서 작업 또는 크롤러를 하나 이상 선택한 후 Add(추가)를 선택합니다.

트리거가 저장되고, 선택한 작업 또는 크롤러가 트리거의 커넥터와 함께 그래프에 나타납니다.

실수로 잘못된 작업 또는 크롤러를 추가한 경우, 해당 트리거 또는 커넥터를 선택하고 Remove(제거)를 선택하면 됩니다.

3단계: 트리거 추가

[이벤트(Event)] 유형의 트리거를 더 추가하여 워크플로를 계속 구축합니다. 그래프 캔버스를 축소 또는 확대하려면 그래프 오른쪽의 아이콘을 사용합니다. 추가할 각 트리거에 대해 다음 단계를 완료합니다.

Note

워크플로를 저장하는 작업은 없습니다. 마지막 트리거를 추가하고 트리거에 작업을 할당하면 워크플로가 완료되고 저장됩니다. 나중에 언제든지 다시 돌아와서 노드를 더 추가할 수 있습니다.

1. 다음 중 하나를 수행하십시오.

- 기존 트리거를 복제하려면 그래프의 노드가 선택되어 있지 않은지 확인하고, Action(작업) 메뉴에서 Add trigger(트리거 추가)를 선택합니다.
- 그래프에서 특정 작업 또는 크롤러를 감시하는 새 트리거를 추가하려면 작업 또는 크롤러 노드를 선택한 후 Add trigger(트리거 추가) 자리 표시자 노드를 선택합니다.

이후 단계에서 이 트리거가 감시할 작업 또는 크롤러를 더 추가할 수 있습니다.

2. Add Folder(폴더 추가) 대화 상자에서 다음 중 하나를 수행합니다.

- Add new(새로 추가)를 선택하고 Add trigger(트리거 추가) 양식을 작성합니다. 그런 다음 추가를 선택합니다.

트리거가 그래프에 나타납니다. 이후 단계에서 트리거를 완료합니다.

- Clone existing(기존 복제)을 선택하고 복제할 트리거를 선택합니다. 그런 다음 추가를 선택합니다.

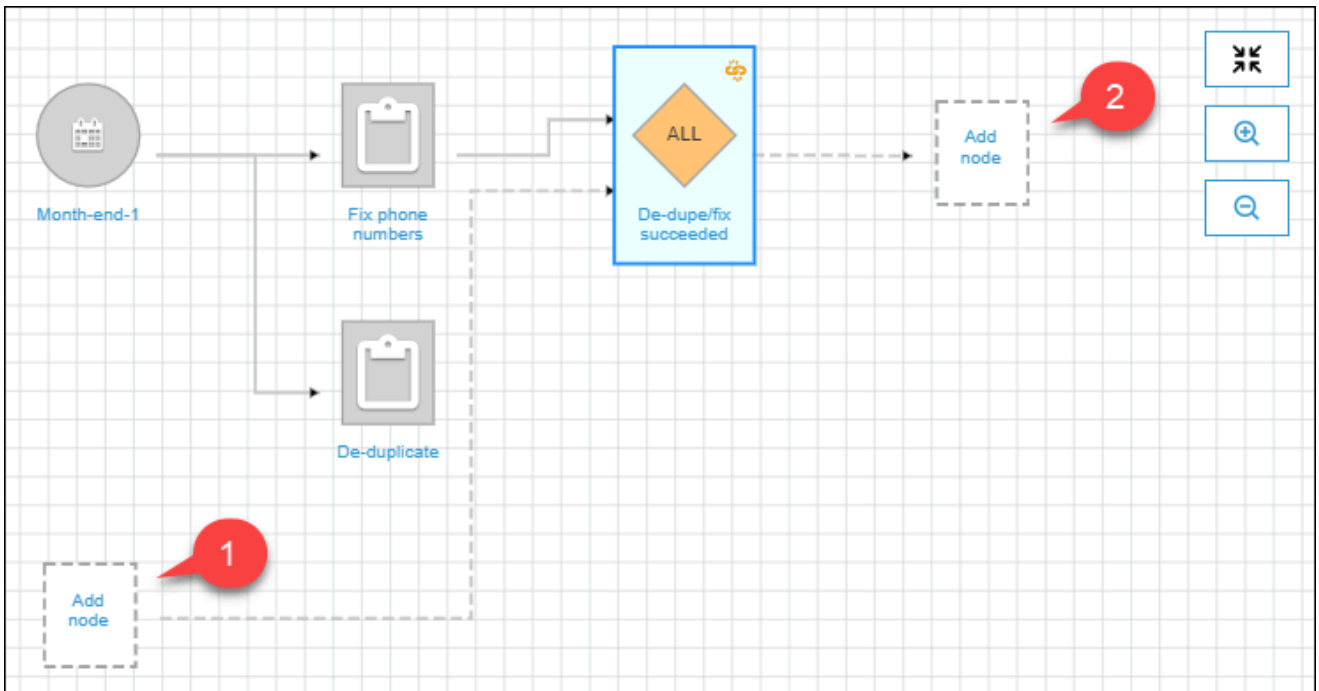
트리거가 감시하는 작업 및 크롤러와 트리거가 시작하는 작업 및 크롤러와 함께 트리거가 그래프에 나타납니다.

실수로 잘못된 트리거를 선택한 경우 그래프에서 해당 트리거를 선택한 후 Remove(제거)를 선택합니다.

3. 새 트리거를 추가한 경우 다음 단계를 완료합니다.

a. 새 트리거를 선택합니다.

다음 그래프에 나와 있듯이 트리거 De-dupe/fix succeeded가 선택되고 감시할 (1) 이벤트와 (2) 작업에 대한 자리 표시자 노드가 나타납니다.



- b. (트리거가 이미 이벤트를 감시하고 감시할 작업 또는 크롤러를 더 추가하려는 경우의 선택 사항) 감시할 이벤트 자리 표시자 노드를 선택하고, Add job(s) and crawler(s) to watch(감시할 작업 및 크롤러 추가) 대화 상자에서 작업 또는 크롤러를 하나 이상 선택합니다. 감시할 이벤트를 선택하고(SUCCEEDED, FAILED 등) Add(추가)를 선택합니다.
- c. 트리거가 선택되었는지 확인하고, 작업 자리 표시자 노드를 선택합니다.
- d. Add job(s) and crawler(s) to watch(감시할 작업 및 크롤러 추가) 대화 상자에서 작업 또는 크롤러를 하나 이상 선택하고 Add(추가)를 선택합니다.

선택한 작업 및 크롤러가 트리거의 커넥터와 함께 그래프에 나타납니다.

워크플로와 블루프린트에 대한 자세한 내용은 다음 주제를 참조하세요.

- [AWS Glue의 워크플로 개요](#)
- [AWS Glue에서 워크플로 실행 및 모니터링](#)
- [AWS Glue의 블루프린트에서 워크플로 생성](#)

Amazon EventBridge 이벤트로 AWS Glue 워크플로 시작

CloudWatch Events라고도 하는 Amazon EventBridge를 사용하면 AWS 서비스를 자동화하고 애플리케이션 가용성 문제나 리소스 변경 같은 시스템 이벤트에 자동으로 대응할 수 있습니다. AWS 서비스

의 이벤트는 거의 실시간으로 EventBridge로 전송됩니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 작업을 지정할 수 있습니다.

EventBridge 지원으로 AWS Glue는 이벤트 중심 아키텍처에서 이벤트 생산자 및 소비자 역할을 할 수 있습니다. 워크플로의 경우 AWS Glue는 모든 유형의 EventBridge 이벤트를 소비자로 지원합니다. 가장 일반적인 사용 사례는 Amazon S3 버킷에 새 객체 도착입니다. 데이터가 불규칙하거나 정의되지 않은 간격으로 도착하는 경우 이 데이터를 도착에 최대한 가깝게 처리할 수 있습니다.

Note

AWS Glue는 EventBridge 메시지의 배달을 보장하지 않습니다. AWS Glue는 EventBridge가 중복 메시지를 전달하는 경우 중복 제거를 수행하지 않습니다. 사용 사례에 따라 멍등성을 관리해야 합니다.

원하지 않는 이벤트가 전송되지 않도록 EventBridge 규칙을 올바르게 구성해야 합니다.

시작하기 전 준비 사항

Amazon S3 데이터 이벤트로 워크플로를 시작하려면 관심 있는 S3 버킷에 대한 이벤트가 AWS CloudTrail 및 EventBridge에 기록되었는지 확인해야 합니다. 그러려면 CloudTrail 추적을 생성해야 합니다. 자세한 내용은 [AWS 계정에 대한 추적 생성](#)을 참조하세요.

EventBridge 이벤트로 워크플로를 시작하려면

Note

다음 명령에서 다음과 같이 바꿉니다.

- `<workflow-name>`을 워크플로에 할당할 이름으로
- `<trigger-name>`을 트리거에 할당할 이름으로
- `<bucket-name>`을 Amazon S3 버킷의 이름으로
- `<account-id>`를 유효한 AWS 계정 ID로
- `<region>`을 리전 이름(예: us-east-1)으로
- `<rule-name>`을 EventBridge 규칙에 할당할 이름으로

1. EventBridge 규칙 및 대상을 생성하고 볼 수 있는 AWS Identity and Access Management(IAM) 권한이 있는지 확인합니다. 다음은 연결할 수 있는 샘플 정책입니다. 작업 및 리소스에 대한 제한을 설정하기 위해 범위를 좁힐 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:DisableRule",
        "events>DeleteRule",
        "events:PutTargets",
        "events:RemoveTargets",
        "events:EnableRule",
        "events:List*",
        "events:Describe*"
      ],
      "Resource": "*"
    }
  ]
}
```

2. AWS Glue에 이벤트를 전달할 때 EventBridge 서비스가 수입할 수 있는 IAM 역할을 생성합니다.
 - a. IAM 콘솔의 역할 생성 페이지에서 AWS 서비스를 선택합니다. 그런 다음 [CloudWatch Events] 서비스를 선택합니다.
 - b. [역할 생성(Create role)] 마법사를 완료합니다. 마법사가 자동으로 CloudWatchEventsBuiltInTargetExecutionAccess 및 CloudWatchEventsInvocationAccess 정책을 연결합니다.
 - c. 다음 인라인 정책을 역할에 연결합니다. 이 정책은 EventBridge 서비스가 이벤트를 AWS Glue로 보낼 수 있도록 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:notifyEvent"
      ]
    }
  ]
}
```

```

    ],
    "Resource": [
      "arn:aws:glue:<region>:<account-id>:workflow/<workflow-name>"
    ]
  }
]
}

```

3. 다음 명령을 입력하여 워크플로를 생성합니다.

선택적 추가 명령줄 파라미터에 대한 자세한 내용은 AWS CLI Command Reference의 [create-workflow](#)를 참조하세요.

```
aws glue create-workflow --name <workflow-name>
```

4. 다음 명령을 입력하여 워크플로에 대한 EventBridge 이벤트 트리거를 생성합니다. 워크플로의 시작 트리거가 됩니다. *<actions>*를 수행할 작업(시작할 작업 및 크롤러)으로 바꿉니다.

actions 인수를 코딩하는 방법에 대한 자세한 내용은 AWS CLI Command Reference의 [create-trigger](#)를 참조하세요.

```
aws glue create-trigger --workflow-name <workflow-name> --type EVENT --
name <trigger-name> --actions <actions>
```

단일 EventBridge 이벤트 대신 이벤트 일괄 처리에 의해 워크플로가 트리거되도록 하려면 대신 다음 명령을 입력합니다.

```
aws glue create-trigger --workflow-name <workflow-name> --type EVENT
--name <trigger-name> --event-batching-condition BatchSize=<number-of-
events>,BatchWindow=<seconds> --actions <actions>
```

event-batching-condition 인수의 경우 BatchSize는 필수이고 BatchWindow는 선택 사항입니다. BatchWindow를 생략하면 기간의 기본값은 최대 기간 크기인 900초입니다.

Example

다음 예제에서는 3개의 EventBridge 이벤트가 도착한 후 또는 첫 번째 이벤트가 도착하고 5분 후 중 먼저 해당되는 시점에 eventtest 워크플로를 시작하는 트리거를 생성합니다.

```
aws glue create-trigger --workflow-name eventttest --type EVENT --name objectArrival
--event-batching-condition BatchSize=3,BatchWindow=300 --actions JobName=test1
```

5. Amazon EventBridge에서 규칙을 생성합니다.

- a. 원하는 텍스트 편집기에서 규칙 세부 정보에 대한 JSON 객체를 생성합니다.

다음 예에서는 Amazon S3를 이벤트 소스로, PutObject를 이벤트 이름으로, 버킷 이름을 요청 파라미터로 지정합니다. 이 규칙은 새 객체가 버킷에 도착하면 워크플로를 시작합니다.

```
{
  "source": [
    "aws.s3"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "s3.amazonaws.com"
    ],
    "eventName": [
      "PutObject"
    ],
    "requestParameters": {
      "bucketName": [
        "<bucket-name>"
      ]
    }
  }
}
```

버킷 내의 폴더에 새 객체가 도착할 때 워크플로를 시작하려면 requestParameters를 다음 코드로 대체합니다.

```
"requestParameters": {
  "bucketName": [
    "<bucket-name>"
  ]
  "key" : [{"prefix" : "<folder1>/<folder2>/*"}]}
}
```

- b. 선호하는 도구를 사용하여 규칙 JSON 객체를 이스케이프 처리된 문자열로 변환합니다.

```
{\n  \"source\": [\n    \"aws.s3\"\n  ],\n  \"detail-type\": [\n    \"AWS API\n    Call via CloudTrail\"\n  ],\n  \"detail\": {\n    \"eventSource\": [\n      \"s3.amazonaws.com\"\n    ],\n    \"eventName\": [\n      \"PutObject\"\n    ],\n    \"requestParameters\": {\n      \"bucketName\": [\n        \"<bucket-  
name>\"\n      ]\n    }\n  }\n}
```

- c. 다음 명령을 실행하여 후속 put-rule 명령에 대한 입력 파라미터를 지정하기 위해 편집할 수 있는 JSON 파라미터 템플릿을 생성합니다. 출력을 파일에 저장합니다. 이 예에서 파일 이름은 ruleCommand입니다.

```
aws events put-rule --name <rule-name> --generate-cli-skeleton >ruleCommand
```

--generate-cli-skeleton 파라미터에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서의 [JSON 또는 YAML 입력 파일에서 AWS CLI 스킴레톤 및 입력 파라미터 생성](#)을 참조하세요.

출력 파일은 다음과 같습니다.

```
{
  "Name": "",
  "ScheduleExpression": "",
  "EventPattern": "",
  "State": "ENABLED",
  "Description": "",
  "RoleArn": "",
  "Tags": [
    {
      "Key": "",
      "Value": ""
    }
  ],
  "EventBusName": ""
}
```

- d. 파일을 편집하여 선택적으로 파라미터를 제거하고 최소한 Name, EventPattern 및 State 파라미터를 지정합니다. EventPattern 파라미터에 대해 이전 단계에서 생성한 규칙 세부 정보에 대한 이스케이프 처리된 문자열을 제공합니다.

```
{
```



```

    "Name": "<rule-name>",
    "EventPattern": "{\n  \"source\": [\n    \"aws.s3\"\n  ],\n  \"detail-
type\": [\n    \"AWS API Call via CloudTrail\"\n  ],\n  \"detail\": {\n
  \"eventSource\": [\n    \"s3.amazonaws.com\"\n  ],\n  \"eventName\": [\n
    \"PutObject\"\n  ],\n  \"requestParameters\": {\n    \"bucketName
\": [\n      \"<bucket-name>\"\n    ]\n  }\n}"
    "State": "DISABLED",
    "Description": "Start an AWS Glue workflow upon new file arrival in an
Amazon S3 bucket"
  }

```

Note

워크플로 구축을 마칠 때까지 규칙을 사용 중지 상태로 두는 것이 가장 좋습니다.

- e. 파일 ruleCommand에서 입력 파라미터를 읽는 다음 put-rule 명령을 입력합니다.

```
aws events put-rule --name <rule-name> --cli-input-json file://ruleCommand
```

다음 출력은 성공을 나타냅니다.

```
{
  "RuleArn": "<rule-arn>"
}
```

6. 다음 명령을 입력하여 대상에 규칙을 연결합니다. 대상은 AWS Glue의 워크플로입니다. <role-name>을 이 절차의 시작 부분에서 생성한 역할로 바꿉니다.

```
aws events put-targets --rule <rule-name> --targets
  "Id"="1", "Arn"="arn:aws:glue:<region>:<account-id>:workflow/<workflow-
name>", "RoleArn"="arn:aws:iam:<account-id>:role/<role-name>" --region <region>
```

다음 출력은 성공을 나타냅니다.

```
{
  "FailedEntryCount": 0,
  "FailedEntries": []
}
```

7. 다음 명령을 입력하여 규칙과 대상이 성공적으로 연결되었는지 확인합니다.

```
aws events list-rule-names-by-target --target-arn arn:aws:glue:<region>:<account-id>:workflow/<workflow-name>
```

다음 출력은 성공을 나타냅니다. 여기서 *<rule-name>*은 생성한 규칙의 이름입니다.

```
{
  "RuleNames": [
    "<rule-name>"
  ]
}
```

8. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
9. 워크플로를 선택하고 시작 트리거와 해당 작업(시작되는 작업 또는 크롤러)이 워크플로 그래프에 나타나는지 확인합니다. 그리고 [3단계: 트리거 추가](#)의 절차로 진행합니다. 또는 AWS Glue API나 AWS Command Line Interface를 사용하여 워크플로에 구성 요소를 더 추가합니다.
10. 워크플로가 완전히 지정되면 규칙을 사용합니다.

```
aws events enable-rule --name <rule-name>
```

이제 EventBridge 이벤트 또는 이벤트 배치로 워크플로를 시작할 준비가 되었습니다.

i 다음 사항도 참조하십시오.

- [Amazon EventBridge 사용 설명서](#)
- [AWS Glue의 워크플로 개요](#)
- [AWS Glue에서 워크플로 수동 생성 및 구축](#)

워크플로를 시작한 EventBridge 이벤트 보기

워크플로를 시작한 Amazon EventBridge 이벤트의 이벤트 ID를 볼 수 있습니다. 워크플로가 이벤트 배치로 시작된 경우 배치에 있는 모든 이벤트의 이벤트 ID를 볼 수 있습니다.

배치 크기가 1보다 큰 워크플로의 경우 어떤 배치 조건이 워크플로를 시작했는지 확인할 수도 있습니다(배치 크기의 이벤트 수 도착 또는 배치 기간 만료).

워크플로를 시작한 EventBridge 이벤트를 보려면(콘솔)

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 [워크플로(Workflows)]를 선택합니다.
3. 워크플로를 선택합니다. 그런 다음 하단에서 [기록(History)] 탭을 선택합니다.
4. 워크플로 실행을 선택한 다음 [실행 세부 정보 보기(View run details)]를 선택합니다.
5. 실행 세부 정보 페이지에서 [실행 속성(Run properties)] 필드를 찾아 [aws:eventIds] 키를 찾습니다.


해당 키의 값은 EventBridge 이벤트 ID 목록입니다.

워크플로를 시작한 EventBridge 이벤트를 보려면(AWS API)

- Python 스크립트에 다음 코드를 포함합니다.

```
workflow_params =
    glue_client.get_workflow_run_properties(Name=workflow_name, RunId=workflow_run_id)
batched_events = workflow_params['aws:eventIds']
```

batched_events는 문자열 목록이며 각 문자열은 이벤트 ID입니다.

 다음 사항도 참조하십시오.

- [Amazon EventBridge 사용 설명서](#)
- [the section called “워크플로우 개요”](#)

AWS Glue에서 워크플로 실행 및 모니터링

워크플로우의 시작 트리거가 온디맨드 트리거인 경우, AWS Glue 콘솔에서 워크플로우를 시작할 수 있습니다. 워크플로를 실행하고 모니터링하려면 다음 단계를 수행합니다. 워크플로가 실패하면 실행 그 래프를 보고 실패한 노드를 확인할 수 있습니다. 블루프린트에서 워크플로가 생성된 경우 문제를 해결 하기 위해 블루프린트 실행을 보고 워크플로를 생성하는 데 사용된 블루프린트 파라미터 값을 볼 수 있습니다. 자세한 내용은 [the section called “블루프린트 실행 보기”](#) 단원을 참조하십시오.

AWS Glue 콘솔, API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 워크플로를 실행하고 모니터링할 수 있습니다.

워크플로를 실행하고 모니터링하려면(콘솔)

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 ETL 아래에서 Workflows(워크플로우)를 선택합니다.
3. 워크플로우를 선택합니다. Actions(작업) 메뉴에서 Run(실행)을 선택합니다.
4. 워크플로 목록에서 [마지막 실행 상태(Last run status)] 열을 확인합니다. 새로 고침 버튼을 선택하여 진행 중인 워크플로 상태를 봅니다.
5. 워크플로가 실행 중이거나 완료된 후(또는 실패한 후) 다음 단계를 완료하여 실행 세부 정보를 확인합니다.
 - a. 워크플로가 선택되어 있는지 확인하고 [기록(History)] 탭을 선택합니다.
 - b. 현재 또는 가장 최근 워크플로 실행을 선택한 다음 [실행 세부 정보 보기(View run details)]를 선택합니다.

워크플로 런타임 그래프는 현재 실행 상태를 보여줍니다.

- c. 그래프에서 노드를 선택하여 노드의 세부 정보와 상태를 봅니다.

The screenshot displays the AWS Glue console interface. On the left, a workflow graph is shown with three nodes: a green circle labeled 'myDemoBPWorkflow1_starti...', a red square labeled 'myDemoBPWorkflow1_etl_j...', and a grey diamond labeled 'myDemoBPWorkflow1_myD...'. The red square node is highlighted with a blue border and a red 'X' icon, indicating it has failed. A 'Resume run' button is visible in the top right of the graph area. On the right side, the 'Job details' panel is open, showing the 'Selected run' as 'Tue, 21 Jul 2020 19:55:10 GMT - FAILED'. Below this, the job details are listed: Name (myDemoBPWorkflow1_etl_jo), Description (-), Job run id (jr_8e74182b093deea6bf63d), Status (Failed), Resume (checkbox), Retry attempt (-), Job run error (Error: Invalid argument type), Execution time (28), Start time (Tue, 21 Jul 2020 19:55:10 G), and End time (Tue, 21 Jul 2020 20:21:17 G).

워크플로를 실행하고 모니터링하려면(AWS CLI)

1. 다음 명령을 입력합니다. `<workflow-name>`을 실행할 워크플로우로 바꿉니다.

```
aws glue start-workflow-run --name <workflow-name>
```


워크플로우가 성공적으로 시작되면 명령은 실행 ID를 반환합니다.

2. `get-workflow-run` 명령을 사용하여 워크플로 실행 상태를 봅니다. 워크플로 이름과 실행 ID를 제공합니다.

```
aws glue get-workflow-run --name myWorkflow --run-id
wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705
```

다음은 샘플 명령 출력입니다.

```
{
  "Run": {
    "Name": "myWorkflow",
    "WorkflowRunId":
"wr_d2af14217e8eae775ba7b1fc6fc7a42c795aed3cbcd8763f9415452e2dbc8705",
    "WorkflowRunProperties": {
      "run_state": "COMPLETED",
      "unique_id": "fee63f30-c512-4742-a9b1-7c8183bdaae2"
    },
    "StartedOn": 1578556843.049,
    "CompletedOn": 1578558649.928,
    "Status": "COMPLETED",
    "Statistics": {
      "TotalActions": 11,
      "TimeoutActions": 0,
      "FailedActions": 0,
      "StoppedActions": 0,
      "SucceededActions": 9,
      "RunningActions": 0,
      "ErroredActions": 0
    }
  }
}
```

 다음 사항도 참조하세요.

- [the section called “워크플로우 개요”](#)
- [the section called “블루프린트 개요”](#)

워크플로 실행 중지

AWS Glue 콘솔, AWS Command Line Interface(AWS CLI) 또는 AWS Glue API를 사용하여 워크플로우 실행을 중지할 수 있습니다. 워크플로우 실행을 중지하면 실행 중인 모든 작업과 크롤러가 즉시 종료되고 아직 시작되지 않은 작업과 크롤러는 시작되지 않습니다. 실행 중인 모든 작업 및 크롤러가 중지되려면 최대 1분이 걸릴 수 있습니다. 워크플로우 실행 상태가 Running(실행 중)에서 Stopping(중지 중)으로 바뀌고, 워크플로우 실행이 완전히 중지되면 상태가 Stopped(중지됨)로 바뀝니다.

워크플로우 실행이 중지된 후 실행 그래프를 보고 완료된 작업 및 크롤러와 시작되지 않은 작업 및 크롤러를 확인할 수 있습니다. 그런 다음 데이터 무결성을 보장하기 위한 단계를 수행해야 하는지 여부를 결정할 수 있습니다. 워크플로우 실행을 중지하면 자동 롤백 작업이 수행되지 않습니다.

워크플로우 실행을 중지하려면(콘솔)

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 ETL 아래에서 Workflows(워크플로우)를 선택합니다.
3. 실행 중인 워크플로우를 선택한 다음 History(기록) 탭을 선택합니다.
4. 워크플로우 실행을 선택한 다음 Stop run(실행 중지)을 선택합니다.

실행 상태가 Stopping(중지 중)으로 변경됩니다.

5. (선택 사항) 워크플로우 실행을 선택하고 View run details(실행 세부 정보 보기)를 선택한 다음 실행 그래프를 검토합니다.

워크플로우 실행을 중지하려면(AWS CLI)

- 다음 명령을 입력합니다. `<workflow-name>`을 워크플로우의 이름으로 바꾸고 `<run-id>`를 중지할 워크플로우 실행의 실행 ID로 바꿉니다.

```
aws glue stop-workflow-run --name <workflow-name> --run-id <run-id>
```

다음은 stop-workflow-run 명령의 예입니다.

```
aws glue stop-workflow-run --name my-workflow --run-id
wr_137b88917411d128081069901e4a80595d97f719282094b7f271d09576770354
```

워크플로 실행 복구 및 재개

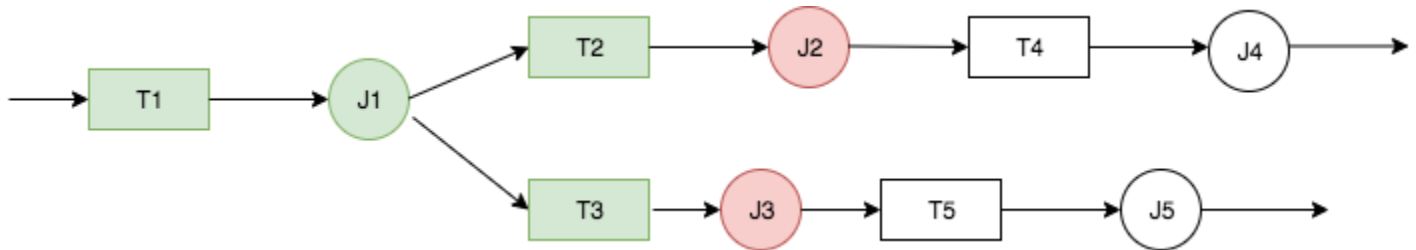
워크플로에서 하나 이상의 노드(작업 또는 크롤러)가 성공적으로 완료되지 않으면 워크플로가 부분적으로만 실행된 것입니다. 근본 원인을 찾아 수정한 후 워크플로 실행을 재개할 노드를 하나 이상 선택한 다음 워크플로 실행을 재개할 수 있습니다. 그런 다음 선택한 노드와 해당 노드에서 다운스트림인 모든 노드가 실행됩니다.

주제

- [워크플로 실행 재개: 작동 방식](#)
- [워크플로 실행 재개](#)
- [워크플로 실행 재개에 대한 참고 사항 및 제한 사항](#)

워크플로 실행 재개: 작동 방식

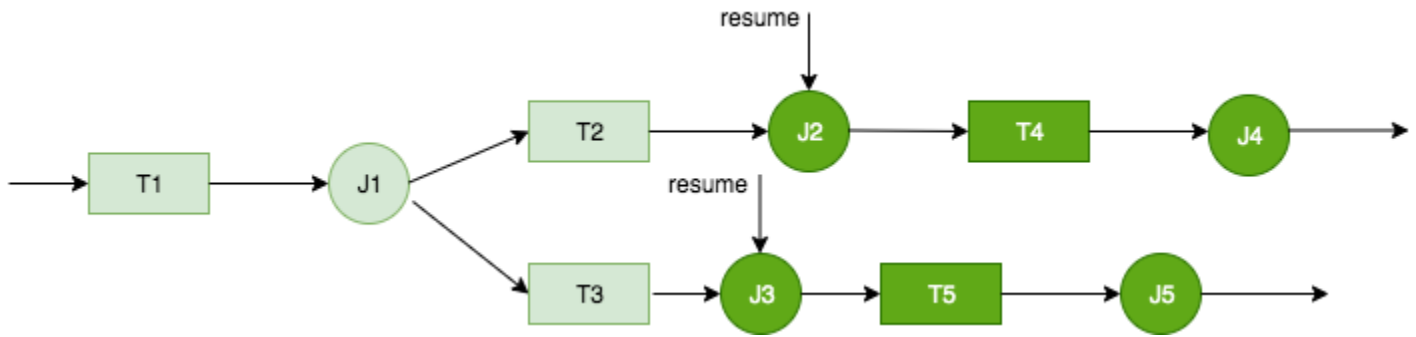
다음 다이어그램에서 워크플로 W1을 고려합니다.



워크플로 실행은 다음과 같이 진행됩니다.

1. 트리거 T1이 작업 J1을 시작합니다.
2. J1 발생이 성공적으로 완료되면 각각 J2 및 J3 작업을 실행하는 T2 및 T3이 트리거됩니다.
3. 작업 J2 및 J3이 실패합니다.
4. 트리거 T4 및 T5는 J2 및 J3의 성공적인 완료에 따라 달라지므로 발생하지 않고 작업 J4 및 J5가 실행되지 않습니다. 워크플로 W1이 부분적으로만 실행됩니다.

이제 J2 및 J3을 실패하게 만든 문제가 수정되었다고 가정합니다. J2 및 J3이 워크플로 실행을 재개할 시작점으로 선택됩니다.



워크플로 실행은 다음과 같이 재개됩니다.

1. 작업 J2 및 J3이 성공적으로 실행됩니다.
2. T4 및 T5 발생을 트리거합니다.
3. 작업 J4 및 J5가 성공적으로 실행됩니다.

재개된 워크플로 실행은 새 실행 ID를 가진 별도의 워크플로 실행으로 추적됩니다. 워크플로 기록을 볼 때 모든 워크플로 실행에 대한 이전 실행 ID를 볼 수 있습니다. 다음 스크린샷의 예에서 실행 ID `wr_c7a22...`(두 번째 행)로 실행된 워크플로에 완료되지 않은 노드가 있습니다. 사용자가 문제를 수정하고 워크플로 실행을 재개하여 실행 ID `wr_a07e55...`(첫 번째 행)가 생성되었습니다.

Run ID	Previous run ID	Run status	Execution time
<code>wr_a07e55f2087afdd415a404403f644a4265278...</code>	<code>wr_c7a2219a8dc412f1366a5b30df3c58be30b9...</code>	Completed	17 Minutes
<code>wr_c7a2219a8dc412f1366a5b30df3c58be30b9...</code>	-	Completed	8 Minutes

Note

이 논의의 나머지 부분에서 "재개된 워크플로 실행"이라는 용어는 이전 워크플로 실행이 재개 될 때 생성된 워크플로 실행을 나타냅니다. "원래 워크플로 실행"은 부분적으로만 실행되어 재개해야 했던 워크플로 실행을 나타냅니다.

재개된 워크플로 실행 그래프

재개된 워크플로 실행에서는 노드의 하위 집합만 실행되지만 실행 그래프는 전체 그래프입니다. 즉, 재개된 워크플로에서 실행되지 않은 노드는 원래 워크플로 실행의 실행 그래프에서 복사됩니다. 원래 워크플로 실행에서 실행된 복사된 작업 및 크롤러 노드에는 실행 세부 정보가 포함됩니다.

이전 다이어그램의 워크플로 W1을 다시 고려하세요. 워크플로 실행이 J2 및 J3부터 재개되면 재개된 워크플로 실행에 대한 실행 그래프에 모든 작업(J1~J5)과 모든 트리거(T1~T5)가 표시됩니다. J1에 대한 실행 세부 정보는 원래 워크플로 실행에서 복사됩니다.

워크플로 실행 스냅샷

워크플로 실행이 시작되면 AWS Glue는 해당 시점의 워크플로 설계 그래프의 스냅샷을 만듭니다. 해당 스냅샷은 워크플로 실행 기간 동안 사용됩니다. 실행이 시작된 후 트리거를 변경하는 경우 해당 변경 사항이 현재 워크플로 실행에 영향을 주지 않습니다. 스냅샷은 워크플로 실행이 일관된 방식으로 진행되도록 합니다.

스냅샷은 트리거만 변경 불가능하게 만듭니다. 워크플로 실행 중 다운스트림 작업 및 크롤러에 대한 변경 사항은 현재 실행에 적용됩니다.

워크플로 실행 재개

워크플로 실행을 재개하려면 다음 단계를 따릅니다. AWS Glue 콘솔, API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 워크플로 실행을 재개할 수 있습니다.

워크플로 실행을 재개하려면(콘솔)

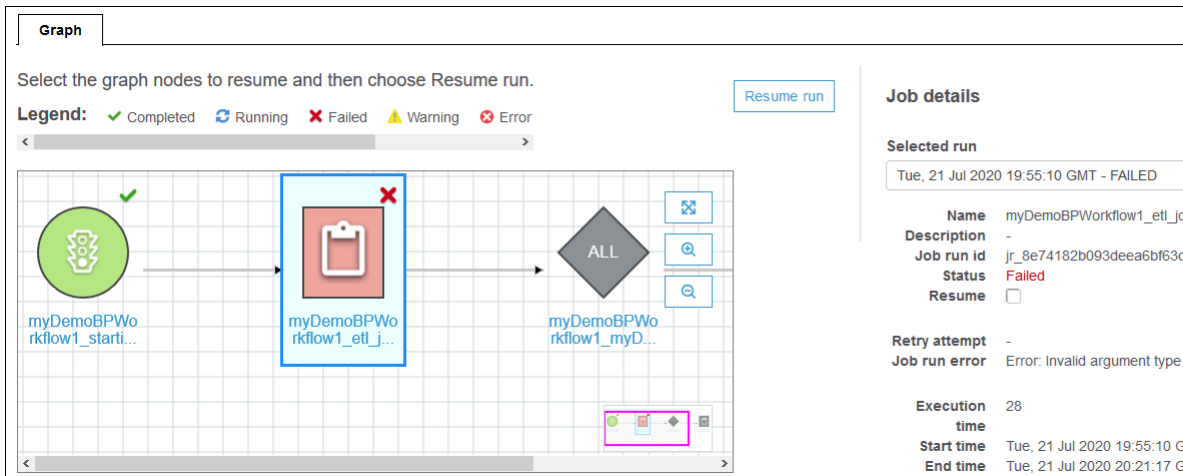
1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.

워크플로를 보고 워크플로 실행을 재개할 수 있는 권한이 있는 사용자로 로그인합니다.

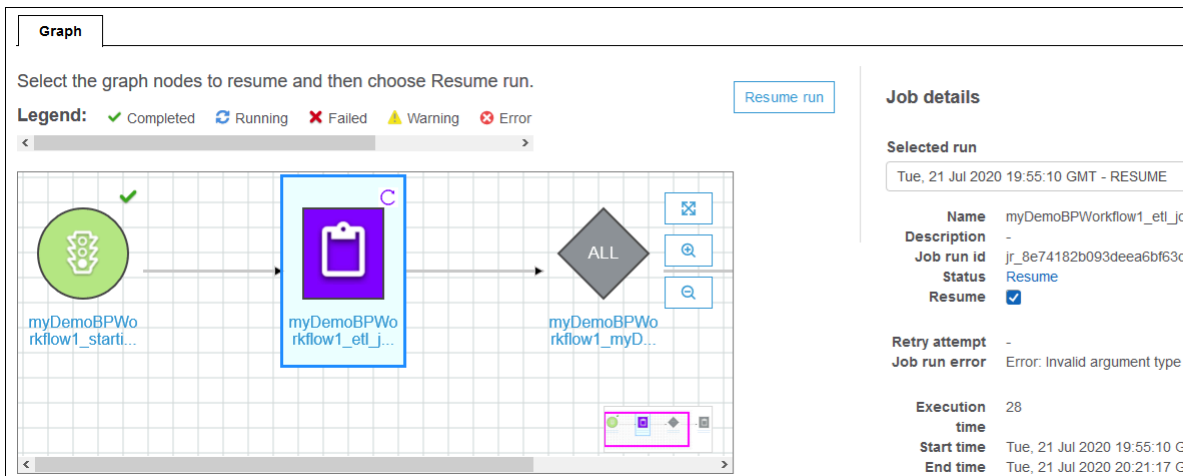
Note

워크플로 실행을 재개하려면 `glue:ResumeWorkflowRun` AWS Identity and Access Management(IAM) 권한이 필요합니다.

2. 탐색 창에서 [워크플로(Workflows)]를 선택합니다.
3. 워크플로를 선택한 다음 [기록(History)] 탭을 선택합니다.
4. 부분적으로만 실행된 워크플로 실행을 선택한 다음 [실행 세부 정보 보기(View run details)]를 선택합니다.
5. 실행 그래프에서 다시 시작하고 워크플로 실행을 재개할 첫 번째(또는 유일한) 노드를 선택합니다.
6. 그래프 오른쪽에 있는 세부 정보 창에서 [재개(Resume)] 확인란을 선택합니다.



노드의 색상이 변경되고 오른쪽 상단에 작은 이력서 아이콘이 표시됩니다.



7. 추가 노드를 다시 시작하려면 이전 두 단계를 완료합니다.
8. [실행 재개(Resume run)]를 선택합니다.

워크플로 실행을 재개하려면(AWS CLI)

1. glue:ResumeWorkflowRun IAM 권한이 있는지 확인합니다.
2. 다시 시작하려는 노드의 노드 ID를 검색합니다.
 - a. 원래 워크플로 실행에 대해 get-workflow-run 명령을 실행합니다. 다음 예와 같이 워크플로 이름과 실행 ID를 제공하고 --include-graph 옵션을 추가합니다. 콘솔의 [기록 (History)] 탭에서 또는 get-workflow 명령을 실행하여 실행 ID를 가져옵니다.

```
aws glue get-workflow-run --name cloudtrailtest1 --run-id
wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --include-
graph
```

이 명령은 그래프의 노드와 가장자리를 큰 JSON 객체로 반환합니다.

- b. 노드 객체의 Type 및 Name 속성으로 관심 노드를 찾습니다.

다음은 출력의 예제 노드 객체입니다.

```
{
  "Type": "JOB",
  "Name": "test1_post_failure_4592978",
  "UniqueId":
  "wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd",
  "JobDetails": {
    "JobRuns": [
      {
        "Id":
        "jr_690b9f7fc5cb399204bc542c6c956f39934496a5d665a42de891e5b01f59e613",
        "Attempt": 0,
        "TriggerName": "test1_aggregate_failure_649b2432",
        "JobName": "test1_post_failure_4592978",
        "StartedOn": 1595358275.375,
        "LastModifiedOn": 1595358298.785,
        "CompletedOn": 1595358298.785,
        "JobRunState": "FAILED",
        "PredecessorRuns": [],
        "AllocatedCapacity": 0,
        "ExecutionTime": 16,
        "Timeout": 2880,
        "MaxCapacity": 0.0625,
        "LogGroupName": "/aws-glue/python-jobs"
      }
    ]
  }
}
```

- c. 노드 객체의 UniqueId 속성에서 노드 ID를 가져옵니다.

3. resume-workflow-run 명령을 실행합니다. 다음 예와 같이 워크플로 이름, 실행 ID 및 노드 ID 목록을 공백으로 구분하여 제공합니다.

```
aws glue resume-workflow-run --name cloudtrailtest1 --run-id
wr_a07e55f2087afdd415a404403f644a4265278f68b13ba3da08c71924ebe3c3a8 --node-
ids wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3
wnode_d1b2563c503078b153142ee76ce545fe5ceef66e053628a786ddd74a05da86fd
```

이 명령은 재개된(새) 워크플로 실행의 실행 ID와 시작될 노드 목록을 출력합니다.

```
{
  "RunId": "wr_2ada0d3209a262fc1156e4291134b3bd643491bcfb0ceead30bd3e4efac24de9",
  "NodeIds": [
    "wnode_ca1f63e918fb855e063aed2f42ec5762ccf71b80082ae2eb5daeb8052442f2f3"
  ]
}
```

예제 `resume-workflow-run` 명령은 다시 시작할 2개의 노드를 나열했지만 예제 출력은 1개의 노드만 다시 시작됨을 나타냅니다. 이는 한 노드가 다른 노드의 다운스트림이었고, 다운스트림 노드는 워크플로의 정상적인 흐름에 따라 다시 시작되기 때문입니다.

워크플로 실행 재개에 대한 참고 사항 및 제한 사항

워크플로 실행을 재개할 때 다음 참고 사항과 제한 사항을 염두에 둡니다.

- COMPLETED 상태인 경우에만 워크플로 실행을 재개할 수 있습니다.

Note

워크플로 실행에서 하나 이상의 노드가 완료되지 않은 경우에도 워크플로 실행 상태는 COMPLETED로 표시됩니다. 성공적으로 완료되지 않은 노드를 검색하려면 실행 그래프를 확인합니다.

- 원래 워크플로 실행이 실행을 시도한 모든 작업 또는 크롤러 노드에서 워크플로 실행을 재개할 수 있습니다. 트리거 노드에서 실행되는 워크플로를 재개할 수 없습니다.
- 노드를 다시 시작해도 상태가 재설정되지 않습니다. 부분적으로 처리된 데이터는 롤백되지 않습니다.
- 동일한 워크플로 실행을 여러 번 재개할 수 있습니다. 재개된 워크플로 실행이 부분적으로만 실행되는 경우 문제를 해결하고 재개된 실행을 재개할 수 있습니다.

- 다시 시작할 노드를 2개 선택하는 경우 이들이 서로 종속되어 있으면 업스트림 노드가 다운스트림 노드보다 먼저 실행됩니다. 실제로 다운스트림 노드를 선택하는 것은 일반적인 워크플로 흐름에 따라 실행되기 때문에 중복됩니다.

AWS Glue에서 워크플로 실행 속성 가져오기 및 설정

워크플로 실행 속성을 사용하여 AWS Glue 워크플로우의 작업 간에 상태를 공유 및 관리합니다. 워크플로를 생성할 때 기본 실행 속성을 설정할 수 있습니다. 그런 다음, 작업이 실행될 때 실행 속성 값을 검색하고 나중에 워크플로에 있는 작업에 대한 입력을 위해 필요에 따라 수정할 수 있습니다. 작업이 실행 속성을 수정할 때 새 값은 워크플로 실행에만 존재합니다. 기본 실행 속성은 영향을 받지 않습니다.

AWSGlue 작업이 워크플로의 일부가 아닌 경우 이러한 속성은 설정되지 않습니다.

ETL(추출, 변환 및 로드) 작업의 다음 샘플 Python 코드는 워크플로우 실행 속성을 가져오는 방법을 보여 줍니다.

```
import sys
import boto3
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from pyspark.context import SparkContext

glue_client = boto3.client("glue")
args = getResolvedOptions(sys.argv, ['JOB_NAME', 'WORKFLOW_NAME', 'WORKFLOW_RUN_ID'])
workflow_name = args['WORKFLOW_NAME']
workflow_run_id = args['WORKFLOW_RUN_ID']
workflow_params = glue_client.get_workflow_run_properties(Name=workflow_name,
                                                         RunId=workflow_run_id)["RunProperties"]

target_database = workflow_params['target_database']
target_s3_location = workflow_params['target_s3_location']
```

다음 코드는 target_format 실행 속성을 'csv'로 설정하여 계속합니다.

```
workflow_params['target_format'] = 'csv'
glue_client.put_workflow_run_properties(Name=workflow_name, RunId=workflow_run_id,
                                       RunProperties=workflow_params)
```

자세한 내용은 다음을 참조하세요.

- [GetWorkflowRunProperties](#) 작업(Python: `get_workflow_run_properties`)
- [PutWorkflowRunProperties](#) 작업(Python: `put_workflow_run_properties`)

AWS Glue API를 사용하여 워크플로 쿼리

AWS Glue는 워크플로우를 관리하기 위한 다양한 API를 제공합니다. AWS Glue API를 사용하여 워크플로우의 정적 보기 또는 실행 중인 워크플로우의 동적 보기를 검색할 수 있습니다. 자세한 내용은 [워크플로](#) 단원을 참조하십시오.

주제

- [정적 보기 쿼리](#)
- [동적 보기 쿼리](#)

정적 보기 쿼리

GetWorkflow API 작업을 사용하여 워크플로우의 설계를 나타내는 정적 보기를 가져옵니다. 이 작업은 노드와 엣지로 구성된 방향성 그래프를 반환합니다. 여기에서 노드는 트리거, 작업 또는 크롤러를 나타냅니다. 엣지는 노드 간의 관계를 정의합니다. 엣지는 AWS Glue 콘솔의 그래프에서 커넥터(화살표)로 표현됩니다.

NetworkX, igraph, JGraphT, Java Universal Network/Graph(JUNG) Framework 등의 인기 있는 그래프 처리 라이브러리에서 이 작업을 사용할 수도 있습니다. 이러한 모든 라이브러리는 그래프를 비슷하게 나타내기 때문에 최소한의 변환이 필요합니다.

이 API가 반환하는 정적 보기는 워크플로우와 연결된 트리거의 최신 정의에 따라 최신 보기입니다.

그래프 정의

워크플로우 그래프 G 는 정렬된 페어 (N, E) 입니다. 여기서 N 은 노드 세트이고 E 는 엣지 세트입니다. 노드는 고유한 숫자로 식별되는 그래프의 꼭짓점입니다. 노드는 트리거, 작업 또는 크롤러 유형일 수 있습니다. 예: `{name:T1, type:Trigger, uniqueId:1}`, `{name:J1, type:Job, uniqueId:2}`.

엣지는 2튜플의 형식 $(src, dest)$ 입니다. 여기에서 src 와 $dest$ 는 노드이고 src 에서 $dest$ 로의 방향성 엣지가 있습니다.

정적 보기 쿼리의 예

작업 J1을 완료하면 작업 J2를 트리거하는 조건부 트리거 T를 생각해 보십시오.

```
J1 ----> T ----> J2
```

노드: J1, T, J2

엣지: (J1, T), (T, J2)

동적 보기 쿼리

GetWorkflowRun API 작업을 사용하여 실행 중인 워크플로우의 동적 보기를 가져옵니다. 이 작업은 워크플로우 실행과 관련된 메타데이터와 함께 그래프의 동일한 정적 보기를 반환합니다.

실행의 경우 GetWorkflowRun 호출의 작업을 나타내는 노드에는 최신 워크플로우 실행의 일부로 시작되는 작업 실행 목록이 있습니다. 이 목록을 사용하여 그래프 자체에 각 작업의 실행 상태를 표시할 수 있습니다. 아직 실행되지 않은 다운스트림 종속성의 경우 이 필드는 null로 설정됩니다. 그래프로 표시된 정보를 통해 언제든지 워크플로우의 현재 상태를 알 수 있습니다.

이 API가 반환하는 동적 보기는 워크플로 실행이 시작될 때 있었던 정적 보기를 기반으로 합니다.

런타임 노드 예: {name:T1, type: Trigger, uniqueId:1}, {name:J1, type:Job, uniqueId:2, jobDetails:{jobRuns}}, {name:C1, type:Crawler, uniqueId:3, crawlerDetails:{crawls}}

예 1: 동적 보기

다음 예에서는 간단한 2 트리거 워크플로우를 보여 줍니다.

- 노드: t1, j1, t2, j2
- 엣지: (t1, j1), (j1, t2), (t2, j2)

GetWorkflow 응답에는 다음이 포함됩니다.

```
{
  Nodes : [
    {
      "type" : Trigger,
      "name" : "t1",
      "uniqueId" : 1
    },
    {
      "type" : Job,
```

```

        "name" : "j1",
        "uniqueId" : 2
    },
    {
        "type" : Trigger,
        "name" : "t2",
        "uniqueId" : 3
    },
    {
        "type" : Job,
        "name" : "j2",
        "uniqueId" : 4
    }
],
Edges : [
    {
        "sourceId" : 1,
        "destinationId" : 2
    },
    {
        "sourceId" : 2,
        "destinationId" : 3
    },
    {
        "sourceId" : 3,
        "destinationId" : 4
    }
}

```

GetWorkflowRun 응답에는 다음이 포함됩니다.

```

{
  Nodes : [
    {
      "type" : Trigger,
      "name" : "t1",
      "uniqueId" : 1,
      "jobDetails" : null,
      "crawlerDetails" : null
    },
    {
      "type" : Job,
      "name" : "j1",

```



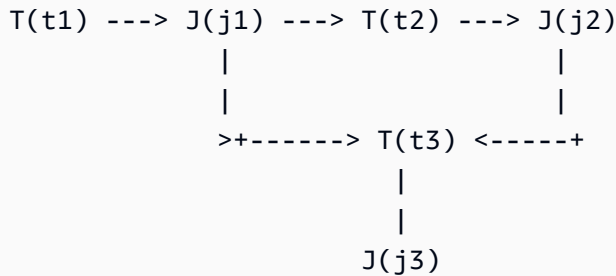
```
    "uniqueId" : 2,
    "jobDetails" : [
      {
        "id" : "jr_12334",
        "jobRunState" : "SUCCEEDED",
        "errorMessage" : "error string"
      }
    ],
    "crawlerDetails" : null
  },
  {
    "type" : Trigger,
    "name" : "t2",
    "uniqueId" : 3,
    "jobDetails" : null,
    "crawlerDetails" : null
  },
  {
    "type" : Job,
    "name" : "j2",
    "uniqueId" : 4,
    "jobDetails" : [
      {
        "id" : "jr_1233sdf4",
        "jobRunState" : "SUCCEEDED",
        "errorMessage" : "error string"
      }
    ],
    "crawlerDetails" : null
  }
],
Edges : [
  {
    "sourceId" : 1,
    "destinationId" : 2
  },
  {
    "sourceId" : 2,
    "destinationId" : 3
  },
  {
    "sourceId" : 3,
    "destinationId" : 4
  }
]
```

}

예 2: 조건부 트리거가 있는 여러 작업

다음 예에서는 작업 여러 개와 조건부 트리거(t3) 하나가 있는 워크플로우를 보여 줍니다.

Consider Flow:



Graph generated:

Nodes: t1, t2, t3, j1, j2, j3

Edges: (t1, j1), (j1, t2), (t2, j2), (j1, t3), (j2, t3), (t3, j3)

AWS Glue의 블루프린트 및 워크플로 제한 사항

다음은 블루프린트와 워크플로에 대한 제한 사항입니다.

블루프린트 제한 사항

다음 블루프린트 제한에 유의하세요.

- 블루프린트는 Amazon S3 버킷이 있는 동일한 AWS 리전에 등록되어야 합니다.
- AWS 계정 간에 블루프린트를 공유하려면 Amazon S3의 블루프린트 ZIP 아카이브에 대한 읽기 권한을 부여해야 합니다. 블루프린트 ZIP 아카이브에 대한 읽기 권한이 있는 고객은 AWS 계정에 블루프린트를 등록하여 사용할 수 있습니다.
- 블루프린트 파라미터 집합은 단일 JSON 객체로 저장됩니다. 이 객체의 최대 길이는 128KB입니다.
- 블루프린트 ZIP 아카이브의 압축되지 않은 최대 크기는 5MB입니다. 압축된 최대 크기는 1MB입니다.
- 워크플로 내 작업, 크롤러, 트리거의 총수를 100개 이하로 제한합니다. 100개가 넘게 포함될 경우 워크플로 실행을 재개하거나 중지하려고 할 때 오류가 발생할 수 있습니다.

워크플로 제한 사항

다음 워크플로 제한 사항에 유의하세요. 이러한 설명 중 일부는 수동으로 워크플로를 생성하는 사용자에 대한 것입니다.

- Amazon EventBridge 이벤트 트리거에 대한 최대 배치 크기는 100입니다. 최대 기간 크기는 900초 (15분)입니다.
- 하나의 트리거는 하나의 워크플로우에만 연결할 수 있습니다.
- 시작 트리거(온디맨드 또는 일정)는 하나만 허용됩니다.
- 워크플로의 작업 또는 크롤러가 워크플로 외부의 트리거에 의해 시작된 경우, 작업 또는 크롤러 완료 (성공 또는 기타)에 종속된 워크플로 내의 트리거는 실행되지 않습니다.
- 마찬가지로 워크플로의 작업 또는 크롤러에 워크플로 내부와 워크플로 외부에서 작업 또는 크롤러 완료(성공 또는 기타)에 종속된 트리거가 있는 경우에는 작업 또는 크롤러가 워크플로 내에서 시작되면 작업 또는 크롤러 완료 시 워크플로 내부의 트리거만 실행됩니다.

AWS Glue의 블루프린트 오류 해결

AWS Glue 블루프린트를 사용할 때 오류가 발생하면 다음 해결 방법에 따라 문제의 원인을 찾아 해결할 수 있습니다.

주제

- [오류: PySpark 모듈 누락](#)
- [오류: 블루프린트 Config 파일 누락](#)
- [오류: 가져온 파일 누락](#)
- [오류: 리소스에서 iamPassRole을 수행할 권한이 없음](#)
- [오류: 잘못된 cron 일정](#)
- [오류: 같은 이름의 트리거가 이미 있음](#)
- [오류: 이름이 foo인 워크플로가 이미 있습니다.](#)
- [오류: 지정된 layoutGenerator 경로에서 모듈을 찾을 수 없음](#)
- [오류: 연결 필드에 검증 오류](#)

오류: PySpark 모듈 누락

AWS Glue에서 [레이아웃 생성기 함수 `ModuleNotFoundError`를 실행하는 동안 알 수 없는 오류 발생: 'pyspark'라는 모듈 없음(Unknown error executing layout generator function `ModuleNotFoundError: No module named 'pyspark'`)]이라는 오류를 반환합니다.

블루프린트 아카이브의 압축을 풀면 다음 중 하나와 같을 수 있습니다.

```
$ unzip compaction.zip
Archive:  compaction.zip
  creating:  compaction/
  inflating:  compaction/blueprint.cfg
  inflating:  compaction/layout.py
  inflating:  compaction/README.md
  inflating:  compaction/compaction.py

$ unzip compaction.zip
Archive:  compaction.zip
  inflating:  blueprint.cfg
  inflating:  compaction.py
  inflating:  layout.py
  inflating:  README.md
```

첫 번째 사례에서는 블루프린트와 관련된 모든 파일이 `compaction`이라는 폴더 아래에 배치된 다음 `compaction.zip`이라는 zip 파일로 변환되었습니다.

두 번째 사례에서는 블루프린트에 필요한 모든 파일이 폴더에 포함되지 않고 zip 파일인 `compaction.zip` 아래에 루트 파일로 추가되었습니다.

위 포맷 중 하나로 파일을 생성할 수 있습니다. 그러나 `blueprint.cfg`에 레이아웃을 생성하는 스크립트의 함수 이름에 대한 올바른 경로가 있는지 확인합니다.

예제

사례 1: 다음과 같이 `blueprint.cfg`에 `layoutGenerator`가 있어야 합니다.

```
layoutGenerator": "compaction.layout.generate_layout"
```

사례 2: 다음과 같이 `blueprint.cfg`에 `layoutGenerator`가 있어야 합니다.

```
layoutGenerator": "layout.generate_layout"
```

이 경로가 올바르게 포함되지 않은 경우 표시된 대로 오류가 발생할 수 있습니다. 예를 들어 사례 2와 같은 폴더 구조를 가지고 있고 사례 1과 같이 layoutGenerator가 표시되어 있다면 위의 오류가 발생할 수 있습니다.

오류: 블루프린트 Config 파일 누락

AWS Glue에서 [레이아웃 생성기 함수 FileNotFoundError를 실행하는 동안 알 수 없는 오류 발생: [Errno 2] 해당 파일 또는 디렉터리 없음: '/tmp/compaction/blueprint.cfg'(Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory: '/tmp/compaction/blueprint.cfg')]라는 오류를 반환합니다.

blueprint.cfg는 ZIP 아카이브의 루트 수준이나 ZIP 아카이브와 같은 이름의 폴더 내에 있어야 합니다.

블루프린트 ZIP 아카이브를 추출할 때 blueprint.cfg가 다음 경로 중 하나에 있어야 합니다. 다음 경로 중 하나에서 없으면 위의 오류가 발생할 수 있습니다.

```
$ unzip compaction.zip
Archive:  compaction.zip
   creating:  compaction/
  inflating:  compaction/blueprint.cfg

$ unzip compaction.zip
Archive:  compaction.zip
  inflating:  blueprint.cfg
```

오류: 가져온 파일 누락

AWS Glue에서 [레이아웃 생성기 함수 FileNotFoundError를 실행하는 동안 알 수 없는 오류 발생: [Errno 2] 해당 파일 또는 디렉터리 없음: * *demo-project/foo.py'(Unknown error executing layout generator function FileNotFoundError: [Errno 2] No such file or directory: * *demo-project/foo.py')]이라는 오류를 반환합니다.

레이아웃 생성 스크립트에 다른 파일을 읽을 수 있는 기능이 있는 경우 가져올 파일의 전체 경로를 지정해야 합니다. 예를 들어 Conversion.py 스크립트는 Layout.py에서 참조될 수 있습니다. 자세한 정보는 [샘플 블루프린트 프로젝트를 참조하세요](#).

오류: 리소스에서 iamPassRole을 수행할 권한이 없음

AWS Glue에서 [사용자: arn:aws:sts::123456789012:assumed-role/AWSGlueServiceRole/GlueSession에게 리소스 arn:aws:iam::123456789012:role/AWSGlueServiceRole에서 iam:PassRole을 수행할 권한이 부여되지 않음(User: arn:aws:sts::123456789012:assumed-role/

AWSGlueServiceRole/GlueSession is not authorized to perform: iam:PassRole on resource: arn:aws:iam::123456789012:role/AWSGlueServiceRole)]이라는 오류를 반환합니다.

워크플로의 작업 및 크롤러가 블루프린트에서 워크플로를 생성하기 위해 전달된 역할과 동일한 역할을 수임하는 경우 블루프린트 역할 자체에 iam:PassRole 권한이 포함되어야 합니다.

워크플로의 작업 및 크롤러가 블루프린트에서 워크플로 엔터티를 생성하기 위해 전달된 역할과 다른 역할을 수임하는 경우 블루프린트 역할에 블루프린트 역할 대신 수임한 다른 역할에 대한 iam:PassRole 권한이 포함되어야 합니다.

자세한 정보는 [블루프린트 역할에 대한 권한](#)을 참조하세요.

오류: 잘못된 cron 일정

AWS Glue에서 ["일정 cron(0 0 * * * *)이 잘못되었습니다.(The schedule cron(0 0 * * * *) is invalid.)"]라는 오류를 반환합니다.

올바른 [cron](#) 표현식을 제공합니다. 자세한 내용은 [크롤러와 작업을 위한 시간 기반 일정을 참조하십시오](#).

오류: 같은 이름의 트리거가 이미 있음

AWS Glue에서 ["오류: 같은 이름의 트리거가 이미 있음(Error: a trigger with the same name already exists)"]이라는 오류를 반환합니다.

블루프린트에서는 워크플로 생성을 위해 레이아웃 스크립트에서 트리거를 정의할 필요가 없습니다. 트리거 생성은 두 작업 간에 정의된 종속성을 기반으로 블루프린트 라이브러리에서 관리합니다.

트리거의 이름은 다음과 같습니다.

- 워크플로의 시작 트리거의 경우 이름은 <workflow_name>_starting_trigger입니다.
- 하나 또는 여러 업스트림 노드의 완료에 종속되는 워크플로의 노드(작업/크롤러)의 경우, AWS Glue 는 이름이 <workflow_name>_<node_name>_trigger인 트리거를 정의합니다.

이 오류는 동일한 이름의 트리거가 이미 존재함을 의미합니다. 기존 트리거를 삭제하고 워크플로 생성을 다시 실행할 수 있습니다.

Note

워크플로를 삭제해도 워크플로 내의 노드는 삭제되지 않습니다. 워크플로가 삭제되더라도 트리거가 남아 있을 수 있습니다. 이로 인해 '워크플로가 이미 존재함' 오류가 발생하지 않을 수

있지만 워크플로를 생성하고 삭제한 다음 동일한 블루프린트에서 동일한 이름으로 다시 생성하려고 하는 경우 '트리거가 이미 존재함' 오류가 발생할 수 있습니다.

오류: 이름이 foo인 워크플로가 이미 있습니다.

워크플로 이름은 고유해야 합니다. 다른 이름으로 시도합니다.

오류: 지정된 layoutGenerator 경로에서 모듈을 찾을 수 없음

AWS Glue에서 "레이아웃 생성기 함수 ModuleNotFoundError를 실행하는 동안 알 수 없는 오류 발생: 'crawl_s3_locations'라는 모듈 없음(Unknown error executing layout generator function ModuleNotFoundError: No module named 'crawl_s3_locations')"

```
layoutGenerator": "crawl_s3_locations.layout.generate_layout"
```

예를 들어 위의 layoutGenerator 경로가 있는 경우 블루프린트 아카이브의 압축을 풀면 다음과 같아야 합니다.

```
$ unzip crawl_s3_locations.zip
Archive:  crawl_s3_locations.zip
  creating:  crawl_s3_locations/
 inflating:  crawl_s3_locations/blueprint.cfg
 inflating:  crawl_s3_locations/layout.py
 inflating:  crawl_s3_locations/README.md
```

아카이브의 압축을 풀 때 블루프린트 아카이브가 다음과 같으면 위와 같은 오류가 발생할 수 있습니다.

```
$ unzip crawl_s3_locations.zip
Archive:  crawl_s3_locations.zip
 inflating:  blueprint.cfg
 inflating:  layout.py
 inflating:  README.md
```

crawl_s3_locations라는 폴더가 없는 것을 볼 수 있으며 layoutGenerator 경로가 모듈 crawl_s3_locations를 통해 레이아웃 파일을 참조할 때 위의 오류가 발생할 수 있습니다.

오류: 연결 필드에 검증 오류

AWS Glue에서 "레이아웃 생성기 함수 TypeError를 실행하는 동안 알 수 없는 오류 발생: 키 연결에 대한 값 ['foo']는 <class 'dict'> 유형이어야 함!(Unknown error executing layout generator function

TypeError: Value ['foo'] for key Connections should be of type <class 'dict'>!"라는 오류가 발생했습니다.

검증 오류입니다. Job 클래스의 Connections 필드에 딕셔너리가 필요하고 대신 값 목록이 제공되어 오류가 발생합니다.

```
User input was list of values
Connections= ['string']

Should be a dict like the following
Connections*={'Connections': ['string']}
```

블루프린트에서 워크플로를 생성하는 동안 이러한 런타임 오류를 방지하려면 [블루프린트 테스트](#)에 설명된 대로 워크플로, 작업 및 크롤러 정의를 검증할 수 있습니다.

레이아웃 스크립트에서 AWS Glue 작업, 크롤러 및 워크플로를 정의하려면 [AWS Glue 블루프린트 클래스 참조](#)의 구문을 참조하세요.

AWS Glue 블루프린트에 대한 페르소나 및 역할의 권한

다음은 일반적인 페르소나 및 AWS Glue 블루프린트에 대한 페르소나와 역할에 대해 제안된 AWS Identity and Access Management(IAM) 권한 정책입니다.

주제

- [블루프린트 페르소나](#)
- [블루프린트 페르소나에 대한 권한](#)
- [블루프린트 역할에 대한 권한](#)

블루프린트 페르소나

다음은 일반적으로 AWS Glue 블루프린트의 수명 주기와 관련된 페르소나입니다.

페르소나	설명
AWS Glue 개발자	블루프린트를 개발, 테스트 및 게시합니다.
AWS Glue 관리자	블루프린트에 대한 권한을 등록, 유지 관리 및 부여합니다.
데이터 분석가	블루프린트를 실행하여 워크플로를 생성합니다.

자세한 내용은 [the section called “블루프린트 개요”](#) 단원을 참조하십시오.

블루프린트 페르소나에 대한 권한

다음은 각 블루프린트 페르소나에 대해 제안된 권한입니다.

블루프린트에 대한 AWS Glue 개발자 권한

AWS Glue 개발자는 블루프린트를 게시하는 데 사용되는 Amazon S3 버킷에 대한 쓰기 권한이 있어야 합니다. 종종 개발자는 블루프린트를 업로드한 후 등록합니다. 이 경우 개발자는 [the section called “블루프린트에 대한 AWS Glue 관리자 권한”](#)에 나열된 권한이 필요합니다. 또한 개발자가 블루프린트를 등록한 후 테스트하려면 [the section called “블루프린트에 대한 데이터 분석가 권한”](#)에 나열된 권한도 필요합니다.

블루프린트에 대한 AWS Glue 관리자 권한

다음 정책은 AWS Glue 블루프린트를 등록하고, 보고, 유지 관리할 수 있는 권한을 부여합니다.

Important

다음 정책에서 *<s3-bucket-name>*과 *<prefix>*를 등록할 업로드된 블루프린트 ZIP 아카이브의 Amazon S3 경로로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateBlueprint",
        "glue:UpdateBlueprint",
        "glue>DeleteBlueprint",
        "glue:GetBlueprint",
        "glue:ListBlueprints",
        "glue:BatchGetBlueprints"
      ],
      "Resource": "*"
    }
  ],
}
```

```

    "Effect": "Allow",
    "Action": [
        "s3:GetObject"
    ],
    "Resource": "arn:aws:s3:::<s3-bucket-name>/<prefix>/*"
  }
]
}

```

블루프린트에 대한 데이터 분석가 권한

다음 정책은 블루프린트를 실행하고 결과 워크플로 및 워크플로 구성 요소를 볼 수 있는 권한을 부여합니다. 또한 AWS Glue가 워크플로와 워크플로 구성 요소를 생성하기 위해 맡는 역할에 PassRole을 부여합니다.

이 정책은 모든 리소스에 대한 권한을 부여합니다. 개별 블루프린트에 대한 세분화된 액세스를 구성하려면 블루프린트 ARN에 다음 포맷을 사용합니다.

```
arn:aws:glue:<region>:<account-id>:blueprint/<blueprint-name>
```

Important

다음 정책에서 *<account-id>*를 유효한 AWS 계정으로 바꾸고 *<role-name>*을 블루프린트를 실행하는 데 사용되는 역할의 이름으로 바꿉니다. 이 역할에 필요한 권한은 [the section called “블루프린트 역할에 대한 권한”](#) 섹션을 참조하세요.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:ListBlueprints",
        "glue:GetBlueprint",
        "glue:StartBlueprintRun",
        "glue:GetBlueprintRun",
        "glue:GetBlueprintRuns",
        "glue:GetCrawler",
        "glue:ListTriggers",

```

```

        "glue:ListJobs",
        "glue:BatchGetCrawlers",
        "glue:GetTrigger",
        "glue:BatchGetWorkflows",
        "glue:BatchGetTriggers",
        "glue:BatchGetJobs",
        "glue:BatchGetBlueprints",
        "glue:GetWorkflowRun",
        "glue:GetWorkflowRuns",
        "glue:ListCrawlers",
        "glue:ListWorkflows",
        "glue:GetJob",
        "glue:GetWorkflow",
        "glue:StartWorkflowRun"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::<account-id>:role/<role-name>"
  }
]
}

```

블루프린트 역할에 대한 권한

다음은 블루프린트에서 워크플로를 생성하는 데 사용되는 IAM 역할에 대해 제안된 권한입니다. 역할이 glue.amazonaws.com과 신뢰 관계를 맺고 있어야 합니다.

Important

다음 정책에서 *<account-id>*를 유효한 AWS 계정으로 바꾸고 *<role-name>*을 역할의 이름으로 바꿉니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [

```

```

        "glue:CreateJob",
        "glue:GetCrawler",
        "glue:GetTrigger",
        "glue>DeleteCrawler",
        "glue:CreateTrigger",
        "glue>DeleteTrigger",
        "glue>DeleteJob",
        "glue:CreateWorkflow",
        "glue>DeleteWorkflow",
        "glue:GetJob",
        "glue:GetWorkflow",
        "glue:CreateCrawler"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::<account-id>:role/<role-name>"
}
]
}

```

Note

워크플로의 작업 및 크롤러가 이 역할이 아닌 다른 역할을 수임하는 경우 이 정책에는 블루프린트 역할 대신 다른 역할에 대한 iam:PassRole 권한이 포함되어야 합니다.

AWS Glue에서 블루프린트 개발

조직에는 단일 워크플로를 파라미터화하여 모두 처리할 수 있는 이점을 얻을 수 있는 유사한 ETL 사용 사례 집합이 있을 수 있습니다. 이 요구를 해결하기 위해 AWS Glue에서 워크플로를 생성하는 데 사용할 수 있는 블루프린트를 정의할 수 있습니다. 블루프린트는 파라미터를 허용하므로 단일 블루프린트에서 데이터 분석가가 유사한 ETL 사용 사례를 처리하기 위해 다양한 워크플로를 생성할 수 있습니다. 블루프린트를 생성한 후에는 다른 부서, 팀 및 프로젝트에 재사용할 수 있습니다.

주제

- [AWS Glue의 블루프린트 개요](#)
- [AWS Glue에서 블루프린트 개발](#)

- [AWS Glue의 블루프린트 등록](#)
- [AWS Glue에서 블루프린트 보기](#)
- [AWS Glue에서 블루프린트 업데이트](#)
- [AWS Glue의 블루프린트에서 워크플로 생성](#)
- [AWS Glue에서 블루프린트 실행 보기](#)

AWS Glue의 블루프린트 개요

Note

청사진 기능은 현재 아시아 태평양(자카르타) 리전 및 중동(UAE) 리전에서 AWS Glue 콘솔을 통해 사용할 수 없습니다.

AWS Glue 블루프린트는 AWS Glue 워크플로를 생성하고 공유하는 방법을 제공합니다. 유사한 사용 사례에 사용할 수 있는 복잡한 ETL 프로세스가 있는 경우 각 사용 사례에 대한 AWS Glue 워크플로를 생성하는 대신 단일 블루프린트를 생성할 수 있습니다.

블루프린트는 워크플로에 포함할 작업 및 크롤러를 지정하고 워크플로 사용자가 블루프린트를 실행하여 워크플로를 생성할 때 제공하는 파라미터를 지정합니다. 파라미터를 사용하면 단일 블루프린트에서 유사한 다양한 사용 사례에 대한 워크플로를 생성할 수 있습니다. 워크플로에 대한 자세한 내용은 [AWS Glue의 워크플로 개요](#) 섹션을 참조하세요.

다음은 블루프린트의 사용 사례 예입니다.

- 기존 데이터 집합을 분할하려고 합니다. 블루프린트의 입력 파라미터는 Amazon Simple Storage Service(Amazon S3) 소스 및 대상 경로와 파티션 열 목록입니다.
- Amazon DynamoDB 테이블의 스냅샷을 Amazon Redshift와 같은 SQL 데이터 스토어로 생성하려고 합니다. 블루프린트에 대한 입력 파라미터는 DynamoDB 테이블 이름과 Amazon Redshift 클러스터 및 대상 데이터베이스를 지정하는 AWS Glue 연결입니다.
- 여러 Amazon S3 경로의 CSV 데이터를 Parquet로 변환하려고 합니다. AWS Glue 워크플로에 각 경로에 대해 별도의 크롤러와 작업을 포함하려고 합니다. 입력 파라미터는 AWS Glue Data Catalog의 대상 데이터베이스와 쉼표로 구분된 Amazon S3 경로 목록입니다. 이 경우 워크플로가 생성하는 크롤러 및 작업의 수는 가변적입니다.

블루프린트 구성 요소

블루프린트는 다음과 같은 구성 요소를 포함하는 ZIP 아카이브 파일입니다.

- Python 레이아웃 생성기 스크립트

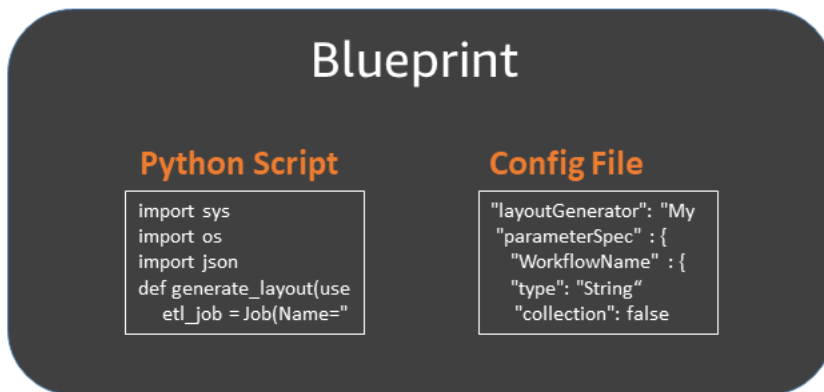
워크플로 레이아웃, 즉 워크플로에 대해 생성할 크롤러 및 작업, 작업 및 크롤러 속성, 작업과 크롤러 간의 종속성을 지정하는 함수를 포함합니다. 이 함수는 블루프린트 파라미터를 허용하고 AWS Glue가 워크플로를 생성하는 데 사용하는 워크플로 구조(JSON 객체)를 반환합니다. Python 스크립트를 사용하여 워크플로를 생성하기 때문에 사용 사례에 적합한 고유한 논리를 추가할 수 있습니다.

- 구성 파일

워크플로 레이아웃을 생성하는 Python 함수의 정규화된 이름을 지정합니다. 또한 스크립트에서 사용하는 모든 블루프린트 파라미터의 이름, 데이터 유형 및 기타 속성을 지정합니다.

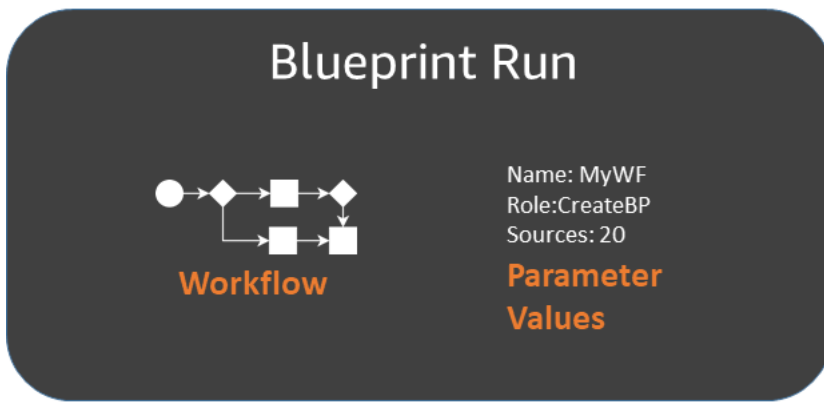
- (선택 사항) ETL 스크립트 및 지원 파일

고급 사용 사례로 작업에서 사용하는 ETL 스크립트의 위치를 파라미터화할 수 있습니다. ZIP 아카이브에 작업 스크립트 파일을 포함하고 스크립트를 복사할 Amazon S3 위치에 대한 블루프린트 파라미터를 지정할 수 있습니다. 레이아웃 생성기 스크립트는 ETL 스크립트를 지정된 위치에 복사하고 해당 위치를 작업 스크립트 위치 속성으로 지정할 수 있습니다. 스크립트에서 처리하는 경우 라이브러리 또는 기타 지원 파일을 포함할 수도 있습니다.



블루프린트 실행

블루프린트에서 워크플로를 생성하는 경우 AWS Glue가 블루프린트를 실행하여 워크플로와 워크플로가 캡슐화하는 작업, 크롤러, 트리거를 생성하는 비동기 프로세스를 시작합니다. AWS Glue는 블루프린트 실행을 사용하여 워크플로 및 해당 구성 요소의 생성을 오케스트레이션합니다. 블루프린트 실행 상태를 확인하여 생성 프로세스의 상태를 봅니다. 블루프린트 실행에는 블루프린트 파라미터에 대해 제공한 값도 저장됩니다.




AWS Glue 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 블루프린트 실행을 볼 수 있습니다. 워크플로를 보거나 문제를 해결할 때 언제든지 블루프린트 실행으로 돌아가 워크플로를 생성하는 데 사용된 블루프린트 파라미터 값을 볼 수 있습니다.

블루프린트의 수명 주기

블루프린트는 AWS Glue로 개발, 테스트, 등록되고 워크플로를 생성하기 위해 실행됩니다. 일반적으로 블루프린트 수명 주기에는 세 가지 페르소나가 포함됩니다.

페르소나	Tasks
AWS Glue 개발자	<ul style="list-style-type: none"> 워크플로 레이아웃 스크립트를 작성하고 구성 파일을 생성합니다. AWS Glue 서비스에서 제공하는 라이브러리를 사용하여 로컬에서 블루프린트를 테스트합니다. 스크립트, 구성 파일 및 지원 파일의 ZIP 아카이브를 생성하고 아카이브를 Amazon S3의 위치에 게시합니다. AWS Glue 관리자의 AWS 계정에 버킷 객체에 대한 읽기 권한을 부여하는 버킷 정책을 Amazon S3 버킷에 추가합니다. Amazon S3의 ZIP 아카이브에 대한 IAM 읽기 권한을 AWS Glue 관리자에게 부여합니다.
AWS Glue 관리자	<ul style="list-style-type: none"> 블루프린트를 AWS Glue에 등록합니다. AWS Glue에서 ZIP 아카이브의 복사본을 예약된 Amazon S3 위치에 만듭니다.

페르소나	Tasks
	<ul style="list-style-type: none"> 데이터 분석가에게 블루프린트에 대한 IAM 권한을 부여합니다.
데이터 분석가	<ul style="list-style-type: none"> 블루프린트를 실행하여 워크플로를 생성하고 블루프린트 파라미터 값을 제공합니다. 블루프린트 실행 상태를 확인하여 워크플로 및 워크플로 구성 요소가 성공적으로 생성되었는지 확인합니다. 워크플로를 실행하고 문제를 해결합니다. 워크플로를 실행하기 전에 AWS Glue 콘솔에서 워크플로 설계 그래프를 보고 워크플로를 확인할 수 있습니다.

 다음 사항도 참조하십시오.


- [AWS Glue에서 블루프린트 개발](#)
- [AWS Glue의 블루프린트에서 워크플로 생성](#)
- [AWS Glue 블루프린트에 대한 페르소나 및 역할의 권한](#)

AWS Glue에서 블루프린트 개발

AWS Glue 개발자는 데이터 분석가가 워크플로를 생성하는 데 사용할 수 있는 블루프린트를 생성하고 게시할 수 있습니다.

주제

- [블루프린트 개발 개요](#)
- [블루프린트 개발을 위한 사전 조건](#)
- [블루프린트 코드 작성](#)
- [샘플 블루프린트 프로젝트](#)
- [블루프린트 테스트](#)
- [블루프린트 게시](#)
- [AWS Glue 블루프린트 클래스 참조](#)
- [블루프린트 샘플](#)

 다음 사항도 참조하십시오.

- [AWS Glue의 블루프린트 개요](#)

블루프린트 개발 개요

개발 프로세스의 첫 번째 단계는 블루프린트의 이점을 얻을 수 있는 일반적인 사용 사례를 파악하는 것입니다. 일반적인 사용 사례에는 일반적인 방식으로 해결해야 한다고 생각하는 반복적인 ETL 문제가 포함됩니다. 다음으로 일반화된 사용 사례를 구현하는 블루프린트를 설계하고 일반화된 사용 사례에서 특정 사용 사례를 정의할 수 있는 블루프린트 입력 파라미터를 정의합니다.

블루프린트는 블루프린트 파라미터 구성 파일이 포함된 프로젝트와 생성할 워크플로의 레이아웃을 정의하는 스크립트로 구성됩니다. 레이아웃은 생성할 작업 및 크롤러(또는 블루프린트 스크립트 용어의 엔터티)를 정의합니다.

레이아웃 스크립트에서 트리거를 직접 지정하지 않습니다. 대신 스크립트가 생성하는 작업과 크롤러 간의 종속성을 지정하는 코드를 작성합니다. AWS Glue에서는 종속성 사양에 따라 트리거를 생성합니다. 레이아웃 스크립트의 출력은 모든 워크플로 엔터티에 대한 사양을 포함하는 워크플로 객체입니다.

다음 AWS Glue 블루프린트 라이브러리를 사용하여 워크플로 객체를 구축합니다.

- `awsglue.blueprint.base_resource` - 라이브러리에서 사용하는 기본 리소스의 라이브러리입니다.
- `awsglue.blueprint.workflow` - Workflow 클래스를 정의하기 위한 라이브러리입니다.
- `awsglue.blueprint.job` - Job 클래스를 정의하기 위한 라이브러리입니다.
- `awsglue.blueprint.crawler` - Crawler 클래스를 정의하기 위한 라이브러리입니다.

레이아웃 생성을 위해 지원되는 유일한 다른 라이브러리는 Python 셸에 사용할 수 있는 라이브러리입니다.

블루프린트를 게시하기 전에 블루프린트 라이브러리에 정의된 방법을 사용하여 블루프린트를 로컬에서 테스트할 수 있습니다.

데이터 분석가에게 블루프린트를 제공할 준비가 되면 스크립트, 파라미터 구성 파일, 지원 파일(예: 추가 스크립트 및 라이브러리)을 배포 가능한 단일 자산으로 패키징합니다. 그런 다음 자산을 Amazon S3에 업로드하고 관리자에게 AWS Glue에 등록하도록 요청합니다.

추가 샘플 블루프린트 프로젝트에 대한 자세한 내용은 [샘플 블루프린트 프로젝트](#) 및 [블루프린트 샘플](#) 섹션을 참조하세요.

블루프린트 개발을 위한 사전 조건

블루프린트를 개발하려면 AWS Glue 사용과 Apache Spark ETL 작업 또는 Python 셸 작업을 위한 스크립트 작성에 익숙해야 합니다. 또한 다음 설정 태스크를 완료해야 합니다.

- 블루프린트 레이아웃 스크립트에 사용할 4개의 AWS Python 라이브러리를 다운로드합니다.
- AWS SDK를 설정합니다.
- AWS CLI를 설정합니다.

Python 라이브러리 다운로드

GitHub에서 다음 라이브러리를 다운로드하고 프로젝트에 설치합니다.

- https://github.com/aws-labs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/base_resource.py
- <https://github.com/aws-labs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/workflow.py>
- <https://github.com/aws-labs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/crawler.py>
- <https://github.com/aws-labs/aws-glue-blueprint-libs/tree/master/awsglue/blueprint/job.py>

AWS Java SDK 설정

AWS Java SDK의 경우 블루프린트용 API가 jar 포함된 파일을 추가해야 합니다.

1. 아직 설정하지 않은 경우 AWS SDK for Java를 설정합니다.
 - Java 1.x의 경우 AWS SDK for Java Developer Guide의 [Set up the AWS SDK for Java](#) 지침을 따르세요.
 - Java 2.x의 경우 AWS SDK for Java 2.x Developer Guide의 [Setting up the AWS SDK for Java 2.x](#) 지침을 따르세요.
2. 블루프린트용 API에 대한 액세스 권한이 있는 클라이언트 jar 파일을 다운로드합니다.
 - Java 1.x의 경우: `s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-preview/AWSGlueJavaClient-1.11.x.jar`
 - Java 2.x의 경우: `s3://awsglue-custom-blueprints-preview-artifacts/awsglue-java-sdk-v2-preview/AwsJavaSdk-Glue-2.0.jar`

3. AWS Java SDK에서 제공하는 AWS Glue 클라이언트를 재정의하려면 Java 클래스 경로 앞에 클라이언트 jar를 추가합니다.

```
export CLASSPATH=<path-to-preview-client-jar>:$CLASSPATH
```

4. (선택 사항) 다음 Java 애플리케이션으로 SDK를 테스트합니다. 애플리케이션이 빈 목록을 출력해야 합니다.

accessKey 및 secretKey를 자격 증명으로 바꾸고 us-east-1을 리전으로 바꿉니다.

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.glue.AWSGlue;
import com.amazonaws.services.glue.AWSGlueClientBuilder;
import com.amazonaws.services.glue.model.ListBlueprintsRequest;

public class App{
    public static void main(String[] args) {
        AWSCredentials credentials = new BasicAWSCredentials("accessKey",
"secretKey");
        AWSCredentialsProvider provider = new
AWSStaticCredentialsProvider(credentials);
        AWSGlue glue = AWSGlueClientBuilder.standard().withCredentials(provider)
                .withRegion("us-east-1").build();
        ListBlueprintsRequest request = new
ListBlueprintsRequest().withMaxResults(2);
        System.out.println(glue.listBlueprints(request));
    }
}
```

AWS Python SDK 설정

다음 단계에서는 컴퓨터에 Python 버전 2.7 이상 또는 버전 3.6 이상이 설치되어 있다고 가정합니다.

1. 다음 boto3 휠 파일을 다운로드합니다. 열거나 저장하라는 메시지가 표시되면 s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/boto3-1.17.31-py2.py3-none-any.whl 파일을 저장합니다.
2. botocore 휠 파일인 s3://awsglue-custom-blueprints-preview-artifacts/aws-python-sdk-preview/botocore-1.20.31-py2.py3-none-any.whl을 다운로드합니다.

3. Python 버전을 확인합니다.

```
python --version
```

4. Python 버전에 따라 다음 명령을 입력합니다(Linux의 경우).

- Python 2.7 이상용입니다.

```
python3 -m pip install --user virtualenv
source env/bin/activate
```

- Python 3.6 이상용입니다.

```
python3 -m venv python-sdk-test
source python-sdk-test/bin/activate
```

5. boto3 휠 파일을 설치합니다.

```
python3 -m pip install <download-directory>/boto3-1.20.31-py2.py3-none-any.whl
```

6. boto3 휠 파일을 설치합니다.

```
python3 -m pip install <download-directory>/boto3-1.17.31-py2.py3-none-any.whl
```

7. ~/.aws/credentials 및 ~/.aws/config 파일에서 자격 증명과 기본 리전을 구성합니다. 자세한 내용은 AWS Command Line Interface User Guide의 [Configuring the AWS CLI](#)를 참조하세요.

8. (선택 사항) 설정을 테스트합니다. 다음 명령은 빈 목록을 반환합니다.

us-east-1을 해당 리전으로 바꿉니다.

```
$ python
>>> import boto3
>>> glue = boto3.client('glue', 'us-east-1')
>>> glue.list_blueprints()
```

미리 보기 AWS CLI 설정

1. 아직 수행하지 않았다면 컴퓨터에 AWS Command Line Interface(AWS CLI)를 설치 및/또는 업데이트합니다. 이를 수행하는 가장 쉬운 방법은 Python 설치 프로그램 유틸리티인 pip를 사용하는 것입니다.

```
pip install awscli --upgrade --user
```

AWS CLI에 대한 전체 설치 지침은 [AWS Command Line Interface 설치](#)에서 찾을 수 있습니다.

2. `s3://awsglue-custom-blueprints-preview-artifacts/awscli-preview-build/awscli-1.19.31-py2.py3-none-any.whl`에서 AWS CLI 휠 파일을 다운로드합니다.
3. AWS CLI 휠 파일을 설치합니다.

```
python3 -m pip install awscli-1.19.31-py2.py3-none-any.whl
```

4. `aws configure` 명령을 실행합니다. AWS 자격 증명(액세스 키 및 보안 암호 키 포함) 및 AWS 리전을 구성합니다. AWS CLI 구성에 대한 자세한 내용은 [AWS CLI 구성](#)에서 찾을 수 있습니다.
5. AWS CLI를 테스트합니다. 다음 명령은 빈 목록을 반환합니다.

`us-east-1`을 해당 리전으로 바꿉니다.

```
aws glue list-blueprints --region us-east-1
```

블루프린트 코드 작성

생성하는 각 블루프린트 프로젝트에는 최소한 다음 파일이 포함되어야 합니다.

- 워크플로를 정의하는 Python 레이아웃 스크립트. 스크립트에는 워크플로의 엔터티(작업 및 크롤러)와 이들 간의 종속성을 정의하는 기능이 포함되어 있습니다.
- 다음을 정의하는 구성 파일, `blueprint.cfg`
 - 워크플로 레이아웃 정의 기능의 전체 경로.
 - 블루프린트에서 허용하는 파라미터.

주제

- [블루프린트 레이아웃 스크립트 생성](#)
- [구성 파일 생성](#)

• [블루프린트 파라미터 지정](#)

블루프린트 레이아웃 스크립트 생성

블루프린트 레이아웃 스크립트에는 워크플로에서 엔터티를 생성하는 함수가 포함되어야 합니다. 원하는 대로 이 함수의 이름을 지정할 수 있습니다. AWS Glue에서는 구성 파일을 사용하여 함수의 정규화된 이름을 확인합니다.

레이아웃 함수는 다음을 수행합니다.

- (선택 사항) Job 클래스를 인스턴스화하여 Job 객체를 생성하고 Command 및 Role 등의 인수를 전달합니다. AWS Glue 콘솔 또는 API를 사용하여 작업을 생성하는 경우 지정할 작업 속성입니다.
- (선택 사항) Crawler 클래스를 인스턴스화하여 Crawler 객체를 생성하고 이름, 역할 및 대상 인수를 전달합니다.
- 객체(워크플로 엔터티) 간의 종속성을 나타내려면 DependsOn 및 WaitForDependencies 추가 인수를 Job() 및 Crawler()에 전달합니다. 이러한 인수는 이 섹션의 뒷부분에서 설명합니다.
- Workflow 클래스를 인스턴스화하여 AWS Glue에 반환되는 워크플로 객체를 생성하고 Name 인수, Entities 인수, 선택적 OnSchedule 인수를 전달합니다. 이 Entities 인수는 워크플로에 포함할 모든 작업 및 크롤러를 지정합니다. Entities 객체를 구성하는 방법을 보려면 이 섹션의 뒷부분에 나오는 샘플 프로젝트를 참조하세요.
- Workflow 객체를 반환합니다.

Job, Crawler 및 Workflow 클래스의 정의는 [AWS Glue 블루프린트 클래스 참조](#) 섹션을 참조하세요.

레이아웃 함수에 사용할 수 있는 입력 인수는 다음과 같습니다.

인수	설명
user_params	블루프린트 파라미터 이름 및 값의 Python 딕셔너리. 자세한 내용은 블루프린트 파라미터 지정 단원을 참조하십시오.
system_params	두 가지 속성(region 및 accountId)을 포함하는 Python 딕셔너리.

다음은 Layout.py라는 파일의 샘플 레이아웃 생성기 스크립트입니다.

```
import argparse
```

```

import sys
import os
import json
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *

def generate_layout(user_params, system_params):

    etl_job = Job(Name="{}_etl_job".format(user_params['WorkflowName']),
                  Command={
                      "Name": "glueetl",
                      "ScriptLocation": user_params['ScriptLocation'],
                      "PythonVersion": "2"
                  },
                  Role=user_params['PassRole'])
    post_process_job = Job(Name="{}_post_process".format(user_params['WorkflowName']),
                           Command={
                               "Name": "pythonshell",
                               "ScriptLocation": user_params['ScriptLocation'],
                               "PythonVersion": "2"
                           },
                           Role=user_params['PassRole'],
                           DependsOn={
                               etl_job: "SUCCEEDED"
                           },
                           WaitForDependencies="AND")
    sample_workflow = Workflow(Name=user_params['WorkflowName'],
                               Entities=Entities(Jobs=[etl_job, post_process_job]))
    return sample_workflow

```

샘플 스크립트는 필요한 Blueprint 라이브러리를 가져오고 2개의 작업으로 워크플로를 생성하는 `generate_layout` 함수를 포함합니다. 이것은 매우 간단한 스크립트입니다. 보다 복잡한 스크립트는 추가 로직 및 파라미터를 사용하여 많은 작업과 크롤러 또는 다양한 수의 작업과 크롤러가 있는 워크플로를 생성할 수 있습니다.

DependsOn 인수 사용

DependsOn 인수는 이 엔터티가 워크플로 내의 다른 엔터티에 대해 갖는 종속성의 딕셔너리 표현입니다. 형식은 다음과 같습니다.

```
DependsOn = {dependency1 : state, dependency2 : state, ...}
```

이 딕셔너리의 키는 엔터티의 이름이 아닌 객체 참조를 나타내는 반면 값은 감시할 상태에 해당하는 문자열입니다. AWS Glue는 적절한 트리거를 유추합니다. 유효한 상태는 [조건 구조](#)를 참조하세요.

예를 들어 작업은 크롤러의 성공적인 완료에 따라 달라질 수 있습니다. 다음과 같이 이름이 crawler2인 크롤러 객체를 정의하는 경우

```
crawler2 = Crawler(Name="my_crawler", ...)
```

그러면 crawler2에 종속된 객체에는 다음과 같은 생성자 인수가 포함됩니다.

```
DependsOn = {crawler2 : "SUCCEEDED"}
```

예:

```
job1 = Job(Name="Job1", ..., DependsOn = {crawler2 : "SUCCEEDED", ...})
```

엔터티에 대해 DependsOn이 생략된 경우 해당 엔터티는 워크플로 시작 트리거에 따라 달라집니다.

WaitForDependencies 인수 사용

WaitForDependencies 인수는 작업 또는 크롤러 엔터티가 종속된 모든 엔터티가 완료될 때까지 또는 하나라도 완료될 때까지 기다려야 하는지 여부를 정의합니다.

허용되는 값은 "AND" 또는 "ANY"입니다.

OnSchedule 인수 사용

Workflow 클래스 생성자의 OnSchedule 인수는 워크플로에 대한 시작 트리거 정의를 정의하는 cron 표현식입니다.

이 인수가 지정되면 AWS Glue는 해당 일정으로 일정 트리거를 생성합니다. 지정되지 않은 경우 워크플로의 시작 트리거는 온디맨드 트리거입니다.

구성 파일 생성

블루프린트 구성 파일은 워크플로 생성을 위한 스크립트 진입점과 블루프린트에서 허용하는 파라미터를 정의하는 필수 파일입니다. 파일의 이름은 blueprint.cfg여야 합니다.

다음은 샘플 구성 파일입니다.

```
{
  "layoutGenerator": "DemoBlueprintProject.Layout.generate_layout",
  "parameterSpec" : {
    "WorkflowName" : {
      "type": "String",
      "collection": false
    },
    "WorkerType" : {
      "type": "String",
      "collection": false,
      "allowedValues": ["G1.X", "G2.X"],
      "defaultValue": "G1.X"
    },
    "Dpu" : {
      "type" : "Integer",
      "allowedValues" : [2, 4, 6],
      "defaultValue" : 2
    },
    "DynamoDBTableName": {
      "type": "String",
      "collection" : false
    },
    "ScriptLocation" : {
      "type": "String",
      "collection": false
    }
  }
}
```

layoutGenerator 속성은 레이아웃을 생성하는 스크립트에서 함수의 정규화된 이름을 지정합니다.

parameterSpec 속성은 이 블루프린트에서 허용하는 파라미터를 지정합니다. 자세한 내용은 [블루프린트 파라미터 지정](#) 단원을 참조하십시오.

Important

구성 파일에 워크플로 이름이 블루프린트 파라미터로 포함되거나 레이아웃 스크립트에서 고유한 워크플로 이름을 생성해야 합니다.

블루프린트 파라미터 지정

구성 파일에는 parameterSpec JSON 객체의 블루프린트 파라미터 사양이 포함되어 있습니다. parameterSpec은 하나 이상의 파라미터 객체를 포함합니다.

```
"parameterSpec": {
  "<parameter_name>": {
    "type": "<parameter-type>",
    "collection": true|false,
    "description": "<parameter-description>",
    "defaultValue": "<default value for the parameter if value not specified>"
    "allowedValues": "<list of allowed values>"
  },
  "<parameter_name>": {
    ...
  }
}
```

다음은 각 파라미터 객체를 코딩하는 규칙입니다.

- 파라미터 이름과 type은 필수입니다. 다른 모든 속성은 선택 사항입니다.
- defaultValue 속성을 지정하는 경우 파라미터는 선택 사항입니다. 그렇지 않으면 파라미터가 필수이며 블루프린트에서 워크플로를 생성하는 데이터 분석가가 해당 값을 제공해야 합니다.
- collection 속성을 true로 설정하면 파라미터가 값 컬렉션을 사용할 수 있습니다. 컬렉션은 모든 데이터 유형이 될 수 있습니다.
- allowedValues를 지정하면 AWS Glue 콘솔에 데이터 분석가가 블루프린트에서 워크플로를 생성할 때 선택할 수 있는 값의 드롭다운 목록이 표시됩니다.

다음은 type에 허용되는 값입니다.

파라미터 데이터 유형	참고
String	-
Integer	-
Double	-

파라미터 데이터 유형	참고
Boolean	가능한 값은 true 및 false입니다. AWS Glue 콘솔의 <블루프린트> 페이지에서 워크플로 생성(Create a workflow from <blueprint>) 페이지에 확인란을 생성합니다.
S3Uri	s3://로 시작하여 Amazon S3 경로를 완성합니다. [<블루프린트>에서 워크플로 생성(Create a workflow from <blueprint>)]에서 텍스트 필드와 [찾아보기(Browse)] 버튼을 생성합니다.
S3Bucket	Amazon S3 버킷 이름만입니다. <블루프린트>에서 워크플로 생성(Create a workflow from <blueprint>) 페이지에서 버킷 선택기를 생성합니다.
IAMRoleArn	AWS Identity and Access Management(IAM) 역할의 Amazon 리소스 이름(ARN)입니다. <블루프린트>에서 워크플로 생성(Create a workflow from <blueprint>) 페이지에서 역할 선택기를 생성합니다.
IAMRoleName	IAM 역할의 이름입니다. <블루프린트>에서 워크플로 생성(Create a workflow from <blueprint>) 페이지에서 역할 선택기를 생성합니다.

샘플 블루프린트 프로젝트

데이터 포맷 변환은 빈번한 추출, 변환, 로드 사용 사례입니다. 일반적인 분석 워크로드에서는 Parquet 또는 ORC와 같은 열 기반 파일 포맷이 CSV 또는 JSON과 같은 텍스트 포맷보다 선호됩니다. 이 샘플 블루프린트를 사용하면 CSV/JSON 등의 데이터를 Amazon S3의 파일용 Parquet로 변환할 수 있습니다.

이 블루프린트는 블루프린트 파라미터로 정의된 S3 경로 목록을 가져와 데이터를 Parquet 포맷으로 변환한 다음 다른 블루프린트 파라미터로 지정된 S3 위치에 씁니다. 레이아웃 스크립트는 각 경로에 대한 크롤러 및 작업을 생성합니다. 또한 레이아웃 스크립트는 Conversion.py의 ETL 스크립트를 다른 블루프린트 파라미터로 지정된 S3 버킷에 업로드합니다. 그런 다음 레이아웃 스크립트는 업로드된 스크립트를 각 작업에 대한 ETL 스크립트로 지정합니다. 프로젝트의 ZIP 아카이브에는 레이아웃 스크립트, ETL 스크립트 및 Blueprint 구성 파일이 포함되어 있습니다.

추가 샘플 블루프린트 프로젝트에 대한 자세한 내용은 [블루프린트 샘플](#) 섹션을 참조하세요.

다음은 Layout.py 파일에 있는 레이아웃 스크립트입니다.

```
from awsglue.blueprint.workflow import *
```

```

from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *
import boto3

s3_client = boto3.client('s3')

# Ingesting all the S3 paths as Glue table in parquet format
def generate_layout(user_params, system_params):
    #Always give the full path for the file
    with open("ConversionBlueprint/Conversion.py", "rb") as f:
        s3_client.upload_fileobj(f, user_params['ScriptsBucket'], "Conversion.py")
    etlScriptLocation = "s3://{}/Conversion.py".format(user_params['ScriptsBucket'])

    crawlers = []
    jobs = []
    workflowName = user_params['WorkflowName']
    for path in user_params['S3Paths']:
        tablePrefix = "source_"
        crawler = Crawler(Name="{}_crawler".format(workflowName),
                          Role=user_params['PassRole'],
                          DatabaseName=user_params['TargetDatabase'],
                          TablePrefix=tablePrefix,
                          Targets= {"S3Targets": [{"Path": path}]})
        crawlers.append(crawler)
    transform_job = Job(Name="{}_transform_job".format(workflowName),
                        Command={"Name": "glueetl",
                                "ScriptLocation": etlScriptLocation,
                                "PythonVersion": "3"},
                        Role=user_params['PassRole'],
                        DefaultArguments={"--database_name":
user_params['TargetDatabase'],
                                        "--table_prefix": tablePrefix,
                                        "--region_name": system_params['region'],
                                        "--output_path":
user_params['TargetS3Location']},
                        DependsOn={crawler: "SUCCEEDED"},
                        WaitForDependencies="AND")
    jobs.append(transform_job)
    conversion_workflow = Workflow(Name=workflowName, Entities=Entities(Jobs=jobs,
Crawlers=crawlers))
    return conversion_workflow

```

다음은 해당 블루프린트 구성 파일인 `blueprint.cfg`입니다.

```

{
  "layoutGenerator": "ConversionBlueprint.Layout.generate_layout",
  "parameterSpec" : {
    "WorkflowName" : {
      "type": "String",
      "collection": false,
      "description": "Name for the workflow."
    },
    "S3Paths" : {
      "type": "S3Uri",
      "collection": true,
      "description": "List of Amazon S3 paths for data ingestion."
    },
    "PassRole" : {
      "type": "IAMRoleName",
      "collection": false,
      "description": "Choose an IAM role to be used in running the job/crawler"
    },
    "TargetDatabase": {
      "type": "String",
      "collection" : false,
      "description": "Choose a database in the Data Catalog."
    },
    "TargetS3Location": {
      "type": "S3Uri",
      "collection" : false,
      "description": "Choose an Amazon S3 output path: ex:s3://<target_path>/."
    },
    "ScriptsBucket": {
      "type": "S3Bucket",
      "collection": false,
      "description": "Provide an S3 bucket name(in the same AWS Region) to store
the scripts."
    }
  }
}

```

Conversion.py 파일의 다음 스크립트는 업로드된 ETL 스크립트입니다. 변환하는 동안 파티션 구성표를 유지합니다.

```

import sys
from pyspark.sql.functions import *
from pyspark.context import SparkContext

```

```
from awsglue.transforms import *
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions
import boto3

args = getResolvedOptions(sys.argv, [
    'JOB_NAME',
    'region_name',
    'database_name',
    'table_prefix',
    'output_path'])
databaseName = args['database_name']
tablePrefix = args['table_prefix']
outputPath = args['output_path']

glue = boto3.client('glue', region_name=args['region_name'])

glue_context = GlueContext(SparkContext.getOrCreate())
spark = glue_context.spark_session
job = Job(glue_context)
job.init(args['JOB_NAME'], args)

def get_tables(database_name, table_prefix):
    tables = []
    paginator = glue.get_paginator('get_tables')
    for page in paginator.paginate(DatabaseName=database_name, Expression=table_prefix
+"""):
        tables.extend(page['TableList'])
    return tables

for table in get_tables(databaseName, tablePrefix):
    tableName = table['Name']
    partitionList = table['PartitionKeys']
    partitionKeys = []
    for partition in partitionList:
        partitionKeys.append(partition['Name'])

# Create DynamicFrame from Catalog
dyf = glue_context.create_dynamic_frame.from_catalog(
    name_space=databaseName,
    table_name=tableName,
    additional_options={
        'useS3ListImplementation': True
```

```
    },
    transformation_ctx='dyf'
)

# Resolve choice type with make_struct
dyf = ResolveChoice.apply(
    frame=dyf,
    choice='make_struct',
    transformation_ctx='resolvechoice_' + tableName
)

# Drop null fields
dyf = DropNullFields.apply(
    frame=dyf,
    transformation_ctx="dropnullfields_" + tableName
)

# Write DynamicFrame to S3 in glueparquet
sink = glue_context.getSink(
    connection_type="s3",
    path=outputPath,
    enableUpdateCatalog=True,
    partitionKeys=partitionKeys
)
sink.setFormat("glueparquet")

sink.setCatalogInfo(
    catalogDatabase=databaseName,
    catalogTableName=tableName[len(tablePrefix):]
)
sink.writeFrame(dyf)

job.commit()
```

Note

2개의 Amazon S3 경로만 샘플 블루프린트에 대한 입력으로 제공될 수 있습니다. 이는 AWS Glue 트리거가 2개의 크롤러 작업만 호출하도록 제한되기 때문입니다.

블루프린트 테스트

코드를 개발하는 동안 워크플로 레이아웃이 올바른지 확인하기 위해 로컬 테스트를 수행해야 합니다.

로컬 테스트는 AWS Glue 작업, 크롤러 또는 트리거를 생성하지 않습니다. 대신 레이아웃 스크립트를 로컬에서 실행하고 `to_json()` 및 `validate()` 메서드를 사용하여 객체를 인쇄하고 오류를 찾습니다. 이러한 방법은 라이브러리에 정의된 3가지 클래스 모두에서 사용할 수 있습니다.

AWS Glue가 레이아웃 함수에 전달하는 `user_params` 및 `system_params` 인수를 처리하는 두 가지 방법이 있습니다. 테스트 벤치 코드는 샘플 블루프린트 파라미터 값의 딕셔너리를 생성하고 이를 `user_params` 인수로 레이아웃 함수에 전달할 수 있습니다. 또는 `user_params`에 대한 참조를 제거하고 하드코딩된 문자열로 바꿀 수 있습니다.

코드에서 `system_params` 인수의 `region` 및 `accountId` 속성을 사용하는 경우 `system_params`에 대한 딕셔너리를 전달할 수 있습니다.

블루프린트를 테스트하려면

1. 라이브러리가 있는 디렉터리에서 Python 인터프리터를 시작하거나 블루프린트 파일과 제공된 라이브러리를 원하는 IDE(통합 개발 환경)로 로드합니다.
2. 코드가 제공된 라이브러리를 가져 오는지 확인합니다.
3. 레이아웃 기능에 코드를 추가하여 엔터티 또는 Workflow 객체에서 `validate()` 또는 `to_json()`을 호출합니다. 예를 들어 코드가 `mycrawler`라는 Crawler 객체를 생성하는 경우 다음과 같이 `validate()`를 호출할 수 있습니다.

```
mycrawler.validate()
```

다음과 같이 `mycrawler`를 인쇄 할 수 있습니다.

```
print(mycrawler.to_json())
```

객체에서 `to_json`을 호출하면 `to_json()`이 `validate()`를 호출하기 때문에 `validate()`도 호출할 필요가 없습니다.

워크플로 객체에서 이러한 메서드를 호출하는 것이 가장 유용합니다. 스크립트가 워크플로 객체의 이름을 `my_workflow`라고 가정하고 다음과 같이 워크플로 객체를 검증하고 인쇄합니다.

```
print(my_workflow.to_json())
```


`to_json()` 및 `validate()`에 대한 자세한 내용은 [클래스 메서드](#) 섹션을 참조하세요.

이 섹션의 뒷부분에 나오는 예제와 같이 `pprint`를 가져와서 워크플로 객체를 예쁘게 인쇄할 수도 있습니다.

4. 코드를 실행하고, 오류를 수정하고, 마지막으로 `validate()` 또는 `to_json()`에 대한 호출을 제거합니다.

Example

다음 예제는 샘플 블루프린트 파라미터의 딕셔너리를 구성하고 레이아웃 함수 `generate_compaction_workflow`에 `user_params` 인수로 전달하는 방법을 보여줍니다. 또한 생성된 워크플로 객체를 예쁘게 인쇄하는 방법을 보여줍니다.

```
from pprint import pprint
from awsglue.blueprint.workflow import *
from awsglue.blueprint.job import *
from awsglue.blueprint.crawler import *

USER_PARAMS = {"WorkflowName": "compaction_workflow",
               "ScriptLocation": "s3://awsexamplebucket1/scripts/threaded-
compaction.py",
               "PassRole": "arn:aws:iam::111122223333:role/GlueRole-ETL",
               "DatabaseName": "cloudtrail",
               "TableName": "ct_cloudtrail",
               "CoalesceFactor": 4,
               "MaxThreadWorkers": 200}

def generate_compaction_workflow(user_params: dict, system_params: dict) -> Workflow:
    compaction_job = Job(Name=f"{user_params['WorkflowName']}_etl_job",
                        Command={"Name": "glueetl",
                                "ScriptLocation": user_params['ScriptLocation'],
                                "PythonVersion": "3"},
                        Role="arn:aws:iam::111122223333:role/
AWSGlueServiceRoleDefault",
                        DefaultArguments={"DatabaseName": user_params['DatabaseName'],
                                          "TableName": user_params['TableName'],
                                          "CoalesceFactor":
user_params['CoalesceFactor'],
                                          "max_thread_workers":
user_params['MaxThreadWorkers']})
```

```

catalog_target = {"CatalogTargets": [{"DatabaseName": user_params['DatabaseName'],
"Tables": [user_params['TableName']]}]}

compact_files_crawler = Crawler(Name=f"{user_params['WorkflowName']}_post_crawl",
                                Targets = catalog_target,
                                Role=user_params['PassRole'],
                                DependsOn={compaction_job: "SUCCEEDED"},
                                WaitForDependencies="AND",
                                SchemaChangePolicy={"DeleteBehavior": "LOG"})

compaction_workflow = Workflow(Name=user_params['WorkflowName'],
                                Entities=Entities(Jobs=[compaction_job],
                                Crawlers=[compact_files_crawler]))
return compaction_workflow

generated = generate_compaction_workflow(user_params=USER_PARAMS, system_params={})
gen_dict = generated.to_json()

pprint(gen_dict)

```

블루프린트 게시

블루프린트를 개발한 후에는 Amazon S3에 업로드해야 합니다. 블루프린트를 게시하는 데 사용하는 Amazon S3 버킷에 대한 쓰기 권한이 있어야 합니다. 또한 블루프린트를 등록할 AWS Glue 관리자에게 Amazon S3 버킷에 대한 읽기 액세스 권한이 있는지 확인해야 합니다. AWS Glue 블루프린트의 페르소나 및 역할에 대해 제안된 AWS Identity and Access Management(IAM) 권한 정책은 [AWS Glue 블루프린트에 대한 페르소나 및 역할의 권한](#) 섹션을 참조하세요.

블루프린트를 게시하려면

1. 필요한 스크립트, 리소스 및 블루프린트 구성 파일을 생성합니다.
2. 모든 파일을 ZIP 아카이브에 추가하고 ZIP 파일을 Amazon S3에 업로드합니다. 사용자가 블루프린트를 등록하고 실행할 리전과 동일한 리전에 있는 S3 버킷을 사용합니다.

다음 명령을 사용하여 명령줄에서 ZIP 파일을 생성할 수 있습니다.


```
zip -r folder.zip folder
```

3. 원하는 AWS 계정에 읽기 권한을 부여하는 버킷 정책을 추가합니다. 다음은 샘플 정책입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-blueprints/*"
    }
  ]
}
```

4. Amazon S3 버킷에 대한 IAM s3:GetObject 권한을 AWS Glue 관리자 또는 블루프린트를 등록할 사람에게 부여합니다. 관리자에게 부여할 샘플 정책은 [블루프린트에 대한 AWS Glue 관리자 권한](#) 섹션을 참조하세요.

블루프린트의 로컬 테스트를 완료한 후 AWS Glue에서 블루프린트를 테스트할 수도 있습니다. AWS Glue에서 블루프린트를 테스트하려면 등록이 필요합니다. IAM 권한 부여를 사용하거나 별도의 테스트 계정을 사용하여 등록된 블루프린트를 볼 수 있는 사람을 제한할 수 있습니다.

 다음 사항도 참조하세요.

- [AWS Glue의 블루프린트 등록](#)

AWS Glue 블루프린트 클래스 참조

AWS Glue 블루프린트용 라이브러리는 워크플로 레이아웃 스크립트에서 사용하는 세 가지 클래스인 Job, Crawler 및 Workflow를 정의합니다.

주제

- [작업 클래스](#)
- [크롤러 클래스](#)
- [워크플로 클래스](#)
- [클래스 메서드](#)

작업 클래스

Job 클래스는 AWS Glue ETL 작업을 나타냅니다.

필수 생성자 인수

다음은 Job 클래스에 대한 필수 생성자 인수입니다.

인수 이름	유형	설명
Name	str	작업에 할당할 이름입니다. AWS Glue는 작업을 다른 블루프린트 실행에서 생성된 작업과 구별하기 위해 이름에 임의로 생성된 접미사를 추가합니다.
Role	str	실행 중 작업이 수입해야 하는 역할의 Amazon 리소스 이름(ARN)입니다.
Command	dict	API 문서의 JobCommand 구조 에 지정된 작업 명령.

선택적 생성자 인수

다음은 Job 클래스에 대한 선택적 생성자 인수입니다.

인수 이름	유형	설명
DependsOn	dict	작업이 종속된 워크플로 엔터티의 목록입니다. 자세한 내용은 DependsOn 인수 사용 단원을 참조하십시오.
WaitForDependencies	str	작업이 종속된 모든 엔터티가 실행되기 전에 완료될 때까지 기다릴지 아니면 하나라도 완료될 때까지 기다릴지 나타냅니다. 자세한 내용은 WaitForDependencies 인수 사용 단원을 참조하십시오. 작업이 하나의 엔터티에만 종속되는 경우 생략합니다.

인수 이름	유형	설명
(작업 속성)	-	AWS Glue API 설명서의 작업 구조 에 나열된 모든 작업 속성(CreatedOn 및 LastModifiedOn 제외)입니다.

크롤러 클래스

Crawler 클래스는 AWS Glue 크롤러를 나타냅니다.

필수 생성자 인수

다음은 Crawler 클래스에 대한 필수 생성자 인수입니다.

인수 이름	유형	설명
Name	str	크롤러에 할당할 이름입니다. AWS Glue는 크롤러를 다른 블루프린트 실행에서 생성된 크롤러와 구별하기 위해 이름에 임의로 생성된 접미사를 추가합니다.
Role	str	크롤러가 실행 중 수입해야 하는 역할의 ARN입니다.
Targets	dict	크롤링할 대상 컬렉션입니다. Targets 클래스 생성자 인수는 API 문서의 CrawlerTargets 구조 에 정의되어 있습니다. 모두 Targets 생성자 인수는 선택 사항이지만 적어도 하나는 전달해야 합니다.

선택적 생성자 인수

다음은 Crawler 클래스에 대한 선택적 생성자 인수입니다.

인수 이름	유형	설명
DependsOn	dict	크롤러가 종속된 워크플로 엔터티의 목록입니다. 자세한 내용은 DependsOn 인수 사용 단원을 참조하십시오.
WaitForDependencies	str	크롤러가 종속된 모든 엔터티가 실행되기 전에 완료될 때까지 기다릴지 아니면 하나라도 완료될 때까지 기다릴지 나타냅니다. 자세한 내용은 WaitForDependencies 인수 사용 단원을 참조하십시오. 크롤러가 하나의 엔터티에만 종속된 경우에는 생략합니다.
(크롤러 속성)	-	다음 예외사항과 함께 AWS Glue API 설명서의 크롤러 구조 에 나열된 모든 크롤러 속성: <ul style="list-style-type: none"> • State • CrawlElapsedTime • CreationTime • LastUpdated • LastCrawl • Version

워크플로 클래스

Workflow 클래스는 AWS Glue 워크플로를 나타냅니다. 워크플로 레이아웃 스크립트는 Workflow 객체를 반환합니다. AWS Glue에서는 이 객체를 기반으로 하여 워크플로를 생성합니다.

필수 생성자 인수

다음은 Workflow 클래스에 대한 필수 생성자 인수입니다.

인수 이름	유형	설명
Name	str	워크플로에 할당할 이름입니다.

인수 이름	유형	설명
Entities	Entities	워크플로에 포함할 엔터티 (작업 및 크롤러)의 컬렉션입니다. Entities 클래스 생성자는 Crawler 객체의 목록인 Crawlers 인수와 Job 객체의 목록인 Jobs 인수를 허용합니다.

선택적 생성자 인수

다음은 Workflow 클래스에 대한 선택적 생성자 인수입니다.

인수 이름	유형	설명
Description	str	워크플루 구조 섹션을 참조하세요.
DefaultRunProperties	dict	워크플루 구조 섹션을 참조하세요.
OnSchedule	str	cron 표현식입니다.

클래스 메서드

세 클래스 모두 다음 메서드를 포함합니다.

validate()

객체의 속성을 검증하고 오류가 발견되면 메시지를 출력하고 종료합니다. 오류가 없으면 출력을 생성하지 않습니다. Workflow 클래스의 경우 워크플로의 모든 엔터티에서 자신을 호출합니다.

to_json()

객체를 JSON으로 직렬화합니다. 또한 validate()를 호출합니다. Workflow 클래스의 경우 JSON 객체에는 작업 및 크롤러 목록과 작업 및 크롤러 종속성 사양에 의해 생성된 트리거 목록이 포함됩니다.

블루프린트 샘플

[AWS Glue 블루프린트 Github 리포지토리](#)에서 사용할 수 있는 샘플 블루프린트 프로젝트가 많이 있습니다. 이 샘플은 참조용이며 프로덕션 용도로 사용되지 않습니다.

샘플 프로젝트의 제목은 다음과 같습니다.

- 압축: 이 블루프린트는 원하는 파일 크기에 따라 입력 파일을 더 큰 청크로 압축하는 작업을 생성합니다.
- 변환: 이 블루프린트는 다양한 표준 파일 포맷의 입력 파일을 분석 워크로드에 최적화된 Apache Parquet 포맷으로 변환합니다.
- Amazon S3 위치 크롤링: 이 블루프린트는 여러 Amazon S3 위치를 크롤링하여 Data Catalog에 메타데이터 테이블을 추가합니다.
- Data Catalog에 대한 사용자 지정 연결: 이 블루프린트는 AWS Glue 사용자 지정 커넥터를 사용하여 데이터 스토어에 액세스하고, 레코드를 읽고, 레코드 스키마를 기준으로 AWS Glue Data Catalog에 테이블 정의를 채웁니다.
- 인코딩: 이 블루프린트는 UTF가 아닌 파일을 UTF로 인코딩된 파일로 변환합니다.
- 분할: 이 블루프린트는 특정 파티션 키를 기반으로 출력 파일을 파티션에 배치하는 분할 작업을 생성합니다.
- Amazon S3 데이터를 DynamoDB 테이블로 가져오기: 이 블루프린트는 Amazon S3에서 DynamoDB 테이블로 데이터를 가져옵니다.
- 관리 대상 표준 테이블: 이 블루프린트는 AWS Glue Data Catalog 테이블을 Lake Formation 테이블로 가져옵니다.

AWS Glue의 블루프린트 등록

AWS Glue 개발자가 블루프린트를 코딩하고 ZIP 아카이브를 Amazon Simple Storage Service(Amazon S3)에 업로드한 후 AWS Glue 관리자가 블루프린트를 등록해야 합니다. 블루프린트를 등록하면 사용할 수 있습니다.

블루프린트를 등록하면 AWS Glue는 블루프린트 아카이브를 예약된 Amazon S3 위치에 복사합니다. 그런 다음 업로드 위치에서 아카이브를 삭제할 수 있습니다.

블루프린트를 등록하려면 업로드된 아카이브가 포함된 Amazon S3 위치에 대한 읽기 권한이 필요합니다. AWS Identity and Access Management(IAM) 권한 `glue:CreateBlueprint`도 필요합니다. 블루프린트를 등록하고, 보고, 유지 관리해야 하는 AWS Glue 관리자에 대해 제안된 권한은 [블루프린트에 대한 AWS Glue 관리자 권한](#) 섹션을 참조하세요.

AWS Glue 콘솔, AWS Glue API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 블루프린트를 등록할 수 있습니다.

블루프린트를 등록하려면(콘솔)

1. Amazon S3의 블루프린트 ZIP 아카이브에 대한 읽기 권한(s3:GetObject)이 있는지 확인합니다.
2. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.

블루프린트를 등록할 권한이 있는 사용자로 로그인합니다. 블루프린트 ZIP 아카이브가 포함된 Amazon S3 버킷과 동일한 AWS 리전으로 전환합니다.

3. 탐색 창에서 블루프린트(Blueprints)를 선택합니다. 그런 다음 블루프린트 페이지에서 블루프린트 추가(Add blueprint)를 선택합니다.
4. 블루프린트 이름과 선택적 설명을 입력합니다.
5. [ZIP 아카이브 위치(S3)(ZIP archive location (S3))]에 업로드된 블루프린트 ZIP 아카이브의 Amazon S3 경로를 입력합니다. 경로에 아카이브 파일 이름을 포함하고 경로를 s3://로 시작합니다.
6. (선택 사항) 태그를 하나 이상 추가합니다.
7. [블루프린트 추가(Add blueprint)]를 선택합니다.

블루프린트 페이지가 반환되고 블루프린트 상태가 CREATING임을 표시합니다. 상태가 ACTIVE 또는 FAILED로 변경될 때까지 새로 고침 버튼을 선택합니다.

8. 상태가 FAILED인 경우 블루프린트를 선택하고 [작업(Actions)] 메뉴에서 [보기(View)]를 선택합니다.

세부 정보 페이지에 실패 이유가 표시됩니다. 오류 메시지가 “위치의 객체에 액세스할 수 없습니다...(Unable to access object at location...)” 또는 “위치의 객체에 대한 액세스가 거부되었습니다...(Access denied on object at location...)”인 경우 다음 요구 사항을 검토합니다.

- 로그인한 사용자는 Amazon S3의 블루프린트 ZIP 아카이브에 대한 읽기 권한이 있어야 합니다.
- ZIP 아카이브가 포함된 Amazon S3 버킷에는 AWS 계정 ID에 객체에 대한 읽기 권한을 부여하는 버킷 정책이 있어야 합니다. 자세한 내용은 [AWS Glue에서 블루프린트 개발](#) 단원을 참조하십시오.
- 사용 중인 Amazon S3 버킷은 콘솔에서 로그인한 리전과 동일한 리전에 있어야 합니다.

9. 데이터 분석가에게 블루프린트에 대한 권한이 있는지 확인합니다.

데이터 분석가를 위한 제안된 IAM 정책은 [블루프린트에 대한 데이터 분석가 권한](#)에 나와 있습니다. 이 정책은 모든 리소스에 glue:GetBlueprint를 부여합니다. 정책이 리소스 수준에서 보다 세분화된 경우 데이터 분석가에게 이 새로 생성된 리소스에 대한 권한을 부여합니다.

블루프린트를 등록하려면(AWS CLI)

1. 다음 명령을 입력합니다.


```
aws glue create-blueprint --name <blueprint-name> [--description <description>] --
blueprint-location s3://<s3-path>/<archive-filename>
```

2. 다음 명령을 입력하여 블루프린트 상태를 확인합니다. 상태가 ACTIVE 또는 FAILED가 될 때까지 명령을 반복합니다.

```
aws glue get-blueprint --name <blueprint-name>
```

상태가 FAILED이고 오류 메시지가 “위치의 객체에 액세스할 수 없습니다...(Unable to access object at location...)” 또는 “위치의 객체에 대한 액세스가 거부되었습니다...(Access denied on object at location...)”인 경우 다음 요구 사항을 검토합니다.

- 로그인한 사용자는 Amazon S3의 블루프린트 ZIP 아카이브에 대한 읽기 권한이 있어야 합니다.
- ZIP 아카이브가 포함된 Amazon S3 버킷에는 AWS 계정 ID에 객체에 대한 읽기 권한을 부여하는 버킷 정책이 있어야 합니다. 자세한 내용은 [블루프린트 게시](#) 단원을 참조하십시오.
- 사용 중인 Amazon S3 버킷은 콘솔에서 로그인한 리전과 동일한 리전에 있어야 합니다.

 다음 사항도 참조하세요.

- [AWS Glue의 블루프린트 개요](#)

AWS Glue에서 블루프린트 보기

블루프린트를 보고 블루프린트 설명, 상태 및 파라미터 사양을 검토하고 블루프린트 ZIP 아카이브를 다운로드합니다.

AWS Glue 콘솔, AWS Glue API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 블루프린트를 볼 수 있습니다.

블루프린트를 보려면(콘솔)

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 블루프린트(Blueprints)를 선택합니다.

3. 블루프린트 페이지에서 블루프린트를 선택합니다. [작업(Actions)] 메뉴에서 [보기(View)]을 선택합니다.

블루프린트를 보려면(AWS CLI)

- 블루프린트 이름, 설명 및 상태만 보려면 다음 명령을 입력합니다. `<blueprint-name>`을 보려는 블루프린트의 이름으로 바꿉니다.

```
aws glue get-blueprint --name <blueprint-name>
```

명령 출력은 다음과 같습니다.

```
{
  "Blueprint": {
    "Name": "myDemoBP",
    "CreatedOn": 1587414516.92,
    "LastModifiedOn": 1587428838.671,
    "BlueprintLocation": "s3://awsexamplebucket1/demo/
DemoBlueprintProject.zip",
    "Status": "ACTIVE"
  }
}
```

파라미터 사양도 보려면 다음 명령을 입력합니다.

```
aws glue get-blueprint --name <blueprint-name> --include-parameter-spec
```

명령 출력은 다음과 같습니다.


```
{
  "Blueprint": {
    "Name": "myDemoBP",
    "CreatedOn": 1587414516.92,
    "LastModifiedOn": 1587428838.671,
    "ParameterSpec": "{\"WorkflowName\":{\"type\":\"String\", \"collection
\":false, \"description\":null, \"defaultValue\":null, \"allowedValues\":null},
\"PassRole\":{\"type\":\"String\", \"collection\":false, \"description\":null,
\"defaultValue\":null, \"allowedValues\":null}, \"DynamoDBTableName\":{\"type
\": \"String\", \"collection\":false, \"description\":null, \"defaultValue\":null,
```

```

{"allowedValues\":"null},"ScriptLocation\":{"type\":"String"},"collection\":"false","description\":"null","defaultValue\":"null","allowedValues\":"null}}",
  "BlueprintLocation": "s3://awsexamplebucket1/demo/
DemoBlueprintProject.zip",
  "Status": "ACTIVE"
}
}

```

--include-blueprint 인수를 추가하여 AWS Glue가 저장한 블루프린트 ZIP 아카이브를 다운로드하기 위해 브라우저에 붙여넣을 수 있는 URL을 출력에 포함합니다.

 다음 사항도 참조하세요.

- [AWS Glue의 블루프린트 개요](#)

AWS Glue에서 블루프린트 업데이트

수정된 레이아웃 스크립트, 수정된 블루프린트 파라미터 집합 또는 수정된 지원 파일이 있는 경우 블루프린트를 업데이트할 수 있습니다. 블루프린트를 업데이트하면 새 버전이 생성됩니다.

블루프린트를 업데이트해도 블루프린트에서 생성된 기존 워크플로에는 영향이 없습니다.

AWS Glue 콘솔, AWS Glue API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 블루프린트를 업데이트할 수 있습니다.

다음 절차에서는 AWS Glue 개발자가 업데이트된 블루프린트 ZIP 아카이브를 생성하여 Amazon S3에 업로드했다고 가정합니다.

블루프린트를 업데이트하려면(콘솔)

1. Amazon S3의 블루프린트 ZIP 아카이브에 대한 읽기 권한(s3:GetObject)이 있는지 확인합니다.
2. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.

블루프린트를 업데이트할 권한이 있는 사용자로 로그인합니다. 블루프린트 ZIP 아카이브가 포함된 Amazon S3 버킷과 동일한 AWS 리전으로 전환합니다.

3. 탐색 창에서 블루프린트(Blueprints)를 선택합니다.

4. 블루프린트 페이지에서 블루프린트를 선택하고 작업(Actions) 메뉴에서 편집(Edit)을 선택합니다.
5. [블루프린터 편집(Edit a blueprint)] 페이지에서 블루프린트 [설명(Description)] 또는 [ZIP 아카이브 위치(S3)(ZIP archive location (S3))]를 업데이트합니다. 경로에 아카이브 파일 이름을 포함해야 합니다.
6. Save(저장)를 선택합니다.

블루프린트 페이지가 반환되고 블루프린트 상태가 UPDATING임을 표시합니다. 상태가 ACTIVE 또는 FAILED로 변경될 때까지 새로 고침 버튼을 선택합니다.

7. 상태가 FAILED인 경우 블루프린트를 선택하고 [작업(Actions)] 메뉴에서 [보기(View)]를 선택합니다.

세부 정보 페이지에 실패 이유가 표시됩니다. 오류 메시지가 “위치의 객체에 액세스할 수 없습니다...(Unable to access object at location...)” 또는 “위치의 객체에 대한 액세스가 거부되었습니다...(Access denied on object at location...)”인 경우 다음 요구 사항을 검토합니다.

- 로그인한 사용자는 Amazon S3의 블루프린트 ZIP 아카이브에 대한 읽기 권한이 있어야 합니다.
- ZIP 아카이브가 포함된 Amazon S3 버킷에는 AWS 계정 ID에 객체에 대한 읽기 권한을 부여하는 버킷 정책이 있어야 합니다. 자세한 내용은 [블루프린트 게시](#) 단원을 참조하십시오.
- 사용 중인 Amazon S3 버킷은 콘솔에서 로그인한 리전과 동일한 리전에 있어야 합니다.

Note

업데이트가 실패하면 성공적으로 등록되거나 업데이트된 블루프린트의 최신 버전이 다음 블루프린트 실행에 사용됩니다.

블루프린트를 업데이트하려면(AWS CLI)

1. 다음 명령을 입력합니다.


```
aws glue update-blueprint --name <blueprint-name> [--description <description>] --blueprint-location s3://<s3-path>/<archive-filename>
```

2. 다음 명령을 입력하여 블루프린트 상태를 확인합니다. 상태가 ACTIVE 또는 FAILED가 될 때까지 명령을 반복합니다.

```
aws glue get-blueprint --name <blueprint-name>
```

상태가 FAILED이고 오류 메시지가 “위치의 객체에 액세스할 수 없습니다...(Unable to access object at location...)” 또는 “위치의 객체에 대한 액세스가 거부되었습니다...(Access denied on object at location...)”인 경우 다음 요구 사항을 검토합니다.

- 로그인한 사용자는 Amazon S3의 블루프린트 ZIP 아카이브에 대한 읽기 권한이 있어야 합니다.
- ZIP 아카이브가 포함된 Amazon S3 버킷에는 AWS 계정 ID에 객체에 대한 읽기 권한을 부여하는 버킷 정책이 있어야 합니다. 자세한 내용은 [블루프린트 게시](#) 단원을 참조하십시오.
- 사용 중인 Amazon S3 버킷은 콘솔에서 로그인한 리전과 동일한 리전에 있어야 합니다.

 다음 사항도 참조하십시오.

- [AWS Glue의 블루프린트 개요](#)

AWS Glue의 블루프린트에서 워크플로 생성

AWS Glue 워크플로를 수동으로 생성하여 한 번에 하나의 구성 요소를 추가하거나 AWS Glue [블루프린트](#)에서 워크플로를 생성할 수 있습니다. AWS Glue에는 일반적인 사용 사례에 대한 블루프린트가 포함되어 있습니다. AWS Glue 개발자가 추가 블루프린트를 생성할 수 있습니다.

Important

워크플로 내 작업, 크롤러, 트리거의 총수를 100개 이하로 제한합니다. 100개가 넘게 포함될 경우 워크플로 실행을 재개하거나 중지하려고 할 때 오류가 발생할 수 있습니다.

블루프린트를 사용하면 블루프린트에서 정의한 일반화된 사용 사례를 기반으로 특정 사용 사례에 대한 워크플로를 빠르게 생성할 수 있습니다. 블루프린트 파라미터에 대한 값을 제공하여 특정 사용 사례를 정의합니다. 예를 들어 데이터 집합을 분할하는 블루프린트에는 Amazon S3 소스 및 대상 경로가 파라미터로 포함될 수 있습니다.

AWS Glue는 블루프린트를 실행하여 블루프린트에서 워크플로를 생성합니다. 블루프린트 실행은 제공한 파라미터 값을 저장하고 워크플로 및 해당 구성 요소 생성의 진행률과 결과를 추적하는 데 사용됩니다. 워크플로 문제를 해결할 때 블루프린트 실행을 보고 워크플로 생성에 사용된 블루프린트 파라미터 값을 확인할 수 있습니다.

워크플로를 생성하고 보려면 특정 IAM 권한이 필요합니다. 제안된 IAM 정책은 [블루프린트에 대한 데이터 분석가 권한](#) 섹션을 참조하세요.

AWS Glue 콘솔, AWS Glue API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 블루프린트에서 워크플로를 생성할 수 있습니다.

블루프린트에서 워크플로를 생성하려면(콘솔)

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.

워크플로를 생성할 권한이 있는 사용자로 로그인합니다.

2. 탐색 창에서 블루프린트(Blueprints)를 선택합니다.
3. 블루프린트를 선택하고 [작업(Actions)] 메뉴에서 [워크플로 생성(Create workflow)]을 선택합니다.
4. [<blueprint-name>에서 워크플로 생성(Create a workflow from <blueprint-name>)] 페이지에서 다음 정보를 입력합니다.

블루프린트 파라미터

블루프린트 파라미터는 블루프린트 설계에 따라 다릅니다. 파라미터에 대한 질문은 개발자에게 문의하세요. 블루프린트에는 일반적으로 워크플로 이름에 대한 파라미터가 포함됩니다.

IAM 역할

AWS Glue가 워크플로 및 해당 구성 요소를 생성하기 위해 맡는 역할입니다. 역할에는 워크플로, 작업, 크롤러 및 트리거를 생성하고 삭제할 수 있는 권한이 있어야 합니다. 역할에 제안되는 정책은 [블루프린트 역할에 대한 권한](#) 섹션을 참조하세요.

5. 제출을 선택합니다.

[블루프린트 세부 정보(Blueprint Details)] 페이지가 나타나고 아래쪽에 블루프린트 실행 목록이 표시됩니다.

6. 블루프린트 실행 목록의 최상위 블루프린트 실행에서 워크플로 생성 상태를 확인합니다.

초기 상태는 RUNNING입니다. 상태가 SUCCEEDED 또는 FAILED로 바뀔 때까지 새로 고침 버튼을 선택합니다.

7. 다음 중 하나를 수행하십시오.

- 완료 상태가 SUCCEEDED이면 [워크플로(Workflows)] 페이지로 이동하여 새로 생성된 워크플로를 선택하여 실행할 수 있습니다. 워크플로를 실행하기 전에 설계 그래프를 검토할 수 있습니다.

- 완료 상태가 FAILED인 경우 블루프린트를 선택하고 [작업(Actions)] 메뉴에서 [보기(View)]를 선택하여 오류 메시지를 봅니다.

워크플로와 블루프린트에 대한 자세한 내용은 다음 주제를 참조하세요.

- [AWS Glue의 워크플로 개요](#)
- [AWS Glue에서 블루프린트 업데이트](#)
- [AWS Glue에서 워크플로 수동 생성 및 구축](#)

AWS Glue에서 블루프린트 실행 보기

블루프린트 실행을 보고 다음 정보를 확인합니다.

- 생성된 워크플로의 이름입니다.
- 워크플로를 생성하는 데 사용된 블루프린트 파라미터 값입니다.
- 워크플로 생성 작업의 상태입니다.

AWS Glue 콘솔, AWS Glue API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 블루프린트 실행을 볼 수 있습니다.


블루프린트 실행을 보려면(콘솔)

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 블루프린트(Blueprints)를 선택합니다.
3. 블루프린트 페이지에서 블루프린트를 선택합니다. [작업(Actions)] 메뉴에서 [보기(View)]를 선택합니다.
4. [블루프린트 세부 정보(Blueprint Details)] 페이지 하단에서 블루프린트 실행을 선택하고 [작업(Actions)] 메뉴에서 [보기(View)]를 선택합니다.

블루프린트 실행을 보려면(AWS CLI)

- 다음 명령을 입력합니다. `<blueprint-name>`을 블루프린트의 이름으로 바꿉니다. `<blueprint-run-id>`를 블루프린트 실행 ID로 바꿉니다.

```
aws glue get-blueprint-run --blueprint-name <blueprint-name> --run-id <blueprint-run-id>
```


 다음 사항도 참조하세요.

- [AWS Glue의 블루프린트 개요](#)

AWS Glue용 AWS CloudFormation

AWS CloudFormation은 여러 AWS 리소스를 생성할 수 있는 서비스입니다. AWS Glue는 AWS Glue Data Catalog에 객체를 생성하는 API 작업을 제공합니다. 그러나 AWS CloudFormation 템플릿 파일에 AWS Glue 객체 및 기타 AWS 리소스 객체를 정의하고 생성할 때 용이할 수 있습니다. 그런 다음 객체 생성 작업을 자동화할 수 있습니다.

AWS CloudFormation은 단순화된 구문((JSON(JavaScript Object Notation) 또는 YAML(YAML Ain't Markup Language))을 제공하여 AWS 리소스의 생성을 보여줍니다. AWS CloudFormation 템플릿을 사용하여 데이터베이스, 테이블, 파티션, 크롤러, 분류자 및 연결과 같은 Data Catalog 객체를 정의할 수 있습니다. 작업, 트리거 및 개발 엔드포인트와 같은 ETL 객체를 정의할 수도 있습니다. 필요한 모든 AWS 리소스를 설명하는 템플릿을 생성하면 AWS CloudFormation이 해당 리소스의 프로비저닝과 구성을 담당합니다.

자세한 내용은 AWS CloudFormation User Guide의 [What Is AWS CloudFormation?](#) 및 [Working with AWS CloudFormation Templates](#)를 참조하세요.

AWS Glue와 호환되는 AWS CloudFormation 템플릿을 사용하려는 경우 관리자는 AWS CloudFormation과 AWS 서비스 및 의존 대상 작업에 액세스 권한을 부여해야 합니다. AWS CloudFormation 리소스를 생성하는 데 필요한 권한을 부여하려면 다음 정책을 AWS CloudFormation으로 작업하는 사용자에게 연결합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:*"
      ],
      "Resource": "*"
    }
  ]
}
```

다음 테이블은 작업을 포함하여 AWS CloudFormation 템플릿을 수행하도록 허용합니다. AWS 리소스 유형 및 속성 유형에 대한 정보 링크를 포함하고 AWS CloudFormation 템플릿에 추가할 수 있습니다.

AWS Glue 리소스	AWS CloudFormation 템플릿	AWS Glue 샘플
분류자	AWS::Glue::Classifier	Grok 분류자 , JSON 분류자 , XML 분류자
연결	AWS::Glue::Connection	MySQL 연결
크롤러	AWS::Glue::Crawler	Amazon S3 크롤러 , MySQL 크롤러
데이터베이스	AWS::Glue::Database	데이터베이스 비우기 , 데이터베이스 및 테이블
개발 엔드포인트	AWS::Glue::DevEndpoint	개발 엔드포인트
작업	AWS::Glue::Job	Amazon S3 작업 , JDBC 작업
기계 학습 변환	AWS::Glue::MLTransform	기계 학습 변환
데이터 품질 규칙 세트	AWS::Glue::DataQualityRuleset	데이터 품질 규칙 세트 , EventBridge 스케줄러에서 데이터 품질 규칙 세트
Partition	AWS::Glue::Partition	테이블 파티션
표	AWS::Glue::Table	데이터베이스의 테이블
트리거	AWS::Glue::Trigger	온디맨드 트리거 , 일정이 정해진 트리거 , 조건부 트리거

시작하려면 다음 샘플 템플릿을 사용하고 메타데이터로 사용자 지정합니다. AWS CloudFormation 콘솔을 사용하여 AWS CloudFormation 스택을 생성한 다음 객체를 AWS Glue 및 기타 관련 서비스에 추가합니다. AWS Glue 객체의 많은 필드는 선택 사항입니다. 이 템플릿은 필요한 필드 또는 작업 및 기능적 AWS Glue 객체에 필요한 필드를 보여줍니다.

AWS CloudFormation 템플릿은 JSON 또는 YAML 형식일 수 있습니다. 이런 예제의 YAML은 더 쉽게 읽게 사용됩니다. 예제는 설명문(#)을 포함하여 템플릿에서 정의된 값을 설명합니다.

AWS CloudFormation 템플릿에 Parameters 섹션을 포함시킬 수 있습니다. 이 섹션은 샘플 텍스트 또는 YAML 파일이 AWS CloudFormation 콘솔에 입력되어 스택을 생성할 경우 변경될 수 있습니다. 템플릿의 Resources 섹션에는 AWS Glue 구문 정의와 관련 객체가 포함됩니다. AWS CloudFormation 템플릿 구문 정의에는 더 세분화된 속성 구문을 포함한 속성이 포함될 수 있습니다. AWS Glue 객체는 모든 속성이 없어도 생성할 수 있습니다. 다음 샘플은 AWS Glue 객체를 생성할 수 있는 일반 속성 예제 값을 보여줍니다.

AWS Glue 데이터베이스의 샘플 AWS CloudFormation 템플릿

Data Catalog의 AWS Glue 데이터베이스는 메타데이터 테이블을 포함합니다. 데이터베이스는 적은 속성으로 구성되어 있고 AWS CloudFormation 템플릿으로 Data Catalog에 생성될 수 있습니다. 다음 샘플 템플릿을 제공하여 시작을 돕고 AWS Glue로 AWS CloudFormation 스택의 사용에 대하여 보여줍니다. 샘플 템플릿이 생성한 유일한 리소스는 `cfn-mysampledatabse`라는 데이터베이스입니다. YAML을 입력한 후, 샘플 텍스트를 편집하거나 AWS CloudFormation 콘솔 값을 변경하면 리소스를 변경할 수 있습니다.

다음은 AWS Glue 데이터베이스를 생성할 수 있는 일반 속성 예제 값을 보여줍니다. AWS Glue용 AWS CloudFormation 데이터베이스 템플릿에 대한 자세한 내용은 [AWS::Glue::Database](#)를 참조하세요.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CloudFormation template in YAML to demonstrate creating a database named
mysampledatabse
# The metadata created in the Data Catalog points to the flights public S3 bucket
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  CFNDatabaseName:
    Type: String
    Default: cfn-mysampledatabse

# Resources section defines metadata for the Data Catalog
Resources:
# Create an AWS Glue database
CFNDatabaseFlights:
  Type: AWS::Glue::Database
  Properties:
    # The database is created in the Data Catalog for your account
```

```

CatalogId: !Ref AWS::AccountId
DatabaseInput:
  # The name of the database is defined in the Parameters section above
  Name: !Ref CFNDatabaseName
  Description: Database to hold tables for flights data
  LocationUri: s3://crawler-public-us-east-1/flight/2016/csv/
  #Parameters: Leave AWS database parameters blank

```

AWS Glue 데이터베이스, 테이블 및 파티션의 샘플 AWS CloudFormation 템플릿

AWS Glue 테이블은 메타데이터를 포함하여 ETL 스크립트에서 진행하고자 하는 데이터의 구조 및 위치를 정의할 수 있습니다. 테이블 내에서 파티션을 정의하여 데이터를 병렬로 진행할 수 있습니다. 파티션은 키로 정의한 데이터 덩어리입니다. 예를 들어, 키로써 월을 사용하여 모든 1월 데이터는 동일한 파티션을 포함합니다. AWS Glue의 경우, 데이터베이스는 테이블을 포함하고 테이블은 파티션을 포함합니다.

다음 샘플에서는 AWS CloudFormation 템플릿을 사용하여 데이터베이스, 테이블 및 파티션을 채우는 방법을 보여줍니다. 기본 데이터 포맷은 csv이고 콤마(,)로 범위가 제한됩니다. 데이터베이스는 테이블을 포함하기 전에 존재하고 테이블은 파티션이 생성되기 전에 존재해야 하기 때문에 템플릿은 DependsOn을 사용하여 생성될 경우, 객체의 종속성을 정의합니다.

샘플 값은 공개적으로 사용 가능한 Amazon S3 버킷에서 항공 데이터를 포함하는 테이블을 정의합니다. 설명하자면 오직 몇몇 데이터 열과 하나의 파티션 키가 정의됩니다. 4개의 파티션은 Data Catalog에도 정의됩니다. 기본 데이터의 스토리지를 설명하는 어떤 필드는 StorageDescriptor 필드에서 보여줍니다.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CloudFormation template in YAML to demonstrate creating a database, a table,
and partitions
# The metadata created in the Data Catalog points to the flights public S3 bucket
#
# Parameters substituted in the Resources section
# These parameters are names of the resources created in the Data Catalog
Parameters:
  CFNDatabaseName:
    Type: String
    Default: cfn-database-flights-1

```

```
CFTableName1:
  Type: String
  Default: cfn-manual-table-flights-1
# Resources to create metadata in the Data Catalog
Resources:
###
# Create an AWS Glue database
CFNDatabaseFlights:
  Type: AWS::Glue::Database
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseInput:
      Name: !Ref CFNDatabaseName
      Description: Database to hold tables for flights data
###
# Create an AWS Glue table
CFNTableFlights:
  # Creating the table waits for the database to be created
  DependsOn: CFNDatabaseFlights
  Type: AWS::Glue::Table
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableInput:
      Name: !Ref CFNTableName1
      Description: Define the first few columns of the flights table
      TableType: EXTERNAL_TABLE
      Parameters: {
"classification": "csv"
}
#
  ViewExpandedText: String
  PartitionKeys:
    # Data is partitioned by month
    - Name: mon
      Type: bigint
  StorageDescriptor:
    OutputFormat: org.apache.hadoop.hive.q1.io.HiveIgnoreKeyTextOutputFormat
    Columns:
      - Name: year
        Type: bigint
      - Name: quarter
        Type: bigint
      - Name: month
        Type: bigint
```

```
- Name: day_of_month
  Type: bigint
  InputFormat: org.apache.hadoop.mapred.TextInputFormat
  Location: s3://crawler-public-us-east-1/flight/2016/csv/
  SerdeInfo:
    Parameters:
      field.delim: ","
    SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 1
# Create an AWS Glue partition
CFNPartitionMon1:
  DependsOn: CFNTableFlights
  Type: AWS::Glue::Partition
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableName: !Ref CFNTableName1
    PartitionInput:
      Values:
        - 1
    StorageDescriptor:
      OutputFormat: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
      Columns:
        - Name: mon
          Type: bigint
      InputFormat: org.apache.hadoop.mapred.TextInputFormat
      Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=1/
      SerdeInfo:
        Parameters:
          field.delim: ","
        SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 2
# Create an AWS Glue partition
CFNPartitionMon2:
  DependsOn: CFNTableFlights
  Type: AWS::Glue::Partition
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableName: !Ref CFNTableName1
    PartitionInput:
      Values:
        - 2
    StorageDescriptor:
```

```
    OutputFormat: org.apache.hadoop.hive.q1.io.HiveIgnoreKeyTextOutputFormat
    Columns:
    - Name: mon
      Type: bigint
    InputFormat: org.apache.hadoop.mapred.TextInputFormat
    Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=2/
    SerdeInfo:
      Parameters:
        field.delim: ","
      SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 3
# Create an AWS Glue partition
CFNPartitionMon3:
  DependsOn: CFNTableFlights
  Type: AWS::Glue::Partition
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableName: !Ref CFNTableName1
    PartitionInput:
      Values:
      - 3
    StorageDescriptor:
      OutputFormat: org.apache.hadoop.hive.q1.io.HiveIgnoreKeyTextOutputFormat
      Columns:
      - Name: mon
        Type: bigint
      InputFormat: org.apache.hadoop.mapred.TextInputFormat
      Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=3/
      SerdeInfo:
        Parameters:
          field.delim: ","
        SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
# Partition 4
# Create an AWS Glue partition
CFNPartitionMon4:
  DependsOn: CFNTableFlights
  Type: AWS::Glue::Partition
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseName: !Ref CFNDatabaseName
    TableName: !Ref CFNTableName1
    PartitionInput:
      Values:
```



```

- 4
StorageDescriptor:
  OutputFormat: org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat
  Columns:
    - Name: mon
      Type: bigint
  InputFormat: org.apache.hadoop.mapred.TextInputFormat
  Location: s3://crawler-public-us-east-1/flight/2016/csv/mon=4/
  SerdeInfo:
    Parameters:
      field.delim: ","
    SerializationLibrary: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

```

AWS Glue Grok 분류자의 샘플 AWS CloudFormation 템플릿

AWS Glue 분류자는 데이터의 스키마를 결정합니다. 사용자 분류자의 한 유형은 grok 패턴을 사용하여 데이터와 매치합니다. 패턴이 맞으면 사용자 분류자는 테이블 스키마를 생성하고 classification을 분류자 정의의 값 세트를 설정합니다.

이 샘플은 분류자를 생성하여 message라는 하나의 열과 함께 스키마를 생성하고 greedy의 분류를 설정합니다.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
CFNClassifierName:
  Type: String
  Default: cfn-classifier-grok-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses grok pattern to put all data in one column and classifies
it as "greedy".

```

```

CFNClassifierFlights:
  Type: AWS::Glue::Classifier
  Properties:
    GrokClassifier:
      #Grok classifier that puts all data in one column
      Name: !Ref CFNClassifierName
      Classification: greedy

      GrokPattern: "%{GREEDYDATA:message}"
      #CustomPatterns: none

```

AWS Glue JSON 분류자의 샘플 AWS CloudFormation 템플릿

AWS Glue 분류자는 데이터의 스키마를 결정합니다. 분류자가 분류해야 하는 JSON 데이터를 정의하는 JsonPath 문자열을 사용하는 유형의 사용자 지정 분류자입니다. AWS Glue은(는) JsonPath에 대한 연산자 하위 집단을 지원합니다. 자세한 설명은 [Writing JsonPath Custom Classifiers\(JsonPath 사용자 지정 분류자 작성\)](#)에서 확인할 수 있습니다.

패턴이 맞으면 사용자 분류자는 테이블 스키마를 생성하는 데 사용됩니다.

이 샘플은 객체 내의 Records3 어레이에 있는 개별 레코드로 스키마를 생성하는 분류자를 생성합니다.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a JSON classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
CFNClassifierName:
  Type: String
  Default: cfn-classifier-json-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses a JSON pattern.

```

```

CFNClassifierFlights:
  Type: AWS::Glue::Classifier
  Properties:
    JSONClassifier:
      #JSON classifier
      Name: !Ref CFNClassifierName
      JsonPath: $.Records3[*]

```

AWS Glue XML 분류자의 샘플 AWS CloudFormation 템플릿

AWS Glue 분류자는 데이터의 스키마를 결정합니다. 분석 중인 XML 문서에 각각의 레코드를 포함하는 요소를 지정하도록 XML 태그를 지정하는 유형의 사용자 지정 분류자입니다. 패턴이 맞으면 사용자 분류자는 테이블 스키마를 생성하고 classification을 분류자 정의의 값 세트를 설정합니다.

이 샘플은 분류자를 생성하여 Record 태그에 있는 각 레코드와 함께 스키마를 생성하고 XML의 분류를 설정합니다.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating an XML classifier
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the classifier to be created
CFNClassifierName:
  Type: String
  Default: cfn-classifier-xml-one-column-1

#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create classifier that uses the XML pattern and classifies it as "XML".
CFNClassifierFlights:
  Type: AWS::Glue::Classifier
  Properties:
    XMLClassifier:
      #XML classifier

```

```
Name: !Ref CFNClassifierName
Classification: XML
RowTag: <Records>
```

Amazon S3용 AWS Glue 크롤러의 샘플 AWS CloudFormation 템플릿

AWS Glue 크롤러는 데이터에 대응하는 Data Catalog에 메타데이터 테이블을 생성합니다. 그런 다음 이 테이블 정의를 ETL 작업의 원본 및 대상으로서 사용할 수 있습니다.

이 샘플은 크롤러, 필요한 IAM 역할 및 AWS Glue 데이터베이스를 Data Catalog에 생성합니다. 크롤러가 실행되면 IAM 역할을 가정하고 퍼블릭 항공 데이터의 데이터베이스에 테이블을 생성합니다. 테이블은 접두사 "cfn_sample_1_"로 생성합니다. 템플릿에 의해 생성된 IAM 역할은 글로벌 권한을 허용하여 사용자 역할을 생성하고자 합니다. 이 분류자는 사용자 지정 분류자를 정의하지 않습니다. 기본적으로 AWS Glue 내장 분류자가 사용됩니다.

이 샘플을 AWS CloudFormation 콘솔에 제출하면 IAM 역할을 생성하고자 하는 확신이 있어야 합니다.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a crawler
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
CFNCrawlerName:
  Type: String
  Default: cfn-crawler-flights-1
CFNDatabaseName:
  Type: String
  Default: cfn-database-flights-1
CFNTablePrefixName:
  Type: String
  Default: cfn_sample_1_
#
#
# Resources section defines metadata for the Data Catalog
Resources:
```

```
#Create IAM Role assumed by the crawler. For demonstration, this role is given all
permissions.
CFNRoleFlights:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        -
          Effect: "Allow"
          Principal:
            Service:
              - "glue.amazonaws.com"
          Action:
            - "sts:AssumeRole"
    Path: "/"
  Policies:
    -
      PolicyName: "root"
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
            Action: "*"
            Resource: "*"
# Create a database to contain tables created by the crawler
CFNDatabaseFlights:
  Type: AWS::Glue::Database
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseInput:
      Name: !Ref CFNDatabaseName
      Description: "AWS Glue container to hold metadata tables for the flights
crawler"
#Create a crawler to crawl the flights data on a public S3 bucket
CFNCrawlerFlights:
  Type: AWS::Glue::Crawler
  Properties:
    Name: !Ref CFNCrawlerName
    Role: !GetAtt CFNRoleFlights.Arn
    #Classifiers: none, use the default classifier
    Description: AWS Glue crawler to crawl flights data
    #Schedule: none, use default run-on-demand
```

```

DatabaseName: !Ref CFNDatabaseName
Targets:
  S3Targets:
    # Public S3 bucket with the flights data
    - Path: "s3://crawler-public-us-east-1/flight/2016/csv"
TablePrefix: !Ref CFNTableName
SchemaChangePolicy:
  UpdateBehavior: "UPDATE_IN_DATABASE"
  DeleteBehavior: "LOG"
Configuration: "{\"Version\":\"1.0\",\"CrawlerOutput\":{\"Partitions\":{\"AddOrUpdateBehavior\":\"InheritFromTable\"},\"Tables\":{\"AddOrUpdateBehavior\":\"MergeNewColumns\"}}}"

```

AWS Glue 연결의 샘플 AWS CloudFormation 템플릿

Data Catalog의 AWS Glue 연결은 JDBC 데이터베이스 연결에 필요한 JDBC 및 네트워크 정보를 포함합니다. 이 정보는 JDBC 데이터베이스에 연결하여 ETL 작업을 크롤하거나 실행할 때 사용됩니다.

이 샘플은 devdb라는 Amazon RDS MySQL 데이터베이스의 연결을 생성합니다. 이 연결이 사용되면 IAM 역할, 데이터베이스 자격 및 네트워크 연결 값도 제공되어야 합니다. 템플릿에 필요한 필드의 상세 정보를 참조하십시오.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a connection
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the connection to be created
CFNConnectionName:
  Type: String
  Default: cfn-connection-mysql-flights-1
CFNJDBCString:
  Type: String
  Default: "jdbc:mysql://xxx-mysql.yyyyyyyyyyyyyyy.us-east-1.rds.amazonaws.com:3306/
devdb"
CFNJDBCUser:
  Type: String
  Default: "master"

```

```

CFNJDBCPassword:
  Type: String
  Default: "12345678"
  NoEcho: true
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNConnectionMySQL:
    Type: AWS::Glue::Connection
    Properties:
      CatalogId: !Ref AWS::AccountId
      ConnectionInput:
        Description: "Connect to MySQL database."
        ConnectionType: "JDBC"
        #MatchCriteria: none
        PhysicalConnectionRequirements:
          AvailabilityZone: "us-east-1d"
          SecurityGroupIdList:
            - "sg-7d52b812"
          SubnetId: "subnet-84f326ee"
        ConnectionProperties: {
          "JDBC_CONNECTION_URL": !Ref CFNJDBCString,
          "USERNAME": !Ref CFNJDBCUser,
          "PASSWORD": !Ref CFNJDBCPassword
        }
      Name: !Ref CFNConnectionName

```

JDBC을 위한 AWS Glue 크롤러의 샘플 AWS CloudFormation 템플릿

AWS Glue 크롤러는 데이터에 대응하는 Data Catalog에 메타데이터 테이블을 생성합니다. 그런 다음 이 테이블 정의를 ETL 작업의 원본 및 대상으로서 사용할 수 있습니다.

이 샘플은 크롤러, 필요한 IAM 역할 및 AWS Glue 데이터베이스를 Data Catalog에 생성합니다. 크롤러가 실행되면 IAM 역할을 가정하고 MySQL 데이터베이스에 저장된 퍼블릭 항공 데이터의 데이터베이스에 테이블을 생성합니다. 테이블은 접두사 "cfn_jdbc_1_"로 생성합니다. 템플릿에 의해 생성된 IAM 역할은 글로벌 권한을 허용하여 사용자 역할을 생성하고자 합니다. JDBC 데이터는 사용자 지정 분류자를 정의할 수 없습니다. 기본적으로 AWS Glue 내장 분류자가 사용됩니다.

이 샘플을 AWS CloudFormation 콘솔에 제출하면 IAM 역할을 생성하고자 하는 확신이 있어야 합니다.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a crawler
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
  CFNCrawlerName:
    Type: String
    Default: cfn-crawler-jdbc-flights-1
# The name of the database to be created to contain tables
  CFNDatabaseName:
    Type: String
    Default: cfn-database-jdbc-flights-1
# The prefix for all tables crawled and created
  CFNTablePrefixName:
    Type: String
    Default: cfn_jdbc_1_
# The name of the existing connection to the MySQL database
  CFNConnectionName:
    Type: String
    Default: cfn-connection-mysql-flights-1
# The name of the JDBC path (database/schema/table) with wildcard (%) to crawl
  CFNJDBCPath:
    Type: String
    Default: saldev/%
#
#
# Resources section defines metadata for the Data Catalog
Resources:
#Create IAM Role assumed by the crawler. For demonstration, this role is given all
permissions.
  CFNRoleFlights:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          -
            Effect: "Allow"
```



```

Principal:
  Service:
    - "glue.amazonaws.com"
  Action:
    - "sts:AssumeRole"
Path: "/"
Policies:
  -
    PolicyName: "root"
    PolicyDocument:
      Version: "2012-10-17"
      Statement:
        -
          Effect: "Allow"
          Action: "*"
          Resource: "*"
# Create a database to contain tables created by the crawler
CFNDatabaseFlights:
  Type: AWS::Glue::Database
  Properties:
    CatalogId: !Ref AWS::AccountId
    DatabaseInput:
      Name: !Ref CFNDatabaseName
      Description: "AWS Glue container to hold metadata tables for the flights
crawler"
#Create a crawler to crawl the flights data in MySQL database
CFNCrawlerFlights:
  Type: AWS::Glue::Crawler
  Properties:
    Name: !Ref CFNCrawlerName
    Role: !GetAtt CFNRoleFlights.Arn
    #Classifiers: none, use the default classifier
    Description: AWS Glue crawler to crawl flights data
    #Schedule: none, use default run-on-demand
    DatabaseName: !Ref CFNDatabaseName
    Targets:
      JdbcTargets:
        # JDBC MySQL database with the flights data
        - ConnectionName: !Ref CFNConnectionName
          Path: !Ref CFNJDBCPath
        #Exclusions: none
    TablePrefix: !Ref CFNTablePrefixName
    SchemaChangePolicy:
      UpdateBehavior: "UPDATE_IN_DATABASE"

```

```

DeleteBehavior: "LOG"
Configuration: "{\"Version\":1.0,\"CrawlerOutput\":{\"Partitions\":{\"AddOrUpdateBehavior\":{\"InheritFromTable\"},\"Tables\":{\"AddOrUpdateBehavior\":{\"MergeNewColumns\"}}}}"}

```

Amazon S3에서 Amazon S3로의 AWS Glue 작업을 위한 샘플 AWS CloudFormation 템플릿

Data Catalog의 AWS Glue 작업은 AWS Glue의 스크립트를 실행하는 데 필요한 파라미터 값을 포함합니다.

이 샘플은 csv 포맷의 Amazon S3 버킷에서 항공 데이터를 읽고 Amazon S3 Parquet 파일에 작성하는 작업을 생성합니다. 이 작업에 실행되는 스크립트는 먼저 존재해야 합니다. AWS Glue 콘솔을 통해 환경의 ETL 스크립트를 생성할 수 있습니다. 이 작업이 실행되면 올바른 권한과 함께 IAM 역할도 제공되어야 합니다.

범용 파라미터 값은 템플릿에서 보여줍니다. 예를 들어 `AllocatedCapacity(DPU)`는 기본값 5로 돌아갑니다.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a job using the public flights S3 table in a
public bucket
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the job to be created
CFNJobName:
  Type: String
  Default: cfn-job-S3-to-S3-2
# The name of the IAM role that the job assumes. It must have access to data, script,
temporary directory
CFNIAMRoleName:
  Type: String
  Default: AWSGlueServiceRoleGA
# The S3 path where the script for this job is located
CFNScriptLocation:
  Type: String

```

```

Default: s3://aws-glue-scripts-123456789012-us-east-1/myid/sal-job-test2
#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create job to run script which accesses flightscsv table and write to S3 file as
parquet.
# The script already exists and is called by this job
CFNJobFlights:
  Type: AWS::Glue::Job
  Properties:
    Role: !Ref CFNIAMRoleName
    #DefaultArguments: JSON object
    # If script written in Scala, then set DefaultArguments={'--job-language';
'scala', '--class': 'your scala class'}
    #Connections: No connection needed for S3 to S3 job
    # ConnectionsList
    #MaxRetries: Double
    Description: Job created with CloudFormation
    #LogUri: String
    Command:
      Name: glueetl
      ScriptLocation: !Ref CFNScriptLocation
        # for access to directories use proper IAM role with permission to buckets
and folders that begin with "aws-glue-"
        # script uses temp directory from job definition if required (temp
directory not used S3 to S3)
        # script defines target for output as s3://aws-glue-target/sal
    AllocatedCapacity: 5
    ExecutionProperty:
      MaxConcurrentRuns: 1
    Name: !Ref CFNJobName

```

JDBC에서 Amazon S3로의 AWS Glue 작업용 샘플 AWS CloudFormation 템플릿

Data Catalog의 AWS Glue 작업은 AWS Glue의 스크립트를 실행하는 데 필요한 파라미터 값을 포함합니다.

이 샘플은 `cfn-connection-mysql-flights-1`이라는 연결에 의해 정의되어 MySQL JDBC 데이터베이스로부터 항공 데이터를 읽고 Amazon S3 Parquet 파일에 작성하는 작업을 생성합니다. 이 작업

에 실행되는 스크립트는 먼저 존재해야 합니다. AWS Glue 콘솔을 통해 환경의 ETL 스트립트를 생성할 수 있습니다. 이 작업이 실행되면 올바른 권한과 함께 IAM 역할도 제공되어야 합니다.

범용 파라미터 값은 템플릿에서 보여줍니다. 예를 들어 AllocatedCapacity(DPU)는 기본값 5로 돌아갑니다.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a job using a MySQL JDBC DB with the flights
  data to an S3 file
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the job to be created
  CFNJobName:
    Type: String
    Default: cfn-job-JDBC-to-S3-1
# The name of the IAM role that the job assumes. It must have access to data, script,
  temporary directory
  CFNIAMRoleName:
    Type: String
    Default: AWSGlueServiceRoleGA
# The S3 path where the script for this job is located
  CFNScriptLocation:
    Type: String
    Default: s3://aws-glue-scripts-123456789012-us-east-1/myid/sal-job-dec4a
# The name of the connection used for JDBC data source
  CFNConnectionName:
    Type: String
    Default: cfn-connection-mysql-flights-1
#
#
# Resources section defines metadata for the Data Catalog
Resources:
# Create job to run script which accesses JDBC flights table via a connection and write
  to S3 file as parquet.
# The script already exists and is called by this job
  CFNJobFlights:
    Type: AWS::Glue::Job
    Properties:
```

```

Role: !Ref CFNIAMRoleName
#DefaultArguments: JSON object
# For example, if required by script, set temporary directory as
DefaultArguments={'--TempDir'; 's3://aws-glue-temporary-xyc/sal'}
Connections:
  Connections:
    - !Ref CFNConnectionName
#MaxRetries: Double
Description: Job created with CloudFormation using existing script
#LogUri: String
Command:
  Name: glueetl
  ScriptLocation: !Ref CFNScriptLocation
    # for access to directories use proper IAM role with permission to buckets
    and folders that begin with "aws-glue-"
    # if required, script defines temp directory as argument TempDir and used
    in script like redshift_tmp_dir = args["TempDir"]
    # script defines target for output as s3://aws-glue-target/sal
AllocatedCapacity: 5
ExecutionProperty:
  MaxConcurrentRuns: 1
Name: !Ref CFNJobName

```

AWS Glue 온디맨드 트리거의 샘플 AWS CloudFormation 템플릿

Data Catalog의 AWS Glue 트리거는 트리거가 실행되어 작업이 시작될 때 필요한 파라미터 값을 포함합니다. 온디맨드 트리거는 작동 시 시작합니다.

이 예제에서는 `cfn-job-S3-to-S3-1`이라는 하나의 작업을 시작하는 온디맨드 트리거를 만듭니다.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating an on-demand trigger
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1

```

```

# The name of the trigger to be created
CFNTriggerName:
  Type: String
  Default: cfn-trigger-ondemand-flights-1
#
# Resources section defines metadata for the Data Catalog
# Sample CFN YAML to demonstrate creating an on-demand trigger for a job
Resources:
# Create trigger to run an existing job (CFNJobName) on an on-demand schedule.
CFNTriggerSample:
  Type: AWS::Glue::Trigger
  Properties:
    Name:
      Ref: CFNTriggerName
    Description: Trigger created with CloudFormation
    Type: ON_DEMAND
    Actions:
      - JobName: !Ref CFNJobName
      # Arguments: JSON object
    #Schedule:
    #Predicate:

```

AWS Glue 일정이 정해진 트리거를 위한 샘플 AWS CloudFormation 템플릿

Data Catalog의 AWS Glue 트리거는 트리거가 실행되어 작업이 시작될 때 필요한 파라미터 값을 포함합니다. 일정이 정해진 트리거는 활성화되고 cron 타이머가 뜨면 시작합니다.

이 예제에서는 `cfn-job-S3-to-S3-1`이라는 하나의 작업을 시작하는 일정이 정해진 트리거를 만듭니다. 이 타이머는 매 10분마다 주중에 작업을 실행하는 cron 표현식입니다.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a scheduled trigger
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:
  # The existing job to be started by this trigger
  CFNJobName:

```

```

    Type: String
    Default: cfn-job-S3-to-S3-1
# The name of the trigger to be created
CFNTriggerName:
    Type: String
    Default: cfn-trigger-scheduled-flights-1
#
# Resources section defines metadata for the Data Catalog
# Sample CFN YAML to demonstrate creating a scheduled trigger for a job
#
Resources:
# Create trigger to run an existing job (CFNJobName) on a cron schedule.
TriggerSample1CFN:
    Type: AWS::Glue::Trigger
    Properties:
        Name:
            Ref: CFNTriggerName
        Description: Trigger created with CloudFormation
        Type: SCHEDULED
        Actions:
            - JobName: !Ref CFNJobName
              # Arguments: JSON object
            # # Run the trigger every 10 minutes on Monday to Friday
            Schedule: cron(0/10 * ? * MON-FRI *)
            #Predicate:

```

AWS Glue 조건부 트리거의 샘플 AWS CloudFormation 템플릿

Data Catalog의 AWS Glue 트리거는 트리거가 실행되어 작업이 시작될 때 필요한 파라미터 값을 포함합니다. 성공적으로 작업을 완료하는 것처럼 활성화되고 조건이 맞으면 조건부 트리거는 시작됩니다.

이 예제에서는 `cfn-job-S3-to-S3-1`이라는 하나의 작업을 시작하는 조건부 트리거를 만듭니다. 이 작업은 `cfn-job-S3-to-S3-2` 라는 작업을 성공적으로 완료하면 시작합니다.

```

---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a conditional trigger for a job, which starts
  when another job completes
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog

```

```

Parameters:
  # The existing job to be started by this trigger
  CFNJobName:
    Type: String
    Default: cfn-job-S3-to-S3-1
  # The existing job that when it finishes causes trigger to fire
  CFNJobName2:
    Type: String
    Default: cfn-job-S3-to-S3-2
  # The name of the trigger to be created
  CFNTriggerName:
    Type: String
    Default: cfn-trigger-conditional-1
#
Resources:
  # Create trigger to run an existing job (CFNJobName) when another job completes
  (CFNJobName2).
  CFNTriggerSample:
    Type: AWS::Glue::Trigger
    Properties:
      Name:
        Ref: CFNTriggerName
      Description: Trigger created with CloudFormation
      Type: CONDITIONAL
      Actions:
        - JobName: !Ref CFNJobName
          # Arguments: JSON object
      #Schedule: none
      Predicate:
        #Value for Logical is required if more than 1 job listed in Conditions
        Logical: AND
        Conditions:
          - LogicalOperator: EQUALS
            JobName: !Ref CFNJobName2
            State: SUCCEEDED

```

AWS Glue 개발 엔드포인트의 샘플 AWS CloudFormation 템플릿

AWS Glue 기계 학습 변환은 데이터를 정리하기 위한 사용자 지정 변환입니다. 여기에는 FindMatches 로 명명된 사용 가능 변환이 하나 있습니다. FindMatches 변환으로는 레코드에 공통된 고유 식별자가 없고 정확히 일치되는 필드 또한 없을 경우에도 데이터 세트에서 중복 레코드나 일치 레코드를 식별할 수 있습니다.

이 샘플에서는 기계 학습 변환을 생성합니다. 기계 학습 변환을 생성하는 데 필요한 파라미터에 대한 자세한 내용을 알아보려면 [AWS Lake Formation FindMatches로 레코드 매칭](#) 섹션을 참조하세요.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a machine learning transform
#
# Resources section defines metadata for the machine learning transform
Resources:
  MyMLTransform:
    Type: "AWS::Glue::MLTransform"
    Condition: "isGlueMLGARegion"
    Properties:
      Name: !Sub "MyTransform"
      Description: "The bestest transform ever"
      Role: !ImportValue MyMLTransformUserRole
      GlueVersion: "1.0"
      WorkerType: "Standard"
      NumberOfWorkers: 5
      Timeout: 120
      MaxRetries: 1
      InputRecordTables:
        GlueTables:
          - DatabaseName: !ImportValue MyMLTransformDatabase
            TableName: !ImportValue MyMLTransformTable
      TransformParameters:
        TransformType: "FIND_MATCHES"
        FindMatchesParameters:
          PrimaryKeyColumnName: "testcolumn"
          PrecisionRecallTradeoff: 0.5
          AccuracyCostTradeoff: 0.5
          EnforceProvidedLabels: True
      Tags:
        key1: "value1"
        key2: "value2"
      TransformEncryption:
        TaskRunSecurityConfigurationName: !ImportValue
MyMLTransformSecurityConfiguration
      MLUserDataEncryption:
        MLUserDataEncryptionMode: "SSE-KMS"
        KmsKeyId: !ImportValue MyMLTransformEncryptionKey
```

AWS Glue Data Quality 규칙 세트에 대한 샘플 AWS CloudFormation 템플릿

AWS Glue 데이터 품질 규칙 세트에는 데이터 카탈로그의 테이블에서 평가할 수 있는 규칙이 포함되어 있습니다. 규칙 세트를 대상 테이블에 배치한 후에는 데이터 카탈로그로 이동하여 규칙 세트 내 해당 규칙에 대해 데이터를 실행하는 평가를 실행할 수 있습니다. 이러한 규칙은 행 수 평가에서 데이터에 대한 참조 무결성 평가에 이르기까지 다양할 수 있습니다.

다음 샘플은 지정된 대상 테이블에서 다양한 규칙이 포함된 규칙 세트를 생성하는 CloudFormation 템플릿입니다.

```
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a DataQualityRuleset
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the ruleset to be created
RulesetName:
  Type: String
  Default: "CFNRulesetName"
RulesetDescription:
  Type: String
  Default: "CFN DataQualityRuleset"
# Rules that will be associated with this ruleset
Rules:
  Type: String
  Default: 'Rules = [
    RowCount > 100,
    IsUnique "id",
    IsComplete "nametype"
  ]'
# Name of database and table within Data Catalog which the ruleset will
# be applied too
DatabaseName:
  Type: String
  Default: "ExampleDatabaseName"
TableName:
  Type: String
  Default: "ExampleTableName"
```

```
# Resources section defines metadata for the Data Catalog
Resources:
  # Creates a Data Quality ruleset under specified rules
  DQRuleset:
    Type: AWS::Glue::DataQualityRuleset
    Properties:
      Name: !Ref RulesetName
      Description: !Ref RulesetDescription
      # The String within rules must be formatted in DQDL, a language
      # used specifically to make rules
      Ruleset: !Ref Rules
      # The targeted table must exist within Data Catalog alongside
      # the correct database
      TargetTable:
        DatabaseName: !Ref DatabaseName
        TableName: !Ref TableName
```

EventBridge 스케줄러에서 AWS Glue Data Quality 규칙 세트에 대한 샘플 AWS CloudFormation 템플릿

AWS Glue 데이터 품질 규칙 세트에는 데이터 카탈로그의 테이블에서 평가할 수 있는 규칙이 포함되어 있습니다. 규칙 세트를 대상 테이블에 배치한 후에는 데이터 카탈로그로 이동하여 규칙 세트 내 해당 규칙에 대해 데이터를 실행하는 평가를 실행할 수 있습니다. 규칙 세트를 평가하기 위해 수동으로 데이터 카탈로그로 이동할 필요 없이, CloudFormation 템플릿 내에 EventBridge 스케줄러를 추가하여 일정 간격으로 이러한 규칙 세트 평가를 예약할 수도 있습니다.

다음 샘플은 데이터 품질 규칙 세트와 EventBridge 스케줄러를 생성하여 앞서 언급한 규칙 세트를 5분마다 평가하는 CloudFormation 템플릿입니다.

```
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a DataQualityRuleset
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

  # The name of the ruleset to be created
  RulesetName:
    Type: String
    Default: "CFNRulesetName"
  # Rules that will be associated with this Ruleset
```

```

Rules:
  Type: String
  Default: 'Rules = [
    RowCount > 100,
    IsUnique "id",
    IsComplete "nametype"
  ]'
# The name of the Schedule to be created
ScheduleName:
  Type: String
  Default: "ScheduleDQRulsetEvaluation"
# This expression determines the rate at which the Schedule will evaluate
# your data using the above ruleset
ScheduleRate:
  Type: String
  Default: "rate(5 minutes)"
# The Request that being sent must match the details of the Data Quality Ruleset
ScheduleRequest:
  Type: String
  Default: '
    { "DataSource": { "GlueTable": { "DatabaseName": "ExampleDatabaseName",
      "TableName": "ExampleTableName" } },
      "Role": "role/AWSGlueServiceRoleDefault",
      "RulesetNames": [ ""CFNRulesetName"" ] }
    ,

# Resources section defines metadata for the Data Catalog
Resources:
# Creates a Data Quality ruleset under specified rules
DQRuleset:
  Type: AWS::Glue::DataQualityRuleset
  Properties:
    Name: !Ref RulesetName
    Description: "CFN DataQualityRuleset"
    # The String within rules must be formatted in DQDL, a language
    # used specifically to make rules
    Ruleset: !Ref Rules
    # The targeted table must exist within Data Catalog alongside
    # the correct database
    TargetTable:
      DatabaseName: "ExampleDatabaseName"
      TableName: "ExampleTableName"
# Create a Scheduler to schedule evaluation runs on the above ruleset
ScheduleDQEval:

```

```

Type: AWS::Scheduler::Schedule
Properties:
  Name: !Ref ScheduleName
  Description: "Schedule DataQualityRuleset Evaluations"
  FlexibleTimeWindow:
    Mode: "OFF"
  ScheduleExpression: !Ref ScheduleRate
  ScheduleExpressionTimezone: "America/New_York"
  State: "ENABLED"
  Target:
    # The ARN is the API that will be run, since we want to evaluate our ruleset
    # we want this specific ARN
    Arn: "arn:aws:scheduler::aws-sdk:glue:startDataQualityRulesetEvaluationRun"
    # Your RoleArn must have approval to schedule
    RoleArn: "arn:aws:iam::123456789012:role/AWSGlueServiceRoleDefault"
    # This is the Request that is being sent to the Arn
    Input: '
      { "DataSource": { "GlueTable": { "DatabaseName": "sampledb", "TableName":
"meteorite" } },
        "Role": "role/AWSGlueServiceRoleDefault",
        "RulesetNames": [ "TestCFN" ] }
    '

```

AWS Glue 개발 엔드포인트의 샘플 AWS CloudFormation 템플릿

AWS Glue 개발 엔드포인트는 AWS Glue 스크립트를 개발하고 테스트하는 데 사용되는 환경입니다.

이 샘플은 성공적으로 샘플을 생성할 때 필요한 최소한의 네트워크 파라미터 값으로 개발 엔드포인트를 생성합니다. 개발 엔드포인트를 설정할 필요가 있는 파라미터에 대한 자세한 내용은 [AWS Glue의 개발에 대한 네트워킹 설정](#)을 참조하십시오.

기존 IAM 역할 ARN(Amazon Resource Name)을 제공하여 개발 엔드포인트를 생성합니다. 노트북 서버를 개발 엔드포인트에 생성하고자 한다면 유효한 RSA 퍼블릭 키를 제공하고 대응하는 프라이빗 키를 사용 가능한 상태로 유지합니다.

Note

개발 엔드포인트와 연결되어 생성한 어떤 노트북 서버도 관리합니다. 따라서 엔드포인트를 삭제하고자 한다면 AWS CloudFormation 콘솔에서 AWS CloudFormation 스택을 삭제해야 노트북 서버를 삭제할 수 있습니다.

```
---
AWSTemplateFormatVersion: '2010-09-09'
# Sample CFN YAML to demonstrate creating a development endpoint
#
# Parameters section contains names that are substituted in the Resources section
# These parameters are the names the resources created in the Data Catalog
Parameters:

# The name of the crawler to be created
CFNEndpointName:
  Type: String
  Default: cfn-devendpoint-1
CFNIAMRoleArn:
  Type: String
  Default: arn:aws:iam::123456789012/role/AWSGlueServiceRoleGA
#
#
# Resources section defines metadata for the Data Catalog
Resources:
  CFNDevEndpoint:
    Type: AWS::Glue::DevEndpoint
    Properties:
      EndpointName: !Ref CFNEndpointName
      #ExtraJarsS3Path: String
      #ExtraPythonLibsS3Path: String
      NumberOfNodes: 5
      PublicKey: ssh-rsa public.....key myuserid-key
      RoleArn: !Ref CFNIAMRoleArn
      SecurityGroupIds:
        - sg-64986c0b
      SubnetId: subnet-c67cccac
```

AWS Glue 프로그래밍 안내서

스크립트는 소스에서 데이터를 추출하고 데이터를 변환한 다음 대상으로 로드하는 스크립트를 실행하는 코드를 포함합니다. AWS Glue는 작업 시작 시 스크립트를 실행합니다.

AWS Glue ETL 스크립트는 Python 또는 Scala에 코딩될 수 있습니다. 모든 작업 유형을 Python으로 작성할 수 있지만 AWS Glue for Spark 작업은 Scala로도 작성할 수 있습니다. AWS Glue Studio에서 작업의 소스 코드 논리를 자동으로 생성하면 스크립트가 생성됩니다. 스크립트를 편집하거나 자체 스크립트를 제공하여 ETL 작업을 실행할 수 있습니다.

사용자 지정 스크립트 제공

스크립트는 AWS Glue에서 추출, 변환 및 로드(ETL) 작업을 수행합니다. 자동적으로 작업의 소스 코드 로직을 생성할 때 스크립트가 생성됩니다. 스크립트를 편집하거나 사용자 지정 스크립트를 제공할 수 있습니다.

AWS Glue의 자체 사용자 지정 스크립트를 제공하려면 다음 일반 절차를 따르십시오.

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. ETL 작업 탭을 선택한 다음 작업 생성 섹션을 확인합니다. 스크립트 편집기 옵션을 선택합니다.
3. This job runs(이 작업이 실행됩니다) 아래에서 다음 중 하나를 선택합니다.
 - 표준 문안 코드로 새 스크립트 생성
 - 기존 스크립트 업로드 및 편집
4. 작업 속성 페이지에서 사용자 지정 스크립트를 실행하는 데 필요한 IAM 역할을 선택합니다. 자세한 내용은 [AWS Glue의 ID 및 액세스 관리](#) 단원을 참조하십시오.
5. 스크립트가 참조하는 모든 연결을 선택합니다. 이런 객체는 필요한 JDBC 데이터 스토어로 연결이 필요합니다.

탄력적 네트워크 인터페이스는 Virtual Private Cloud(VPC)에서 인스턴스에 장착할 수 있는 가상 네트워크 인터페이스입니다. 스크립트에 사용된 데이터 스토어로 연결할 때 필요한 탄력적 네트워크 인터페이스를 선택합니다.

6. 작업 유형별 파라미터를 비롯한 추가 구성을 제공합니다. 작업 유형의 구성에 대한 자세한 내용은 [AWS Glue Studio를 사용하여 시각적 ETL 작업 구축](#) 섹션을 참조하세요.
7. 스크립트 탭에서 사용자 지정 스크립트를 붙여넣거나 작성합니다.

이 섹션의 콘텐츠를 사용하여 사용자 지정 스크립트를 작성하는 프로세스를 안내합니다.

AWS Glue의 작업 추가에 대한 자세한 내용은 [AWS Glue Studio를 사용하여 시각적 ETL 작업 구축](#) 단원을 참조하십시오.

단계별 지침은 AWS Glue 콘솔의 [작업 추가(Add job)] 튜토리얼을 참조하세요.

Spark 스크립트 프로그래밍

AWS Glue를 사용하면 테스트 및 실행 외에도 ETL(추출, 변환 및 로드) 스크립트를 쉽게 작성하거나 자동 생성할 수 있습니다. AWS Glue가 도입한 아파치 스파크의 확장성을 보여주고 Python 혹은 Scala에서 ETL 스크립트를 코딩하고 실행하는 방법의 예를 제공합니다.

Important

AWS Glue 버전에 따라 지원되는 Apache Spark 버전이 다릅니다. 사용자 지정 스크립트는 지원되는 Apache Spark 버전과 호환되어야 합니다. AWS Glue 버전에 대한 자세한 내용은 [Glue version job property](#) 단원을 참조하십시오.

주제

- [튜토리얼: Spark용 AWS Glue 스크립트 작성](#)
- [에서 AWS Glue ETL 스크립트 프로그래밍 PySpark](#)
- [Scala의 AWS Glue ETL 스크립트 프로그래밍](#)
- [AWS Glue for Spark ETL 스크립트 프로그래밍을 위한 기능 및 최적화](#)

튜토리얼: Spark용 AWS Glue 스크립트 작성

이 자습서에서는 AWS Glue 스크립트를 작성하는 프로세스를 소개합니다. 작업을 사용하여 일정에 따라 스크립트를 실행하거나, 대화형 세션을 사용하여 대화형으로 실행할 수 있습니다. 작업에 대한 자세한 내용은 [AWS Glue Studio를 사용하여 시각적 ETL 작업 구축](#) 섹션을 참조하세요. 대화형 세션에 대한 자세한 내용을 알아보려면 [the section called “AWS Glue 대화형 세션 개요”](#)을(를) 참조하세요.

AWS Glue Studio 비주얼 에디터는 AWS Glue 작업을 빌드하기 위한 그래픽, 코드 없는 인터페이스를 제공합니다. AWS Glue 스크립트는 시각적 작업을 뒷받침합니다. 이를 통해 Apache Spark 프로그램과 함께 작동하는 데 사용할 수 있는 확장된 도구 세트에 액세스할 수 있습니다. 네이티브 Spark API와

AWS Glue 스크립트 내에서 추출, 변환, 로드 (ETL) 워크플로를 지원하는 Glue AWS 라이브러리에 액세스할 수 있습니다.

이 자습서에서는 주차 티켓 데이터 세트를 추출, 변환 및 로드합니다. 이 작업을 수행하는 스크립트는 Glue Studio 비주얼 에디터를 소개하는 AWS 빅데이터 블로그의 [AWS Glue Studio로 ETL을 더 쉽게 만들기에서](#) 생성한 스크립트와 형식 및 기능이 동일합니다. AWS 작업에서 이 스크립트를 실행하면 시각적 작업과 비교하여 AWS Glue ETL 스크립트의 작동 방식을 확인할 수 있습니다. 이렇게 하면 시각적 작업에서 아직 사용할 수 없는 추가 기능을 사용할 수 있습니다.

이 자습서에서는 Python 언어 및 라이브러리를 사용합니다. Scala에서도 비슷한 기능을 사용할 수 있습니다. 이 튜토리얼을 완료한 후에는 샘플 Scala 스크립트를 생성하고 검사하여 Scala AWS Glue ETL 스크립트 작성 프로세스를 수행하는 방법을 이해할 수 있을 것입니다.

필수 조건

이 자습서의 사전 요구 사항은 다음과 같습니다.

- 템플릿을 실행하도록 안내하는 AWS Glue Studio 블로그 게시물과 동일한 사전 요구 사항입니다. [AWS CloudFormation](#)

이 템플릿은 AWS Glue Data Catalog를 사용하여 `s3://aws-bigdata-blog/artifacts/gluestudio/`에 제공되는 주차 티켓 데이터 세트를 관리합니다. 참조할 다음 리소스를 생성합니다.

- AWS Glue Studio역할 — AWS Glue 작업을 실행할 IAM 역할
- AWS Glue StudioAmazon S3Bucket — 블로그 관련 파일을 저장할 Amazon S3 버킷의 이름
- AWS Glue StudioTicketsYYZDB – AWS Glue 데이터 카탈로그 데이터베이스
- AWS Glue StudioTableTickets— 소스로 사용할 데이터 카탈로그 테이블
- AWS Glue StudioTableTrials— 원본으로 사용할 데이터 카탈로그 테이블
- AWS Glue StudioParkingTicketCount — 대상으로 사용할 데이터 카탈로그 테이블
- AWS Glue Studio 블로그 게시물에서 생성된 스크립트입니다. 블로그 게시물이 변경되면 다음 텍스트에서도 스크립트를 사용할 수 있습니다.

샘플 스크립트 생성

AWS Glue Studio 비주얼 에디터를 강력한 코드 생성 도구로 사용하여 작성하려는 스크립트를 위한 스케폴드를 만들 수 있습니다. 이 도구를 사용하여 샘플 스크립트를 생성합니다.

이 단계를 건너뛰려는 경우, 스크립트가 제공됩니다.

자습서 샘플 스크립트

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 bucket
S3bucket_node1 = glueContext.create_dynamic_frame.from_catalog(
    database="yyz-tickets", table_name="tickets", transformation_ctx="S3bucket_node1"
)

# Script generated for node ApplyMapping
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3bucket_node1,
    mappings=[
        ("tag_number_masked", "string", "tag_number_masked", "string"),
        ("date_of_infraction", "string", "date_of_infraction", "string"),
        ("ticket_date", "string", "ticket_date", "string"),
        ("ticket_number", "decimal", "ticket_number", "float"),
        ("officer", "decimal", "officer_name", "decimal"),
        ("infraction_code", "decimal", "infraction_code", "decimal"),
        ("infraction_description", "string", "infraction_description", "string"),
        ("set_fine_amount", "decimal", "set_fine_amount", "float"),
        ("time_of_infraction", "decimal", "time_of_infraction", "decimal"),
    ],
    transformation_ctx="ApplyMapping_node2",
)

# Script generated for node S3 bucket
S3bucket_node3 = glueContext.write_dynamic_frame.from_options(
    frame=ApplyMapping_node2,
    connection_type="s3",
    format="glueparquet",
    connection_options={"path": "s3://DOC-EXAMPLE-BUCKET", "partitionKeys": []},
```

```

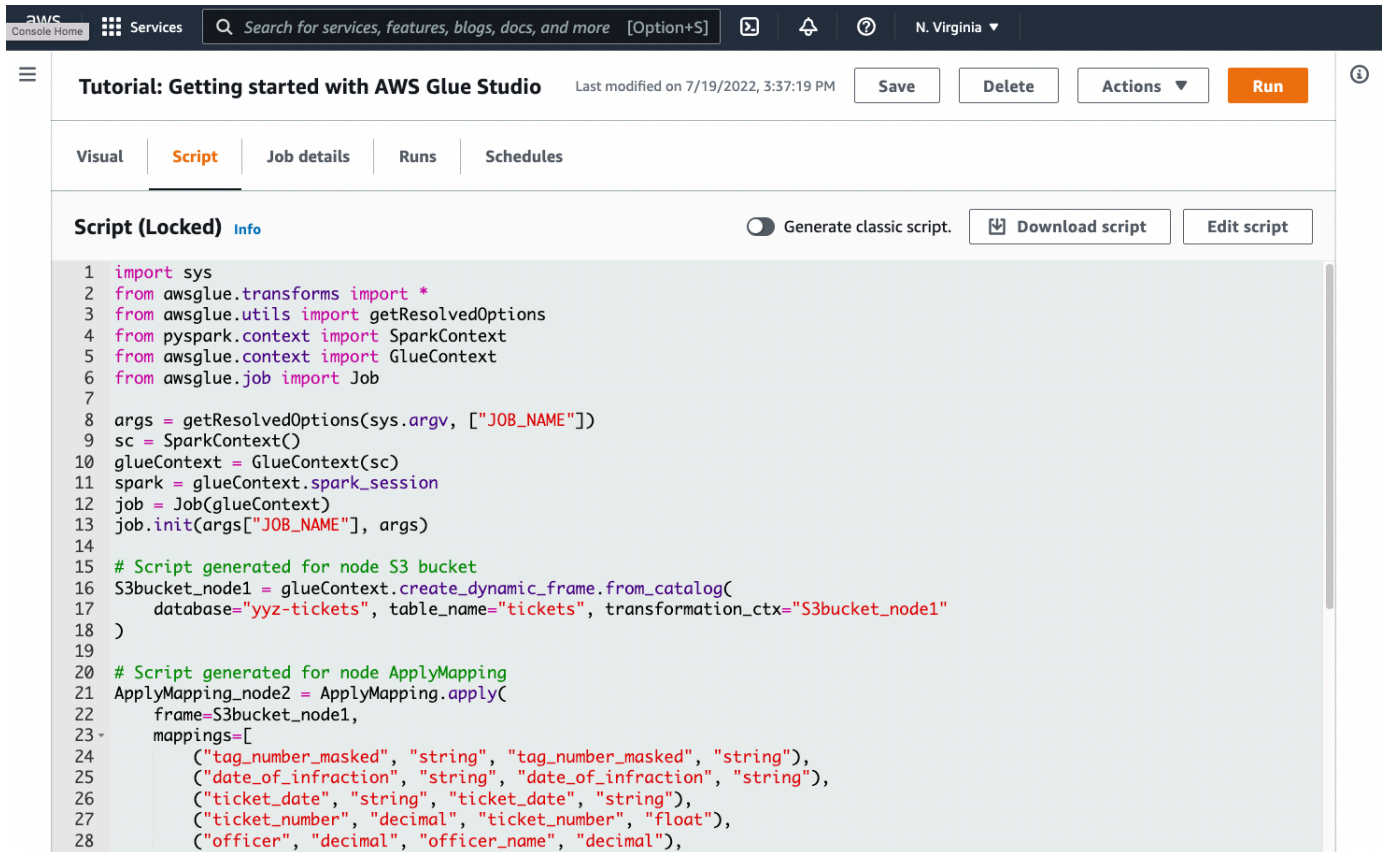
format_options={"compression": "gzip"},
transformation_ctx="S3bucket_node3",
)

job.commit()

```

샘플 스크립트 생성

1. AWS Glue 스튜디오 튜토리얼을 완료하세요. 이 튜토리얼을 완료하려면 [예제 작업에서 AWS Glue Studio에서 작업 생성을 참조하십시오](#).
2. 다음 스크린샷에 나와 있는 것처럼 작업 페이지의 Script(스크립트) 탭으로 이동합니다.



3. Script(스크립트) 탭의 전체 내용을 복사합니다. Job details(작업 세부 정보)에서 스크립트 언어를 설정하여 Python 또는 Scala 코드 생성 간을 전환할 수 있습니다.

단계 1. 작업 생성 및 스크립트 붙여넣기

이 단계에서는 에서 AWS Glue 작업을 생성합니다 AWS Management Console. 그러면 AWS Glue에서 스크립트를 실행할 수 있는 구성이 설정됩니다. 또한 스크립트를 저장하고 편집할 수 있는 장소가 생성됩니다.

작업을 생성하는 방법

1. 에서 AWS Management Console AWS Glue 랜딩 페이지로 이동합니다.
2. 왼쪽 탐색 창에서 Jobs(작업)을 선택합니다.
3. Create job(작업 생성)에서 Spark script editor(Spark 스크립트 편집기)를 선택한 다음 Create(생성)을 선택합니다.
4. 선택 사항 - 스크립트의 전체 텍스트를 Script(스크립트) 창에 붙여 넣습니다. 자습서를 따라 할 수도 있습니다.

단계 2. AWS Glue 라이브러리 가져오기

스크립트 외부에 정의된 코드 및 구성과 상호 작용하도록 스크립트를 설정해야 합니다. 이 작업은 AWS Glue Studio의 무대 뒤에서 수행됩니다.

이 단계에서 다음 작업을 수행합니다.

- GlueContext 객체를 가져오고 초기화합니다. 이는 스크립트 작성 관점에서 가장 중요한 가져오기입니다. 이를 통해 모든 ETL 스크립트의 시작점인 소스 및 대상 데이터 세트를 정의하기 위한 표준 메서드가 노출됩니다. GlueContext 클래스에 대한 자세한 내용을 알아보려면 [GlueContext 수업](#) 섹션을 참조하세요.
- SparkContext와 SparkSession을 초기화합니다. 이를 통해 AWS Glue 작업 내에서 사용할 수 있는 Spark 엔진을 구성할 수 있습니다. 입문용 AWS Glue 스크립트 내에서 직접 사용할 필요는 없습니다.
- getResolvedOptions를 호출하여 스크립트 내에서 사용할 작업 인수를 준비합니다. 작업 파라미터 확인에 대한 자세한 내용을 알아보려면 [the section called “getResolvedOptions 옵션”](#) 섹션을 참조하세요.
- Job을 초기화합니다. Job객체는 구성을 설정하고 다양한 선택적 AWS Glue 기능의 상태를 추적합니다. 스크립트는 Job 객체 없이 실행될 수 있지만, 이러한 기능이 나중에 통합되는 경우 혼동을 피하기 위해 작업을 초기화하는 것이 좋습니다.

이러한 기능 중 하나는 이 자습서에서 선택적으로 구성할 수 있는 작업 북마크입니다. [the section called “선택 사항 - 작업 북마크 활성화”](#) 섹션에서 작업 북마크에 대해 알아볼 수 있습니다.

이 절차에서는 다음 코드를 작성합니다. 이 코드는 생성된 샘플 스크립트의 일부입니다.

```
from awsglue.transforms import *
```

```

from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

```

AWS Glue 라이브러리를 가져오려면

- 코드의 이 섹션을 복사하여 Script(스크립트) 편집기에 붙여 넣습니다.

Note

코드 복사는 잘못된 엔지니어링 관행으로 간주될 수도 있습니다. 이 자습서에서는 모든 AWS Glue ETL 스크립트에서 핵심 변수의 이름을 일관되게 지정하도록 권장하기 위해 이 방법을 제안합니다.

단계 3. 소스에서 데이터 추출

모든 ETL 프로세스에서 먼저 변경하려는 소스 데이터 세트를 정의해야 합니다. AWS Glue Studio 비주얼 에디터에서 소스 노드를 생성하여 이 정보를 제공합니다.

이 단계에서는 AWS Glue Data Catalog에 구성된 소스에서 데이터를 추출할 database 및 table_name을(를) create_dynamic_frame.from_catalog 메서드에 제공합니다.

전 단계에서 GlueContext 객체를 초기화했습니다. 이 객체를 사용하여 소스를 구성하는 데 사용되는 create_dynamic_frame.from_catalog와(과) 같은 메서드를 찾습니다.

이 절차에서는 create_dynamic_frame.from_catalog를 사용하여 다음 코드를 작성합니다. 이 코드는 생성된 샘플 스크립트의 일부입니다.

```

S3bucket_node1 = glueContext.create_dynamic_frame.from_catalog(
    database="yyz-tickets", table_name="tickets", transformation_ctx="S3bucket_node1"
)

```

소스에서 데이터 추출

1. 설명서를 검토하여 AWS Glue Data Catalog에 GlueContext 정의된 소스에서 데이터를 추출하는 방법을 찾아보세요. 이러한 메서드는 [the section called “GlueContext”](#)에 문서화되어 있습니다. [create_dynamic_frame_from_catalog](#) 메서드를 선택합니다. glueContext에서 이 메서드를 호출합니다.
2. 설명서에서 create_dynamic_frame_from_catalog를 검토합니다. 이 메서드에는 database 및 table_name 파라미터가 필요합니다. create_dynamic_frame_from_catalog에 필요한 파라미터를 제공합니다.

AWS Glue Data Catalog는 소스 데이터의 위치 및 형식에 대한 정보를 저장하며 사전 요구 사항 섹션에서 설정되었습니다. 스크립트에 해당 정보를 직접 제공할 필요는 없습니다.

3. 선택 사항 - 작업 책갈피를 지원하려면 메서드에 transformation_ctx 파라미터를 제공합니다. [the section called “선택 사항 - 작업 북마크 활성화”](#) 섹션에서 작업 북마크에 대해 알아볼 수 있습니다.

Note

데이터를 추출하는 데 일반적으로 사용되는 메서드

[the section called “create_dynamic_frame_from_catalog”](#) AWS Glue 데이터 카탈로그의 테이블에 연결하는 데 사용됩니다.

소스의 구조와 위치를 설명하는 구성을 작업에 직접 제공해야 하는 경우 [the section called “create_dynamic_frame_from_options”](#) 메서드를 참조하세요.

create_dynamic_frame_from_catalog를 사용할 때보다 데이터를 설명하는 더 자세한 파라미터를 제공해야 합니다.

필요한 파라미터를 식별하려면 format_options 및 connection_parameters에 대한 보충 문서를 참조하세요. 소스 데이터 형식에 대한 스크립트 정보를 제공하는 방법에 대한 설명을 보려면 [the section called “데이터 포맷 옵션”](#) 섹션을 참조하세요. 소스 데이터 위치에 대한 스크립트 정보를 제공하는 방법에 대한 설명을 보려면 [the section called “연결 파라미터”](#) 섹션을 참조하세요.

스트리밍 소스에서 정보를 읽는 경우 [the section called “create_data_frame_from_catalog”](#) 또는 [the section called “create_data_frame_from_options”](#) 메서드를 통해 소스 정보를 작업에 제공합니다. 이러한 메서드는 Apache Spark DataFrames를 반환합니다.

생성된 코드는 create_dynamic_frame_from_catalog를 호출하는 반면 참조 문서는 create_dynamic_frame_from_catalog를 참조합니다. 이러한 메서드는 최종적으로 동일

한 코드를 호출하며 더 깔끔한 코드를 작성할 수 있도록 포함되어 있습니다. [aws-glue-libs](#)에서 제공되는 Python 래퍼의 소스를 보고 이를 확인할 수 있습니다.

4단계. AWS Glue로 데이터 변환

ETL 프로세스에서 소스 데이터를 추출한 후 데이터를 어떻게 변경할지 설명해야 합니다. AWS Glue Studio 비주얼 에디터에서 트랜스폼 노드를 생성하여 이 정보를 제공합니다.

이 단계에서는 ApplyMapping 메서드에 현재 필드 이름 및 유형과 원하는 필드 이름 및 유형의 맵을 제공하여 DynamicFrame을(를) 변환합니다.

다음 변환을 수행할 수 있습니다.

- 4개의 location 및 province 키를 삭제합니다.
- officer의 이름을 officer_name으로 변경합니다.
- ticket_number와 set_fine_amount의 유형을 float로 변경합니다.

`create_dynamic_frame.from_catalog`가 DynamicFrame 객체를 제공합니다. A는 AWS Glue의 데이터세트를 DynamicFrame 나타냅니다. AWS Glue 변환은 변화하는 DynamicFrames 작업입니다.

Note

DynamicFrame란 무엇입니까?

DynamicFrame은 데이터에 있는 항목의 이름 및 유형에 대한 설명과 함께 데이터 세트를 연결할 수 있는 추상화입니다. Apache Spark에는 a라는 유사한 추상화가 있습니다. DataFrame에 대한 설명은 [Spark DataFrames](#) SQL 가이드를 참조하십시오.

DynamicFrames을(를) 사용하면 데이터 세트 스키마를 동적으로 설명할 수 있습니다. 가격 열이 있는 데이터세트를 예로 들어 보겠습니다. 이 데이터셋에서는 가격을 문자열로 저장하는 항목도 있고 가격을 두 배로 저장하는 항목도 있습니다. AWS Glue는 스키마를 계산하여 on-the-fly 각 행에 대해 자체 설명 레코드를 생성합니다.

가격과 같은 불일치 필드는 프레임 스키마에서 유형(ChoiceType)으로 명시적으로 표시됩니다. 불일치 필드는 DropFields(으)로 삭제하거나 ResolveChoice(으)로 확인하여 해결할 수 있습니다. 다음은 DynamicFrame에서 사용할 수 있는 변환입니다. 그런 다음 writeDynamicFrame을 사용하여 데이터를 데이터 레이크에 다시 쓸 수 있습니다.

DynamicFrame 클래스의 메서드에서 동일한 변환을 많이 호출할 수 있으므로 스크립트를 더 쉽게 읽을 수 있습니다. DynamicFrame에 대한 자세한 정보는 [the section called “DynamicFrame”](#) 섹션을 참조하십시오.

이 절차에서는 ApplyMapping를 사용하여 다음 코드를 작성합니다. 이 코드는 생성된 샘플 스크립트의 일부입니다.

```
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3bucket_node1,
    mappings=[
        ("tag_number_masked", "string", "tag_number_masked", "string"),
        ("date_of_infraction", "string", "date_of_infraction", "string"),
        ("ticket_date", "string", "ticket_date", "string"),
        ("ticket_number", "decimal", "ticket_number", "float"),
        ("officer", "decimal", "officer_name", "decimal"),
        ("infraction_code", "decimal", "infraction_code", "decimal"),
        ("infraction_description", "string", "infraction_description", "string"),
        ("set_fine_amount", "decimal", "set_fine_amount", "float"),
        ("time_of_infraction", "decimal", "time_of_infraction", "decimal"),
    ],
    transformation_ctx="ApplyMapping_node2",
)
```

AWS Glue를 사용하여 데이터를 변환하려면

1. 설명서를 검토하여 필드를 변경하고 삭제하는 변환을 식별합니다. 자세한 내용은 [the section called “GlueTransform”](#) 단원을 참조하세요. ApplyMapping 변환을 선택합니다. ApplyMapping에 대한 자세한 정보는 [the section called “ApplyMapping”](#) 섹션을 참조하십시오. ApplyMapping 변환 객체에서 apply를 호출합니다.

Note

ApplyMapping이란 무엇인가요?

ApplyMapping은 DynamicFrame을 가져와서 변환합니다. 이는 필드의 변환을 나타내는 튜플 목록인 '매핑'을 가져옵니다. 처음 두 튜플 요소인 필드 이름과 유형은 프레임에서 필드를 식별하는 데 사용됩니다. 두 번째 두 파라미터는 필드 이름과 유형이기도 합니다. ApplyMapping 소스 필드를 대상 이름으로 변환하고 새 DynamicFrame 이름을 입력하면 반환됩니다. 제공되지 않은 필드는 반환 값에서 삭제됩니다.

apply를 호출하는 대신 DynamicFrame에서 apply_mapping 메서드로 동일한 변환을 호출하여 보다 매끄럽고 읽기 쉬운 코드를 생성할 수 있습니다. 자세한 정보는 [the section called “apply_mapping”](#)을 참조하세요.

2. 설명서에서 ApplyMapping을 검토하여 필수 파라미터를 식별합니다. [the section called “ApplyMapping”](#) 섹션을 참조하십시오. 이 메서드에는 frame 및 mappings 파라미터가 필요합니다. ApplyMapping에 필요한 파라미터를 제공합니다.
3. 선택 사항 - 작업 책갈피를 지원하려면 메서드에 transformation_ctx을(를) 제공합니다. [the section called “선택 사항 - 작업 북마크 활성화”](#) 섹션에서 작업 북마크에 대해 알아볼 수 있습니다.

Note

Apache Spark 기능

작업 내에서 ETL 워크플로를 간소화하기 위한 변환을 제공합니다. 또한 작업 중에 보다 일반적인 용도로 구축된 Spark 프로그램에서 사용할 수 있는 라이브러리에 액세스할 수 있습니다. 이러한 라이브러리를 사용하기 위해 DynamicFrame와(과) DataFrame 간을 변환합니다.

[the section called “toDF”](#)(으)로 DataFrame을(를) 생성할 수 있습니다. 그런 다음 에서 사용 가능한 메서드를 사용하여 데이터셋을 DataFrame 변환할 수 있습니다. 이러한 방법에 대한 자세한 내용은 을 참조하십시오 [DataFrame](#). 그런 다음 AWS Glue 작업을 사용하여 프레임을 타겟에 [the section called “fromDF”](#) 로드하도록 역방향으로 변환할 수 있습니다.

5단계. 대상에 데이터 로드

데이터를 변환한 후 일반적으로 변환된 데이터를 소스와 다른 장소에 저장합니다. 이 작업은 AWS Glue Studio 비주얼 에디터에서 대상 노드를 생성하여 수행합니다.

이 단계에서는 `write_dynamic_frame.from_options` 메서드에 `connection_type`, `connection_options`, `format` 및 `format_options`을(를) 제공하여 Amazon S3의 대상 버킷에 데이터를 로드합니다.

1단계에서 GlueContext 객체를 초기화했습니다. AWS Glue에서는 소스와 마찬가지로 타겟을 구성하는 데 사용되는 메서드를 찾을 수 있습니다.

이 절차에서는 `write_dynamic_frame.from_options`를 사용하여 다음 코드를 작성합니다. 이 코드는 생성된 샘플 스크립트의 일부입니다.

```
S3bucket_node3 = glueContext.write_dynamic_frame.from_options(
```

```

frame=ApplyMapping_node2,
connection_type="s3",
format="glueparquet",
connection_options={"path": "s3://DOC-EXAMPLE-BUCKET", "partitionKeys": []},
format_options={"compression": "gzip"},
transformation_ctx="S3bucket_node3",
)

```

대상에 데이터 로드

1. 설명서를 검토하여 대상 Amazon S3 버킷에 데이터를 로드하는 방법을 찾습니다. 이러한 메서드는 [the section called “GlueContext”](#)에 문서화되어 있습니다. [the section called “write_dynamic_frame_from_options”](#) 메서드를 선택합니다. glueContext에서 이 메서드를 호출합니다.

Note

데이터를 로드하는 데 일반적으로 사용되는 메서드 write_dynamic_frame.from_options는 데이터를 로드하는 데 가장 일반적으로 사용되는 메서드로서, AWS Glue에서 사용할 수 있는 모든 타겟을 지원합니다.

AWS Glue 연결에 정의된 JDBC 타겟에 쓰는 경우 메서드를 [the section called “write_dynamic_frame_from_jdbc_conf”](#) 사용하십시오. AWS Glue 연결은 데이터 원본에 연결하는 방법에 대한 정보를 저장합니다. 이렇게 하면 해당 정보를 connection_options에 제공할 필요가 없습니다. 그러나 dbtable을(를) 제공하려면 여전히 connection_options을(를) 사용해야 합니다.

write_dynamic_frame.from_catalog는 데이터를 로드하는 데 일반적으로 사용되는 메서드가 아닙니다. 이 메서드는 기본 데이터세트를 업데이트하지 않고 AWS Glue Data Catalog를 업데이트하며, 기본 데이터세트를 변경하는 다른 프로세스와 함께 사용됩니다. 자세한 정보는 [the section called “스키마 업데이트 및 새 파티션 추가”](#)을 참조하십시오.

2. 설명서에서 [the section called “write_dynamic_frame_from_options”](#)를 검토합니다. 이 메서드에는 frame, connection_type, format, connection_options 및 format_options이(가) 필요합니다. glueContext에서 이 메서드를 호출합니다.
 - a. 필요한 파라미터를 식별하려면 format_options와 format에 대한 보충 문서를 참조하십시오. 데이터 형식에 대한 설명을 보려면 [the section called “데이터 포맷 옵션”](#) 섹션을 참조하십시오.

- b. 필요한 파라미터를 식별하려면 `connection_type`와 `connection_options`에 대한 보충 문서를 참조하세요. 연결에 대한 설명을 보려면 [the section called “연결 파라미터”](#) 섹션을 참조하세요.
 - c. `write_dynamic_frame.from_options`에 필요한 파라미터를 제공합니다. 이 메서드는 `create_dynamic_frame.from_options`와(과) 유사한 구성을 가지고 있습니다.
3. 선택 사항 – 작업 책갈피를 지원하려면 `write_dynamic_frame.from_options`에 `transformation_ctx`을(를) 제공합니다. [the section called “선택 사항 - 작업 북마크 활성화”](#) 섹션에서 작업 북마크에 대해 알아볼 수 있습니다.

6단계. Job 객체 커밋

1단계에서 Job 객체를 초기화했습니다. 스크립트가 끝날 때 수명 주기를 수동으로 종료해야 합니다. 특정 옵션 기능이 제대로 작동하려면 이 기능이 필요합니다. 이 작업은 AWS Glue Studio의 무대 뒤에서 수행됩니다.

이 단계에서는 Job 객체에 대한 `commit` 메서드를 호출합니다.

이 절차에서는 다음 코드를 작성합니다. 이 코드는 생성된 샘플 스크립트의 일부입니다.

```
job.commit()
```

Job 객체 커밋

1. 이 단계를 아직 수행하지 않은 경우 이전 섹션에 설명된 선택적 단계를 수행하여 `transformation_ctx`을(를) 포함합니다.
2. `commit`을 호출합니다.

선택 사항 - 작업 북마크 활성화

모든 이전 단계에서 `transformation_ctx` 파라미터를 설정하라는 지시를 받았습니다. 이는 작업 북마크라는 기능과 관련이 있습니다.

작업 북마크를 사용하면 이전 작업을 쉽게 추적할 수 있는 데이터 세트에 대해 반복적으로 실행되는 작업을 통해 시간과 비용을 절약할 수 있습니다. Job 북마크는 이전 실행의 데이터세트에서 AWS Glue 변환의 진행 상황을 추적합니다. AWS Glue는 이전 실행이 종료된 위치를 추적하여 이전에 처리하지 않은 행으로 작업을 제한할 수 있습니다. 작업 북마크에 대한 자세한 내용을 알아보려면 [the section called “처리된 데이터를 작업 북마크로 추적”](#)을(를) 참조하세요.

작업 북마크를 활성화하려면 먼저 이전 예제에서 설명한 대로 제공된 함수에 `transformation_ctx` 문을 추가합니다. 작업 북마크 상태는 실행 시에도 유지됩니다. `transformation_ctx` 파라미터는 해당 상태에 액세스하는 데 사용되는 키입니다. 이러한 문은 그 자체로는 아무 것도 하지 않습니다. 작업에 대한 구성에서도 기능을 활성화해야 합니다.

이 절차에서는 AWS Management Console을(를) 사용하여 작업 북마크를 활성화합니다.

작업 북마크 활성화

1. 해당 작업의 Job details(작업 세부 정보) 섹션으로 이동합니다.
2. Job bookmark(작업 북마크)를 Enable(활성화)로 설정합니다.

7단계. 코드를 작업으로 실행

이 단계에서는 작업을 실행하여 이 자습서를 성공적으로 완료했는지 확인합니다. 이 작업은 AWS Glue Studio 비주얼 에디터에서처럼 버튼 클릭 한 번으로 완료됩니다.

코드를 작업으로 실행

1. 제목 표시줄에서 Untitled job(제목 없는 작업)을 선택하여 작업 이름을 편집하고 설정합니다.
2. Job details(작업 세부 정보) 탭으로 이동합니다. 작업에 IAM Role(IAM 역할)을 할당합니다. AWS Glue Studio 튜토리얼의 사전 요구 사항에서 AWS CloudFormation 템플릿으로 만든 템플릿을 사용할 수 있습니다. 해당 자습서를 완료했다면 AWS Glue StudioRole을 사용할 수 있습니다.
3. Save(저장)를 선택하여 스크립트를 저장합니다.
4. Run(실행)을 선택하여 작업을 실행합니다.
5. Runs(실행) 탭으로 이동하여 작업이 완료되었는지 확인합니다.
6. `write_dynamic_frame.from_options`의 대상인 `DOC-EXAMPLE-BUCKET`으로 이동합니다. 출력이 예상과 일치하는지 확인합니다.

작업 구성 및 관리에 대한 자세한 내용을 알아보려면 [the section called “사용자 지정 스크립트 제공”](#)을(를) 참조하세요.

추가 정보

Apache Spark 라이브러리 및 메서드는 AWS Glue 스크립트에서 사용할 수 있습니다. 포함된 라이브러리로 수행할 수 있는 작업을 알아보려면 Spark 설명서를 참조하세요. 자세한 내용을 알아보려면 [Spark 소스 리포지토리의 예제 섹션](#)을 참조하세요.

AWS Glue 2.0+에는 기본적으로 몇 가지 일반적인 Python 라이브러리가 포함되어 있습니다. Scala 또는 Python 환경의 AWS Glue 작업에 종속 항목을 로드하는 메커니즘도 있습니다. Python 종속성에 대한 내용을 알아보려면 [the section called “Python 라이브러리”](#)을(를) 참조하세요.

Python에서 AWS Glue 기능을 사용하는 방법에 대한 추가 예제는 [the section called “Python 샘플”](#). Scala 및 Python 작업은 기능 패리티가 있으므로 Python 예제를 통해 Scala에서 비슷한 작업을 수행하는 방법을 알아볼 수 있습니다.

에서 AWS Glue ETL 스크립트 프로그래밍 PySpark

GitHub 웹 사이트의 [AWS Glue 샘플 저장소에서](#) Python 코드 예제와 유틸리티를 찾을 수 있습니다. AWS Glue

AWS Glue와 함께 Python 사용하기

AWS Glue는 스크립팅 추출, 변환, 로드 (ETL) 작업을 위한 PySpark Python 언어의 확장을 지원합니다. 이 단원에서는 AWS Glue API와 함께 Python의 ETL 스트립트를 사용하는 방법을 설명합니다.

- [AWS Glue과 함께 Python을 사용하기 위한 설정](#)
- [Python에서 AWS Glue API 호출](#)
- [AWS Glue와 함께 Python 라이브러리 사용](#)
- [AWS Glue Python 코드 샘플](#)

AWS Glue PySpark 확장 프로그램

AWS Glue PySpark Python 방언에 대한 다음과 같은 확장을 만들었습니다.

- [getResolvedOptions를 사용한 파라미터 액세스](#)
- [PySpark 확장 유형](#)
- [DynamicFrame 클래스](#)
- [DynamicFrameCollection 클래스](#)
- [DynamicFrameWriter 클래스](#)
- [DynamicFrameReader 수업](#)
- [GlueContext 수업](#)

AWS Glue PySpark 변환

AWS Glue PySpark ETL 작업에 사용할 다음과 같은 변환 클래스를 만들었습니다.

- [GlueTransform 베이스 클래스](#)
- [ApplyMapping 클래스](#)
- [DropFields 클래스](#)
- [DropNullField 클래스](#)
- [ErrorsAsDynamicFrame 클래스](#)
- [FillMissingValues 클래스](#)
- [Filter 클래스](#)
- [FindIncrementalMatches 클래스](#)
- [FindMatches 클래스](#)
- [FlatMap 클래스](#)
- [Join 클래스](#)
- [Map 클래스](#)
- [MapToCollection 클래스](#)
- [mergeDynamicFrame](#)
- [Relationalize 클래스](#)
- [RenameField 클래스](#)
- [ResolveChoice 클래스](#)
- [SelectFields 클래스](#)
- [SelectFromCollection 클래스](#)
- [Spigot 클래스](#)
- [SplitFields 클래스](#)
- [SplitRows 클래스](#)
- [Unbox 클래스](#)
- [UnnestFrame 클래스](#)

AWS Glue과 함께 Python을 사용하기 위한 설정

Python을 사용하여 Spark 작업에 대한 ETL 스크립트를 개발합니다. ETL 작업에 대해 지원되는 Python 버전은 작업의 AWS Glue 버전에 따라 다릅니다. AWS Glue 버전에 대한 자세한 내용은 [Glue version job property](#) 섹션을 참조하세요.

AWS Glue와 함께 Python을 사용하기 위해 시스템을 설정합니다.

다음 관계에 따라 Python을 설치하고 AWS Glue API를 호출할 수 있도록 합니다.

1. Python을 아직 설치하지 않았다면 [Python.org 다운로드 페이지](#)에서 다운로드하여 설치합니다.
2. AWS Command Line Interface(AWS CLI)를 [AWS CLI 설명서](#)에 기록하여 설치합니다.

AWS CLI는 Python을 사용하는 데 직접적으로 필요하지 않습니다. 하지만 Python을 설치하고 구성하는 것은 계정 자격으로 AWS를 설치하고 계정 자격이 호용성이 있는지 확인하는 편리한 방법입니다.

3. AWS SDK for Python(Boto 3)이 [Boto3 빠른 시작](#)에 기록되어 있어 설치할 수 있습니다.

AWS Glue에서는 Boto 3 리소스 API를 사용할 수 없습니다. 현재, Boto 3 사용자 API만 사용할 수 있습니다.

Boto 3에 대한 자세한 내용은 [AWS SDK for Python\(Boto3\) 시작하기](#)를 참조하세요.

AWS Glue용 Python 코드 예제와 유틸리티는 GitHub 웹 사이트의 [AWS Glue 샘플 리포지토리](#)에서 찾을 수 있습니다.

Python에서 AWS Glue API 호출

AWS Glue에서는 Boto 3 리소스 API를 사용할 수 없습니다. 현재, Boto 3 사용자 API만 사용할 수 있습니다.

Python에 있는 AWS Glue API 이름

Java 언어로 된 AWS Glue API 이름과 기타 프로그래밍 언어는 보통 카멜케이스(CamelCased)된 단어(단어들 사이의 간격이 없는 단어)입니다. 하지만 Python이 호출하면 일반 이름이 Python식으로 단어들 사이가 밀줄 문자로 분리된 소문자로 바뀝니다. [AWS Glue API](#) 참조 문서에서는 이 Python식 이름이 일반 카멜케이스(CamelCased)된 이름 다음에 괄호 안에 기록됩니다.

그러나 비록 AWS Glue API 이름 자체가 소문자로 변환되었지만 이 파라미터 이름은 대문자로 남겨집니다. 이런 점을 기억해야 하는 이유는 파라미터가 다음 설명처럼 AWS Glue API를 호출할 때 이름에 따라 통과하기 때문입니다.

AWS Glue에서 Python 파라미터 전달 및 액세스

Python은 AWS Glue API를 호출할 경우, 파라미터를 이름에 따라 정확하게 통과하는 것이 최선의 방법입니다. 예:

```
job = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                     Command={'Name': 'glueetl',
                              'ScriptLocation': 's3://my_script_bucket/scripts/
my_etl_script.py'})
```

Python이 [작업 구조](#) 또는 [JobRun 구조](#)의 ETL 스크립트 논제로 지정한 이름/값의 딕셔너리를 생성한다는 점을 이해하면 도움이 됩니다. Boto 3은 JSON 형식으로 REST API 호출을 거쳐 AWS Glue를 통과합니다. 이런 과정은 스크립트에서 논제로 액세스할 경우 논제의 순서에 의존하지 않습니다.

예를 들어, Python Lambda 관리 함수에서 JobRun을 시작하고자 하고 몇 가지 파라미터를 지정하고자 한다고 가정합니다. 코드는 다음과 같을 것입니다.

```
from datetime import datetime, timedelta

client = boto3.client('glue')

def lambda_handler(event, context):
    last_hour_date_time = datetime.now() - timedelta(hours = 1)
    day_partition_value = last_hour_date_time.strftime("%Y-%m-%d")
    hour_partition_value = last_hour_date_time.strftime("%-H")

    response = client.start_job_run(
        JobName = 'my_test_Job',
        Arguments = {
            '--day_partition_key': 'partition_0',
            '--hour_partition_key': 'partition_1',
            '--day_partition_value': day_partition_value,
            '--hour_partition_value': hour_partition_value } )
```

안전하게 ETL에서 이 파라미터로 액세스하기 위해서는 AWS Glue의 `getResolvedOptions` 함수를 사용하여 이름에 따라 지정하고 딕셔너리 결과로부터 파라미터로 액세스합니다.

```
import sys
```



```

from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv,
                          ['JOB_NAME',
                          'day_partition_key',
                          'hour_partition_key',
                          'day_partition_value',
                          'hour_partition_value'])

print "The day partition key is: ", args['day_partition_key']
print "and the day partition value is: ", args['day_partition_value']

```

중첩된 JSON 문자열인 인수를 전달하려는 경우 파라미터 값이 AWS Glue ETL 작업에 전달될 때 해당 값을 유지하려면 작업 실행을 시작하기 전에 파라미터 문자열을 인코딩한 다음, 작업 스크립트를 참조하기 전에 파라미터 문자열을 디코딩해야 합니다. 예를 들어 다음 인수 문자열을 고려하세요.

```

glue_client.start_job_run(JobName = "gluejobname", Arguments={
"--my_curly_braces_string": '{"a": {"b": {"c": [{"d": {"e": 42}]}}}}'
})

```

이 파라미터를 올바르게 전달하려면 인수를 Base64로 인코딩한 문자열로 인코딩해야 합니다.

```

import base64
...
sample_string='{"a": {"b": {"c": [{"d": {"e": 42}]}}}}'
sample_string_bytes = sample_string.encode("ascii")

base64_bytes = base64.b64encode(sample_string_bytes)
base64_string = base64_bytes.decode("ascii")
...
glue_client.start_job_run(JobName = "gluejobname", Arguments={
"--my_curly_braces_string": base64_bytes})
...
sample_string_bytes = base64.b64decode(base64_bytes)
sample_string = sample_string_bytes.decode("ascii")
print(f"Decoded string: {sample_string}")
...

```

예: 작업 생성 및 실행

다음 예제는 Python을 사용하여 AWS Glue API를 호출하여 ETL 작업을 실행할 수 있는 방법을 보여줍니다.

예: 작업을 생성하고 실행하기 위해서

1. AWS Glue 클라이언트의 인스턴스를 만듭니다.

```
import boto3
glue = boto3.client(service_name='glue', region_name='us-east-1',
                    endpoint_url='https://glue.us-east-1.amazonaws.com')
```

2. 작업 만들기 다음 코드에서 보여지듯이 glueetl를 ETL 명령어로 사용해야 합니다.

```
myJob = glue.create_job(Name='sample', Role='Glue_DefaultRole',
                        Command={'Name': 'glueetl',
                                'ScriptLocation': 's3://my_script_bucket/
scripts/my_etl_script.py'})
```

3. 이전 단계에서 만든 작업을 새롭게 시작합니다.

```
myNewJobRun = glue.start_job_run(JobName=myJob['Name'])
```

4. 작업 상태 얻기.

```
status = glue.get_job_run(JobName=myJob['Name'], RunId=myNewJobRun['JobRunId'])
```

5. 작업 실행의 현재 상태를 프린트합니다.

```
print(status['JobRun']['JobRunState'])
```

AWS Glue와 함께 Python 라이브러리 사용

AWS Glue를 사용하면 AWS Glue ETL에서 사용할 추가 Python 모듈과 라이브러리를 설치할 수 있습니다.

주제

- [AWS Glue 2.0+에서 pip를 사용하여 추가 Python 모듈 설치](#)
- [PySpark 기본 기능을 갖춘 Python 파일 포함](#)
- [시각적 변환을 사용하는 프로그래밍 스크립트](#)
- [AWS Glue에서 이미 제공되는 Python 모듈](#)
- [라이브러리 압축하여 포함](#)

- [AWS Glue 스튜디오 노트북에 Python 라이브러리 불러오기](#)
- [개발 엔드포인트에서 Python 라이브러리 로딩](#)
- [작업에서 Python 라이브러리 사용 또는 JobRun](#)

AWS Glue 2.0+에서 pip를 사용하여 추가 Python 모듈 설치

AWS Glue는 Python 패키지 설치 프로그램(pip3)을 사용하여 AWS Glue ETL에서 사용할 추가 모듈을 설치합니다. `--additional-python-modules` 파라미터를 쉼표로 구분된 Python 모듈 목록과 함께 사용하여 새 모듈을 추가하거나 기존 모듈의 버전을 변경할 수 있습니다. Amazon S3에 배포를 업로드 하여 라이브러리의 사용자 지정 배포를 설치한 다음, 모듈 목록에 Amazon S3 객체의 경로를 포함시킬 수 있습니다.

`--python-modules-installer-option` 파라미터를 사용하여 pip3에 추가 옵션을 전달할 수 있습니다. 예를 들어, `--upgrade`를 전달하여 `--additional-python-modules`로 지정된 패키지를 업그레이드할 수 있습니다. 더 많은 예제는 [Glue AWS 2.0을 사용하여 Spark ETL 워크로드를 위한 휠에서 Python 모듈 빌드를 참조하십시오](#).

Python 종속성이 컴파일된 네이티브 코드에 전이적으로 의존하는 경우 다음과 같은 제한에 직면할 수 있습니다. AWS Glue는 작업 환경에서의 네이티브 코드 컴파일을 지원하지 않습니다. 하지만 AWS Glue 작업은 Amazon Linux 2 환경에서 실행됩니다. Wheel 배포판을 통해 컴파일된 형식으로 네이티브 종속성을 제공할 수 있습니다.

예를 들어 새 `scikit-learn` 모듈을 업데이트하거나 추가하려면 키값 `--additional-python-modules`, `"scikit-learn==0.21.3"`을 사용합니다.

또한 `--additional-python-modules` 옵션 내에서 Python 휠 모듈에 대한 Amazon S3 경로를 지정할 수 있습니다. 예:

```
--additional-python-modules s3://aws-glue-native-spark/tests/j4.2/ephem-3.7.7.1-cp37-cp37m-linux_x86_64.whl,s3://aws-glue-native-spark/tests/j4.2/fbprophet-0.6-py3-none-any.whl,scikit-learn==0.21.3
```

AWS Glue 콘솔의 Job parameters 필드에 지정하거나 AWS SDK의 작업 인수를 변경하여 지정합니다. `--additional-python-modules` 작업 파라미터 설정에 대한 자세한 내용을 알아보려면 [the section called “작업 파라미터”](#) 섹션을 참조하세요.

PySpark 기본 기능을 갖춘 Python 파일 포함

AWS Glue AWS Glue ETL 작업에 Python 파일을 포함하는 PySpark 데 사용합니다. 사용 가능한 경우 종속성을 관리하기 위해 `--additional-python-modules`를 사용하고 싶을 것입니다. `--extra-`

py-files 작업 파라미터를 사용하여 Python 파일을 포함할 수 있습니다. 종속성은 Amazon S3에서 호스팅되어야 하며, 인수 값은 공백 없이 쉼표로 구분된 Amazon S3 경로 목록이어야 합니다. 이 기능은 Spark에서 사용하는 Python 종속성 관리처럼 작동합니다. Spark의 Python 종속성 관리에 대한 자세한 내용은 Apache Spark [PySpark 설명서의 기본 기능 사용](#) 페이지를 참조하십시오. --extra-py-files 추가 코드가 패키징되지 않은 경우 또는 종속성 관리를 위해 기존 틀체인을 사용하여 Spark 프로그램을 마이그레이션하는 경우에 유용합니다. 종속성 도구를 유지 관리하려면 제출하기 전에 종속성을 번들로 묶어야 합니다.

시각적 변환을 사용하는 프로그래밍 스크립트

AWS Glue Studio 시각적 인터페이스를 사용하여 AWS Glue 작업을 생성하면 관리형 데이터 변환 노드와 사용자 지정 시각적 변환으로 데이터를 변환할 수 있습니다. 관리형 데이터 변환 노드에 대한 자세한 내용은 [the section called “AWS Glue 관리형 데이터 변환 노드 편집”](#) 섹션을 참조하세요. 사용자 지정 시각적 변환에 대한 자세한 내용은 [the section called “사용자 지정 시각적 변환”](#) 섹션을 참조하세요. 시각적 변환을 사용하는 스크립트는 작업 언어가 Python 사용으로 설정된 경우에만 생성할 수 있습니다.

시각적 변환을 사용하여 AWS Glue 작업을 생성할 때 AWS Glue Studio는 작업 구성의 --extra-py-files 매개변수를 사용하여 런타임 환경에 이러한 변환을 포함합니다. 작업 파라미터에 대한 자세한 내용을 알아보려면 [the section called “작업 파라미터”](#) 섹션을 참조하세요. 생성된 스크립트 또는 런타임 환경을 변경할 때 스크립트가 성공적으로 실행되도록 하려면 이 작업 구성을 유지해야 합니다.

AWS Glue에서 이미 제공되는 Python 모듈

이러한 제공된 모듈의 버전을 변경하려면 새 버전에 --additional-python-modules 작업 파라미터를 제공합니다.

AWS Glue version 2.0

AWS Glue 버전 2.0은 기본적으로 다음과 같은 Python 모듈을 포함합니다.

- avro-python3==1.10.0
- awscli==1.27.60
- boto3==1.12.4
- botocore==1.15.4
- certifi==2019.11.28
- chardet==3.0.4
- click==8.1.3

- colorama==0.4.4
- cycler==0.10.0
- Cython==0.29.15
- docutils==0.15.2
- enum34==1.1.9
- fsspec==0.6.2
- idna==2.9
- importlib-metadata==6.0.0
- jmespath==0.9.4
- joblib==0.14.1
- kiwisolver==1.1.0
- matplotlib==3.1.3
- mpmath==1.1.0
- nltk==3.5
- numpy==1.18.1
- pandas==1.0.1
- patsy==0.5.1
- pmdarima==1.5.3
- ptvsd==4.3.2
- pyarrow==0.16.0
- pyasn1==0.4.8
- pydevd==1.9.0
- pyhocon==0.3.54
- PyMySQL==0.9.3
- pyparsing==2.4.6
- python-dateutil==2.8.1
- pytz==2019.3
- PyYAML==5.3.1
- regex==2022.10.31
- requests==2.23.0

- rsa==4.7.2
- s3fs==0.4.0
- s3transfer==0.3.3
- scikit-learn==0.22.1
- scipy==1.4.1
- setuptools==45.2.0
- six==1.14.0
- Spark==1.0
- statsmodels==0.11.1
- subprocess32==3.5.4
- sympy==1.5.1
- tbats==1.0.9
- tqdm==4.64.1
- typing-extensions==4.4.0
- urllib3==1.25.8
- wheel==0.35.1
- zipp==3.12.0

AWS Glue 버전 3.0

AWS Glue 버전 3.0은 기본적으로 다음과 같은 Python 모듈을 포함합니다.

- aiobotocore==1.4.2
- aiohttp==3.8.3
- aioitertools==0.11.0
- aiosignal==1.3.1
- async-timeout==4.0.2
- asyncctest==0.13.0
- attrs==22.2.0
- avro-python3==1.10.2
- boto3==1.18.50

- boto3==1.21.50
- certifi==2021.5.30
- chardet==3.0.4
- charset-normalizer==2.1.1
- click==8.1.3
- cyclers==0.10.0
- Cython==0.29.4
- docutils==0.17.1
- enum34==1.1.10
- frozenlist==1.3.3
- fsspec==2021.8.1
- idna==2.10
- importlib-metadata==6.0.0
- jmespath==0.10.0
- joblib==1.0.1
- kiwisolver==1.3.2
- matplotlib==3.4.3
- mpmath==1.2.1
- multidict==6.0.4
- nltk==3.6.3
- numpy==1.19.5
- packaging==23.0
- pandas==1.3.2
- patsy==0.5.1
- Pillow==9.4.0
- pip==23.0
- pmdarima==1.8.2
- ptvsd==4.3.2
- pyarrow==5.0.0
- pydevd==2.5.0

- pyhocon==0.3.58
- PyMySQL==1.0.2
- pyparsing==2.4.7
- python-dateutil==2.8.2
- pytz==2021.1
- PyYAML==5.4.1
- regex==2022.10.31
- requests==2.23.0
- s3fs==2021.8.1
- s3transfer==0.5.0
- scikit-learn==0.24.2
- scipy==1.7.1
- six==1.16.0
- Spark==1.0
- statsmodels==0.12.2
- subprocess32==3.5.4
- sympy==1.8
- tbats==1.1.0
- threadpoolctl==3.1.0
- tqdm==4.64.1
- typing_extensions==4.4.0
- urllib3==1.25.11
- wheel==0.37.0
- wrapt==1.14.1
- yarl==1.8.2
- zipp==3.12.0

AWS Glue 버전 4.0

AWS Glue 버전 4.0은 기본적으로 다음과 같은 Python 모듈을 포함합니다.

- aiobotocore==2.4.1
- aiohttp==3.8.3
- aioitertools==0.11.0
- aiosignal==1.3.1
- async-timeout==4.0.2
- asynctest==0.13.0
- attrs==22.2.0
- avro-python3==1.10.2
- boto3==1.24.70
- botocore==1.27.59
- certifi==2021.5.30
- chardet==3.0.4
- charset-normalizer==2.1.1
- click==8.1.3
- cycler==0.10.0
- Cython==0.29.32
- docutils==0.17.1
- enum34==1.1.10
- frozenlist==1.3.3
- fsspec==2021.8.1
- idna==2.10
- importlib-metadata==5.0.0
- jmespath==0.10.0
- joblib==1.0.1
- kaleido==0.2.1
- kiwisolver==1.4.4
- matplotlib==3.4.3
- mpmath==1.2.1
- multidict==6.0.4

- nltk==3.7
- numpy==1.23.5
- packaging==23.0
- pandas==1.5.1
- patsy==0.5.1
- Pillow==9.4.0
- pip==23.0.1
- plotly==5.16.0
- pmdarima==2.0.1
- ptvsd==4.3.2
- pyarrow==10.0.0
- pydevd==2.5.0
- pyhocon==0.3.58
- PyMySQL==1.0.2
- pyparsing==2.4.7
- python-dateutil==2.8.2
- pytz==2021.1
- PyYAML==6.0.1
- regex==2022.10.31
- requests==2.23.0
- s3fs==2022.11.0
- s3transfer==0.6.0
- scikit-learn==1.1.3
- scipy==1.9.3
- setuptools==49.1.3
- six==1.16.0
- statsmodels==0.13.5
- subprocess32==3.5.4
- sympy==1.8
- tbats==1.1.0

- threadpoolctl==3.1.0
- tqdm==4.64.1
- typing_extensions==4.4.0
- urllib3==1.25.11
- wheel==0.37.0
- wrapt==1.14.1
- yarl==1.8.2
- zipp==3.10.0

라이브러리 압축하여 포함

라이브러리가 하나의 .py 파일로 포함되지 않는 한 .zip 아카이브로 패키징되어야 합니다. 패키지 디렉터리는 아카이브의 루트에 있어야 하고 패키지용 `__init__.py` 파일을 포함해야 합니다. 그러면 Python은 정상적인 방법으로 패키지를 가져올 수 있습니다.

라이브러리가 하나의 .py 파일에 하나의 Python 모듈만 구성할 수 있으면 모듈을 .zip에 넣을 필요가 없습니다.

AWS Glue 스튜디오 노트북에 Python 라이브러리 불러오기

AWS Glue Studio 노트북에서 Python 라이브러리를 지정하려면 [추가 Python 모듈 설치](#)를 참조하십시오.

개발 엔드포인트에서 Python 라이브러리 로딩

다른 ETL 스크립트를 설치하기 위해 다른 라이브러리를 사용할 경우, 각 세트에 따라 개별 개발 엔드포인트를 설치하거나 개발 엔드포인트가 스크립트를 스위치할 때마다 로딩되는 라이브러리 .zip 파일(들)을 덮어쓰기할 수 있습니다.

콘솔을 사용하여 개발 엔드포인트를 생성할 때 개발 엔드포인트를 위한 하나 이상의 .zip 파일을 지정할 수 있습니다. 이름과 IAM 역할을 할당한 후 [스크립트 라이브러리 및 작업 파라미터(선택 사항)] (Script Libraries and job parameters (optional))를 선택하고 [Python 라이브러리 경로(Python library path)] 상자에 라이브러리 .zip 파일에 대한 전체 Amazon S3 경로를 입력합니다. 예:

```
s3://bucket/prefix/site-packages.zip
```

원한다면, 다음과 같이 빈 칸없이 쉼표로만 나뉜 파일까지의 전체 경로를 지정할 수 있습니다.

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

이런 .zip 파일을 나중에 업데이트하면 콘솔을 사용하여 개발 엔드포인트로 이러 파일을 다시 가져올 수 있습니다. 해당 개발자 엔드포인트로 이동하여 옆에 있는 확인란을 선택한 다음 [작업(Action)] 메뉴에서 [ETL 라이브러리 업데이트(Update ETL libraries)]를 선택합니다.

비슷한 방식으로 AWS Glue API를 사용하여 라이브러리 파일을 지정할 수 있습니다.

[CreateDevEndpoint 작업\(Python: create_dev_endpoint\)](#)를 호출하여 개발 엔드포인트를 생성하고자 할 때 다음과 같은 호출에 따라 ExtraPythonLibsS3Path 파라미터의 라이브러리까지 하나 이상의 전체 경로를 지정할 수 있습니다.

```
dep = glue.create_dev_endpoint(
    EndpointName="testDevEndpoint",
    RoleArn="arn:aws:iam::123456789012",
    SecurityGroupIds="sg-7f5ad1ff",
    SubnetId="subnet-c12fdb4",
    PublicKey="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCTp04H/y...",
    NumberOfNodes=3,
    ExtraPythonLibsS3Path="s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/
lib_X.zip")
```

개발 엔드포인트를 업데이트할 때 [DevEndpointCustomLibraries](#) 객체를 사용하고 [UpdateDevEndpoint\(update_dev_endpoint\)](#)를 호출할 때 UpdateEtlLibraries 파라미터를 True로 설정하여 로드하는 라이브러리를 업데이트할 수도 있습니다.

작업에서 Python 라이브러리 사용 또는 JobRun

콘솔에서 새로운 작업을 생성하고자 할 경우, [스크립트 라이브러리 및 작업 파라미터(선택 사항) (Script Libraries and job parameters (optional))]를 선택하고 개발 엔드포인트를 생성할 때와 동일한 방식으로 전체 Amazon S3 라이브러리 경로를 입력하여 라이브러리 .zip 파일을 하나 이상 지정할 수 있습니다.

```
s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip
```

[CreateJob \(생성_작업\)](#)를 호출한다면 다음과 같은 --extra-py-files 기본 파라미터를 사용하여 기본 라이브러리까지 하나 이상의 전체 경로를 지정할 수 있습니다.

```
job = glue.create_job(Name='sampleJob',
```

```

        Role='Glue_DefaultRole',
        Command={'Name': 'glueetl',
                 'ScriptLocation': 's3://my_script_bucket/scripts/
my_etl_script.py'},
        DefaultArguments={'--extra-py-files': 's3://bucket/prefix/
lib_A.zip,s3://bucket_B/prefix/lib_X.zip'})

```

그런 다음 JobRun a를 시작할 때 기본 라이브러리 설정을 다른 것으로 재정의할 수 있습니다.

```

runId = glue.start_job_run(JobName='sampleJob',
                           Arguments={'--extra-py-files': 's3://bucket/prefix/
lib_B.zip'})

```

AWS Glue Python 코드 샘플

- [코드 예: 데이터 조인 및 관계화](#)
- [코드 예: ResolveChoice, Lambda, 및 ApplyMapping을 사용한 데이터 준비](#)

코드 예: 데이터 조인 및 관계화

이 예제에서는 <http://everypolitician.org/>에서 Amazon Simple Storage Service(Amazon S3)(s3://awsglue-datasets/examples/us-legislators/all)의 sample-dataset 버킷으로 다운로드한 데이터 집합을 사용합니다. 데이터 세트에는 미국 입법 기관과 미 상원 및 하원에서 확보한 의석에 대한 JSON 형식의 데이터가 포함되어 있으며, 이 자습서의 용도에 맞게 퍼블릭 Amazon S3 버킷에서 사용할 수 있도록 데이터 세트가 약간 수정되었습니다.

이 예제의 소스 코드는 GitHub 웹 사이트의 [AWS Glue 샘플 리포지토리](#)에 있는 join_and_relationalize.py 파일에서 찾을 수 있습니다.

이 데이터를 사용하여 이 자습서에서는 다음을 수행하는 방법을 보여줍니다.

- AWS Glue 크롤러를 사용하여 퍼블릭 Amazon S3 버킷에 저장된 객체를 분류하고 AWS Glue Data Catalog에 객체의 스키마를 저장합니다.
- 크롤의 결과인 테이블 메타데이터 및 스키마를 검토합니다.
- Data Catalog의 메타데이터를 사용하여 Python 추출, 변환, 로드(ETL) 스크립트를 작성하고 다음을 실행합니다.
 - 하나의 데이터 테이블로 다른 원본 파일의 데이터를 모읍니다 (즉, 데이터를 비정규화합니다).
 - 제정자 유형에 따라 모아진 테이블을 개별 테이블로 필터링합니다.

- 다음 분석을 위해 결과 데이터를 Apache Parquet 파일을 작성합니다.

AWS에서 실행하는 중에 Python 또는 PySpark 스크립트를 디버깅하는 기본적인 방법은 [AWS Glue Studio에서 노트북](#)을 사용하는 것입니다.

1단계: Amazon S3 버킷에서 데이터 크롤

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. [크롤러 구성](#)의 단계에 따라 AWS Glue Data Catalog에서 legislators라는 데이터베이스로 s3://awsglue-datasets/examples/us-legislators/all 데이터 집합을 크롤링할 수 있는 새 크롤러를 만듭니다. 이 예제 데이터는 이 퍼블릭 Amazon S3 버킷에 이미 존재합니다.
3. 새로운 크롤러를 실행한 다음 legislators 데이터베이스를 확인합니다.

크롤러는 다음 메타데이터 테이블을 생성합니다.

- persons_json
- memberships_json
- organizations_json
- events_json
- areas_json
- countries_r_json

제정자와 제정자 기록을 포함한 테이블 반정규화된 컬렉션입니다.

2단계: 개발 엔드포인트 노트북에 표준 문안 스크립트 추가

다음 표준 문안 스크립트를 개발 엔드포인트 노트북에 복사하고 필요한 AWS Glue 라이브러리로 들어와 단일 GlueContext를 설정합니다.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
```

```
glueContext = GlueContext(SparkContext.getOrCreate())
```

3단계: Data Catalog의 데이터에서 스키마 검토

다음으로 AWS Glue Data Catalog에서 검사 DynamicFrame을 쉽게 생성하고 데이터의 스키마를 검사할 수 있습니다. 예를 들어, persons_json 테이블의 스키마를 보고 노트북에 다음을 추가합니다.

```
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators",
    table_name="persons_json")
print "Count: ", persons.count()
persons.printSchema()
```

프린트 호출에 따른 출력값입니다.

```
Count: 1961
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
```

```
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
```

테이블에 있는 각자는 미국 의회 기관의 멤버입니다.

memberships_json 테이블의 스키마를 보고 다음을 입력합니다.

```
memberships = glueContext.create_dynamic_frame.from_catalog(
    database="legislators",
    table_name="memberships_json")
print "Count: ", memberships.count()
memberships.printSchema()
```

출력값은 다음과 같습니다.

```
Count: 10439
root
|-- area_id: string
|-- on_behalf_of_id: string
|-- organization_id: string
|-- role: string
|-- person_id: string
|-- legislative_period_id: string
|-- start_date: string
|-- end_date: string
```

organizations는 정당이고 미국의 상원과 하원인 의회 양원입니다. organizations_json 테이블의 스키마를 보고 다음을 입력합니다.

```
orgs = glueContext.create_dynamic_frame.from_catalog(
    database="legislators",
    table_name="organizations_json")
print "Count: ", orgs.count()
orgs.printSchema()
```


출력값은 다음과 같습니다.

```
Count: 13
root
|-- classification: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- id: string
|-- name: string
|-- seats: int
|-- type: string
```

4단계: 데이터 필터링

원하는 필드만 남기고 id를 org_id로 이름을 바꿉니다. 데이터셋은 전체를 볼 수 있을만큼 작습니다.

toDF()는 DynamicFrame를 Apache Spark DataFrame로 변환하기 때문에 Apache Spark SQL에 이미 존재한 변환을 적용할 수 있습니다.

```
orgs = orgs.drop_fields(['other_names',
                        'identifiers']).rename_field(
    'id', 'org_id').rename_field(
    'name', 'org_name')
orgs.toDF().show()
```

다음은 출력값을 보여줍니다.

```

+-----+-----+-----+-----+-----+
+-----+-----+
|classification|          org_id|          org_name|          links|seats|
|      type|          image|
+-----+-----+-----+-----+-----+
+-----+-----+
|      party|      party/al|          AL|          null| null|
|      null|          null|
|      party|      party/democrat|          Democrat|[[website,http://...| null|
|      null|https://upload.wi...|
|      party|party/democrat-li...|          Democrat-Liberal|[[website,http://...| null|
|      null|          null|
| legislature|d56acebe-8fdc-47b...|House of Represen...|          null| 435|
lower house|          null|
|      party|      party/independent|          Independent|          null| null|
|      null|          null|
|      party|party/new_progres...|          New Progressive|[[website,http://...| null|
|      null|https://upload.wi...|
|      party|party/popular_dem...|          Popular Democrat|[[website,http://...| null|
|      null|          null|
|      party|      party/republican|          Republican|[[website,http://...| null|
|      null|https://upload.wi...|
|      party|party/republican-...|Republican-Conser...|[[website,http://...| null|
|      null|          null|
|      party|      party/democrat|          Democrat|[[website,http://...| null|
|      null|https://upload.wi...|
|      party|      party/independent|          Independent|          null| null|
|      null|          null|
|      party|      party/republican|          Republican|[[website,http://...| null|
|      null|https://upload.wi...|
| legislature|8fa6c3d2-71dc-478...|          Senate|          null| 100|
upper house|          null|
+-----+-----+-----+-----+-----+
+-----+-----+

```

memberships에 나타나는 organizations를 보려면 다음을 입력합니다.

```
memberships.select_fields(['organization_id']).toDF().distinct().show()
```

다음은 출력값을 보여줍니다.

```
+-----+
| organization_id|
+-----+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+-----+
```

5단계: 데이터 조인

이제 AWS Glue를 사용하여 이런 관계형 테이블을 조인하고 제정자 memberships의 전체 기록 테이블과 제정자에 대응하는 organizations을 생성합니다.

1. 우선, persons와 memberships를 id와 person_id로 조인합니다.
2. 다음으로 org_id 및 organization_id에서 orgs와 결과를 조인합니다.
3. 그 후, 중복 필드, person_id 및 org_id을 드롭합니다.

이 모든 작업을 하나 줄의 (확장된) 코드로 실행할 수 있습니다.

```
l_history = Join.apply(orgs,
                      Join.apply(persons, memberships, 'id', 'person_id'),
                      'org_id', 'organization_id').drop_fields(['person_id',
                        'org_id'])
print "Count: ", l_history.count()
l_history.printSchema()
```

출력값은 다음과 같습니다.

```
Count: 10439
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- type: string
|-- sort_name: string
```

```

|-- area_id: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- death_date: string
|-- legislative_period_id: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- image: string
|-- given_name: string
|-- family_name: string
|-- id: string
|-- start_date: string
|-- end_date: string

```

이제, 분석을 위한 최종 테이블이 있습니다. AWS Glue, Amazon Athena 또는 Amazon Redshift Spectrum에서 SQL을 실행할 수 있는 압축되고 효율적인 분석 형식(즉, Parquet)으로 작성할 수 있습니다.

다음 호출은 여러 파일의 테이블을 작성하여 나중에 분석을 할 때 빠른 병렬 판독을 지원합니다.

```

glueContext.write_dynamic_frame.from_options(frame = l_history,
      connection_type = "s3",
      connection_options = {"path": "s3://glue-sample-target/output-dir/
legislator_history"},

```

```
format = "parquet")
```

하나의 파일에 기록 데이터 모두를 넣으면 데이터 프레임으로 변환하고 다시 분할하며 작성할 수 있습니다.

```
s_history = l_history.toDF().repartition(1)
s_history.write.parquet('s3://glue-sample-target/output-dir/legislator_single')
```

상원과 하원으로 나누고자 한다면

```
l_history.toDF().write.parquet('s3://glue-sample-target/output-dir/legislator_part',
                               partitionBy=['org_name'])
```

6단계: 관계형 데이터베이스를 위한 데이터 변환

AWS Glue를 사용하면 반정형 데이터가 포함된, Amazon Redshift와 같은 관계형 데이터베이스에도 데이터를 쉽게 쓸 수 있습니다. 프레임의 객체가 아무리 복잡하더라도 DynamicFrames를 평면화하는 변환 `relationalize`를 제공합니다.

이 예제에 `l_history` DynamicFrame를 사용하여 루트 테이블의 이름(`hist_root`)과 임시 작업 경로를 `relationalize`로 넘깁니다. 이렇게 하면 DynamicFrameCollection가 반환됩니다. 컬렉션의 DynamicFrames의 이름을 열거할 수 있습니다.

```
dfc = l_history.relationalize("hist_root", "s3://glue-sample-target/temp-dir/")
dfc.keys()
```

다음은 이 keys 호출의 출력입니다.

```
[u'hist_root', u'hist_root_contact_details', u'hist_root_links',
 u'hist_root_other_names', u'hist_root_images', u'hist_root_identifiers']
```

Relationalize는 DynamicFrame의 각 객체의 기록을 포함한 루트 테이블과 배열의 보조 테이블을 포함한 6개의 새로운 테이블로 기록 테이블을 나눕니다. 관계형 데이터베이스에서 처리한 배열은 보통 차선책일 수 있습니다. 특히 배열이 커지면 그렇습니다. 배열을 다른 테이블로 나누면 쿼리 과정이 빨라집니다.

이제, `contact_details`를 보고 나누는 과정에 대해 알아보니다.

```
l_history.select_fields('contact_details').printSchema()
dfc.select('hist_root_contact_details').toDF().where("id = 10 or id =
75").orderBy(['id', 'index']).show()
```

다음은 이 `show` 호출의 출력입니다.

```
root
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
+-----+-----+-----+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+-----+-----+-----+-----+-----+-----+
| 10|  0|          fax| |
| 10|  1|          |      202-225-1314|
| 10|  2|        phone|
| 10|  3|          |      202-225-3772|
| 10|  4|        twitter|
| 10|  5|          |      MikeRossUpdates|
| 75|  0|          fax|
| 75|  1|          |      202-225-7856|
| 75|  2|        phone|
| 75|  3|          |      202-225-2711|
| 75|  4|        twitter|
| 75|  5|          |      SenCapito|
+-----+-----+-----+-----+-----+-----+
```

`contact_details` 필드는 기존 `DynamicFrame`의 구조 배열이었습니다. 이 배열에 있는 각 요소가 `index`에 따라 보조 테이블의 별도 행입니다. 여기서 `id`는 `contact_details` 키가 있는 `hist_root` 테이블의 외래 키입니다.

```
dfc.select('hist_root').toDF().where(
    "contact_details = 10 or contact_details = 75").select(
    ['id', 'given_name', 'family_name', 'contact_details']).show()
```

출력값은 다음과 같습니다.

```

+-----+-----+-----+-----+
|           id|given_name|family_name|contact_details|
+-----+-----+-----+-----+
|f4fc30ee-7b42-432...|    Mike|    Ross|    10|
|e3c60f34-7d1b-4c0...|  Shelley|    Capito|    75|
+-----+-----+-----+-----+

```

이 명령어에는 `toDF()`와 `where` 순으로 선택하여 보고자 하는 열을 필터링합니다.

따라서 보조 테이블과 함께 `hist_root`을 조인하여 다음과 같은 작업을 수행할 수 있습니다.

- 배열 지원 없이 데이터베이스로 데이터를 로드합니다.
- SQL을 사용하여 배열에서 개별 항목을 쿼리합니다.

AWS Glue 연결을 사용하여 Amazon Redshift 자격 증명을 안전하게 저장하고 액세스합니다. 자체 연결을 만드는 방법에 대한 자세한 내용은 [데이터에 연결](#)을 참조하십시오.

이제 `DynamicFrames`를 차례로 순환하여 데이터를 연결에 쓸 준비가 되었습니다.

```

for df_name in dfc.keys():
    m_df = dfc.select(df_name)
    print "Writing to table: ", df_name
    glueContext.write_dynamic_frame.from_jdbc_conf(frame = m_df, connection settings here)

```

연결 설정은 관계형 데이터베이스 유형에 따라 달라집니다.

- Amazon Redshift에 쓰는 방법에 대한 지침은 [the section called “Redshift 연결”](#) 섹션을 참조하세요.
- 다른 데이터베이스의 경우 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#) 섹션을 참조하세요.

결론

전체적으로 AWS Glue는 유연성이 뛰어납니다. 보통 며칠을 걸러 작성해야 가능한 이 작업을 단 몇 줄의 코드로 얻을 수 있습니다. GitHub의 [AWS Glue 샘플](#)에 있는 Python 파일 `join_and_relationalize.py`에서 전체 소스-대상 ETL 스크립트를 찾을 수 있습니다.

코드 예: ResolveChoice, Lambda, 및 ApplyMapping을 사용한 데이터 준비

이 예에서 사용된 데이터 집합은 두 [Data.CMS.gov](https://data.cms.gov) 데이터 집합("상위 100개 진단 관련 그룹에 대한 입원 환자 예상 지불 시스템 제공자 요약 - FY2011" 및 "FY2011 입원 환자 비용 데이터")에서 다운로드한 Medicare Provider 지불 데이터로 구성됩니다. 데이터 다운로드 후 데이터 집합을 수정하여 파일 끝에 몇 가지 잘못된 기록을 소개합니다. 이 수정된 파일은 `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv`의 퍼블릭 Amazon S3 버킷에 있습니다.

이 예제에 대한 소스 코드는 [AWS Glue 예제](#) GitHub 리포지토리의 `data_cleaning_and_lambda.py` 파일에서 찾을 수 있습니다.

AWS에서 실행하는 중에 Python 또는 PySpark 스크립트를 디버깅하는 기본적인 방법은 [AWS Glue Studio에서 노트북](#)을 사용하는 것입니다.

1단계: Amazon S3 버킷에서 데이터 크롤

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. [크롤러 구성](#)의 절차를 밟고 `s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv` 파일을 크롤할 수 있는 새로운 크롤러를 생성하며 그 결과 생성된 메타데이터를 AWS Glue Data Catalog의 `payments`라는 데이터베이스에 둡니다.
3. 새로운 크롤러를 실행한 다음 `payments` 데이터베이스를 확인합니다. 파일의 시작 부분을 읽어 형식과 구분 기호를 확인한 후 크롤러가 데이터베이스에 `medicare`라는 이름의 메타데이터 테이블을 생성했을 것입니다.

새로운 `medicare`의 스키마는 다음과 같습니다.

Column name	Data type
drg definition	string
provider id	bigint
provider name	string
provider street address	string
provider city	string
provider state	string
provider zip code	bigint
hospital referral region description	string
total discharges	bigint
average covered charges	string
average total payments	string

average medicare payments

string

2단계: 개발 엔드포인트 노트북에 표준 문안 스크립트 추가

다음 표준 문안 스크립트를 개발 엔드포인트 노트북에 복사하고 필요한 AWS Glue 라이브러리로 들어와 단일 GlueContext를 설정합니다.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

glueContext = GlueContext(SparkContext.getOrCreate())
```

3단계: 다른 스키마 파싱과 비교

그 다음, Apache Spark DataFrame이 인지한 스키마가 AWS Glue 크롤러가 기록한 것과 동일한지 알아봅니다. 이 코드를 실행합니다.

```
medicare = spark.read.format(
    "com.databricks.spark.csv").option(
    "header", "true").option(
    "inferSchema", "true").load(
    's3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv')
medicare.printSchema()
```

printSchema 호출에 따른 출력값입니다.

```
root
|-- DRG Definition: string (nullable = true)
|-- Provider Id: string (nullable = true)
|-- Provider Name: string (nullable = true)
|-- Provider Street Address: string (nullable = true)
|-- Provider City: string (nullable = true)
|-- Provider State: string (nullable = true)
|-- Provider Zip Code: integer (nullable = true)
|-- Hospital Referral Region Description: string (nullable = true)
```

```
|-- Total Discharges : integer (nullable = true)
|-- Average Covered Charges : string (nullable = true)
|-- Average Total Payments : string (nullable = true)
|-- Average Medicare Payments: string (nullable = true)
```

다음, AWS Glue DynamicFrame가 생성한 스키마를 알아봅니다.

```
medicare_dynamicframe = glueContext.create_dynamic_frame.from_catalog(
    database = "payments",
    table_name = "medicare")
medicare_dynamicframe.printSchema()
```

printSchema의 출력값은 다음과 같습니다.

```
root
 |-- drg definition: string
 |-- provider id: choice
 |   |-- long
 |   |-- string
 |-- provider name: string
 |-- provider street address: string
 |-- provider city: string
 |-- provider state: string
 |-- provider zip code: long
 |-- hospital referral region description: string
 |-- total discharges: long
 |-- average covered charges: string
 |-- average total payments: string
 |-- average medicare payments: string
```

DynamicFrame는 provider id가 long 혹은 string 유형일 수 있는 스키마를 생성합니다. DataFrame 스키마는 Provider Id를 string 유형으로 목록에 기록하고 Data Catalog는 provider id를 bigint 유형으로 목록에 기록합니다.

무엇이 정답입니까? 파일의 끝에는 이 열에 있는 string 값과 함께 (160,000 기록 중) 두 기록이 있습니다. 이 두 기록은 문제를 보여주기 위한 잘못된 기록입니다.

이런 문제를 설명하기 위해서 AWS Glue DynamicFrame는 선택 유형의 개념을 도입합니다. 이 경우, DynamicFrame는 이 열에 나타나는 long 및 string 모두를 보여줍니다. AWS Glue 크롤러

는 string 값을 누락했는데 그 이유는 데이터 2MB 접두사만 고려했기 때문입니다. Apache Spark DataFrame는 전체 데이터 세트를 고려했지만 string이라는 열에 가장 일반적인 유형을 지정하도록 강요되었습니다. 사실, Spark는 익숙하지 않은 복잡한 유형이나 변수가 있으면 가장 일반적인 케이스를 적용합니다.

provider id 열을 쿼리하려면 먼저 선택 유형을 선택합니다. DynamicFrame의 resolveChoice 변환 방법을 사용하여 cast:long 옵션으로 string 값을 long 값으로 변환할 수 있습니다.

```
medicare_res = medicare_dynamicframe.resolveChoice(specs = [('provider
  id', 'cast:long')])
medicare_res.printSchema()
```

이제 printSchema 출력값은 다음과 같습니다.

```
root
 |-- drg definition: string
 |-- provider id: long
 |-- provider name: string
 |-- provider street address: string
 |-- provider city: string
 |-- provider state: string
 |-- provider zip code: long
 |-- hospital referral region description: string
 |-- total discharges: long
 |-- average covered charges: string
 |-- average total payments: string
 |-- average medicare payments: string
```

값이 보낼 수 없는 string이면 AWS Glue는 null을 삽입합니다.

다른 방법으로 선택 유형을 struct으로 변환하는 것인데 두 유형의 값은 유지됩니다.

다음, 이례적인 열을 알아봅니다.

```
medicare_res.toDF().where("'provider id' is NULL").show()
```

다음을 알아봅니다.

```

+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
|   drg definition|provider id|  provider name|provider street address|provider
city|provider state|provider zip code|hospital referral region description|total
discharges|average covered charges|average total payments|average medicare payments|
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
|948 - SIGNS & SYM...|      null|          INC|      1050 DIVISION ST|
MAUSTON|          WI|      53948|          WI - Madison|
   12|      $11961.41|          $4619.00|          $3775.33|
|948 - SIGNS & SYM...|      null| INC- ST JOSEPH|      5000 W CHAMBERS ST|
MILWAUKEE|          WI|      53210|          WI - Milwaukee|
   14|      $10514.28|          $5562.50|          $4522.78|
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+

```

이제, 오류 기록을 다음과 같이 제거합니다.

```

medicare_dataframe = medicare_res.toDF()
medicare_dataframe = medicare_dataframe.where("'provider id' is NOT NULL")

```

4단계: 데이터 매핑 및 Apache Spark Lambda 함수 사용

AWS Glue는 사용자 정의 함수인 Lambda 함수를 직접 지원하지 않습니다. 하지만 항상 DynamicFrame와 Apache Spark DataFrame 간에 변환하여 DynamicFrames의 특별한 기능 외에도 Spark 기능도 활용할 수 있습니다.

그런 다음 결제 정보를 숫자로 변환하여 Amazon Redshift 또는 Amazon Athena와 같은 분석 엔진이 빠르게 수 처리를 할 수 있게 만듭니다.

```

from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

chop_f = udf(lambda x: x[1:], StringType())
medicare_dataframe = medicare_dataframe.withColumn(
    "ACC", chop_f(
        medicare_dataframe["average covered charges"])).withColumn(
    "ATP", chop_f(
        medicare_dataframe["average total payments"])).withColumn(
    "AMP", chop_f(

```

```
medicare_dataframe["average medicare payments"]))
medicare_dataframe.select(['ACC', 'ATP', 'AMP']).show()
```

show 호출의 출력값은 다음과 같습니다.

```
+-----+-----+-----+
|   ACC|   ATP|   AMP|
+-----+-----+-----+
|32963.07|5777.24|4763.73|
|15131.85|5787.57|4976.71|
|37560.37|5434.95|4453.79|
|13998.28|5417.56|4129.16|
|31633.27|5658.33|4851.44|
|16920.79|6653.80|5374.14|
|11977.13|5834.74|4761.41|
|35841.09|8031.12|5858.50|
|28523.39|6113.38|5228.40|
|75233.38|5541.05|4386.94|
|67327.92|5461.57|4493.57|
|39607.28|5356.28|4408.20|
|22862.23|5374.65|4186.02|
|31110.85|5366.23|4376.23|
|25411.33|5282.93|4383.73|
| 9234.51|5676.55|4509.11|
|15895.85|5930.11|3972.85|
|19721.16|6192.54|5179.38|
|10710.88|4968.00|3898.88|
|51343.75|5996.00|4962.45|
+-----+-----+-----+
only showing top 20 rows
```

이것은 아직도 모두 데이터에서 문자열입니다. 강력한 `apply_mapping` 변환 방법을 사용하여 데이터를 드롭, 이름 바꾸기, 중첩할 수 있어 다른 데이터 프로그램 언어 및 시스템이 쉽게 접근할 수 있도록 합니다.

```
from awsglue.dynamicframe import DynamicFrame
medicare_tmp_dyf = DynamicFrame.fromDF(medicare_dataframe, glueContext, "nested")
medicare_nest_dyf = medicare_tmp_dyf.apply_mapping([('drg definition', 'string', 'drg',
'provider id', 'long', 'provider.id', 'long'),
('provider name', 'string', 'provider.name', 'string'),
('provider city', 'string', 'provider.city', 'string'),
```

```

        ('provider state', 'string', 'provider.state', 'string'),
        ('provider zip code', 'long', 'provider.zip', 'long'),
        ('hospital referral region description', 'string', 'rr', 'string'),
        ('ACC', 'string', 'charges.covered', 'double'),
        ('ATP', 'string', 'charges.total_pay', 'double'),
        ('AMP', 'string', 'charges.medicare_pay', 'double']]
medicare_nest_dyf.printSchema()

```

printSchema 출력값은 다음과 같습니다.

```

root
 |-- drg: string
 |-- provider: struct
 |   |-- id: long
 |   |-- name: string
 |   |-- city: string
 |   |-- state: string
 |   |-- zip: long
 |-- rr: string
 |-- charges: struct
 |   |-- covered: double
 |   |-- total_pay: double
 |   |-- medicare_pay: double

```

데이터를 Spark DataFrame으로 되돌리면 현재는 어떻게 생겼는지 볼 수 있습니다.

```
medicare_nest_dyf.toDF().show()
```

출력값은 다음과 같습니다.

```

+-----+-----+-----+-----+
|          drg|          provider|          rr|          charges|
+-----+-----+-----+-----+
|039 - EXTRACRANIA...|[10001,SOUTHEAST ...|    AL - Dothan|[32963.07,5777.24...|
|039 - EXTRACRANIA...|[10005,MARSHALL M...|AL - Birmingham|[15131.85,5787.57...|
|039 - EXTRACRANIA...|[10006,ELIZA COFF...|AL - Birmingham|[37560.37,5434.95...|
|039 - EXTRACRANIA...|[10011,ST VINCENT...|AL - Birmingham|[13998.28,5417.56...|
|039 - EXTRACRANIA...|[10016,SHELBY BAP...|AL - Birmingham|[31633.27,5658.33...|
|039 - EXTRACRANIA...|[10023,BAPTIST ME...|AL - Montgomery|[16920.79,6653.8,...|
|039 - EXTRACRANIA...|[10029,EAST ALABA...|AL - Birmingham|[11977.13,5834.74...|
|039 - EXTRACRANIA...|[10033,UNIVERSITY...|AL - Birmingham|[35841.09,8031.12...|
|039 - EXTRACRANIA...|[10039,HUNTSVILLE...|AL - Huntsville|[28523.39,6113.38...|

```

```
|039 - EXTRACRANIA...|[10040,GADSDEN RE...|AL - Birmingham|[75233.38,5541.05...|
|039 - EXTRACRANIA...|[10046,RIVERVIEW ...|AL - Birmingham|[67327.92,5461.57...|
|039 - EXTRACRANIA...|[10055,FLOWERS HO...|    AL - Dothan|[39607.28,5356.28...|
|039 - EXTRACRANIA...|[10056,ST VINCENT...|AL - Birmingham|[22862.23,5374.65...|
|039 - EXTRACRANIA...|[10078,NORTHEAST ...|AL - Birmingham|[31110.85,5366.23...|
|039 - EXTRACRANIA...|[10083,SOUTH BALD...|    AL - Mobile|[25411.33,5282.93...|
|039 - EXTRACRANIA...|[10085,DECATUR GE...|AL - Huntsville|[9234.51,5676.55,...|
|039 - EXTRACRANIA...|[10090,PROVIDENCE...|    AL - Mobile|[15895.85,5930.11...|
|039 - EXTRACRANIA...|[10092,D C H REGI...|AL - Tuscaloosa|[19721.16,6192.54...|
|039 - EXTRACRANIA...|[10100,THOMAS HOS...|    AL - Mobile|[10710.88,4968.0,...|
|039 - EXTRACRANIA...|[10103,BAPTIST ME...|AL - Birmingham|[51343.75,5996.0,...|
+-----+-----+-----+-----+
only showing top 20 rows
```

5단계: 데이터를 Apache Parquet에 쓰기

AWS Glue는 Apache Parquet과 같은 형식으로 데이터를 쉽게 작성할 수 있어 관계형 데이터베이스가 효과적으로 소비될 수 있습니다.

```
glueContext.write_dynamic_frame.from_options(
    frame = medicare_nest_dyf,
    connection_type = "s3",
    connection_options = {"path": "s3://glue-sample-target/output-dir/
medicare_parquet"},
    format = "parquet")
```

AWS Glue PySpark 확장 참조

AWS Glue는 다음 확장 기능을 PySpark Python 언어에 생성합니다.

- [getResolvedOptions를 사용한 파라미터 액세스](#)
- [PySpark 확장 유형](#)
- [DynamicFrame 클래스](#)
- [DynamicFrameCollection 클래스](#)
- [DynamicFrameWriter 클래스](#)
- [DynamicFrameReader 수업](#)
- [GlueContext 수업](#)

getResolvedOptions를 사용한 파라미터 액세스

AWS Glue `getResolvedOptions(args, options)` 유틸리티 함수는 작업이 실행되면 스크립트로 통과된 인수로 액세스할 수 있도록 도와줍니다. 이 함수를 사용하려면 먼저 `sys` 모듈과 함께 AWS Glue `utils` 모듈에서 가져옵니다.

```
import sys
from awsglue.utils import getResolvedOptions
```

getResolvedOptions(args, options)

- `args` - `sys.argv`에 포함된 인수 목록입니다.
- `options` - 가져오고자 하는 인수 이름의 Python 배열입니다.

Example JobRun으로 통과된 인수 가져오기

JobRun을 Lambda 함수 내 스크립트에 생성했다고 가정합니다.

```
response = client.start_job_run(
    JobName = 'my_test_job',
    Arguments = {
        '--day_partition_key': 'partition_0',
        '--hour_partition_key': 'partition_1',
        '--day_partition_value': day_partition_value,
        '--hour_partition_value': hour_partition_value } )
```

통과된 인수를 가져오기 위해서는 다음과 같이 `getResolvedOptions` 함수를 사용합니다.

```
import sys
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv,
                          ['JOB_NAME',
                           'day_partition_key',
                           'hour_partition_key',
                           'day_partition_value',
                           'hour_partition_value'])

print "The day-partition key is: ", args['day_partition_key']
print "and the day-partition value is: ", args['day_partition_value']
```


각 인수는 두 개의 하이픈으로 시작된 다음 하이픈 없이 스크립트에서 참조되는 것으로 정의됩니다. 인수는 하이픈이 아닌 밑줄만 사용합니다. 인수가 해석되려면 이 규칙을 따라야 합니다.

PySpark 확장 유형

AWS Glue PySpark 확장에 사용된 유형.

데이터 유형

다른 AWS Glue 유형의 베이스 클래스.

__init__(properties={})

- `properties` - 데이터 유형의 속성(선택 사항)입니다.

typeName(cls)

AWS Glue 유형 클래스(즉, 끝에서 "Type"이 제거된 클래스)의 유형을 반환합니다.

- `cls` - `DataType`에서 파생된 AWS Glue 클래스 인스턴스입니다.

`jsonValue()`

클래스 속성의 데이터 유형을 포함하는 JSON 객체를 반환합니다.

```
{
  "dataType": typeName,
  "properties": properties
}
```

AtomicType 및 단순 파생물

[데이터 유형](#) 클래스에서 상속되고 확장되며 모든 AWS Glue 원자 데이터 유형의 베이스 클래스로서 역할을 합니다.

fromJsonValue(cls, json_value)

JSON 객체의 값으로 클래스 인스턴스를 초기화합니다.

- `cls` - 초기화할 AWS Glue 유형 클래스 인스턴스입니다.
- `json_value` - JSON 객체에서 키-값 페어를 로드합니다.

다음 유형은 [AtomicType](#) 클래스의 단순 파생물입니다.

- `BinaryType` - 이진 데이터입니다.
- `BooleanType` - 부울 값입니다.
- `ByteType` - 바이트 값입니다.
- `DateType` - 날짜/시간 값입니다.
- `DoubleType` - 부동 소수점 값입니다.
- `IntegerType` - 정수 값입니다.
- `LongType` - 정수(long) 값입니다.
- `NullType` - null 값입니다.
- `ShortType` - 정수(short) 값입니다.
- `StringType` - 텍스트 문자열입니다.
- `TimestampType` - 타임스탬프 값(일반적으로 1/1/1970에서 초 단위)입니다.
- `UnknownType` - 식별되지 않은 유형의 값입니다.

`DecimalType(AtomicType)`

[AtomicType](#) 클래스에서 상속받고 확장하여 십진수(이진수 기반 2 숫자와 반대 개념인 십진수로 표현된 수)를 나타냅니다.

`__init__(precision=10, scale=2, properties={})`

- `precision` - 십진수의 단위 수입니다(선택; 기본값이 10).
- `scale` - 십진수 오른쪽의 단위 수입니다(선택; 기본값이 2).
- `properties` - 십진수 유형의 속성(선택 사항).

`EnumType(AtomicType)`

[AtomicType](#) 클래스에서 상속받고 확장되어 유효한 옵션을 열거합니다.

__init__(options)

- options – 열거된 옵션 목록입니다.

컬렉션 형식

- [ArrayType\(DataType\)](#)
- [ChoiceType\(DataType\)](#)
- [MapType\(DataType\)](#)
- [필드\(객체\)](#)
- [StructType\(DataType\)](#)
- [EntityType\(DataType\)](#)

ArrayType(DataType)

__init__(elementType=UnknownType(), properties={})

- elementType – 배열의 요소 유형(선택; 기본값은 UnknownType입니다).
- properties - 배열의 속성입니다(선택 사항).

ChoiceType(DataType)

__init__(choices=[], properties={})

- choices - 가능한 선택 항목의 목록입니다(선택 사항).
- properties - 이러한 선택 항목의 속성입니다(선택 사항).

add(new_choice)

가능한 선택 목록에 새로운 선택을 추가합니다.

- new_choice – 가능한 선택 항목 목록에 추가할 선택 항목입니다.

merge(new_choices)

기존 선택 목록과 새로운 선택 목록을 병합합니다.

- `new_choices` – 기존 선택 항목과 병합할 새 선택 항목 목록입니다.

`MapType(DataType)`

`__init__(valueType=UnknownType, properties={})`

- `valueType` – 맵 값의 유형(선택; 기본값은 `UnknownType`입니다).
- `properties` - 맵의 속성입니다(선택 사항).

필드(객체)

[데이터 유형](#)에서 상속받은 객체에서 필드 객체를 생성합니다.

`__init__(name, dataType, properties={})`

- `name` – 필드에 지정된 이름입니다.
- `dataType` - 필드를 생성할 객체입니다.
- `properties` - 필드의 속성입니다(선택 사항).

`StructType(DataType)`

데이터 구조를 정의합니다 (struct).

`__init__(fields=[], properties={})`

- `fields` – 구조에 포함할 `Field` 유형의 필드는 목록입니다(선택 사항).
- `properties` - 구조의 속성입니다(선택 사항).

`add(field)`

- `field` – 구조에 추가할 `Field` 유형의 객체입니다.

`hasField(field)`

이 구조가 동일 이름의 필드를 가지고 있다면 `True`를 반환하고 그렇지 않으면 `False`를 반환합니다.

- `field` - 필드 이름 혹은 이름이 사용된 `Field` 유형의 객체입니다.

getField(field)

- `field` - 필드 이름 혹은 이름이 사용된 `Field` 유형의 객체입니다. 구조가 동일 이름의 필드를 가지고 있다면 이것은 반환됩니다.

EntityType(DataType)

`__init__(entity, base_type, properties)`

이 클래스는 아직 구현되지 않았습니다.

기타 유형

- [DataSource\(객체\)](#)
- [DataSink\(객체\)](#)

DataSource(객체)

`__init__(j_source, sql_ctx, name)`

- `j_source` - 데이터 원본입니다.
- `sql_ctx` - SQL 컨텍스트입니다.
- `name` - 데이터 원본 이름입니다.

setFormat(format, **options)

- `format` - 데이터 원본을 설정하기 위한 포맷입니다.
- `options` - 데이터 원본을 설정하기 위한 옵션 모음입니다. 형식 옵션에 대한 자세한 내용은 [the section called “데이터 포맷 옵션”](#) 섹션을 참조하세요.

getFrame()

데이터 원본을 위한 `DynamicFrame`을 반환합니다.

DataSink(객체)

`__init__(j_sink, sql_ctx)`

- `j_sink` - 생성할 싱크입니다.
- `sql_ctx` - 데이터 싱크를 위한 SQL 컨텍스트입니다.

`setFormat(format, **options)`

- `format` - 데이터 싱크에 대해 설정할 포맷입니다.
- `options` - 데이터 싱크를 설정하기 위한 옵션 모음입니다. 형식 옵션에 대한 자세한 내용은 [the section called “데이터 포맷 옵션”](#) 섹션을 참조하세요.

`setAccumulableSize(size)`

- `size` - 바이트 단위로 설정할 늘어날 수 있는 사이즈입니다.

`writeFrame(dynamic_frame, info="")`

- `dynamic_frame` - 작성할 `DynamicFrame`입니다.
- `info` - `DynamicFrame`에 대한 정보입니다(선택 사항).

`write(dynamic_frame_or_dfc, info="")`

`DynamicFrame` 혹은 `DynamicFrameCollection` 작성.

- `dynamic_frame_or_dfc` - 작성할 `DynamicFrame` 객체 또는 `DynamicFrameCollection` 객체입니다.
- `info` - 작성할 `DynamicFrame` 또는 `DynamicFrames`에 대한 정보입니다(선택 사항).

DynamicFrame 클래스

Apache Spark의 주요 추상화 중 하나는 SparkSQL `DataFrame`이며, R 및 Pandas에서 찾아볼 수 있는 `DataFrame` 구문과 유사합니다. `DataFrame`은 테이블과 비슷한 기능적 스타일(맵/줄임/필터 등) 작업과 SQL 작업(선택, 계획, 집계)을 지원합니다.

DataFrames 는 광범위하게 사용되는 유용한 것이지만 추출, 변환, 로드(ETL) 작업 시 제한이 있습니다. 무엇보다도 데이터가 로딩되기 전에 스키마를 명시해야 합니다. 첫 번째는 스키마를 추론하고 두 번째는 데이터를 로드하도록 두 개를 데이터에 통과시켜 SparkSQL이 이를 해결합니다. 하지만 이러한 추론은 제한적이며 복잡한 데이터의 현실을 해결해주지 않습니다. 예를 들어, 동일한 필드는 다른 기록에서 다른 유형을 가져야 합니다. 아파치 스파크는 간혹 실행을 포기하고 기존 필드 텍스트를 사용하여 string으로써 유형을 보고합니다. 이는 올바르지 않을 수 있고 스키마 차이를 해결할 수 있는 좀 더 확실한 관리법을 알고 싶을 겁니다. 라지 데이터세트의 경우, 추가로 소스 데이터를 통과하는 것이 매우 비쌀 수 있습니다.

이러한 상황을 해결하려면 AWS Glue는 DynamicFrame을 도입합니다. DynamicFrame는 애초에 스키마가 필요없이 각 기록이 자기 설명적인 것을 제외하고는 DataFrame와 비슷합니다. 대신에 AWS Glue는 필요하면 스키마 온더플라이를 계산하고 선택 (혹은 재결합) 유형을 사용하여 스키마 불일치를 확실하게 암호화합니다. 이런 불일치를 해결하여 데이터세트와 고정 스키마가 필요한 데이터 스토어를 호환하게 만들 수 있습니다.

비슷하게, DynamicRecord는 DynamicFrame내 논리적 기록입니다. 자기 설명적이고 고정 스키마를 준수하지 않은 데이터에 사용될 수 있다는 점을 제외하면 스파크 DataFrame의 열과 같습니다. PySpark에서 AWS Glue를 사용하는 경우 일반적으로 독립된 DynamicRecords를 조작하지 않습니다. 대신, 해당 DynamicFrame을 통해 데이터 세트를 함께 변환합니다.

모든 스키마 불일치를 해결한 후 DynamicFrames로 그리고 DataFrames에서 변환할 수 있습니다.

- 생성 -

- [__init__](#)
- [fromDF](#)
- [toDF](#)

[__init__](#)

`__init__(jdf, glue_ctx, name)`

- jdf - Java Virtual Machine(JVM)의 데이터 프레임 참조입니다.
- glue_ctx - [GlueContext](#) 수업 객체입니다.
- name - 기본값이 빈 선택적 이름 문자열입니다.

fromDF

fromDF(dataframe, glue_ctx, name)

DataFrame 필드를 DynamicRecord 필드로 변환하여 DataFrame를 DynamicFrame로 변환합니다. 새로운 DynamicFrame을 반환합니다.

DynamicRecord는 DynamicFrame내 논리적 기록입니다. 자기 설명적이고 고정 스키마를 준수하지 않은 데이터에 사용될 수 있다는 점을 제외하면 스파크 DataFrame의 열과 비슷합니다.

이 함수는 DataFrame에서 이름이 중복된 열이 이미 해결된 것으로 예상합니다.

- dataframe - Apache Spark SQL DataFrame으로 변환합니다(필수).
- glue_ctx - 이 변환의 맥락을 명시하는 [GlueContext 수업](#) 객체입니다(필수).
- name - 결과 DynamicFrame의 이름(AWS Glue 3.0부터 선택 사항).

toDF

toDF(options)

DynamicRecords를 DataFrame 필드로 변환하여 DynamicFrame을 Apache Spark DataFrame으로 변환합니다. 새로운 DataFrame을 반환합니다.

DynamicRecord는 DynamicFrame내 논리적 기록입니다. 자기 설명적이고 고정 스키마를 준수하지 않은 데이터에 사용될 수 있다는 점을 제외하면 스파크 DataFrame의 열과 비슷합니다.

- options - 옵션의 목록입니다. Project와 Cast 작업 유형을 선택한 경우 대상 유형을 지정합니다. 예는 다음과 같습니다.

```
>>>toDF([ResolveOption("a.b.c", "KeepAsStruct")])
>>>toDF([ResolveOption("a.b.c", "Project", DoubleType())])
```

- 정보 -

- [count](#)
- [schema](#)
- [printSchema](#)
- [show](#)

- [repartition](#)
- [coalesce](#)

count

`count()` - 기본 DataFrame의 행 수를 반환합니다.

schema

`schema()` - 이 DynamicFrame의 스키마를 반환하거나, 가능하지 않는 경우 기본 DataFrame의 스키마를 반환합니다.

이 스키마를 구성하는 DynamicFrame 유형에 대한 자세한 내용은 [the section called “타입”](#) 섹션을 참조하세요.

printSchema

`printSchema()` - 기본 DataFrame의 스키마를 인쇄합니다.

show

`show(num_rows)` - 기본 DataFrame으로부터 지정된 수의 행을 인쇄합니다.

repartition

`repartition(numPartitions)` - numPartitions 파티션이 있는 새 DynamicFrame을 반환합니다.

coalesce

`coalesce(numPartitions)` - numPartitions 파티션이 있는 새 DynamicFrame을 반환합니다.

- 변형 -

- [apply_mapping](#)
- [drop_fields](#)
- [필터](#)
- [join](#)
- [map](#)

- [mergeDynamicFrame](#)
- [relationalize](#)
- [rename_field](#)
- [resolveChoice](#)
- [select_fields](#)
- [spigot](#)
- [split_fields](#)
- [split_rows](#)
- [unbox](#)
- [the section called “결합”](#)
- [unnest](#)
- [unnest_ddb_json](#)
- [write](#)

apply_mapping

apply_mapping(mappings, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

DynamicFrame으로 매핑을 선언하고 지정한 필드에 해당 매핑이 적용된 새로운 DynamicFrame을 반환합니다. 지정되지 않은 필드는 새 DynamicFrame 필드에서 생략됩니다.

- mappings - 매핑 튜플 목록(필수). 각각 (소스 열, 소스 유형, 대상 열, 대상 유형)으로 구성된 매핑 튜플 목록입니다.

소스 열 이름에 점('.')이 있는 경우 백틱('`')으로 묶어야 합니다. 예를 들어 this.old.name(문자열)을 thisNewName에 매핑하려면 다음 튜플을 사용합니다.

```
("`this.old.name`", "string", "thisNewName", "string")
```

- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- stageThreshold - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

- `totalThreshold` - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예: `apply_mapping`을 사용하여 필드 이름을 바꾸고 필드 유형을 변경합니다.

다음 코드 예제에서는 `apply_mapping` 메서드를 사용하여 선택한 필드의 이름을 바꾸고 필드 유형을 변경하는 방법을 보여 줍니다.

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: 데이터 조인 및 관계화](#) 항목을 참조하고 [1단계: Amazon S3 버킷에서 데이터 크롤](#)의 지침을 따르세요.

```
# Example: Use apply_mapping to reshape source data into
# the desired column names and types as a new DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Select and rename fields, change field type
print("Schema for the persons_mapped DynamicFrame, created with apply_mapping:")
persons_mapped = persons.apply_mapping(
    [
        ("family_name", "String", "last_name", "String"),
        ("name", "String", "first_name", "String"),
        ("birth_date", "String", "date_of_birth", "Date"),
    ]
)
persons_mapped.printSchema()
```

출력

Schema for the persons DynamicFrame:

```
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
```

Schema for the persons_mapped DynamicFrame, created with apply_mapping:

```
root
|-- last_name: string
|-- first_name: string
|-- date_of_birth: date
```

drop_fields

drop_fields(paths, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

[FlatMap 클래스](#) 변환을 불러들여 DynamicFrame에서 필드를 제거합니다. 특정 필드가 드롭된 새로운 DynamicFrame을 반환합니다.

- **paths** - 문자열 목록입니다. 각각에는 드롭할 필드 노드에 대한 전체 경로가 포함되어 있습니다. 점 표기를 사용하여 중첩된 필드를 지정할 수 있습니다. 예를 들어 first 필드가 트리에서 name 필드의 자식 필드인 경우 해당 경로에 "name.first"를 지정합니다.

필드 노드 이름에 . 리터럴이 있는 경우 이름을 백틱(`)으로 묶어야 합니다.

- **transformation_ctx** - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- **info** - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- **stageThreshold** - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- **totalThreshold** - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예: drop_fields를 사용하여 **DynamicFrame**에서 필드를 제거합니다.

이 코드 예제에서는 drop_fields 메서드를 사용하여 DynamicFrame에서 선택한 최상위 및 중첩 필드를 제거합니다.

예제 데이터 세트

이 예제에서는 코드의 EXAMPLE-FRIENDS-DATA 테이블로 표시되는 다음 데이터 세트를 사용합니다.

```
{
  "name": "Sally", "age": 23, "location": {"state": "WY", "county": "Fremont"},
  "friends": []
},
{
  "name": "Varun", "age": 34, "location": {"state": "NE", "county": "Douglas"},
  "friends": [{"name": "Arjun", "age": 3}]
},
{
  "name": "George", "age": 52, "location": {"state": "NY"},
  "friends": [{"name": "Fred"}, {"name": "Amy", "age": 15}]
},
{
  "name": "Haruki", "age": 21, "location": {"state": "AK", "county": "Denali"}
},
{
  "name": "Sheila", "age": 63, "friends": [{"name": "Nancy", "age": 22}]
}
```

예제 코드

```

# Example: Use drop_fields to remove top-level and nested fields from a DynamicFrame.
# Replace MY-EXAMPLE-DATABASE with your Glue Data Catalog database name.
# Replace EXAMPLE-FRIENDS-DATA with your table name.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame from Glue Data Catalog
glue_source_database = "MY-EXAMPLE-DATABASE"
glue_source_table = "EXAMPLE-FRIENDS-DATA"

friends = glueContext.create_dynamic_frame.from_catalog(
    database=glue_source_database, table_name=glue_source_table
)
print("Schema for friends DynamicFrame before calling drop_fields:")
friends.printSchema()

# Remove location.county, remove friends.age, remove age
friends = friends.drop_fields(paths=["age", "location.county", "friends.age"])
print("Schema for friends DynamicFrame after removing age, county, and friend age:")
friends.printSchema()

```

출력

```

Schema for friends DynamicFrame before calling drop_fields:
root
|-- name: string
|-- age: int
|-- location: struct
|   |-- state: string
|   |-- county: string
|-- friends: array
|   |-- element: struct
|   |   |-- name: string
|   |   |-- age: int

Schema for friends DynamicFrame after removing age, county, and friend age:
root
|-- name: string

```

```
 |-- location: struct
 |   |-- state: string
 |-- friends: array
 |   |-- element: struct
 |       |-- name: string
```

필터

filter(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

지정된 술어 함수 `f`를 만족하는 `DynamicFrame`이 입력된 `DynamicRecords`에 모두 포함된 새로운 `DynamicFrame`을 반환합니다.

- `f` - `DynamicFrame`에 적용하는 조건자 함수입니다. 함수는 `DynamicRecord`를 논리로 받아들이며, `DynamicRecord`가 필터 요구 사항과 맞으면 `True`을 반환하고 아니면 `False`를 반환합니다(필수).

`DynamicRecord`는 `DynamicFrame`내 논리적 기록입니다. 자기 설명적이고 고정 스키마를 준수하지 않은 데이터에 사용될 수 있다는 점을 제외하면 스파크 `DataFrame`의 열과 비슷합니다.

- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- `totalThreshold` - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예: 필터를 사용하여 필터링된 필드 선택 가져오기

이 예에서는 `filter` 메서드를 사용하여 다른 사람의 `DynamicFrame` 필드에 대한 필터링된 선택을 포함하는 새 `DynamicFrame` 항목을 만듭니다.

`map` 메서드와 같이 `filter`는 함수를 인수로 취하여 원본 `DynamicFrame`의 각 레코드에 적용됩니다. 이 함수는 레코드를 입력으로 받아 부울 값을 반환합니다. 반환 값이 `true`이면 레코드가 결과 `DynamicFrame`에 포함됩니다. 거짓이면 레코드가 제외됩니다.

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: ResolveChoice, Lambda, 및 ApplyMapping을 사용한 데이터 준비](#) 항목을 참조하고 [1단계: Amazon S3 버킷에서 데이터 크롤의 지침을 따르세요.](#)

```
# Example: Use filter to create a new DynamicFrame
# with a filtered selection of records

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create DynamicFrame from Glue Data Catalog
medicare = glueContext.create_dynamic_frame.from_options(
    "s3",
    {
        "paths": [
            "s3://awsglue-datasets/examples/medicare/Medicare_Hospital_Provider.csv"
        ]
    },
    "csv",
    {"withHeader": True},
)

# Create filtered DynamicFrame with custom lambda
# to filter records by Provider State and Provider City
sac_or_mon = medicare.filter(
    f=lambda x: x["Provider State"] in ["CA", "AL"]
    and x["Provider City"] in ["SACRAMENTO", "MONTGOMERY"]
)

# Compare record counts
print("Unfiltered record count: ", medicare.count())
print("Filtered record count: ", sac_or_mon.count())
```


출력

```
Unfiltered record count: 163065
Filtered record count: 564
```

join

```
join(paths1, paths2, frame2, transformation_ctx="", info="",
stageThreshold=0, totalThreshold=0)
```

다른 DynamicFrame로 균등 연결을 실행하고 결과인 DynamicFrame을 반환합니다.

- paths1 - 조인할 이 프레임의 키 목록입니다.
- paths2 - 조인할 다른 프레임의 키 목록입니다.
- frame2 - 조인할 다른 DynamicFrame입니다.
- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- stageThreshold - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- totalThreshold - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예: 조인을 사용하여 DynamicFrames 결합

이 예제에서는 join 메서드를 사용하여 3개의 DynamicFrames에서 조인을 수행합니다. AWS Glue 는 사용자가 제공한 필드 키를 기반으로 조인을 수행합니다. 결과 DynamicFrame에는 지정된 키가 일치하는 두 원본 프레임의 행이 포함됩니다.

join 변환은 모든 필드를 그대로 유지한다는 점에 유의하세요. 즉, 일치하도록 지정하는 필드가 중복 되고 동일한 키가 포함된 경우에도 결과 DynamicFrame에 나타납니다. 이 예에서는 drop_fields를 사용하여 조인 후 이러한 중복 키를 제거합니다.

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: 데이터 조인 및 관계화](#) 항목을 참조하고 [1단계: Amazon S3 버킷에서 데이터 크롤](#)의 지침을 따르세요.

```
# Example: Use join to combine data from three DynamicFrames

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load DynamicFrames from Glue Data Catalog
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
memberships = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="memberships_json"
)
orgs = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="organizations_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()
print("Schema for the memberships DynamicFrame:")
memberships.printSchema()
print("Schema for the orgs DynamicFrame:")
orgs.printSchema()

# Join persons and memberships by ID
persons_memberships = persons.join(
    paths1=["id"], paths2=["person_id"], frame2=memberships
)

# Rename and drop fields from orgs
# to prevent field name collisions with persons_memberships
orgs = (
    orgs.drop_fields(["other_names", "identifiers"])
    .rename_field("id", "org_id")
    .rename_field("name", "org_name")
)

# Create final join of all three DynamicFrames
legislators_combined = orgs.join(
    paths1=["org_id"], paths2=["organization_id"], frame2=persons_memberships
).drop_fields(["person_id", "org_id"])
```

```
# Inspect the schema for the joined data
print("Schema for the new legislators_combined DynamicFrame:")
legislators_combined.printSchema()
```

출력

Schema for the persons DynamicFrame:

```
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
```

Schema for the memberships DynamicFrame:

```
root
|-- area_id: string
|-- on_behalf_of_id: string
```

```
|-- organization_id: string
|-- role: string
|-- person_id: string
|-- legislative_period_id: string
|-- start_date: string
|-- end_date: string
```

Schema for the orgs DynamicFrame:

```
root
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- id: string
|-- classification: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- image: string
|-- seats: int
|-- type: string
```

Schema for the new legislators_combined DynamicFrame:

```
root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
```

```

|-- on_behalf_of_id: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- legislative_period_id: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- image: string
|-- given_name: string
|-- start_date: string
|-- family_name: string
|-- id: string
|-- death_date: string
|-- end_date: string

```

map

map(f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

특정 매핑 함수를 기존 DynamicFrame의 모든 기록에 적용한 결과인 새로운 DynamicFrame을 반환합니다.

- f - DynamicFrame의 모든 레코드에 적용하는 매핑 함수입니다. 함수는 DynamicRecord를 인수로 받아들이고 새로운 DynamicRecord를 반환합니다(필수).

DynamicRecord는 DynamicFrame내 논리적 기록입니다. 자기 설명적이고 고정 스키마를 준수하지 않은 데이터에 사용될 수 있다는 점을 제외하면 Apache Spark DataFrame의 열과 비슷합니다.

- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 변환에 따른 오류 관련 문자열입니다(선택 사항).

- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

예: `map`을 사용하여 **DynamicFrame**의 모든 레코드에 함수를 적용합니다.

이 예제에서는 `map` 메서드를 사용하여 `DynamicFrame`의 모든 레코드에 기능을 적용하는 방법을 보여 줍니다. 특히 이 예에서는 여러 주소 필드를 단일 `struct` 유형으로 병합하기 위해 각 레코드에 `MergeAddress`라는 함수를 적용합니다.

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: ResolveChoice, Lambda, 및 ApplyMapping을 사용한 데이터 준비 항목을 참조하고 1단계: Amazon S3 버킷에서 데이터 크롤의 지침을 따르세요.](#)

```
# Example: Use map to combine fields in all records
# of a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
medicare = glueContext.create_dynamic_frame.from_options(
    "s3",
    [{"paths": ["s3://awsglue-datasets/examples/medicare/
Medicare_Hospital_Provider.csv"]}],
    "csv",
    {"withHeader": True})
print("Schema for medicare DynamicFrame:")
medicare.printSchema()

# Define a function to supply to the map transform
# that merges address fields into a single field
```

```
def MergeAddress(rec):
    rec["Address"] = {}
    rec["Address"]["Street"] = rec["Provider Street Address"]
    rec["Address"]["City"] = rec["Provider City"]
    rec["Address"]["State"] = rec["Provider State"]
    rec["Address"]["Zip.Code"] = rec["Provider Zip Code"]
    rec["Address"]["Array"] = [rec["Provider Street Address"], rec["Provider City"],
rec["Provider State"], rec["Provider Zip Code"]]
    del rec["Provider Street Address"]
    del rec["Provider City"]
    del rec["Provider State"]
    del rec["Provider Zip Code"]
    return rec

# Use map to apply MergeAddress to every record
mapped_medicare = medicare.map(f = MergeAddress)
print("Schema for mapped_medicare DynamicFrame:")
mapped_medicare.printSchema()
```

출력

```
Schema for medicare DynamicFrame:
root
|-- DRG Definition: string
|-- Provider Id: string
|-- Provider Name: string
|-- Provider Street Address: string
|-- Provider City: string
|-- Provider State: string
|-- Provider Zip Code: string
|-- Hospital Referral Region Description: string
|-- Total Discharges: string
|-- Average Covered Charges: string
|-- Average Total Payments: string
|-- Average Medicare Payments: string

Schema for mapped_medicare DynamicFrame:
root
|-- Average Total Payments: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
```

```

|-- Hospital Referral Region Description: string
|-- Address: struct
|   |-- Zip.Code: string
|   |-- City: string
|   |-- Array: array
|       |-- element: string
|   |-- State: string
|   |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string

```

mergeDynamicFrame

mergeDynamicFrame(stage_dynamic_frame, primary_keys, transformation_ctx = "", options = {}, info = "", stageThreshold = 0, totalThreshold = 0)

레코드를 식별하기 위해, 지정된 기본 키를 기반으로 이 DynamicFrame을 스테이징 DynamicFrame과 병합합니다. 중복 레코드(기본 키가 동일한 레코드)는 중복 제거되지 않습니다. 스테이징 프레임에 일치하는 레코드가 없으면 모든 레코드(중복 레코드 포함)가 소스에서 유지됩니다. 스테이징 프레임에 일치하는 레코드가 있으면 스테이징 프레임의 레코드가 AWS Glue의 소스에 있는 레코드를 덮어씁니다.

- `stage_dynamic_frame` - 병합할 스테이징 DynamicFrame입니다.
- `primary_keys` - 소스 및 스테이징 동적 프레임의 레코드와 일치시킬 기본 키 필드 목록입니다.
- `transformation_ctx` - 현재 변환에 대한 메타데이터를 검색하는 데 사용되는 고유한 문자열입니다(선택 사항).
- `options` - 이 변환에 대한 추가 정보를 제공하는 JSON 이름-값 페어 문자열입니다. 이 인수는 현재 사용되지 않습니다.
- `info` - String입니다. 이 변환에 따른 오류와 관련된 문자열입니다.
- `stageThreshold` - Long입니다. 오류가 발생하는 지정된 변환의 오류 수입니다.
- `totalThreshold` - Long입니다. 오류가 발생하는 이 변환의 총 오류 수입니다.

이 메소드는 이 DynamicFrame을 스테이징 DynamicFrame과 병합하여 얻은 새 DynamicFrame을 반환합니다.

다음과 같은 경우 반환된 DynamicFrame에 레코드 A가 포함되어 있습니다.

- A가 소스 프레임과 스테이징 프레임 모두에 있는 경우에는 스테이징 프레임의 A가 반환됩니다.

- A가 소스 테이블에 있고 A.primaryKeys가 stagingDynamicFrame에 없는 경우에는 A는 스테이징 테이블에서 업데이트되지 않습니다.

소스 프레임과 스테이징 프레임이 동일한 스키마를 가질 필요는 없습니다.

예: MergeDynamicFrame을 사용하여 기본 키를 기반으로 두 개 **DynamicFrames**를 병합합니다

다음 코드 예제는 기본 키 id를 기반으로 mergeDynamicFrame 메서드를 사용하여 DynamicFrame를 “스테이징” DynamicFrame과 병합하는 방법을 보여줍니다.

예제 데이터 세트

이 예제에서는 split_rows_collection로 호출된 DynamicFrameCollection에서 두 개의 DynamicFrames을 사용합니다. 다음은 split_rows_collection에서의 키 목록입니다.

```
dict_keys(['high', 'low'])
```

예제 코드

```
# Example: Use mergeDynamicFrame to merge DynamicFrames
# based on a set of specified primary keys

from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import SelectFromCollection

# Inspect the original DynamicFrames
frame_low = SelectFromCollection.apply(dfc=split_rows_collection, key="low")
print("Inspect the DynamicFrame that contains rows where ID < 10")
frame_low.toDF().show()

frame_high = SelectFromCollection.apply(dfc=split_rows_collection, key="high")
print("Inspect the DynamicFrame that contains rows where ID > 10")
frame_high.toDF().show()

# Merge the DynamicFrames based on the "id" primary key
merged_high_low = frame_high.mergeDynamicFrame(
    stage_dynamic_frame=frame_low, primary_keys=["id"]
)

# View the results where the ID is 1 or 20
print("Inspect the merged DynamicFrame that contains the combined rows")
```

```
merged_high_low.toDF().where("id = 1 or id= 20").orderBy("id").show()
```

출력

Inspect the DynamicFrame that contains rows where ID < 10

```
+---+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+-----+
| 1|  0|          fax|          202-225-3307|
| 1|  1|          phone|          202-225-5731|
| 2|  0|          fax|          202-225-3307|
| 2|  1|          phone|          202-225-5731|
| 3|  0|          fax|          202-225-3307|
| 3|  1|          phone|          202-225-5731|
| 4|  0|          fax|          202-225-3307|
| 4|  1|          phone|          202-225-5731|
| 5|  0|          fax|          202-225-3307|
| 5|  1|          phone|          202-225-5731|
| 6|  0|          fax|          202-225-3307|
| 6|  1|          phone|          202-225-5731|
| 7|  0|          fax|          202-225-3307|
| 7|  1|          phone|          202-225-5731|
| 8|  0|          fax|          202-225-3307|
| 8|  1|          phone|          202-225-5731|
| 9|  0|          fax|          202-225-3307|
| 9|  1|          phone|          202-225-5731|
| 10| 0|          fax|          202-225-6328|
| 10| 1|          phone|          202-225-4576|
+---+-----+-----+-----+
```

only showing top 20 rows

Inspect the DynamicFrame that contains rows where ID > 10

```
+---+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+-----+
| 11|  0|          fax|          202-225-6328|
| 11|  1|          phone|          202-225-4576|
| 11|  2|        twitter|        RepTrentFranks|
| 12|  0|          fax|          202-225-6328|
| 12|  1|          phone|          202-225-4576|
| 12|  2|        twitter|        RepTrentFranks|
| 13|  0|          fax|          202-225-6328|
| 13|  1|          phone|          202-225-4576|
```

```

| 13| 2|          twitter| RepTrentFranks|
| 14| 0|           fax|    202-225-6328|
| 14| 1|          phone|    202-225-4576|
| 14| 2|          twitter| RepTrentFranks|
| 15| 0|           fax|    202-225-6328|
| 15| 1|          phone|    202-225-4576|
| 15| 2|          twitter| RepTrentFranks|
| 16| 0|           fax|    202-225-6328|
| 16| 1|          phone|    202-225-4576|
| 16| 2|          twitter| RepTrentFranks|
| 17| 0|           fax|    202-225-6328|
| 17| 1|          phone|    202-225-4576|

```

```

+---+-----+-----+-----+-----+
only showing top 20 rows

```

Inspect the merged `DynamicFrame` that contains the combined rows

```

+---+-----+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+-----+-----+
| 1| 0|          fax|    202-225-3307|
| 1| 1|          phone|    202-225-5731|
| 20| 0|          fax|    202-225-5604|
| 20| 1|          phone|    202-225-6536|
| 20| 2|          twitter|    USRepLong|
+---+-----+-----+-----+-----+

```

relationalize

```

relationalize(root_table_name, staging_path, options,
transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

```

`DynamicFrame`을 관계형 데이터베이스에 맞는 형식으로 변환합니다. `DynamicFrame`을 관계형으로 만들면 DynamoDB와 같은 NoSQL 환경에서 MySQL과 같은 관계형 데이터베이스로 데이터를 이동하려는 경우에 특히 유용합니다.

변환은 중첩된 열의 중첩을 해제하고 배열 열을 회전하여 프레임 목록을 생성합니다. 회전 배열 열은 중첩되지 않는 상태에서 생성된 조인 키를 사용하여 루트 테이블에 연결될 수 있습니다.

- `root_table_name` - 루트 테이블의 스키마 이름입니다.
- `staging_path` - 메소드가 CSV 포맷으로 회전 테이블의 파티션을 저장하는 경로입니다(선택 사항). 회전 테이블은 이 경로부터 다시 읽습니다.

- `options` - 선택적 파라미터의 디렉터리입니다.
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- `totalThreshold` - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예: 관계형 만들기를 사용하여 **DynamicFrame** 내 중첩된 스키마를 플랫폼하게 만들기

이 코드 예제는 `relationalize` 메서드를 사용하여 중첩된 스키마를 관계형 데이터베이스에 맞는 형식으로 평면화합니다.

예제 데이터 세트

이 예제에서는 다음 스키마와 함께 `legislators_combined`를 호출한 `DynamicFrame`을 사용합니다. `legislators_combined`은 `links`, `images`, 및 `contact_details`와 같은 여러 개의 중첩 필드를 가지고, `relationalize` 변환에 의해 평면화됩니다.

```

root
|-- role: string
|-- seats: int
|-- org_name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- type: string
|-- sort_name: string
|-- area_id: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- on_behalf_of_id: string
|-- other_names: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- name: string
|   |   |-- lang: string
|-- contact_details: array

```

```

|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- name: string
|-- birth_date: string
|-- organization_id: string
|-- gender: string
|-- classification: string
|-- legislative_period_id: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- image: string
|-- given_name: string
|-- start_date: string
|-- family_name: string
|-- id: string
|-- death_date: string
|-- end_date: string

```

예제 코드

```

# Example: Use relationalize to flatten
# a nested schema into a format that fits
# into a relational database.
# Replace DOC-EXAMPLE-S3-BUCKET/tmpDir with your own location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Apply relationalize and inspect new tables
legislators_relationalized = legislators_combined.relationalize(
    "l_root", "s3://DOC-EXAMPLE-BUCKET/tmpDir"
)
legislators_relationalized.keys()

# Compare the schema of the contact_details
# nested field to the new relationalized table that

```

```
# represents it
legislators_combined.select_fields("contact_details").printSchema()
legislators_relationalized.select("l_root_contact_details").toDF().where(
    "id = 10 or id = 75"
).orderBy(["id", "index"]).show()
```

출력

다음 출력을 통해 `contact_details`을 호출한 중첩 필드의 스키마를 `relationalize` 변환이 만든 테이블과 비교할 수 있습니다. 테이블 레코드는 `id`라는 외래 키와 배열의 위치를 나타내는 `index` 열을 사용하여 기본 테이블로 다시 연결됩니다.

```
dict_keys(['l_root', 'l_root_images', 'l_root_links', 'l_root_other_names',
'l_root_contact_details', 'l_root_identifiers'])
```

```
root
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
```

```
+---+-----+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+-----+
| 10|  0|          fax|          202-225-4160|
| 10|  1|          phone|          202-225-3436|
| 75|  0|          fax|          202-225-6791|
| 75|  1|          phone|          202-225-2861|
| 75|  2|        twitter|          RepSamFarr|
+---+-----+-----+-----+
```

rename_field

```
rename_field(oldName, newName, transformation_ctx="", info="",
stageThreshold=0, totalThreshold=0)
```

`DynamicFrame`에서 필드 이름을 바꾸고 바꾼 필드로 새로운 `DynamicFrame`을 반환합니다.

- `oldName` - 바꾸고자 하는 노드의 전체 경로입니다.

기존 이름에 점이 있으면 그 주위로 역음 부호(`)를 하지 않는 한 `RenameField`는 실행되지 않습니다. 예를 들어 `this.old.name`를 `thisNewName`으로 바꾸려면 다음과 같이 `rename_field`를 불러올 수 있습니다.

```
newDyF = oldDyF.rename_field("`this.old.name`", "thisNewName")
```

- `newName` - 전체 경로로써 새로운 이름입니다.
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- `totalThreshold` - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예: `rename_field`를 사용하여 `DynamicFrame`의 필드 이름을 변경합니다

이 코드 예제는 `rename_field` 메서드를 사용하여 `DynamicFrame`의 필드 이름을 변경합니다. 참고로 이 예제에서는 메서드 체인을 사용하여 여러 필드의 이름을 동시에 바꾸고 있습니다.

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: 데이터 조인 및 관계화](#) 항목을 참조하고 [1단계: Amazon S3 버킷에서 데이터 크롤](#)의 지침을 따르세요.

예제 코드

```
# Example: Use rename_field to rename fields
# in a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Inspect the original orgs schema
```

```

orgs = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="organizations_json"
)
print("Original orgs schema: ")
orgs.printSchema()

# Rename fields and view the new schema
orgs = orgs.rename_field("id", "org_id").rename_field("name", "org_name")
print("New orgs schema with renamed fields: ")
orgs.printSchema()

```

출력

```

Original orgs schema:
root
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- id: string
|-- classification: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- image: string
|-- seats: int
|-- type: string

New orgs schema with renamed fields:
root
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct

```



```

|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- classification: string
|-- org_id: string
|-- org_name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- image: string
|-- seats: int
|-- type: string

```

resolveChoice

```
resolveChoice(specs = None, choice = "" , database = None , table_name = None , transformation_ctx="", info="", stageThreshold=0, totalThreshold=0, catalog_id = None)
```

이 DynamicFrame로 선택 유형을 해결하고 새로운 DynamicFrame을 반환합니다.

- specs - 해결할 특정 모호성 목록이며, 각각은 (field_path, action)의 튜플 형식입니다.

resolveChoice를 사용하는 방법은 두 가지입니다. 첫 번째는 specs 인수를 사용하여 특정 필드의 시퀀스와 이를 확인하는 방법을 지정하는 것입니다. resolveChoice의 다른 모드는 choice 인수를 사용하여 모든 ChoiceTypes에 대해 단 하나의 해결책을 지정합니다.

specs의 값은 (field_path, action) 페어로 구성된 튜플로 지정됩니다. field_path 가치는 특정 모호한 요소를 확인하고 action 가치는 관련 해결 방안을 제안합니다. 가능한 작업은 다음과 같습니다.

- cast:*type* - 모든 값을 지정된 유형으로 캐스트하는 시도를 합니다. 예: cast:int.
- make_cols - 각 고유 유형을 *columnName_type*이라는 이름을 가진 열로 변환합니다. 데이터를 평면화하여 잠재적 모호성을 해결합니다. 예를 들어, columnA가 int 혹은 string이면 해결 방안은 DynamicFrame에서 columnA_int와 columnA_string으로 된 두 열을 생산하는 것입니다.
- make_struct - 데이터를 나타내도록 struct를 사용하여 잠재적 모호성을 해결합니다. 예를 들어, 열에서 데이터가 int 혹은 string이면 make_struct 작업은 결과적인 DynamicFrame의 구조 열을 생성합니다. 각 구조에는 int 및 string가 모두 포함됩니다.

- `project:type` - 모든 데이터를 가능한 데이터 유형 중 하나로 투영하여 잠재적인 모호성을 해결합니다. 예를 들어, 열에서 데이터가 `int` 혹은 `string`이면 `project:string` 작업을 사용하면 모든 `int` 값이 문자열로 변환된 결과인 `DynamicFrame`의 열을 생성합니다.

만약 `field_path`가 배열을 확인하면, 빈 대괄호를 배열 이름 다음에 만들어 모호성을 피합니다. 예를 들어, 다음과 같은 구조화된 데이터와 작업한다고 가정합니다.

```
"myList": [
  { "price": 100.00 },
  { "price": "$100.00" }
]
```

`field_path`를 `"myList[].price"`로 설정하고, `action`을 `"cast:double"`로 설정하여 가격의 문자열 버전보다 숫자 버전을 선택할 수 있습니다.

Note

`specs` 및 `choice` 파라미터 중 하나만 사용할 수 있습니다. `specs` 파라미터가 `None`이 아니면 `choice` 파라미터는 빈 문자열이어야 합니다. 반대로 `choice` 파라미터가 빈 문자열이 아니면, `specs` 파라미터는 `None`이어야 합니다.

- `choice`- 모든 `ChoiceTypes`에 대해 단일 해상도를 지정합니다. 실행 시간 전에 `ChoiceTypes`의 완전한 목록을 모르는 경우에 사용할 수 있습니다. 이 인수는 `specs`에 대해 앞에서 나열한 작업 외에 다음 작업을 지원합니다.
 - `match_catalog` - 각 `ChoiceType`을 지정된 Data Catalog 테이블의 해당 유형에 캐스팅해봅니다.
 - `database` - `match_catalog` 작업에 사용할 Data Catalog 데이터베이스입니다.
 - `table_name` - `match_catalog` 작업에 사용할 Data Catalog 테이블입니다.
 - `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
 - `info` - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
 - `stageThreshold` - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
 - `totalThreshold` - 오류가 발생하는 변환을 포함하여 최대 발생하는 오류 수입니다(선택). 기본 값은 0이고 이는 프로세스에서 오류가 발생되지 않음을 나타냅니다.
 - `catalog_id` - 액세스 중인 Data Catalog의 카탈로그 ID(Data Catalog의 계정 ID)입니다. `None`(기본값)으로 설정하면 호출 계정의 카탈로그 ID를 사용합니다.

예: `resolveChoice`를 사용하여 여러 유형이 포함된 열을 처리합니다

이 코드 예제는 `resolveChoice` 메서드를 사용하여 여러 유형의 값이 포함된 `DynamicFrame` 열을 처리하는 방법을 지정합니다. 이 예제는 유형이 다른 열을 처리하는 두 가지 일반적인 방법을 보여줍니다.

- 열을 단일 데이터 유형으로 보내버립니다.
- 모든 유형을 별도의 열에 보관합니다.

예제 데이터 세트

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: ResolveChoice, Lambda, 및 ApplyMapping을 사용한 데이터 준비](#) 항목을 참조하고 [1단계: Amazon S3 버킷에서 데이터 크롤의 지침을 따르세요.](#)

이 예제에서는 다음 스키마와 함께 `medicare`를 호출한 `DynamicFrame`을 사용합니다.

```
root
|-- drg definition: string
|-- provider id: choice
|   |-- long
|   |-- string
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string
```

예제 코드

```
# Example: Use resolveChoice to handle
# a column that contains multiple types
```

```

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load the input data and inspect the "provider id" column
medicare = glueContext.create_dynamic_frame.from_catalog(
    database="payments", table_name="medicare_hospital_provider_csv"
)
print("Inspect the provider id column:")
medicare.toDF().select("provider id").show()

# Cast provider id to type long
medicare_resolved_long = medicare.resolveChoice(specs=[("provider id", "cast:long")])
print("Schema after casting provider id to type long:")
medicare_resolved_long.printSchema()
medicare_resolved_long.toDF().select("provider id").show()

# Create separate columns
# for each provider id type
medicare_resolved_cols = medicare.resolveChoice(choice="make_cols")
print("Schema after creating separate columns for each type:")
medicare_resolved_cols.printSchema()
medicare_resolved_cols.toDF().select("provider id_long", "provider id_string").show()

```

출력

```

Inspect the 'provider id' column:
+-----+
|provider id|
+-----+
| [10001,]|
| [10005,]|
| [10006,]|
| [10011,]|
| [10016,]|
| [10023,]|
| [10029,]|
| [10033,]|
| [10039,]|

```

```
| [10040,]|
| [10046,]|
| [10055,]|
| [10056,]|
| [10078,]|
| [10083,]|
| [10085,]|
| [10090,]|
| [10092,]|
| [10100,]|
| [10103,]|
```

```
+-----+
```

only showing top 20 rows

Schema after casting 'provider id' to type long:

root

```
|-- drg definition: string
|-- provider id: long
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string
```

```
+-----+
```

```
|provider id|
```

```
+-----+
```

```
| 10001|
| 10005|
| 10006|
| 10011|
| 10016|
| 10023|
| 10029|
| 10033|
| 10039|
| 10040|
| 10046|
| 10055|
```

```
|      10056|
|      10078|
|      10083|
|      10085|
|      10090|
|      10092|
|      10100|
|      10103|
```

```
+-----+
```

only showing top 20 rows

Schema after creating separate columns for each type:

root

```
|-- drg definition: string
|-- provider id_string: string
|-- provider id_long: long
|-- provider name: string
|-- provider street address: string
|-- provider city: string
|-- provider state: string
|-- provider zip code: long
|-- hospital referral region description: string
|-- total discharges: long
|-- average covered charges: string
|-- average total payments: string
|-- average medicare payments: string
```

```
+-----+
```

```
|provider id_long|provider id_string|
```

```
+-----+
```

```
|      10001|          null|
|      10005|          null|
|      10006|          null|
|      10011|          null|
|      10016|          null|
|      10023|          null|
|      10029|          null|
|      10033|          null|
|      10039|          null|
|      10040|          null|
|      10046|          null|
|      10055|          null|
|      10056|          null|
|      10078|          null|
```

```
|          10083|          null|
|          10085|          null|
|          10090|          null|
|          10092|          null|
|          10100|          null|
|          10103|          null|
+-----+-----+
```

only showing top 20 rows

select_fields

select_fields(paths, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

선택된 필드를 포함한 새로운 DynamicFrame을 반환합니다.

- paths - 문자열 목록입니다. 각 문자열은 선택하려는 최상위 노드에 대한 경로입니다.
- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- stageThreshold - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- totalThreshold - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예: select_fields를 사용하여 선택한 필드로 새 **DynamicFrame** 생성

다음 코드 예제에서는 select_fields 메서드를 사용하여 기존 DynamicFrame에서 선택한 필드 목록으로 새 DynamicFrame을 만드는 방법을 보여줍니다.

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: 데이터 조인 및 관계화](#) 항목을 참조하고 [1단계: Amazon S3 버킷에서 데이터 크롤](#)의 지침을 따르세요.

```
# Example: Use select_fields to select specific fields from a DynamicFrame
```

```
from pyspark.context import SparkContext
```

```

from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create a DynamicFrame and view its schema
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Create a new DynamicFrame with chosen fields
names = persons.select_fields(paths=["family_name", "given_name"])
print("Schema for the names DynamicFrame, created with select_fields:")
names.printSchema()
names.toDF().show()

```

출력

```

Schema for the persons DynamicFrame:
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string
|-- images: array
|   |-- element: struct

```



```

|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string

```

Schema for the names DynamicFrame:

```
root
```

```

|-- family_name: string
|-- given_name: string

```

```
+-----+-----+
```

```
|family_name|given_name|
```

```
+-----+-----+
```

```
|   Collins|   Michael|
```

```
| Huizenga|     Bill|
```

```
|  Clawson|   Curtis|
```

```
| Solomon|   Gerald|
```

```
|  Rigell|   Edward|
```

```
|   Crapo| Michael|
```

```
|   Hutto|    Earl|
```

```
|   Ertel|   Allen|
```

```
| Minish|   Joseph|
```

```
| Andrews|  Robert|
```

```
| Walden|    Greg|
```

```
|   Kazen| Abraham|
```

```
| Turner|  Michael|
```

```
|  Kolbe|    James|
```

```
|Lowenthal|   Alan|
```

```
| Capuano| Michael|
```

```
| Schrader|   Kurt|
```

```
|  Nadler| Jerrold|
```

```
| Graves|    Tom|
```

```
| McMillan|   John|
```

```
+-----+-----+
```

```
only showing top 20 rows
```

simplify_ddb_json

simplify_ddb_json(): DynamicFrame

DynamicFrame의 중첩된 열 중 특히 DynamoDB JSON 구조에 있는 열을 단순화하고 단순화된 새로운 DynamicFrame을 반환합니다. 목록 유형에 여러 유형 또는 맵 유형이 있는 경우 목록의 요소는 단순화되지 않습니다. 이 변환은 일반적인 unnest 변환과 다르게 작동하는 특정 유형의 변환이며 데이터가 이미 DynamoDB JSON 구조에 있어야 한다는 점에 유의하세요. 자세한 내용은 [DynamoDB JSON](#)을 참조하세요.

예를 들어, DynamoDB JSON 구조를 사용하여 내보내기를 읽는 스키마는 다음과 같을 수 있습니다.

```

root
|-- Item: struct
|   |-- parentMap: struct
|   |   |-- M: struct
|   |   |   |-- childMap: struct
|   |   |   |   |-- M: struct
|   |   |   |   |   |-- appName: struct
|   |   |   |   |   |   |-- S: string
|   |   |   |   |   |   |-- packageName: struct
|   |   |   |   |   |   |   |-- S: string
|   |   |   |   |   |   |   |-- updatedAt: struct
|   |   |   |   |   |   |   |   |-- N: string
|   |   |-- strings: struct
|   |   |   |-- SS: array
|   |   |   |   |-- element: string
|   |   |-- numbers: struct
|   |   |   |-- NS: array
|   |   |   |   |-- element: string
|   |   |-- binaries: struct
|   |   |   |-- BS: array
|   |   |   |   |-- element: string
|   |-- isDDBJson: struct
|   |   |-- BOOL: boolean
|   |-- nullValue: struct
|   |   |-- NULL: boolean

```

simplify_ddb_json() 변환은 이 구조를 다음과 같이 변환합니다.

```

root
|-- parentMap: struct
|   |-- childMap: struct

```

```

|   |   |-- appName: string
|   |   |-- packageName: string
|   |   |-- updatedAt: string
|-- strings: array
|   |-- element: string
|-- numbers: array
|   |-- element: string
|-- binaries: array
|   |-- element: string
|-- isDDBJson: boolean
|-- nullValue: null

```

예제: `simplify_ddb_json`을 사용하여 DynamoDB JSON 단순화 호출

이 코드 예제에서는 `simplify_ddb_json` 메서드를 사용하여 AWS Glue DynamoDB 내보내기 커넥터를 사용하고, DynamoDB JSON 단순화를 호출하고, 파티션 수를 인쇄합니다.

예제 코드

```

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type = "dynamodb",
    connection_options = {
        'dynamodb.export': 'ddb',
        'dynamodb.tableArn': '<table arn>',
        'dynamodb.s3.bucket': '<bucket name>',
        'dynamodb.s3.prefix': '<bucket prefix>',
        'dynamodb.s3.bucketOwner': '<account_id of bucket>'
    }
)
simplified = dynamicFrame.simplify_ddb_json()
print(simplified.getNumPartitions())

```

spigot

spigot(path, options={})

작업에 의해 수행된 변환을 확인하는 데 도움이 되도록 샘플 레코드를 지정된 대상에 기록합니다.

- `path` - 작성할 기본 대상 주소 경로입니다(필수).
- `options` - 옵션을 지정하는 키-값 페어입니다(선택 사항). "topk" 옵션은 첫 번째 k 기록이 작성되어야 한다는 것을 명시합니다. "prob" 옵션은 특정 레코드를 선택할 가능성(십진수)을 지정합니다. 작성할 레코드를 선택할 때 사용할 수 있습니다.
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).

예: spigot을 사용하여 **DynamicFrame**에서 Amazon S3에 샘플 필드를 작성합니다

이 코드 예제는 `select_fields` 변환을 적용한 후 spigot 메서드를 사용하여 Amazon S3 버킷에 샘플 레코드를 작성합니다.

예제 데이터 세트

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: 데이터 조인 및 관계화](#) 항목을 참조하고 [1단계: Amazon S3 버킷에서 데이터 크롤](#)의 지침을 따르세요.

이 예제에서는 다음 스키마와 함께 `persons`를 호출한 `DynamicFrame`을 사용합니다.

```
root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string
|-- images: array
```

```

|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string

```

예제 코드

```

# Example: Use spigot to write sample records
# to a destination during a transformation
# from pyspark.context import SparkContext.
# Replace DOC-EXAMPLE-BUCKET with your own location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load table data into a DynamicFrame
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)

# Perform the select_fields on the DynamicFrame
persons = persons.select_fields(paths=["family_name", "given_name", "birth_date"])

# Use spigot to write a sample of the transformed data
# (the first 10 records)
spigot_output = persons.spigot(
    path="s3://DOC-EXAMPLE-BUCKET", options={"topk": 10}
)

```

출력

다음은 spigot이 Amazon S3에 작성한 데이터 예제입니다. 예제 코드가 options={"topk": 10}를 지정하였으므로 샘플 데이터에는 처음 10개의 레코드가 포함됩니다.

```

{"family_name":"Collins","given_name":"Michael","birth_date":"1944-10-15"}
{"family_name":"Huizenga","given_name":"Bill","birth_date":"1969-01-31"}
{"family_name":"Clawson","given_name":"Curtis","birth_date":"1959-09-28"}
{"family_name":"Solomon","given_name":"Gerald","birth_date":"1930-08-14"}
{"family_name":"Rigell","given_name":"Edward","birth_date":"1960-05-28"}
{"family_name":"Crapo","given_name":"Michael","birth_date":"1951-05-20"}
{"family_name":"Hutto","given_name":"Earl","birth_date":"1926-05-12"}
{"family_name":"Ertel","given_name":"Allen","birth_date":"1937-11-07"}
{"family_name":"Minish","given_name":"Joseph","birth_date":"1916-09-01"}
{"family_name":"Andrews","given_name":"Robert","birth_date":"1957-08-04"}

```

split_fields

split_fields(paths, name1, name2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

두 개의 DynamicFrames이 포함된 새 DynamicFrameCollection을 반환합니다. 첫 번째 DynamicFrame은 분할된 모든 노드를 포함하고 두 번째는 남겨진 노드를 포함합니다.

- paths - 문자열 목록이며, 각 문자열 목록은 새로운 DynamicFrame으로 스플릿하려는 노드의 전체 경로입니다.
- name1 - 스플릿된 DynamicFrame의 이름 문자열입니다.
- name2 - 스플릿된 특정 노드 이후에 남겨진 DynamicFrame을 위한 이름 문자열입니다.
- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- stageThreshold - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- totalThreshold - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예제: split_fields를 사용하여 선택한 필드를 별도의 **DynamicFrame**로 분할합니다

이 코드 예제는 split_fields 메서드를 사용하여 지정된 필드 목록을 별도의 DynamicFrame으로 분할합니다.

예제 데이터 세트

이 예제에서는 legislators_relationalized라는 컬렉션의 l_root_contact_details를 호출한 DynamicFrame을 사용합니다.

l_root_contact_details에는 다음 스키마와 엔트리가 포함됩니다.

```
root
|-- id: long
|-- index: int
|-- contact_details.val.type: string
|-- contact_details.val.value: string
```

id	index	contact_details.val.type	contact_details.val.value
1	0	phone	202-225-5265
1	1	twitter	kathyhochul
2	0	phone	202-225-3252
2	1	twitter	repjackyrosen
3	0	fax	202-225-1314
3	1	phone	202-225-3772
...			

예제 코드

```
# Example: Use split_fields to split selected
# fields into a separate DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Load the input DynamicFrame and inspect its schema
frame_to_split = legislators_relationalized.select("l_root_contact_details")
print("Inspect the input DynamicFrame schema:")
frame_to_split.printSchema()

# Split id and index fields into a separate DynamicFrame
split_fields_collection = frame_to_split.split_fields(["id", "index"], "left", "right")

# Inspect the resulting DynamicFrames
```

```
print("Inspect the schemas of the DynamicFrames created with split_fields:")
split_fields_collection.select("left").printSchema()
split_fields_collection.select("right").printSchema()
```

출력

Inspect the input DynamicFrame's schema:

```
root
|-- id: long
|-- index: int
|-- contact_details.val.type: string
|-- contact_details.val.value: string
```

Inspect the schemas of the DynamicFrames created with split_fields:

```
root
|-- id: long
|-- index: int
```

```
root
|-- contact_details.val.type: string
|-- contact_details.val.value: string
```

split_rows

split_rows(comparison_dict, name1, name2, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

새로운 DynamicFrame으로 DynamicFrame의 하나 이상의 열을 스플릿합니다.

이 메서드는 2개의 DynamicFrames가 포함된 새 DynamicFrameCollection을 반환합니다. 첫 번째 DynamicFrame은 스플릿된 모든 행을 포함하고 두 번째는 남겨진 열을 포함합니다.

- **comparison_dict** - 열까지 경로의 키와 열 값이 비교된 값의 매핑 비교기를 위한 다른 딕셔너리인 값의 딕셔너리입니다. 예를 들어, {"age": {">": 10, "<": 20}}는 나이 열에서 가치가 10 초과 20 미만인 모든 열을 스플릿합니다.
- **name1** - 스플릿된 DynamicFrame의 이름 문자열입니다.
- **name2** - 스플릿된 특정 노드 이후에 남겨진 DynamicFrame을 위한 이름 문자열입니다.
- **transformation_ctx** - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- **info** - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).

- `stageThreshold` - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- `totalThreshold` - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예제: `split_rows`를 사용하여 **DynamicFrame** 행을 분할합니다

이 코드 예제는 `split_rows` 메서드를 사용하여 `id` 필드 값을 기준으로 `DynamicFrame`에서 행을 분할합니다.

예제 데이터 세트

이 예제에서는 `legislators_relationalized`라는 컬렉션에서 선택한 `l_root_contact_details`를 호출한 `DynamicFrame`을 사용합니다.

`l_root_contact_details`에는 다음 스키마와 엔트리가 포함됩니다.

```
root
|-- id: long
|-- index: int
|-- contact_details.val.type: string
|-- contact_details.val.value: string

+---+-----+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+-----+-----+
| 1|  0|           phone|      202-225-5265|
| 1|  1|        twitter|      kathyhochul|
| 2|  0|           phone|      202-225-3252|
| 2|  1|        twitter|      repjackyroser|
| 3|  0|            fax|      202-225-1314|
| 3|  1|           phone|      202-225-3772|
| 3|  2|        twitter|      MikeRossUpdates|
| 4|  0|            fax|      202-225-1314|
| 4|  1|           phone|      202-225-3772|
| 4|  2|        twitter|      MikeRossUpdates|
| 5|  0|            fax|      202-225-1314|
| 5|  1|           phone|      202-225-3772|
| 5|  2|        twitter|      MikeRossUpdates|
| 6|  0|            fax|      202-225-1314|
| 6|  1|           phone|      202-225-3772|
| 6|  2|        twitter|      MikeRossUpdates|
```

```

| 7| 0|          fax|          202-225-1314|
| 7| 1|          phone|          202-225-3772|
| 7| 2|        twitter|        MikeRossUpdates|
| 8| 0|          fax|          202-225-1314|
+---+-----+-----+-----+

```

예제 코드

```

# Example: Use split_rows to split up
# rows in a DynamicFrame based on value

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Retrieve the DynamicFrame to split
frame_to_split = legislators_relationalized.select("l_root_contact_details")

# Split up rows by ID
split_rows_collection = frame_to_split.split_rows({"id": {">": 10}}, "high", "low")

# Inspect the resulting DynamicFrames
print("Inspect the DynamicFrame that contains IDs < 10")
split_rows_collection.select("low").toDF().show()
print("Inspect the DynamicFrame that contains IDs > 10")
split_rows_collection.select("high").toDF().show()

```

출력

```

Inspect the DynamicFrame that contains IDs < 10
+---+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+-----+
| 1| 0|          phone|          202-225-5265|
| 1| 1|        twitter|        kathyhochul|
| 2| 0|          phone|          202-225-3252|
| 2| 1|        twitter|        repjackyroten|
| 3| 0|          fax|          202-225-1314|
| 3| 1|          phone|          202-225-3772|
| 3| 2|        twitter|        MikeRossUpdates|

```

```

| 4| 0| fax| 202-225-1314|
| 4| 1| phone| 202-225-3772|
| 4| 2| twitter| MikeRossUpdates|
| 5| 0| fax| 202-225-1314|
| 5| 1| phone| 202-225-3772|
| 5| 2| twitter| MikeRossUpdates|
| 6| 0| fax| 202-225-1314|
| 6| 1| phone| 202-225-3772|
| 6| 2| twitter| MikeRossUpdates|
| 7| 0| fax| 202-225-1314|
| 7| 1| phone| 202-225-3772|
| 7| 2| twitter| MikeRossUpdates|
| 8| 0| fax| 202-225-1314|

```

```

+-----+
only showing top 20 rows

```

Inspect the DynamicFrame that contains IDs > 10

```

+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+-----+
| 11| 0| phone| 202-225-5476|
| 11| 1| twitter| RepDavidYoung|
| 12| 0| phone| 202-225-4035|
| 12| 1| twitter| RepStephMurphy|
| 13| 0| fax| 202-226-0774|
| 13| 1| phone| 202-225-6335|
| 14| 0| fax| 202-226-0774|
| 14| 1| phone| 202-225-6335|
| 15| 0| fax| 202-226-0774|
| 15| 1| phone| 202-225-6335|
| 16| 0| fax| 202-226-0774|
| 16| 1| phone| 202-225-6335|
| 17| 0| fax| 202-226-0774|
| 17| 1| phone| 202-225-6335|
| 18| 0| fax| 202-226-0774|
| 18| 1| phone| 202-225-6335|
| 19| 0| fax| 202-226-0774|
| 19| 1| phone| 202-225-6335|
| 20| 0| fax| 202-226-0774|
| 20| 1| phone| 202-225-6335|

```

```

+-----+
only showing top 20 rows

```

unbox

unbox(path, format, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0, **options)

DynamicFrame의 문자열 필드를 열고(포맷 재지정) 개봉된 DynamicRecords를 포함한 새로운 DynamicFrame를 반환합니다.

DynamicRecord는 DynamicFrame내 논리적 기록입니다. 자기 설명적이고 고정 스키마를 준수하지 않은 데이터에 사용될 수 있다는 점을 제외하면 Apache Spark DataFrame의 열과 비슷합니다.

- path - 열고자 하는 문자열 노드의 전체 경로입니다.
- format - 포맷 사양입니다(선택 사항). 여러 포맷을 지원하는 Amazon S3 또는 AWS Glue 연결에 사용할 수 있습니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하세요.
- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- stageThreshold - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- totalThreshold - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- options - 다음 중 한 개 이상을 수행할 수 있습니다.
 - separator - 구분자 문자를 포함한 문자열입니다.
 - escaper - 이스케이프 문자를 포함한 문자열입니다.
 - skipFirst - 첫 번째 인스턴스를 넘어갈지 여부를 나타내는 부울 값입니다.
 - withSchema - 노드 스키마의 JSON 표현을 포함하는 문자열입니다. 스키마의 JSON 표현 형식은 StructType.json()의 출력에 의해 정의됩니다.
 - withHeader - 헤더가 포함되었는지 여부를 나타내는 부울 값입니다.

예제: 개봉하기를 사용하여 문자열 필드를 구조에 개봉하기

이 코드 예제는 unbox 메서드를 사용하여 DynamicFrame의 문자열 필드를 유형 구조 필드로 개봉하거나 포맷을 다시 지정합니다.

예제 데이터 세트

이 예제에서는 다음과 같은 스키마 및 항목과 함께 `mapped_with_string`를 호출한 `DynamicFrame`을 사용합니다.

`AddressString`로 이름이 지정된 필드를 확인하십시오. 이 필드는 예제에서 구조로 개봉되는 필드입니다.

```

root
|-- Average Total Payments: string
|-- AddressString: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|   |-- Zip.Code: string
|   |-- City: string
|   |-- Array: array
|   |   |-- element: string
|   |-- State: string
|   |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string

+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|Average Total Payments|      AddressString|Average Covered Charges|      DRG
|Definition|Average Medicare Payments|Hospital Referral Region Description|
|Address|Provider Id|Total Discharges|      Provider Name|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|          $5777.24|{"Street": "1108 ...|          $32963.07|039 -
EXTRACRANIA...|          $4763.73|          AL - Dothan|[36301,
DOTHAN, [...|          10001|          91|SOUTHEAST ALABAMA...|
|          $5787.57|{"Street": "2505 ...|          $15131.85|039 -
EXTRACRANIA...|          $4976.71|          AL - Birmingham|[35957,
BOAZ, [25...|          10005|          14|MARSHALL MEDICAL ...|
|          $5434.95|{"Street": "205 M...|          $37560.37|039 -
EXTRACRANIA...|          $4453.79|          AL - Birmingham|[35631,
FLORENCE,...|          10006|          24|ELIZA COFFEE MEMO...|

```

```
|          $5417.56|{"Street": "50 ME...|          $13998.28|039 -
EXTRACRANIA...|          $4129.16|          AL - Birmingham|[35235,
BIRMINGHA...|          10011|          25| ST VINCENT'S EAST|
...
```

예제 코드

```
# Example: Use unbox to unbox a string field
# into a struct in a DynamicFrame

from pyspark.context import SparkContext
from aws glue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

unboxed = mapped_with_string.unbox("AddressString", "json")
unboxed.printSchema()
unboxed.toDF().show()
```

출력

```
root
|-- Average Total Payments: string
|-- AddressString: struct
|   |-- Street: string
|   |-- City: string
|   |-- State: string
|   |-- Zip.Code: string
|   |-- Array: array
|   |   |-- element: string
|-- Average Covered Charges: string
|-- DRG Definition: string
|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|   |-- Zip.Code: string
|   |-- City: string
|   |-- Array: array
|   |   |-- element: string
|   |-- State: string
```

```

|-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string

+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
|Average Total Payments|      AddressString|Average Covered Charges|      DRG
|Definition|Average Medicare Payments|Hospital Referral Region Description|
|Address|Provider Id|Total Discharges|      Provider Name|
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
|          $5777.24|[1108 ROSS CLARK ...|          $32963.07|039 -
EXTRACRANIA...|          $4763.73|          AL - Dothan|[36301,
DOTHAN, [...]          10001|          91|SOUTHEAST ALABAMA...|
|          $5787.57|[2505 U S HIGHWAY...|          $15131.85|039 -
EXTRACRANIA...|          $4976.71|          AL - Birmingham|[35957,
BOAZ, [25...|          10005|          14|MARSHALL MEDICAL ...|
|          $5434.95|[205 MARENGO STRE...|          $37560.37|039 -
EXTRACRANIA...|          $4453.79|          AL - Birmingham|[35631,
FLORENCE,...|          10006|          24|ELIZA COFFEE MEMO...|
|          $5417.56|[50 MEDICAL PARK ...|          $13998.28|039 -
EXTRACRANIA...|          $4129.16|          AL - Birmingham|[35235,
BIRMINGHA...|          10011|          25|  ST VINCENT'S EAST|
|          $5658.33|[1000 FIRST STREE...|          $31633.27|039 -
EXTRACRANIA...|          $4851.44|          AL - Birmingham|[35007,
ALABASTER...|          10016|          18|SHELBY BAPTIST ME...|
|          $6653.80|[2105 EAST SOUTH ...|          $16920.79|039 -
EXTRACRANIA...|          $5374.14|          AL - Montgomery|[36116,
MONTGOMER...|          10023|          67|BAPTIST MEDICAL C...|
|          $5834.74|[2000 PEPPERELL P...|          $11977.13|039 -
EXTRACRANIA...|          $4761.41|          AL - Birmingham|[36801,
OPELIKA, ...|          10029|          51|EAST ALABAMA MEDI...|
|          $8031.12|[619 SOUTH 19TH S...|          $35841.09|039 -
EXTRACRANIA...|          $5858.50|          AL - Birmingham|[35233,
BIRMINGHA...|          10033|          32|UNIVERSITY OF ALA...|
|          $6113.38|[101 SIVLEY RD, H...|          $28523.39|039 -
EXTRACRANIA...|          $5228.40|          AL - Huntsville|[35801,
HUNTSVILL...|          10039|          135| HUNTSVILLE HOSPITAL|
|          $5541.05|[1007 GOODYEAR AV...|          $75233.38|039 -
EXTRACRANIA...|          $4386.94|          AL - Birmingham|[35903,
GADSDEN, ...|          10040|          34|GADSDEN REGIONAL ...|

```

```

|          $5461.57|[600 SOUTH THIRD ...|          $67327.92|039 -
EXTRACRANIA...|          $4493.57|          AL - Birmingham|[35901,
GADSDEN, ...|          10046|          14|RIVERVIEW REGIONA...|
|          $5356.28|[4370 WEST MAIN S...|          $39607.28|039 -
EXTRACRANIA...|          $4408.20|          AL - Dothan|[36305,
DOTHAN, [...|          10055|          45| FLOWERS HOSPITAL|
|          $5374.65|[810 ST VINCENT'S...|          $22862.23|039 -
EXTRACRANIA...|          $4186.02|          AL - Birmingham|[35205,
BIRMINGHA...|          10056|          43|ST VINCENT'S BIRM...|
|          $5366.23|[400 EAST 10TH ST...|          $31110.85|039 -
EXTRACRANIA...|          $4376.23|          AL - Birmingham|[36207,
ANNISTON,...|          10078|          21|NORTHEAST ALABAMA...|
|          $5282.93|[1613 NORTH MCKEN...|          $25411.33|039 -
EXTRACRANIA...|          $4383.73|          AL - Mobile|[36535,
FOLEY, [1...|          10083|          15|SOUTH BALDWIN REG...|
|          $5676.55|[1201 7TH STREET ...|          $9234.51|039 -
EXTRACRANIA...|          $4509.11|          AL - Huntsville|[35609,
DECATUR, ...|          10085|          27|DECATUR GENERAL H...|
|          $5930.11|[6801 AIRPORT BOU...|          $15895.85|039 -
EXTRACRANIA...|          $3972.85|          AL - Mobile|[36608,
MOBILE, [...|          10090|          27| PROVIDENCE HOSPITAL|
|          $6192.54|[809 UNIVERSITY B...|          $19721.16|039 -
EXTRACRANIA...|          $5179.38|          AL - Tuscaloosa|[35401,
TUSCALOOS...|          10092|          31|D C H REGIONAL ME...|
|          $4968.00|[750 MORPHY AVENU...|          $10710.88|039 -
EXTRACRANIA...|          $3898.88|          AL - Mobile|[36532,
FAIRHOPE,...|          10100|          18| THOMAS HOSPITAL|
|          $5996.00|[701 PRINCETON AV...|          $51343.75|039 -
EXTRACRANIA...|          $4962.45|          AL - Birmingham|[35211,
BIRMINGHA...|          10103|          33|BAPTIST MEDICAL C...|

```

```

+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+

```

only showing top 20 rows

결합

```
union(frame1, frame2, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)
```

두 개의 DynamicFrame을 결합합니다. 두 입력 DynamicFrames의 모든 레코드를 포함하는 DynamicFrame을 반환합니다. 이 변환은 두 DataFrame을 동일한 데이터와 결합하여 다른 결과를 반

환할 수 있습니다. Spark DataFrame 통합 동작이 필요한 경우 toDF을(를) 사용하는 것을 고려해 보세요.

- `frame1`— 결합할 첫 번째 DynamicFrame.
- `frame2`— 결합할 두 번째 DynamicFrame.
- `transformation_ctx` - (선택 사항) 통계/상태 정보를 확인하는 데 사용되는 고유 문자열
- `info` - (선택 사항) 변환에 따른 오류 관련 문자열
- `stageThreshold`— (선택 사항) 처리 오류가 발생할 때까지 변환에서 발생한 최대 오류 수
- `totalThreshold`— (선택 사항) 처리 오류가 발생할 때까지의 총 최대 오류 수.

`unnest`

`unnest(transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)`

DynamicFrame에서 중첩된 객체를 중첩시키지 않고 상위 객체로 만들어서 새로운 중첩되지 않은 DynamicFrame을 반환합니다.

- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하는 변환 중에 발생하는 오류 수(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.
- `totalThreshold` - 프로세스에서 오류가 발생해야 하는 이 변환을 포함하여 최대 발생한 오류 수입니다(선택 사항). 기본값은 0이며, 이는 프로세스에 오류가 발생하지 않아야 함을 나타냅니다.

예제: 중첩 해제를 사용하여 중첩된 필드를 최상위 필드로 전환합니다

이 코드 예제는 `unnest` 메서드를 사용하여 DynamicFrame의 모든 중첩된 필드를 최상위 필드로 병합합니다.

예제 데이터 세트

이 예제에서는 다음 스키마와 함께 `mapped_medicare`를 호출하는 DynamicFrame을 사용합니다. `Address` 필드는 중첩된 데이터를 포함하는 유일한 필드라는 점에 유의하십시오.

```
root
|-- Average Total Payments: string
|-- Average Covered Charges: string
|-- DRG Definition: string
```

```

|-- Average Medicare Payments: string
|-- Hospital Referral Region Description: string
|-- Address: struct
|   |-- Zip.Code: string
|   |-- City: string
|   |-- Array: array
|   |   |-- element: string
|   |-- State: string
|   |-- Street: string
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string

```

예제 코드

```

# Example: Use unnest to unnest nested
# objects in a DynamicFrame

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Unnest all nested fields
unnested = mapped_medicare.unnest()
unnested.printSchema()

```

출력

```

root
 |-- Average Total Payments: string
 |-- Average Covered Charges: string
 |-- DRG Definition: string
 |-- Average Medicare Payments: string
 |-- Hospital Referral Region Description: string
 |-- Address.Zip.Code: string
 |-- Address.City: string
 |-- Address.Array: array
 |   |-- element: string
 |-- Address.State: string
 |-- Address.Street: string

```

```
|-- Provider Id: string
|-- Total Discharges: string
|-- Provider Name: string
```

unnest_ddb_json

DynamoDB JSON 구조의 DynamicFrame에 있는 중첩된 열을 한정해서 중첩 해제하고, 중첩되지 않은 새 DynamicFrame을 반환합니다. 구조체 유형 배열인 열은 중첩 해제되지 않습니다. 이 변환은 일반적인 unnest 변환과는 다르게 작동하는 특정 유형의 중첩 해제 변환이며, 데이터가 이미 DynamoDB JSON 구조에 있어야 합니다. 자세한 내용은 [DynamoDB JSON](#)을 참조하세요.

unnest_ddb_json(transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)

- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 이 변환에 따른 오류 보고 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하는 변환 중에 발생하는 오류 수 (선택: 오류가 발생되지 않은 변환을 나타내는 기본값은 0입니다).
- `totalThreshold` - 오류가 발생하는 변환을 포함하여 최대 발생하는 오류 수 (선택: 오류가 발생되지 않은 변환을 나타내는 기본값은 0입니다).

예를 들어, DynamoDB JSON 구조를 사용하여 내보내기를 읽는 스키마는 다음과 같을 수 있습니다.

```
root
|-- Item: struct
|   |-- ColA: struct
|   |   |-- S: string
|   |-- ColB: struct
|   |   |-- S: string
|   |-- ColC: struct
|   |   |-- N: string
|   |-- ColD: struct
|   |   |-- L: array
|   |   |   |-- element: null
```

`unnest_ddb_json()` 변환은 이 구조를 다음과 같이 변환합니다.

```
root
|-- ColA: string
```

```
|-- ColB: string
|-- ColC: string
|-- ColD: array
|   |-- element: null
```

아래 코드 예제에서는 AWS Glue DynamoDB 내보내기 커넥터를 사용하고, DynamoDB JSON unnest 를 호출하고, 파티션 수를 인쇄하는 방법을 보여줍니다.

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dynamicFrame = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source>",
        "dynamodb.s3.bucket": "<bucket name>",
        "dynamodb.s3.prefix": "<bucket prefix>",
        "dynamodb.s3.bucketOwner": "<account_id>",
    }
)
unnested = dynamicFrame.unnest_ddb_json()
print(unnested.getNumPartitions())

job.commit()
```

write

write(connection_type, connection_options, format, format_options, accumulator_size)

이 DynamicFrame의 [GlueContext](#) 수업에서 지정된 연결 유형의 [DataSink\(객체\)](#)를 얻어 DynamicFrame 내용을 포맷하고 작성하는 데 사용합니다. 명시된 대로 포맷되고 작성된 새로운 DynamicFrame을 반환합니다.

- `connection_type` - 사용할 연결 유형입니다. 유효한 값에는 `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver` 및 `oracle`가 있습니다.
- `connection_options` - 사용할 연결 옵션입니다(선택 사항). `s3`의 `connection_type`의 경우, Amazon S3 경로가 정의됩니다.

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

JDBC 연결의 경우, 몇 가지 속성이 정의되어야 합니다. 단, 데이터베이스 이름이 URL의 일부여야 합니다. 연결 옵션에 선택적으로 포함될 수 있습니다.

⚠ Warning

스크립트에 암호를 저장하는 것은 권장되지 않습니다. AWS Secrets Manager 또는 AWS Glue 데이터 카탈로그에서 데이터를 검색할 때 `boto3` 사용을 고려합니다.

```
connection_options = {"url": "jdbc-url/database", "user": "username",
  "password": passwordVariable, "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

- `format` - 포맷 사양입니다(선택 사항). 여러 포맷을 지원하는 Amazon Simple Storage Service(Amazon S3) 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- `format_options` - 지정된 포맷에 대한 포맷 옵션입니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- `accumulator_size` - (선택 사항) 사용할 누적 가능한 크기(바이트)입니다.

- 오류 -

- [assertErrorThreshold](#)
- [errorsAsDynamicFrame](#)
- [errorsCount](#)
- [stageErrorsCount](#)

assertErrorThreshold

`assertErrorThreshold()` - 이 `DynamicFrame`을 생성한 변환의 오류에 대한 어설션입니다. 기본 `DataFrame`으로부터 `Exception`을 반환합니다.

errorsAsDynamicFrame

`errorsAsDynamicFrame()` - 내부에 중첩된 오류 기록을 보유한 `DynamicFrame`을 반환합니다.

예: `errorsAsDynamicFrame`을 사용하여 오류 레코드 보기

다음 코드 예제에서는 `errorsAsDynamicFrame` 메서드를 사용하여 `DynamicFrame`에 대한 오류 레코드를 보는 방법을 보여 줍니다.

예제 데이터 세트

이 예제에서는 Amazon S3에 JSON으로 업로드할 수 있는 다음 데이터 세트를 사용합니다. 두 번째 레코드의 형식이 잘못되었습니다. 잘못된 형식의 데이터는 일반적으로 SparkSQL을 사용할 때 파일 구문 분석을 중단합니다. 그러나 `DynamicFrame`은 잘못된 형식 문제를 인식하고 잘못된 형식 행을 개별적으로 처리할 수 있는 오류 레코드로 바꿉니다.

```
{"id": 1, "name": "george", "surname": "washington", "height": 178}
{"id": 2, "name": "benjamin", "surname": "franklin",
{"id": 3, "name": "alexander", "surname": "hamilton", "height": 171}
{"id": 4, "name": "john", "surname": "jay", "height": 190}
```

예제 코드

```
# Example: Use errorsAsDynamicFrame to view error records.
# Replace s3://DOC-EXAMPLE-S3-BUCKET/error_data.json with your location.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create errors DynamicFrame, view schema
errors = glueContext.create_dynamic_frame.from_options(
    "s3", {"paths": ["s3://DOC-EXAMPLE-S3-BUCKET/error_data.json"]}, "json"
)
```

```

print("Schema of errors DynamicFrame:")
errors.printSchema()

# Show that errors only contains valid entries from the dataset
print("errors contains only valid records from the input dataset (2 of 4 records)")
errors.toDF().show()

# View errors
print("Errors count:", str(errors.errorsCount()))
print("Errors:")
errors.errorsAsDynamicFrame().toDF().show()

# View error fields and error data
error_record = errors.errorsAsDynamicFrame().toDF().head()

error_fields = error_record["error"]
print("Error fields: ")
print(error_fields.asDict().keys())

print("\nError record data:")
for key in error_fields.asDict().keys():
    print("\n", key, ": ", str(error_fields[key]))

```

출력

```

Schema of errors DynamicFrame:
root
 |-- id: int
 |-- name: string
 |-- surname: string
 |-- height: int

errors contains only valid records from the input dataset (2 of 4 records)
+---+-----+-----+-----+
| id| name|  surname|height|
+---+-----+-----+-----+
|  1|george|washington|  178|
|  4| john|      jay|  190|
+---+-----+-----+-----+

Errors count: 1
Errors:
+-----+

```

```
|          error|
+-----+
|[[ File "/tmp/20...|
+-----+
```

Error fields:

```
dict_keys(['callsite', 'msg', 'stackTrace', 'input', 'bytesread', 'source',
'dynamicRecord'])
```

Error record data:

```
callsite : Row(site=' File "/tmp/2060612586885849088", line 549, in <module>\n
sys.exit(main())\n File "/tmp/2060612586885849088", line 523, in main\n   response
= handler(content)\n File "/tmp/2060612586885849088", line 197, in execute_request
\n   result = node.execute()\n File "/tmp/2060612586885849088", line 103, in
execute\n   exec(code, global_dict)\n File "<stdin>", line 10, in <module>\n
File "/opt/amazon/lib/python3.6/site-packages/awsglue/dynamicframe.py", line 625, in
from_options\n   format_options, transformation_ctx, push_down_predicate, **kwargs)\n
File "/opt/amazon/lib/python3.6/site-packages/awsglue/context.py", line 233, in
create_dynamic_frame_from_options\n   source.setFormat(format, **format_options)\n',
info='')
```

```
msg : error in jackson reader
```

```
stackTrace : com.fasterxml.jackson.core.JsonParseException: Unexpected character
('{ ' (code 123)): was expecting either valid name character (for unquoted name) or
double-quote (for quoted) to start field name
at [Source: com.amazonaws.services.glue.readers.BufferedStream@73492578; line: 3,
column: 2]
at com.fasterxml.jackson.core.JsonParser._constructError(JsonParser.java:1581)
at
com.fasterxml.jackson.core.base.ParserMinimalBase._reportError(ParserMinimalBase.java:533)
at
com.fasterxml.jackson.core.base.ParserMinimalBase._reportUnexpectedChar(ParserMinimalBase.java:
at
com.fasterxml.jackson.core.json.UTF8StreamJsonParser._handleOddName(UTF8StreamJsonParser.java:
at
com.fasterxml.jackson.core.json.UTF8StreamJsonParser._parseName(UTF8StreamJsonParser.java:1650)
at
com.fasterxml.jackson.core.json.UTF8StreamJsonParser.nextToken(UTF8StreamJsonParser.java:740)
at com.amazonaws.services.glue.readers.JacksonReader$$anonfun$hasNextGoodToken
$1.apply(JacksonReader.scala:57)
at com.amazonaws.services.glue.readers.JacksonReader$$anonfun$hasNextGoodToken
$1.apply(JacksonReader.scala:57)
```



```
at scala.collection.Iterator$$$anon$9.next(Iterator.scala:162)
at scala.collection.Iterator$$$anon$16.hasNext(Iterator.scala:599)
at scala.collection.Iterator$$$anon$16.hasNext(Iterator.scala:598)
at scala.collection.Iterator$class.foreach(Iterator.scala:891)
at scala.collection.AbstractIterator.foreach(Iterator.scala:1334)
at com.amazonaws.services.glue.readers.JacksonReader$$$anonfun
$1.apply(JacksonReader.scala:120)
at com.amazonaws.services.glue.readers.JacksonReader$$$anonfun
$1.apply(JacksonReader.scala:116)
at
com.amazonaws.services.glue.DynamicRecordBuilder.handleErr(DynamicRecordBuilder.scala:209)
at
com.amazonaws.services.glue.DynamicRecordBuilder.handleErrorWithException(DynamicRecordBuilder
at
com.amazonaws.services.glue.readers.JacksonReader.nextFailSafe(JacksonReader.scala:116)
at com.amazonaws.services.glue.readers.JacksonReader.next(JacksonReader.scala:109)
at com.amazonaws.services.glue.readers.JSONReader.next(JSONReader.scala:247)
at
com.amazonaws.services.glue.hadoop.TapeHadoopRecordReaderSplittable.nextKeyValue(TapeHadoopRec
at org.apache.spark.rdd.NewHadoopRDD$$$anon$1.hasNext(NewHadoopRDD.scala:230)
at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37)
at scala.collection.Iterator$$$anon$11.hasNext(Iterator.scala:409)
at scala.collection.Iterator$$$anon$11.hasNext(Iterator.scala:409)
at scala.collection.Iterator$$$anon$13.hasNext(Iterator.scala:462)
at scala.collection.Iterator$$$anon$11.hasNext(Iterator.scala:409)
at scala.collection.Iterator$$$anon$11.hasNext(Iterator.scala:409)
at scala.collection.Iterator$$$anon$13.hasNext(Iterator.scala:462)
at scala.collection.Iterator$$$anon$11.hasNext(Iterator.scala:409)
at scala.collection.Iterator$$$anon$11.hasNext(Iterator.scala:409)
at org.apache.spark.sql.execution.SparkPlan$$$anonfun$2.apply(SparkPlan.scala:255)
at org.apache.spark.sql.execution.SparkPlan$$$anonfun$2.apply(SparkPlan.scala:247)
at org.apache.spark.rdd.RDD$$$anonfun$mapPartitionsInternal$1$$$anonfun$apply
$24.apply(RDD.scala:836)
at org.apache.spark.rdd.RDD$$$anonfun$mapPartitionsInternal$1$$$anonfun$apply
$24.apply(RDD.scala:836)
at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:324)
at org.apache.spark.rdd.RDD.iterator(RDD.scala:288)
at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:324)
at org.apache.spark.rdd.RDD.iterator(RDD.scala:288)
at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
at org.apache.spark.scheduler.Task.run(Task.scala:121)
at org.apache.spark.executor.Executor$TaskRunner$$$anonfun$10.apply(Executor.scala:408)
```

```

at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1360)
at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:414)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
at java.lang.Thread.run(Thread.java:750)

```

input :

bytesread : 252

source :

dynamicRecord : Row(id=2, name='benjamin', surname='franklin')

errorsCount

errorsCount() - DynamicFrame의 전체 오류 수를 반환합니다.

stageErrorsCount

stageErrorsCount - 이 DynamicFrame을 생성하는 과정에서 발생한 오류 수를 반환합니다.

DynamicFrameCollection 클래스

DynamicFrameCollection는 키가 DynamicFrames의 이름이고 값이 DynamicFrame 객체인 [DynamicFrame 클래스](#) 객체의 목록입니다.

`__init__`

`__init__(dynamic_frames, glue_ctx)`

- dynamic_frames – [DynamicFrame 클래스](#) 객체의 딕셔너리입니다.
- glue_ctx – [GlueContext 수업](#) 객체입니다.

키

keys() - 이 컬렉션에 키 목록을 반환합니다. 일반적으로 DynamicFrame 값과 대응하는 이름을 포함합니다.

값

`values(key)` - 이 컬렉션에 `DynamicFrame` 값의 목록을 반환합니다.

Select

`select(key)`

지정된 키(일반적으로 `DynamicFrame`의 이름)에 해당되는 `DynamicFrame`를 반환합니다.

- `key` - `DynamicFrameCollection`의 키이고 보통 `DynamicFrame`의 이름을 나타냅니다.

맵

`map(callable, transformation_ctx="")`

입장 함수를 사용하여 이 컬렉션의 `DynamicFrames` 기반 새로운 `DynamicFrameCollection`를 생성하고 반환합니다.

- `callable` - `DynamicFrame`과 지정 변환 내용을 파라미터로 여기고 `DynamicFrame`을 반환하는 함수입니다.
- `transformation_ctx` - 호출 기능에 사용되는 변환 내용입니다(선택 사항).

flatMap

`flatMap(f, transformation_ctx="")`

입장 함수를 사용하여 이 컬렉션의 `DynamicFrames` 기반 새로운 `DynamicFrameCollection`를 생성하고 반환합니다.

- `f` - `DynamicFrame`을 파라미터로 여기고 `DynamicFrame` 혹은 `DynamicFrameCollection`을 반환하는 함수입니다.
- `transformation_ctx` - 함수에 사용되는 변환 내용입니다(선택 사항).

DynamicFrameWriter 클래스

메서드

- [`__init__`](#)

- [from_options](#)
- [from_catalog](#)
- [from_jdbc_conf](#)

`__init__`

`__init__(glue_context)`

- `glue_context` - 사용할 [GlueContext](#) 수업입니다.

`from_options`

`from_options(frame, connection_type, connection_options={}, format=None, format_options={}, transformation_ctx="")`

지정한 연결 및 포맷을 사용하여 `DynamicFrame`를 작성합니다.

- `frame` - 작성할 `DynamicFrame`입니다.
- `connection_type` - 연결 유형입니다. 유효한 값에는 `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver` 및 `oracle`가 있습니다.
- `connection_options` - 경로 및 데이터베이스 테이블과 같은 연결 옵션입니다(선택 사항). `s3`의 `connection_type`의 경우, Amazon S3 경로가 정의됩니다.

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

JDBC 연결의 경우, 몇 가지 속성이 정의되어야 합니다. 단, 데이터베이스 이름이 URL의 일부여야 합니다. 연결 옵션에 선택적으로 포함될 수 있습니다.

Warning

스크립트에 암호를 저장하는 것은 권장되지 않습니다. AWS Secrets Manager 또는 AWS Glue 데이터 카탈로그에서 데이터를 검색할 때 `boto3` 사용을 고려합니다.

```
connection_options = {"url": "jdbc-url/database", "user": "username",
  "password": passwordVariable, "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 속성은 JDBC 테이블의 이름입니다. 데이터베이스 내의 스키마를 지원하는 JDBC 데이터 스토어의 경우 schema.table-name에 대해 지정합니다. 스키마가 제공되지 않으면 기본 "퍼블릭" 스키마가 사용됩니다.

자세한 내용은 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#) 단원을 참조하십시오.

- format - 포맷 사양입니다(선택 사항). 여러 포맷을 지원하는 Amazon Simple Storage Service(Amazon S3) 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- format_options - 지정된 포맷에 대한 포맷 옵션입니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- transformation_ctx - 사용할 변환 내용입니다(선택 사항).

from_catalog

```
from_catalog(frame, name_space, table_name, redshift_tmp_dir="", transformation_ctx="")
```

지정한 카탈로그 데이터베이스 및 테이블 이름을 사용하여 DynamicFrame를 작성합니다.

- frame - 작성할 DynamicFrame입니다.
- name_space - 사용할 데이터베이스입니다.
- table_name - 사용할 table_name입니다.
- redshift_tmp_dir - 사용할 Amazon Redshift 임시 디렉터리입니다(선택 사항).
- transformation_ctx - 사용할 변환 내용입니다(선택 사항).
- additional_options - AWS Glue에 제공되는 추가 옵션입니다.

Lake Formation 관리형 테이블에 쓰려면 다음과 같은 추가 옵션을 사용할 수 있습니다.

- transactionId - (문자열) 관리형 테이블에 쓰기를 수행할 트랜잭션 ID입니다. 이 트랜잭션은 이미 커밋되거나 중단되었을 수 없습니다. 그렇지 않으면 쓰기에 실패합니다.
- callDeleteObjectsOnCancel - (부울, 선택 사항) true(기본값)로 설정된 경우 AWS Glue는 객체가 Amazon S3에 기록된 후 DeleteObjectsOnCancel API를 자동으로 호출합니다. 자세한 내용은 AWS Lake Formation 개발자 가이드의 [DeleteObjectsOnCancel](#)을 참조하세요.

Example 예: Lake Formation의 관리형 테이블에 쓰기

```
txId = glueContext.start_transaction(read_only=False)
```

```
glueContext.write_dynamic_frame.from_catalog(
    frame=dyf,
    database = db,
    table_name = tbl,
    transformation_ctx = "datasource0",
    additional_options={"transactionId":txId})
...
glueContext.commit_transaction(txId)
```

from_jdbc_conf

```
from_jdbc_conf(frame, catalog_connection, connection_options={},
redshift_tmp_dir = "", transformation_ctx="")
```

지정한 JDBC 연결 정보를 사용하여 DynamicFrame를 작성합니다.

- frame - 작성할 DynamicFrame입니다.
- catalog_connection - 사용할 카탈로그 연결입니다.
- connection_options - 경로 및 데이터베이스 테이블과 같은 연결 옵션입니다(선택 사항).
- redshift_tmp_dir - 사용할 Amazon Redshift 임시 디렉터리입니다(선택 사항).
- transformation_ctx - 사용할 변환 내용입니다(선택 사항).

write_dynamic_frame의 예

이 예에서는 connection_options의 POSIX 경로 인수와 함께 S3의 connection_type을 사용하여 로컬 스토리지에 쓰기를 허용하는 출력을 로컬로 씁니다.

```
glueContext.write_dynamic_frame.from_options(\
    frame = dyf_splitFields,\
    connection_options = {'path': '/home/glue/GlueLocalOutput/'},\
    connection_type = 's3',\
    format = 'json')
```

DynamicFrameReader 수업

- 메서드 -

- [init](#)
- [from_rdd](#)

- [from_options](#)
- [from_catalog](#)

`__init__`

`__init__(glue_context)`

- `glue_context` - 사용할 [GlueContext](#) 수업입니다.

`from_rdd`

`from_rdd(data, name, schema=None, sampleRatio=None)`

탄력적 분산형 데이터셋(RDD)로부터 `DynamicFrame`을 판독합니다.

- `data` - 읽어야 할 데이터 집합입니다.
- `name` - 읽어야 할 이름입니다.
- `schema` - 판독할 스키마입니다(선택 사항).
- `sampleRatio` - 사용할 예제 비율입니다(선택 사항).

`from_options`

`from_options(connection_type, connection_options={}, format=None, format_options={}, transformation_ctx="")`

지정한 연결 및 포맷을 사용하여 `DynamicFrame`를 판독합니다.

- `connection_type` - 연결 유형입니다. 유효한 값에는 `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, `oracle`, `dynamodb`, `snowflake`가 있습니다.
- `connection_options` - 경로 및 데이터베이스 테이블과 같은 연결 옵션입니다(선택 사항). 자세한 내용은 [AWS Glue for Spark의 ETL 연결 유형 및 옵션](#)을 참조하십시오. `s3`의 `connection_type`의 경우, Amazon S3 경로가 배열에 정의됩니다.

```
connection_options = {"paths": [ "s3://mybucket/object_a", "s3://mybucket/object_b" ]}
```

JDBC 연결의 경우, 몇 가지 속성이 정의되어야 합니다. 단, 데이터베이스 이름이 URL의 일부여야 합니다. 연결 옵션에 선택적으로 포함될 수 있습니다.

⚠ Warning

스크립트에 암호를 저장하는 것은 권장되지 않습니다. AWS Secrets Manager 또는 AWS Glue 데이터 카탈로그에서 데이터를 검색하는 boto3 데 사용하는 것을 고려해 보십시오.

```
connection_options = {"url": "jdbc-url/database", "user": "username",
  "password": passwordVariable, "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

병렬 읽기를 수행하는 JDBC 연결의 경우 해시 필드 옵션을 설정할 수 있습니다. 예:

```
connection_options = {"url": "jdbc-url/database", "user": "username",
  "password": passwordVariable, "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path" , "hashfield": "month"}
```

자세한 정보는 [JDBC 테이블을 병렬로 읽기](#)을 참조하세요.

- `format` - 포맷 사양입니다(선택 사항). 여러 포맷을 지원하는 Amazon Simple Storage Service(Amazon S3) 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- `format_options` - 지정된 포맷에 대한 포맷 옵션입니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).
- `push_down_predicate` - 데이터 집합 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링할 수 있습니다. 자세한 내용은 [푸시다운 조건자를 사용하여 사전 필터링](#)을 참조하세요.

`from_catalog`

```
from_catalog(database, table_name, redshift_tmp_dir="",
  transformation_ctx="", push_down_predicate="", additional_options={})
```

지정한 카탈로그 네임스페이스 및 테이블 이름을 사용하여 `DynamicFrame`를 판독합니다.

- `database` - 읽어야 할 데이터베이스입니다.
- `table_name` - 읽어올 테이블의 이름입니다.

- `redshift_tmp_dir` - 사용할 Amazon Redshift 임시 디렉터리(Redshift에서 데이터를 읽지 않는 경우 선택 사항)입니다.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).
- `push_down_predicate` - 데이터 집합 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링할 수 있습니다. 자세한 정보는 [푸시다운 조건자를 사용하여 예비 필터링](#)을 참조하세요.
- `additional_options` - AWS Glue에 제공되는 추가 옵션입니다.
 - 병렬 읽기를 수행하는 JDBC 연결을 사용하기 위해 `hashfield`, `hasexpression` 또는 `hashpartitions` 옵션을 설정할 수 있습니다. 예:

```
additional_options = {"hashfield": "month"}
```

자세한 정보는 [JDBC 테이블을 병렬로 읽기](#)을 참조하세요.

- 인덱스 열을 기준으로 필터링할 카탈로그 표현식을 전달하려면 `catalogPartitionPredicate` 옵션을 볼 수 있습니다.

`catalogPartitionPredicate` - 카탈로그 표현식을 전달하여 인덱스 열을 기준으로 필터링할 수 있습니다. 이렇게 하면 필터링이 서버 측으로 푸시됩니다. 자세한 내용은 [AWS Glue 파티션 인덱스](#)를 참조하세요. `push_down_predicate`와 `catalogPartitionPredicate`는 다른 구문을 사용합니다. 전자는 Spark SQL 표준 구문을 사용하고 후자는 JSQL 구문 분석기를 사용합니다.

자세한 내용은 [AWS Glue에서 ETL 출력의 파티션 관리](#)을(를) 참조하세요.

GlueContext 수업

Apache Spark [SparkContext](#) 객체를 래핑하여 Apache Spark 플랫폼과 상호 작용하는 메커니즘을 제공합니다.

`__init__`

`__init__(sparkContext)`

- `sparkContext` - 사용할 아파치 스파크 내용입니다.

[생성 중]

- [__init__](#)
- [getSource](#)

- [create_dynamic_frame_from_rdd](#)
- [create_dynamic_frame_from_catalog](#)
- [create_dynamic_frame_from_options](#)
- [create_sample_dynamic_frame_from_catalog](#)
- [create_sample_dynamic_frame_from_options](#)
- [add_ingestion_time_columns](#)
- [create_data_frame_from_catalog](#)
- [create_data_frame_from_options](#)
- [forEachBatch](#)

getSource

getSource(connection_type, transformation_ctx = "", **options)

외부 소스에서 DataSource를 읽을 때 사용되는 DynamicFrames 객체를 생성합니다.

- **connection_type** - Amazon Simple Storage Service(Amazon S3), Amazon Redshift 및 JDBC와 같이 사용할 연결 유형입니다. 유효한 값에는 s3, mysql, postgresql, redshift, sqlserver, oracle 및 dynamodb가 있습니다.
- **transformation_ctx** - 사용할 변환 내용입니다(선택 사항).
- **options** - 선택적 이름-값 페어의 모음입니다. 자세한 정보는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)을 참조하세요.

다음은 getSource를 사용한 예입니다.

```
>>> data_source = context.getSource("file", paths=["/in/path"])
>>> data_source.setFormat("json")
>>> myFrame = data_source.getFrame()
```

create_dynamic_frame_from_rdd

create_dynamic_frame_from_rdd(data, name, schema=None, sample_ratio=None, transformation_ctx="")

Apache Spark Resilient Distributed Dataset(RDD)로부터 생성된 DynamicFrame을 반환합니다.

- data - 사용할 데이터 원본입니다.
- name - 사용할 데이터 이름입니다.
- schema - 사용할 스키마입니다(선택 사항).
- sample_ratio - 사용할 예제 비율입니다(선택 사항).
- transformation_ctx - 사용할 변환 내용입니다(선택 사항).

```
create_dynamic_frame_from_catalog
```

```
create_dynamic_frame_from_catalog(database, table_name, redshift_tmp_dir,
transformation_ctx = "", push_down_predicate= "", additional_options = {},
catalog_id = None)
```

데이터 카탈로그 데이터베이스 및 테이블 이름을 사용하여 생성된 DynamicFrame을 반환합니다. 이 메서드를 사용할 때는 지정된 AWS Glue Data Catalog 테이블의 테이블 속성과 additional_options 인수를 통해 기타 옵션을 제공합니다 format_options.

- Database - 읽어야 할 데이터베이스입니다.
- table_name - 읽어올 테이블의 이름입니다.
- redshift_tmp_dir - 사용할 Amazon Redshift 임시 디렉터리입니다(선택 사항).
- transformation_ctx - 사용할 변환 내용입니다(선택 사항).
- push_down_predicate - 데이터 집합 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링할 수 있습니다. 지원되는 소스 및 제한 사항은 [AWS Glue ETL의 푸시다운을 통한 읽기 최적화를 참조](#) 하십시오. 자세한 정보는 [푸시다운 조건자를 사용하여 예비 필터링](#)을 참조하세요.
- additional_options - 선택적 이름-값 페어의 모음입니다. 가능한 옵션에는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)에 나열된 옵션이 포함됩니다(endpointUrl, streamName, bootstrap.servers, security.protocol, topicName, classification 및 delimiter 제외). 지원되는 또 다른 옵션은 catalogPartitionPredicate입니다.

catalogPartitionPredicate - 카탈로그 표현식을 전달하여 인덱스 열을 기준으로 필터링할 수 있습니다. 이렇게 하면 필터링이 서버 측으로 푸시됩니다. 자세한 내용은 [AWS Glue 파티션 인덱스](#)를 참조하세요. push_down_predicate와 catalogPartitionPredicate는 다른 구문을 사용합니다. 전자는 Spark SQL 표준 구문을 사용하고 후자는 JSQL 구문 분석기를 사용합니다.

- catalog_id - 액세스 중인 데이터 카탈로그의 카탈로그 ID(계정 ID)입니다. None인 경우 호출자의 기본 계정 ID가 사용됩니다.

`create_dynamic_frame_from_options`

```
create_dynamic_frame_from_options(connection_type, connection_options={},
format=None, format_options={}, transformation_ctx = "")
```

지정된 연결 및 포맷으로 생성된 DynamicFrame을 반환합니다.

- `connection_type` - Amazon S3, Amazon Redshift 및 JDBC 등의 연결 유형입니다. 유효한 값에는 `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, `oracle` 및 `dynamodb`가 있습니다.
- `connection_options` - 경로 및 데이터베이스 테이블과 같은 연결 옵션(선택 사항). `s3`의 `connection_type`인 경우, Amazon S3 경로 목록이 정의됩니다.

```
connection_options = {"paths": ["s3://aws-glue-target/temp"]}
```

JDBC 연결의 경우, 몇 가지 속성이 정의되어야 합니다. 단, 데이터베이스 이름이 URL의 일부여야 합니다. 연결 옵션에 선택적으로 포함될 수 있습니다.

Warning

스크립트에 암호를 저장하는 것은 권장되지 않습니다. AWS Secrets Manager 또는 AWS Glue 데이터 카탈로그에서 데이터를 검색하는 boto3 데 사용하는 것을 고려해 보십시오.

```
connection_options = {"url": "jdbc-url/database", "user": "username",
"password": passwordVariable, "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

`dbtable` 속성은 JDBC 테이블의 이름입니다. 데이터베이스 내의 스키마를 지원하는 JDBC 데이터 스토어의 경우 `schema.table-name`에 대해 지정합니다. 스키마가 제공되지 않으면 기본 "퍼블릭" 스키마가 사용됩니다.

자세한 정보는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)을 참조하세요.

- `format`— 형식 사양. 여러 포맷을 지원하는 Amazon S3 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- `format_options` - 지정된 포맷에 대한 포맷 옵션입니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).

- `push_down_predicate` - 데이터 집합 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링할 수 있습니다. 지원되는 소스 및 제한 사항은 [AWS Glue ETL의 푸시다운을 통한 읽기 최적화를 참조](#) 하십시오. 자세한 내용은 [푸시다운 조건자를 사용하여 사전 필터링을 참조](#)하세요.

`create_sample_dynamic_frame_from_catalog`

```
create_sample_dynamic_frame_from_catalog(database, table_name, num,
redshift_tmp_dir, transformation_ctx = "", push_down_predicate= "",
additional_options = {}, sample_options = {}, catalog_id = None)
```

데이터 카탈로그 데이터베이스와 테이블 이름을 사용하여 생성된 샘플 DynamicFrame을 반환합니다. DynamicFrame에는 데이터 소스의 첫 번째 num 레코드만 포함됩니다.

- `database` - 읽어야 할 데이터베이스입니다.
- `table_name` - 읽어올 테이블의 이름입니다.
- `num` - 반환된 샘플 동적 프레임의 최대 레코드 수입니다.
- `redshift_tmp_dir` - 사용할 Amazon Redshift 임시 디렉터리입니다(선택 사항).
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).
- `push_down_predicate` - 데이터 집합 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링할 수 있습니다. 자세한 정보는 [푸시다운 조건자를 사용하여 예비 필터링을 참조](#)하세요.
- `additional_options` - 선택적 이름-값 페어의 모음입니다. 가능한 옵션에는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)에 나열된 옵션이 포함됩니다(`endpointUrl`, `streamName`, `bootstrap.servers`, `security.protocol`, `topicName`, `classification` 및 `delimiter` 제외).
- `sample_options` - 샘플링 동작을 제어하는 파라미터(선택 사항)입니다. Amazon S3 소스에 현재 사용 가능한 파라미터:
 - `maxSamplePartitions` - 샘플링에서 읽을 최대 파티션 수입니다. 기본값은 10입니다.
 - `maxSampleFilesPerPartition` - 샘플링이 한 파티션에서 읽을 최대 파일 수입니다. 기본값은 10입니다.

이러한 파라미터는 파일 나열에 소요되는 시간을 줄이는 데 도움이 됩니다. 예를 들어 데이터 세트에 1,000개의 파티션이 있고 각 파티션에 10개의 파일이 있다고 가정합니다.

`maxSamplePartitions = 10` 및 `maxSampleFilesPerPartition = 10`으로 설정하면 10,000개의 파일을 모두 나열하는 대신 샘플링은 각각에 처음 10개의 파일이 있는 처음 10개의 파티션만 나열하고 읽습니다($10 \times 10 =$ 총 100개의 파일).

- `catalog_id` - 액세스 중인 데이터 카탈로그의 카탈로그 ID(데이터 카탈로그의 계정 ID)입니다. 기본적으로 `None`으로 설정됩니다. `None`인 경우 서비스에 있는 호출 계정의 카탈로그 ID가 기본 설정됩니다.

`create_sample_dynamic_frame_from_options`

```
create_sample_dynamic_frame_from_options(connection_type,
connection_options={}, num, sample_options={}, format=None,
format_options={}, transformation_ctx = "")
```

지정된 연결과 형식으로 생성된 샘플 `DynamicFrame`을 반환합니다. `DynamicFrame`에는 데이터 소스의 첫 번째 `num` 레코드만 포함됩니다.

- `connection_type` - Amazon S3, Amazon Redshift 및 JDBC 등의 연결 유형입니다. 유효한 값에는 `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, `oracle` 및 `dynamodb`가 있습니다.
- `connection_options` - 경로 및 데이터베이스 테이블과 같은 연결 옵션(선택 사항). 자세한 정보는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)을 참조하세요.
- `num` - 반환된 샘플 동적 프레임의 최대 레코드 수입니다.
- `sample_options` - 샘플링 동작을 제어하는 파라미터(선택 사항)입니다. Amazon S3 소스에 현재 사용 가능한 파라미터:
 - `maxSamplePartitions` - 샘플링에서 읽을 최대 파티션 수입니다. 기본값은 10입니다.
 - `maxSampleFilesPerPartition` - 샘플링이 한 파티션에서 읽을 최대 파일 수입니다. 기본값은 10입니다.

이러한 파라미터는 파일 나열에 소요되는 시간을 줄이는 데 도움이 됩니다. 예를 들어 데이터 세트에 1,000개의 파티션이 있고 각 파티션에 10개의 파일이 있다고 가정합니다.

`maxSamplePartitions = 10` 및 `maxSampleFilesPerPartition = 10`으로 설정하면 10,000개의 파일을 모두 나열하는 대신 샘플링은 각각에 처음 10개의 파일이 있는 처음 10개의 파티션만 나열하고 읽습니다($10 \times 10 =$ 총 100개의 파일).

- `format`— 형식 사양. 여러 포맷을 지원하는 Amazon S3 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- `format_options` - 지정된 포맷에 대한 포맷 옵션입니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).
- `push_down_predicate` - 데이터 집합 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링할 수 있습니다. 자세한 정보는 [푸시다운 조건자를 사용하여 예비 필터링](#)을 참조하세요.

add_ingestion_time_columns

add_ingestion_time_columns(dataFrame, timeGranularity = "")

입력 DataFrame에 ingest_year, ingest_month, ingest_day, ingest_hour, ingest_minute와 같은 수집 시간 열을 추가합니다. 이 함수는 Amazon S3을 대상으로 하는 데이터 카탈로그 테이블을 지정할 때 AWS Glue가 생성하는 스크립트에서 자동으로 생성됩니다. 이 함수는 출력 테이블의 수집 시간 열로 파티션을 자동으로 업데이트합니다. 이를 통해 입력 데이터에 명시적인 수집 시간 열을 요구하지 않고도 출력 데이터를 수집 시간에 자동으로 분할할 수 있습니다.

- dataFrame - 수집 시간 열을 추가할 DataFrame입니다.
- timeGranularity - 시간 열의 세분성입니다. 유효 값은 "day", "hour" 및 "minute"입니다. 예를 들어 "hour"가 함수에 전달되면 원래 DataFrame에 "ingest_year", "ingest_month", "ingest_day" 및 "ingest_hour" 시간 열이 추가됩니다.

시간 세분성 열을 추가한 후 데이터 프레임을 반환합니다.

예제

```
dynamic_frame = DynamicFrame.fromDF(glueContext.add_ingestion_time_columns(dataFrame, "hour"))
```

create_data_frame_from_catalog

create_data_frame_from_catalog(database, table_name, transformation_ctx = "", additional_options = {})

데이터 카탈로그 테이블의 정보를 사용하여 생성된 DataFrame을 반환합니다.

- database - 읽을 데이터 카탈로그 데이터베이스입니다.
- table_name - 읽을 데이터 카탈로그 테이블의 이름입니다.
- transformation_ctx - 사용할 변환 내용입니다(선택 사항).
- additional_options - 선택적 이름-값 페어의 모음입니다. 가능한 옵션에는 스트리밍 소스에 대해 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)에 나열된 옵션(예: startingPosition, maxFetchTimeInMs, startingOffsets)이 있습니다.
- useSparkDataSource— true로 설정하면 AWS Glue가 네이티브 Spark 데이터 소스 API를 사용하여 테이블을 읽도록 강제합니다. Spark 데이터 소스 API는 AVRO, 바이너리, CSV, JSON, ORC, Parquet 및 텍스트 형식을 지원합니다. 데이터 카탈로그 테이블에서는 classification 속성을

사용하여 형식을 지정합니다. Spark 데이터 소스 API에 대한 자세한 내용은 공식 [Apache Spark 설명서](#)를 참조하세요.

`create_data_frame_from_catalog`를 `useSparkDataSource`와 함께 사용하면 다음과 같은 이점이 있습니다.

- `DataFrame`을 직접 반환하고 `create_dynamic_frame.from_catalog().toDF()`에 대한 대안을 제공합니다.
- 네이티브 포맷에 대한 AWS Lake Formation 테이블 수준의 권한 제어를 지원합니다.
- AWS Lake Formation 테이블 수준의 권한 제어 없이 데이터 레이크 형식을 읽을 수 있습니다. 자세한 정보는 [AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용](#)을 참조하세요.

`useSparkDataSource`를 활성화하면 필요에 따라 [Spark 데이터 소스 옵션](#)을 추가할 수도 있습니다. `additional_options` AWS Glue는 이러한 옵션을 Spark 리더에 직접 전달합니다.

- `useCatalogSchema`— `true`로 설정하면 AWS Glue는 데이터 카탈로그 스키마를 결과에 적용합니다. `DataFrame`. 그렇지 않으면 리더는 데이터에서 스키마를 추론합니다. 또한 `useCatalogSchema`를 활성화할 경우 `useSparkDataSource`를 `true`로 설정해야 합니다.

제한 사항

`useSparkDataSource` 옵션을 사용할 경우 다음과 같은 제한 사항을 고려하세요.

- 이를 사용하면 `useSparkDataSource` AWS Glue는 원래 Spark 세션과는 다른 별도의 Spark `DataFrame` 세션에서 새 세션을 생성합니다.
- Spark `DataFrame` 파티션 필터링은 다음 AWS Glue 기능에서 작동하지 않습니다.
 - [작업 복마크](#)
 - [Amazon S3 스토리지 클래스 제외](#)
 - [카탈로그 파티션 조건자](#)

이러한 기능과 함께 파티션 필터링을 사용하려면 AWS Glue 푸시다운 조건자를 사용할 수 있습니다. 자세한 정보는 [푸시다운 조건자를 사용하여 예비 필터링](#)을 참조하세요. 분할되지 않은 열에 대한 필터링에는 영향을 주지 않습니다.

다음 예제 스크립트는 `excludeStorageClasses` 옵션을 사용하여 파티션 필터링을 수행하는 잘못된 방법을 보여줍니다.

```
// Incorrect partition filtering using Spark filter with excludeStorageClasses
```

```
read_df = glueContext.create_data_frame_from_catalog(
```



```

    database=database_name,
    table_name=table_name,
    additional_options = {
        "useSparkDataSource": True,
        "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)

// Suppose year and month are partition keys.
// Filtering on year and month won't work, the filtered_df will still
// contain data with other year/month values.
filtered_df = read_df.filter("year == '2017' and month == '04' and 'state == 'CA'")

```

다음 예제 스크립트는 `excludeStorageClasses` 옵션을 사용하여 파티션 필터링을 수행하기 위해 푸시다운 조건자를 사용하는 올바른 방법을 보여줍니다.

```

// Correct partition filtering using the AWS Glue pushdown predicate
// with excludeStorageClasses
read_df = glueContext.create_data_frame.from_catalog(
    database=database_name,
    table_name=table_name,
    // Use AWS Glue pushdown predicate to perform partition filtering
    push_down_predicate = "(year=='2017' and month=='04')",
    additional_options = {
        "useSparkDataSource": True,
        "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)

// Use Spark filter only on non-partitioned columns
filtered_df = read_df.filter("state == 'CA'")

```

예: Spark 데이터 소스 리더를 사용하여 CSV 테이블 생성

```

// Read a CSV table with '\t' as separator
read_df = glueContext.create_data_frame.from_catalog(
    database=<database_name>,
    table_name=<table_name>,
    additional_options = {"useSparkDataSource": True, "sep": '\t'}
)

```

`create_data_frame_from_options`

```
create_data_frame_from_options(connection_type, connection_options={},
format=None, format_options={}, transformation_ctx = "")
```

이 API는 이제 더 이상 사용되지 않습니다. 대신에 `getSource()` API를 사용하십시오. 지정된 연결 및 포맷으로 생성된 `DataFrame`을 반환합니다. 이 함수는 AWS Glue 스트리밍 소스에만 사용됩니다.

- `connection_type` - 스트리밍 연결 유형입니다. 유효한 값에는 `kinesis` 및 `kafka(이)`가 있습니다.
- `connection_options` - Kinesis 및 Kafka에 대해 서로 다른 연결 옵션입니다. [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)에서 각 스트리밍 데이터 원본에 대한 모든 연결 옵션 목록을 찾을 수 있습니다. 스트리밍 연결 옵션에는 다음과 같은 차이점이 있습니다.
 - Kinesis 스트리밍 소스에는 `streamARN`, `startingPosition`, `inferSchema` 및 `classification`이 필요합니다.
 - Kafka 스트리밍 소스에는 `connectionName`, `topicName`, `startingOffsets`, `inferSchema` 및 `classification`이 필요합니다.
- `format`— 형식 사양. 여러 포맷을 지원하는 Amazon S3 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷에 관한 내용은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#) 섹션을 참조하세요.
- `format_options` - 지정된 포맷에 대한 포맷 옵션입니다. 지원되는 포맷 옵션에 대한 자세한 내용은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#) 섹션을 참조하세요.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).

Amazon Kinesis 스트리밍 소스의 예:

```
kinesis_options =
  { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
    "startingPosition": "TRIM_HORIZON",
    "inferSchema": "true",
    "classification": "json"
  }
data_frame_datasource0 =
  glueContext.create_data_frame.from_options(connection_type="kinesis",
  connection_options=kinesis_options)
```

Kafka 스트리밍 소스의 예:

```
kafka_options =
  { "connectionName": "ConfluentKafka",
    "topicName": "kafka-auth-topic",
    "startingOffsets": "earliest",
    "inferSchema": "true",
    "classification": "json"
  }
data_frame_datasource0 =
  glueContext.create_data_frame.from_options(connection_type="kafka",
  connection_options=kafka_options)
```

forEachBatch

forEachBatch(frame, batch_function, options)

스트리밍 소스에서 읽는 모든 마이크로 배치에 전달된 `batch_function`을 적용합니다.

- `frame`— 현재 마이크로 배치를 DataFrame 포함합니다.
- `batch_function` - 모든 마이크로 배치에 적용될 함수입니다.
- `options` - 마이크로 배치를 처리하는 방법에 대한 정보가 들어 있는 키-값 페어 컬렉션입니다. 다음 옵션이 필요합니다.
 - `windowSize` - 각 배치를 처리하는 데 소요되는 시간입니다.
 - `checkpointLocation` - 스트리밍 ETL 작업에 대해 체크포인트가 저장되는 위치입니다.
 - `batchMaxRetries` - 실패한 경우 배치를 다시 시도할 수 있는 최대 횟수입니다. 기본값은 3입니다. 이 옵션은 Glue 버전 2.0 이상에서만 구성할 수 있습니다.

예:

```
glueContext.forEachBatch(
  frame = data_frame_datasource0,
  batch_function = processBatch,
  options = {
    "windowSize": "100 seconds",
    "checkpointLocation": "s3://kafka-auth-dataplane/confluent-test/output/
checkpoint/"
  }
)

def processBatch(data_frame, batchId):
```

```

if (data_frame.count() > 0):
    datasource0 = DynamicFrame.fromDF(
        glueContext.add_ingestion_time_columns(data_frame, "hour"),
        glueContext, "from_data_frame"
    )
    additionalOptions_datasink1 = {"enableUpdateCatalog": True}
    additionalOptions_datasink1["partitionKeys"] = ["ingest_yr", "ingest_mo",
"ingest_day"]
    datasink1 = glueContext.write_dynamic_frame.from_catalog(
        frame = datasource0,
        database = "tempdb",
        table_name = "kafka-auth-table-output",
        transformation_ctx = "datasink1",
        additional_options = additionalOptions_datasink1
    )

```

Amazon S3의 데이터 집합 작업

- [purge_table](#)
- [purge_s3_path](#)
- [transition_table](#)
- [transition_s3_path](#)

purge_table

purge_table(catalog_id=None, database="", table_name="", options={}, transformation_ctx="")

지정된 카탈로그의 데이터베이스 및 테이블에 대한 파일을 Amazon S3에서 삭제합니다. 파티션의 모든 파일을 삭제하면 해당 파티션도 카탈로그에서 삭제됩니다.

삭제된 객체를 복구하려면 Amazon S3 버킷에서 [객체 버전 관리](#)를 설정할 수 있습니다. 객체 버전 관리가 활성화되어 있지 않은 버킷에서 객체를 삭제하는 경우에는 객체를 복구할 수 없습니다. 버전 관리가 사용되는 버킷에서 삭제된 객체를 복구하는 방법에 대한 자세한 내용은 AWS Support 지식 센터에서 [삭제된 Amazon S3 객체를 검색하려면 어떻게 해야 합니까?](#)를 참조하세요.

- **catalog_id** - 액세스 중인 데이터 카탈로그의 카탈로그 ID(데이터 카탈로그의 계정 ID)입니다. 기본적으로 None으로 설정됩니다. None인 경우 서비스에 있는 호출 계정의 카탈로그 ID가 기본 설정됩니다.
- **database** - 사용할 데이터베이스입니다.

- `table_name` - 사용할 테이블의 이름입니다.
- `options` - 삭제할 파일 필터링 및 매니페스트 파일 생성을 위한 옵션입니다.
 - `retentionPeriod` - 파일을 보존할 기간(시간)을 지정합니다. 보존 기간 내의 파일은 유지됩니다. 기본적으로 168시간(7일)으로 설정됩니다.
 - `partitionPredicate` - 이 조건자를 충족하는 파티션이 삭제됩니다. 이러한 파티션에서 보존 기간 내에 있는 파일은 삭제되지 않습니다. 기본적으로 ""(비움)로 설정합니다.
 - `excludeStorageClasses` - `excludeStorageClasses` 집합에 스토리지 클래스가 있는 파일은 삭제되지 않습니다. 기본값은 `Set()`(빈 집합)입니다.
- `manifestFilePath` - 매니페스트 파일 생성을 위한 선택적 경로입니다. 성공적으로 제거된 모든 파일은 `Success.csv`에 기록되고 실패한 파일은 `Failed.csv`에 기록됩니다.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항). 매니페스트 파일 경로에 사용됩니다.

Example

```
glueContext.purge_table("database", "table", {"partitionPredicate": "(month=='march')",
"retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath":
"s3://bucketmanifest/"})
```

purge_s3_path

`purge_s3_path(s3_path, options={}, transformation_ctx="")`

지정된 Amazon S3 경로에서 파일을 재귀적으로 삭제합니다.

삭제된 객체를 복구하려면 Amazon S3 버킷에서 [객체 버전 관리](#)를 설정할 수 있습니다. 객체 버전 관리가 설정되어 있지 않은 버킷에서 객체를 삭제하는 경우에는 객체를 복구할 수 없습니다. 버전 관리를 사용하여 버킷에서 삭제된 객체를 복구하는 방법에 대한 자세한 내용은 [삭제된 Amazon S3 객체를 검색하려면 어떻게 해야 합니까?](#) 를 참조하십시오. AWS Support 지식 센터에서.

- `s3_path` - `s3://<bucket>/<prefix>/` 포맷으로 삭제할 파일에 대한 Amazon S3의 경로입니다.
- `options` - 삭제할 파일 필터링 및 매니페스트 파일 생성을 위한 옵션입니다.
 - `retentionPeriod` - 파일을 보존할 기간(시간)을 지정합니다. 보존 기간 내의 파일은 유지됩니다. 기본적으로 168시간(7일)으로 설정됩니다.
 - `excludeStorageClasses` - `excludeStorageClasses` 집합에 스토리지 클래스가 있는 파일은 삭제되지 않습니다. 기본값은 `Set()`(빈 집합)입니다.

- `manifestFilePath` - 매니페스트 파일 생성을 위한 선택적 경로입니다. 성공적으로 제거된 모든 파일은 `Success.csv`에 기록되고 실패한 파일은 `Failed.csv`에 기록됩니다.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항). 매니페스트 파일 경로에 사용됩니다.

Example

```
glueContext.purge_s3_path("s3://bucket/path/", {"retentionPeriod": 1,
"excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/"})
```

transition_table

transition_table(database, table_name, transition_to, options={}, transformation_ctx="", catalog_id=None)

지정된 카탈로그의 데이터베이스 및 테이블에 대해 Amazon S3에 저장된 파일의 스토리지 클래스를 전환합니다.

두 스토리지 클래스 간에 전환할 수 있습니다. GLACIER 및 DEEP_ARCHIVE 스토리지 클래스의 경우 이러한 클래스로 전환할 수 있습니다. 하지만 S3 RESTORE를 사용하여 GLACIER 및 DEEP_ARCHIVE 스토리지 클래스에서 전환할 수 있습니다.

Amazon S3에서 파일 또는 파티션을 읽는 AWS Glue ETL 작업을 실행하는 경우 일부 Amazon S3 스토리지 클래스 유형을 제외할 수 있습니다. 자세한 내용은 [Amazon S3 스토리지 클래스 제외](#)를 참조하세요.

- `database` - 사용할 데이터베이스입니다.
- `table_name` - 사용할 테이블의 이름입니다.
- `transition_to` - 전환할 [Amazon S3 스토리지 클래스](#)입니다.
- `options` - 삭제할 파일 필터링 및 매니페스트 파일 생성을 위한 옵션입니다.
 - `retentionPeriod` - 파일을 보존할 기간(시간)을 지정합니다. 보존 기간 내의 파일은 유지됩니다. 기본적으로 168시간(7일)으로 설정됩니다.
 - `partitionPredicate` - 이 조건자를 충족하는 파티션이 전환됩니다. 이러한 파티션에서 보존 기간 내에 있는 파일은 전환되지 않습니다. 기본적으로 ""(비움)로 설정합니다.
 - `excludeStorageClasses` - `excludeStorageClasses` 집합에 스토리지 클래스가 있는 파일은 전환되지 않습니다. 기본값은 `Set()`(빈 집합)입니다.
 - `manifestFilePath` - 매니페스트 파일 생성을 위한 선택적 경로입니다. 성공적으로 전환된 모든 파일은 `Success.csv`에 기록되고 실패한 파일은 `Failed.csv`에 기록됩니다.

- `accountId` - 전환 변환을 실행할 Amazon Web Services 계정 ID입니다. 이 변환에 대해 필수입니다.
- `roleArn`— 전환 변환을 실행하는 AWS 역할. 이 변환에 대해 필수입니다.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항). 매니페스트 파일 경로에 사용됩니다.
- `catalog_id` - 액세스 중인 데이터 카탈로그의 카탈로그 ID(데이터 카탈로그의 계정 ID)입니다. 기본적으로 `None`으로 설정됩니다. `None`인 경우 서비스에 있는 호출 계정의 카탈로그 ID가 기본 설정됩니다.

Example

```
glueContext.transition_table("database", "table", "STANDARD_IA", {"retentionPeriod": 1,
  "excludeStorageClasses": ["STANDARD_IA"], "manifestFilePath": "s3://bucketmanifest/",
  "accountId": "12345678901", "roleArn": "arn:aws:iam::123456789012:user/example-username"})
```

transition_s3_path

transition_s3_path(s3_path, transition_to, options={}, transformation_ctx="")

지정된 Amazon S3 경로에 있는 파일의 스토리지 클래스를 재귀적으로 전환합니다.

두 스토리지 클래스 간에 전환할 수 있습니다. GLACIER 및 DEEP_ARCHIVE 스토리지 클래스의 경우 이러한 클래스로 전환할 수 있습니다. 하지만 S3 RESTORE를 사용하여 GLACIER 및 DEEP_ARCHIVE 스토리지 클래스에서 전환할 수 있습니다.

Amazon S3에서 파일 또는 파티션을 읽는 AWS Glue ETL 작업을 실행하는 경우 일부 Amazon S3 스토리지 클래스 유형을 제외할 수 있습니다. 자세한 내용은 [Amazon S3 스토리지 클래스 제외](#)를 참조하세요.

- `s3_path` - `s3://<bucket>/<prefix>/` 포맷으로 전환할 파일에 대한 Amazon S3의 경로입니다.
- `transition_to` - 전환할 [Amazon S3 스토리지 클래스](#)입니다.
- `options` - 삭제할 파일 필터링 및 매니페스트 파일 생성을 위한 옵션입니다.
 - `retentionPeriod` - 파일을 보존할 기간(시간)을 지정합니다. 보존 기간 내의 파일은 유지됩니다. 기본적으로 168시간(7일)으로 설정됩니다.
 - `partitionPredicate` - 이 조건자를 충족하는 파티션이 전환됩니다. 이러한 파티션에서 보존 기간 내에 있는 파일은 전환되지 않습니다. 기본적으로 ""(비움)로 설정합니다.

- `excludeStorageClasses` – `excludeStorageClasses` 집합에 스토리지 클래스가 있는 파일은 전환되지 않습니다. 기본값은 `Set()`(빈 집합)입니다.
- `manifestFilePath` - 매니페스트 파일 생성을 위한 선택적 경로입니다. 성공적으로 전환된 모든 파일은 `Success.csv`에 기록되고 실패한 파일은 `Failed.csv`에 기록됩니다.
- `accountId` - 전환 변환을 실행할 Amazon Web Services 계정 ID입니다. 이 변환에 대해 필수입니다.
- `roleArn`— 전환 변환을 실행하는 AWS 역할. 이 변환에 대해 필수입니다.
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항). 매니페스트 파일 경로에 사용됩니다.

Example

```
glueContext.transition_s3_path("s3://bucket/prefix/", "STANDARD_IA",
{"retentionPeriod": 1, "excludeStorageClasses": ["STANDARD_IA"],
"manifestFilePath": "s3://bucketmanifest/", "accountId": "12345678901", "roleArn":
"arn:aws:iam::123456789012:user/example-username"})
```

추출

- [extract_jdbc_conf](#)

extract_jdbc_conf

extract_jdbc_conf(connection_name, catalog_id = None)

데이터 카탈로그에 있는 AWS Glue 연결 개체의 구성 속성이 있는 dict와 키를 반환합니다.

- `user` – 데이터베이스 사용자 이름입니다.
- `password` – 데이터베이스 암호입니다.
- `vendor` – 공급업체를 지정합니다 (mysql, postgresql, oracle, sqlserver 등).
- `enforceSSL` – 보안 연결이 필요한지 여부를 나타내는 부울 문자열입니다.
- `customJDBCCert` – 표시된 Amazon S3 경로의 특정 클라이언트 인증서를 사용합니다.
- `skipCustomJDBCCertValidation` – CA에서 `customJDBCCert`의 유효성을 검사해야 하는지 여부를 나타내는 부울 문자열입니다.
- `customJDBCCertString` – 드라이버 유형에 맞는 사용자 지정 인증서에 대한 추가 정보입니다.
- `url` – (더 이상 사용되지 않음) 프로토콜, 서버 및 포트만 있는 JDBC URL입니다.

- `fullUrl` - 연결이 생성될 때 입력한 JDBC URL입니다(AWS Glue 버전 3.0 이상에서 사용 가능).

JDBC 구성 검색의 예:

```
jdbc_conf = glueContext.extract_jdbc_conf(connection_name="your_glue_connection_name")
print(jdbc_conf)
>>> {'enforceSSL': 'false', 'skipCustomJDBCCertValidation': 'false', 'url':
'jdbc:mysql://myserver:3306', 'fullUrl': 'jdbc:mysql://myserver:3306/mydb',
'customJDBCCertString': '', 'user': 'admin', 'customJDBCCert': '', 'password': '1234',
'vendor': 'mysql'}
```

트랜잭션

- [start_transaction](#)
- [commit_transaction](#)
- [cancel_transaction](#)

`start_transaction`

start_transaction(read_only)

새 트랜잭션을 시작합니다. 내부적으로 Lake Formation [startTransaction](#) API를 호출합니다.

- `read_only` - (부울) 이 트랜잭션이 읽기 전용인지 또는 읽기/쓰기인지를 나타냅니다. 읽기 전용 트랜잭션 ID를 사용하여 수행된 쓰기는 거부됩니다. 읽기 전용 트랜잭션은 커밋할 필요가 없습니다.

트랜잭션 ID를 반환합니다.

`commit_transaction`

commit_transaction(transaction_id, wait_for_commit = True)

지정된 트랜잭션을 커밋하려는 시도입니다. 트랜잭션이 커밋을 완료하기 전에 `commit_transaction`이 반환될 수 있습니다. 내부적으로 Lake Formation [commitTransaction](#) API를 호출합니다.

- `transaction_id` - (문자열) 커밋할 트랜잭션입니다.

- `wait_for_commit` - (부울) `commit_transaction`이 즉시 반환되는지 여부를 결정합니다. 기본 값은 `true`입니다. `false`인 경우 `commit_transaction`은 폴링한 후 트랜잭션이 커밋될 때까지 기다립니다. 대기 시간은 최대 재시도 횟수가 6회인 지수 백오프를 사용하여 1분으로 제한됩니다.

커밋이 수행되었는지 여부를 나타내는 부울을 반환합니다.

`cancel_transaction`

`cancel_transaction(transaction_id)`

지정된 트랜잭션을 취소하려는 시도입니다. 트랜잭션이 이전에 커밋된 경우 `TransactionCommittedException` 예외를 반환합니다. 내부적으로 레이크 포메이션 [CancelTransaction](#) API를 호출합니다.

- `transaction_id` - (문자열) 취소할 트랜잭션입니다.

작성

- [getSink](#)
- [write_dynamic_frame_from_options](#)
- [write_from_options](#)
- [write_dynamic_frame_from_catalog](#)
- [write_data_frame_from_catalog](#)
- [write_dynamic_frame_from_jdbc_conf](#)
- [write_from_jdbc_conf](#)

`getSink`

`getSink(connection_type, format = None, transformation_ctx = "", **options)`

외부 소스로 `DataSink`를 읽을 때 사용되는 `DynamicFrames` 객체를 얻습니다. 먼저 SparkSQL `format`를 확인하여 예상 싱크를 얻도록 합니다.

- `connection_type` - Amazon S3, Amazon Redshift 및 JDBC와 같이 사용할 연결 유형입니다. 유효한 값에는 `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, `oracle`, `kinesis`, `kafka`가 있습니다.

- `format` - 사용할 SparkSQL 포맷입니다(선택 사항).
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).
- `options` - 연결 옵션을 지정하는 데 사용되는 이름-값 페어의 컬렉션입니다. 몇 가지 가능한 값은 다음과 같습니다.
 - `user` 및 `password`: 권한 부여용
 - `url`: 데이터 스토어에 대한 엔드포인트
 - `dbtable`: 대상 테이블의 이름
 - `bulkSize`: 삽입 작업의 병렬 처리 수준

지정할 수 있는 옵션은 연결 유형에 따라 다릅니다. 추가 값과 예제는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#) 섹션을 참조하세요.

예제

```
>>> data_sink = context.getSink("s3")
>>> data_sink.setFormat("json"),
>>> data_sink.writeFrame(myFrame)
```

`write_dynamic_frame_from_options`

```
write_dynamic_frame_from_options(frame, connection_type,  
connection_options={}, format=None, format_options={}, transformation_ctx =  
""')
```

지정한 연결 및 포맷을 사용하여 `DynamicFrame`를 작성하고 반환합니다.

- `frame` - 작성할 `DynamicFrame`입니다.
- `connection_type` - Amazon S3, Amazon Redshift 및 JDBC 등의 연결 유형입니다. 유효한 값에는 `s3`, `mysql`, `postgresql`, `redshift`, `sqlserver`, `oracle`, `kinesis`, `kafka`가 있습니다.
- `connection_options` - 경로 및 데이터베이스 테이블과 같은 연결 옵션입니다(선택 사항). `s3`의 `connection_type`의 경우, Amazon S3 경로가 정의됩니다.

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

JDBC 연결의 경우, 몇 가지 속성이 정의되어야 합니다. 단, 데이터베이스 이름이 URL의 일부여야 합니다. 연결 옵션에 선택적으로 포함될 수 있습니다.

⚠ Warning

스크립트에 암호를 저장하는 것은 권장되지 않습니다. AWS Secrets Manager 또는 AWS Glue 데이터 카탈로그에서 데이터를 검색하는 boto3 데 사용하는 것을 고려해 보십시오.

```
connection_options = {"url": "jdbc-url/database", "user": "username",
  "password": passwordVariable, "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 속성은 JDBC 테이블의 이름입니다. 데이터베이스 내의 스키마를 지원하는 JDBC 데이터 스토어의 경우 schema.table-name에 대해 지정합니다. 스키마가 제공되지 않으면 기본 "퍼블릭" 스키마가 사용됩니다.

자세한 정보는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)을 참조하세요.

- **format**— 형식 사양. 여러 포맷을 지원하는 Amazon S3 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- **format_options** - 지정된 포맷에 대한 포맷 옵션입니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- **transformation_ctx** - 사용할 변환 내용입니다(선택 사항).

write_from_options

```
write_from_options(frame_or_dfc, connection_type, connection_options={},
  format={}, format_options={}, transformation_ctx = "")
```

지정한 연결 및 포맷 정보로 생성된 DynamicFrame 또는 DynamicFrameCollection을 작성하고 반환합니다.

- **frame_or_dfc** - 작성할 DynamicFrame 또는 DynamicFrameCollection입니다.
- **connection_type** - Amazon S3, Amazon Redshift 및 JDBC 등의 연결 유형입니다. 유효한 값에는 s3, mysql, postgresql, redshift, sqlserver 및 oracle가 있습니다.
- **connection_options** - 경로 및 데이터베이스 테이블과 같은 연결 옵션입니다(선택 사항). s3의 connection_type의 경우, Amazon S3 경로가 정의됩니다.

```
connection_options = {"path": "s3://aws-glue-target/temp"}
```

JDBC 연결의 경우, 몇 가지 속성이 정의되어야 합니다. 단, 데이터베이스 이름이 URL의 일부여야 합니다. 연결 옵션에 선택적으로 포함될 수 있습니다.

⚠ Warning

스크립트에 암호를 저장하는 것은 권장되지 않습니다. AWS Secrets Manager 또는 AWS Glue 데이터 카탈로그에서 데이터를 검색하는 boto3 데 사용하는 것을 고려해 보십시오.

```
connection_options = {"url": "jdbc-url/database", "user": "username",
  "password": passwordVariable, "dbtable": "table-name", "redshiftTmpDir": "s3-tempdir-path"}
```

dbtable 속성은 JDBC 테이블의 이름입니다. 데이터베이스 내의 스키마를 지원하는 JDBC 데이터 스토어의 경우 schema.table-name에 대해 지정합니다. 스키마가 제공되지 않으면 기본 "퍼블릭" 스키마가 사용됩니다.

자세한 정보는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)을 참조하세요.

- **format**— 형식 사양. 여러 포맷을 지원하는 Amazon S3 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- **format_options** - 지정된 포맷에 대한 포맷 옵션입니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하십시오.
- **transformation_ctx** - 사용할 변환 내용입니다(선택 사항).

write_dynamic_frame_from_catalog

```
write_dynamic_frame_from_catalog(frame, database, table_name,
redshift_tmp_dir, transformation_ctx = "", additional_options = {},
catalog_id = None)
```

데이터 카탈로그 데이터베이스 및 테이블의 정보를 사용하여 DynamicFrame을 작성하고 반환합니다.

- **frame** - 작성할 DynamicFrame입니다.
- **Database** - 테이블이 들어 있는 데이터 카탈로그 데이터베이스입니다.
- **table_name** - 대상과 연결된 데이터 카탈로그 테이블의 이름입니다.
- **redshift_tmp_dir** - 사용할 Amazon Redshift 임시 디렉터리입니다(선택 사항).

- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).
- `additional_options` - 선택적 이름-값 페어의 모음입니다.
- `catalog_id` - 액세스 중인 데이터 카탈로그의 카탈로그 ID(계정 ID)입니다. None인 경우 호출자의 기본 계정 ID가 사용됩니다.

`write_data_frame_from_catalog`

```
write_data_frame_from_catalog(frame, database, table_name,
redshift_tmp_dir, transformation_ctx = "", additional_options = {},
catalog_id = None)
```

데이터 카탈로그 데이터베이스 및 테이블의 정보를 사용하여 DataFrame을 작성하고 반환합니다. 이 메서드는 데이터 레이크 형식(Hudi, Iceberg, Delta Lake) 쓰기를 지원합니다. 자세한 정보는 [AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용](#)을 참조하세요.

- `frame` - 작성할 DataFrame입니다.
- `Database` - 테이블이 들어 있는 데이터 카탈로그 데이터베이스입니다.
- `table_name` - 대상과 연결된 데이터 카탈로그 테이블의 이름입니다.
- `redshift_tmp_dir` - 사용할 Amazon Redshift 임시 디렉터리입니다(선택 사항).
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).
- `additional_options` - 선택적 이름-값 페어의 모음입니다.
 - `useSparkDataSink`— true로 설정하면 AWS Glue가 네이티브 Spark 데이터 싱크 API를 사용하여 테이블에 쓰도록 강제합니다. 이 옵션을 활성화하면 필요에 `additional_options` 따라 [Spark 데이터 소스 옵션](#)을 추가할 수 있습니다. AWS Glue는 이러한 옵션을 Spark 라이터에게 직접 전달합니다.
- `catalog_id` - 액세스 중인 데이터 카탈로그의 카탈로그 ID(계정 ID)입니다. 값을 지정하지 않으면 발신자의 기본 계정 ID가 사용됩니다.

제한 사항

`useSparkDataSink` 옵션을 사용할 경우 다음과 같은 제한 사항을 고려하세요.

- `useSparkDataSink` 옵션을 사용하는 경우 [enableUpdateCatalog](#) 옵션이 지원되지 않습니다.

예: Spark 데이터 소스 라이터를 사용하여 Hudi 테이블에 쓰기

```

hoodie_options = {
    'useSparkDataSink': True,
    'hoodie.table.name': <table_name>,
    'hoodie.datasource.write.storage.type': 'COPY_ON_WRITE',
    'hoodie.datasource.write.recordkey.field': 'product_id',
    'hoodie.datasource.write.table.name': <table_name>,
    'hoodie.datasource.write.operation': 'upsert',
    'hoodie.datasource.write.precombine.field': 'updated_at',
    'hoodie.datasource.write.hive_style_partitioning': 'true',
    'hoodie.upsert.shuffle.parallelism': 2,
    'hoodie.insert.shuffle.parallelism': 2,
    'hoodie.datasource.hive_sync.enable': 'true',
    'hoodie.datasource.hive_sync.database': <database_name>,
    'hoodie.datasource.hive_sync.table': <table_name>,
    'hoodie.datasource.hive_sync.use_jdbc': 'false',
    'hoodie.datasource.hive_sync.mode': 'hms'}

glueContext.write_data_frame.from_catalog(
    frame = <df_product_inserts>,
    database = <database_name>,
    table_name = <table_name>,
    additional_options = hoodie_options
)

```

write_dynamic_frame_from_jdbc_conf

```

write_dynamic_frame_from_jdbc_conf(frame, catalog_connection,
connection_options={}, redshift_tmp_dir = "", transformation_ctx = "",
catalog_id = None)

```

지정한 JDBC 연결 정보를 사용하여 DynamicFrame를 작성하고 반환합니다.

- frame - 작성할 DynamicFrame입니다.
- catalog_connection - 사용할 카탈로그 연결입니다.
- connection_options - 경로 및 데이터베이스 테이블과 같은 연결 옵션입니다(선택 사항). 자세한 정보는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)을 참조하세요.
- redshift_tmp_dir - 사용할 Amazon Redshift 임시 디렉터리입니다(선택 사항).
- transformation_ctx - 사용할 변환 내용입니다(선택 사항).
- catalog_id - 액세스 중인 데이터 카탈로그의 카탈로그 ID(계정 ID)입니다. None인 경우 호출자의 기본 계정 ID가 사용됩니다.

`write_from_jdbc_conf`

```
write_from_jdbc_conf(frame_or_dfc, catalog_connection,
connection_options={}, redshift_tmp_dir = "", transformation_ctx = "",
catalog_id = None)
```

지정한 JDBC 연결 정보를 사용하여 `DynamicFrame` 또는 `DynamicFrameCollection`을 작성하고 반환합니다.

- `frame_or_dfc` - 작성할 `DynamicFrame` 또는 `DynamicFrameCollection`입니다.
- `catalog_connection` - 사용할 카탈로그 연결입니다.
- `connection_options` - 경로 및 데이터베이스 테이블과 같은 연결 옵션입니다(선택 사항). 자세한 정보는 [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)을 참조하세요.
- `redshift_tmp_dir` - 사용할 Amazon Redshift 임시 디렉터리입니다(선택 사항).
- `transformation_ctx` - 사용할 변환 내용입니다(선택 사항).
- `catalog_id` - 액세스 중인 데이터 카탈로그의 카탈로그 ID(계정 ID)입니다. None인 경우 호출자의 기본 계정 ID가 사용됩니다.

AWS Glue PySpark 변환 참조

AWS Glue는 PySpark ETL 작업에 사용할 수 있는 다음과 같은 기본 제공 변환을 제공합니다. 데이터는 Apache Spark SQL의 `DynamicFrame` 확장인 `a`라는 데이터 구조를 통해 변환에서 변환으로 전달됩니다. `DataFrame` `DynamicFrame`은 데이터를 포함하고 데이터 스키마를 참조하여 데이터를 진행합니다.

이러한 변환의 대부분은 `DynamicFrame` 클래스의 메서드로도 존재합니다. [자세한 내용은 변환을 참조하십시오DynamicFrame](#).

- [GlueTransform 베이스 클래스](#)
- [ApplyMapping 클래스](#)
- [DropFields 클래스](#)
- [DropNullField 클래스](#)
- [ErrorsAsDynamicFrame 클래스](#)
- [EvaluateDataQuality 클래스](#)
- [FillMissingValues 클래스](#)

- [Filter 클래스](#)
- [FindIncrementalMatches 클래스](#)
- [FindMatches 클래스](#)
- [FlatMap 클래스](#)
- [Join 클래스](#)
- [Map 클래스](#)
- [MapToCollection 클래스](#)
- [mergeDynamicFrame](#)
- [Relationalize 클래스](#)
- [RenameField 클래스](#)
- [ResolveChoice 클래스](#)
- [SelectFields 클래스](#)
- [SelectFromCollection 클래스](#)
- [Simplify_ddb_json 클래스](#)
- [Spigot 클래스](#)
- [SplitFields 클래스](#)
- [SplitRows 클래스](#)
- [Unbox 클래스](#)
- [UnnestFrame 클래스](#)

GlueTransform 베이스 클래스

이 베이스 클래스에서 모든 `awsglue.transforms` 클래스가 물려받습니다.

이 클래스는 `__call__` 방법을 정의합니다. 이것들은 다음 섹션에 있는 GlueTransform 클래스 방법을 재정의하거나 기본값으로 클래스 이름을 사용하여 호출됩니다.

메서드

- [apply\(cls, *args, **kwargs\)](#)
- [name\(cls\)](#)
- [describeArgs\(cls\)](#)
- [describeReturn\(cls\)](#)

- [describeTransform\(cls\)](#)
- [describeErrors\(cls\)](#)
- [describe\(cls\)](#)

`apply(cls, *args, **kwargs)`

변환 클래스를 호출하여 변환을 적용하고 결과를 반환합니다.

- `cls` – `self` 클래스 객체입니다.

`name(cls)`

파생된 변환 클래스의 이름을 반환합니다.

- `cls` – `self` 클래스 객체입니다.

`describeArgs(cls)`

- `cls` – `self` 클래스 객체입니다.

명명된 논제와 관련된 딕셔너리 목록을 다음과 같은 포맷으로 반환합니다.

```
[
  {
    "name": "(name of argument)",
    "type": "(type of argument)",
    "description": "(description of argument)",
    "optional": "(Boolean, True if the argument is optional)",
    "defaultValue": "(Default value string, or None)(String; the default value, or None)"
  },
  ...
]
```

파생된 변환이 실행되지 않고 호출되지 않으면 `NotImplementedError` 예외가 발생합니다.

`describeReturn(cls)`

- `cls` – `self` 클래스 객체입니다.

반환 유형에 대한 정보와 함께 다음 포맷으로 딕셔너리를 반환합니다.

```
{
  "type": "(return type)",
  "description": "(description of output)"
}
```

파생된 변환이 실행되지 않고 호출되지 않으면 `NotImplementedError` 예외가 발생합니다.

`describeTransform(cls)`

변환을 나타내는 문자열을 반환합니다.

- `cls` – `self` 클래스 객체입니다.

파생된 변환이 실행되지 않고 호출되지 않으면 `NotImplementedError` 예외가 발생합니다.

`describeErrors(cls)`

- `cls` – `self` 클래스 객체입니다.

변환에 따른 예외를 나타내는 딕셔너리 목록을 다음과 같은 포맷으로 반환합니다.

```
[
  {
    "type": "(type of error)",
    "description": "(description of error)"
  },
  ...
]
```

`describe(cls)`

- `cls` – `self` 클래스 객체입니다.

다음의 형식으로 객체를 반환합니다.

```
{
  "transform" : {
    "name" : cls.name( ),
```

```

    "args" : cls.describeArgs( ),
    "returns" : cls.describeReturn( ),
    "raises" : cls.describeErrors( ),
    "location" : "internal"
  }
}

```

ApplyMapping 클래스

DynamicFrame에서 매핑을 적용합니다.

예

DynamicFrame에 매핑을 적용하기 위해서는 [DynamicFrame.apply_mapping\(\)](#) 메서드를 사용하는 것을 권장합니다. 코드에 대한 예제는 [예: apply_mapping을 사용하여 필드 이름을 바꾸고 필드 유형을 변경합니다.](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [설명](#)

`__call__(frame, mappings, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

지정 DynamicFrame에 서술식 매핑을 적용합니다.

- `frame` - DynamicFrame은 매핑을 적용할 대상(필수).
- `mappings` - 매핑 튜플 목록(필수). 각각 (소스 열, 소스 유형, 대상 열, 대상 유형)으로 구성된 매핑 튜플 목록입니다.

소스 열에 점(".")이 있는 경우 주위에 백틱("` `")을 넣어야 합니다. 예를 들어 `this.old.name`(문자 열)을 `thisNewName`에 매핑하려면 다음 튜플을 사용합니다.

```
("`this.old.name`", "string", "thisNewName", "string")
```

- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

“매핑” 튜플에 지정된 `DynamicFrame`의 필드만 반환합니다.

```
apply(cls, *args, **kwargs)
```

GlueTransform [apply](#)에서 상속됩니다.

```
name(cls)
```

GlueTransform [이름](#)에서 상속됩니다.

```
describeArgs(cls)
```

GlueTransform [describeArgs](#)에서 상속됩니다.

```
describeReturn(cls)
```

GlueTransform [describeReturn](#)에서 상속됩니다.

```
describeTransform(cls)
```

GlueTransform [describeTransform](#)에서 상속됩니다.

```
describeErrors(cls)
```

GlueTransform [describeErrors](#)에서 상속됩니다.

```
describe(cls)
```

GlueTransform [describe](#)에서 상속됩니다.

DropFields 클래스

DynamicFrame 내에서 필드를 드롭합니다.

예

DynamicFrame.drop_fields()에서 필드를 드롭하기 위해 [DynamicFrame](#) 방법을 사용하는 것이 좋습니다. 코드에 대한 예제는 [예: drop_fields를 사용하여 DynamicFrame에서 필드를 제거합니다.](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [설명](#)

`__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

DynamicFrame 내에서 노드를 드롭합니다.

- `frame` - DynamicFrame은 노드를 드롭합니다(필수).
- `paths` - 드롭할 노드의 전체 경로 목록입니다(필수).
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

지정된 필드없이 새로운 DynamicFrame을 반환합니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

DropNullField 클래스

DynamicFrame 에서 NullType 인 유형의 모든 널 필드를 드롭합니다. DynamicFrame 데이터 세트의 모든 기록에서 누락된 혹은 null 값인 필드입니다.

예

이 예제에서는 NullType 유형의 필드가 삭제된 새 DynamicFrame을 만드는 데 DropNullFields를 사용합니다. DropNullFields를 보여 주기 위해 이미 로드된 persons 데이터 세트에 null 유형의 empty_column라는 이름의 새 열을 추가합니다.

Note

이 예제에서 사용되는 데이터 세트에 액세스하려면 [코드 예: 데이터 조인 및 관계화](#) 항목을 참조하고 [1단계: Amazon S3 버킷에서 데이터 크롤](#)의 지침을 따르세요.

```
# Example: Use DropNullFields to create a new DynamicFrame without NullType fields

from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql.functions import lit
from pyspark.sql.types import NullType
from awsglue.dynamicframe import DynamicFrame
from awsglue.transforms import DropNullFields

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Create DynamicFrame
persons = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="persons_json"
)
print("Schema for the persons DynamicFrame:")
persons.printSchema()

# Add new column "empty_column" with NullType
persons_with_nulls = persons.toDF().withColumn("empty_column",
    lit(None).cast(NullType()))
persons_with_nulls_dyf = DynamicFrame.fromDF(persons_with_nulls, glueContext,
    "persons_with_nulls")
print("Schema for the persons_with_nulls_dyf DynamicFrame:")
persons_with_nulls_dyf.printSchema()

# Remove the NullType field
persons_no_nulls = DropNullFields.apply(persons_with_nulls_dyf)
print("Schema for the persons_no_nulls DynamicFrame:")
persons_no_nulls.printSchema()
```

출력

```
Schema for the persons DynamicFrame:
root
 |-- family_name: string
 |-- name: string
 |-- links: array
 |    |-- element: struct
 |    |    |-- note: string
 |    |    |-- url: string
```



```

|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string

```

Schema for the persons_with_nulls_dyf DynamicFrame:

```

root
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string

```

```
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
|-- death_date: string
|-- empty_column: null
```

```
null_fields ['empty_column']
```

```
Schema for the persons_no_nulls DynamicFrame:
```

```
root
```

```
|-- family_name: string
|-- name: string
|-- links: array
|   |-- element: struct
|   |   |-- note: string
|   |   |-- url: string
|-- gender: string
|-- image: string
|-- identifiers: array
|   |-- element: struct
|   |   |-- scheme: string
|   |   |-- identifier: string
|-- other_names: array
|   |-- element: struct
|   |   |-- lang: string
|   |   |-- note: string
|   |   |-- name: string
|-- sort_name: string
|-- images: array
|   |-- element: struct
|   |   |-- url: string
|-- given_name: string
|-- birth_date: string
|-- id: string
|-- contact_details: array
|   |-- element: struct
|   |   |-- type: string
|   |   |-- value: string
```

```
|-- death_date: string
```

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [설명](#)

`__call__(frame, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

DynamicFrame 에서 NullType 인 유형의 모든 널 필드를 드롭합니다. DynamicFrame 데이터 세트의 모든 기록에서 누락된 혹은 null 값인 필드입니다.

- `frame` - DynamicFrame은 null 필드를 드롭합니다(필수).
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

널값이 없는 새로운 DynamicFrame을 반환합니다.

`apply(cls, *args, **kwargs)`

- `cls` - cls

`name(cls)`

- `cls` - cls

describeArgs(cls)

- cls – cls

describeReturn(cls)

- cls – cls

describeTransform(cls)

- cls – cls

describeErrors(cls)

- cls – cls

describe(cls)

- cls – cls

ErrorsAsDynamicFrame 클래스

소스 DynamicFrame이 생성되는 동안 발생한 오류에 대한 중첩 레코드가 포함된 DynamicFrame을 반환합니다.

예

오류 레코드를 조회하고 보는 데는 [DynamicFrame.errorsAsDynamicFrame\(\)](#) 메서드를 권장합니다. 코드에 대한 예제는 [예: errorsAsDynamicFrame을 사용하여 오류 레코드 보기](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)

- [describeErrors](#)
- [설명](#)

`__call__(frame)`

DynamicFrame 소스 생성과 관련된 중첩된 오류 기록을 포함하는 DynamicFrame을 반환합니다.

- `frame` – 소스 DynamicFrame입니다(필수).

`apply(cls, *args, **kwargs)`

- `cls` – cls

`name(cls)`

- `cls` – cls

`describeArgs(cls)`

- `cls` – cls

`describeReturn(cls)`

- `cls` – cls

`describeTransform(cls)`

- `cls` – cls

`describeErrors(cls)`

- `cls` – cls

`describe(cls)`

- `cls` – cls

EvaluateDataQuality 클래스

DynamicFrame을 기준으로 데이터 품질 규칙 세트를 평가하고 새 DynamicFrame을 평가 결과와 함께 반환합니다.

예

다음 예제 코드는 DynamicFrame의 데이터 품질을 평가한 다음 데이터 품질 결과를 확인하는 방법을 보여줍니다.

```
from awsglue.transforms import *
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsgluedq.transforms import EvaluateDataQuality

#Create Glue context
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Define DynamicFrame
legislatorsAreas = glueContext.create_dynamic_frame.from_catalog(
    database="legislators", table_name="areas_json")

# Create data quality ruleset
ruleset = """"Rules = [ColumnExists "id", IsComplete "id"]""""

# Evaluate data quality
dqResults = EvaluateDataQuality.apply(
    frame=legislatorsAreas,
    ruleset=ruleset,
    publishing_options={
        "dataQualityEvaluationContext": "legislatorsAreas",
        "enableDataQualityCloudWatchMetrics": True,
        "enableDataQualityResultsPublishing": True,
        "resultsS3Prefix": "DOC-EXAMPLE-BUCKET1",
    },
)

# Inspect data quality results
dqResults.printSchema()
dqResults.toDF().show()
```

출력

```

root
|-- Rule: string
|-- Outcome: string
|-- FailureReason: string
|-- EvaluatedMetrics: map
|   |-- keyType: string
|   |-- valueType: double

```

```

+-----+-----+-----+-----+
|Rule           |Outcome|FailureReason|EvaluatedMetrics          |
+-----+-----+-----+-----+
|ColumnExists "id"  |Passed |null         |{}                         |
|IsComplete "id"   |Passed |null         |{Column.first_name.Completeness -> 1.0}|
+-----+-----+-----+-----+

```

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__(frame, ruleset, publishing_options = {})`

- `frame` - 데이터 품질을 평가하려는 `DynamicFrame`입니다.
- `ruleset` - 문자열 형식의 DQDL(데이터 품질 정의 언어) 규칙 세트입니다. DQDL에 대한 자세한 내용은 [데이터 품질 정의 언어\(DQDL\) 참조](#) 안내서를 참조하세요.
- `publishing_options` - 평가 결과 및 지표를 게시하기 위해 다음 옵션을 지정하는 사전입니다.

- `dataQualityEvaluationContext` - AWS Glue가 Amazon CloudWatch 지표와 데이터 품질 결과를 게시할 네임스페이스를 지정하는 문자열입니다. 집계된 지표는 CloudWatch에 표시되고 전체 결과는 AWS Glue Studio 인터페이스에 표시됩니다.
 - 필수 항목 여부: 아니요
 - 기본 값: `default_context`
- `enableDataQualityCloudWatchMetrics` - 데이터 품질 평가 결과를 CloudWatch에 게시할 지 여부를 지정합니다. `dataQualityEvaluationContext` 옵션을 사용하여 지표의 네임스페이스를 지정합니다.
 - 필수 항목 여부: 아니요
 - 기본 값: `False`
- `enableDataQualityResultsPublishing` - AWS Glue Studio 인터페이스의 Data Quality(데이터 품질) 탭에 데이터 품질 결과를 표시할지 여부를 지정합니다.
 - 필수 항목 여부: 아니요
 - 기본 값: `True`
- `resultsS3Prefix` - AWS Glue가 데이터 품질 평가 결과를 기록할 수 있는 Amazon S3 위치를 지정합니다.
 - 필수 항목 여부: 아니요
 - 기본 값: `""`(빈 문자열)

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

FillMissingValues 클래스

FillMissingValues 클래스는 지정된 DynamicFrame에서 null 값과 빈 문자열을 찾고 선형 회귀 및 랜덤 포레스트와 같은 기계 학습 방법을 사용하여 누락된 값을 예측합니다. ETL 작업은 입력 데이터 집합의 값을 사용하여 기계 학습 모델을 훈련합니다. 이 모델은 누락된 값이 무엇인지 예측합니다.

Tip

중분 데이터 집합을 사용하는 경우 각 중분 집합은 기계 학습 모델의 훈련 데이터로 사용되므로 결과가 정확하지 않을 수 있습니다.

가져오려면

```
from awsglueml.transforms import FillMissingValues
```

메서드

- [Apply](#)

```
apply(frame, missing_values_column, output_column = "", transformation_ctx = "", info = "",
stageThreshold = 0, totalThreshold = 0)
```

지정된 열에 동적 프레임의 누락된 값을 채우고 새 열에 추정 값과 함께 새 프레임을 반환합니다. 누락된 값이 없는 행의 경우 지정된 열의 값이 새 열에 복제됩니다.

- `frame` - 누락된 값을 채울 DynamicFrame입니다 필수 사항입니다.
- `missing_values_column` - 누락된 값(null 값 및 빈 문자열)이 포함된 열입니다. 필수 사항입니다.
- `output_column` - 값이 누락된 모든 행에 대한 예상 값을 포함할 새 열의 이름입니다. 선택 사항이며 기본값은 "_filled"가 접미사로 사용된 `missing_values_column`의 이름입니다.

- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항, 기본값은 0).
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항, 기본값은 0).

누락된 값이 있는 행에 대한 추정 값과 다른 행에 대한 현재 값이 포함된 하나의 추가 열이 있는 새 `DynamicFrame`을 반환합니다.

Filter 클래스

지정된 술어 함수를 충족하는 `DynamicFrame` 입력의 레코드를 포함하는 새 `DynamicFrame` 항목을 빌드합니다.

예

`DynamicFrame`에서 레코드를 필터링하기 위해 [DynamicFrame.filter\(\)](#) 방법을 사용하는 것이 좋습니다. 코드에 대한 예제는 [예: 필터를 사용하여 필터링된 필드 선택 가져오기](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

```
__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0))
```

지정된 조건자 함수를 만족하는 입력 `DynamicFrame`에서 기록을 선택하여 만든 새로운 `DynamicFrame`을 반환합니다.

- `frame` - 소스 `DynamicFrame`으로 지정된 필터 함수를 적용합니다(필수).

- `f` - `DynamicFrame`의 각 `DynamicRecord`를 적용하는 조건자 함수입니다. 함수는 `DynamicRecord`를 인수로 받아들이고 `DynamicRecord`가 필터 요구 사항과 맞으면 `True`을 반환하고 아니면 `False`를 반환합니다 (필수).

`DynamicRecord`는 `DynamicFrame`내 논리적 기록입니다. 자기 설명적이고 고정 스키마를 준수하지 않은 데이터에 사용될 수 있다는 점을 제외하면 스파크 `DataFrame`의 열과 비슷합니다.

- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

FindIncrementalMatches 클래스

기존 및 증분 DynamicFrame에서 일치하는 레코드를 식별하고 일치하는 레코드의 각 그룹에 할당된 고유 식별자로 새 DynamicFrame을 생성합니다.

가져오려면

```
from awsglueml.transforms import FindIncrementalMatches
```

메서드

- [Apply](#)

```
apply(existingFrame, incrementalFrame, transformId, transformation_ctx = "", info = "",
stageThreshold = 0, totalThreshold = 0, enforcedMatches = none, computeMatchConfidenceScores =
0)
```

입력 DynamicFrame에서 일치하는 레코드를 식별하고 일치하는 레코드의 각 그룹에 할당된 고유 식별자를 사용하여 새 DynamicFrame을 생성합니다.

- `existingFrame` - FindIncrementalMatches 변환을 적용하기 위한 기존 및 사전 일치 DynamicFrame입니다. 필수 사항입니다.
- `incrementalFrame` - `existingFrame`과 일치하도록 FindIncrementalMatches 변환을 적용하는 증분 DynamicFrame입니다. 필수 사항입니다.
- `transformId` - DynamicFrames의 레코드에 적용할 FindIncrementalMatches 변환과 연결된 고유 ID입니다. 필수 사항입니다.
- `transformation_ctx` - 고유 문자열을 통해 통계/상태 정보를 확인합니다. 선택 사항입니다.
- `info` - 변환에 따른 오류 관련 문자열입니다. 선택 사항입니다.
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다. 선택 사항입니다. 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다. 선택 사항입니다. 기본값은 0입니다.
- `enforcedMatches` - 일치를 적용하는 데 사용되는 DynamicFrame입니다. 선택 사항입니다. 기본값은 없습니다.
- `computeMatchConfidenceScores` - 각 일치 레코드 그룹의 신뢰도 점수를 컴퓨팅할지 여부를 나타내는 부울 값입니다. 선택 사항입니다. 기본값은 false입니다.

일치하는 레코드의 각 그룹에 고유 식별자가 할당된 새 DynamicFrame을 반환합니다.

FindMatches 클래스

입력 DynamicFrame에서 일치하는 레코드를 식별하고 일치하는 레코드의 각 그룹에 할당된 고유 식별자를 사용하여 새 DynamicFrame을 생성합니다.

가져오려면

```
from awsglueml.transforms import FindMatches
```

메서드

- [Apply](#)

```
apply(frame, transformId, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0,
enforcedMatches = none, computeMatchConfidenceScores = 0)
```

입력 DynamicFrame에서 일치하는 레코드를 식별하고 일치하는 레코드의 각 그룹에 할당된 고유 식별자를 사용하여 새 DynamicFrame을 생성합니다.

- `frame` - FindMatches 변환을 적용할 DynamicFrame입니다. 필수 사항입니다.
- `transformId` - DynamicFrame의 레코드에 적용할 FindMatches 변환과 연결된 고유 ID입니다. 필수 사항입니다.
- `transformation_ctx` - 고유 문자열을 통해 통계/상태 정보를 확인합니다. 선택 사항입니다.
- `info` - 변환에 따른 오류 관련 문자열입니다. 선택 사항입니다.
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다. 선택 사항입니다. 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다. 선택 사항입니다. 기본값은 0입니다.
- `enforcedMatches` - 일치를 적용하는 데 사용되는 DynamicFrame입니다. 선택 사항입니다. 기본값은 없습니다.
- `computeMatchConfidenceScores` - 각 일치 레코드 그룹의 신뢰도 점수를 컴퓨팅할지 여부를 나타내는 부울 값입니다. 선택 사항입니다. 기본값은 false입니다.

일치하는 레코드의 각 그룹에 고유 식별자가 할당된 새 DynamicFrame을 반환합니다.

FlatMap 클래스

컬렉션에서 각 DynamicFrame에 변환을 적용합니다. 결과는 하나의 DynamicFrame으로 병합되지 않고 컬렉션으로 유지됩니다.

FlatMap 예제

다음 스니펫 예제에서는 FlatMap에 적용할 때 동적 프레임 컬렉션에서 ResolveChoice 변환을 사용하는 방법을 보여줍니다. 입력에 사용되는 데이터는 JSON에서 Amazon S3 주소 s3://bucket/path-for-data/sample.json 자리 표시자에 있으며, 다음 데이터를 포함합니다.

예시 JSON 데이터

```
[{
  "firstname": "Arnav",
  "lastname": "Desai",
  "address": {
    "street": "6 Anyroad Avenue",
    "city": "London",
    "state": "England",
    "country": "UK"
  },
  "phone": 17235550101,
  "affiliations": [
    "General Anonymous Example Products",
    "Example Independent Research",
    "Government Department of Examples"
  ]
},
{
  "firstname": "Mary",
  "lastname": "Major",
  "address": {
    "street": "7821 Spot Place",
    "city": "Centerville",
    "state": "OK",
    "country": "US"
  },
  "phone": 19185550023,
  "affiliations": [
    "Example Dot Com",
    "Example Independent Research",
    "Example.io"
  ]
}
```

```

    ]
  },
  {
    "firstname": "Paulo",
    "lastname": "Santos",
    "address": {
      "street": "123 Maple Street",
      "city": "London",
      "state": "Ontario",
      "country": "CA"
    },
    "phone": 12175550181,
    "affiliations": [
      "General Anonymous Example Products",
      "Example Dot Com"
    ]
  }
]
]]

```

Example ResolveChoice를 DynamicFrameCollection에 적용하고 출력을 표시합니다.

```

#Read DynamicFrame
datasource = glueContext.create_dynamic_frame_from_options("s3", connection_options =
  {"paths":["s3://bucket/path/to/file/mysamplejson.json"]}, format="json")
datasource.printSchema()
datasource.show()

## Split to create a DynamicFrameCollection
split_frame=datasource.split_fields(["firstname","lastname","address"],"personal_info","business_info")
split_frame.keys()
print("---")

## Use FlatMap to run ResolveChoice
kwargs = {"choice": "cast:string"}
flat = FlatMap.apply(split_frame, ResolveChoice, frame_name="frame",
  transformation_ctx='tcx', **kwargs)
flat.keys()

##Select one of the DynamicFrames
personal_info = flat.select("personal_info")
personal_info.printSchema()
personal_info.show()
print("---")

```

```
business_info = flat.select("business_info")
business_info.printSchema()
business_info.show()
```

⚠ Important

FlatMap.apply 직접 호출 시 frame_name 파라미터는 반드시 "frame"이어야 합니다. 현재 다른 값은 허용되지 않습니다.

출력 예시

```
root
|-- firstname: string
|-- lastname: string
|-- address: struct
|   |-- street: string
|   |-- city: string
|   |-- state: string
|   |-- country: string
|-- phone: long
|-- affiliations: array
|   |-- element: string
---
{
  "firstname": "Mary",
  "lastname": "Major",
  "address": {
    "street": "7821 Spot Place",
    "city": "Centerville",
    "state": "OK",
    "country": "US"
  },
  "phone": 19185550023,
  "affiliations": [
    "Example Dot Com",
    "Example Independent Research",
    "Example.io"
  ]
}
```



```
"firstname": "Paulo",
"lastname": "Santos",
"address": {
  "street": "123 Maple Street",
  "city": "London",
  "state": "Ontario",
  "country": "CA"
},
"phone": 12175550181,
"affiliations": [
  "General Anonymous Example Products",
  "Example Dot Com"
]
}
---
root
|-- firstname: string
|-- lastname: string
|-- address: struct
|   |-- street: string
|   |-- city: string
|   |-- state: string
|   |-- country: string
{
  "firstname": "Mary",
  "lastname": "Major",
  "address": {
    "street": "7821 Spot Place",
    "city": "Centerville",
    "state": "OK",
    "country": "US"
  }
}
{
  "firstname": "Paulo",
  "lastname": "Santos",
  "address": {
    "street": "123 Maple Street",
    "city": "London",
    "state": "Ontario",
    "country": "CA"
  }
}
```

```
}
---
root
|-- phone: long
|-- affiliations: array
|   |-- element: string

{
  "phone": 19185550023,
  "affiliations": [
    "Example Dot Com",
    "Example Independent Research",
    "Example.io"
  ]
}

{
  "phone": 12175550181,
  "affiliations": [
    "General Anonymous Example Products",
    "Example Dot Com"
  ]
}
```

메서드

- [__call__](#)
- [Apply](#)
- [명칭](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [설명](#)

`__call__(dfc, BaseTransform, frame_name, transformation_ctx = "", **base_kwargs)`

컬렉션의 각 `DynamicFrame`에 변환을 적용하고 결과를 플랫폼합니다.

- `dfc` – 플랫폼할 `DynamicFrameCollection`입니다(필수).

- BaseTransform – GlueTransform로부터 얻어진 변환을 사용하여 컬렉션의 각 멤버에게 적용합니다(필수).
- frame_name - 컬렉션의 요소를 전달하기 위한 인수 이름입니다(필수).
- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- base_kwargs - 기본 변환으로 전달을 위한 인수입니다(필수).

DynamicFrameCollection 원본의 변환을 각 DynamicFrame에 적용하여 생성한 새로운 DynamicFrameCollection을 반환합니다.

apply(cls, *args, **kwargs)

GlueTransform [apply](#)에서 상속됩니다.

name(cls)

GlueTransform [이름](#)에서 상속됩니다.

describeArgs(cls)

GlueTransform [describeArgs](#)에서 상속됩니다.

describeReturn(cls)

GlueTransform [describeReturn](#)에서 상속됩니다.

describeTransform(cls)

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

Join 클래스

두 개의 DynamicFrames에 동일한 조인을 실행합니다.

예

DynamicFrames에 조인하려면 [DynamicFrame.join\(\)](#) 메서드를 사용하는 것이 좋습니다. 코드에 대한 예제는 [예: 조인을 사용하여 DynamicFrames 결합](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [설명](#)

```
__call__(frame1, frame2, keys1, keys2, transformation_ctx = "")
```

두 개의 DynamicFrames에 동일한 조인을 실행합니다.

- frame1 - 조인할 첫 번째 DynamicFrame입니다(필수).
- frame2 - 조인할 두 번째 DynamicFrame입니다(필수).
- keys1 - 첫 번째 프레임을 조인할 키입니다(필수).
- keys2 - 두 번째 프레임을 조인할 키입니다(필수).
- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).

두 DynamicFrames를 조인하여 생성된 새 DynamicFrame 항목을 반환합니다.

```
apply(cls, *args, **kwargs)
```

GlueTransform [apply](#)에서 상속됩니다.

```
name(cls)
```

GlueTransform [이름](#)에서 상속됩니다.

describeArgs(cls)

GlueTransform [describeArgs](#)에서 상속됩니다.

describeReturn(cls)

GlueTransform [describeReturn](#)에서 상속됩니다.

describeTransform(cls)

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

Map 클래스

입력 DynamicFrame의 모든 기록에 함수를 적용하여 새로운 DynamicFrame을 설계합니다.

예

DynamicFrame의 모든 레코드에 함수를 적용하는 [DynamicFrame.map\(\)](#) 메서드를 사용하는 것이 좋습니다. 코드에 대한 예제는 [예: map을 사용하여 DynamicFrame의 모든 레코드에 함수를 적용합니다.](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)

- [설명](#)

```
__call__(frame, f, transformation_ctx="", info="", stageThreshold=0, totalThreshold=0)
```

기존 DynamicFrame의 모든 DynamicRecords에 지정된 함수를 적용한 결과로써 새로운 DynamicFrame을 반환합니다.

- frame - 원본 DynamicFrame은 매핑 함수를 적용합니다(필수).
- f - DynamicFrame에 모든 DynamicRecords를 적용하려는 함수입니다. 함수는 매핑 결과에 따라 DynamicRecord를 논증으로 받아들이고 새로운 DynamicRecord를 반환합니다(필수).

DynamicRecord는 DynamicFrame내 논리적 기록입니다. 자기 설명적이고 고정 스키마를 준수하지 않은 데이터에 사용될 수 있다는 점을 제외하면 Apache Spark DataFrame의 열과 비슷합니다.

- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- stageThreshold - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- totalThreshold - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

기존 DynamicFrame의 모든 DynamicRecords에 지정된 함수를 적용한 결과로써 새로운 DynamicFrame을 반환합니다.

```
apply(cls, *args, **kwargs)
```

GlueTransform [apply](#)에서 상속됩니다.

```
name(cls)
```

GlueTransform [이름](#)에서 상속됩니다.

```
describeArgs(cls)
```

GlueTransform [describeArgs](#)에서 상속됩니다.

```
describeReturn(cls)
```

GlueTransform [describeReturn](#)에서 상속됩니다.

describeTransform(cls)

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

MapToCollection 클래스

지정된 DynamicFrameCollection의 각 DynamicFrame에 변환을 적용합니다.

메서드

- [__call__](#)
- [Apply](#)
- [명칭](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [설명](#)

`__call__(dfc, BaseTransform, frame_name, transformation_ctx = "", **base_kwargs)`

지정된 DynamicFrameCollection의 각 DynamicFrame에 변환 함수를 적용합니다.

- `dfc` - 원본 DynamicFrameCollection은 변환 함수를 적용합니다(필수).
- `callable` - 호출 가능한 변환 함수는 각 컬렉션의 수에 적용됩니다(필수).
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).

DynamicFrameCollection 원본의 변환을 각 DynamicFrame에 적용하여 생성한 새로운 DynamicFrameCollection을 반환합니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

Relationalize 클래스

중첩된 스키마를 DynamicFrame에 평면화하고 평면화된 프레임에서 배열의 열을 회전합니다.

예

DynamicFrame을 관계형으로 만들려면 [DynamicFrame.relationalize\(\)](#) 메서드를 사용하는 것이 좋습니다. 코드에 대한 예제는 [예: 관계형 만들기를 사용하여 DynamicFrame 내 중첩된 스키마를 플랫폼하게 만들기](#) 단원을 참조하세요.

메서드

- [__call__](#)

- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

```
__call__(frame, staging_path=None, name='roottable', options=None, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)
```

DynamicFrame를 관계화하고 중첩되지 않은 열과 회전되는 배열 열이 생성한 프레임 목록을 생산합니다. 회전된 배열 열은 중첩되지 않는 상태에서 생성된 조인 키를 사용하여 루트 테이블에 연결될 수 있습니다.

- `frame` - 관계화할 DynamicFrame입니다(필수).
- `staging_path` - 메소드가 CSV 포맷으로 회전 테이블의 파티션을 저장하는 경로입니다(선택 사항). 회전 테이블은 이 경로부터 다시 읽습니다.
- `name` - 루트 테이블의 이름입니다(선택 사항).
- `options` - 선택적 파라미터의 딕셔너리입니다. 현재 사용되지 않습니다.
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

```
apply(cls, *args, **kwargs)
```

GlueTransform [apply](#)에서 상속됩니다.

```
name(cls)
```

GlueTransform [이름](#)에서 상속됩니다.

describeArgs(cls)

GlueTransform [describeArgs](#)에서 상속됩니다.

describeReturn(cls)

GlueTransform [describeReturn](#)에서 상속됩니다.

describeTransform(cls)

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

RenameField 클래스

DynamicFrame 내에서 노드 이름을 바꿉니다.

예

DynamicFrame에서 필드 이름을 다시 지정하기 위해 [DynamicFrame.rename_field\(\)](#) 메소드를 사용하는 것이 좋습니다. 코드에 대한 예제는 [예: rename_field를 사용하여 DynamicFrame의 필드 이름을 변경합니다](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)

- [describe](#)

```
__call__(frame, old_name, new_name, transformation_ctx = "", info = "", stageThreshold = 0,
totalThreshold = 0)
```

DynamicFrame 내에서 노드 이름을 바꿉니다.

- `frame` - DynamicFrame은 노드 이름을 바꿉니다(필수).
- `old_name` - 이름을 바꿀 노드의 전체 경로입니다(필수).

기존 이름에 점이 있으면 그 주위로 역음 부호(` `)를 하지 않는 한 RenameField는 실행되지 않습니다. 예를 들어, `this.old.name`를 `thisNewName`로 바꾸려면 다음과 같이 RenameField를 불러올 수 있습니다.

```
newDyF = RenameField(oldDyF, "`this.old.name`", "thisNewName")
```

- `new_name` - 전체 경로를 포함한 새로운 이름입니다(필수).
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

```
apply(cls, *args, **kwargs)
```

GlueTransform [apply](#)에서 상속됩니다.

```
name(cls)
```

GlueTransform [이름](#)에서 상속됩니다.

```
describeArgs(cls)
```

GlueTransform [describeArgs](#)에서 상속됩니다.

```
describeReturn(cls)
```

GlueTransform [describeReturn](#)에서 상속됩니다.

describeTransform(cls)

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

ResolveChoice 클래스

선택 유형을 DynamicFrame 내에서 해결합니다.

예

[DynamicFrame.resolveChoice\(\)](#) 메서드를 사용하여 DynamicFrame에 여러 유형이 포함된 필드를 처리하는 것이 좋습니다. 코드에 대한 예제는 [예: resolveChoice를 사용하여 여러 유형이 포함된 열을 처리합니다](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__(frame, specs = none, choice = "", transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)`

모호한 유형을 DynamicFrame 내에서 해결하기 위한 정보를 제공합니다. 결과 DynamicFrame를 반환합니다.

- `frame` - `DynamicFrame`은 선택 유형을 해결합니다(필수).
- `specs` - 해결할 특정 모호성 목록이며, 각각은 (`path`, `action`)의 튜플 형식입니다. `path` 가치는 특정 모호한 요소를 확인하고 `action` 가치는 관련 해결 방안을 제안합니다.

`spec` 및 `choice` 파라미터 중 하나만 사용할 수 있습니다. `spec` 파라미터가 `None`이 아니면 `choice` 파라미터는 빈 문자열이어야 합니다. 반대로 `choice` 파라미터가 빈 문자열이 아니면, `spec` 파라미터는 `None`이어야 합니다. 어떠한 파라미터도 제공되지 않을 경우, AWS Glue는 스키마를 구문분석하고 모호함을 해결하기 위해 사용합니다.

`specs` 튜플의 `action` 부분에서 다음 해결 전략 중 하나를 지정할 수 있습니다.

- `cast` - 보내버릴 유형을 명시할 수 있도록 도와줍니다. (예를 들어, `cast:int`).
- `make_cols` - 데이터를 평면화하여 잠재적 모호성을 해결합니다. 예를 들어, `columnA`가 `int` 혹은 `string`이면 해결 방안은 `DynamicFrame`에서 `columnA_int`와 `columnA_string`으로 된 두 열을 생산하는 것입니다.
- `make_struct` - 데이터를 나타내도록 구조를 사용하여 잠재적 모호성을 해결합니다. 예를 들어, 열에서 데이터가 `int` 혹은 `string`이면 `make_struct` 작업은 각 `DynamicFrame`와 `int`을 포함하는 결과인 `string`의 구조 열을 생성합니다.
- `project` - 결과 `DynamicFrame`에 지정된 유형의 값만 유지하여 잠재적 모호성을 해결합니다. 예를 들어 `ChoiceType` 열에서 데이터가 `int` 혹은 `string`이면 `project:string` 작업은 `string` 유형이 아닌 결과 `DynamicFrame`의 값을 삭제합니다.

만약 `path`가 배열을 확인하면, 빈 대괄호를 배열 이름 다음에 만들어 모호성을 피합니다. 예를 들어, 다음과 같은 구조화된 데이터와 작업한다고 가정합니다.

```
"myList": [
  { "price": 100.00 },
  { "price": "$100.00" }
]
```

`path`를 `"myList[].price"`로 설정하고, `action`을 `"cast:double"`로 설정하여 가격의 문자열 버전보다 숫자 버전을 선택할 수 있습니다.

- `choice` - `specs` 파라미터가 `None`인 경우의 기본 해결 작업입니다. `specs` 파라미터가 `None`이 아니면 이것은 어떠한 것에도 설정되지 않고 빈 문자열이어야 합니다.

이 인수는 앞에서 설명된 `specs` 작업 외에 다음 작업을 지원합니다.

- `MATCH_CATALOG` - 각 `ChoiceType`을 지정된 Data Catalog 테이블의 해당 유형에 캐스팅해봅니다.

- `database` — `MATCH_CATALOG` 선택 항목과 함께 사용할 AWS Glue 데이터 카탈로그 데이터베이스입니다(`MATCH_CATALOG`에 대해 필수).
- `table_name`— `MATCH_CATALOG` 작업에 사용할 AWS Glue 데이터 카탈로그 테이블 이름입니다 (`MATCH_CATALOG`에 대해 필수).
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

SelectFields 클래스

SelectFields 클래스는 기존 DynamicFrame에서 새 DynamicFrame 항목을 만들고 지정한 필드만 유지합니다. SelectFields는 SQL SELECT 문과 유사한 기능을 제공합니다.

예

DynamicFrame에서 필드를 선택하기 위해 [DynamicFrame.select_fields\(\)](#) 방법을 사용하는 것이 좋습니다. 코드에 대한 예제는 [예: select_fields를 사용하여 선택한 필드로 새 DynamicFrame 생성 단원을 참조하세요.](#)

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [설명](#)

```
__call__(frame, paths, transformation_ctx = "", info = "", stageThreshold = 0, totalThreshold = 0)
```

DynamicFrame의 필드(노드)를 얻습니다.

- `frame` - DynamicFrame에서 필드를 선택합니다(필수).
- `paths` - 선택할 필드의 전체 경로 목록입니다(필수).
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

특정 필드만 포함한 새로운 DynamicFrame을 반환합니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

SelectFromCollection 클래스

DynamicFrameCollection에서 DynamicFrame 하나를 선택합니다.

예

이 예제에서는 SelectFromCollection을 사용하여 DynamicFrameCollection에서 DynamicFrame를 선택합니다.

예제 데이터 세트

예제에서는 `split_rows_collection`을 호출한 DynamicFrameCollection에서 두 개의 DynamicFrames을 선택합니다. 다음은 `split_rows_collection`에서의 키 목록입니다.

```
dict_keys(['high', 'low'])
```


예제 코드

```
# Example: Use SelectFromCollection to select
# DynamicFrames from a DynamicFrameCollection

from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.transforms import SelectFromCollection

# Create GlueContext
sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

# Select frames and inspect entries
frame_low = SelectFromCollection.apply(dfc=split_rows_collection, key="low")
frame_low.toDF().show()

frame_high = SelectFromCollection.apply(dfc=split_rows_collection, key="high")
frame_high.toDF().show()
```

출력

```
+---+-----+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+-----+-----+
| 1| 0| fax| 202-225-3307|
| 1| 1| phone| 202-225-5731|
| 2| 0| fax| 202-225-3307|
| 2| 1| phone| 202-225-5731|
| 3| 0| fax| 202-225-3307|
| 3| 1| phone| 202-225-5731|
| 4| 0| fax| 202-225-3307|
| 4| 1| phone| 202-225-5731|
| 5| 0| fax| 202-225-3307|
| 5| 1| phone| 202-225-5731|
| 6| 0| fax| 202-225-3307|
| 6| 1| phone| 202-225-5731|
| 7| 0| fax| 202-225-3307|
| 7| 1| phone| 202-225-5731|
| 8| 0| fax| 202-225-3307|
| 8| 1| phone| 202-225-5731|
| 9| 0| fax| 202-225-3307|
| 9| 1| phone| 202-225-5731|
```

```

| 10|    0|          fax|      202-225-6328|
| 10|    1|          phone|     202-225-4576|
+---+-----+-----+-----+
only showing top 20 rows

+---+-----+-----+-----+
| id|index|contact_details.val.type|contact_details.val.value|
+---+-----+-----+-----+
| 11|    0|          fax|      202-225-6328|
| 11|    1|          phone|     202-225-4576|
| 11|    2|        twitter|    RepTrentFranks|
| 12|    0|          fax|      202-225-6328|
| 12|    1|          phone|     202-225-4576|
| 12|    2|        twitter|    RepTrentFranks|
| 13|    0|          fax|      202-225-6328|
| 13|    1|          phone|     202-225-4576|
| 13|    2|        twitter|    RepTrentFranks|
| 14|    0|          fax|      202-225-6328|
| 14|    1|          phone|     202-225-4576|
| 14|    2|        twitter|    RepTrentFranks|
| 15|    0|          fax|      202-225-6328|
| 15|    1|          phone|     202-225-4576|
| 15|    2|        twitter|    RepTrentFranks|
| 16|    0|          fax|      202-225-6328|
| 16|    1|          phone|     202-225-4576|
| 16|    2|        twitter|    RepTrentFranks|
| 17|    0|          fax|      202-225-6328|
| 17|    1|          phone|     202-225-4576|
+---+-----+-----+-----+
only showing top 20 rows

```

메서드

- [_call_](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)

- [describe](#)

```
__call__(dfc, key, transformation_ctx = "")
```

DynamicFrameCollection에서 DynamicFrame 하나를 가져옵니다.

- `dfc` - DynamicFrame을 선택해야 하는 DynamicFrameCollection입니다(필수).
- `key` - 선택할 DynamicFrame의 키입니다(필수).
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).

```
apply(cls, *args, **kwargs)
```

GlueTransform [apply](#)에서 상속됩니다.

```
name(cls)
```

GlueTransform [이름](#)에서 상속됩니다.

```
describeArgs(cls)
```

GlueTransform [describeArgs](#)에서 상속됩니다.

```
describeReturn(cls)
```

GlueTransform [describeReturn](#)에서 상속됩니다.

```
describeTransform(cls)
```

GlueTransform [describeTransform](#)에서 상속됩니다.

```
describeErrors(cls)
```

GlueTransform [describeErrors](#)에서 상속됩니다.

```
describe(cls)
```

GlueTransform [describe](#)에서 상속됩니다.

Simplify_ddb_json 클래스

DynamicFrame의 중첩된 열 중 특히 DynamoDB JSON 구조에 있는 열을 단순화하고 단순화된 새로운 DynamicFrame을 반환합니다.

예

특히 DynamoDB JSON 구조에 있는 `DynamicFrame`의 중첩 열을 단순화하려면 `DynamicFrame.simplify_ddb_json()` 메서드를 사용하는 것이 좋습니다. 코드에 대한 예제는 [예제: simplify_ddb_json을 사용하여 DynamoDB JSON 단순화 호출](#) 단원을 참조하세요.

Spigot 클래스

AWS Glue 작업에 의해 수행된 변환을 확인하는 데 도움이 되도록 샘플 레코드를 지정된 대상에 기록합니다.

예

`DynamicFrame.spigot()` 메서드를 사용하여 `DynamicFrame`에서 지정한 대상에 레코드의 하위 집합을 쓰는 것이 좋습니다. 코드에 대한 예제는 [예: spigot을 사용하여 DynamicFrame에서 Amazon S3에 샘플 필드를 작성합니다](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

```
__call__(frame, path, options, transformation_ctx = "")
```

변환 중에 특정 대상 주소에 샘플 기록을 씁니다.

- `frame` - `spigot`으로 `DynamicFrame`입니다(필수).
- `path` - 작성할 기본 대상 주소 경로입니다(필수).
- `options` - 옵션을 지정하는 JSON 키-값 페어입니다(선택 사항). "`topk`" 옵션은 첫 번째 `k` 레코드가 작성되어야 한다는 것을 명시합니다. "`prob`" 옵션은 특정 레코드를 선택할 확률(10진수)을 명시합니다. 작성할 레코드를 선택할 때 사용합니다.

- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

SplitFields 클래스

지정된 필드로 `DynamicFrame`을 두 개의 새로운 것으로 쪼갭니다.

예

`DynamicFrame`에서 레코드를 분할하기 위해 [DynamicFrame.split_fields\(\)](#) 메소드를 사용하는 것이 좋습니다. 코드에 대한 예제는 [예제: split_fields를 사용하여 선택한 필드를 별도의 DynamicFrame로 분할합니다](#) 단원을 참조하세요.

메서드

- [__call__](#)

- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

```
__call__(frame, paths, name1 = none, name2 = none, transformation_ctx = "", info = "",
stageThreshold = 0, totalThreshold = 0)
```

DynamicFrame의 하나 이상의 필드를 새로운 DynamicFrame으로 분할하고 남겨진 필드를 포함하는 새로운 다른 DynamicFrame을 생성합니다.

- frame - 소스 DynamicFrame으로 새로운 두 개로 스플릿합니다(필수).
- paths - 스플릿할 필드의 전체 경로 목록입니다(필수).
- name1 - 스플릿될 필드를 포함할 DynamicFrame에 지정되는 이름입니다. 이름이 제공되지 않으면 소스 프레임의 이름은 "1"이 붙여져 사용됩니다.
- name2 - 지정된 필드가 스플릿된 후 남겨진 필드를 포함하는 DynamicFrame에 지정되는 이름입니다. 이름이 제공되지 않으면 소스 프레임의 이름은 "2"이 붙여져 사용됩니다.
- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- stageThreshold - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- totalThreshold - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

```
apply(cls, *args, **kwargs)
```

GlueTransform [apply](#)에서 상속됩니다.

```
name(cls)
```

GlueTransform [이름](#)에서 상속됩니다.

describeArgs(cls)

GlueTransform [describeArgs](#)에서 상속됩니다.

describeReturn(cls)

GlueTransform [describeReturn](#)에서 상속됩니다.

describeTransform(cls)

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

SplitRows 클래스

두 개의 DynamicFrames가 포함된 DynamicFrameCollection를 만듭니다. DynamicFrame은 분할 예정으로 지정된 행만 포함하고 다른 하나는 남겨진 모든 행을 포함합니다.

예

DynamicFrame의 행을 분할하려면 [DynamicFrame.split_rows\(\)](#) 메서드를 사용하는 것이 좋습니다. 코드에 대한 예제는 [예제: split_rows를 사용하여 DynamicFrame 행을 분할합니다](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)

- [describe](#)

```
__call__(frame, comparison_dict, name1="frame1", name2="frame2", transformation_ctx = "", info = none, stageThreshold = 0, totalThreshold = 0)
```

새로운 DynamicFrame으로 DynamicFrame의 하나 이상의 열을 스플릿합니다.

- frame - 소스 DynamicFrame으로 새로운 두 개로 스플릿합니다(필수).
- comparison_dict - 열까지 전체 경로의 키와 열 값이 비교된 값의 매핑 비교기를 위한 다른 딕셔너리인 값의 딕셔너리입니다. 예를 들어, {"age": {">": 10, "<": 20}}는 "수명"이 범위를 벗어난 행에서 "수명" 값이 10과 20 사이인 행을 제외하고 스플릿합니다.
- name1 - 스플릿될 행을 포함할 DynamicFrame에 지정되는 이름입니다.
- name2 - 지정된 행이 스플릿된 후 남겨진 DynamicFrame에 지정되는 이름입니다.
- transformation_ctx - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- info - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- stageThreshold - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- totalThreshold - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

```
apply(cls, *args, **kwargs)
```

GlueTransform [apply](#)에서 상속됩니다.

```
name(cls)
```

GlueTransform [이름](#)에서 상속됩니다.

```
describeArgs(cls)
```

GlueTransform [describeArgs](#)에서 상속됩니다.

```
describeReturn(cls)
```

GlueTransform [describeReturn](#)에서 상속됩니다.

```
describeTransform(cls)
```

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

Unbox 클래스

DynamicFrame의 문자열 필드를 개봉합니다(다시 포맷합니다).

예

DynamicFrame에서 필드를 레코드를 개봉하기 위해 [DynamicFrame.unbox\(\)](#) 메소드를 사용하는 것이 좋습니다. 코드에 대한 예제는 [예제: 개봉하기를 사용하여 문자열 필드를 구조에 개봉하기 단원을 참조하세요.](#)

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__(frame, path, format, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0, **options)`

DynamicFrame의 문자열 필드를 개봉합니다.

- `frame` - DynamicFrame에서 필드를 개봉합니다(필수).
- `path` - 개봉할 StringNode로의 전체 경로입니다(필수).
- `format` - 포맷 사양입니다(선택 사항). 여러 포맷을 지원하는 Amazon S3 또는 AWS Glue 연결에 사용됩니다. 지원되는 포맷은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)를 참조하세요.

- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.
- `separator` - 구분자 토큰입니다(선택 사항).
- `escaper` - 이스케이프 토큰입니다(선택 사항).
- `skipFirst` - 첫 번째 데이터 행을 건너뛰어야 한다면 `True`이고 건너뛰지 않아야 한다면 `False`입니다(선택 사항).
- `withSchema` - 개봉할 데이터 스키마를 포함하는 문자열입니다(선택 사항). 이것은 `StructType.json`을 사용하여 생성되어야 합니다.
- `withHeader` - 헤더를 포함해 데이터가 개봉되면 `True`이고 그렇지 않으면 `False`입니다(선택 사항).

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

UnnestFrame 클래스

DynamicFrame을 내보내고 중첩된 객체를 최상위 요소로 평면화하여 배열 객체의 조인 키를 생성합니다.

예

[DynamicFrame.unnest\(\)](#) 메소드를 사용하여 DynamicFrame에서 중첩된 구조를 평면화하는 것이 좋습니다. 코드에 대한 예제는 [예제: 중첩 해제를 사용하여 중첩된 필드를 최상위 필드로 전환합니다](#) 단원을 참조하세요.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__(frame, transformation_ctx = "", info="", stageThreshold=0, totalThreshold=0)`

DynamicFrame을 내보내고 중첩된 객체를 최상위 요소로 평면화하여 배열 객체의 조인 키를 생성합니다.

- `frame` - 중첩하지 않을 DynamicFrame입니다(필수).
- `transformation_ctx` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `info` - 변환에 따른 오류 관련 문자열입니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항). 기본값은 0입니다.

- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항). 기본값은 0입니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

FlagDuplicatesInColumn 수업

FlagDuplicatesInColumn 변환은 각 행에 지정된 값이 있는 새 열을 반환합니다. 이 값은 행의 소스 열에 있는 값이 소스 열의 이전 행에 있는 값과 일치하는지 여부를 나타냅니다. 일치하는 항목이 발견되면 중복 항목으로 플래그가 지정됩니다. 처음 나타나는 항목은 이전 행과 일치하지 않으므로 플래그가 지정되지 않습니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
```

```

from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

datasource1 = spark.read.json("s3://${BUCKET}/json/zips/raw/data")

try:
    df_output = column.FlagDuplicatesInColumn.apply(
        data_frame=datasource1,
        spark_context=sc,
        source_column="city",
        target_column="flag_col",
        true_string="True",
        false_string="False"
    )
except:
    print("Unexpected Error happened ")
    raise

```

출력

FlagDuplicatesInColumn 변환하면 `df_output`에 새 열 `flag_col`이 추가됩니다. DataFrame 이 열에는 해당 행의 'city' 열에 중복된 값이 있는지 여부를 나타내는 문자열 값이 포함됩니다. 행에 중복된 '도시' 값이 있는 경우, 'flag_col'은 'true_string' 값 "True"를 포함합니다. 행에 고유한 '도시' 값이 있는 경우, 'flag_col'에는 'false_string' 값 "False"가 포함됩니다.

결과 `df_output`에는 원본 `datasource1`의 모든 열과 중복된 `city` 값을 나타내는 추가 `flag_col` 열이 포함됩니다 DataFrame . DataFrame

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)

- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 소스_열, 대상_열, True_String=Default_true_String, False_String=Default_False_String)

변환은 행의 소스 열의 값이 소스 열의 이전 행에 있는 값과 일치하는지 여부를 나타내는 각 행에 지정된 값이 있는 새 열을 반환합니다. FlagDuplicatesInColumn 일치하는 항목이 발견되면 중복 항목으로 플래그가 지정됩니다. 처음 나타나는 항목은 이전 행과 일치하지 않으므로 플래그가 지정되지 않습니다.

- `source_column`— 소스 열의 이름.
- `target_column`— 대상 열의 이름.
- `true_string`— 소스 열 값이 해당 열의 이전 값과 중복되는 경우 대상 열에 삽입되는 문자열입니다.
- `false_string`— 소스 열 값이 해당 열의 이전 값과 다른 경우 대상 열에 삽입되는 문자열입니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

FormatPhoneNumber 수업

FormatPhoneNumber 변환은 전화번호 문자열이 형식이 지정된 값으로 변환된 열을 반환합니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        ("408-341-5669",),
        ("4083415669",)
    ],
    ["phone"],
)

try:
    df_output = column_formatting.FormatPhoneNumber.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="phone",
        default_region="US"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

출력

출력은 다음과 같습니다.

...

```
+-----+
| phone|
+-----+
|(408) 341-5669|
|(408) 341-5669|
+-----+
` ``
```

FormatPhoneNumber 변환은 `source_column`을 `"전화"`, `default_region`은 `"US"`로 취합니다.

변환은 초기 형식에 상관없이 두 전화번호의 형식을 미국 표준 형식 `(408) 341-5669`로 성공적으로 지정합니다.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크 컨텍스트, 데이터 프레임, 소스 열, 전화 번호 형식=없음, Default_region=없음, Default_region_column=없음)

변환은 전화번호 문자열이 형식이 지정된 값으로 변환된 열을 반환합니다FormatPhoneNumber.

- `source_column` – 기존 열의 이름입니다.
- `phone_number_format`— 전화번호를 변환할 때 사용할 형식입니다. 형식을 지정하지 않은 경우 기본값은 국제적으로 인정되는 표준 전화번호 형식입니다E.164. 유효한 값은 다음과 같습니다.
 - E164 (E 뒤의 마침표 생략)
- `default_region`— 전화번호 자체에 국가 코드가 없는 경우 전화번호의 지역을 지정하는 2~3개의 대문자로 구성된 유효한 지역 코드입니다. 기껏해야 다음 중 하나를 `defaultRegion` 제공할 `defaultRegionColumn` 수 있습니다.

- `default_region_column`— 고급 데이터 유형의 열 이름 `Country`. 번호 자체에 국가 코드가 없는 경우 지정된 열의 지역 코드를 사용하여 전화 번호의 국가 코드를 결정합니다. 기껏해야 `defaultRegion` 둘 중 하나를 제공할 `defaultRegionColumn` 수 있습니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

FormatCase 수업

FormatCase 변환은 열의 각 문자열을 지정된 케이스 유형으로 변경합니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
```

```

spark = SparkSession(sc)

datasource1 = spark.read.json("s3://${BUCKET}/json/zips/raw/data")

try:
    df_output = data_cleaning.FormatCase.apply(
        data_frame=datasource1,
        spark_context=sc,
        source_column="city",
        case_type="LOWER"
    )
except:
    print("Unexpected Error happened ")
    raise

```

출력

FormatCase 변환은 `case_type="Lower"` 매개변수를 기반으로 `city` 열의 값을 소문자로 변환합니다. 결과 `df_output`에는 원본 DataFrame `datasource1`의 모든 열이 포함되지만 `city` 열 값은 소문자입니다. DataFrame

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 소스_컬럼, 케이스_타입)

변환은 열의 각 문자열을 지정된 케이스 유형으로 변경합니다. FormatCase

- `source_column` – 기존 열의 이름입니다.
- `case_type`— 지원되는 케이스 유형은 CAPITAL,, LOWERUPPER, SENTENCE 입니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

FillWithMode 수업

FillWithMode 변환은 지정한 전화번호 형식에 따라 열의 형식을 지정합니다. 일부 값이 동일한 타이 브레이커 로직을 지정할 수도 있습니다. 예를 들어, 다음 값을 고려해 보십시오. 1 2 2 3 3 4

MINIMUMmodeType은 모드 값으로 2를 FillWithMode 반환하도록 합니다. 모드 타입이 MAXIMUM 인 경우 모드는 3입니다. 의 AVERAGE 경우 모드는 2.5입니다.

예

```
from awsglue.context import *
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)
```

```

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (1055.123, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
)

try:
    df_output = data_quality.FillWithMode.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_1",
        mode_type="MAXIMUM"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise

```

출력

주어진 코드의 출력은 다음과 같습니다.

```

...
+-----+-----+
|source_column_1|source_column_2|
+-----+-----+
| 105.111| 13.12|
| 1055.123| 13.12|
| 1055.123| 13.12|
| 13.12| 13.12|
| 1055.123| 13.12|
+-----+-----+
...

```

'awsglue.data_quality' 모듈에서의 FillWithMode 변환은 'input_df'에 적용됩니다. DataFrame 열의 'null' 값을 해당 열에 있는 null이 아닌 값의 최대값 ('mode_type="Maximum"') 으로 대체합니다.

source_column_1

이 경우 열의 `source_column_1` 최대값은 ``1055.123``입니다. 따라서 출력 ``df_output``에서 의 ``null`` 값은 ``1055.123``으로 대체됩니다. `source_column_1` DataFrame

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 소스_컬럼, 모드_타입)

변환은 열에 있는 문자열의 `FillWithMode` 대/소문자 형식을 지정합니다.

- `source_column` – 기존 열의 이름입니다.
- `mode_type`— 데이터의 동점 값을 해결하는 방법. 이 값은 `MINIMUM`, `NONEAVERAGE`, 또는 중 하나여야 `MAXIMUM` 합니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

describeTransform(cls)

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

FlagDuplicateRows 수업

FlagDuplicateRows 변환은 각 행에 해당 행이 데이터셋의 이전 행과 정확히 일치하는지 여부를 나타내는 지정된 값이 포함된 새 열을 반환합니다. 일치하는 항목이 발견되면 중복으로 플래그가 지정됩니다. 처음 나타나는 항목은 이전 행과 일치하지 않으므로 플래그가 지정되지 않습니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsglue.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (13.12, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
)

try:
    df_output = data_quality.FlagDuplicateRows.apply(
        data_frame=input_df,
        spark_context=sc,
        target_column="flag_row",
    )
```

```

        true_string="True",
        false_string="False",
        target_index=1
    )
except:
    print("Unexpected Error happened ")
    raise

```

출력

열을 기준으로 행의 중복 여부를 flag_row 나타내는 추가 열이 PySpark DataFrame 포함된 a가 출력됩니다. source_column_1 결과 DataFrame `df_output`에는 다음 행이 포함됩니다.

```

...
+-----+-----+-----+
|source_column_1|source_column_2|flag_row|
+-----+-----+-----+
| 105.111| 13.12| False|
| 13.12| 13.12| True|
| null| 13.12| True|
| 13.12| 13.12| True|
| null| 13.12| True|
+-----+-----+-----+
...

```

이 flag_row 열은 행이 중복되었는지 여부를 나타냅니다. `true_string`은 “True”로 설정되고 `false_string`은 “False”로 설정됩니다. `target_index`는 1로 설정되어 있습니다. 즉, 출력의 두 번째 위치 (인덱스 1)에 열이 삽입됩니다. flag_row DataFrame

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)

- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 대상_열, True_String=Default_true_String, False_String=Default_False_String, Target_index=없음)

변환은 각 행에 지정된 값이 있는 새 열을 반환합니다. 이 열은 해당 행이 데이터셋의 이전 행과 정확히 일치하는지 여부를 나타냅니다. FlagDuplicateRows 일치하는 항목이 발견되면 중복으로 플래그가 지정됩니다. 처음 나타나는 항목은 이전 행과 일치하지 않으므로 플래그가 지정되지 않습니다.

- `true_string`— 행이 이전 행과 일치하는 경우 삽입되는 값입니다.
- `false_string`— 행이 고유한 경우 삽입되는 값입니다.
- `target_column`— 데이터세트에 삽입된 새 열의 이름.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

RemoveDuplicates 수업

선택한 RemoveDuplicates 소스 열에 중복된 값이 있는 경우 변환은 전체 행을 삭제합니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsglue.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (105.111, 13.12),
        (13.12, 13.12),
        (None, 13.12),
        (13.12, 13.12),
        (None, 13.12),
    ],
    ["source_column_1", "source_column_2"],
)

try:
    df_output = data_quality.RemoveDuplicates.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_1"
    )
except:
    print("Unexpected Error happened ")
    raise
```

출력

출력은 열을 기준으로 중복 항목이 PySpark DataFrame 제거된 상태로 a가 됩니다. source_column_1 결과 DataFrame `df_output`에는 다음 행이 포함됩니다.

```
...
+-----+-----+
|source_column_1|source_column_2|
```

```
+-----+-----+
| 105.111| 13.12|
| 13.12| 13.12|
| null| 13.12|
+-----+-----+
...
```

참고로, `source_column_1` 값이 `13.12`와 `null`인 행은 열을 기준으로 중복 항목이 제거되었으므로 출력에 한 번만 나타납니다. `DataFrame source_column_1`

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 소스_컬럼)

선택한 소스 열에 중복된 값이 있는 경우 `RemoveDuplicates` 변환은 전체 행을 삭제합니다.

- `source_column` - 기존 열의 이름입니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

describeReturn(cls)

GlueTransform [describeReturn](#)에서 상속됩니다.

describeTransform(cls)

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

MonthName 수업

MonthName 변환은 날짜를 나타내는 문자열에서 월 이름이 포함된 새 열을 만듭니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsglue.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")

input_df = spark.createDataFrame(
    [
        ("20-2018-12",),
        ("2018-20-12",),
        ("20182012",),
        ("12202018",),
        ("20122018",),
        ("20-12-2018",),
        ("12/20/2018",),
        ("02/02/02",),
        ("02 02 2009",),
        ("02/02/2009",),
        ("August/02/2009",),
    ],
```

```

        ("02/june/2009",),
        ("02/2020/june",),
        ("2013-02-21 06:35:45.658505",),
        ("August 02 2009",),
        ("2013/02/21",),
        (None,),
    ],
    ["column_1"],
)

try:
    df_output = datetime_functions.MonthName.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="column_1",
        target_column="target_column"
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise

```

출력

출력은 다음과 같습니다.

```

...
+-----+-----+
| column_1|target_column|
+-----+-----+
|20-2018-12 | December |
|2018-20-12 | null |
| 20182012| null |
| 12202018| null |
| 20122018| null |
|20-12-2018 | December |
|12/20/2018 | December |
| 02/02/02 | February |
|02 02 2009 | February |
|02/02/2009 | February |
|August/02/2009| August |
|02/june/2009| null |
|02/2020/june| null |

```

```
|2013-02-21 06:35:45.658505| February |
|August 02 2009| August |
| 2013/02/21| February |
| null | null |
+-----+-----+
...
```

MonthName변환은 `소스_열_1`으로, `대상_열`은 `대상_열`으로 취합니다. 이 함수는 `"column_1" 열의 날짜/시간 문자열에서 월 이름을 추출하여 "target_column" 열에 배치하려고 시도합니다. 날짜/시간 문자열이 인식할 수 없는 형식이거나 파싱할 수 없는 경우 "target_column" 값은 `null`로 설정됩니다.

변환은 "20-12-2018", "12/20/2018", "02/02/2009", "2013-02-21 06:35:45.658 505", "2009년 8월 2일과 같은 다양한 날짜/시간 형식에서 월 이름을 성공적으로 추출합니다.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 대상_열, 소스_열=없음, 값=없음)

MonthName변환은 날짜를 나타내는 문자열에서 월 이름이 포함된 새 열을 만듭니다.

- `source_column`— 기존 열의 이름입니다.
- `value`— 평가할 문자열입니다.
- `target_column`— 새로 만든 열의 이름.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

IsEven 수업

IsEven변환은 소스 열 또는 값이 짝수인지 여부를 나타내는 부울 값을 새 열에 반환합니다. 소스 열 또는 값이 10진수인 경우 결과는 false입니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsglue.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
```

```

    data_frame=input_df,
    spark_context=sc,
    source_column="source_column",
    target_column="target_column",
    value=None,
    true_string="Even",
    false_string="Not even",
)
df_output.show()
except:
    print("Unexpected Error happened ")
    raise

```

출력

출력은 다음과 같습니다.

```

'''
+-----+-----+
|source_column|target_column|
+-----+-----+
| 5| Not even|
| 0| Even|
| -1| Not even|
| 2| Even|
| null| null|
+-----+-----+
'''

```

IsEven 변환은 `소스_열`을 “소스_열”로, `대상_열`은 “대상_열”로 취합니다. `소스_컬럼`의 값이 짝수인지 아닌지를 확인합니다. 값이 짝수이면 `대상_열` 값을 `true_string` “짝수”로 설정합니다. 값이 홀수이면 `대상_열` 값을 `false_string` “짝수 아님”으로 설정합니다. `소스_컬럼` 값이 `null`인 경우, `타겟_컬럼` 값은 `null`로 설정됩니다.

변환은 짝수 (0과 2) 를 올바르게 식별하고 `target_column` 값을 “짝수”로 설정합니다. 홀수 (5와 -1) 의 경우 `target_column` 값을 “짝수 아님”으로 설정합니다. `소스_컬럼`에 있는 `null` 값의 경우, `타겟_컬럼` 값은 `null`로 설정됩니다.

메서드

- [__call__](#)

- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 대상_열, 소스_컬럼=없음, 트루_문자열=기본값_트루_문자열, 거짓_문자열=기본값_거짓_문자열, 값=없음)

IsEven변환은 소스 열 또는 값이 짝수인지 여부를 나타내는 부울 값을 새 열에 반환합니다. 소스 열 또는 값이 10진수인 경우 결과는 false입니다.

- `source_column` - 기존 열의 이름입니다.
- `target_column` - 생성할 새 열의 이름.
- `true_string` - 값이 짝수인지 여부를 나타내는 문자열.
- `false_string` - 값이 짝수가 아닌지 여부를 나타내는 문자열입니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

CryptographicHash 수업

CryptographicHash변환은 열의 해시 값에 알고리즘을 적용합니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsglue.transforms import *

secret = "${SECRET}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
        (7, "1234560006"),
        (8, "1234560007"),
        (9, "1234560008"),
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
    df_output = pii.CryptographicHash.apply(
        data_frame=input_df,
        spark_context=sc,
        source_columns=["id", "phone"],
        secret_id=secret,
```

```

        algorithm="HMAC_SHA256",
        output_format="BASE64",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise

```

출력

출력은 다음과 같습니다.

```

...
+---+-----+-----+-----+
| id| phone | id_hashed | phone_hashed |
+---+-----+-----+-----+
| 1| 1234560000 | QUI1zXTJiXmfIb... | juDBAmiRnn03g... |
| 2| 1234560001 | ZAUWiZ3dVTzCo... | vC81gUqBVDMNQ... |
| 3| 1234560002 | ZP4VvZWkqYifu... | K13QAkgsWYpzB... |
| 4| 1234560003 | 3u8v03wQ8EQfj... | CPBzK1P8PZZkV... |
| 5| 1234560004 | eWkQJk4zA0Izx... | aLf7+mHcXqbLs... |
| 6| 1234560005 | xtI9fZCJZCvsa... | dy2DFgdYWmr0p... |
| 7| 1234560006 | iW9hew7jnHu0f... | wwfgMCOEv6o0v... |
| 8| 1234560007 | H9V1pqvgkFhfS... | g9WKhagIXy9ht... |
| 9| 1234560008 | xDhEuHaxAUbU5... | b3uQLKPY+Q5vU... |
| 10| 1234560009 | GRN6nFXkxk349... | VJdsKt8VbxBbt... |
+---+-----+-----+-----+
...

```

변환은 지정된 알고리즘과 비밀 키를 사용하여 `id` 및 `phone` 열에 있는 값의 암호화 해시를 계산하고 해시를 Base64 형식으로 인코딩합니다. 결과 `df_output`에는 원래 `input_df`의 모든 열과 계산된 해시가 DataFrame 포함된 추가 `id_hashed` 및 `phone_hashed` 열이 포함됩니다. DataFrame

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)

- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 소스_컬럼, 비밀_id, 알고리즘=없음, 비밀_버전=없음, Create_Secret_IF_Missing=false, 출력_형식=없음, entity_type_filter=없음)

변환은 열의 해시 값에 알고리즘을 적용합니다. CryptographicHash

- `source_columns`— 기존 열로 구성된 배열.
- `secret_id`— 시크릿 매니저 시크릿 키의 ARN 해시 기반 메시지 인증 코드 (HMAC) 접두사 알고리즘에서 소스 열을 해시하는 데 사용되는 키입니다.
- `secret_version`— 선택 사항입니다. 기본값은 최신 시크릿 버전입니다.
- `entity_type_filter`— 엔티티 유형의 선택적 배열. 감지된 PII만 자유 텍스트 열에서 암호화하는 데 사용할 수 있습니다.
- `create_secret_if_missing`— 선택적 부울. true인 경우 호출자를 대신하여 시크릿 생성을 시도합니다.
- `algorithm`— 데이터를 해시하는 데 사용되는 알고리즘. 유효한 열거형 값: MD5, SHA1, SHA256, SHA512, HMAC_MD5, HMAC_SHA1, HMAC_SHA256, HMAC_SHA512

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

클래스 복호화

Decrypt변환은 Glue AWS 내부의 암호를 해독합니다. 암호화 SDK를 사용하여 AWS Glue 외부에서 데이터를 해독할 수도 있습니다. AWS 제공된 KMS 키 ARN이 열을 암호화하는 데 사용된 것과 일치하지 않으면 암호 해독 작업이 실패합니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

kms = "${KMS}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
        (7, "1234560006"),
        (8, "1234560007"),
        (9, "1234560008"),
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
    df_encrypt = pii.Encrypt.apply(
        data_frame=input_df,
        spark_context=sc,
```

```

        source_columns=["phone"],
        kms_key_arn=kms
    )
    df_decrypt = pii.Decrypt.apply(
        data_frame=df_encrypt,
        spark_context=sc,
        source_columns=["phone"],
        kms_key_arn=kms
    )
    df_decrypt.show()
except:
    print("Unexpected Error happened ")
    raise

```

출력

출력은 원래 `id` 열과 해독된 `phone` 열이 PySpark DataFrame 포함된 a가 됩니다.

```

...
+---+-----+
| id| phone|
+---+-----+
| 1| 1234560000|
| 2| 1234560001|
| 3| 1234560002|
| 4| 1234560003|
| 5| 1234560004|
| 6| 1234560005|
| 7| 1234560006|
| 8| 1234560007|
| 9| 1234560008|
| 10| 1234560009|
+---+-----+
...

```

Encrypt 변환은 `source_columns`를 `[\"전화\"]`로, `kms_key_arn`은 `\${KMS}` 환경 변수의 값으로 취합니다. 변환은 지정된 KMS 키를 사용하여 '전화' 열의 값을 암호화합니다. 그런 다음 암호화된 DataFrame `df_encrypt`가 `awsglue.pii` 모듈에서 변환으로 전달됩니다. Decrypt '소스_컬럼'은 `[\"전화\"]`로, `kms_key_arn`은 `\${KMS}` 환경 변수의 값으로 취합니다. 변환은 동일한 KMS 키를 사용하여 'phone' 열의 암호화된 값을 해독합니다. 결과 DataFrame `df_decrypt`에는 원래 `id` 열과 해독된 `전화` 열이 포함됩니다.

메서드

- [__call__](#)
- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 소스_컬럼, kms_key_arn)

Decrypt 변환은 Glue AWS 내부의 암호를 해독합니다. 암호화 SDK를 사용하여 AWS Glue 외부에서 데이터를 해독할 수도 있습니다. AWS 제공된 KMS 키 ARN이 열을 암호화하는 데 사용된 것과 일치하지 않으면 암호 해독 작업이 실패합니다.

- `source_columns`— 기존 열의 배열.
- `kms_key_arn`— 소스 열을 해독하는 데 사용할 AWS 키 관리 서비스 키의 키 ARN입니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

`describe(cls)`

GlueTransform [describe](#)에서 상속됩니다.

암호화 클래스

Encrypt변환은 키 관리 서비스 키를 사용하여 소스 열을 암호화합니다. AWS Encrypt변환은 셀당 최대 128MiB까지 암호화할 수 있습니다. 복호화 시 형식 보존을 시도합니다. 데이터 유형을 보존하려면 데이터 유형 메타데이터를 1KB 미만으로 직렬화해야 합니다. 그렇지 않으면 `preserve_data_type` 파라미터를 `false`로 설정해야 합니다. 데이터 유형 메타데이터는 암호화 컨텍스트에서 일반 텍스트로 저장됩니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

kms = "${KMS}"
sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (1, "1234560000"),
        (2, "1234560001"),
        (3, "1234560002"),
        (4, "1234560003"),
        (5, "1234560004"),
        (6, "1234560005"),
        (7, "1234560006"),
        (8, "1234560007"),
        (9, "1234560008"),
        (10, "1234560009"),
    ],
    ["id", "phone"],
)

try:
```

```

df_encrypt = pii.Encrypt.apply(
    data_frame=input_df,
    spark_context=sc,
    source_columns=["phone"],
    kms_key_arn=kms
)
except:
    print("Unexpected Error happened ")
    raise

```

출력

출력은 원래 `id` 열과 `phone` 열의 암호화된 값을 포함하는 추가 열이 포함된 PySpark DataFrame a가 됩니다.

```

...
+---+-----+-----+
| id| phone | phone_encrypted |
+---+-----+-----+
| 1| 1234560000| EncryptedData1234...abc |
| 2| 1234560001| EncryptedData5678...def |
| 3| 1234560002| EncryptedData9012...ghi |
| 4| 1234560003| EncryptedData3456...jkl |
| 5| 1234560004| EncryptedData7890...mno |
| 6| 1234560005| EncryptedData1234...pqr |
| 7| 1234560006| EncryptedData5678...stu |
| 8| 1234560007| EncryptedData9012...vwx |
| 9| 1234560008| EncryptedData3456...yz0 |
| 10| 1234560009| EncryptedData7890...123 |
+---+-----+-----+
...

```

Encrypt 변환은 `source_columns`를 `[\"전화\"]`로 취하고 `kms_key_arn`은 `\${KMS}` 환경 변수의 값으로 취합니다. 변환은 지정된 KMS 키를 사용하여 '전화' 열의 값을 암호화합니다. 결과 `df_encrypt`에는 원래 `id` 열, 원래 '전화' 열, `phone` 열의 암호화된 값을 DataFrame 포함하는 `phone_encrypted`라는 추가 열이 포함됩니다.

메서드

- [__call__](#)
- [apply](#)

- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 소스_열, kms_key_arn, Entity_Type_Filter=없음, Preserve_Data_Type=없음)

Encrypt변환은 키 관리 서비스 키를 사용하여 소스 열을 AWS 암호화합니다.

- `source_columns`— 기존 열의 배열.
- `kms_key_arn`— 소스 열을 암호화하는 데 사용할 AWS 키 관리 서비스 키의 키 ARN입니다.
- `entity_type_filter`— 엔티티 유형의 선택적 배열. 감지된 PII만 자유 텍스트 열에서 암호화하는 데 사용할 수 있습니다.
- `preserve_data_type`— 선택적 부울. 기본값은 true입니다. false인 경우 데이터 유형이 저장되지 않습니다.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

describeErrors(cls)

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

IntToIp 수업

IntToIp 변환은 소스 열 또는 기타 값의 정수 값을 대상 열의 해당 IPv4 값으로 변환하고 결과를 새 열에 반환합니다.

예

```
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from awsgluedi.transforms import *

sc = SparkContext()
spark = SparkSession(sc)

input_df = spark.createDataFrame(
    [
        (3221225473,),
        (0,),
        (1,),
        (100,),
        (168430090,),
        (4294967295,),
        (4294967294,),
        (4294967296,),
        (-1,),
        (None,)
    ],
    ["source_column_int"],
)

try:
    df_output = web_functions.IntToIp.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column_int",
        target_column="target_column",
```

```

        value=None
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise

```

출력

출력은 다음과 같습니다.

```

...
+-----+-----+
|source_column_int|target_column|
+-----+-----+
| 3221225473| 192.0.0.1 |
| 0| 0.0.0.0 |
| 1| 0.0.0.1 |
| 100| 0.0.0.100|
| 168430090 | 10.0.0.10 |
| 4294967295| 255.255.255.255|
| 4294967294| 255.255.255.254|
| 4294967296| null |
| -1| null |
| null| null |
+-----+-----+
...

```

IntToIp.apply변환은 `소스_컬럼`을 `소스_컬럼_int`로, `타겟_컬럼`을 `타겟_컬럼`으로 취하고, `source_column_int` 열의 정수 값을 해당하는 IPv4 주소 표현으로 변환하고 결과를 `타겟_컬럼` 열에 저장합니다.

IPv4 주소 범위 (0~4294967295) 내에 있는 유효한 정수 값의 경우 변환 시 해당 값이 해당 IPv4 주소 표현 (예: 192.0.0.1, 0.0.0.0, 10.0.0.10, 255.255.255.255) 으로 성공적으로 변환됩니다.

유효 범위를 벗어난 정수 값 (예: 4294967296, -1) 의 경우, `타겟_컬럼` 값은 `null`로 설정됩니다. `소스_컬럼_int` 열에 있는 `null` 값의 경우 `타겟_컬럼` 값도 `null`로 설정됩니다.

메서드

- [__call__](#)

- [apply](#)
- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 대상_열, 소스_열=없음, 값=없음)

IntToIp변환은 소스 열 또는 기타 값의 정수 값을 대상 열의 해당 IPv4 값으로 변환하고 결과를 새 열에 반환합니다.

- `sourceColumn`— 기존 열의 이름입니다.
- `value`— 평가할 문자열.
- `targetColumn`— 생성할 새 열의 이름.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

IpToInt 수업

IpToInt 변환은 소스 열의 IPv4 (인터넷 프로토콜 버전 4) 값이나 기타 값을 대상 열의 해당 정수 값으로 변환하고 결과를 새 열에 반환합니다.

예

AWS Glue 4.0 이상의 경우 다음과 같이 작업 인수를 생성하거나 업데이트하십시오. key: --enable-glue-di-transforms, value: true

```
from pyspark.context import SparkContext
from awsgluedi.transforms import *

sc = SparkContext()

input_df = spark.createDataFrame(
    [
        ("192.0.0.1",),
        ("10.10.10.10",),
        ("1.2.3.4",),
        ("1.2.3.6",),
        ("http://12.13.14.15",),
        ("https://16.17.18.19",),
        ("1.2.3.4",),
        (None,),
        ("abc",),
        ("abc.abc.abc.abc",),
        ("321.123.123.123",),
        ("244.4.4.4",),
        ("255.255.255.255",),
    ],
    ["source_column_ip"],
)

df_output = web_functions.IpToInt.apply(
    data_frame=input_df,
    spark_context=sc,
    source_column="source_column_ip",
    target_column="target_column",
    value=None
```

```
)
df_output.show()
```

출력

출력은 다음과 같습니다.

```
...
+-----+-----+
|source_column_ip| target_column|
+-----+-----+
| 192.0.0.1| 3221225473|
| 10.10.10.10| 168427722|
| 1.2.3.4| 16909060|
| 1.2.3.6| 16909062|
|http://12.13.14.15| null|
|https://16.17.18.19| null|
| 1.2.3.4| 16909060|
| null| null|
| abc| null|
|abc.abc.abc.abc| null|
| 321.123.123.123| null|
| 244.4.4.4| 4102444804|
| 255.255.255.255| 4294967295|
+-----+-----+
...
```

IpToInt변환은 `소스_컬럼`을 `소스_컬럼_ip`로 취하고, `타겟_컬럼`은 `타겟_컬럼`으로 취하고, `소스_컬럼_ip` 열의 유효한 IPv4 주소 문자열을 해당하는 32비트 정수 표현으로 변환하고 결과를 `타겟_컬럼` 열에 저장합니다.

유효한 IPv4 주소 문자열 (예: "192.0.0.1", "10.10.10", "1.2.3.4") 의 경우 변환은 해당 문자열을 정수 표현 (예: 3221225473, 168427722, 16909060) 으로 성공적으로 변환합니다. 유효한 IPv4 주소가 아닌 문자열 (예: URL, "abc"와 같은 비 IP 문자열, "abc.abc.abc.abc"와 같은 잘못된 IP 형식) 의 경우, `대상_열` 값은 `null`로 설정됩니다. `소스_컬럼_ip` 열에 있는 `null` 값의 경우, `타겟_컬럼` 값도 `null`로 설정됩니다.

메서드

- [_call](#)
- [apply](#)

- [이름](#)
- [describeArgs](#)
- [describeReturn](#)
- [describeTransform](#)
- [describeErrors](#)
- [describe](#)

`__call__` (스파크_컨텍스트, 데이터_프레임, 대상_열, 소스_컬럼=없음, 값=없음)

IpToInt 변환은 소스 열의 IPv4 (인터넷 프로토콜 버전 4) 값이나 기타 값을 대상 열의 해당 정수 값으로 변환하고 결과를 새 열에 반환합니다.

- `sourceColumn` – 기존 열의 이름입니다.
- `value`— 평가할 문자열입니다.
- `targetColumn`— 생성할 새 열의 이름.

`apply(cls, *args, **kwargs)`

GlueTransform [apply](#)에서 상속됩니다.

`name(cls)`

GlueTransform [이름](#)에서 상속됩니다.

`describeArgs(cls)`

GlueTransform [describeArgs](#)에서 상속됩니다.

`describeReturn(cls)`

GlueTransform [describeReturn](#)에서 상속됩니다.

`describeTransform(cls)`

GlueTransform [describeTransform](#)에서 상속됩니다.

`describeErrors(cls)`

GlueTransform [describeErrors](#)에서 상속됩니다.

describe(cls)

GlueTransform [describe](#)에서 상속됩니다.

데이터 통합 변환

AWS Glue 4.0 이상의 경우 를 사용하여 작업 인수를 생성하거나 업데이트하십시오key: `--enable-glue-di-transforms`, value: `true`.

예제 작업 스크립트:

```
from pyspark.context import SparkContext

from awsgluedi.transforms import *
sc = SparkContext()

input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column",
        target_column="target_column",
        value=None,
        true_string="Even",
        false_string="Not even",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

노트북을 사용한 예제 세션

```
%idle_timeout 2880
%glue_version 4.0
%worker_type G.1X
%number_of_workers 5
```



```
%region eu-west-1
```

```
%%configure
{
  "--enable-glue-di-transforms": "true"
}
```

```
from pyspark.context import SparkContext
from awsgluedi.transforms import *

sc = SparkContext()

input_df = spark.createDataFrame(
    [(5,), (0,), (-1,), (2,), (None,)],
    ["source_column"],
)

try:
    df_output = math_functions.IsEven.apply(
        data_frame=input_df,
        spark_context=sc,
        source_column="source_column",
        target_column="target_column",
        value=None,
        true_string="Even",
        false_string="Not even",
    )
    df_output.show()
except:
    print("Unexpected Error happened ")
    raise
```

다음을 사용하는 예제: 세션 AWS CLI

```
aws glue create-session --default-arguments "--enable-glue-di-transforms=true"
```

DI 변환:

- [FlagDuplicatesInColumn 수업](#)
- [FormatPhoneNumber 수업](#)
- [FormatCase 수업](#)

- [FillWithMode](#) 수업
- [FlagDuplicateRows](#) 수업
- [RemoveDuplicates](#) 수업
- [MonthName](#) 수업
- [IsEven](#) 수업
- [CryptographicHash](#) 수업
- [클래스 복호화](#)
- [암호화 클래스](#)
- [IntToIp](#) 수업
- [IpToInt](#) 수업

Maven: Spark 애플리케이션과 플러그인 번들

Spark 애플리케이션을 로컬에서 개발하는 동안 Maven에 플러그인 종속성을 추가하여 Spark 애플리케이션 및 Spark 배포판 (버전 3.3) 과 함께 변환 종속성을 번들로 제공할 수 있습니다. pom.xml

```
<repositories>
  ...
  <repository>
    <id>aws-glue-etl-artifacts</id>
    <url>https://aws-glue-etl-artifacts.s3.amazonaws.com/release/ </url>
  </repository>
</repositories>
...
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>AWSGlueTransforms</artifactId>
  <version>4.0.0</version>
</dependency>
```

다음과 같이 AWS Glue Maven 아티팩트에서 바이너리를 직접 다운로드하여 Spark 애플리케이션에 포함시킬 수도 있습니다.

```
#!/bin/bash
sudo wget -v https://aws-glue-etl-artifacts.s3.amazonaws.com/release/com.amazonaws/AWSGlueTransforms/4.0.0/AWSGlueTransforms-4.0.0.jar -P /usr/lib/spark/jars/
```

Scala의 AWS Glue ETL 스크립트 프로그래밍

AWS Glue용 Scala 코드 예제와 유틸리티는 GitHub 웹 사이트의 [AWS Glue 샘플 리포지토리](#)에서 찾을 수 있습니다.

AWS Glue는 추출, 변환, 로드 작업 스크립트의 PySpark Scala의 확장 언어를 지원합니다. 다음 섹션은 AWS GlueScala 라이브러리와 ETL 스크립트의 AWS Glue API를 사용하고 라이브러리를 참조 문서를 제공하는 방법에 대해 설명합니다.

목차

- [Scala 사용하여 AWS Glue ETL 스크립트 프로그래밍](#)
 - [Scala ETL 프로그램을 개발 엔드포인트의 Jupyter Notebook에서 테스트](#)
 - [Scala REPL에서 Scala ETL 프로그램 테스트](#)
- [Scala 스크립트 예 - 스트리밍 ETL](#)
- [AWS Glue Scala 라이브러리의 API](#)
 - [com.amazonaws.services.glue](#)
 - [com.amazonaws.services.glue.ml](#)
 - [com.amazonaws.services.glue.dq](#)
 - [com.amazonaws.services.glue.types](#)
 - [com.amazonaws.services.glue.util](#)
- [AWS Glue Scala ChoiceOption API](#)
 - [ChoiceOption 특성](#)
 - [ChoiceOption 객체](#)
 - [정의 적용](#)
 - [케이스 클래스 ChoiceOptionWithResolver](#)
 - [케이스 클래스 MatchCatalogSchemaChoiceOption](#)
- [추상 DataSink 클래스](#)
 - [writeDynamicFrame 정의](#)
 - [pyWriteDynamicFrame 정의](#)
 - [Def writeDataFrame](#)
 - [Def pyWriteDataFrame](#)
 - [setCatalogInfo 정의](#)
 - [supportsFormat 정의](#)

- [setFormat 정의](#)
- [withFormat 정의](#)
- [setAccumulableSize 정의](#)
- [getOutputErrorRecordsAccumulable 정의](#)
- [errorsAsDynamicFrame 정의](#)
- [DataSink 객체](#)
 - [recordMetrics 정의](#)
- [AWS Glue Scala DataSource 특성](#)
- [AWS Glue Scala DynamicFrame API](#)
 - [AWS Glue Scala DynamicFrame 클래스](#)
 - [val errorsCount](#)
 - [def applyMapping](#)
 - [def assertErrorThreshold](#)
 - [def count](#)
 - [def dropField](#)
 - [def dropFields](#)
 - [def dropNulls](#)
 - [def errorsAsDynamicFrame](#)
 - [def filter](#)
 - [def getName](#)
 - [def getNumPartitions](#)
 - [def getSchemalfComputed](#)
 - [def isSchemaComputed](#)
 - [def javaToPython](#)
 - [def join](#)
 - [def map](#)
 - [def mergeDynamicFrames](#)
 - [def printSchema](#)
 - [def recomputeSchema](#)
 - [def relationalize](#)

- [def renameField](#)
- [def repartition](#)
- [def resolveChoice](#)
- [def schema](#)
- [def selectField](#)
- [def selectFields](#)
- [def show](#)
- [Def simplifyDDBJson](#)
- [def spigot](#)
- [def splitFields](#)
- [def splitRows](#)
- [def stageErrorsCount](#)
- [def toDF](#)
- [def unbox](#)
- [def unnest](#)
- [def unnestDDBJson](#)
- [def withFrameSchema](#)
- [def withName](#)
- [def withTransformationContext](#)
- [DynamicFrame 객체](#)
 - [def apply](#)
 - [def emptyDynamicFrame](#)
 - [def fromPythonRDD](#)
 - [def ignoreErrors](#)
 - [def inlineErrors](#)
 - [def newFrameWithErrors](#)
- [AWS Glue Scala DynamicRecord 클래스](#)
 - [def addField](#)
 - [def dropField](#)
 - [def setError](#)

- [def isError](#)
- [def getError](#)
- [def clearError](#)
- [def write](#)
- [def readFields](#)
- [def clone](#)
- [def schema](#)
- [def getRoot](#)
- [def toJson](#)
- [def getFieldNode](#)
- [def getField](#)
- [def hashCode](#)
- [def equals](#)
- [DynamicRecord 객체](#)
 - [정의 적용](#)
- [RecordTraverser 특성](#)
- [AWS Glue스칼라 API GlueContext](#)
 - [데프 컬럼 addIngestionTime](#)
 - [데프 createDataFrame FromOptions](#)
 - [forEachBatch](#)
 - [def getCatalogSink](#)
 - [데프 getCatalogSource](#)
 - [def getJDBCSink](#)
 - [def getSink](#)
 - [getSinkWithdef 포맷](#)
 - [def getSource](#)
 - [def 포맷 getSourceWith](#)
 - [def getSparkSession](#)
 - [def startTransaction](#)
 - [def commitTransaction](#)

- [def cancelTransaction](#)
- [def this](#)
- [def this](#)
- [def this](#)
- [MappingSpec](#)
 - [MappingSpec 케이스 클래스](#)
 - [MappingSpec 객체](#)
 - [Val orderingByTarget](#)
 - [정의 적용](#)
 - [정의 적용](#)
 - [정의 적용](#)
- [AWS Glue Scala ResolveSpec API](#)
 - [ResolveSpec 객체](#)
 - [def](#)
 - [def](#)
 - [ResolveSpec 케이스 클래스](#)
 - [ResolveSpec def 메서드](#)
- [AWS Glue Scala ArrayNode API](#)
 - [ArrayNode 케이스 클래스](#)
 - [ArrayNode 정의 메서드](#)
- [AWS Glue Scala BinaryNode API](#)
 - [BinaryNode 케이스 클래스](#)
 - [BinaryNode val 필드](#)
 - [BinaryNode 정의 메서드](#)
- [AWS Glue Scala BooleanNode API](#)
 - [BooleanNode 케이스 클래스](#)
 - [BooleanNode val 필드](#)
 - [BooleanNode 정의 메서드](#)
- [AWS Glue Scala ByteNode API](#)
 - [ByteNode 케이스 클래스](#)

- [ByteNode val 필드](#)
- [ByteNode 정의 메서드](#)
- [AWS Glue Scala DateNode API](#)
 - [DateNode 케이스 클래스](#)
 - [DateNode val 필드](#)
 - [ByteNode def 메서드](#)
- [AWS Glue Scala DecimalNode API](#)
 - [DecimalNode 케이스 클래스](#)
 - [DecimalNode val 필드](#)
 - [DecimalNode def 메서드](#)
- [AWS Glue Scala DoubleNode API](#)
 - [DoubleNode 케이스 클래스](#)
 - [DoubleNode val 필드](#)
 - [DoubleNode def 메서드](#)
- [AWS Glue Scala DynamicNode API](#)
 - [DynamicNode 클래스](#)
 - [DynamicNode def 메서드](#)
 - [DynamicNode 객체](#)
 - [DynamicNode def 메서드](#)
- [EvaluateDataQuality 클래스](#)
 - [def apply](#)
 - [예](#)
- [AWS Glue Scala FloatNode API](#)
 - [FloatNode 케이스 클래스](#)
 - [FloatNode val 필드](#)
 - [FloatNode def 메서드](#)
- [FillMissingValues 클래스](#)
 - [def apply](#)
- [FindMatches 클래스](#)
 - [def apply](#)

- [FindIncrementalMatches 클래스](#)
 - [def apply](#)
- [AWS Glue Scala IntegerNode API](#)
 - [IntegerNode 케이스 클래스](#)
 - [IntegerNode val 필드](#)
 - [IntegerNode def 메서드](#)
- [AWS Glue Scala LongNode API](#)
 - [LongNode 케이스 클래스](#)
 - [LongNode val 필드](#)
 - [LongNode def 메서드](#)
- [AWS Glue Scala MapLikeNode API](#)
 - [MapLikeNode 클래스](#)
 - [MapLikeNode def 메서드](#)
- [AWS Glue Scala MapNode API](#)
 - [MapNode 케이스 클래스](#)
 - [MapNode def 메서드](#)
- [AWS Glue Scala NullNode API](#)
 - [NullNode 클래스](#)
 - [NullNode 케이스 객체](#)
- [AWS Glue Scala ObjectNode API](#)
 - [ObjectNode 객체](#)
 - [ObjectNode def 메서드](#)
 - [ObjectNode 케이스 클래스](#)
 - [ObjectNode def 메서드](#)
- [AWS Glue Scala ScalarNode API](#)
 - [ScalarNode 클래스](#)
 - [ScalarNode def 메서드](#)
 - [ScalarNode 객체](#)
 - [ScalarNode def 메서드](#)
- [AWS Glue Scala ShortNode API](#)

- [ShortNode 케이스 클래스](#)
 - [ShortNode val 필드](#)
 - [ShortNode def 메서드](#)
- [AWS Glue Scala StringNode API](#)
 - [StringNode 케이스 클래스](#)
 - [StringNode val 필드](#)
 - [StringNode def 메서드](#)
- [AWS Glue Scala TimestampNode API](#)
 - [TimestampNode 케이스 클래스](#)
 - [TimestampNode val 필드](#)
 - [TimestampNode def 메서드](#)
- [AWS Glue Scala GlueArgParser API](#)
 - [GlueArgParser 객체](#)
 - [GlueArgParser def 메서드](#)
- [AWS Glue Scala 작업 API](#)
 - [작업 객체](#)
 - [작업 def 메서드](#)

Scala 사용하여 AWS Glue ETL 스크립트 프로그래밍

AWS Glue 콘솔을 사용하여 Scala ETL(extract, transform, load: 추출, 변환, 로드) 프로그램을 자동으로 생성하고, 필요에 따라 수정한 후 작업에 할당할 수 있습니다. 또는 처음부터 자체 프로그램을 작성할 수 있습니다. 자세한 내용은 [에서 Spark 작업에 대한 작업 속성 구성 AWS Glue](#)를 참조하십시오. 그런 다음 AWS Glue는 관련 작업을 실행하기 전에 Scala 프로그램을 서버에 포함합니다.

프로그램이 오류없이 포함되고 예상대로 실행되는지 확인합니다. 이런 과정은 읽기, 평가, 출력 루프(REPL) 혹은 Jupyter Notebook의 개발 엔드포인트에서 프로그램을 로드하고 작업에서 실행하기 전에 프로그램을 테스트할 때 중요합니다. 서버에서 포함하는 과정이 발생하기 때문에 여기에서 진행되는 어떠한 문제도 눈으로 볼 수 없습니다.

Scala ETL 프로그램을 개발 엔드포인트의 Jupyter Notebook에서 테스트

AWS Glue 개발 엔드포인트에서 Scala 프로그램을 테스트하려면 [개발 엔드포인트 추가](#)에 나오는 설명대로 개발 엔드포인트를 설정합니다.

그런 다음, 사용자의 시스템에서 로컬로 실행되거나 Amazon EC2 노트북 서버에서 원격으로 실행되는 Jupyter Notebook에 연결합니다. [자습서: JupyterLab의 Jupyter Notebook](#)의 지시에 따라 Jupyter Notebook의 로컬 버전을 설치합니다.

노트북의 Scala 코드를 실행하는 것과 PySpark 코드를 실행하는 것의 단 한 가지 차이는 다음을 통해 Notebook의 각 단락을 시작해야 합니다.

```
%spark
```

이런 과정은 노트북 서버가 Spark 인터프리터의 PySpark에 맞게 초기화되는 것을 방지합니다.

Scala REPL에서 Scala ETL 프로그램 테스트

AWS Glue Scala REPL을 사용하여 개발 엔드포인트의 Scala 프로그램을 테스트할 수 있습니다. [자습서: SageMaker 노트북 사용](#) 지침에 따라 SSH-to-REPL 명령 끝부분을 제외하고 `-t gluepyspark`를 `-t glue-spark-shell`로 바꿉니다. 그러면 AWS Glue Scala REPL이 호출됩니다.

완료되면 REPL을 닫아서 `sys.exit`을 입력합니다.

Scala 스크립트 예 - 스트리밍 ETL

Example

다음 예제 스크립트는 Amazon Kinesis Data Streams에 연결하고 Data Catalog의 스키마를 사용하여 데이터 스트림을 구문 분석하고, Amazon S3의 정적 데이터 집합에 스트림을 조인하고, 조인된 결과를 parquet 포맷의 Amazon S3에 출력합니다.

```
// This script connects to an Amazon Kinesis stream, uses a schema from the data
// catalog to parse the stream,
// joins the stream to a static dataset on Amazon S3, and outputs the joined results to
// Amazon S3 in parquet format.
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import java.util.Calendar
import org.apache.spark.SparkContext
import org.apache.spark.sql.Dataset
import org.apache.spark.sql.Row
import org.apache.spark.sql.SaveMode
import org.apache.spark.sql.Session
import org.apache.spark.sql.functions.from_json
import org.apache.spark.sql.streaming.Trigger
import scala.collection.JavaConverters._
```

```

object streamJoiner {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val sparkSession: SparkSession = glueContext.getSparkSession
    import sparkSession.implicits._
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val staticData = sparkSession.read          // read() returns type DataFrameReader
      .format("csv")
      .option("header", "true")
      .load("s3://awsexamplebucket-streaming-demo2/inputs/productsStatic.csv") //
load() returns a DataFrame

    val datasource0 = sparkSession.readStream // readstream() returns type
DataStreamReader
      .format("kinesis")
      .option("streamName", "stream-join-demo")
      .option("endpointUrl", "https://kinesis.us-east-1.amazonaws.com")
      .option("startingPosition", "TRIM_HORIZON")
      .load // load() returns a DataFrame

    val selectfields1 = datasource0.select(from_json($"data".cast("string"),
glueContext.getCatalogSchemaAsSparkSchema("stream-demos", "stream-join-demo2")) as
"data").select("data.*")

    val datasink2 = selectfields1.writeStream.foreachBatch { (dataFrame: Dataset[Row],
batchId: Long) => { //foreachBatch() returns type DataStreamWriter
      val joined = dataFrame.join(staticData, "product_id")
      val year: Int = Calendar.getInstance().get(Calendar.YEAR)
      val month :Int = Calendar.getInstance().get(Calendar.MONTH) + 1
      val day: Int = Calendar.getInstance().get(Calendar.DATE)
      val hour: Int = Calendar.getInstance().get(Calendar.HOUR_OF_DAY)

      if (dataFrame.count() > 0) {
        joined.write // joined.write returns type
DataFrameWriter
          .mode(SaveMode.Append)
          .format("parquet")
          .option("quote", " ")

```

```

        .save("s3://awsexamplebucket-streaming-demo2/output/" + "/year=" +
"%04d".format(year) + "/month=" + "%02d".format(month) + "/day=" + "%02d".format(day)
+ "/hour=" + "%02d".format(hour) + "/")
    }
}
} // end foreachBatch()
    .trigger(Trigger.ProcessingTime("100 seconds"))
    .option("checkpointLocation", "s3://awsexamplebucket-streaming-demo2/
checkpoint/")
    .start().awaitTermination() // start() returns type StreamingQuery
    Job.commit()
}
}

```

AWS Glue Scala 라이브러리의 API

AWS Glue는 추출, 변환 및 로드(ETL) 작업 스크립트의 PySpark Scala의 확장 언어를 지원합니다. 다음 단원에서는 AWS Glue Scala 라이브러리의 API를 설명합니다.

`com.amazonaws.services.glue`

AWS Glue Scala 라이브러리의 `com.amazonaws.services.glue` 패키지에는 다음 API가 포함되어 있습니다.

- [ChoiceOption](#)
- [DataSink](#)
- [DataSource 특성](#)
- [DynamicFrame](#)
- [DynamicRecord](#)
- [GlueContext](#)
- [MappingSpec](#)
- [ResolveSpec](#)

`com.amazonaws.services.glue.ml`

AWS Glue Scala 라이브러리의 `com.amazonaws.services.glue.ml` 패키지에는 다음 API가 포함되어 있습니다.

- [FillMissingValues](#)

- [FindIncrementalMatches](#)
- [FindMatches](#)

com.amazonaws.services.glue.dq

AWS Glue Scala 라이브러리의 com.amazonaws.services.glue.dq 패키지에는 다음 API가 포함되어 있습니다.

- [EvaluateDataQuality](#)

com.amazonaws.services.glue.types

AWS Glue Scala 라이브러리의 com.amazonaws.services.glue.types 패키지에는 다음 API가 포함되어 있습니다.

- [ArrayNode](#)
- [BinaryNode](#)
- [BooleanNode](#)
- [ByteNode](#)
- [DateNode](#)
- [DecimalNode](#)
- [DoubleNode](#)
- [DynamicNode](#)
- [FloatNode](#)
- [IntegerNode](#)
- [LongNode](#)
- [MapLikeNode](#)
- [MapNode](#)
- [NullNode](#)
- [ObjectNode](#)
- [ScalarNode](#)
- [ShortNode](#)
- [StringNode](#)

- [TimestampNode](#)

com.amazonaws.services.glue.util

AWS Glue Scala 라이브러리의 com.amazonaws.services.glue.util 패키지에는 다음 API가 포함되어 있습니다.

- [GlueArgParser](#)
- [작업](#)

AWS Glue Scala ChoiceOption API

주제

- [ChoiceOption 특성](#)
- [ChoiceOption 객체](#)
- [케이스 클래스 ChoiceOptionWithResolver](#)
- [케이스 클래스 MatchCatalogSchemaChoiceOption](#)

패키지: com.amazonaws.services.glue

ChoiceOption 특성

```
trait ChoiceOption extends Serializable
```

ChoiceOption 객체

ChoiceOption

```
object ChoiceOption
```

DynamicFrame의 모든 ChoiceType 노드에 적용 가능한 선택을 확인하기 위한 일반적인 전략입니다.

- val CAST
- val MAKE_COLS
- val MAKE_STRUCT

- val MATCH_CATALOG
- val PROJECT

정의 적용

```
def apply(choice: String): ChoiceOption
```

케이스 클래스 ChoiceOptionWithResolver

```
case class ChoiceOptionWithResolver(name: String, choiceResolver: ChoiceResolver)  
  extends ChoiceOption {}
```

케이스 클래스 MatchCatalogSchemaChoiceOption

```
case class MatchCatalogSchemaChoiceOption() extends ChoiceOption {}
```

추상 DataSink 클래스

주제

- [writeDynamicFrame 정의](#)
- [pyWriteDynamicFrame 정의](#)
- [Def writeDataFrame](#)
- [Def pyWriteDataFrame](#)
- [setCatalogInfo 정의](#)
- [supportsFormat 정의](#)
- [setFormat 정의](#)
- [withFormat 정의](#)
- [setAccumulableSize 정의](#)
- [getOutputErrorRecordsAccumulable 정의](#)
- [errorsAsDynamicFrame 정의](#)
- [DataSink 객체](#)

패키지: `com.amazonaws.services.glue`

```
abstract class DataSink
```

`DataSource`에 대한 쓰기 아날로그. `DataSink`는 `DynamicFrame`이 기록될 수 있는 대상과 형식을 요약합니다.

`writeDynamicFrame` 정의

```
def writeDynamicFrame( frame : DynamicFrame,  
                      callSite : CallSite = CallSite("Not provided", "")  
                      ) : DynamicFrame
```

`pyWriteDynamicFrame` 정의

```
def pyWriteDynamicFrame( frame : DynamicFrame,  
                        site : String = "Not provided",  
                        info : String = "" )
```

Def `writeDataFrame`

```
def writeDataFrame(frame: DataFrame,  
                  glueContext: GlueContext,  
                  callSite: CallSite = CallSite("Not provided", ""))  
  ): DataFrame
```

Def `pyWriteDataFrame`

```
def pyWriteDataFrame(frame: DataFrame,  
                    glueContext: GlueContext,  
                    site: String = "Not provided",  
                    info: String = ""  
                    ): DataFrame
```

`setCatalogInfo` 정의

```
def setCatalogInfo(catalogDatabase: String,
```

```
catalogTableName : String,  
catalogId : String = "")
```

supportsFormat 정의

```
def supportsFormat( format : String ) : Boolean
```

setFormat 정의

```
def setFormat( format : String,  
              options : JsonOptions  
              ) : Unit
```

withFormat 정의

```
def withFormat( format : String,  
               options : JsonOptions = JsonOptions.empty  
               ) : DataSink
```

setAccumulableSize 정의

```
def setAccumulableSize( size : Int ) : Unit
```

getOutputErrorRecordsAccumulable 정의

```
def getOutputErrorRecordsAccumulable : Accumulable[List[OutputError], OutputError]
```

errorsAsDynamicFrame 정의

```
def errorsAsDynamicFrame : DynamicFrame
```

DataSink 객체

```
object DataSink
```

recordMetrics 정의

```
def recordMetrics( frame : DynamicFrame,
                  ctxt : String
                  ) : DynamicFrame
```

AWS Glue Scala DataSource 특성

패키지: com.amazonaws.services.glue

DynamicFrame을 생산하는데 필요한 높은 수준의 인터페이스입니다.

```
trait DataSource {

  def getDynamicFrame : DynamicFrame

  def getDynamicFrame( minPartitions : Int,
                      targetPartitions : Int
                      ) : DynamicFrame

  def getDataFrame : DataFrame

  /** @param num: the number of records for sampling.
    * @param options: optional parameters to control sampling behavior. Current
    available parameter for Amazon S3 sources in options:
    * 1. maxSamplePartitions: the maximum number of partitions the sampling will
    read.
    * 2. maxSampleFilesPerPartition: the maximum number of files the sampling will
    read in one partition.
    */
  def getSampleDynamicFrame(num:Int, options: JsonOptions = JsonOptions.empty):
  DynamicFrame

  def glueContext : GlueContext

  def setFormat( format : String,
                options : String
                ) : Unit

  def setFormat( format : String,
                options : JsonOptions
                ) : Unit
```

```
def supportsFormat( format : String ) : Boolean

def withFormat( format : String,
                options : JsonOptions = JsonOptions.empty
                ) : DataSource
}
```

AWS Glue Scala DynamicFrame API

패키지: `com.amazonaws.services.glue`

목차

- [AWS Glue Scala DynamicFrame 클래스](#)
 - [val errorsCount](#)
 - [def applyMapping](#)
 - [def assertErrorThreshold](#)
 - [def count](#)
 - [def dropField](#)
 - [def dropFields](#)
 - [def dropNulls](#)
 - [def errorsAsDynamicFrame](#)
 - [def filter](#)
 - [def getName](#)
 - [def getNumPartitions](#)
 - [def getSchemalfComputed](#)
 - [def isSchemaComputed](#)
 - [def javaToPython](#)
 - [def join](#)
 - [def map](#)
 - [def mergeDynamicFrames](#)
 - [def printSchema](#)
 - [def recomputeSchema](#)
 - [def relationalize](#)
 - [def renameField](#)

- [def repartition](#)
- [def resolveChoice](#)
- [def schema](#)
- [def selectField](#)
- [def selectFields](#)
- [def show](#)
- [Def simplifyDDBJson](#)
- [def spigot](#)
- [def splitFields](#)
- [def splitRows](#)
- [def stageErrorsCount](#)
- [def toDF](#)
- [def unbox](#)
- [def unnest](#)
- [def unnestDDBJson](#)
- [def withFrameSchema](#)
- [def withName](#)
- [def withTransformationContext](#)
- [DynamicFrame 객체](#)
 - [def apply](#)
 - [def emptyDynamicFrame](#)
 - [def fromPythonRDD](#)
 - [def ignoreErrors](#)
 - [def inlineErrors](#)
 - [def newFrameWithErrors](#)

AWS Glue Scala DynamicFrame 클래스

패키지: com.amazonaws.services.glue

```
class DynamicFrame extends Serializable with Logging {  
Scala에서 ETL    val glueContext : GlueContext,
```

```

_records : RDD[DynamicRecord],
val name : String = s"",
val transformationContext : String = DynamicFrame.UNDEFINED,
callSite : CallSite = CallSite("Not provided", ""),
stageThreshold : Long = 0,
totalThreshold : Long = 0,
prevErrors : => Long = 0,
errorExpr : => Unit = {} )

```

DynamicFrame은 자기 기술형 [DynamicRecord](#) 객체의 분산형 모음입니다.

DynamicFrame은 ETL(extract, transform, load) 작업에 유연한 데이터 모델을 제공할 수 있게끔 설계되어 있습니다. 스키마를 생성할 필요가 없고, 복잡하거나 일관되지 않은 값과 유형을 가진 데이터를 읽고 변환할 때 사용할 수 있습니다. 스키마가 필요한 작업에만 '온 디맨드(필요에 따라)'로 스키마를 계산할 수 있습니다.

DynamicFrame은 데이터 정리 및 ETL에 필요한 다양한 변환을 지원합니다. 또한 DataFrame이 제공하는 많은 분석 작업과 기존 코드를 통합할 수 있도록 SparkSQL DataFrame 변환을 지원합니다.

다음 파라미터는 DynamicFrame을 구성하는 수많은 AWS Glue 변환에서 공유됩니다.

- `transformationContext` - 이 DynamicFrame의 식별자입니다. `transformationContext`는 실행 간 지속되는 작업 북마크 상태에 대한 키로 사용됩니다.
- `callSite` - 오류 보고에 필요한 컨텍스트 정보를 제공합니다. Python에서 호출 할 때 자동으로 이 값이 설정됩니다.
- `stageThreshold` - 예외가 발생하기 전에 이 DynamicFrame 계산에 허용되는 오류 레코드의 최대 개수입니다(이전 DynamicFrame에 존재하는 레코드 제외).
- `totalThreshold` - 예외가 발생하기 전 전체 오류 레코드의 최대 개수입니다(이전 프레임의 레코드 포함).

`val errorsCount`

```
val errorsCount
```

이 DynamicFrame의 오류 레코드 개수입니다. 여기에는 이전 작업의 오류가 포함됩니다.

`def applyMapping`

```
def applyMapping( mappings : Seq[Product4[String, String, String, String]],
```

```

    caseSensitive : Boolean = true,
    transformationContext : String = "",
    callSite : CallSite = CallSite("Not provided", ""),
    stageThreshold : Long = 0,
    totalThreshold : Long = 0
  ) : DynamicFrame

```

- mappings - 새 DynamicFrame을 구성하는 매핑 시퀀스입니다.
- caseSensitive - 소스 열을 대소문자를 구분해서 처리할지 여부입니다. 이것을 false로 설정하면 AWS Glue Data Catalog 같이 대/소문자를 구분하지 않는 스토어 통합에 도움이 됩니다.

매핑 시퀀스에 따라 열을 선택, 프로젝트, 캐스트 합니다.

각 매핑은 소스 열과 유형 및 대상 열과 유형으로 구성되어 있습니다. 매핑을 4개 튜플(source_path, source_type, target_path, target_type) 또는 동일한 정보가 있는 [MappingSpec](#) 객체로 지정할 수 있습니다.

간단한 프로젝션과 캐스팅에 매핑을 사용하는 것 외에 '.'(마침표)가 있는 경로의 구성 요소를 분리해 필드를 중첩하거나 중첩을 해제할 때도 매핑을 사용할 수 있습니다.

예를 들어 DynamicFrame에 다음 스키마가 있다고 가정해보겠습니다.

```

{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |     |-- state: string
  |     |-- zip: int
}}}

```

다음 호출을 통해 state 및 zip 필드의 중첩을 해제할 수 있습니다.

```

{{{
  df.applyMapping(
    Seq(("name", "string", "name", "string"),
        ("age", "int", "age", "int"),
        ("address.state", "string", "state", "string"),
        ("address.zip", "int", "zip", "int"))
  )
}}}

```

결과 스키마는 다음과 같습니다.

```

{{{
  root
  |-- name: string
  |-- age: int
  |-- state: string
  |-- zip: int
}}}
```

또한 `applyMapping`을 사용하여 열을 다시 중첩시킬 수 있습니다. 예를 들어 다음은 이전 변환을 반전시킨 다음에 대상에 `address`라는 이름의 구조체를 생성합니다.

```

{{{
  df.applyMapping(
    Seq(("name", "string", "name", "string"),
      ("age", "int", "age", "int"),
      ("state", "string", "address.state", "string"),
      ("zip", "int", "address.zip", "int"))
  )
}}}
```

'.'(마침표) 문자가 있는 필드 이름은 백틱(` `)을 인용 부호로 사용할 수 있습니다.

Note

현재는 `applyMapping` 방법을 사용하여 어레이 아래에 중첩된 열을 매핑할 수 없습니다.

`def assertErrorThreshold`

```
def assertErrorThreshold : Unit
```

계산을 강제 실행하고 오류 레코드의 개수가 `stageThreshold` 및 `totalThreshold` 미만인지 확인하는 작업입니다. 두 조건 중 하나가 충족되지 않는 경우 예외가 발생합니다.

`def count`

```
lazy
def count
```


이 `DynamicFrame`의 요소 개수를 반환합니다.

def dropField

```
def dropField( path : String,
               transformationContext : String = "",
               callSite : CallSite = CallSite("Not provided", ""),
               stageThreshold : Long = 0,
               totalThreshold : Long = 0
             ) : DynamicFrame
```

지정한 열이 제거된 새 `DynamicFrame`을 반환합니다.

def dropFields

```
def dropFields( fieldNames : Seq[String], // The column names to drop.
                transformationContext : String = "",
                callSite : CallSite = CallSite("Not provided", ""),
                stageThreshold : Long = 0,
                totalThreshold : Long = 0
              ) : DynamicFrame
```

지정한 열이 제거된 새 `DynamicFrame`을 반환합니다.

이 방법을 사용하여 어레이 내부의 열을 포함해 중첩된 열을 삭제할 수 있지만, 지정 어레이 요소를 없앨 수는 없습니다.

def dropNulls

```
def dropNulls( transformationContext : String = "",
               callSite : CallSite = CallSite("Not provided", ""),
               stageThreshold : Long = 0,
               totalThreshold : Long = 0 )
```

null 열이 모두 제거된 새 `DynamicFrame`을 반환합니다.

Note

`NullType` 유형의 열만 제거합니다. 다른 열의 개별 null 값은 제거되거나 수정되지 않습니다.

def errorsAsDynamicFrame

```
def errorsAsDynamicFrame
```

이 DynamicFrame의 오류 레코드를 포함하는 새 DynamicFrame을 반환합니다.

def filter

```
def filter( f : DynamicRecord => Boolean,
           errorMsg : String = "",
           transformationContext : String = "",
           callSite : CallSite = CallSite("Not provided"),
           stageThreshold : Long = 0,
           totalThreshold : Long = 0
         ) : DynamicFrame
```

함수 'f'이 true를 반환하는 레코드만 포함하는 새로운 DynamicFrame을 생성합니다. 필터 함수 'f'는 입력 레코드를 바꾸지 않습니다.

def getName

```
def getName : String
```

이 DynamicFrame의 이름을 반환합니다.

def getNumPartitions

```
def getNumPartitions
```

이 DynamicFrame의 파티션 개수를 반환합니다.

def getSchemaIfComputed

```
def getSchemaIfComputed : Option[Schema]
```

계산이 완료된 스키마를 반환합니다. 스키마 계산이 완료되지 않은 경우 데이터를 스캔하지 않습니다.

def isSchemaComputed

```
def isSchemaComputed : Boolean
```

이 `DynamicFrame`에 대해 스키마가 계산되었으면 `true`를 반환하고 그렇지 않으면 `false`를 반환합니다. 이 방법이 `false`를 반환할 경우 `schema` 방법을 호출하려면 이 `DynamicFrame`의 레코드를 한 번 더 전달해야 합니다.

def javaToPython

```
def javaToPython : JavaRDD[Array[Byte]]
```

def join

```
def join( keys1 : Seq[String],
          keys2 : Seq[String],
          frame2 : DynamicFrame,
          transformationContext : String = "",
          callSite : CallSite = CallSite("Not provided", ""),
          stageThreshold : Long = 0,
          totalThreshold : Long = 0
        ) : DynamicFrame
```

- `keys1` - 이 `DynamicFrame`에서 조인에 사용할 수 있는 열입니다.
- `keys2` - `frame2`에서 조인에 사용할 수 있는 열입니다. `keys1`과 길이가 같아야 합니다.
- `frame2` - `DynamicFrame` 조인 기준입니다.

지정된 키를 사용하여 `frame2`로 동등 조인을 수행한 결과를 반환합니다.

def map

```
def map( f : DynamicRecord => DynamicRecord,
         errorMsg : String = "",
         transformationContext : String = "",
         callSite : CallSite = CallSite("Not provided", ""),
         stageThreshold : Long = 0,
         totalThreshold : Long = 0
       ) : DynamicFrame
```

이 `DynamicFrame`의 각 레코드에 지정된 함수 'f'를 적용하여 생성된 새로운 `DynamicFrame`을 반환합니다.

이 방법은 레코드가 바뀌지 않도록 각 레코드를 복사한 후에 지정된 함수를 적용합니다. 매핑 함수가 특정 레코드에 대해 예외를 발생시킨 경우, 이 레코드를 오류로 표시하고 스택 트레이스를 오류 레코드의 열로 저장합니다.

def mergeDynamicFrames

```
def mergeDynamicFrames( stageDynamicFrame: DynamicFrame, primaryKeys: Seq[String],
  transformationContext: String = "",
                        options: JsonOptions = JsonOptions.empty, callSite: CallSite =
  CallSite("Not provided"),
                        stageThreshold: Long = 0, totalThreshold: Long = 0):
  DynamicFrame
```

- `stageDynamicFrame` - 병합할 스테이징 `DynamicFrame`입니다.
- `primaryKeys` - 소스 및 스테이징 `DynamicFrame`의 레코드와 일치시킬 기본 키 필드 목록입니다.
- `transformationContext` - 현재 변환에 대한 메타데이터를 검색하는 데 사용되는 고유한 문자열입니다(선택 사항).
- `options` - 이 변환에 대한 추가 정보를 제공하는 JSON 이름-값 페어 문자열입니다.
- `callSite` - 오류 보고에 필요한 컨텍스트 정보를 제공하는 데 사용합니다.
- `stageThreshold` - Long입니다. 오류가 발생하는 지정된 변환의 오류 수입니다.
- `totalThreshold` - Long입니다. 오류가 발생하는 이 변환의 총 오류 수입니다.

레코드를 식별하기 위해, 지정된 기본 키를 기반으로 이 `DynamicFrame`을 스테이징 `DynamicFrame`과 병합합니다. 중복 레코드(기본 키가 동일한 레코드)는 중복 제거되지 않습니다. 스테이징 프레임에 일치하는 레코드가 없으면 모든 레코드(중복 레코드 포함)가 소스에서 유지됩니다. 스테이징 프레임에 일치하는 레코드가 있으면 스테이징 프레임의 레코드가 AWS Glue의 소스에 있는 레코드를 덮어씁니다.

다음과 같은 경우 반환된 `DynamicFrame`에 레코드 A가 포함되어 있습니다.

1. A가 소스 프레임과 스테이징 프레임 모두에 있는 경우에는 스테이징 프레임의 A가 반환됩니다.
2. A가 소스 테이블에 있고 A.primaryKeys가 stagingDynamicFrame에 없는 경우에는 A는 스테이징 테이블에서 업데이트되지 않습니다.

소스 프레임과 스테이징 프레임이 동일한 스키마를 가질 필요는 없습니다.

Example

```
val mergedFrame: DynamicFrame = srcFrame.mergeDynamicFrames(stageFrame, Seq("id1",
  "id2"))
```

def printSchema

```
def printSchema : Unit
```

이 DynamicFrame의 스키마를 사람이 읽을 수 있는 형식의 stdout으로 인쇄합니다.

def recomputeSchema

```
def recomputeSchema : Schema
```

스키마 재계산을 강제 실행합니다. 이를 위해 데이터를 스캔해야 하지만, 현재 스키마의 일부 필드가 데이터에 존재하지 않는 경우에는 스키마를 줄여야 할 수도 있습니다.

재계산한 스키마를 반환합니다.

def relationalize

```
def relationalize( rootTableName : String,
  stagingPath : String,
  options : JsonOptions = JsonOptions.empty,
  transformationContext : String = "",
  callSite : CallSite = CallSite("Not provided"),
  stageThreshold : Long = 0,
  totalThreshold : Long = 0
) : Seq[DynamicFrame]
```

- `rootTableName` - 출력에서 기본 DynamicFrame에 사용하는 이름. 이를 접두사로 시작하는 어레이를 피벗하여 생성되는 DynamicFrame입니다.
- `stagingPath` - 중간 데이터 쓰기를 위한 Amazon Simple Storage Service(Amazon S3) 경로입니다.
- `options` - 관계화 옵션 및 구성입니다. 현재 사용되지 않습니다.

모든 중첩된 구조와 피벗 어레이를 별개 테이블로 평면화합니다.

이 작업을 사용하여 크게 중첩된 데이터를 관계형 데이터베이스가 처리하도록 준비할 수 있습니다. 중첩된 구조를 [unnest](#) 변환과 동일한 방식으로 평면화합니다. 여기에 더해, 어레이를 각 어레이 요소가 행이 되는 별개 테이블로 피벗합니다. 예를 들어 DynamicFrame에 다음 데이터가 있다고 가정해보겠습니다.

```
{ "name": "Nancy", "age": 47, "friends": ["Fred", "Lakshmi"]}
{ "name": "Stephanie", "age": 28, "friends": ["Yao", "Phil", "Alvin"]}
{ "name": "Nathan", "age": 54, "friends": ["Nicolai", "Karen"]}
```

다음 코드를 실행합니다.

```
{{{
  df.relationalize("people", "s3:/my_bucket/my_path", JsonOptions.empty)
}}}
```

그러면 두 개 테이블이 생성됩니다. 첫 번째 테이블 이름은 "people"이며, 다음이 포함되어 있습니다.

```
{{{
  {"name": "Nancy", "age": 47, "friends": 1}
  {"name": "Stephanie", "age": 28, "friends": 2}
  {"name": "Nathan", "age": 54, "friends": 3)
}}}
```

여기에서는 친구 어레이가 자동 생성된 조인 키로 교체되어 있습니다. 다음 내용이 포함된 `people.friends`라는 이름의 별개 테이블이 생성됩니다.

```
{{{
  {"id": 1, "index": 0, "val": "Fred"}
  {"id": 1, "index": 1, "val": "Lakshmi"}
  {"id": 2, "index": 0, "val": "Yao"}
  {"id": 2, "index": 1, "val": "Phil"}
  {"id": 2, "index": 2, "val": "Alvin"}
  {"id": 3, "index": 0, "val": "Nicolai"}
  {"id": 3, "index": 1, "val": "Karen"}
}}}
```

이 테이블에서 'id'는 어레이 요소의 출처인 레코드를 식별하는 조인 키이며, 'index'는 원본 어레이에서 그 위치를 나타내고, 'val'는 실제 어레이 항목입니다.

`relationalize` 방법은 이 프로세스를 모든 어레이에 반복 적용하여 생성되는 DynamicFrame의 시퀀스를 반환합니다.

Note

AWS Glue 라이브러리는 자동으로 새 테이블에 대한 조인 키를 생성합니다. 작업 실행 간 조인 키가 고유하도록 만들려면 작업 북마크를 활성화해야 합니다.

def renameField

```
def renameField( oldName : String,
                 newName : String,
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
                 ) : DynamicFrame
```

- oldName - 열의 원래 이름입니다.
- newName - 열의 새 이름입니다.

이름이 바뀐 지정 필드로 새 DynamicFrame을 반환합니다.

이 방법을 사용하여 중첩된 필드의 이름을 바꿀 수 있습니다. 예를 들어 주소 구조체의 state라는 이름을 state_code로 바꿉니다.

```
{{{
  df.renameField("address.state", "address.state_code")
}}}
```

def repartition

```
def repartition( numPartitions : Int,
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
                 ) : DynamicFrame
```

numPartitions 파티션이 있는 새 DynamicFrame을 반환합니다.

def resolveChoice

```
def resolveChoice( specs : Seq[Product2[String, String]] = Seq.empty[ResolveSpec],
                  choiceOption : Option[ChoiceOption] = None,
                  database : Option[String] = None,
                  tableName : Option[String] = None,
                  transformationContext : String = "",
                  callSite : CallSite = CallSite("Not provided", ""),
                  stageThreshold : Long = 0,
                  totalThreshold : Long = 0
                ) : DynamicFrame
```

- choiceOption - 사양 시퀀스에 열거되지 않는 모든 ChoiceType 열에 적용되는 작업입니다.
- database - match_catalog 작업에 사용할 Data Catalog 데이터베이스입니다.
- tableName - match_catalog 작업에 사용할 Data Catalog 테이블입니다.

하나 이상의 ChoiceType을 보다 구체적인 유형으로 대체하여 새로운 DynamicFrame을 반환합니다.

resolveChoice를 사용하는 방법은 두 가지입니다. 첫 번째는 지정 열의 시퀀스와 확인 방법을 지정하는 방법입니다. (열과 작업)쌍으로 구성된 튜플로 지정합니다.

가능한 작업은 다음과 같습니다.

- cast:type - 모든 값을 지정된 유형으로 캐스트하는 시도를 합니다.
- make_cols - 각 유형을 columnName_type이라는 이름을 가진 열로 변환합니다.
- make_struct - 열을 각 유형에 대한 키가 있는 구조체로 변환합니다.
- project:type - 지정된 유형의 값만 유지합니다.

resolveChoice의 또 다른 모드는 모든 ChoiceType에 대해 단 하나의 해결책을 지정하는 것입니다. 실행 전에 ChoiceType의 완전한 목록을 모르는 경우에 사용할 수 있습니다. 이 모드는 위에서 나열한 작업 외에 다음 작업을 지원합니다.

- match_catalog - 각 ChoiceType을 지정된 카탈로그 테이블의 해당 유형에 캐스팅해봅니다.

예제:

int로 캐스팅을 해서 user.id 열을 확인하고, address 필드에는 구조체만 유지합니다.


```

{{{
  df.resolveChoice(specs = Seq(("user.id", "cast:int"), ("address", "project:struct")))
}}}
```

각 선택을 개별 열로 변환하여 전체 ChoiceType을 확인합니다.

```

{{{
  df.resolveChoice(choiceOption = Some(ChoiceOption("make_cols")))
}}}
```

지정된 카탈로그 테이블의 유형으로 캐스팅하여 전체 ChoiceType을 확인합니다.

```

{{{
  df.resolveChoice(choiceOption = Some(ChoiceOption("match_catalog")),
                  database = Some("my_database"),
                  tableName = Some("my_table"))
}}}
```

def schema

```
def schema : Schema
```

이 DynamicFrame의 스키마를 반환합니다.

반환되는 스키마에는 이 DynamicFrame의 레코드에 있는 모든 필드가 있습니다. 단, 많지는 않지만 추가 필드가 포함되는 경우도 있을 수 있습니다. [unnest](#) 방법을 사용하여 이 DynamicFrame의 레코드에 따라 스키마를 "줄일" 수 있습니다.

def selectField

```

def selectField( fieldName : String,
                transformationContext : String = "",
                callSite : CallSite = CallSite("Not provided", ""),
                stageThreshold : Long = 0,
                totalThreshold : Long = 0
                ) : DynamicFrame
```

단일 필드를 DynamicFrame으로 반환합니다.

def selectFields

```
def selectFields( paths : Seq[String],
                 transformationContext : String = "",
                 callSite : CallSite = CallSite("Not provided", ""),
                 stageThreshold : Long = 0,
                 totalThreshold : Long = 0
                 ) : DynamicFrame
```

- paths - 선택할 열 이름의 시퀀스입니다.

지정한 열이 포함된 새 DynamicFrame을 반환합니다.

Note

최상위 열에만 selectFields 방법을 사용할 수 있습니다. [applyMapping](#) 방법을 사용하여 중첩된 열을 선택할 수 있습니다.

def show

```
def show( numRows : Int = 20 ) : Unit
```

- numRows - 인쇄할 행의 개수입니다.

이 DynamicFrame의 행을 JSON 형식으로 인쇄합니다.

Def simplifyDDBJson

AWS Glue DynamoDB 내보내기 커넥터를 사용하여 DynamoDB 내보내기를 수행하면 특정 중첩 구조의 JSON 파일이 생성됩니다. 자세한 내용은 [데이터 객체](#)를 참조하세요. simplifyDDBJson 이러한 유형의 데이터에 대한 DynamicFrame의 중첩 열을 단순화하고 단순화된 새로운 DynamicFrame을 반환합니다. 목록 유형에 여러 유형이 있거나 맵 유형이 포함된 경우 목록의 요소는 단순화되지 않습니다. 이 메서드는 DynamoDB 내보내기 JSON 형식의 데이터만 지원합니다. 다른 종류의 데이터에 대해 유사한 변경을 수행하려면 unnest를 고려하세요.

```
def simplifyDDBJson() : DynamicFrame
```

이 메서드에서는 파라미터를 사용하지 않습니다.

입력 예

DynamoDB 내보내기로 생성된 다음 스키마를 고려해 보세요.

```

root
|-- Item: struct
|   |-- parentMap: struct
|   |   |-- M: struct
|   |   |   |-- childMap: struct
|   |   |   |   |-- M: struct
|   |   |   |   |   |-- appName: struct
|   |   |   |   |   |   |-- S: string
|   |   |   |   |   |   |-- packageName: struct
|   |   |   |   |   |   |   |-- S: string
|   |   |   |   |   |   |   |-- updatedAt: struct
|   |   |   |   |   |   |   |   |-- N: string
|   |   |-- strings: struct
|   |   |   |-- SS: array
|   |   |   |   |-- element: string
|   |   |-- numbers: struct
|   |   |   |-- NS: array
|   |   |   |   |-- element: string
|   |   |-- binaries: struct
|   |   |   |-- BS: array
|   |   |   |   |-- element: string
|   |-- isDDBJson: struct
|   |   |-- BOOL: boolean
|   |-- nullValue: struct
|   |   |-- NULL: boolean

```

예제 코드

```

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext import scala.collection.JavaConverters._

object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())

```

```

val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
Job.init(args("JOB_NAME"), glueContext, args.asJava)

val dynamicFrame = glueContext.getSourceWithFormat(
  connectionType = "dynamodb",
  options = JsonOptions(Map(
    "dynamodb.export" -> "ddb",
    "dynamodb.tableArn" -> "ddbTableARN",
    "dynamodb.s3.bucket" -> "exportBucketLocation",
    "dynamodb.s3.prefix" -> "exportBucketPrefix",
    "dynamodb.s3.bucketOwner" -> "exportBucketAccountID",
  ))
).getDynamicFrame()

val simplified = dynamicFrame.simplifyDDBJson()
simplified.printSchema()

Job.commit()
}
}

```

출력 예시

simplifyDDBJson 변환은 이를 다음과 같이 간소화합니다.

```

root
|-- parentMap: struct
|   |-- childMap: struct
|   |   |-- appName: string
|   |   |-- packageName: string
|   |   |-- updatedAt: string
|-- strings: array
|   |-- element: string
|-- numbers: array
|   |-- element: string
|-- binaries: array
|   |-- element: string
|-- isDDBJson: boolean
|-- nullValue: null

```

def spigot

```
def spigot( path : String,
           options : JsonOptions = new JsonOptions("{}"),
           transformationContext : String = "",
           callSite : CallSite = CallSite("Not provided"),
           stageThreshold : Long = 0,
           totalThreshold : Long = 0
         ) : DynamicFrame
```

동일한 레코드를 반환하지만 부수적으로 레코드의 하위 집합을 작성하는 패스스루 변환입니다.

- path - s3://bucket//path 형식으로 출력을 쓸 Amazon S3의 경로입니다.
- options - 샘플링 동작을 설명하는 선택 사항인 JsonOptions 맵입니다.

이것과 동일한 레코드가 포함된 DynamicFrame을 반환합니다.

path로 지정한 위치에 임의의 레코드 100개를 작성하는 것이 기본값입니다. options 맵을 사용하여 이 동작을 사용자 지정할 수 있습니다. 유효한 키에는 다음이 포함되어 있습니다.

- topk - 작성할 레코드의 총 개수를 지정합니다. 기본값은 100입니다.
- prob - 개별 레코드의 포함 확률(십진수)을 지정합니다. 기본값은 1.

예를 들어 다음 호출은 20퍼센트의 확률로 각 레코드를 선택하고 200개의 레코드를 작성한 후에 중단하는 방법으로 데이터 세트를 샘플링합니다.

```
{{{
  df.spigot("s3://my_bucket/my_path", JsonOptions(Map("topk" -> 200, "prob" ->
    0.2)))
}}}
```

def splitFields

```
def splitFields( paths : Seq[String],
               transformationContext : String = "",
               callSite : CallSite = CallSite("Not provided", ""),
               stageThreshold : Long = 0,
               totalThreshold : Long = 0
             ) : Seq[DynamicFrame]
```

- `paths` - 첫 `DynamicFrame`에 포함될 경로입니다.

두 개 `DynamicFrame`의 시퀀스를 반환합니다. 첫 번째 `DynamicFrame`에는 지정 경로가, 두 번째에는 나머지 열 전부가 포함됩니다.

예

이 예제에서는 AWS Glue Data Catalog의 `legislators` 데이터베이스에 있는 `persons` 테이블에서 생성된 `DynamicFrame`을 사용하고 두 개의 `DynamicFrame`으로 분할합니다. 지정된 필드는 첫 번째 `DynamicFrame`으로 이동하고, 나머지 필드는 두 번째 `DynamicFrame`으로 이동합니다. 그런 다음, 예제는 결과에서 첫 번째 `DynamicFrame`을 선택합니다.

```
val InputFrame = glueContext.getCatalogSource(database="legislators",
    tableName="persons",
    transformationContext="InputFrame").getDynamicFrame()

val SplitField_collection = InputFrame.splitFields(paths=Seq("family_name", "name",
    "links.note",
    "links.url", "gender", "image", "identifiers.scheme", "identifiers.identifier",
    "other_names.lang",
    "other_names.note", "other_names.name"), transformationContext="SplitField_collection")

val ResultFrame = SplitField_collection(0)
```

`def splitRows`

```
def splitRows( paths : Seq[String],
    values : Seq[Any],
    operators : Seq[String],
    transformationContext : String,
    callSite : CallSite,
    stageThreshold : Long,
    totalThreshold : Long
    ) : Seq[DynamicFrame]
```

열을 상수에 비교하는 조건자에 따라 행을 분할합니다.

- `paths` - 비교에 사용할 열입니다.
- `values` - 비교에 사용할 상수 값입니다.
- `operators` - 비교에 사용할 연산자입니다.

두 개 DynamicFrame의 시퀀스를 반환합니다. 첫 번째에는 조건자가 true, 두 번째에는 조건자가 false 인 행이 포함됩니다.

조건자는 다음 세 개의 시퀀스를 사용하여 지정됩니다. 'paths'에는 (중첩될 수 있는) 열 이름이 포함되고, 'values'에는 비교할 상수 값이 포함되며, 'operators'에는 비교에 사용할 연산자가 포함됩니다. 세 가지 시퀀스는 모두 길이가 같아야 합니다. n번째 연산자를 사용하여 n번째 열과 n번째 값을 비교합니다.

각 연산자는 "!=", "=", "<=", "<", ">=" 또는 ">" 중 하나여야 합니다.

예를 들어 다음 호출은 첫 번째 출력 프레임에는 65세 이상의 미국인에 대한 레코드가 포함되고 두 번째에는 나머지 레코드 전부가 포함되도록 DynamicFrame을 분할합니다.

```

{{{
  df.splitRows(Seq("age", "address.country"), Seq(65, "USA"), Seq(">=", "="))
}}}
```

def stageErrorsCount

```
def stageErrorsCount
```

이 DynamicFrame을 계산하는 동안 생성된 오류 레코드의 개수를 반환합니다. 이 DynamicFrame에 입력으로 전달된 이전 작업의 오류는 제외됩니다.

def toDF

```
def toDF( specs : Seq[ResolveSpec] = Seq.empty[ResolveSpec] ) : DataFrame
```

이 DynamicFrame을 스키마와 레코드가 동일한 Apache Spark SQL DataFrame으로 변환합니다.

Note

DataFrame은 ChoiceType을 지원하지 않으므로 이 방법은 ChoiceType 열을 StructType로 자동 변환합니다. 선택할 수 있는 확인(해결) 방법에 대한 자세한 내용은 [resolveChoice](#) 단원을 참조하십시오.

def unbox

```
def unbox( path : String,
```

```

format : String,
optionString : String = "{}",
transformationContext : String = "",
callSite : CallSite = CallSite("Not provided"),
stageThreshold : Long = 0,
totalThreshold : Long = 0
) : DynamicFrame

```

- path - 구문 분석할 열입니다. 문자열 또는 이진수여야 합니다.
- format - 구문 분석에 사용할 포맷입니다.
- optionString - CSV 구분자 같은 포맷으로 전달되는 옵션입니다.

지정 형식에 따라 포함된 문자열이나 이진수 열을 구문 분석합니다. 구문 분석한 열은 원래 열 이름을 가진 구조체 아래 중첩됩니다.

예를 들어 JSON 열이 포함된 CSV 파일이 있다고 가정해보겠습니다.

```

name, age, address
Sally, 36, {"state": "NE", "city": "Omaha"}
...

```

최초 구문 분석 후에 다음 스키마를 갖는 DynamicFrame이 생깁니다.

```

{{{
  root
  |-- name: string
  |-- age: int
  |-- address: string
}}

```

주소 열에 unbox를 호출해 지정 구성 요소를 구문 분석할 수 있습니다.

```

{{{
  df.unbox("address", "json")
}}

```

그러면 다음 스키마를 갖는 DynamicFrame이 생깁니다.

```

{{{
  root

```



```

|-- name: string
|-- age: int
|-- address: struct
|   |-- state: string
|   |-- city: string
}}}
```

def unnest

```

def unnest( transformationContext : String = "",
            callSite : CallSite = CallSite("Not Provided"),
            stageThreshold : Long = 0,
            totalThreshold : Long = 0
            ) : DynamicFrame
```

중첩된 구조가 모두 평면화된 새 DynamicFrame이 반환됩니다. 이름은 '.'(마침표) 문자를 사용하여 생성합니다.

예를 들어 DynamicFrame에 다음 스키마가 있다고 가정해보겠습니다.

```

{{{
  root
  |-- name: string
  |-- age: int
  |-- address: struct
  |   |-- state: string
  |   |-- city: string
}}}
```

다음 호출은 주소 구조체의 중첩을 해제합니다.

```

{{{
  df.unnest()
}}}
```

결과 스키마는 다음과 같습니다.

```

{{{
  root
  |-- name: string
  |-- age: int
```

```

|-- address.state: string
|-- address.city: string
}}

```

이 방법도 어레이 내부에 중첩된 구조체의 중첩을 해제합니다. 그러나 기록을 위해 이러한 필드는 포함되는 어레이의 이름과 ".val"이 이름 앞에 붙습니다.

def unnestDDBJson

```

unnestDDBJson(transformationContext : String = "",
              callSite : CallSite = CallSite("Not Provided"),
              stageThreshold : Long = 0,
              totalThreshold : Long = 0): DynamicFrame

```

DynamoDB JSON 구조의 DynamicFrame에 있는 중첩된 열을 한정해서 중첩 해제하고, 중첩되지 않은 새 DynamicFrame을 반환합니다. 구조체 유형 배열인 열은 중첩 해제되지 않습니다. 이 변환은 일반적인 unnest 변환과는 다르게 작동하는 특정 유형의 중첩 해제 변환이며, 데이터가 이미 DynamoDB JSON 구조에 있어야 합니다. 자세한 내용은 [DynamoDB JSON](#)을 참조하세요.

예를 들어, DynamoDB JSON 구조를 사용하여 내보내기를 읽는 스키마는 다음과 같을 수 있습니다.

```

root
|-- Item: struct
|   |-- ColA: struct
|   |   |-- S: string
|   |-- ColB: struct
|   |   |-- S: string
|   |-- ColC: struct
|   |   |-- N: string
|   |-- ColD: struct
|   |   |-- L: array
|   |   |   |-- element: null

```

unnestDDBJson() 변환은 이 구조를 다음과 같이 변환합니다.

```

root
|-- ColA: string
|-- ColB: string
|-- ColC: string
|-- ColD: array
|   |-- element: null

```

다음 코드 예제에서는 AWS Glue DynamoDB 내보내기 커넥터를 사용하고, DynamoDB JSON unnest를 호출하고, 파티션 수를 인쇄하는 방법을 보여줍니다.

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.export" -> "ddb",
        "dynamodb.tableArn" -> "<test_source>",
        "dynamodb.s3.bucket" -> "<bucket name>",
        "dynamodb.s3.prefix" -> "<bucket prefix>",
        "dynamodb.s3.bucketOwner" -> "<account_id of bucket>",
      ))
    ).getDynamicFrame()

    val unnested = dynamicFrame.unnestDDBJson()
    print(unnested.getNumPartitions())

    Job.commit()
  }
}
```

def withFrameSchema

```
def withFrameSchema( getSchema : () => Schema ) : DynamicFrame
```

- `getSchema` - 사용할 스키마를 반환하는 함수입니다. 0이라는 파라미터 함수로 지정을 하면 비용이 많이 드는 계산을 연기할 수 있습니다.

이 `DynamicFrame`의 스키마를 지정된 값으로 설정합니다. 비용이 많이 드는 스키마 재계산을 피하기 위해 주로 내부에서 사용합니다. 통과된 스키마에는 데이터에 존재하는 모든 열이 포함되어 있어야 합니다.

def withName

```
def withName( name : String ) : DynamicFrame
```

- `name` - 사용할 새 이름입니다.

새 이름이 있는 이 `DynamicFrame`의 복사본을 반환합니다.

def withTransformationContext

```
def withTransformationContext( ctx : String ) : DynamicFrame
```

지정된 변환 컨텍스트가 있는 이 `DynamicFrame`의 복사본을 반환합니다.

DynamicFrame 객체

패키지: `com.amazonaws.services.glue`

```
object DynamicFrame
```

def apply

```
def apply( df : DataFrame,  
          glueContext : GlueContext  
          ) : DynamicFrame
```

def emptyDynamicFrame

```
def emptyDynamicFrame( glueContext : GlueContext ) : DynamicFrame
```

def fromPythonRDD

```
def fromPythonRDD( rdd : JavaRDD[Array[Byte]],
                  glueContext : GlueContext
                  ) : DynamicFrame
```

def ignoreErrors

```
def ignoreErrors( fn : DynamicRecord => DynamicRecord ) : DynamicRecord
```

def inlineErrors

```
def inlineErrors( msg : String,
                 callSite : CallSite
                 ) : (DynamicRecord => DynamicRecord)
```

def newFrameWithErrors

```
def newFrameWithErrors( prevFrame : DynamicFrame,
                        rdd : RDD[DynamicRecord],
                        name : String = "",
                        transformationContext : String = "",
                        callSite : CallSite,
                        stageThreshold : Long,
                        totalThreshold : Long
                        ) : DynamicFrame
```

AWS Glue Scala DynamicRecord 클래스

주제

- [def addField](#)
- [def dropField](#)
- [def setError](#)
- [def isError](#)
- [def getError](#)
- [def clearError](#)

- [def write](#)
- [def readFields](#)
- [def clone](#)
- [def schema](#)
- [def getRoot](#)
- [def toJson](#)
- [def getFieldNode](#)
- [def getField](#)
- [def hashCode](#)
- [def equals](#)
- [DynamicRecord 객체](#)
- [RecordTraverser 특성](#)

패키지: `com.amazonaws.services.glue`

```
class DynamicRecord extends Serializable with Writable with Cloneable
```

`DynamicRecord`는 진행된 데이터 세트의 데이터 행을 나타내는 자체적으로 설명되는 데이터 구조입니다. 기록 자체를 검사한 `DynamicRecord`에 의해 보여지는 스키마 행을 얻을 수 있다는 의미에서 자체 설명적입니다. `DynamicRecord`는 Apache Spark의 `Row`와 유사합니다.

`def addField`

```
def addField( path : String,
              dynamicNode : DynamicNode
              ) : Unit
```

지정된 경로로 [DynamicNode](#)를 추가합니다.

- `path` - 필드를 추가할 경로입니다.
- `dynamicNode` - 지정된 경로에 추가될 [DynamicNode](#)입니다.

`def dropField`

```
def dropField(path: String, underRename: Boolean = false): Option[DynamicNode]
```

지정된 경로로부터 [DynamicNode](#)을 드롭하고 지정된 경로에 배열이 없으면 드롭된 노트를 반환합니다.

- path - 필드를 제거할 경로입니다.
- underRename - dropField가 이름 변경 변환의 일부로 호출되는 경우는 true이고, 아닌 경우는 false입니다(기본값은 false).

scala.Option Option([DynamicNode](#))을 반환합니다.

def setError

```
def setError( error : Error )
```

error 파라미터에 의해 지정되어 이 기록을 오류 기록으로 설정합니다.

DynamicRecord을 반환합니다.

def isError

```
def isError
```

이 기록이 오류 기록인지 확인합니다.

def getError

```
def getError
```

이 기록이 오류 기록이면 Error를 얻습니다. 이 기록이 오류 기록이면 scala.Some Some(오류)를 반환하고 그렇지 않으면 scala.None입니다.

def clearError

```
def clearError
```

Error을 scala.None.None로 설정합니다.

def write

```
override def write( out : DataOutput ) : Unit
```

def readFields

```
override def readFields( in : DataInput ) : Unit
```

def clone

```
override def clone : DynamicRecord
```

이 레코드를 새 `DynamicRecord`로 복제하고 반환합니다.

def schema

```
def schema
```

이 기록을 검사하여 `Schema`를 얻습니다.

def getRoot

```
def getRoot : ObjectNode
```

이 레코드의 루트 `ObjectNode`를 얻습니다.

def toJson

```
def toJson : String
```

이 레코드의 JSON 문자열을 얻습니다.

def getFieldNode

```
def getFieldNode( path : String ) : Option[DynamicNode]
```

`DynamicNode`의 옵션으로 지정된 `path`의 필드 값을 가져옵니다.

필드가 존재하는 경우 `scala.Some Some(DynamicNode)`을 반환하고, 그렇지 않으면 `scala.None.None`입니다.

def getField

```
def getField( path : String ) : Option[Any]
```

DynamicNode의 옵션으로 지정된 path의 필드 값을 가져옵니다.

scala.Some Some(값)을 반환합니다.

def hashCode

```
override def hashCode : Int
```

def equals

```
override def equals( other : Any )
```

DynamicRecord 객체

```
object DynamicRecord
```

정의 적용

```
def apply( row : Row,  
          schema : SparkStructType )
```

Apache Spark SQL Row를 [DynamicRecord](#)로 변환하는 방법을 적용합니다.

- row - Spark SQL Row.
- schema - 행의 Schema입니다.

DynamicRecord을 반환합니다.

RecordTraverser 특성

```
trait RecordTraverser {  
  def nullValue(): Unit
```

```

def byteValue(value: Byte): Unit
def binaryValue(value: Array[Byte]): Unit
def booleanValue(value: Boolean): Unit
def shortValue(value: Short) : Unit
def intValue(value: Int) : Unit
def longValue(value: Long) : Unit
def floatValue(value: Float): Unit
def doubleValue(value: Double): Unit
def decimalValue(value: BigDecimal): Unit
def stringValue(value: String): Unit
def dateValue(value: Date): Unit
def timestampValue(value: Timestamp): Unit
def objectStart(length: Int): Unit
def objectKey(key: String): Unit
def objectEnd(): Unit
def mapStart(length: Int): Unit
def mapKey(key: String): Unit
def mapEnd(): Unit
def arrayStart(length: Int): Unit
def arrayEnd(): Unit
}

```

AWS Glue스칼라 API GlueContext

패키지: `com.amazonaws.services.glue`

```

class GlueContext extends SQLContext(sc) (
    @transient val sc : SparkContext,
    val defaultSourcePartitioner : PartitioningStrategy )

```

GlueContext는 Amazon Simple Storage Service(Amazon S3), AWS Glue Data Catalog, JDBC 등에서 [DynamicFrame](#)을 읽고 쓰기 위한 진입점입니다. 이 클래스는 [DataSource 특성](#) 및 [DataSink](#) 객체를 생성하는 유틸리티 함수를 제공하며, 이 객체는 DynamicFrame을 읽고 쓸 때 사용할 수 있습니다.

GlueContext를 사용하여 DynamicFrame에 목표한 수의 파티션(기본값 20)을 설정할 수도 있습니다. 단, 소스에서 생성된 파티션 수가 파티션 최소 임계값(기본값 10) 미만이어야 합니다.

데프 컬럼 addIngestionTime

```

def addIngestionTimeColumns(
    df : DataFrame,
    timeGranularity : String = "") : DataFrame

```

입력 DataFrame에 ingest_year, ingest_month, ingest_day, ingest_hour, ingest_minute와 같은 수집 시간 열을 추가합니다. 이 함수는 Amazon S3을 대상으로 하는 데이터 카탈로그 테이블을 지정할 때 AWS Glue가 생성하는 스크립트에서 자동으로 생성됩니다. 이 함수는 출력 테이블의 수집 시간 열로 파티션을 자동으로 업데이트합니다. 이를 통해 입력 데이터에 명시적인 수집 시간 열을 요구하지 않고도 출력 데이터를 수집 시간에 자동으로 분할할 수 있습니다.

- `dataFrame` - 수집 시간 열을 추가할 `dataFrame`입니다.
- `timeGranularity` - 시간 열의 세분성입니다. 유효 값은 "day", "hour" 및 "minute"입니다. 예를 들어 "hour"가 함수에 전달되면 원래 `dataFrame`에 "ingest_year", "ingest_month", "ingest_day" 및 "ingest_hour" 시간 열이 추가됩니다.

시간 세분성 열을 추가한 후 데이터 프레임을 반환합니다.

예제

```
glueContext.addIngestionTimeColumns(dataFrame, "hour")
```

데프 createDataFrame FromOptions

```
def createDataFrameFromOptions( connectionType : String,
                                connectionOptions : JsonOptions,
                                transformationContext : String = "",
                                format : String = null,
                                formatOptions : JsonOptions = JsonOptions.empty
                                ) : DataSource
```

지정된 연결 및 포맷으로 생성된 DataFrame을 반환합니다. 이 함수는 AWS Glue 스트리밍 소스에서 만 사용하십시오.

- `connectionType` - 스트리밍 연결 유형입니다. 유효한 값에는 `kinesis` 및 `kafka(이)`가 있습니다.
- `connectionOptions` - 연결 옵션이며, Kinesis 및 Kafka에 대해 서로 다릅니다. [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)에서 각 스트리밍 데이터 원본에 대한 모든 연결 옵션 목록을 찾을 수 있습니다. 스트리밍 연결 옵션에는 다음과 같은 차이점이 있습니다.
 - Kinesis 스트리밍 소스에는 `streamARN`, `startingPosition`, `inferSchema` 및 `classification`이 필요합니다.
 - Kafka 스트리밍 소스에는 `connectionName`, `topicName`, `startingOffsets`, `inferSchema` 및 `classification`이 필요합니다.

- `transformationContext` - 사용할 변환 컨텍스트입니다(선택 사항).
- `format` - 형식 사양입니다(선택 사항). 여러 포맷을 지원하는 Amazon S3 또는 AWS Glue 연결에 사용됩니다. 지원되는 형식에 관한 내용은 [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#) 섹션을 참조하세요.
- `formatOptions` - 지정된 형식에 대한 형식 옵션입니다. 지원되는 포맷 옵션에 대한 자세한 내용은 [데이터 포맷 옵션](#) 섹션을 참조하세요.

Amazon Kinesis 스트리밍 소스의 예:

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
connectionType = "kinesis",
connectionOptions = JsonOptions("""{"streamName": "example_stream", "startingPosition":
"TRIM_HORIZON", "inferSchema": "true", "classification": "json"}"""))
```

Kafka 스트리밍 소스의 예:

```
val data_frame_datasource0 =
glueContext.createDataFrameFromOptions(transformationContext = "datasource0",
connectionType = "kafka",
connectionOptions = JsonOptions("""{"connectionName": "example_connection",
"topicName": "example_topic", "startingPosition": "earliest", "inferSchema": "false",
"classification": "json", "schema": "`column1` STRING, `column2` STRING"}"""))
```

`forEachBatch`

forEachBatch(frame, batch_function, options)

스트리밍 소스에서 읽는 모든 마이크로 배치에 전달된 `batch_function`을 적용합니다.

- `frame`— 현재 마이크로 배치를 `DataFrame` 포함하고 있습니다.
- `batch_function` - 모든 마이크로 배치에 적용될 함수입니다.
- `options` - 마이크로 배치를 처리하는 방법에 대한 정보가 들어 있는 키-값 페어 컬렉션입니다. 다음 옵션이 필요합니다.
 - `windowSize` - 각 배치를 처리하는 데 소요되는 시간입니다.
 - `checkpointLocation` - 스트리밍 ETL 작업에 대해 체크포인트가 저장되는 위치입니다.
 - `batchMaxRetries` - 실패한 경우 배치를 다시 시도할 수 있는 최대 횟수입니다. 기본값은 3입니다. 이 옵션은 Glue 버전 2.0 이상에서만 구성할 수 있습니다.

예:

```
glueContext.forEachBatch(data_frame_datasource0, (dataFrame: Dataset[Row], batchId:
Long) =>
{
  if (dataFrame.count() > 0)
  {
    val datasource0 = DynamicFrame(glueContext.addIngestionTimeColumns(dataFrame,
"hour"), glueContext)
    // @type: DataSink
    // @args: [database = "tempdb", table_name = "fromoptionsoutput",
stream_batch_time = "100 seconds",
//      stream_checkpoint_location = "s3://from-options-testing-eu-central-1/
fromOptionsOutput/checkpoint/",
//      transformation_ctx = "datasink1"]
    // @return: datasink1
    // @inputs: [frame = datasource0]
    val options_datasink1 = JsonOptions(
      Map("partitionKeys" -> Seq("ingest_year", "ingest_month","ingest_day",
"ingest_hour"),
        "enableUpdateCatalog" -> true))
    val datasink1 = glueContext.getCatalogSink(
      database = "tempdb",
      tableName = "fromoptionsoutput",
      redshiftTmpDir = "",
      transformationContext = "datasink1",
      additionalOptions = options_datasink1).writeDynamicFrame(datasource0)
  }
}, JsonOptions("""{"windowSize" : "100 seconds",
  "checkpointLocation" : "s3://from-options-testing-eu-central-1/
fromOptionsOutput/checkpoint/"}"""))
```

def getCatalogSink

```
def getCatalogSink( database : String,
  tableName : String,
  redshiftTmpDir : String = "",
  transformationContext : String = ""
  additionalOptions: JsonOptions = JsonOptions.empty,
  catalogId: String = null
) : DataSink
```

Data Catalog에 정의된 테이블에 지정된 위치에 쓰는 [DataSink](#)를 생성합니다.

- `database` - Data Catalog에 있는 데이터베이스 이름입니다.
- `tableName` - Data Catalog에 있는 테이블 이름입니다.
- `redshiftTmpDir` - 특정 데이터 싱크와 사용되는 임시 준비 디렉터리입니다. 기본적으로 비우도록 설정합니다.
- `transformationContext` - 작업 북마크에서 사용되는 싱크와 관련된 변환 컨텍스트입니다. 기본적으로 비우도록 설정합니다.
- `additionalOptions` - AWS Glue에 제공되는 추가 옵션입니다.
- `catalogId` - 액세스 중인 Data Catalog의 카탈로그 ID(계정 ID)입니다. null인 경우 호출자의 기본 계정 ID가 사용됩니다.

`DataSink`을 반환합니다.

데프 `getCatalogSource`

```
def getCatalogSource( database : String,
                      tableName : String,
                      redshiftTmpDir : String = "",
                      transformationContext : String = ""
                      pushDownPredicate : String = " "
                      additionalOptions: JsonOptions = JsonOptions.empty,
                      catalogId: String = null
                      ) : DataSource
```

Data Catalog의 테이블 정의에서 데이터를 읽는 [DataSource 특성](#)을 생성합니다.

- `database` - Data Catalog에 있는 데이터베이스 이름입니다.
- `tableName` - Data Catalog에 있는 테이블 이름입니다.
- `redshiftTmpDir` - 특정 데이터 싱크와 사용되는 임시 준비 디렉터리입니다. 기본적으로 비우도록 설정합니다.
- `transformationContext` - 작업 북마크에서 사용되는 싱크와 관련된 변환 컨텍스트입니다. 기본적으로 비우도록 설정합니다.
- `pushDownPredicate` - 데이터 집합 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링할 수 있습니다. 자세한 정보는 [푸시다운 조건자를 사용하여 예비 필터링](#)을 참조하세요.
- `additionalOptions` - 선택적 이름-값 페어의 모음입니다. 가능한 옵션에는 [AWS Glue for Spark 에서 ETL에 대한 연결 유형 및 옵션](#)에 나열된 옵션이 포함됩니다(`endpointUrl`, `streamName`,

bootstrap.servers, security.protocol, topicName, classification 및 delimiter 제외). 지원되는 또 다른 옵션은 catalogPartitionPredicate입니다.

catalogPartitionPredicate - 카탈로그 표현식을 전달하여 인덱스 열을 기준으로 필터링할 수 있습니다. 이렇게 하면 필터링이 서버 측으로 푸시됩니다. 자세한 내용은 [AWS Glue 파티션 인덱스](#)를 참조하세요. push_down_predicate와 catalogPartitionPredicate는 다른 구문을 사용합니다. 전자는 Spark SQL 표준 구문을 사용하고 후자는 JSQL 구문 분석기를 사용합니다.

- catalogId - 액세스 중인 Data Catalog의 카탈로그 ID(계정 ID)입니다. null인 경우 호출자의 기본 계정 ID가 사용됩니다.

DataSource을 반환합니다.

스트리밍 소스의 예

```
val data_frame_datasource0 = glueContext.getCatalogSource(
  database = "tempdb",
  tableName = "test-stream-input",
  redshiftTmpDir = "",
  transformationContext = "datasource0",
  additionalOptions = JsonOptions("""{
    "startingPosition": "TRIM_HORIZON", "inferSchema": "false"}""")
).getDataFrame()
```

def getJDBCSink

```
def getJDBCSink( catalogConnection : String,
  options : JsonOptions,
  redshiftTmpDir : String = "",
  transformationContext : String = "",
  catalogId: String = null
) : DataSink
```

Data Catalog의 Connection 객체에 지정된 JDBC 데이터베이스에 쓰는 [DataSink](#)를 생성합니다. Connection 객체는 URL, 사용자 이름, 암호, VPC, 서브넷 및 보안 그룹을 포함한 JDBC 싱크로 연결하는 정보를 가지고 있습니다.

- catalogConnection - 작성할 JDBC URL을 포함하는 Data Catalog의 연결 이름입니다.
- options - JDBC 데이터 스토어로 작성할 때 필요한 추가 정보를 제공하는 JSON 이름-값 페어의 문자열입니다. 여기에는 다음이 포함됩니다.

- `dbtable`(필수) - JDBC 테이블의 이름입니다. 데이터베이스 내의 스키마를 지원하는 JDBC 데이터 스토어의 경우 `schema.table-name`에 대해 지정합니다. 스키마가 제공되지 않으면 기본 "퍼블릭" 스키마가 사용됩니다. 다음 예제에서는 `test`라는 스키마와 `test_db` 데이터베이스의 `test_table` 테이블을 가리키는 옵션 파라미터를 보여 줍니다.

```
options = JsonOptions("""{"dbtable": "test.test_table", "database": "test_db"}""")
```

- `database`(필수) - JDBC 데이터베이스의 이름입니다.
- SparkSQL JDBC 라이터에게 직접 전달되는 추가 옵션. 자세한 내용은 [Redshift data source for Spark](#)를 참조하십시오.
- `redshiftTmpDir` - 특정 데이터 싱크와 사용되는 임시 준비 디렉터리입니다. 기본적으로 비우도록 설정합니다.
- `transformationContext` - 작업 북마크에서 사용되는 싱크와 관련된 변환 컨텍스트입니다. 기본적으로 비우도록 설정합니다.
- `catalogId` - 액세스 중인 Data Catalog의 카탈로그 ID(계정 ID)입니다. null인 경우 호출자의 기본 계정 ID가 사용됩니다.

예제 코드:

```
getJDBCSink(catalogConnection = "my-connection-name", options =
  JsonOptions("""{"dbtable": "my-jdbc-table", "database": "my-jdbc-db"}"""),
  redshiftTmpDir = "", transformationContext = "datasink4")
```

`DataSink`을 반환합니다.

def `getSink`

```
def getSink( connectionType : String,
             connectionOptions : JsonOptions,
             transformationContext : String = ""
           ) : DataSink
```

Amazon Simple Storage Service (Amazon S3), JDBC, AWS Glue 데이터 카탈로그, 아파치 카프카 또는 Amazon Kinesis 데이터 스트림과 같은 대상에 데이터를 쓰는 데이터를 생성합니다. [DataSink](#)

- `connectionType` - 연결 유형입니다. [the section called “연결 파라미터”](#) 섹션을 참조하십시오.

- `connectionOptions` - 데이터 싱크 연결하려는 추가 정보를 제공하는 JSON 이름-값 페어 문자열입니다. [the section called “연결 파라미터”](#) 섹션을 참조하십시오.
- `transformationContext` - 작업 북마크에서 사용되는 싱크와 관련된 변환 컨텍스트입니다. 기본적으로 비우도록 설정합니다.

`DataSink`을 반환합니다.

`getSinkWithdef` 포맷

```
def getSinkWithFormat( connectionType : String,
                      options : JsonOptions,
                      transformationContext : String = "",
                      format : String = null,
                      formatOptions : JsonOptions = JsonOptions.empty
                      ) : DataSink
```

Amazon S3, JDBC, 데이터 카탈로그나 Apache Kafka 또는 Amazon Kinesis 데이터 스트림과 같은 대상에 데이터를 쓰는 [DataSink](#)를 생성합니다. 또한 대상에 기록할 데이터의 형식을 설정합니다.

- `connectionType` - 연결 유형입니다. [the section called “연결 파라미터”](#) 섹션을 참조하십시오.
- `options` - 데이터 싱크 연결하려는 추가 정보를 제공하는 JSON 이름-값 페어 문자열입니다. [the section called “연결 파라미터”](#) 섹션을 참조하십시오.
- `transformationContext` - 작업 북마크에서 사용되는 싱크와 관련된 변환 컨텍스트입니다. 기본적으로 비우도록 설정합니다.
- `format` - 대상에 작성되는 데이터의 포맷입니다.
- `formatOptions` - 대상 주소로 데이터를 구성하는 추가 옵션을 제공하는 JSON 이름-값 페어 문자열입니다. [데이터 포맷 옵션](#) 섹션을 참조하십시오.

`DataSink`을 반환합니다.

`def getSource`

```
def getSource( connectionType : String,
              connectionOptions : JsonOptions,
              transformationContext : String = ""
              pushDownPredicate
              ) : DataSource
```

Amazon S3, JDBC 또는 AWS Glue 데이터 카탈로그과 같은 소스에서 데이터를 읽는 데이터를 생성합니다. [DataSource 특성](#) Kafka 및 Kinesis 스트리밍 데이터 원본도 지원합니다.

- `connectionType` - 데이터 원본의 유형입니다. [the section called “연결 파라미터”](#) 섹션을 참조하십시오.
- `connectionOptions` - 데이터 원본을 연결하려는 추가 정보를 제공하는 JSON 이름-값 페어 문자열입니다. 자세한 정보는 [the section called “연결 파라미터”](#)을 참조하세요.

Kinesis 스트리밍 소스에는 `streamARN`, `startingPosition`, `inferSchema` 및 `classification` 연결 옵션이 필요합니다.

Kafka 스트리밍 소스에는 `connectionName`, `topicName`, `startingOffsets`, `inferSchema` 및 `classification` 연결 옵션이 필요합니다.

- `transformationContext` - 작업 북마크에서 사용되는 싱크와 관련된 변환 컨텍스트입니다. 기본적으로 비우도록 설정합니다.
- `pushDownPredicate` - 파티션 열에 대한 조건자입니다.

`DataSource`을 반환합니다.

Amazon Kinesis 스트리밍 소스의 예:

```
val kinesisOptions = jsonOptions()
data_frame_datasource0 = glueContext.getSource("kinesis",
  kinesisOptions).getDataFrame()

private def jsonOptions(): JsonOptions = {
  new JsonOptions(
    s""""{"streamARN": "arn:aws:kinesis:eu-central-1:123456789012:stream/
fromOptionsStream",
      |"startingPosition": "TRIM_HORIZON",
      |"inferSchema": "true",
      |"classification": "json"}"""".stripMargin)
}
```

Kafka 스트리밍 소스의 예:

```
val kafkaOptions = jsonOptions()
val data_frame_datasource0 = glueContext.getSource("kafka",
  kafkaOptions).getDataFrame()
```

```
private def jsonOptions(): JsonOptions = {
  new JsonOptions(
    s""""{"connectionName": "ConfluentKafka",
      |"topicName": "kafka-auth-topic",
      |"startingOffsets": "earliest",
      |"inferSchema": "true",
      |"classification": "json"}"""".stripMargin)
}
```

def 포맷 getSourceWith

```
def getSourceWithFormat( connectionType : String,
                          options : JsonOptions,
                          transformationContext : String = "",
                          format : String = null,
                          formatOptions : JsonOptions = JsonOptions.empty
                        ) : DataSource
```

Amazon S3, JDBC 또는 AWS Glue Data Catalog와 같은 원본에서 데이터를 읽고 원본에 저장된 데이터의 형식도 설정하는 코드를 생성합니다. [DataSource 특성](#)

- `connectionType` - 데이터 원본의 유형입니다. [the section called “연결 파라미터”](#) 섹션을 참조하십시오.
- `options` - 데이터 원본을 연결하려는 추가 정보를 제공하는 JSON 이름-값 페어 문자열입니다. [the section called “연결 파라미터”](#) 섹션을 참조하십시오.
- `transformationContext` - 작업 북마크에서 사용되는 싱크와 관련된 변환 컨텍스트입니다. 기본적으로 비우도록 설정합니다.
- `format` - 소스에 저장된 데이터의 포맷입니다. `connectionType`이 "s3"일 경우에도 `format`을 지정할 수 있습니다. “avro”, “csv”, “grokLog”, “ion”, “json”, “xml”, “parquet”, “orc” 중 하나가 이에 해당합니다.
- `formatOptions` - 소스로 데이터를 파싱하는 추가 옵션을 제공하는 JSON 이름-값 페어 문자열입니다. [데이터 포맷 옵션](#) 섹션을 참조하십시오.

`DataSource`을 반환합니다.

예제

Amazon DynamicFrame S3의 쉼표로 구분된 값 (CSV) 파일인 데이터 소스에서 다음을 생성합니다.

```
val datasource0 = glueContext.getSourceWithFormat(
  connectionType="s3",
  options =JsonOptions(s""""{"paths": [ "s3://csv/nycflights.csv"]}""""),
  transformationContext = "datasource0",
  format = "csv",
  formatOptions=JsonOptions(s""""{"withHeader":"true","separator": ","}""""))
).getDynamicFrame()
```

DynamicFrame JDBC 연결을 사용하여 PostgreSQL인 데이터 소스에서 다음을 생성합니다.

```
val datasource0 = glueContext.getSourceWithFormat(
  connectionType="postgresql",
  options =JsonOptions(s""""{
    "url":"jdbc:postgresql://databasePostgres-1.rds.amazonaws.com:5432/testdb",
    "dbtable": "public.company",
    "redshiftTmpDir":"","
    "user":"username",
    "password":"password123"
  }""""),
  transformationContext = "datasource0").getDynamicFrame()
```

DynamicFrame JDBC 연결을 사용하여 MySQL인 데이터 소스에서 를 생성합니다.

```
val datasource0 = glueContext.getSourceWithFormat(
  connectionType="mysql",
  options =JsonOptions(s""""{
    "url":"jdbc:mysql://databaseMysql-1.rds.amazonaws.com:3306/testdb",
    "dbtable": "athenatest_nycflights13_csv",
    "redshiftTmpDir":"","
    "user":"username",
    "password":"password123"
  }""""),
  transformationContext = "datasource0").getDynamicFrame()
```

def getSparkSession

```
def getSparkSession : SparkSession
```

이 GlueContext와 관련된 SparkSession 객체를 얻습니다. 이 SparkSession 객체를 사용하여 DataFrame 생성된 원본에 사용할 테이블과 UDF를 등록할 수 있습니다. DynamicFrames

를 반환합니다. `SparkSession`

`def startTransaction`

```
def startTransaction(readOnly: Boolean):String
```

새 트랜잭션을 시작합니다. 내부적으로 Lake Formation [startTransaction](#) API를 호출합니다.

- `readOnly` - (부울) 이 트랜잭션이 읽기 전용인지 또는 읽기/쓰기인지를 나타냅니다. 읽기 전용 트랜잭션 ID를 사용하여 수행된 쓰기는 거부됩니다. 읽기 전용 트랜잭션은 커밋할 필요가 없습니다.

트랜잭션 ID를 반환합니다.

`def commitTransaction`

```
def commitTransaction(transactionId: String, waitForCommit: Boolean): Boolean
```

지정된 트랜잭션을 커밋하려는 시도입니다. 트랜잭션이 커밋을 완료하기 전에 `commitTransaction`이 반환될 수 있습니다. 내부적으로 Lake Formation [commitTransaction](#) API를 호출합니다.

- `transactionId` - (문자열) 커밋할 트랜잭션입니다.
- `waitForCommit` - (부울) `commitTransaction`이 즉시 반환되는지 여부를 결정합니다. 기본값은 `true`입니다. `false`인 경우 `commitTransaction`은 폴링한 후 트랜잭션이 커밋될 때까지 기다립니다. 대기 시간은 최대 재시도 횟수가 6회인 지수 백오프를 사용하여 1분으로 제한됩니다.

커밋이 수행되었는지 여부를 나타내는 부울을 반환합니다.

`def cancelTransaction`

```
def cancelTransaction(transactionId: String): Unit
```

지정된 트랜잭션을 취소하려는 시도입니다. 내부적으로 레이크 포메이션 [CancelTransaction](#) API를 호출합니다.

- `transactionId` - (문자열) 취소할 트랜잭션입니다.

트랜잭션이 이전에 커밋된 경우 `TransactionCommittedException` 예외를 반환합니다.

def this

```
def this( sc : SparkContext,  
         minPartitions : Int,  
         targetPartitions : Int )
```

지정된 SparkContext, 최소 파티션 및 대상 파티션을 사용하여 GlueContext 객체를 생성합니다.

- sc - SparkContext
- minPartitions - 파티션의 최소 수입니다.
- targetPartitions - 파티션의 목표 수입니다.

GlueContext을 반환합니다.

def this

```
def this( sc : SparkContext )
```

제공된 SparkContext로 GlueContext 객체를 생성합니다. 최소 파티션을 10으로 대상 파티션을 20으로 설정합니다.

- sc - SparkContext

GlueContext을 반환합니다.

def this

```
def this( sparkContext : JavaSparkContext )
```

제공된 JavaSparkContext로 GlueContext 객체를 생성합니다. 최소 파티션을 10으로 대상 파티션을 20으로 설정합니다.

- sparkContext - JavaSparkContext

GlueContext을 반환합니다.

MappingSpec

패키지: com.amazonaws.services.glue

MappingSpec 케이스 클래스

```
case class MappingSpec( sourcePath: SchemaPath,
                       sourceType: DataType,
                       targetPath: SchemaPath,
                       targetType: DataType
                       ) extends Product4[String, String, String, String] {
  override def _1: String = sourcePath.toString
  override def _2: String = ExtendedTypeName.fromDataType(sourceType)
  override def _3: String = targetPath.toString
  override def _4: String = ExtendedTypeName.fromDataType(targetType)
}
```

- sourcePath - 소스 필드의 SchemaPath입니다.
- sourceType - 소스 필드의 DataType입니다.
- targetPath - 대상 필드의 SchemaPath입니다.
- targetType - 대상 필드의 DataType입니다.

MappingSpec은 소스 경로 및 소스 데이터 유형에서 대상 경로 및 대상 데이터 유형까지의 매핑을 지정합니다. 소스 프레임의 소스 경로 값은 대상 경로의 대상 프레임에서 볼 수 있습니다. 소스 데이터 유형은 대상 데이터 유형으로 캐스팅됩니다.

applyMapping 인터페이스에서 Product4을 처리할 수 있도록 Product4에서 확장됩니다.

MappingSpec 객체

```
object MappingSpec
```

MappingSpec 객체에는 다음 멤버가 있습니다.

Val orderingByTarget

```
val orderingByTarget: Ordering[MappingSpec]
```

정의 적용

```
def apply( sourcePath : String,
          sourceType : DataType,
```

```
targetPath : String,
targetType : DataType
) : MappingSpec
```

MappingSpec을 만듭니다.

- sourcePath - 소스 경로의 문자열 표현입니다.
- sourceType - 소스 DataType입니다.
- targetPath - 대상 경로의 문자열 표현입니다.
- targetType - 대상 DataType입니다.

MappingSpec을 반환합니다.

정의 적용

```
def apply( sourcePath : String,
sourceTypeString : String,
targetPath : String,
targetTypeString : String
) : MappingSpec
```

MappingSpec을 만듭니다.

- sourcePath - 소스 경로의 문자열 표현입니다.
- sourceType - 원본 데이터 유형의 문자열 표현입니다.
- targetPath - 대상 경로의 문자열 표현입니다.
- targetType - 대상 데이터 유형의 문자열 표현입니다.

MappingSpec을 반환합니다.

정의 적용

```
def apply( product : Product4[String, String, String, String] ) : MappingSpec
```

MappingSpec을 만듭니다.

- product - 소스 경로, 원본 데이터 유형, 대상 경로 및 대상 데이터 유형의 Product4입니다.

MappingSpec을 반환합니다.

AWS Glue Scala ResolveSpec API

주제

- [ResolveSpec 객체](#)
- [ResolveSpec 케이스 클래스](#)

패키지: com.amazonaws.services.glue

ResolveSpec 객체

ResolveSpec

```
object ResolveSpec
```

def

```
def apply( path : String,
           action : String
         ) : ResolveSpec
```

ResolveSpec을 만듭니다.

- path - 확인해야 할 선택 필드를 나타내는 문자열입니다.
- action - 확인 작업입니다. Project, KeepAsStruct 또는 Cast 중 하나가 작업이 될 수 있습니다.

ResolveSpec을 반환합니다.

def

```
def apply( product : Product2[String, String] ) : ResolveSpec
```

ResolveSpec을 만듭니다.

- product - 소스 경로, 확인 작업의 Product2입니다.

ResolveSpec을 반환합니다.

ResolveSpec 케이스 클래스

```
case class ResolveSpec extends Product2[String, String] (  
    path : SchemaPath,  
    action : String )
```

ResolveSpec을 만듭니다.

- path - 확인해야 할 선택 필드의 SchemaPath입니다.
- action - 확인 작업입니다. Project, KeepAsStruct 또는 Cast 중 하나가 작업이 될 수 있습니다.

ResolveSpec def 메서드

```
def _1 : String
```

```
def _2 : String
```

AWS Glue Scala ArrayNode API

패키지: com.amazonaws.services.glue.types

ArrayNode 케이스 클래스

ArrayNode

```
case class ArrayNode extends DynamicNode (  
    value : ArrayBuffer[DynamicNode] )
```

ArrayNode 정의 메서드

```
def add( node : DynamicNode )
```

```
def clone
```

```
def equals( other : Any )
```

```
def get( index : Int ) : Option[DynamicNode]
```

```
def getValue
```

```
def hashCode : Int
```

```
def isEmpty : Boolean
```

```
def nodeType
```

```
def remove( index : Int )
```

```
def this
```

```
def toIterator : Iterator[DynamicNode]
```

```
def toJson : String
```

```
def update( index : Int,  
            node : DynamicNode )
```

AWS Glue Scala BinaryNode API

패키지: `com.amazonaws.services.glue.types`

BinaryNode 케이스 클래스

BinaryNode

```
case class BinaryNode extends ScalarNode(value, TypeCode.BINARY) (  
    value : Array[Byte] )
```

BinaryNode val 필드

- `ordering`

BinaryNode 정의 메서드

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

AWS Glue Scala BooleanNode API

패키지: `com.amazonaws.services.glue.types`

BooleanNode 케이스 클래스

BooleanNode

```
case class BooleanNode extends ScalarNode(value, TypeCode.BOOLEAN) (  
    value : Boolean )
```

BooleanNode val 필드

- `ordering`

BooleanNode 정의 메서드

```
def equals( other : Any )
```

AWS Glue Scala ByteNode API

패키지: `com.amazonaws.services.glue.types`

ByteNode 케이스 클래스

ByteNode

```
case class ByteNode extends ScalarNode(value, TypeCode.BYTE) (  
    value : Byte )
```

ByteNode val 필드

- ordering

ByteNode 정의 메서드

```
def equals( other : Any )
```

AWS Glue Scala DateNode API

패키지: com.amazonaws.services.glue.types

DateNode 케이스 클래스

DateNode

```
case class DateNode extends ScalarNode(value, TypeCode.DATE) (  
    value : Date )
```

DateNode val 필드

- ordering

ByteNode def 메서드

```
def equals( other : Any )
```

```
def this( value : Int )
```

AWS Glue Scala DecimalNode API

패키지: com.amazonaws.services.glue.types

DecimalNode 케이스 클래스

DecimalNode

```
case class DecimalNode extends ScalarNode(value, TypeCode.DECIMAL) (  
    value : Double )
```

```
value : BigDecimal )
```

DecimalNode val 필드

- ordering

DecimalNode def 메서드

```
def equals( other : Any )
```

```
def this( value : Decimal )
```

AWS Glue Scala DoubleNode API

패키지: `com.amazonaws.services.glue.types`

DoubleNode 케이스 클래스

DoubleNode

```
case class DoubleNode extends ScalarNode(value, TypeCode.DOUBLE) (  
    value : Double )
```

DoubleNode val 필드

- ordering

DoubleNode def 메서드

```
def equals( other : Any )
```

AWS Glue Scala DynamicNode API

주제

- [DynamicNode 클래스](#)
- [DynamicNode 객체](#)

패키지: `com.amazonaws.services.glue.types`

DynamicNode 클래스

DynamicNode

```
class DynamicNode extends Serializable with Cloneable
```

DynamicNode def 메서드

```
def getValue : Any
```

플레인 값을 가져오며 현재 레코드에 바인딩됩니다.

```
def nodeType : TypeCode
```

```
def toJson : String
```

디버깅 방법

```
def toRow( schema : Schema,  
           options : Map[String, ResolveOption]  
           ) : Row
```

```
def typeName : String
```

DynamicNode 객체

DynamicNode

```
object DynamicNode
```

DynamicNode def 메서드

```
def quote( field : String,  
           useQuotes : Boolean  
           ) : String
```

```
def quote( node : DynamicNode,
```

```

        useQuotes : Boolean
    ) : String

```

EvaluateDataQuality 클래스

AWS Glue 데이터 품질은 AWS Glue의 미리 보기 릴리스이므로 변경될 수 있습니다.

패키지: com.amazonaws.services.glue.dq

```
object EvaluateDataQuality
```

def apply

```

def apply(frame: DynamicFrame,
          ruleset: String,
          publishingOptions: JsonOptions = JsonOptions.empty): DynamicFrame

```

DynamicFrame을 기준으로 데이터 품질 규칙 세트를 평가하고 새 DynamicFrame을 평가 결과와 함께 반환합니다. AWS Glue 데이터 품질에 대한 자세한 내용은 [AWS Glue 데이터 품질](#) 섹션을 참조하세요.

- frame - 데이터 품질을 평가하려는 DynamicFrame입니다.
- ruleset - 문자열 형식의 DQDL(데이터 품질 정의 언어) 규칙 세트입니다. DQDL에 대한 자세한 내용은 [데이터 품질 정의 언어\(DQDL\) 참조](#) 안내서를 참조하세요.
- publishingOptions - 평가 결과 및 지표를 게시하기 위해 다음 옵션을 지정하는 사전입니다.
 - dataQualityEvaluationContext - AWS Glue가 Amazon CloudWatch 지표와 데이터 품질 결과를 게시할 네임스페이스를 지정하는 문자열입니다. 집계된 지표는 CloudWatch에 표시되고 전체 결과는 AWS Glue Studio 인터페이스에 표시됩니다.
 - 필수 항목 여부: 아니요
 - 기본 값: default_context
 - enableDataQualityCloudWatchMetrics - 데이터 품질 평가 결과를 CloudWatch에 게시할지 여부를 지정합니다. dataQualityEvaluationContext 옵션을 사용하여 지표의 네임스페이스를 지정합니다.
 - 필수 항목 여부: 아니요
 - 기본 값: False

- `enableDataQualityResultsPublishing` - AWS Glue Studio 인터페이스의 Data Quality(데이터 품질) 탭에 데이터 품질 결과를 표시할지 여부를 지정합니다.
 - 필수 항목 여부: 아니요
 - 기본값: `True`
- `resultsS3Prefix` - AWS Glue가 데이터 품질 평가 결과를 기록할 수 있는 Amazon S3 위치를 지정합니다.
 - 필수 항목 여부: 아니요
 - 기본값: ""(빈 문자열)

예

다음 예제 코드는 `SelectFields` 변환을 수행하기 전에 `DynamicFrame`의 데이터 품질을 평가하는 방법을 보여줍니다. 스크립트는 변환을 시도하기 전에 모든 데이터 품질 규칙이 통과하는지 확인합니다.

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.dq.EvaluateDataQuality

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Create DynamicFrame with data
    val Legislators_Area = glueContext.getCatalogSource(database="legislators",
    tableName="areas_json", transformationContext="S3bucket_node1").getDynamicFrame()

    // Define data quality ruleset
    val DQ_Ruleset = ""
    Rules = [ColumnExists "id"]
  }
}
```

```

    ""

    // Evaluate data quality
    val DQ_Results = EvaluateDataQuality.apply(frame=Legislators_Area,
    ruleset=DQ_Ruleset, publishingOptions=JsonOptions("""{"dataQualityEvaluationContext":
    "Legislators_Area", "enableDataQualityMetrics": "true",
    "enableDataQualityResultsPublishing": "true"}""))
    assert(DQ_Results.filter(_.getField("Outcome").contains("Failed")).count == 0,
    "Failing DQ rules for Legislators_Area caused the job to fail.")

    // Script generated for node Select Fields
    val SelectFields_Results = Legislators_Area.selectFields(paths=Seq("id", "name"),
    transformationContext="Legislators_Area")

    Job.commit()
  }
}

```

AWS Glue Scala FloatNode API

패키지: `com.amazonaws.services.glue.types`

FloatNode 케이스 클래스

FloatNode

```

case class FloatNode extends ScalarNode(value, TypeCode.FLOAT) (
    value : Float )

```

FloatNode val 필드

- `ordering`

FloatNode def 메서드

```

def equals( other : Any )

```

FillMissingValues 클래스

패키지: `com.amazonaws.services.glue.ml`

```

object FillMissingValues

```

def apply

```
def apply(frame: DynamicFrame,
          missingValuesColumn: String,
          outputColumn: String = "",
          transformationContext: String = "",
          callSite: CallSite = CallSite("Not provided", ""),
          stageThreshold: Long = 0,
          totalThreshold: Long = 0): DynamicFrame
```

지정된 열에 동적 프레임의 누락된 값을 채우고 새 열에 추정 값과 함께 새 프레임을 반환합니다. 누락된 값이 없는 행의 경우 지정된 열의 값이 새 열에 복제됩니다.

- `frame` - 누락된 값을 채울 `DynamicFrame`입니다. 필수 사항입니다.
- `missingValuesColumn` - 누락된 값(`null` 값 및 빈 문자열)이 포함된 열입니다. 필수 사항입니다.
- `outputColumn` - 값이 누락된 모든 행에 대한 예상 값을 포함할 새 열의 이름입니다. 선택 사항이며 기본값은 `"_filled"`가 접미사로 사용된 `missingValuesColumn`의 값입니다.
- `transformationContext` - 고유 문자열을 통해 상태 정보를 확인합니다(선택 사항).
- `callSite` - 오류 보고에 필요한 컨텍스트 정보를 제공하는 데 사용합니다(선택 사항).
- `stageThreshold` - 오류가 발생하기 전까지 변환에 따라 생길 수 있는 최대 오류 수입니다(선택 사항, 기본값은 0).
- `totalThreshold` - 오류가 진행되기 전까지 생길 수 있는 최대 전체 오류 수입니다(선택 사항, 기본값은 0).

누락된 값이 있는 행에 대한 추정 값과 다른 행에 대한 현재 값이 포함된 하나의 추가 열이 있는 새 동적 프레임을 반환합니다.

FindMatches 클래스

패키지: `com.amazonaws.services.glue.ml`

```
object FindMatches
```

def apply

```
def apply(frame: DynamicFrame,
          transformId: String,
          transformationContext: String = "",
```

```

    callSite: CallSite = CallSite("Not provided", ""),
    stageThreshold: Long = 0,
    totalThreshold: Long = 0,
    enforcedMatches: DynamicFrame = null): DynamicFrame,
computeMatchConfidenceScores: Boolean

```

입력 프레임에서 일치 항목을 찾고 일치 그룹당 고유 ID를 포함하는 새 열이 있는 새 프레임을 반환합니다.

- `frame` - 일치하는 항목을 찾을 `DynamicFrame`입니다. 필수 사항입니다.
- `transformId` - 입력 프레임에 적용할 `FindMatches` 변환과 연결된 고유 ID입니다. 필수 사항입니다.
- `transformationContext` - 이 `DynamicFrame`의 식별자입니다. `transformationContext`는 실행 간 지속되는 작업 북마크 상태에 대한 키로 사용됩니다. 선택 사항입니다.
- `callSite` - 오류 보고에 필요한 컨텍스트 정보를 제공하는 데 사용됩니다. Python에서 호출 할 때 자동으로 이 값이 설정됩니다. 선택 사항입니다.
- `stageThreshold` - 예외가 발생하기 전에 이 `DynamicFrame` 계산에 허용되는 오류 레코드의 최대 개수입니다(이전 `DynamicFrame`에 존재하는 레코드 제외). 선택 사항입니다. 기본값은 0입니다.
- `totalThreshold` - 예외가 발생하기 전 전체 오류 레코드의 최대 개수입니다(이전 프레임의 레코드 포함). 선택 사항입니다. 기본값은 0입니다.
- `enforcedMatches` - 강제 일치 항목에 대한 프레임입니다. 선택 사항입니다. 기본값은 `null`입니다.
- `computeMatchConfidenceScores` - 각 일치 레코드 그룹의 신뢰도 점수를 컴퓨팅할지 여부를 나타내는 부울 값입니다. 선택 사항입니다. 기본값은 `false`입니다.

일치하는 레코드의 각 그룹에 고유 식별자가 할당된 새 동적 프레임을 반환합니다.

`FindIncrementalMatches` 클래스

패키지: `com.amazonaws.services.glue.ml`

```
object FindIncrementalMatches
```

`def apply`

```

apply(existingFrame: DynamicFrame,
       incrementalFrame: DynamicFrame,

```

```

transformId: String,
transformationContext: String = "",
callSite: CallSite = CallSite("Not provided", ""),
stageThreshold: Long = 0,
totalThreshold: Long = 0,
enforcedMatches: DynamicFrame = null): DynamicFrame,
computeMatchConfidenceScores: Boolean

```

기존 및 증분 프레임에서 일치 항목을 찾고 일치 그룹당 고유 ID가 포함된 열이 있는 새 프레임을 반환합니다.

- `existingframe` - 각 그룹에 대해 일치하는 ID가 할당된 기존 프레임입니다. 필수 사항입니다.
- `incrementalframe` - 기존 프레임과 일치하는 항목을 찾는 데 사용되는 증분 프레임입니다. 필수 사항입니다.
- `transformId` - 입력 프레임에 적용할 `FindIncrementalMatches` 변환과 연결된 고유 ID입니다. 필수 사항입니다.
- `transformationContext` - 이 `DynamicFrame`의 식별자입니다. `transformationContext`는 실행 간 지속되는 작업 북마크 상태에 대한 키로 사용됩니다. 선택 사항입니다.
- `callSite` - 오류 보고에 필요한 컨텍스트 정보를 제공하는 데 사용됩니다. Python에서 호출 할 때 자동으로 이 값이 설정됩니다. 선택 사항입니다.
- `stageThreshold` - 예외가 발생하기 전에 이 `DynamicFrame` 계산에 허용되는 오류 레코드의 최대 개수입니다(이전 `DynamicFrame`에 존재하는 레코드 제외). 선택 사항입니다. 기본값은 0입니다.
- `totalThreshold` - 예외가 발생하기 전 전체 오류 레코드의 최대 개수입니다(이전 프레임의 레코드 포함). 선택 사항입니다. 기본값은 0입니다.
- `enforcedMatches` - 강제 일치 항목에 대한 프레임입니다. 선택 사항입니다. 기본값은 `null`입니다.
- `computeMatchConfidenceScores` - 각 일치 레코드 그룹의 신뢰도 점수를 컴퓨팅할지 여부를 나타내는 부울 값입니다. 선택 사항입니다. 기본값은 `false`입니다.

일치하는 레코드의 각 그룹에 고유 식별자가 할당된 새 동적 프레임을 반환합니다.

AWS Glue Scala IntegerNode API

패키지: `com.amazonaws.services.glue.types`

IntegerNode 케이스 클래스

IntegerNode

```
case class IntegerNode extends ScalarNode(value, TypeCode.INT) (  
    value : Int )
```

IntegerNode val 필드

- ordering

IntegerNode def 메서드

```
def equals( other : Any )
```

AWS Glue Scala LongNode API

패키지: com.amazonaws.services.glue.types

LongNode 케이스 클래스

LongNode

```
case class LongNode extends ScalarNode(value, TypeCode.LONG) (  
    value : Long )
```

LongNode val 필드

- ordering

LongNode def 메서드

```
def equals( other : Any )
```

AWS Glue Scala MapLikeNode API

패키지: com.amazonaws.services.glue.types

MapLikeNode 클래스

MapLikeNode

```
class MapLikeNode extends DynamicNode (
```

```
value : mutable.Map[String, DynamicNode] )
```

MapLikeNode def 메서드

```
def clear : Unit
```

```
def get( name : String ) : Option[DynamicNode]
```

```
def getValue
```

```
def has( name : String ) : Boolean
```

```
def isEmpty : Boolean
```

```
def put( name : String,  
        node : DynamicNode  
        ) : Option[DynamicNode]
```

```
def remove( name : String ) : Option[DynamicNode]
```

```
def toIterator : Iterator[(String, DynamicNode)]
```

```
def toJson : String
```

```
def toJson( useQuotes : Boolean ) : String
```

예제: 이 JSON을 고려할 때

```
{"foo": "bar"}
```

`useQuotes == true`이면 `toJson`이 `{"foo": "bar"}`를 생성합니다. `useQuotes == false`이면 `toJson`이 `{foo: bar} @return`을 생성합니다.

AWS Glue Scala MapNode API

패키지: `com.amazonaws.services.glue.types`

MapNode 케이스 클래스

MapNode

```
case class MapNode extends MapLikeNode(value) (  
    value : mutable.Map[String, DynamicNode] )
```

MapNode def 메서드

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

```
def nodeType
```

```
def this
```

AWS Glue Scala NullNode API

주제

- [NullNode 클래스](#)
- [NullNode 케이스 객체](#)

패키지: `com.amazonaws.services.glue.types`

NullNode 클래스

NullNode

```
class NullNode
```

NullNode 케이스 객체

NullNode


```
case object NullNode extends NullNode
```

AWS Glue Scala ObjectNode API

주제

- [ObjectNode 객체](#)
- [ObjectNode 케이스 클래스](#)

패키지: `com.amazonaws.services.glue.types`

ObjectNode 객체

ObjectNode

```
object ObjectNode
```

ObjectNode def 메서드

```
def apply( frameKeys : Set[String],  
          v1 : mutable.Map[String, DynamicNode],  
          v2 : mutable.Map[String, DynamicNode],  
          resolveWith : String  
        ) : ObjectNode
```

ObjectNode 케이스 클래스

ObjectNode

```
case class ObjectNode extends MapLikeNode(value) (  
  val value : mutable.Map[String, DynamicNode] )
```

ObjectNode def 메서드

```
def clone
```

```
def equals( other : Any )
```

```
def hashCode : Int
```

```
def nodeType
```

```
def this
```

AWS Glue Scala ScalarNode API

주제

- [ScalarNode 클래스](#)
- [ScalarNode 객체](#)

패키지: `com.amazonaws.services.glue.types`

ScalarNode 클래스

ScalarNode

```
class ScalarNode extends DynamicNode (  
    value : Any,  
    scalarType : TypeCode )
```

ScalarNode def 메서드

```
def compare( other : Any,  
            operator : String  
            ) : Boolean
```

```
def getValue
```

```
def hashCode : Int
```

```
def nodeType
```

```
def toJson
```

ScalarNode 객체

ScalarNode

```
object ScalarNode
```

ScalarNode def 메서드

```
def apply( v : Any ) : DynamicNode
```

```
def compare( tv : Ordered[T],  
            other : T,  
            operator : String  
            ) : Boolean
```

```
def compareAny( v : Any,  
               y : Any,  
               o : String )
```

```
def withEscapedSpecialCharacters( jsonToEscape : String ) : String
```

AWS Glue Scala ShortNode API

패키지: `com.amazonaws.services.glue.types`

ShortNode 케이스 클래스

ShortNode

```
case class ShortNode extends ScalarNode(value, TypeCode.SHORT) (  
    value : Short )
```

ShortNode val 필드

- `ordering`

ShortNode def 메서드

```
def equals( other : Any )
```

AWS Glue Scala StringNode API

패키지: `com.amazonaws.services.glue.types`

StringNode 케이스 클래스

StringNode

```
case class StringNode extends ScalarNode(value, TypeCode.STRING) (  
    value : String )
```

StringNode val 필드

- ordering

StringNode def 메서드

```
def equals( other : Any )
```

```
def this( value : UTF8String )
```

AWS Glue Scala TimestampNode API

패키지: `com.amazonaws.services.glue.types`

TimestampNode 케이스 클래스

TimestampNode

```
case class TimestampNode extends ScalarNode(value, TypeCode.TIMESTAMP) (  
    value : Timestamp )
```

TimestampNode val 필드

- ordering

TimestampNode def 메서드

```
def equals( other : Any )
```

```
def this( value : Long )
```

AWS Glue Scala GlueArgParser API

패키지: `com.amazonaws.services.glue.util`

GlueArgParser 객체

GlueArgParser

```
object GlueArgParser
```

이것은 `AWSGlueDataplanePython` 패키지에 있는 `utils.getResolvedOptions`의 Python 버전과 엄격하게 일치합니다.

GlueArgParser def 메서드

```
def getResolvedOptions( args : Array[String],
                        options : Array[String]
                        ) : Map[String, String]
```

```
def initParser( userOptionsSet : mutable.Set[String] ) : ArgumentParser
```

Example 작업으로 전달된 인수 검색

작업 인수를 검색하려면 `getResolvedOptions` 메서드를 사용할 수 있습니다. `aws_region` 작업 인수를 검색하는 다음 예제를 검토합니다.

```
val args = GlueArgParser.getResolvedOptions(sysArgs,
      Seq("JOB_NAME","aws_region").toArray)
Job.init(args("JOB_NAME"), glueContext, args.asJava)
val region = args("aws_region")
println(region)
```

AWS Glue Scala 작업 API

패키지: `com.amazonaws.services.glue.util`

작업 객체

작업

```
object Job
```

작업 def 메서드

```
def commit
```

```
def init( jobName : String,
          glueContext : GlueContext,
          args : java.util.Map[String, String] = Map[String, String]()
        ) : this.type
```

```
def init( jobName : String,
          glueContext : GlueContext,
          endpoint : String,
          args : java.util.Map[String, String]
        ) : this.type
```

```
def isInitialized
```

```
def reset
```

```
def runId
```

AWS Glue for Spark ETL 스크립트 프로그래밍을 위한 기능 및 최적화

다음 섹션은 어떤 언어든지 AWS Glue for Spark 추출, 전환 및 적재(ETL) 프로그래밍에 일반적으로 적용되는 기술 및 값을 설명합니다.

주제

- [AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션](#)
- [AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션](#)
- [Spark SQL 작업에 대해 AWS Glue Data Catalog 지원](#)
- [작업 북마크 사용](#)
- [AWS Glue Studio 외부에서 민감한 데이터 탐지 사용](#)

- [AWS Glue Visual Job API](#)

AWS Glue for Spark에서 ETL에 대한 연결 유형 및 옵션

AWS Glue for Spark에서 다양한 PySpark와 Scala 메서드 및 변환은 `connectionType` 파라미터를 사용하여 연결 유형을 지정합니다. `connectionOptions` 또는 `options` 파라미터를 사용하여 연결 옵션을 지정합니다.

`connectionType` 파라미터는 다음 표에 표시된 값을 사용할 수 있습니다. 각 유형에 대한 연관된 `connectionOptions`(또는 `options`) 파라미터 값은 다음 섹션에 설명되어 있습니다. 별도의 언급이 없으면 이 파라미터는 연결이 소스 또는 싱크로 사용될 때 적용됩니다.

연결 옵션을 설정하고 사용하는 방법을 보여주는 샘플 코드는 각 연결 유형별 홈페이지를 참조하세요.

<code>connectionType</code>	연결 대상
dynamodb	Amazon DynamoDB 데이터베이스
kinesis	Amazon Kinesis Data Streams
s3	Amazon S3
documentdb	Amazon DocumentDB(MongoDB와 호환) 데이터베이스
OpenSearch	Amazon OpenSearch Service .
redshift	Amazon Redshift 데이터베이스
kafka	Kafka 또는 Amazon Managed Streaming for Apache Kafka
azurecosmos	NoSQL용 Azure Cosmos.
azuresql	Azure SQL.
bigquery	Google BigQuery.
mongodb	MongoDB Atlas를 포함한 MongoDB 데이터베이스.
sqlserver	Microsoft SQL Server 데이터베이스(JDBC 연결 참조)
mysql	MySQL 데이터베이스(JDBC 연결 참조)

connectionType	연결 대상
oracle	Oracle Database(JDBC 연결 참조)
postgresql	PostgreSQL 데이터베이스(JDBC 연결 참조)
saphana	SAP HANA.
snowflake	Snowflake 데이터 레이크
Teradata	Teradata Vantage.
vertica	Vertica.
custom.*	Spark, Athena 또는 JDBC 데이터 원본(사용자 정의 및 AWS Marketplace 연결 유형 값 참조)
marketplace.*	Spark, Athena 또는 JDBC 데이터 원본(사용자 정의 및 AWS Marketplace 연결 유형 값 참조)

DynamoDB 연결

AWS Glue for Spark를 사용하여 AWS Glue에서 DynamoDB의 테이블을 읽고 쓸 수 있습니다. AWS Glue 작업에 연결된 IAM 권한을 사용하여 DynamoDB에 연결합니다. AWS Glue는 다른 AWS 계정의 DynamoDB 테이블에 데이터 쓰기를 지원합니다. 자세한 내용은 [the section called “DynamoDB 테이블에 대한 교차 계정 교차 리전 액세스”](#) 단원을 참조하십시오.

AWS Glue DynamoDB ETL 커넥터 외에도 DynamoDB ExportTableToPointInTime 요청을 호출하여 사용자가 제공한 Amazon S3 위치에 [DynamoDB JSON](#) 형식으로 저장하는 DynamoDB 내보내기 커넥터를 사용하여 DynamoDB에서 읽을 수 있습니다. 그런 다음 AWS Glue는 Amazon S3 내보내기 위치에서 데이터를 읽어와서 DynamicFrame 객체를 생성합니다.

DynamoDB 기록기는 AWS Glue 버전 1.0 이상에서 지원됩니다. AWS Glue DynamoDB 내보내기 커넥터는 AWS Glue 버전 2.0 이상 버전에서 사용할 수 있습니다.

DynamoDB에 대한 자세한 내용은 [Amazon DynamoDB](#) 설명서를 참조하세요.

Note

DynamoDB ETL 리더는 필터 또는 푸시다운 조건자를 지원하지 않습니다.

DynamoDB 연결 구성

AWS Glue에서 DynamoDB에 연결하려면 AWS Glue 작업과 관련된 IAM 역할에 DynamoDB와 상호 작용할 수 있는 권한을 부여합니다. DynamoDB에서 읽거나 쓰는 데 필요한 권한에 대한 자세한 내용은 IAM 설명서에서 [DynamoDB에 사용되는 작업, 리소스 및 조건 키](#)를 참조하세요.

다음과 같은 상황에서는 추가 구성이 필요할 수도 있습니다.

- DynamoDB 내보내기 커넥터를 사용하는 경우에는 작업에서 DynamoDB 테이블 내보내기를 요청할 수 있도록 IAM을 구성해야 합니다. 또한 내보내기를 위한 Amazon S3 버킷을 식별하고 DynamoDB가 이 버킷에 쓸 수 있고 AWS Glue 작업이 이 버킷에서 읽을 수 있도록 IAM에서 적절한 권한을 제공해야 합니다. 자세한 내용은 [DynamoDB에서 테이블 내보내기 요청](#)을 참조하세요.
- AWS Glue 작업에 특정 Amazon VPC 연결 요구 사항이 있는 경우 NETWORK AWS Glue 연결 유형을 사용하여 네트워크 옵션을 제공하세요. DynamoDB에 대한 액세스는 IAM에 의해 권한이 부여되므로 AWS Glue DynamoDB 연결 유형이 필요하지 않습니다.

DynamoDB에서 읽고 쓰기

다음 코드 예제에서는 ETL 커넥터를 통해 DynamoDB 테이블에서 읽고, DynamoDB 테이블에 쓰는 방법을 보여줍니다. 한 테이블에서 읽고 다른 테이블에 쓰는 것을 보여줍니다.

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={"dynamodb.input.tableName": test_source,
                        "dynamodb.throughput.read.percent": "1.0",
                        "dynamodb.splits": "100"
    }
)
```

```
print(dyf.getNumPartitions())

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={"dynamodb.output.tableName": test_sink,
                        "dynamodb.throughput.write.percent": "1.0"}
)

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

  def main(sysArgs: Array[String]): Unit = {
    val glueContext = new GlueContext(SparkContext.getOrCreate())
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType = "dynamodb",
      options = JsonOptions(Map(
        "dynamodb.input.tableName" -> test_source,
        "dynamodb.throughput.read.percent" -> "1.0",
        "dynamodb.splits" -> "100"
      ))
    ).getDynamicFrame()

    print(dynamicFrame.getNumPartitions())

    val dynamoDbSink: DynamoDbDataSink = glueContext.getSinkWithFormat(
      connectionType = "dynamodb",
```

```

    options = JsonOptions(Map(
      "dynamodb.output.tableName" -> test_sink,
      "dynamodb.throughput.write.percent" -> "1.0"
    ))
  ).asInstanceOf[DynamoDbDataSink]

  dynamoDbSink.writeDynamicFrame(dynamicFrame)

  Job.commit()
}
}

```

DynamoDB 내보내기 커넥터 사용

내보내기 커넥터는 DynamoDB 테이블 크기가 80GB보다 큰 경우 ETL 커넥터보다 성능이 우수합니다. 또한 내보내기 요청이 AWS Glue 작업의 Spark 프로세스 외부에서 수행된다는 점을 고려할 때, [AWS Glue 작업의 Auto Scaling](#)을 사용 설정하여 내보내기 요청이 진행되는 동안 DPU 사용량을 줄일 수 있습니다. 내보내기 커넥터를 사용하면 Spark 실행기 병렬 처리 또는 DynamoDB 처리량 읽기 비율에 대한 분할 수를 구성할 필요가 없습니다.

Note

DynamoDB에는 ExportTableToPointInTime 요청을 호출하기 위한 특정 요구 사항이 있습니다. 자세한 내용은 [DynamoDB에서 테이블 내보내기 요청](#)을 참조하세요. 예를 들어, 이 커넥터를 사용하려면 테이블에서 PITR(특정 시점으로 복구)을 사용 설정해야 합니다. 또한 DynamoDB 커넥터는 Amazon S3으로의 DynamoDB 내보내기 작업에 대해 AWS KMS 암호화를 지원합니다. AWS Glue 작업 구성에서 보안 구성을 제공하면 DynamoDB 내보내기에 대한 AWS KMS 암호화가 사용 설정됩니다. KMS 키는 Amazon S3 버킷과 동일한 리전에 있어야 합니다.

DynamoDB 내보내기 및 Amazon S3 스토리지 비용에 대한 추가 요금이 부과됩니다. Amazon S3에서 내보낸 데이터는 작업 실행이 완료된 후에도 유지되므로 추가 DynamoDB 내보내기 없이 다시 사용할 수 있습니다. 이 커넥터 사용 시 요구 사항은 테이블에 대해 PITR(특정 시점으로 복구)을 사용 설정하는 것입니다.

DynamoDB ETL 커넥터 또는 내보내기 커넥터는 DynamoDB 소스에 적용할 필터 또는 푸시다운 조건자를 지원하지 않습니다.

다음 코드 예제에서는 내보내기 커넥터를 통해 파티션 수를 읽고, 파티션 수를 인쇄하는 방법을 보여줍니다.

Python

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame.from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": test_source,
        "dynamodb.s3.bucket": bucket_name,
        "dynamodb.s3.prefix": bucket_prefix,
        "dynamodb.s3.bucketOwner": account_id_of_bucket,
    }
)
print(dyf.getNumPartitions())

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
```

```

def main(sysArgs: Array[String]): Unit = {
  val glueContext = new GlueContext(SparkContext.getOrCreate())
  val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
  Job.init(args("JOB_NAME"), glueContext, args.asJava)

  val dynamicFrame = glueContext.getSourceWithFormat(
    connectionType = "dynamodb",
    options = JsonOptions(Map(
      "dynamodb.export" -> "ddb",
      "dynamodb.tableArn" -> test_source,
      "dynamodb.s3.bucket" -> bucket_name,
      "dynamodb.s3.prefix" -> bucket_prefix,
      "dynamodb.s3.bucketOwner" -> account_id_of_bucket,
    ))
  ).getDynamicFrame()

  print(dynamicFrame.getNumPartitions())

  Job.commit()
}
}

```

다음 예제에서는 내보내기 커넥터를 통해 dynamodb 분류가 있는 AWS Glue 데이터 카탈로그 테이블에서 읽기 작업을 수행하고 파티션 수를 인쇄하는 방법을 보여줍니다.

Python

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dynamicFrame = glue_context.create_dynamic_frame.from_catalog(
    database=catalog_database,
    table_name=catalog_table_name,

```

```
        additional_options={
            "dynamodb.export": "ddb",
            "dynamodb.s3.bucket": s3_bucket,
            "dynamodb.s3.prefix": s3_bucket_prefix
        }
    )
print(dynamicFrame.getNumPartitions())

job.commit()
```

Scala

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamoDbDataSink
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {

    def main(sysArgs: Array[String]): Unit = {
        val glueContext = new GlueContext(SparkContext.getOrCreate())
        val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
        Job.init(args("JOB_NAME"), glueContext, args.asJava)

        val dynamicFrame = glueContext.getCatalogSource(
            database = catalog_database,
            tableName = catalog_table_name,
            additionalOptions = JsonOptions(Map(
                "dynamodb.export" -> "ddb",
                "dynamodb.s3.bucket" -> s3_bucket,
                "dynamodb.s3.prefix" -> s3_bucket_prefix
            ))
        ).getDynamicFrame()
        print(dynamicFrame.getNumPartitions())
    }
}
```

DynamoDB 내보내기 JSON 사용 간소화

AWS Glue DynamoDB 내보내기 커넥터를 사용하여 DynamoDB 내보내기를 수행하면 특정 중첩 구조의 JSON 파일이 생성됩니다. 자세한 내용은 [데이터 객체](#)를 참조하세요. AWS Glue는 이러한 구조를 다운스트림 애플리케이션에서 사용하기 쉬운 형태로 중첩 해제를 수행할 수 있는 DynamicFrame 변환을 제공합니다.

두 가지 방법 중 하나로 변환을 호출할 수 있습니다. DynamoDB에서 읽을 메서드를 직접적으로 호출할 때 "true" 값을 사용하여 "dynamodb.simplifyDDBJson" 연결 옵션을 설정할 수 있습니다. AWS Glue 라이브러리에서 독립적으로 사용할 수 있는 메서드로 변환을 직접적으로 호출할 수도 있습니다.

DynamoDB 내보내기로 생성된 다음 스키마를 고려해 보세요.

```

root
|-- Item: struct
|   |-- parentMap: struct
|   |   |-- M: struct
|   |   |   |-- childMap: struct
|   |   |   |   |-- M: struct
|   |   |   |   |   |-- appName: struct
|   |   |   |   |   |   |-- S: string
|   |   |   |   |   |   |-- packageName: struct
|   |   |   |   |   |   |   |-- S: string
|   |   |   |   |   |   |   |-- updatedAt: struct
|   |   |   |   |   |   |   |   |-- N: string
|   |   |-- strings: struct
|   |   |   |-- SS: array
|   |   |   |   |-- element: string
|   |   |-- numbers: struct
|   |   |   |-- NS: array
|   |   |   |   |-- element: string
|   |   |-- binaries: struct
|   |   |   |-- BS: array
|   |   |   |   |-- element: string
|   |-- isDDBJson: struct
|   |   |-- BOOL: boolean
|   |-- nullValue: struct
|   |   |-- NULL: boolean

```

simplifyDDBJson 변환은 이를 다음과 같이 간소화합니다.

```

root

```

```

|-- parentMap: struct
|   |-- childMap: struct
|   |   |-- appName: string
|   |   |-- packageName: string
|   |   |-- updatedAt: string
|-- strings: array
|   |-- element: string
|-- numbers: array
|   |-- element: string
|-- binaries: array
|   |-- element: string
|-- isDDBJson: boolean
|-- nullValue: null

```

Note

simplifyDDBJson은 AWS Glue 3.0 이상 버전에서 사용할 수 있습니다. unnestDDBJson 변환은 DynamoDB 내보내기 JSON을 간소화하는 데도 사용할 수 있습니다. 사용자가 simplifyDDBJson에서 unnestDDBJson으로 전환할 것을 권장합니다.

DynamoDB 작업의 병렬 처리 구성

성능을 향상시키기 위해 DynamoDB 커넥터에 사용할 수 있는 특정 파라미터를 조정할 수 있습니다. 병렬 처리 파라미터를 조정할 때 목표는 프로비저닝된 AWS Glue 작업자의 사용을 극대화하는 것입니다. 더 높은 성능이 필요한 경우 DPU 수를 늘려 작업을 스케일 아웃하는 것이 좋습니다.

ETL 커넥터를 사용할 때 dynamodb.splits 파라미터를 사용하여 DynamoDB 읽기 작업의 병렬 처리를 변경할 수 있습니다. 내보내기 커넥터를 사용하여 읽을 때는 Spark 실행기 병렬 처리를 위한 분할 횟수를 구성할 필요가 없습니다. dynamodb.output.numParallelTasks를 사용하여 DynamoDB 쓰기 작업의 병렬 처리를 변경할 수 있습니다.

DynamoDB ETL 커넥터를 사용하여 읽기

작업 구성에 설정된 최대 작업자 수와 다음 numSlots 계산을 기준으로 dynamodb.splits를 계산하는 것이 좋습니다. 자동 스케일링의 경우 실제 사용 가능한 작업자 수는 해당 한도 내에서 변경될 수 있습니다. 최대 작업자 수 설정에 대한 자세한 내용은 [the section called “Spark 작업 속성 구성”](#)의 작업자 수(NumberOfWorkers)를 참조하세요.

- numExecutors = NumberOfWorkers - 1

상황에 따라 한 실행기는 Spark 드라이버용으로 예약되어 있고 다른 실행기는 데이터 처리에 사용됩니다.

- `numSlotsPerExecutor` =

AWS Glue 3.0 and later versions

- `WorkerType`이 G.1X일 경우 4
- `WorkerType`이 G.2X일 경우 8
- `WorkerType`이 G.4X일 경우 16
- `WorkerType`이 G.8X일 경우 32

AWS Glue 2.0 and legacy versions

- `WorkerType`이 G.1X일 경우 8
- `WorkerType`이 G.2X일 경우 16

- `numSlots` = `numSlotsPerExecutor` * `numExecutors`

사용 가능한 슬롯 수인 `numSlots`에 `dynamodb.splits`를 설정하는 것이 좋습니다.

DynamoDB에 쓰기

`dynamodb.output.numParallelTasks` 파라미터는 다음 계산을 사용하여 Spark 태스크당 WCU를 결정하는 데 사용됩니다.

$$\text{permittedWcuPerTask} = \left(\text{TableWCU} * \text{dynamodb.throughput.write.percent} \right) / \text{dynamodb.output.numParallelTasks}$$

구성이 DynamoDB에 쓰는 Spark 태스크의 수를 정확하게 나타내는 경우 DynamoDB 작성기가 가장 잘 작동합니다. 경우에 따라 쓰기 성능을 개선하기 위해 기본 계산을 재정의해야 할 수 있습니다. 해당 파라미터를 지정하지 않으면 Spark 태스크당 허용되는 WCU는 다음 공식에 따라 자동으로 계산됩니다.

- `numPartitions` = `dynamicframe.getNumPartitions()`
- `numSlots`(이 섹션에서 이전에 정의한 대로)
- `numParallelTasks` = `min(numPartitions, numSlots)`
- 예 1. `DPU=10`, `WorkerType=Standard`. 입력 `DynamicFrame`에는 100개의 RDD 파티션이 있습니다.
 - `numPartitions` = 100
 - `numExecutors` = $(10 - 1) * 2 - 1 = 17$

- $\text{numSlots} = 4 * 17 = 68$
- $\text{numParallelTasks} = \min(100, 68) = 68$
- 예 2. $\text{DPU}=10, \text{WorkerType}=\text{Standard}$. 입력 `DynamicFrame`에는 20개의 RDD 파티션이 있습니다.
 - $\text{numPartitions} = 20$
 - $\text{numExecutors} = (10 - 1) * 2 - 1 = 17$
 - $\text{numSlots} = 4 * 17 = 68$
 - $\text{numParallelTasks} = \min(20, 68) = 20$

Note

레거시 AWS Glue 버전의 작업과 Standard 작업자를 사용하는 작업은 슬롯 수를 계산하는 방법이 다릅니다. 이러한 작업의 성능을 조정해야 하는 경우 지원되는 AWS Glue 버전으로 전환하는 것이 좋습니다.

DynamoDB 연결 옵션 참조

Amazon DynamoDB에 대한 연결을 지정합니다.

연결 옵션은 소스 연결 및 싱크 연결에 따라 다릅니다.

ETL 커넥터를 소스로 사용하는 "connectionType": "dynamodb"

AWS Glue DynamoDB ETL 커넥터 사용 시 "connectionType": "dynamodb"를 소스로 사용하여 다음 연결 옵션을 사용합니다.

- "dynamodb.input.tableName": (필수 사항) 읽을 DynamoDB 테이블입니다.
- "dynamodb.throughput.read.percent": (선택 사항) 사용할 읽기 용량 유닛(RCU)의 백분율. 기본값은 "0.5"로 설정되어 있습니다. 허용되는 값은 "0.1"에서 "1.5"(포함)입니다.
 - 0.5는 기본 읽기 속도입니다. 즉, AWS Glue가 테이블 읽기 용량의 절반을 사용하려고 시도합니다. 값을 0.5보다 높게 설정할 경우 AWS Glue가 요청 속도를 높이고, 값을 0.5보다 낮게 설정할 경우 읽기 요청 속도를 낮춥니다. (실제 읽기 속도는 DynamoDB 테이블의 키 분포가 균일한지 여부와 같은 요소에 따라 달라집니다.)
- DynamoDB 테이블이 온디맨드 모드인 경우 AWS Glue는 테이블의 읽기 용량을 40,000으로 처리합니다. 큰 테이블을 내보내려면 DynamoDB 테이블을 온디맨드 모드로 전환하는 것이 좋습니다.

- "dynamodb.splits": (선택 사항) 읽는 동안 이 DynamoDB 테이블을 몇 개의 분할로 파티셔닝하는지 정의합니다. 기본값은 "1"로 설정되어 있습니다. 허용되는 값은 "1"에서 "1,000,000"(포함)입니다.

1은 병렬 처리가 없음을 나타냅니다. 아래 공식을 사용하여 더 나은 성능을 위해 더 큰 값을 지정하는 것이 좋습니다. 값을 적절하게 설정하는 방법에 대한 자세한 내용은 [the section called "DynamoDB 병렬 처리"](#) 섹션을 참조하세요.

- "dynamodb.sts.roleArn": (선택 사항) 교차 계정 액세스를 위해 수임할 IAM 역할 ARN입니다. 이 파라미터는 AWS Glue 1.0 이상에서 사용 가능합니다.
- "dynamodb.sts.roleSessionName": (선택 사항) STS 세션 이름입니다. 기본값은 "glue-dynamodb-read-sts-session"으로 설정됩니다. 이 파라미터는 AWS Glue 1.0 이상에서 사용 가능합니다.

AWS Glue DynamoDB 내보내기 커넥터를 소스로 사용하는 "connectionType": "dynamodb"

AWS Glue 버전 2.0 이상에서만 사용할 수 있는 AWS Glue DynamoDB 내보내기 커넥터 사용 시 "connectionType": "dynamodb"를 소스로 사용하는 다음 연결 옵션을 사용합니다.

- "dynamodb.export": (필수) 문자열 값:
 - ddb로 설정한 경우 AWS Glue 작업 중 새 ExportTableToPointInTimeRequest가 호출되는 AWS Glue DynamoDB 내보내기 커넥터가 사용 설정됩니다. dynamodb.s3.bucket 및 dynamodb.s3.prefix에서 전달된 위치로 새 내보내기가 생성됩니다.
 - s3로 설정한 경우 AWS Glue DynamoDB 내보내기 커넥터가 사용 설정되지만, 새 DynamoDB 내보내기 생성을 건너뛰고 대신 dynamodb.s3.bucket 및 dynamodb.s3.prefix를 해당 테이블의 과거 내보내기 Amazon S3 위치로 사용합니다.
- "dynamodb.tableArn": (필수 사항) 읽을 DynamoDB 테이블입니다.
- "dynamodb.unnestDDBJson": (선택 사항) 기본값: false. 유효 값: 부울. true로 설정된 경우 내보내기에 있는 DynamoDB JSON 구조의 비중첩 변환을 수행합니다. "dynamodb.unnestDDBJson"과 "dynamodb.simplifyDDBJson"을 동시에 true로 설정하면 오류가 발생합니다. AWS Glue 3.0 이상 버전에서는 DynamoDB 맵 유형을 간소화할 때 더 나은 동작을 위해 "dynamodb.simplifyDDBJson"을 사용하는 것이 좋습니다. 자세한 내용은 [the section called "DynamoDB 내보내기 JSON 사용 간소화"](#) 단원을 참조하십시오.
- "dynamodb.simplifyDDBJson": (선택 사항) 기본값: false. 유효 값: 부울. true로 설정하면 변환을 수행하여 내보내기에 있는 DynamoDB JSON 구조의 스키마를 간소화합니다. "dynamodb.unnestDDBJson" 옵션과 용도는 동일하지만 DynamoDB 맵 유형 또는 DynamoDB 테이블의 중첩된 맵 유형에 대한 지원을 강화합니다. 이 옵션은 AWS Glue 3.0 이상 버전에서 사용할

수 있습니다. "dynamodb.unnestDDBJson"과 "dynamodb.simplifyDDBJson"을 동시에 true로 설정하면 오류가 발생합니다. 자세한 내용은 [the section called "DynamoDB 내보내기 JSON 사용 간소화"](#) 단원을 참조하십시오.

- "dynamodb.s3.bucket": (선택 사항) DynamoDB ExportTableToPointInTime 프로세스가 수행될 Amazon S3 버킷 위치를 나타냅니다. 내보내기의 파일 형식은 DynamoDB JSON입니다.
- "dynamodb.s3.prefix": (선택 사항) DynamoDB ExportTableToPointInTime 로드가 저장될 Amazon S3 버킷 내부의 Amazon S3 접두사 위치를 나타냅니다. dynamodb.s3.prefix와 dynamodb.s3.bucket을 모두 지정하지 않을 경우 이 값은 기본적으로 AWS Glue 작업 구성에 지정된 임시 디렉터리 위치로 설정됩니다. 자세한 내용은 [AWS Glue가 사용하는 특수 파라미터](#)를 참조하세요.
- "dynamodb.s3.bucketOwner": 교차 계정 Amazon S3 액세스를 위해 필요한 버킷 소유자를 나타냅니다.
- "dynamodb.sts.roleArn": (선택 사항) DynamoDB 테이블에 대한 교차 계정 액세스 및/또는 교차 리전 액세스에 대해 수입될 IAM 역할 ARN입니다. 참고: 동일한 IAM 역할 ARN은 ExportTableToPointInTime 요청에 대해 지정된 Amazon S3 위치에 액세스하는 데 사용됩니다.
- "dynamodb.sts.roleSessionName": (선택 사항) STS 세션 이름입니다. 기본값은 "glue-dynamodb-read-sts-session"으로 설정됩니다.
- "dynamodb.exportTime": (선택 사항) 유효 값: ISO-8601 인스턴트를 나타내는 문자열. 내보내기를 수행해야 하는 시점입니다.
- "dynamodb.sts.region": (리전 엔드포인트를 사용하여 리전 간 직접 호출하는 경우 필수임) 읽으려는 DynamoDB 테이블을 호스팅하는 지역입니다.

싱크로 ETL 커넥터를 사용하는 "connectionType": "dynamodb"

싱크로서의 "connectionType": "dynamodb"에는 다음 연결 옵션을 사용합니다.

- "dynamodb.output.tableName": (필수) 쓸 대상 DynamoDB 테이블입니다.
- "dynamodb.throughput.write.percent": (선택 사항) 사용할 쓰기 용량 유닛(WCU)의 백분율입니다. 기본값은 "0.5"로 설정되어 있습니다. 허용되는 값은 "0.1"에서 "1.5"(포함)입니다.
 - 0.5는 기본 쓰기 속도입니다. 즉, AWS Glue가 테이블 쓰기 용량의 절반을 사용하려고 시도합니다. 값을 0.5보다 높게 설정할 경우 AWS Glue가 요청 속도를 높이고, 값을 0.5보다 낮게 설정할 경우 쓰기 요청 속도를 낮춥니다. (실제 쓰기 속도는 DynamoDB 테이블의 키 분포가 균일한지 여부와 같은 요소에 따라 달라집니다.)

- DynamoDB 테이블이 온디맨드 모드인 경우 AWS Glue는 테이블의 쓰기 용량을 40000으로 처리합니다. 큰 테이블을 가져오려면 DynamoDB 테이블을 온디맨드 모드로 전환하는 것이 좋습니다.
- "dynamodb.output.numParallelTasks": (선택 사항) 동시에 DynamoDB에 쓰는 병렬 태스크 수를 정의합니다. Spark 태스크당 허용되는 WCU를 계산하는 데 사용됩니다. 대부분의 경우 AWS Glue는 해당 값에 대해 적절한 기본값을 계산합니다. 자세한 내용은 [the section called "DynamoDB 병렬 처리"](#) 단원을 참조하십시오.
- "dynamodb.output.retry": (선택 사항) DynamoDB에서 ProvisionedThroughputExceededException이 있을 때 수행할 재시도 횟수를 정의합니다. 기본값은 "10"으로 설정됩니다.
- "dynamodb.sts.roleArn": (선택 사항) 교차 계정 액세스를 위해 수입할 IAM 역할 ARN입니다.
- "dynamodb.sts.roleSessionName": (선택 사항) STS 세션 이름입니다. 기본값은 "glue-dynamodb-write-sts-session"으로 설정됩니다.

DynamoDB 테이블에 대한 교차 계정 교차 리전 액세스

AWS Glue ETL 작업은 DynamoDB 테이블에 대한 교차 리전 및 교차 계정 액세스를 지원합니다. AWS Glue ETL 작업은 다른 AWS 계정의 DynamoDB 테이블에서 데이터 읽기와 다른 AWS 계정의 DynamoDB 테이블에 데이터 쓰기를 모두 지원합니다. AWS Glue는 또한 다른 리전의 DynamoDB 테이블에서 읽기와 다른 리전의 DynamoDB 테이블에 쓰기를 모두 지원합니다. 이 섹션에서는 액세스 설정에 대한 지침과 예제 스크립트를 제공합니다.

이 섹션의 절차에서는 IAM 역할을 생성하고 역할에 대한 액세스 권한을 부여하는 IAM 튜토리얼을 참조합니다. 이 튜토리얼에서는 역할 수입에 대해서도 설명하지만 여기서는 대신 작업 스크립트를 사용하여 AWS Glue에서 역할을 수입합니다. 이 튜토리얼에는 일반적인 교차 계정 사례에 대한 정보도 들어 있습니다. 자세한 내용은 IAM User Guide의 [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](#)를 참조하세요.

역할 생성

[튜토리얼의 1단계](#)에 따라 계정 A에서 IAM 역할을 생성합니다. 역할의 권한을 정의할 때 역할이 DynamoDB를 읽고 쓸 수 있도록 AmazonDynamoDBReadOnlyAccess 또는 AmazonDynamoDBFullAccess 등의 기존 정책을 연결하도록 선택할 수 있습니다. 다음 예에서는 권한 정책 AmazonDynamoDBFullAccess를 사용하여 DynamoDBCrossAccessRole이라는 역할을 생성하는 방법을 보여줍니다.

역할에 대한 액세스 권한 부여

IAM User Guide의 [튜토리얼의 1단계](#)에 따라 계정 B가 새로 생성된 역할로 전환하도록 허용할 수 있습니다. 다음 예에서는 다음 문을 사용하여 새 정책을 생성합니다.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "<DynamoDBCrossAccessRole's ARN>"
  }
}
```

그런 다음 이 정책을 DynamoDB에 액세스하는 데 사용할 그룹/역할/사용자에 연결할 수 있습니다.

AWS Glue 작업 스크립트에서 역할 수임

이제 B 계정에 로그인하여 AWS Glue 작업을 생성할 수 있습니다. 작업을 생성하려면 [에서 Spark 작업에 대한 작업 속성 구성 AWS Glue](#)의 지침을 참조하세요.

작업 스크립트에서 DynamoDBCrossAccessRole 역할을 수임하려면 dynamodb.sts.roleArn 파라미터를 사용해야 합니다. 이 역할을 수임하면 계정 B의 DynamoDB에 액세스하는 데 필요한 임시 자격 증명을 얻을 수 있습니다. 다음 예제 스크립트를 검토하세요.

리전 간 교차 계정 읽기의 경우(ETL 커넥터):

```
import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-east-1",
```

```

    "dynamodb.input.tableName": "test_source",
    "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
  }
)
dyf.show()
job.commit()

```

리전 간 교차 계정 읽기의 경우(ELT 커넥터):

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.export": "ddb",
        "dynamodb.tableArn": "<test_source ARN>",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)
dyf.show()
job.commit()

```

리전 간 읽기 및 교차 계정 쓰기의 경우:

```

import sys
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.utils import getResolvedOptions

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
glue_context= GlueContext(SparkContext.getOrCreate())
job = Job(glue_context)
job.init(args["JOB_NAME"], args)

```

```

dyf = glue_context.create_dynamic_frame_from_options(
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-east-1",
        "dynamodb.input.tableName": "test_source"
    }
)
dyf.show()

glue_context.write_dynamic_frame_from_options(
    frame=dyf,
    connection_type="dynamodb",
    connection_options={
        "dynamodb.region": "us-west-2",
        "dynamodb.output.tableName": "test_sink",
        "dynamodb.sts.roleArn": "<DynamoDBCrossAccessRole's ARN>"
    }
)

job.commit()

```

Kinesis 연결

데이터 카탈로그 테이블에 저장된 정보를 사용하거나 데이터 스트림에 직접 액세스할 수 있는 정보를 제공하여 Amazon Kinesis Data Streams에서 읽고 쓸 수 있습니다. Kinesis에서 DataFrame Spark로 정보를 읽은 다음 이를 Glue로 변환할 수 있습니다. AWS DynamicFrame DynamicFrames Kinesis에 JSON 형식으로 쓸 수 있습니다. 데이터 스트림에 직접 액세스하는 경우 이러한 옵션을 사용하여 데이터 스트림에 액세스하는 방법에 대한 정보를 제공합니다.

getCatalogSource 또는 create_data_frame_from_catalog를 통해 Kinesis 스트리밍 소스의 레코드를 사용하는 경우 작업에 데이터 카탈로그 데이터베이스 및 테이블 이름 정보가 있으며, 해당 정보를 사용하여 Kinesis 스트리밍 소스에서 읽기 위한 몇 가지 기본 파라미터를 얻을 수 있습니다. getSource, getSourceWithFormat, createDataFrameFromOptions 또는 create_data_frame_from_options를 사용하는 경우 여기에 설명된 연결 옵션을 통해 이러한 기본 파라미터를 지정해야 합니다.

GlueContext 클래스에 지정된 메서드에 대해 다음 인수를 사용하여 Kinesis에 대한 연결 옵션을 지정할 수 있습니다.

- Scala

- `connectionOptions`: `getSource`, `createDataFrameFromOptions`, `getSink`와(과) 함께 사용
- `additionalOptions`: `getCatalogSource`, `getCatalogSink`와 함께 사용
- `options`: `getSourceWithFormat`, `getSinkWithFormat`와 함께 사용
- Python
 - `connection_options`: `create_data_frame_from_options`, `write_dynamic_frame_from_options`와 함께 사용
 - `additional_options`: `create_data_frame_from_catalog`, `write_dynamic_frame_from_catalog`와 함께 사용
 - `options`: `getSource`, `getSink`와 함께 사용

스트리밍 ETL 작업에 대한 참고 및 제한 사항은 [the section called “스트리밍 ETL 참고 사항 및 제한 사항”](#) 섹션을 참조하세요.

Kinesis 구성

AWS Glue Spark 작업에서 Kinesis 데이터 스트림에 연결하려면 몇 가지 사전 요구 사항이 필요합니다.

- 읽는 경우 AWS Glue 작업에는 Kinesis 데이터 스트림에 대한 읽기 액세스 수준 IAM 권한이 있어야 합니다.
- 작성하는 경우 AWS Glue 작업에는 Kinesis 데이터 스트림에 대한 쓰기 액세스 수준 IAM 권한이 있어야 합니다.

경우에 따라 추가 필수 조건을 구성해야 합니다.

- AWS Glue 작업이 추가 네트워크 연결 (일반적으로 다른 데이터세트에 연결) 으로 구성되어 있고 이러한 연결 중 하나가 Amazon VPC 네트워크 옵션을 제공하는 경우, 이 작업은 Amazon VPC를 통해 통신하도록 지시합니다. 이 경우 Amazon VPC를 통해 통신하도록 Kinesis 데이터 스트림도 구성해야 합니다. Amazon VPC와 Kinesis 데이터 스트림 사이에서 인터페이스 VPC 엔드포인트를 생성하면 됩니다. 자세한 내용은 [Using Kinesis Data Streams with Interface VPC Endpoints](#)를 참조하세요.
- 다른 계정에서 Amazon Kinesis Data Streams를 지정할 때 크로스 계정 액세스를 허용하도록 역할과 정책을 설정해야 합니다. 자세한 내용은 [예: 다른 계정의 Kinesis 스트림에서 읽기](#)를 참조하세요.

스트리밍 ETL 작업 필수 조건에 대한 자세한 내용은 [the section called “스트리밍 ETL 작업”](#) 섹션을 참조하세요.

예제: Kinesis 스트림에서 읽기

예제: Kinesis 스트림에서 읽기

[the section called “forEachBatch”](#)과(와) 함께 사용합니다.

Amazon Kinesis 스트리밍 소스의 예:

```
kinesis_options =
  { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
    "startingPosition": "TRIM_HORIZON",
    "inferSchema": "true",
    "classification": "json"
  }
data_frame_datasource0 =
  glueContext.create_data_frame.from_options(connection_type="kinesis",
  connection_options=kinesis_options)
```

예: Kinesis 스트림에 쓰기

예제: Kinesis 스트림에서 읽기

[the section called “forEachBatch”](#)과(와) 함께 사용합니다.

Amazon Kinesis 스트리밍 소스의 예:

```
kinesis_options =
  { "streamARN": "arn:aws:kinesis:us-east-2:777788889999:stream/fromOptionsStream",
    "startingPosition": "TRIM_HORIZON",
    "inferSchema": "true",
    "classification": "json"
  }
data_frame_datasource0 =
  glueContext.create_data_frame.from_options(connection_type="kinesis",
  connection_options=kinesis_options)
```

Kinesis 연결 옵션 참조

Amazon Kinesis Data Streams에 대한 연결 옵션을 지정합니다.

Kinesis 스트리밍 데이터 원본에 대한 다음 연결 옵션을 사용합니다.

- "streamARN" (필수) 읽기/쓰기에 사용됩니다. Kinesis 데이터 스트림의 ARN.
- "classification" (읽기 필수) 읽기에 사용됩니다. 레코드의 데이터에서 사용하는 파일 형식. 데이터 카탈로그를 통해 제공되지 않는 한 필수입니다.
- "streamName" – (선택 사항) 읽기에 사용됩니다. 읽을 Kinesis 데이터 스트림의 이름. `endpointUrl`와(과) 함께 사용됩니다.
- "endpointUrl" – (선택 사항) 읽기에 사용됩니다. 기본값: "https://kinesis.us-east-1.amazonaws.com". Kinesis 스트림의 AWS 엔드포인트입니다. 특정 지역에 연결하는 경우가 아니면 변경하지 않아도 됩니다.
- "partitionKey" – (선택 사항) 쓰기에 사용됩니다. 레코드를 생성할 때 사용되는 Kinesis 파티션 키.
- "delimiter" (선택 사항) 읽기에 사용됩니다. `classification`이(가) CSV일 때 사용되는 값 구분 기호입니다. 기본값은 ','입니다.
- "startingPosition": (선택 사항) 읽기에 사용됩니다. 데이터를 읽을 Kinesis 데이터 스트림의 시작 위치입니다. 가능한 값은 yyyy-mm-ddTHH:MM:SSZ 패턴에서 UTC 형식의 타임스탬프 문자열이나 "latest", "trim_horizon" 또는 "earliest"입니다(여기서, Z는 UTC 시간대 오프셋(+/-)임, 예: '2023-04-04T08:00:00-04:00'). 기본 값은 "latest"입니다. 참고: UTC 형식의 타임스탬프 문자열은 AWS Glue 버전 4.0 이상에서만 지원됩니다. "startingPosition"
- "failOnDataLoss": (선택 사항) 활성 샤드가 누락되거나 만료된 경우 작업이 실패합니다. 기본 값은 "false"입니다.
- "awsSTSRoleARN": (선택 사항) 읽기/쓰기에 사용됩니다. () 를 사용하여 위임할 역할의 Amazon 리소스 이름 AWS Security Token Service (AWS STS ARN) 입니다. 이 역할에는 Kinesis 데이터 스트림에 대한 레코드 작업을 설명하거나 읽을 수 있는 권한이 있어야 합니다. 다른 계정의 데이터 스트림에 액세스할 때 이 파라미터를 사용해야 합니다. "awsSTSSessionName"과(와) 함께 사용됩니다.
- "awsSTSSessionName": (선택 사항) 읽기/쓰기에 사용됩니다. AWS STS을(를) 사용하여 역할을 맡는 세션의 식별자입니다. 다른 계정의 데이터 스트림에 액세스할 때 이 파라미터를 사용해야 합니다. "awsSTSRoleARN"과(와) 함께 사용됩니다.
- "awsSTSEndpoint": (선택 사항) 위임된 역할을 사용하여 Kinesis에 연결할 때 사용할 AWS STS 엔드포인트입니다. 이를 통해 VPC에서 리전 AWS STS 엔드포인트를 사용할 수 있지만, 기본 글로벌 엔드포인트로는 불가능합니다.
- "maxFetchTimeInMs": (선택 사항) 읽기에 사용됩니다. 작업 실행기가 Kinesis 데이터 스트림에서 현재 배치에 대한 레코드를 읽는 데 걸리는 최대 시간(밀리초(ms) 단위로 지정)입니다. 이 시간 내에 여러 개의 GetRecords API 호출을 할 수 있습니다. 기본 값은 1000입니다.

- "maxFetchRecordsPerShard": (선택 사항) 읽기에 사용됩니다. 마이크로 배치에 따라 Kinesis 데이터 스트림에서 샤드당 가져올 최대 레코드 수입니다. 참고: 스트리밍 작업이 이미 Kinesis의 동일한 get-records 호출에서 추가 레코드를 읽은 경우 클라이언트가 이 제한을 초과할 수 있습니다. maxFetchRecordsPerShard가 엄격해야 한다면 maxRecordPerRead의 배수여야 합니다. 기본 값은 100000입니다.
- "maxRecordPerRead": (선택 사항) 읽기에 사용됩니다. 각 getRecords 작업에서 Kinesis 데이터 스트림에서 가져올 최대 레코드 수. 기본 값은 10000입니다.
- "addIdleTimeBetweenReads": (선택 사항) 읽기에 사용됩니다. 연속 두 getRecords 작업 사이에 시간 지연을 추가합니다. 기본 값은 "False"입니다. 이 옵션은 Glue 버전 2.0 이상에서만 구성할 수 있습니다.
- "idleTimeBetweenReadsInMs": (선택 사항) 읽기에 사용됩니다. 연속 두 getRecords 작업 사이의 최소 시간 지연으로, ms 단위로 지정됩니다. 기본 값은 1000입니다. 이 옵션은 Glue 버전 2.0 이상에서만 구성할 수 있습니다.
- "describeShardInterval": (선택 사항) 읽기에 사용됩니다. 스크립트가 리샤딩을 고려하기 위한 두 ListShards API 호출 사이의 최소 시간 간격. 자세한 내용은 Amazon Kinesis Data Streams Developer Guide의 [Strategies for Resharding](#)을 참조하세요. 기본 값은 1s입니다.
- "numRetries": (선택 사항) 읽기에 사용됩니다. Kinesis Data Streams API 요청의 최대 재시도 횟수입니다. 기본 값은 3입니다.
- "retryIntervalMs": (선택 사항) 읽기에 사용됩니다. Kinesis Data Streams API 호출을 재시도하기 전의 휴지 기간(ms 단위로 지정)입니다. 기본 값은 1000입니다.
- "maxRetryIntervalMs": (선택 사항) 읽기에 사용됩니다. Kinesis Data Streams API 호출을 두 번 재시도하는 사이의 최대 휴지 시간(ms 단위로 지정)입니다. 기본 값은 10000입니다.
- "avoidEmptyBatches": (선택 사항) 읽기에 사용됩니다. 배치가 시작되기 전에 Kinesis 데이터 스트림에서 읽지 않은 데이터를 확인하여 빈 마이크로 배치 작업 생성을 방지합니다. 기본 값은 "False"입니다.
- "schema": (inferSchema가 거짓으로 설정된 경우 필수 사항) 읽기에 사용됩니다. 페이로드를 처리하는 데 사용하는 스키마. 분류가 avro인 경우 제공된 스키마는 Avro 스키마 형식이어야 합니다. 분류가 avro가 아닌 경우 제공된 스키마는 DDL 스키마 형식이어야 합니다.

다음은 스키마의 예입니다.

Example in DDL schema format

```
`column1` INT, `column2` STRING , `column3` FLOAT
```

Example in Avro schema format

```
{
  "type": "array",
  "items":
  {
    "type": "record",
    "name": "test",
    "fields":
    [
      {
        "name": "_id",
        "type": "string"
      },
      {
        "name": "index",
        "type":
        [
          "int",
          "string",
          "float"
        ]
      }
    ]
  }
}
```

- "inferSchema": (선택 사항) 읽기에 사용됩니다. 기본값은 'false'입니다. 'true'로 설정하면 스키마가 런타임 시 foreachbatch 내의 페이로드에서 감지됩니다.
- "avroSchema": (더 이상 사용되지 않음) 읽기에 사용됩니다. Avro 형식을 사용할 때 Avro 데이터의 스키마를 지정하는 데 사용되는 파라미터입니다. 이 파라미터는 이제 사용 중단되었습니다. schema 파라미터를 사용합니다.
- "addRecordTimestamp": (선택 사항) 읽기에 사용됩니다. 이 옵션이 'true'로 설정되면 데이터 출력에는 이름이 '_src_timestamp'라는 추가 열이 포함됩니다. 이 열은 스트림에서 해당 레코드를 수신한 시간을 나타냅니다. 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.
- "emitConsumerLagMetrics": (선택 사항) 읽기에 사용됩니다. 옵션을 'true'로 설정하면 각 배치에 대해 스트림이 수신한 가장 오래된 레코드와 스트림이 수신된 시간 사이의 기간 동안 측정치를 내보냅니다. AWS Glue CloudWatch 메트릭의 이름은 "glue.driver.streaming"입니다.

maxConsumerLagInMs". 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.

- "fanoutConsumerARN": (선택 사항) 읽기에 사용됩니다. streamARN에서 지정된 스트림에 대한 Kinesis 스트림 소비자의 ARN. Kinesis 연결에 대해 향상된 팬아웃 모드를 활성화하는 데 사용됩니다. 향상된 팬아웃과 함께 Kinesis 스트림을 사용하는 방법에 대한 자세한 내용은 [the section called "Kinesis 스트리밍 작업에서 향상된 팬아웃 사용"](#) 섹션을 참조하세요.
- "recordMaxBufferedTime" - (선택 사항) 쓰기에 사용됩니다. 기본값: 1000(ms). 레코드 쓰기를 대기하는 중 레코드가 버퍼링되는 최대 시간.
- "aggregationEnabled" - (선택 사항) 쓰기에 사용됩니다. 기본값: true. Kinesis로 보내기 전에 레코드를 집계해야 하는지 여부를 지정합니다.
- "aggregationMaxSize" - (선택 사항) 쓰기에 사용됩니다. 기본값: 51200(바이트). 레코드가 이 한도보다 크면 애그리게이터를 우회합니다. 참고 Kinesis는 레코드 크기를 50KB로 제한합니다. 이 값이 50KB를 초과하도록 설정하면 Kinesis에서 크기가 초과된 레코드를 거부합니다.
- "aggregationMaxCount" - (선택 사항) 쓰기에 사용됩니다. 기본값: 4294967295. 집계된 레코드에 포함할 최대 항목 수.
- "producerRateLimit" - (선택 사항) 쓰기에 사용됩니다. 기본값: 150(%). 단일 생성자(예: 작업)에서 보내는 샤드당 처리량을 백엔드 한도의 백분율로 제한합니다.
- "collectionMaxCount" - (선택 사항) 쓰기에 사용됩니다. 기본값: 500. PutRecords 요청 시 포함할 수 있는 최대 품목 수입니다.
- "collectionMaxSize" - (선택 사항) 쓰기에 사용됩니다. 기본값: 5242880(바이트). PutRecords 요청과 함께 전송할 수 있는 최대 데이터 양.

Kinesis 스트리밍 작업에서 향상된 팬아웃 사용

향상된 팬아웃 소비자는 일반 소비자보다 더 높은 전용 처리량으로 Kinesis 스트림에서 레코드를 수신할 수 있습니다. 이는 Kinesis 소비자에게 데이터(예: 작업)를 제공하는 데 사용되는 전송 프로토콜을 최적화하는 방식으로 지원됩니다. Kinesis의 향상된 팬아웃에 대한 자세한 내용은 [Kinesis 설명서](#)를 참조하세요.

향상된 팬아웃 모드에서는 maxRecordPerRead 및 idleTimeBetweenReadsInMs 연결 옵션이 더 이상 적용되지 않습니다. 향상된 팬아웃을 사용할 때 해당 파라미터는 구성할 수 없기 때문입니다. 재시도를 위한 구성 옵션은 설명한 대로 수행됩니다.

다음 절차를 사용하여 스트리밍 작업에 대한 향상된 팬아웃을 활성화 및 비활성화합니다. 스트림에서 데이터를 소비하는 각 작업에 대해 스트림 소비자를 등록해야 합니다.

작업에서 향상된 팬아웃 소비를 활성화하려면:

1. Kinesis API를 사용하여 작업에 대한 스트림 소비자를 등록합니다. [Kinesis 설명서](#)의 register a consumer with enhanced fan-out using the Kinesis Data Streams API에 대한 지침을 따릅니다. 첫 번째 단계인 [RegisterStreamConsumer](#)를 직접적으로 호출합니다. 요청은 ARN, *consumerARN*을 반환해야 합니다.
2. 연결 메서드 인수에서 fanoutConsumerARN 연결 옵션을 *consumerARN*으로 설정합니다.
3. 작업을 다시 시작합니다.

작업에서 향상된 팬아웃 소비를 비활성화하려면:

1. 메서드 직접 호출에서 fanoutConsumerARN 연결 옵션을 제거합니다.
2. 작업을 다시 시작합니다.
3. [Kinesis 설명서](#)의 deregister a consumer에 대한 지침을 따릅니다. 이 지침은 콘솔에도 적용되지만 Kinesis API를 통해서도 수행할 수 있습니다. Kinesis API를 통한 스트림 소비자 등록 취소에 대한 자세한 내용은 Kinesis 설명서의 [DeregisterStreamConsumer](#)를 참조하세요.

Amazon S3 연결

AWS Glue for Spark를 사용하여 Amazon S3에서 파일을 읽고 쓸 수 있습니다. AWS Glue for Spark는 CSV, Avro, JSON, Orc, Parquet을 포함하여 기본적으로 Amazon S3에 저장되는 여러 가지 공통 데이터 형식을 지원합니다. 지원되는 데이터 형식에 대한 자세한 내용은 [the section called “데이터 포맷 옵션”](#) 섹션을 참조하세요. 각 데이터 형식은 AWS Glue 기능의 여러 가지 세트를 지원할 수 있습니다. 특정 기능 지원에 대해서는 해당 데이터 형식 페이지를 참조하세요. 또한 Hudi, Iceberg 및 Delta Lake 데이터 레이크 프레임워크에 저장되는 버전 관리되는 파일을 읽고 쓸 수 있습니다. 데이터 레이크 프레임워크에 대한 자세한 내용은 [the section called “데이터 레이크 프레임워크”](#) 섹션을 참조하세요.

AWS Glue를 사용하면 쓰는 동안 Amazon S3 객체를 폴더 구조로 파티셔닝한 다음 파티션별로 검색하여 간단한 구성으로 성능을 개선할 수 있습니다. 성능을 개선하기 위해 데이터를 변환할 때 작은 파일을 함께 그룹화하도록 구성을 설정할 수도 있습니다. Amazon S3에서 bzip2 및 gzip 아카이브를 읽고 쓸 수 있습니다.

주제

- [S3 연결 구성](#)
- [Amazon S3 연결 옵션 참조](#)
- [데이터 형식에 대한 연결 구문은 더 이상 사용되지 않습니다.](#)

- [Amazon S3 스토리지 클래스 제외](#)
- [AWS Glue에서 ETL 출력의 파티션 관리](#)
- [입력 파일을 더 큰 그룹에서 읽기](#)
- [Amazon S3용 Amazon VPC 엔드포인트](#)

S3 연결 구성

AWS Glue에서 Spark 작업으로 Amazon S3에 연결하려면 몇 가지 필수 조건이 필요합니다.

- AWS Glue 작업에는 관련 Amazon S3 버킷에 대한 IAM 권한이 있어야 합니다.

경우에 따라 추가 필수 조건을 구성해야 합니다.

- 크로스 계정 액세스를 구성할 때 Amazon S3 버킷에서 적절한 액세스 제어.
- 보안상의 이유로 Amazon S3 요청을 Amazon VPC를 통해 라우팅하도록 선택할 수 있습니다. 단, 이러한 접근 방식은 대역폭 및 가용성 문제를 일으킬 수 있습니다. 자세한 내용은 [the section called "Amazon S3용 Amazon VPC 엔드포인트"](#) 단원을 참조하십시오.

Amazon S3 연결 옵션 참조

Amazon S3에 대한 연결을 지정합니다.

Amazon S3에서는 테이블이 아닌 파일을 관리하므로 이 문서에 제공된 연결 속성을 지정하는 것 외에도 파일 유형에 대한 추가 구성을 지정해야 합니다. 데이터 형식 옵션을 통해 이 정보를 지정합니다. 형식 옵션에 대한 자세한 내용은 [the section called "데이터 포맷 옵션"](#) 섹션을 참조하세요. AWS Glue 데이터 카탈로그와 통합하여 이 정보를 지정할 수도 있습니다.

연결 옵션과 형식 옵션 간의 차이점에 대한 예로, [the section called "create_dynamic_frame_from_options"](#) 메서드에서 `connection_type`, `connection_options`, `format` 및 `format_options`를 사용하는 방법을 고려합니다. 이 섹션에서는 `connection_options`에 제공된 파라미터에 대해 자세히 논의합니다.

"connectionType": "s3"에는 다음 연결 옵션을 사용합니다.

- "paths": (필수 사항) 읽을 소스 Amazon S3 경로 목록입니다.
- "exclusions": (선택사항) 배제할 Unix 식 glob 패턴의 JSON 목록이 포함된 문자열. 예를 들어 "[\\".***.pdf\\]"는 모든 PDF 파일을 제외합니다. AWS Glue에서 지원하는 glob 구문에 대한 자세한 내용은 [포함 및 제외 패턴](#)을 참조하십시오.

- "compressionType": 또는 "compression": (선택 사항) 데이터 압축 방식을 지정합니다. Amazon S3 소스에는 "compressionType"을 사용하고 Amazon S3 대상에는 "compression"을 사용합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip2"입니다. 특정 형식에 대해 추가 압축 형식이 지원될 수 있습니다. 자세한 기능 지원은 데이터 형식 페이지를 참조하세요.
- "groupFiles": (선택 사항) 입력에 50,000개 이상의 파일이 포함된 경우에는 기본적으로 파일 그룹화가 설정됩니다. 파일이 50,000개 미만일 때 그룹화를 설정하려면 이 파라미터를 "inPartition"으로 설정합니다. 파일이 50,000개 이상일 때 그룹화를 비활성화하려면 이 파라미터를 "none"으로 설정합니다.
- "groupSize": (선택 사항) 대상 그룹 크기(바이트). 입력 데이터 크기와 클러스터 크기에 따라 기본 값을 계산합니다. 입력 파일이 50,000개 미만일 때는 "groupFiles"을 "inPartition"으로 설정해야 적용됩니다.
- "recurse": (선택 사항) true로 설정할 경우 지정된 경로의 모든 하위 디렉터리에 있는 파일을 반복적으로 읽습니다.
- "maxBand": (선택 사항, 고급) 이 옵션은 s3 목록이 일정할 것으로 예상하는 시간(밀리초)을 제어합니다. 마지막 maxBand밀리초에 속하는 수정 타임스탬프가 있는 파일은 특히 JobBookmarks를 사용하여 Amazon S3 최종 일관성을 고려할 때 추적됩니다. 대부분의 사용자는 이 옵션을 설정할 필요가 없습니다. 기본값은 900,000밀리초 또는 15분입니다.
- "maxFilesInBand": (선택 사항, 고급) 이 옵션은 마지막 maxBand초부터 저장할 최대 파일 수를 지정합니다. 이 수를 초과할 경우 추가 파일은 건너뛰고 다음 작업 실행에서만 처리됩니다. 대부분의 사용자는 이 옵션을 설정할 필요가 없습니다.
- "isFailFast": (선택 사항) 이 옵션은 AWS Glue ETL 작업에서 리더 구문 분석 예외가 발생하는지 여부를 결정합니다. true로 설정하면 Spark 태스크의 4번 재시도가 데이터를 제대로 구문 분석하지 못하면 작업이 빠르게 실패합니다.
- "catalogPartitionPredicate": (선택 사항) 읽기에 사용됩니다. SQL WHERE 절의 콘텐츠입니다. 파티션 수가 매우 많은 데이터 카탈로그 테이블에서 읽을 때 사용됩니다. 데이터 카탈로그 인덱스에서 일치하는 파티션을 검색합니다. [the section called "create_dynamic_frame_from_catalog"](#) 메서드 및 기타 유사한 메서드의 옵션인 push_down_predicate와 함께 사용됩니다. 자세한 내용은 [the section called "카탈로그 파티션 조건자"](#) 단원을 참조하십시오.
- "partitionKeys": (선택 사항) 쓰기에 사용됩니다. 열 레이블 문자열의 배열입니다. AWS Glue는 이 구성에 지정된 대로 데이터를 파티셔닝합니다. 자세한 내용은 [the section called "파티션 작성"](#) 단원을 참조하십시오.
- "excludeStorageClasses": (선택 사항) 읽기에 사용됩니다. Amazon S3 스토리지 클래스를 지정하는 문자열 배열입니다. AWS Glue는 이 구성을 기반으로 Amazon S3 객체를 제외합니다. 자세한 내용은 [the section called "Amazon S3 스토리지 클래스 제외"](#) 단원을 참조하십시오.

데이터 형식에 대한 연결 구문은 더 이상 사용되지 않습니다.

특정 연결 유형 구문을 사용하여 특정 데이터 형식에 액세스할 수 있습니다. 이 구문은 더 이상 사용되지 않습니다. 대신 [the section called “데이터 포맷 옵션”](#)에 제공된 s3 연결 유형과 형식 옵션을 사용하여 형식을 지정하는 것이 좋습니다.

```
"connectionType": "Orc"
```

Amazon S3에 저장된 파일에 [Apache ORC\(Hive Optimized Row Columnar\)](#) 파일 포맷으로 연결을 지정합니다.

```
"connectionType": "orc"에는 다음 연결 옵션을 사용합니다.
```

- paths: (필수 사항) 읽을 소스 Amazon S3 경로 목록입니다.
- (기타 옵션 이름/값 페어): 형식 지정 옵션을 포함한 모든 추가 옵션이 SparkSQL DataSource에 직접 전달됩니다.

```
"connectionType": "parquet"
```

Amazon S3에 [Apache Parquet](#) 파일 포맷으로 저장된 파일에 대한 연결을 지정합니다.

```
"connectionType": "parquet"에는 다음 연결 옵션을 사용합니다.
```

- paths: (필수 사항) 읽을 소스 Amazon S3 경로 목록입니다.
- (기타 옵션 이름/값 페어): 형식 지정 옵션을 포함한 모든 추가 옵션이 SparkSQL DataSource에 직접 전달됩니다.

Amazon S3 스토리지 클래스 제외

Amazon Simple Storage Service(Amazon S3)에서 파일 또는 파티션을 읽는 AWS Glue ETL 작업을 실행하는 경우 일부 Amazon S3 스토리지 클래스 유형을 제외할 수 있습니다.

Amazon S3에서 제공되는 스토리지 클래스는 다음과 같습니다.

- STANDARD - 자주 액세스하는 데이터의 범용 스토리지일 때
- INTELLIGENT_TIERING - 액세스 패턴이 불확실하거나 바뀌는 데이터일 때
- STANDARD_IA 및 ONEZONE_IA - 저장 기간이 길지만 액세스 횟수가 비교적 적은 데이터일 때
- GLACIER, DEEP_ARCHIVE 및 REDUCED_REDUNDANCY - 장기 아카이브 및 디지털 보존이 필요할 때

자세한 내용은 Amazon S3 Developer Guide의 [Amazon S3 Storage Classes](#)를 참조하세요.

이번 단원의 예제는 GLACIER 및 DEEP_ARCHIVE 스토리지 클래스를 제외하는 방법을 나타낸 것입니다. 두 클래스는 파일을 나열하는 데 사용되지만 복원하지 않을 경우 파일을 읽어오지는 못합니다. 자세한 내용은 Amazon S3 Developer Guide의 [Restoring Archived Objects](#)를 참조하세요.

스토리지 클래스 제외를 사용하면 해당 스토리지 클래스 계층에서 파티션이 포함된 테이블에 대해 AWS Glue 작업을 실행할 수 있습니다. 제외를 사용하지 않으면 해당 계층에서 데이터를 읽어오는 작업은 다음 오류와 함께 중단됩니다. AmazonS3Exception: The operation is not valid for the object's storage class.

AWS Glue에서 Amazon S3 스토리지 클래스를 필터링할 수 있는 여러 가지 방법이 있습니다.

주제

- [동적 프레임 생성 시 Amazon S3 스토리지 클래스 제외](#)
- [Data Catalog 테이블에서 Amazon S3 스토리지 클래스 제외](#)

동적 프레임 생성 시 Amazon S3 스토리지 클래스 제외

동적 프레임을 생성할 때 Amazon S3 스토리지 클래스를 제외하려면 `additionalOptions`에서 `excludeStorageClasses`를 사용합니다. AWS Glue는 자동으로 고유한 Amazon S3 Lister 구현을 사용하여 지정된 스토리지 클래스에 해당하는 파일을 나열하고 제외합니다.

다음 Python 및 Scala 예제는 동적 프레임을 생성할 때 GLACIER 및 DEEP_ARCHIVE 스토리지 클래스를 제외하는 방법을 나타낸 것입니다.

Python 예제:

```
glueContext.create_dynamic_frame.from_catalog(
    database = "my_database",
    tableName = "my_table_name",
    redshift_tmp_dir = "",
    transformation_ctx = "my_transformation_context",
    additional_options = {
        "excludeStorageClasses" : ["GLACIER", "DEEP_ARCHIVE"]
    }
)
```

Scala 예제:

```
val* *df = glueContext.getCatalogSource(
    nameSpace, tableName, "", "my_transformation_context",
    additionalOptions = JsonOptions(
        Map("excludeStorageClasses" -> List("GLACIER", "DEEP_ARCHIVE"))
    )
).getDynamicFrame()
```

Data Catalog 테이블에서 Amazon S3 스토리지 클래스 제외

AWS Glue ETL 작업에서 사용할 스토리지 클래스 제외를 AWS Glue Data Catalog의 테이블 파라미터로 지정할 수 있습니다. 이러한 파라미터는 AWS Command Line Interface(AWS CLI) 를 사용하거나 API에서 프로그래밍 방식으로 CreateTable 작업에 추가할 수 있습니다. 자세한 내용은 [테이블 구조](#) 단원과 [CreateTable](#) 단원을 참조하십시오.

그 밖에 AWS Glue 콘솔에서 제외된 스토리지 클래스를 지정하는 방법도 있습니다.

Amazon S3 스토리지 클래스를 제외하려면(콘솔)

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 테이블을 선택합니다.
3. 목록에서 테이블 이름과 테이블 편집을 차례대로 선택합니다.
4. 테이블 속성에서 **excludeStorageClasses**를 키로, 그리고 `["GLACIER", "DEEP_ARCHIVE"]`를 값으로 추가합니다.
5. 적용을 선택합니다.

AWS Glue에서 ETL 출력의 파티션 관리

파티셔닝은 데이터 세트를 조직하기 위한 중요한 기술로써 효율적으로 데이터 세트를 쿼리할 수 있습니다. 하나 이상의 열에 대한 구별되는 값을 기반으로 계층적 디렉터리 구조에서 데이터를 정리합니다.

예를 들어, 날짜를 기준으로 Amazon Simple Storage Service(Amazon S3)에서 애플리케이션 로그를 분할하고 연, 월, 일을 기준으로 쪼갤 수 있습니다. 하루만큼의 데이터와 대응하는 파일은 `s3://my_bucket/logs/year=2018/month=01/day=23`와 같은 접두사 아래 위치합니다. 이제 AWS Glue는 Amazon Athena, Amazon Redshift Spectru과 같은 시스템과 더불어 Amazon S3의 기본 데이터를 모두 읽지 않고도 이러한 파티션을 사용하여 파티션 값을 기준으로 데이터를 필터링할 수 있습니다.

크롤러는 파일 유형 및 스키마를 추론할 뿐만 아니라 크롤러가 AWS Glue Data Catalog를 채울 경우 데이터 집합의 파티션 구조를 자동적으로 확인합니다. 결과로 얻은 파티션 열을 AWS Glue ETL 작업 또는 Amazon Athena와 같은 쿼리 엔진에서 쿼리할 수 있습니다.

테이블을 크롤링한 후 크롤러가 생성한 파티션을 볼 수 있습니다. AWS Glue 콘솔의 왼쪽 탐색 창에서 테이블(Tables)을 선택합니다. 크롤러에서 생성한 테이블을 선택한 다음 [파티션 보기(View Partitions)]를 선택합니다.

key=val 스타일에서 Apache Hive 스타일로 분할된 경로의 경우, 크롤러는 열 이름에 자동으로 키 이름을 채웁니다. 그렇지 않은 경우에는 partition_0, partition_1 같은 기본 이름을 사용합니다. 콘솔에서 기본 이름을 변경할 수 있습니다. 이를 위해 테이블로 이동합니다. 인덱스 탭 아래에 인덱스가 있는지 확인합니다. 인덱스가 있는 경우 계속 진행하려면 해당 인덱스를 삭제해야 합니다(이후 새 열 이름을 사용하여 다시 생성할 수 있음). 그런 다음 스키마 편집을 선택하고 여기에서 파티션 열의 이름을 수정합니다.

ETL 스크립트에서 파티션 열에 필터링합니다. 파티션 정보는 Data Catalog에 저장되므로 from_catalog API 호출을 사용하여 DynamicFrame에 파티션 열을 포함합니다. 예를 들어 create_dynamic_frame.from_options 대신 create_dynamic_frame.from_catalog을 사용하세요.

파티셔닝은 데이터 스캔을 줄이는 최적화 기법입니다. 이 기법이 적절한 시기를 식별하는 프로세스에 대한 자세한 내용은 AWS 권장 가이드의 Apache Spark용 AWS Glue 작업 성능 조정 모범 사례 가이드에 있는 [데이터 스캔량 감소](#)를 참조하세요.

푸시다운 조건자를 사용하여 예비 필터링

많은 경우, 조건자를 푸시다운하여 데이터 세트 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링할 수 있습니다. 데이터 집합 전체를 읽는 대신 DynamicFrame에서 필터링한 다음 Data Catalog에서 파티션 메타데이터에 직접 필터링을 적용할 수 있습니다. 그런 다음 실제로 필요한 것을 DynamicFrame에서 나열하고 읽습니다.

예를 들어, Python에서는 다음을 작성할 수 있습니다.

```
glue_context.create_dynamic_frame.from_catalog(
    database = "my_S3_data_set",
    table_name = "catalog_data_table",
    push_down_predicate = my_partition_predicate)
```

조건자 표현식을 만족하는 Data Catalog의 파티션을 로딩만 하는 DynamicFrame을 생성합니다. 로딩하는 데이터의 서버셋이 얼마나 작은지에 따라 진행 시간을 많이 줄일 수 있습니다.

조건자 표현식은 Spark SQL가 지원한 부울 확장일 수 있습니다. Spark SQL 쿼리의 WHERE에 넣는 어떤 것도 만족할 수 있습니다. 예를 들어 조건자 표현식 `pushDownPredicate = "(year=='2017' and month=='04')"`는 year가 2017이고 month가 04인 Data Catalog의 파티션만 로드합니다. 자세한 내용은 [Apache Spark SQL 설명서](#), 특히 [Scala SQL 함수 참조](#)를 참조하십시오.

카탈로그 파티션 조건자를 사용한 서버 측 필터링

`push_down_predicate` 옵션은 카탈로그의 모든 파티션을 나열한 후 해당 파티션에 대한 Amazon S3의 파일을 나열하기 전에 적용됩니다. 테이블에 대한 파티션이 많은 경우 카탈로그 파티션 목록에 여전히 추가 시간 오버헤드가 발생할 수 있습니다. 이 오버헤드를 해결하기 위해 AWS Glue Data Catalog의 [파티션 인덱스](#)를 사용하는 `catalogPartitionPredicate` 옵션과 함께 서버 측 파티션 정리를 사용할 수 있습니다. 따라서 한 테이블에 수백만 개의 파티션이 있는 경우 파티션 필터링이 훨씬 빨라집니다. 카탈로그 파티션 인덱스에서 아직 지원되지 않는 조건자 구문이 `catalogPartitionPredicate`에 필요한 경우 `additional_options`에서 `push_down_predicate`와 `catalogPartitionPredicate`를 함께 사용할 수 있습니다.

Python:

```
dynamic_frame = glueContext.create_dynamic_frame.from_catalog(
    database=dbname,
    table_name=tablename,
    transformation_ctx="datasource0",
    push_down_predicate="day>=10 and customer_id like '10%",
    additional_options={"catalogPartitionPredicate":"year='2021' and month='06'"}
)
```

Scala:

```
val dynamicFrame = glueContext.getCatalogSource(
    database = dbname,
    tableName = tablename,
    transformationContext = "datasource0",
    pushDownPredicate="day>=10 and customer_id like '10%",
    additionalOptions = JsonOptions("""{
        "catalogPartitionPredicate": "year='2021' and month='06'"}""")
).getDynamicFrame()
```

Note

`push_down_predicate`와 `catalogPartitionPredicate`는 다른 구문을 사용합니다. 전자는 Spark SQL 표준 구문을 사용하고 후자는 JSQL 구문 분석기를 사용합니다.

파티션 작성

기본적으로 `DynamicFrame`은 작성될 때 파티션이 되지 않습니다. 출력 파일 모두는 지정된 출력 경로의 상위 수준에 작성됩니다. 최근까지 파티션에 `DynamicFrame`을 작성하는 유일한 방법은 작성 전에 `DynamicFrame`을 Spark SQL `DataFrame`으로 전환하는 방법이었습니다.

그러나 `DynamicFrame`은 싱크를 생성할 때 `partitionKeys` 옵션을 사용하여 키의 시퀀스를 사용한 본래 파티셔닝을 지원합니다. 예를 들어, 다음 Python 코드는 Parquet 포맷의 Amazon S3로 데이터 집합을 유형 필드로 분할된 디렉터리에 작성합니다. 그런 다음, Amazon Athena와 같은 다른 시스템을 사용하여 이런 파티션을 진행할 수 있습니다.

```
glue_context.write_dynamic_frame.from_options(
    frame = projectedEvents,
    connection_type = "s3",
    connection_options = {"path": "$outpath", "partitionKeys": ["type"]},
    format = "parquet")
```

입력 파일을 더 큰 그룹에서 읽기

테이블에 속성을 설정하여 AWS Glue ETL 작업을 활성화하여 속성을 Amazon S3 데이터 스토어에서 읽을 때 파일을 모읍니다. 이 속성은 각 ETL 태스크이 입력 파일 그룹을 단일 인메모리 파티션으로 읽을 수 있도록 만듭니다. 이렇게 하는 것은 Amazon S3 데이터 스토어에 많은 수의 작은 파일을 있을 때 유용합니다. 특정 속성을 설정하면 AWS Glue에 지시를 내려 Amazon S3 데이터 파티션 내 파일을 모으고 읽을 그룹의 크기를 설정합니다. `create_dynamic_frame.from_options` 메서드로 Amazon S3 데이터 스토어에서 읽을 때 이러한 옵션을 설정할 수도 있습니다.

테이블 파일을 모으려면 테이블 구조의 파라미터 필드에서 키 값 페어를 설정합니다. JSON 표기법을 사용하여 테이블 파라미터 필드 값을 설정합니다. 테이블 속성을 편집하는 방법에 대한 더 자세한 정보는 [테이블 세부 정보 보기 및 편집](#)를 참조하십시오.

이 방법을 사용하여 Amazon S3 데이터 스토어로 Data Catalog의 테이블을 모을 수 있도록 합니다.

groupFiles

groupFiles를 inPartition으로 설정하면 Amazon S3 데이터 파티션 내 파일 그룹화가 가능합니다. 입력 파일이 50,000개 이상일 경우 AWS Glue가 그룹화를 자동으로 활성화합니다.

```
'groupFiles': 'inPartition'
```

groupSize

groupSize를 바이트 단위로 그룹의 대상 크기로 설정합니다. groupSize 속성은 조건부입니다. 제공되지 않으면 총 ETL 작업 수 및 인메모리 파티션을 줄이면서 AWS Glue는 크기를 계산하여 클러스터의 모든 CPU 코어를 사용합니다.

예를 들어, 다음은 그룹 크기를 1MB로 설정합니다.

```
'groupSize': '1048576'
```

계산 결과로 groupsize를 설정해야 합니다. 예: $1024 * 1024 = 1048576$.

recurse

recurse를 True로 설정하여 paths를 경로 어레이로 지정할 때 모든 하위 디렉터리의 파일을 재귀적으로 읽을 수 있습니다. 다음 예와 같이 paths(이가) Amazon S3의 객체 키 배열이거나 입력 형식이 parquet/orc인 경우 재귀를 설정하지 않아도 됩니다.

```
'recurse': True
```

create_dynamic_frame.from_options 메서드를 사용하여 Amazon S3에서 직접 읽는 경우 이러한 연결 옵션을 추가합니다. 예를 들어, 다음과 같이 파일을 1MB 그룹으로 그룹화하려는 시도합니다.

```
df = glueContext.create_dynamic_frame.from_options("s3", {'paths': ["s3://s3path/"],
  'recurse': True, 'groupFiles': 'inPartition', 'groupSize': '1048576'}, format="json")
```


Note

groupFiles는 csv, ion, grokLog, json, xml 등의 데이터 포맷에서 생성된 DynamicFrames에서 지원됩니다. avro, parquet, orc에서는 이 옵션이 지원되지 않습니다.

Amazon S3용 Amazon VPC 엔드포인트

보안상의 이유로 많은 AWS 고객은 Amazon Virtual Private Cloud 환경(Amazon VPC)에서 애플리케이션을 실행합니다. Amazon VPC를 통해 Amazon EC2 인스턴스를 퍼블릭 인터넷을 비롯한 다른 네트워크와 논리적으로 분리된 Virtual Private Cloud로 시작할 수 있습니다. Amazon VPC를 사용하면 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 및 보안 설정을 통제할 수 있습니다.

Note

2013년 12월 4일 이후 AWS 계정을 생성했다면 각 AWS 리전에 기본 VPC가 갖춰져 있습니다. 별도의 구성 단계 없이 기본 VPC를 사용하여 즉시 시작할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [기본 VPC 및 서브넷](#)을 참조하세요.

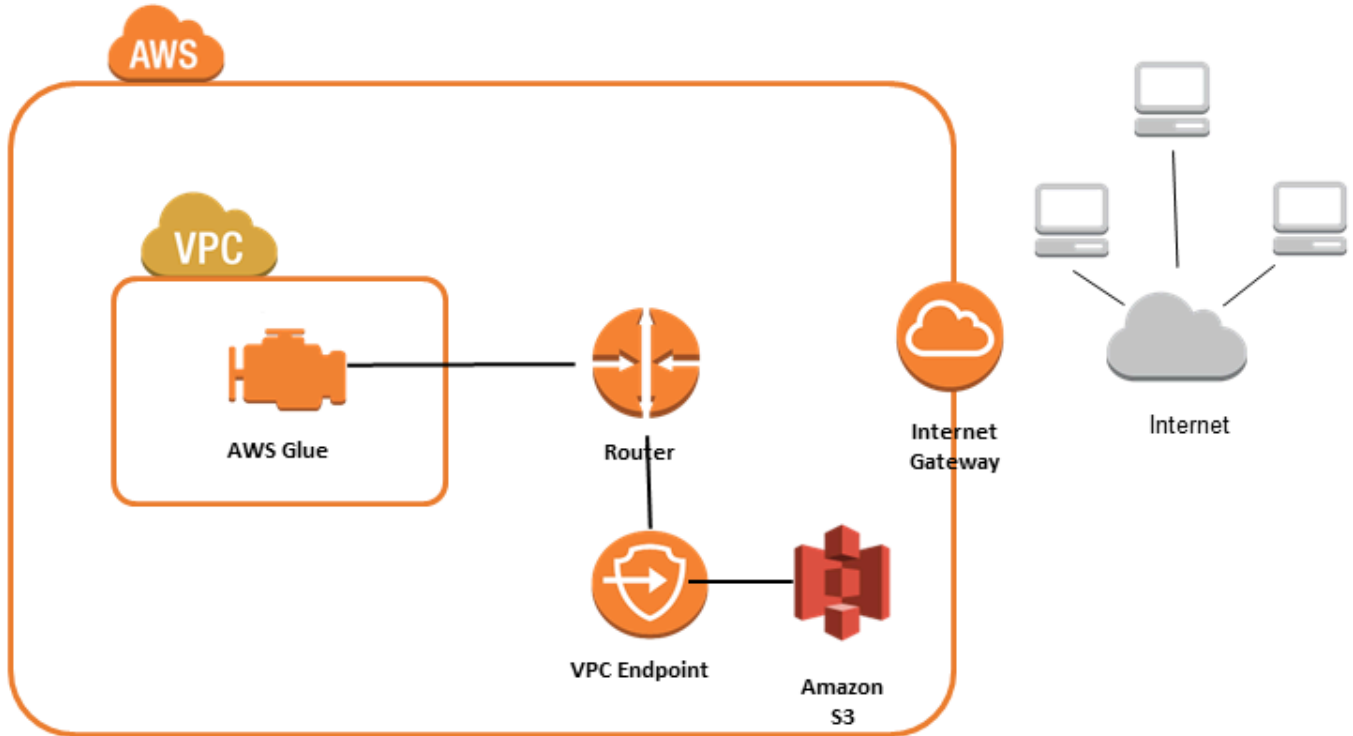
많은 고객이 퍼블릭 인터넷을 통해 데이터를 주고받는 것에 대한 합법적인 개인 정보 보호 및 보안 문제를 우려합니다. 가상 사설 네트워크(VPN)를 사용하여 모든 Amazon S3 네트워크 트래픽을 자체 기업 네트워크 인프라를 통해 라우팅함으로써 이러한 문제를 해결할 수 있습니다. 그러나 이러한 접근 방식은 대역폭 및 가용성 문제를 일으킬 수 있습니다.

Amazon S3에 대한 VPC 엔드포인트는 이러한 문제를 완화할 수 있습니다. Amazon S3에 대한 VPC 엔드포인트를 사용하여 AWS Glue가 퍼블릭 인터넷에 노출되지 않고도 프라이빗 IP 주소를 사용하여 Amazon S3에 액세스할 수 있습니다. AWS Glue에 퍼블릭 IP 주소를 지정할 필요가 없으며 VPC에서 인터넷 게이트웨이, NAT 디바이스 또는 가상 프라이빗 게이트웨이가 필요 없습니다. 엔드포인트 정책을 사용하여 Amazon S3에 대한 액세스를 제어합니다. VPC와 AWS 서비스 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

Amazon S3에 대한 VPC 엔드포인트를 생성하면 리전 내의 Amazon S3 엔드포인트(예: s3.us-west-2.amazonaws.com)에 대한 요청이 Amazon 네트워크 내의 프라이빗 Amazon S3 엔드포인트로 라우팅됩니다. VPC의 Amazon EC2 인스턴스에서 실행되는 애플리케이션을 수정할 필요가 없습니다. 엔드포인트 이름은 동일하게 남아있지만 Amazon S3에 대한 경로는 전적으로 Amazon 네트워크에 유지되며 퍼블릭 인터넷에 액세스하지 않습니다.

VPC 엔드포인트에 대한 자세한 내용은 Amazon VPC 사용 설명서의 [VPC 엔드포인트](#)를 참조하세요.

다음 다이어그램은 AWS Glue가 Amazon S3에 액세스하기 위해 VPC 엔드포인트를 사용하는 방법을 보여줍니다.



Amazon S3에 대한 액세스를 설정하려면

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 엔드포인트를 선택합니다.
3. [엔드포인트 생성(Create Endpoint)]을 선택하고 단계에 따라 게이트웨이 유형의 Amazon S3 VPC 엔드포인트를 생성합니다.

Amazon DocumentDB 연결

AWS Glue for Spark를 사용하여 Amazon DocumentDB 데이터베이스의 테이블에서 읽고 쓸 수 있습니다. AWS Glue 연결을 통해 AWS Secrets Manager에 저장된 보안 인증 정보를 사용하여 Amazon DocumentDB에 연결할 수 있습니다.

Amazon DocumentDB에 대한 자세한 내용은 [Amazon DocumentDB 설명서](#)를 참조하세요.

Note

Amazon DocumentDB Elastic Clusters는 현재 AWS Glue 커넥터를 사용하는 경우 지원되지 않습니다. Elastic Clusters에 대한 자세한 내용은 [Using Amazon DocumentDB elastic clusters](#)를 참조하세요.

Amazon DocumentDB 컬렉션 읽기 및 쓰기

Note

Amazon DocumentDB에 연결하는 ETL 작업을 생성할 때 Connections 작업 속성에 대해 Amazon DocumentDB가 실행 중인 Virtual Private Cloud(VPC)를 지정하는 연결 객체를 지정해야 합니다. 연결 객체의 경우 연결 유형은 JDBC여야 하며 JDBC URL은 `mongo://<DocumentDB_host>:27017`이어야 합니다.

Note

이 코드 샘플은 AWS Glue 3.0용으로 개발되었습니다. AWS Glue 4.0으로 마이그레이션하려면 [the section called "MongoDB"](#) 섹션을 참조하세요. `uri` 파라미터가 변경되었습니다.

Note

Amazon DocumentDB를 사용하는 경우 작성된 문서에서 `_id`를 지정하는 경우와 같은 특정 상황에서는 `retryWrites`를 `false`로 설정해야 합니다. 자세한 내용은 Amazon DocumentDB 설명서에서 [Functional Differences with MongoDB](#)를 참조하세요.

다음 Python 스크립트는 Amazon DocumentDB를 읽고 쓰기 위한 연결 유형 및 연결 옵션을 사용하는 방법을 보여줍니다.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'], args)

output_path = "s3://some_bucket/output/" + str(time.time()) + "/"
documentdb_uri = "mongodb://<mongo-instanced-ip-address>:27017"
documentdb_write_uri = "mongodb://<mongo-instanced-ip-address>:27017"

read_docdb_options = {
    "uri": documentdb_uri,
    "database": "test",
    "collection": "coll",
    "username": "username",
    "password": "1234567890",
    "ssl": "true",
    "ssl.domain_match": "false",
    "partitioner": "MongoSamplePartitioner",
    "partitionerOptions.partitionSizeMB": "10",
    "partitionerOptions.partitionKey": "_id"
}

write_documentdb_options = {
    "retryWrites": "false",
    "uri": documentdb_write_uri,
    "database": "test",
    "collection": "coll",
```

```

    "username": "username",
    "password": "pwd"
  }

# Get DynamicFrame from DocumentDB
dynamic_frame2 =
  glueContext.create_dynamic_frame.from_options(connection_type="documentdb",

  connection_options=read_docdb_options)

# Write DynamicFrame to MongoDB and DocumentDB
glueContext.write_dynamic_frame.from_options(dynamic_frame2,
  connection_type="documentdb",

  connection_options=write_documentdb_options)

job.commit()

```

다음 Scala 스크립트는 Amazon DocumentDB를 읽고 쓰기 위한 연결 유형 및 연결 옵션을 사용하는 방법을 보여줍니다.

```

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  val DOC_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  val DOC_WRITE_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  lazy val documentDBJsonOption = jsonOptions(DOC_URI)
  lazy val writeDocumentDBJsonOption = jsonOptions(DOC_WRITE_URI)
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Get DynamicFrame from DocumentDB

```

```

    val resultFrame2: DynamicFrame = glueContext.getSource("documentdb",
documentDBJsonOption).getDynamicFrame()

    // Write DynamicFrame to DocumentDB
    glueContext.getSink("documentdb", writeJsonOption).writeDynamicFrame(resultFrame2)

    Job.commit()
}

private def jsonOptions(uri: String): JsonOptions = {
  new JsonOptions(
    s""""{"uri": "${uri}",
      |"database":"test",
      |"collection":"coll",
      |"username": "username",
      |"password": "pwd",
      |"ssl":"true",
      |"ssl.domain_match":"false",
      |"partitioner": "MongoSamplePartitioner",
      |"partitionerOptions.partitionSizeMB": "10",
      |"partitionerOptions.partitionKey": "_id"}"""".stripMargin)
  }
}
}

```

Amazon DocumentDB 연결 옵션 참조

Amazon DocumentDB에 대한 연결을 지정합니다(MongoDB와 호환).

연결 옵션은 소스 연결 및 싱크 연결에 따라 다릅니다.

소스로서의 "connectionType": "Documentdb"

소스로서의 "connectionType": "documentdb"에는 다음 연결 옵션을 사용합니다.

- "uri": (필수 사항) 읽을 소스 Amazon DocumentDB 호스트로, mongodb://<host>:<port> 포맷입니다.
- "database": (필수) 읽을 소스 Amazon DocumentDB 데이터베이스입니다.
- "collection": (필수) 읽을 소스 Amazon DocumentDB 컬렉션입니다.
- "username": (필수) Amazon DocumentDB 사용자 이름입니다.
- "password": (필수) Amazon DocumentDB 암호입니다.

- "ssl": (SSL을 사용하는 경우 필수) 연결에서 SSL을 사용하는 경우 "true" 값과 함께 이 옵션을 포함해야 합니다.
- "ssl.domain_match": (SSL을 사용하는 경우 필수) 연결에서 SSL을 사용하는 경우 "false" 값과 함께 이 옵션을 포함해야 합니다.
- "batchSize": (선택 사항) 내부 배치의 커서 내에서 사용되는 배치당 반환할 문서 수입니다.
- "partitioner": (선택 사항) Amazon DocumentDB에서 입력 데이터를 읽는 파티셔너의 클래스 이름입니다. 커넥터는 다음 파티셔너를 제공합니다.
 - MongoDefaultPartitioner(기본값)(AWS Glue 4.0에서는 지원되지 않음)
 - MongoSamplePartitioner(AWS Glue 4.0에서는 지원되지 않음)
 - MongoShardedPartitioner
 - MongoSplitVectorPartitioner
 - MongoPaginateByCountPartitioner
 - MongoPaginateBySizePartitioner(AWS Glue 4.0에서는 지원되지 않음)
- "partitionerOptions": (선택 사항) 지정된 파티셔너에 대한 옵션입니다. 각 파티셔너에 대해 다음 옵션이 지원됩니다.
 - MongoSamplePartitioner: partitionKey, partitionSizeMB, samplesPerPartition
 - MongoShardedPartitioner: shardkey
 - MongoSplitVectorPartitioner: partitionKey, partitionSizeMB
 - MongoPaginateByCountPartitioner: partitionKey, numberOfPartitions
 - MongoPaginateBySizePartitioner: partitionKey, partitionSizeMB

이러한 옵션에 대한 자세한 내용은 MongoDB 설명서의 [파티셔너 구성](#)을 참조하십시오.

싱크로서의 "connectionType": "Documentdb"

싱크로서의 "connectionType": "documentdb"에는 다음 연결 옵션을 사용합니다.

- "uri": (필수 사항) 쓸 대상 Amazon DocumentDB 호스트로, mongodb://<host>:<port> 포맷입니다.
- "database": (필수) 쓸 대상 Amazon DocumentDB 데이터베이스입니다.
- "collection": (필수) 쓸 대상 소스 Amazon DocumentDB 컬렉션입니다.
- "username": (필수) Amazon DocumentDB 사용자 이름입니다.
- "password": (필수) Amazon DocumentDB 암호입니다.

- "extendedBsonTypes": (선택 사항) true이면 데이터를 Amazon DocumentDB에 쓸 때 확장 BSON 유형을 허용합니다. 기본값은 true입니다.
- "replaceDocument": (선택 사항) true이면 `_id` 필드가 포함된 데이터 세트를 저장할 때 전체 문서를 대체합니다. false이면 데이터 세트의 필드와 일치하는 문서의 필드만 업데이트됩니다. 기본값은 true입니다.
- "maxBatchSize": (선택 사항) 데이터를 저장할 때 대량 작업의 최대 배치 크기입니다. 기본값은 512입니다.
- "retryWrites": (선택 사항): AWS Glue에서 네트워크 오류가 발생하는 경우 특정 쓰기 작업을 한 번 자동으로 재시도합니다.

OpenSearch Service 연결

AWS Glue for Spark를 사용하여 AWS Glue 4.0 이상 버전에서 OpenSearch Service의 테이블에서 읽고 쓸 수 있습니다. OpenSearch 쿼리를 사용하여 OpenSearch Service에서 읽을 내용을 정의할 수 있습니다. AWS Glue 연결을 통해 AWS Secrets Manager에 저장된 HTTP 기본 인증 보안 인증 정보를 사용하여 OpenSearch Service에 연결합니다. 이 기능은 OpenSearch Service 서버리스와 호환되지 않습니다.

Amazon OpenSearch Service에 대한 자세한 정보는 [Amazon OpenSearch Service 설명서](#)를 참조하십시오.

OpenSearch Service 연결 구성

AWS Glue에서 OpenSearch Service에 연결하려면 OpenSearch Service 보안 인증 정보를 생성하여 암호에 저장한 다음 해당 AWS Secrets Manager 암호를 OpenSearch Service AWS Glue 연결에 연결해야 합니다.

사전 조건:

- Amazon OpenSearch Service 설명서의 지침에 따라 읽으려는 도메인 엔드포인트, *aosEndpoint* 및 포트, *aosPort*를 식별하거나 리소스를 생성하십시오. 도메인 생성에 관해 자세한 내용을 알아보려면 Amazon OpenSearch Service 설명서의 [Amazon OpenSearch Service 도메인 생성 및 관리](#)를 참조하십시오.

Amazon OpenSearch Service 도메인 엔드포인트는 기본 형식이 `https://search-domainName-unstructuredIdContent.region.es.amazonaws.com`입니다. 도메인 엔드포인트에 관해 자세한 내용을 알아보려면 Amazon OpenSearch Service 설명서의 [Amazon OpenSearch Service 도메인 생성 및 관리](#)를 참조하십시오.

도메인의 HTTP 기본 보안 인증 정보, *aosUser*와 *aosPassword*를 확인하거나 생성하십시오.

OpenSearch Service에 대한 연결을 구성하는 방법:

1. AWS Secrets Manager에서 OpenSearch 보안 인증을 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *aosUser*이 포함된 키 `opensearch.net.http.auth.user`에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *aosPassword*이 포함된 키 `opensearch.net.http.auth.pass`에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때는 OpenSearch Service를 선택합니다.
 - 도메인 엔드포인트를 선택할 때는 *aosEndpoint*를 제공하십시오.
 - 포트를 선택할 때는 *aosPort*를 제공하십시오.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.

AWS Glue OpenSearch Service 연결을 생성한 후에는 AWS Glue 작업을 실행하기 전에 다음 단계를 수행해야 합니다.

- AWS Glue 작업과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여합니다.
- AWS Glue 작업 구성에서 추가 네트워크 연결로 *connectionName*을 제공합니다.

OpenSearch Service 인덱스에서 읽기

사전 조건:

- 읽으려는 OpenSearch Service 인덱스, *aosIndex*.
- 인증 및 네트워크 위치 정보를 제공하도록 구성된 AWS Glue OpenSearch Service 연결입니다. 이 정보를 얻으려면 앞 절차인 OpenSearch Service에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

이 예제는 Amazon OpenSearch Service에서 인덱스를 읽습니다. `pushdown` 파라미터를 제공해야 합니다.

예:

```
opensearch_read = glueContext.create_dynamic_frame.from_options(
    connection_type="opensearch",
    connection_options={
        "connectionName": "connectionName",
        "opensearch.resource": "aosIndex",
        "pushdown": "true",
    }
)
```

DynamicFrame에 반환되는 결과를 필터링하는 쿼리 문자열을 제공할 수도 있습니다. `opensearch.query`를 구성해야 합니다.

`opensearch.query`는 URL 쿼리 매개 변수 문자열 *queryString* 또는 쿼리 DSL JSON 객체 *queryObject*를 사용할 수 있습니다. 쿼리 DSL에 대한 자세한 내용은 OpenSearch 설명서의 [쿼리 DSL](#)을 참조하십시오. URL 쿼리 매개 변수 문자열을 제공하려면 정규화된 URL에서처럼 쿼리 앞에 `?`를 추가하십시오. 쿼리 DSL 객체를 제공하려면 JSON 객체를 제공하기 전에 문자열을 이스케이프 처리해야 합니다.

예:

```
queryObject = "{ \"query\": { \"multi_match\": { \"query\": \"Sample\", \"fields\": [ \"sample\" ] } } }"
queryString = "?q=queryString"

opensearch_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="opensearch",
    connection_options={
        "connectionName": "connectionName",
        "opensearch.resource": "aosIndex",
        "opensearch.query": queryString,
        "pushdown": "true",
    }
)
```

특정 구문을 벗어나 쿼리를 작성하는 방법에 대한 자세한 내용은 OpenSearch 설명서의 [쿼리 문자열 구문](#)을 참조하십시오.

배열 형식 데이터가 포함된 OpenSearch 컬렉션에서 읽을 때는 `opensearch.read.field.as.array.include` 파라미터를 사용하여 메서드 직접 호출에서 어떤 필드가 배열 형식인지 지정해야 합니다.

예를 들어 다음 문서를 읽을 때 `genre` 및 `actor` 배열 필드가 나타납니다.

```
{
  "_index": "movies",
  "_id": "2",
  "_version": 1,
  "_seq_no": 0,
  "_primary_term": 1,
  "found": true,
  "_source": {
    "director": "Frankenheimer, John",
    "genre": [
      "Drama",
      "Mystery",
      "Thriller",
      "Crime"
    ],
    "year": 1962,
    "actor": [
      "Lansbury, Angela",
      "Sinatra, Frank",
      "Leigh, Janet",
      "Harvey, Laurence",
      "Silva, Henry",
      "Frees, Paul",
      "Gregory, James",
      "Bissell, Whit",
      "McGiver, John",
      "Parrish, Leslie",
      "Edwards, James",
      "Flowers, Bess",
      "Dhiegh, Khigh",
      "Payne, Julie",
      "Kleeb, Helen",
      "Gray, Joe",
      "Nalder, Reggie",
      "Stevens, Bert",
      "Masters, Michael",
      "Lowell, Tom"
    ]
  }
}
```

```
    ],
    "title": "The Manchurian Candidate"
  }
}
```

이 경우 메서드 직접 호출에 해당 필드 이름을 포함하면 됩니다. 예:

```
"opensearch.read.field.as.array.include": "genre,actor"
```

배열 필드가 문서 구조 안에 중첩되어 있는 경우 점 표기법("genre,actor,foo.bar.baz")을 사용하여 참조하세요. 이는 포함된 문서(bar)를 포함하는 포함된 문서(foo)를 통해 소스 문서에 포함된 배열(baz)을 지정합니다.

OpenSearch Service 테이블에 쓰기

이 예제에서는 기존 DynamicFrame, *dynamicFrame*의 정보를 OpenSearch Service에 기록합니다. 인덱스에 이미 정보가 있는 경우 AWS Glue는 DynamicFrame의 데이터를 추가합니다. pushdown 파라미터를 제공해야 합니다.

사전 조건:

- 쓰려는 OpenSearch Service 테이블. 테이블에 대한 식별 정보가 필요합니다. 이 *tableName*이라고 부르겠습니다.
- 인증 및 네트워크 위치 정보를 제공하도록 구성된 AWS Glue OpenSearch Service 연결입니다. 이 정보를 얻으려면 앞 절차인 OpenSearch Service에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="opensearch",
    connection_options={
        "connectionName": "connectionName",
        "opensearch.resource": "aosIndex",
    },
)
```

OpenSearch Service 연결 옵션 참조

- `connectionName` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 연결 방법에 인증 및 네트워크 위치 정보를 제공하도록 구성된 AWS Glue OpenSearch Service 연결의 이름입니다.
- `opensearch.resource` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 유효한 값: OpenSearch 인덱스 이름. 연결 방법이 상호 작용하는 인덱스의 이름입니다.
- `opensearch.query` — 읽기에 사용됩니다. 유효한 값: 문자열이 JSON으로 이스케이프되거나 이 문자열이 ?로 시작되는 경우 URL의 검색 부분. 읽을 때 검색해야 하는 항목을 필터링하는 OpenSearch 쿼리입니다. 이 매개변수 사용에 대한 자세한 내용은 이전 섹션 [the section called “OpenSearch Service에서 읽기”](#)를 참조하십시오.
- `pushdown` - 다음과 같은 경우 필수입니다. 읽기에 사용됩니다. 유효 값: 부울. 데이터베이스가 관련 문서만 반환하도록 Spark가 읽기 쿼리를 OpenSearch에 전달하도록 지시합니다.
- `opensearch.read.field.as.array.include` - 배열 유형 데이터를 읽을 때 필요합니다. 읽기에 사용됩니다. 유효 값: 쉼표로 구분된 필드 이름 목록. OpenSearch 문서에서 배열로 읽을 필드를 지정합니다. 이 매개변수 사용에 대한 자세한 내용은 이전 섹션 [the section called “OpenSearch Service에서 읽기”](#)를 참조하십시오.

Redshift 연결

AWS Glue for Spark를 사용하여 Amazon Redshift 데이터베이스의 테이블에서 읽고 쓸 수 있습니다. Amazon Redshift 데이터베이스에 연결할 때 AWS Glue는 Amazon COPY Redshift SQL 및 명령을 사용하여 Amazon S3를 통해 데이터를 이동하여 처리량을 극대화합니다. UNLOAD AWS Glue 4.0 이상에서는 [Apache Spark용 Amazon Redshift 통합](#)을 사용하여 이전 버전을 통해 연결할 때 사용할 수 있는 기능 외에도 Amazon Redshift에만 적용되는 최적화 및 기능을 사용하여 읽고 쓸 수 있습니다.

Amazon Redshift 사용자가 서버리스 데이터 통합 및 ETL을 위해 AWS Glue로 이전하는 것을 그 어느 때보다 쉽게 AWS Glue로 마이그레이션할 수 있도록 지원하는 방법에 대해 알아보십시오.

Redshift 연결 구성

AWS Glue에서 Amazon Redshift 클러스터를 사용하려면 다음과 같은 몇 가지 사전 요구 사항이 필요합니다.

- 데이터베이스에서 읽고 쓸 때 임시 스토리지로 사용하는 Amazon S3 디렉터리.
- Amazon Redshift 클러스터, AWS Glue 작업 및 Amazon S3 디렉터리 간의 통신을 지원하는 Amazon VPC입니다.
- AWS Glue 작업 및 Amazon Redshift 클러스터에 대한 적절한 IAM 권한.

IAM 역할 구성

Amazon Redshift 클러스터의 역할 설정

AWS Glue 작업과 통합하려면 Amazon Redshift 클러스터가 Amazon S3에 읽고 쓸 수 있어야 합니다. 이를 허용하기 위해 연결하려는 Amazon Redshift 클러스터에 IAM 역할을 연결할 수 있습니다. 역할에는 Amazon S3 임시 디렉터리에서의 읽기 및 쓰기를 허용하는 정책이 있어야 합니다. 역할에는 AssumeRole을 위해 redshift.amazonaws.com 서비스를 허용하는 신뢰 관계가 있어야 합니다.

IAM 역할을 Amazon Redshift에 연결하려면

- 필수 조건: 파일의 임시 저장에 사용되는 Amazon S3 버킷 또는 디렉터리.
- Amazon Redshift 클러스터에 필요한 Amazon S3 권한을 식별합니다. Amazon Redshift 클러스터에서 데이터를 이동할 때 AWS Glue 작업은 Amazon Redshift에 대해 COPY 및 UNLOAD 명령문을 실행합니다. Amazon Redshift에서 작업을 통해 테이블을 수정하는 경우 AWS Glue는 라이브러리 생성 명령문도 발행합니다. Amazon Redshift에서 이러한 명령문을 실행하는 데 필요한 특정 Amazon S3 권한에 대한 자세한 내용은 Amazon Redshift 설명서인 [Amazon Redshift](#): 다른 리소스에 액세스할 수 있는 권한을 참조하십시오. AWS
- IAM 콘솔에서 필요한 권한이 포함된 IAM 정책을 생성합니다. IAM 정책 생성에 대한 자세한 내용은 [IAM 정책 생성](#)을 참조하세요.
- IAM 콘솔에서 Amazon Redshift가 역할을 수입할 수 있는 역할 및 신뢰 관계를 생성합니다. 서비스에 [대한 역할을 생성하려면 IAM 설명서의 지침을 따르십시오 \(콘솔\) AWS](#).
 - AWS 서비스 사용 사례를 선택하라는 메시지가 표시되면 “Redshift - 사용자 지정 가능”을 선택합니다.
 - 정책을 연결하라는 메시지가 표시되면 이전에 정의한 정책을 선택합니다.

Note

Amazon Redshift의 역할을 구성하는 방법에 대한 자세한 내용은 Amazon Redshift 설명서에서 [Amazon Redshift가 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여](#)를 참조하십시오.

- Amazon Redshift 콘솔에서 Amazon Redshift 클러스터에 역할을 연결합니다. [Amazon Redshift](#) 설명서의 지침을 수행합니다.

Amazon Redshift 콘솔에서 강조 표시된 옵션을 선택하여 다음 설정을 구성합니다.

Amazon Redshift > Clusters > flight-2016

flight-2016

General information

Cluster identifier flight-2016	Status ✔ Available
Cluster namespace [REDACTED]	Date created [REDACTED]
Cluster configuration Production	Storage used 0.25% (0.41 of 160)
	Multi-AZ No

Actions ▲

Manage cluster

- Resize
- Reboot
- Pause
- Delete
- Defer maintenance
- Modify publicly accessible setting

Backup and disaster recovery

- Restore table
- Create snapshot
- Configure cross-region snapshot
- Relocate

Permissions

- Manage IAM roles
- Change admin user password
- Manage tags

Endpoint

JDBC URL

ODBC URL

Driver={Amazon Redshift (...

Cluster performance

Query monitoring

S...

rties

i Note

기본적으로 AWS Glue 작업은 작업을 실행하도록 지정한 역할을 사용하여 생성된 Amazon Redshift 임시 자격 증명을 전달합니다. 이 보안 인증은 사용하지 않는 것이 좋습니다. 보안을 위해 이러한 보안 인증은 1시간 후에 만료됩니다.

AWS Glue 작업의 역할 설정

AWS Glue 작업에는 Amazon S3 버킷에 액세스하기 위한 역할이 필요합니다. Amazon Redshift 클러스터에는 IAM 권한이 필요하지 않습니다. 액세스는 데이터베이스 보안 인증 및 Amazon VPC의 연결을 통해 제어됩니다.

Amazon VPC 설정

Amazon Redshift 데이터 스토어에 대한 액세스를 설정하려면

1. AWS Management Console [로그인하고 https://console.aws.amazon.com/redshiftv2/](https://console.aws.amazon.com/redshiftv2/) 에서 [Amazon Redshift 콘솔을 엽니다.](#)
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. AWS Glue로부터 액세스하고자 하는 클러스터 이름을 선택합니다.
4. Cluster Properties(클러스터 속성) 섹션에서 VPC security groups(VPC 보안 그룹)에서 보안 그룹을 선택하여 사용할 AWS Glue를 허용합니다. 미래 참조를 위해 선택하는 보안 그룹의 이름을 기록합니다. 보안 그룹을 선택하면 Amazon EC2 콘솔 [보안 그룹(Security Groups)] 목록을 엽니다.
5. 보안 그룹을 선택하여 [Inbound(인바운드)] 탭으로 수정하고 탐색합니다.
6. 자기 참조 규칙을 추가하여 AWS Glue 구성 요소를 허용하여 통신합니다. 특히, [Type(유형)] All TCP 의 규칙을 추가하고 확인합니다. [Protocol(프로토콜)]는 TCP이고 [Port Range(포트 범위)]는 모든 포트를 포함하고 포트의 [Source(원본)]은 [Group ID(그룹 ID)]이라는 동일한 보안 그룹입니다.

인바운드 규칙은 다음과 비슷하게 보입니다.

유형	프로토콜	포트 범위	소스
모든 TCP	TCP	0~65535	database-security-group

예:

7. 아웃바운드 트래픽 규칙도 추가합니다. 예를 들어 아웃바운드 트래픽을 모든 포트에 엽니다.

유형	프로토콜	포트 범위	대상
모든 트래픽	ALL	ALL	0.0.0.0/0

또는 자기 참조 규칙을 생성합니다. 여기서 Type(유형) 은 All TCP이고 Protocol(프로토콜)은 TCP이고 Port Range(포트 범위)는 모든 포트를 포함하고 포트의 Destination(대상)은 Group ID(그룹 ID)와 동일한 보안 그룹 이름입니다. Amazon S3 VPC 엔드포인트를 사용할 경우 Amazon S3

액세스에 대한 HTTPS 규칙도 추가합니다. VPC에서 *Amazon S3 VPC ##### #### ##### ## ## ##### prefix-list-id s3-#* 필요합니다.

예:

유형	프로토콜	포트 범위	대상
모든 TCP	TCP	0~65535	<i>security-group</i>
HTTPS	TCP	443	<i>s3- prefix-li st-id</i>

AWS Glue 설정

Amazon VPC 연결 정보를 제공하는 AWS Glue 데이터 카탈로그 연결을 생성해야 합니다.

콘솔에서 AWS Glue에 대한 Amazon Redshift 아마존 VPC 연결을 구성하려면

1. [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 데이터 카탈로그 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때는 Amazon Redshift를 선택합니다.
 - Redshift 클러스터를 선택할 때는 이름을 기준으로 클러스터를 선택합니다.
 - 클러스터의 Amazon Redshift 사용자에 대한 기본 연결 정보를 제공합니다.
 - Amazon VPC 설정이 자동으로 구성됩니다.

Note

AWS SDK를 통해 Amazon Redshift 연결을 생성할 때는 Amazon VPC에 대한 `PhysicalConnectionRequirements`를 수동으로 제공해야 합니다.

2. AWS Glue 작업 구성에서 *## ### ##* 네트워크 연결로 제공하십시오.

예제: Amazon Redshift 테이블에서 읽기

Amazon Redshift 클러스터와 Amazon Redshift 서버리스 환경에서 읽을 수 있습니다.

필수 조건: 읽으려는 Amazon Redshift 테이블. 이전 [the section called “Redshift 구성”](#) 섹션의 단계를 수행하면 임시 디렉터리에 대한 Amazon S3 URI인 `temp-s3-dir#` IAM 역할인, (계정 내) 이 있어야 합니다. `rs-role-namerole-account-id`

Using the Data Catalog

추가 필수 조건: 읽으려는 Amazon Redshift 테이블의 데이터 카탈로그 데이터베이스 및 테이블. 데이터 카탈로그에 대한 자세한 내용은 [데이터 검색 및 카탈로그 작성](#) 섹션을 참조하세요. Amazon Redshift 테이블에 대한 항목을 생성한 후에는 `redshift-dc-database-nameredshift-table-name`를 사용하여 연결을 식별할 수 있습니다.

구성: 함수 옵션에서 `database` 및 `table_name` 파라미터로 데이터 카탈로그 테이블을 식별합니다. Amazon S3 임시 디렉터리를 `redshift_tmp_dir`로 식별합니다. 또한 `additional_options` 파라미터의 `aws_iam_role` 키를 `rs-role-name` 사용하여 제공할 것입니다.

```
glueContext.create_dynamic_frame.from_catalog(
    database = "redshift-dc-database-name",
    table_name = "redshift-table-name",
    redshift_tmp_dir = args["temp-s3-dir"],
    additional_options = {"aws_iam_role": "arn:aws:iam::role-account-id:role/rs-role-name"})
```

Connecting directly

추가 전제 조건: Amazon Redshift 테이블의 이름이 필요합니다 (. `redshift-table-name` 해당 테이블을 저장하는 Amazon Redshift 클러스터에 대한 JDBC 연결 정보가 필요합니다. `###, ## redshift-database-name, ### ## # ##### ## ## ### ##### ###`.

Amazon Redshift 클러스터를 사용할 때 Amazon Redshift 콘솔에서 연결 정보를 검색할 수 있습니다. Amazon Redshift Serverless를 사용하는 경우 Amazon Redshift 설명서에서 [Amazon Redshift Serverless에 연결](#)을 참조하세요.

구성: 함수 옵션에서 `url`, `dbtable`, `user` 및 `password`로 연결 파라미터를 식별합니다. Amazon S3 임시 디렉터리를 `redshift_tmp_dir`로 식별합니다. `from_options`를 사용할 때 `aws_iam_role`을 사용하여 IAM 역할을 지정할 수 있습니다. 구문은 데이터 카탈로그를 통한 연결과 비슷하지만 `connection_options` 맵에 파라미터를 입력합니다.

비밀번호를 AWS Glue 스크립트로 하드코딩하는 것은 좋지 않습니다. Python용 SDK (Boto3) 를 사용하여 암호를 저장하고 스크립트에서 검색하는 것을 고려해 보세요. AWS Secrets Manager

```
my_conn_options = {
    "url": "jdbc:redshift://host:port/redshift-database-name",
    "dbtable": "redshift-table-name",
    "user": "username",
    "password": "password",
    "redshiftTmpDir": args["temp-s3-dir"],
    "aws_iam_role": "arn:aws:iam::account id:role/rs-role-name"
}

df = glueContext.create_dynamic_frame.from_options("redshift", my_conn_options)
```

예제: Amazon Redshift 테이블에 쓰기

Amazon Redshift 클러스터와 Amazon Redshift 서버리스 환경에 쓸 수 있습니다.

사전 요구 사항: Amazon Redshift 클러스터를 사용하고 이전 [the section called “Redshift 구성”](#) 섹션의 단계를 따르세요. 그러면 임시 디렉터리에 대한 Amazon S3 URI인 `temp-s3-dir` 및 IAM 역할 (,) 이 있어야 합니다. `rs-role-name``role-account-id` 또한 데이터베이스에 해당 콘텐츠를 쓰려는 `DynamicFrame`도 필요합니다.

Using the Data Catalog

추가 필수 조건: 쓰려는 Amazon Redshift 클러스터 및 테이블의 데이터 카탈로그 데이터베이스. 데이터 카탈로그에 대한 자세한 내용은 [데이터 검색 및 카탈로그 작성](#) 섹션을 참조하세요. 위의 연결과 대상 테이블을 식별하게 됩니다. `redshift-dc-database-name``redshift-table-name`

구성: 함수 옵션에서 `database` 파라미터로 데이터 카탈로그 데이터베이스를 식별하고 `table_name`에서 테이블을 제공합니다. Amazon S3 임시 디렉터리를 `redshift_tmp_dir`로 식별합니다. `additional_options` 파라미터의 `aws_iam_role` 키를 `rs-role-name` 사용하여 제공할 수도 있습니다.

```
glueContext.write_dynamic_frame.from_catalog(
    frame = input dynamic frame,
    database = "redshift-dc-database-name",
```

```

table_name = "redshift-table-name",
redshift_tmp_dir = args["temp-s3-dir"],
additional_options = {"aws_iam_role": "arn:aws:iam::account-id:role/rs-role-name"})

```

Connecting through a AWS Glue connection

`write_dynamic_frame.from_options` 메서드를 사용하여 Amazon Redshift에 직접 연결할 수 있습니다. 하지만 스크립트에 연결 세부 정보를 직접 삽입하는 대신 `from_jdbc_conf` 메서드를 사용하여 데이터 카탈로그 연결에 저장된 연결 세부 정보를 참조할 수 있습니다. 데이터베이스의 데이터 카탈로그 테이블을 크롤링하거나 생성하지 않고도 이 작업을 수행할 수 있습니다. 데이터 카탈로그 연결에 대한 자세한 내용은 [데이터에 연결](#) 섹션을 참조하세요.

추가 필수 조건: 읽으려는 Amazon Redshift 테이블, 사용자 데이터베이스에 대한 데이터 카탈로그 연결.

구성: 데이터 카탈로그 연결을 식별할 수 `dc-connection-name` 있습니다. Amazon Redshift 데이터베이스 및 테이블을 및 로 `redshift-table-name` 식별할 수 있습니다. `redshift-database-name` `catalog_connection`으로 데이터 카탈로그 연결 정보를 제공하고 `dbtable` 및 `database`로 Amazon Redshift 정보를 제공합니다. 구문은 데이터 카탈로그를 통한 연결과 비슷하지만 `connection_options` 맵에 파라미터를 입력합니다.

```

my_conn_options = {
    "dbtable": "redshift-table-name",
    "database": "redshift-database-name",
    "aws_iam_role": "arn:aws:iam::role-account-id:role/rs-role-name"
}

glueContext.write_dynamic_frame.from_jdbc_conf(
    frame = input_dynamic_frame,
    catalog_connection = "dc-connection-name",
    connection_options = my_conn_options,
    redshift_tmp_dir = args["temp-s3-dir"])

```

Amazon Redshift 연결 옵션 참조

모든 AWS Glue JDBC 연결에 사용되는 기본 연결 옵션은 모든 JDBC url 유형과 user 유사하고 password 동일한 정보를 설정합니다. 표준 JDBC 클러스터 파라미터에 대한 자세한 내용은 [the section called "JDBC 연결 파라미터"](#) 섹션을 참조하세요.

Amazon Redshift 연결 유형에는 다음과 같은 몇 가지 추가 연결 옵션이 있습니다.

- "redshiftTmpDir": (필수) 데이터베이스 외부에서 복사할 때 임시 데이터를 스테이징할 수 있는 Amazon S3 경로.
- "aws_iam_role": (선택 사항) IAM 역할의 ARN. AWS Glue 작업은 이 역할을 Amazon Redshift 클러스터로 전달하여 작업의 지침을 완료하는 데 필요한 권한을 클러스터에 부여합니다.

AWS Glue 4.0+에서 사용할 수 있는 추가 연결 옵션

AWS Glue 연결 옵션을 통해 새 Amazon Redshift 커넥터의 옵션을 전달할 수도 있습니다. 지원되는 커넥터 옵션의 전체 목록은 [Apache Spark용 Amazon Redshift 통합](#)의 Spark SQL 파라미터 섹션을 참조하세요.

편의를 위해 다음과 같은 새로운 특정 옵션을 다시 설명합니다.

명칭	필수	기본값	설명
autopushdown	아니요	TRUE	SQL 작업에 대한 Spark 논리 계획을 캡처하고 분석하여 조건자 및 쿼리 푸시다운을 적용합니다. 작업은 SQL 쿼리로 변환된 다음 Amazon Redshift에서 실행되어 성능이 향상됩니다.
autopushdown.s3_result_cache	아니요	FALSE	동일한 쿼리가 동일한 Spark 세션에서 다시 실행될 필요가 없도록 SQL 쿼리를 캐시하여 Amazon S3 경

명칭	필수	기본값	설명
			로 매핑 데이터를 메모리에서 언로드합니다. autopushdown 이 활성화된 경우에만 지원됩니다.
unload_s3_format	아니요	PARQUET	PARQUET - 쿼리 결과를 Parquet 형식으로 언로드합니다. TEXT - 쿼리 결과를 파이프로 구분된 텍스트 형식으로 언로드합니다.
sse_kms_key	아니요	N/A	기본 암호화 대신 UNLOAD 작업 중 암호화에 사용할 AWS SSE-KMS 키입니다. AWS

명칭	필수	기본값	설명
extracopyoptions	아니요	N/A	<p>데이터를 로드할 때 Amazon Redshift COPY 명령에 추가할 추가 옵션 목록(예: TRUNCATECOLUMNS 또는 MAXERROR n)입니다. 다른 옵션은 COPY: 선택적 파라미터를 참조하세요.</p> <p>이러한 옵션은 COPY 명령 끝에 추가되므로 명령 끝에 의미가 있는 옵션만 사용할 수 있습니다. 여기서는 가능한 대부분의 사용 사례를 다루어야 합니다.</p>
csvnullstring(실험용)	아니요	NULL	<p>CSV tempformat 을 사용할 때 null에 쓸 문자열 값입니다. 실제 데이터에 나타나지 않는 값이어야 합니다.</p>

이러한 새 파라미터는 다음과 같은 방법으로 사용할 수 있습니다.

성능 개선을 위한 새로운 옵션

새 커넥터에는 몇 가지 새로운 성능 개선 옵션이 도입되었습니다.

- autopushdown: 기본적으로 활성화됩니다.
- autopushdown.s3_result_cache: 기본적으로 비활성화됩니다.
- unload_s3_format: 기본적으로 PARQUET입니다.

이러한 옵션을 사용하는 방법에 대한 자세한 내용은 [Apache Spark용 Amazon Redshift 통합](#)을 참조하세요. 캐시된 결과에 오래된 정보가 포함될 수 있으므로 읽기 작업과 쓰기 작업이 혼합된 경우 `autopushdown.s3_result_cache`를 켜지 않는 것이 좋습니다. 성능을 개선하고 스토리지 비용을 절감하기 위해 UNLOAD 명령의 `unload_s3_format` 옵션은 기본적으로 PARQUET로 설정됩니다. UNLOAD 명령 기본 동작을 사용하려면 옵션을 TEXT로 재설정합니다.

읽기를 위한 새 암호화 옵션

Amazon Redshift 테이블에서 데이터를 읽을 때 AWS Glue가 사용하는 임시 폴더의 데이터는 기본적으로 SSE-S3 암호화를 사용하여 암호화됩니다. 고객 관리 키 from AWS Key Management Service (AWS KMS) 를 사용하여 데이터를 암호화하려면 버전 3.0의 기존 설정 옵션 대신 KSMKey가 [키 ID](#)의 출처인 ("`sse_kms_key`" # `kmsKey`) 위치를 설정할 수 있습니다. AWS KMS("`extraunloadoptions`" # `s"ENCRYPTED KMS_KEY_ID '$kmsKey'"`) AWS Glue

```
datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = "database-name",
    table_name = "table-name",
    redshift_tmp_dir = args["TempDir"],
    additional_options = {"sse_kms_key": "<KMS_KEY_ID>"},
    transformation_ctx = "datasource0"
)
```

IAM 기반 JDBC URL 지원

새 커넥터는 IAM 기반 JDBC URL을 지원하므로 사용자 및 암호 또는 보안 암호를 전달할 필요가 없습니다. IAM 기반 JDBC URL을 사용하는 커넥터에서는 작업 런타임 역할을 사용하여 Amazon Redshift 데이터 소스에 액세스합니다.

1단계: 다음과 같은 최소 필수 정책을 AWS Glue 작업 런타임 역할에 연결합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "redshift:GetClusterCredentials",
      "Resource": [
        "arn:aws:redshift:<region>:<account>:dbgroup:<cluster name>/*",
        "arn:aws:redshift:*:<account>:dbuser:/*/*",
        "arn:aws:redshift:<region>:<account>:dbname:<cluster name>/<database name>"
      ]
    }
  ]
}
```



```

    ]
  },
  {
    "Sid": "VisualEditor1",
    "Effect": "Allow",
    "Action": "redshift:DescribeClusters",
    "Resource": "*"
  }
]
}

```

2단계: IAM 기반 JDBC URL을 다음과 같이 사용합니다. 연결하려는 Amazon Redshift 사용자 이름을 사용하여 새 옵션 DbUser를 지정합니다.

```

conn_options = {
  // IAM-based JDBC URL
  "url": "jdbc:redshift:iam://<cluster name>:<region>/<database name>",
  "dbtable": dbtable,
  "redshiftTmpDir": redshiftTmpDir,
  "aws_iam_role": aws_iam_role,
  "DbUser": "<Redshift User name>" // required for IAM-based JDBC URL
}

redshift_write = glueContext.write_dynamic_frame.from_options(
  frame=dyf,
  connection_type="redshift",
  connection_options=conn_options
)

redshift_read = glueContext.create_dynamic_frame.from_options(
  connection_type="redshift",
  connection_options=conn_options
)

```

Note

DynamicFrame은 현재 GlueContext.create_dynamic_frame.from_options 워크플로에 DbUser가 있는 IAM 기반 JDBC URL만 지원합니다.

AWS Glue 3.0 버전에서 4.0 버전으로 마이그레이션

AWS Glue 4.0에서 ETL 작업은 다양한 옵션 및 구성을 갖춘 새로운 Amazon Redshift Spark 커넥터와 새로운 JDBC 드라이버에 액세스할 수 있습니다. 새로운 Amazon Redshift 커넥터 및 드라이버는 성능을 염두에 두고 작성되었으며 데이터의 트랜잭션 일관성을 유지합니다. 이러한 제품은 Amazon Redshift 설명서에 문서화되어 있습니다. 자세한 내용은 다음을 참조하세요.

- [Apache Spark용 Amazon Redshift 통합](#)
- [Amazon Redshift JDBC 드라이버, 버전 2.1](#)

테이블/열 이름 및 식별자 제한

새로운 Amazon Redshift Spark 커넥터 및 드라이버에서는 Redshift 테이블 이름에 대한 요구 사항이 더 제한적입니다. 자세한 내용은 Amazon Redshift 테이블 이름을 정의하기 위한 [이름 및 식별자](#)를 참조하세요. 규칙을 따르지 않는 테이블 이름 및 특정 문자(예: 공백)에서는 작업 북마크 워크플로가 작동하지 않을 수 있습니다.

[이름 및 식별자](#) 규칙을 따르지 않는 이름을 가진 레거시 테이블이 있고 북마크에 문제가 있는 경우(이전 Amazon Redshift 테이블 데이터를 재처리하는 작업) 테이블 이름을 변경하는 것이 좋습니다. 자세한 내용은 [ALTER TABLE 예](#)를 참조하세요.

DataFrame의 기본 tempformat 변경

AWS Glue 3.0 버전 Spark 커넥터에서는 Amazon Redshift Redshift에 쓰는 동안 기본적으로 tempformat이 CSV로 지정됩니다. 일관성을 유지하기 위해 AWS Glue 3.0 버전에서는 DynamicFrame의 기본 tempformat로 CSV를 계속 사용합니다. 이전에 Amazon Redshift Spark 커넥터에서 Spark Dataframe API를 직접 사용한 적이 있다면 DataframeReader 및 Writer 옵션에서 tempformat을 CSV로 명시적으로 설정할 수 있습니다. 그렇지 않으면 새 Spark 커넥터에서 tempformat이 기본적으로 AVRO로 설정됩니다.

동작 변경: Amazon Redshift 데이터 형식 REAL을 DOUBLE 대신 Spark 데이터 형식 FLOAT에 매핑합니다.

AWS Glue 3.0 버전에서는 Amazon Redshift REAL이 Spark DOUBLE 유형으로 변환됩니다. Amazon Redshift REAL 유형과 Spark FLOAT 유형 간에 상호 변환되도록 새로운 Amazon Redshift Spark 커넥터의 동작을 업데이트했습니다. 레거시 사용 사례에서 Amazon RedshiftREAL 유형을 SparkDOUBLE 유형에 계속 매핑하려는 경우 다음 해결 방법을 사용할 수 있습니다.

- DynamicFrame의 경우 DynamicFrame.ApplyMapping을 사용하여 Float 유형을 Double 유형에 매핑합니다. Dataframe의 경우 cast를 사용해야 합니다.

코드 예제:

```
dyf_cast = dyf.apply_mapping([('a', 'long', 'a', 'long'), ('b', 'float', 'b', 'double')])
```

Kafka 연결

Kafka 클러스터 또는 Amazon Managed Streaming for Apache Kafka 클러스터에 대한 연결을 지정합니다.

데이터 카탈로그 테이블에 저장된 정보를 사용하거나 데이터 스트림에 직접 액세스할 수 있는 정보를 제공하여 Kafka Data Streams에서 읽고 쓸 수 있습니다. Kafka에서 DataFrame Spark로 정보를 읽은 다음 Glue로 변환할 수 있습니다. AWS DynamicFrame JSON 형식으로 DynamicFrames Kafka에 글을 쓸 수 있습니다. 데이터 스트림에 직접 액세스하는 경우 이러한 옵션을 사용하여 데이터 스트림에 액세스하는 방법에 대한 정보를 제공합니다.

getCatalogSource 또는 create_data_frame_from_catalog를 사용하여 Kafka 스트리밍 소스에서 레코드를 소비하거나 getCatalogSink 또는 write_dynamic_frame_from_catalog를 사용하여 Kafka에 레코드를 쓰는 경우 작업에는 데이터 카탈로그 데이터베이스와 테이블 이름 정보가 있으며 이를 사용하여 Kafka 스트리밍 소스에서 읽기 위한 몇 가지 기본 파라미터를 얻을 수 있습니다. getSource, getCatalogSink, getSourceWithFormat, getSinkWithFormat, createDataFrameFromOptions, create_data_frame_from_options 또는 write_dynamic_frame_from_catalog를 사용하는 경우 여기에 설명된 연결 옵션을 통해 이러한 기본 파라미터를 지정해야 합니다.

GlueContext 클래스에 지정된 메서드에 대해 다음 인수를 사용하여 Kafka에 대한 연결 옵션을 지정할 수 있습니다.

- Scala
 - connectionOptions: getSource, createDataFrameFromOptions, getSink와(과) 함께 사용
 - additionalOptions: getCatalogSource, getCatalogSink와 함께 사용
 - options: getSourceWithFormat, getSinkWithFormat와 함께 사용
- Python
 - connection_options: create_data_frame_from_options, write_dynamic_frame_from_options와 함께 사용
 - additional_options: create_data_frame_from_catalog, write_dynamic_frame_from_catalog와 함께 사용

- options: getSource, getSink와 함께 사용

스트리밍 ETL 작업에 대한 참고 및 제한 사항은 [the section called “스트리밍 ETL 참고 사항 및 제한 사항”](#) 섹션을 참조하세요.

Kafka 구성

인터넷을 통해 사용할 수 있는 AWS Kafka 스트림에 연결하기 위한 전제 조건은 없습니다.

AWS Glue Kafka 연결을 생성하여 연결 자격 증명을 관리할 수 있습니다. 자세한 정보는 [the section called “Kafka 데이터 스트림에 대한 연결 생성”](#)을 참조하세요. AWS Glue 작업 구성에서 `ConnectionName#` 추가 네트워크 연결로 제공한 다음, 메서드 호출에서 `ConnectionName# #### #` 제공하십시오. `connectionName`

경우에 따라 추가 필수 조건을 구성해야 합니다.

- IAM 인증과 함께 Amazon Managed Streaming for Apache Kafka를 사용하는 경우 적절한 IAM 구성 이 필요합니다.
- Amazon VPC 내에서 Amazon Managed Streaming for Apache Kafka를 사용하는 경우 적절한 Amazon VPC 구성 이 필요합니다. Amazon VPC 연결 정보를 제공하는 AWS Glue 연결을 생성해야 합니다. AWS Glue 연결을 추가 네트워크 연결로 포함하려면 작업 구성 이 필요합니다.

스트리밍 ETL 작업 필수 조건에 대한 자세한 내용은 [the section called “스트리밍 ETL 작업”](#) 섹션을 참조하세요.

예제: Kafka 스트림에서 읽기

[the section called “forEachBatch”](#)과(와) 함께 사용합니다.

Kafka 스트리밍 소스의 예:

```
kafka_options =
  { "connectionName": "ConfluentKafka",
    "topicName": "kafka-auth-topic",
    "startingOffsets": "earliest",
    "inferSchema": "true",
    "classification": "json"
  }
```

```
data_frame_datasource0 =
  glueContext.create_data_frame.from_options(connection_type="kafka",
  connection_options=kafka_options)
```

예: Kafka 스트림에 쓰기

Kafka에 쓰기의 예제:

getSink 메서드를 사용한 예제:

```
data_frame_datasource0 =
  glueContext.getSink(
    connectionType="kafka",
    connectionOptions={
      JsonObject(""{
        "connectionName": "ConfluentKafka",
        "classification": "json",
        "topic": "kafka-auth-topic",
        "typeOfData": "kafka"}
      "}")},
    transformationContext="dataframe_ApacheKafka_node1711729173428")
  .getDataFrame()
```

write_dynamic_frame.from_options 메서드를 사용한 예제:

```
kafka_options =
  { "connectionName": "ConfluentKafka",
    "topicName": "kafka-auth-topic",
    "classification": "json"
  }
data_frame_datasource0 =
  glueContext.write_dynamic_frame.from_options(connection_type="kafka",
  connection_options=kafka_options)
```

Kafka 연결 옵션 참조

읽을 때는 "connectionType": "kafka"를 사용한 다음 연결 옵션을 사용합니다.

- "bootstrap.servers" (필수) 부트스트랩 서버 URL의 목록입니다(예: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094). 이 옵션은 API 호출에 지정하거나 데이터 카탈로그의 테이블 메타데이터에 정의해야 합니다.

- "security.protocol" (필수) 브로커와 통신하는 데 사용되는 프로토콜입니다. 가능한 값은 "SSL" 또는 "PLAINTEXT"입니다.
- "topicName": (필수) 쉽표로 구분된 구독할 주제의 목록입니다. "topicName", "assign" 또는 "subscribePattern" 중에서 하나만 지정해야 합니다.
- "assign": (필수) 사용할 특정 TopicPartitions을 지정하는 JSON 문자열입니다. "topicName", "assign" 또는 "subscribePattern" 중에서 하나만 지정해야 합니다.

예: '{"topicA":[0,1],"topicB":[2,4]}'

- "subscribePattern": (필수) 구독할 주제 목록을 식별하는 Java 정규식 문자열입니다. "topicName", "assign" 또는 "subscribePattern" 중에서 하나만 지정해야 합니다.

예: 'topic.*'

- "classification"(필수) 레코드의 데이터에서 사용하는 파일 형식입니다. 데이터 카탈로그를 통해 제공되지 않는 한 필수입니다.
- "delimiter"(선택 사항) classification이 CSV일 때 사용되는 값 구분 기호입니다. 기본값은 ','입니다.
- "startingOffsets": (선택 사항) 데이터를 읽을 Kafka 주제의 시작 위치입니다. 가능한 값은 "earliest" 또는 "latest"입니다. 기본값은 "latest"입니다.
- "startingTimestamp": (선택 사항, AWS Glue 버전 4.0 이상에서만 지원됨) Kafka 주제에서 데이터를 읽을 레코드의 타임스탬프입니다. 가능한 값은 yyyy-mm-ddTHH:MM:SSZ 패턴에서 UTC 형식의 타임스탬프 문자열입니다(여기서, Z는 UTC 시간대 오프셋(+/-)임, 예: '2023-04-04T08:00:00-04:00').

참고: AWS Glue 스트리밍 스크립트의 연결 옵션 목록에는 'StartingOffsets' 또는 'StartingTimestamp' 중 하나만 표시될 수 있습니다. 이 두 속성을 모두 포함하면 작업이 실패합니다.

- "endingOffsets": (선택 사항) 일괄 쿼리가 종료되는 엔드포인트입니다. 가능한 값은 "latest" 또는 각 TopicPartition의 끝 오프셋을 지정하는 JSON 문자열입니다.

JSON 문자열의 경우 포맷은 {"topicA":{"0":23,"1":-1},"topicB":{"0":-1}}입니다. 오프셋으로 값 -1은 "latest"를 나타냅니다.

- "pollTimeoutMs": (선택 사항) Spark 작업 실행기의 Kafka에서 데이터를 폴링하는 시간 제한(밀리초)입니다. 기본 값은 512입니다.
- "numRetries": (선택 사항) Kafka 오프셋 가져오기에 실패하기 전에 재시도할 횟수입니다. 기본 값은 3입니다.

- "retryIntervalMs": (선택 사항) Kafka 오프셋을 가져오기를 다시 시도하기 전에 대기하는 시간 (밀리초)입니다. 기본 값은 10입니다.
- "maxOffsetsPerTrigger": (선택 사항) 트리거 간격당 처리되는 최대 오프셋 수에 대한 속도 제한입니다. 지정된 총 오프셋 수는 서로 다른 볼륨의 topicPartitions에 비례하여 분할됩니다. 기본값은 null입니다. 즉, 소비자가 알려진 최신 오프셋까지 모든 오프셋을 읽습니다.
- "minPartitions": (선택 사항) Kafka에서 읽을 원하는 최소 파티션 수입니다. 기본값은 null이며 이는 Spark 파티션의 수가 Kafka 파티션의 수와 동일함을 의미합니다.
- "includeHeaders": (선택 사항) Kafka 헤더를 포함할지 여부입니다. 옵션이 "true"로 설정되면 데이터 출력에는 유형이 Array[Struct(key: String, value: String)]인 "glue_streaming_kafka_headers"라는 추가 열이 포함됩니다. 기본값은 "false"입니다. 이 옵션은 AWS Glue 버전 3.0 이상에서만 사용할 수 있습니다.
- "schema": (InferSchema가 false로 설정된 경우 필수) 페이로드를 처리하는 데 사용할 스키마입니다. 분류가 avro인 경우 제공된 스키마는 Avro 스키마 형식이어야 합니다. 분류가 avro가 아닌 경우 제공된 스키마는 DDL 스키마 형식이어야 합니다.

다음은 스키마의 예입니다.

Example in DDL schema format

```
'column1' INT, 'column2' STRING , 'column3' FLOAT
```

Example in Avro schema format

```
{
  "type": "array",
  "items":
  {
    "type": "record",
    "name": "test",
    "fields":
    [
      {
        "name": "_id",
        "type": "string"
      },
      {
        "name": "index",
        "type":
        [
          "int",
```

```

        "string",
        "float"
    ]
}
]
}
}

```

- "inferSchema": (선택 사항) 기본값은 'false'입니다. 'true'로 설정하면 스키마가 런타임 시 foreachbatch 내의 페이로드에서 감지됩니다.
- "avroSchema": (더 이상 사용되지 않음) Avro 형식을 사용할 때 Avro 데이터의 스키마를 지정하는 데 사용되는 파라미터입니다. 이 파라미터는 이제 사용 중단되었습니다. schema 파라미터를 사용합니다.
- "addRecordTimestamp": (선택 사항) 이 옵션이 'true'로 설정되면 데이터 출력에는 이름이 '_src_timestamp'라는 추가 열이 포함됩니다. 이 열은 주제에서 해당 레코드를 수신한 시간을 나타냅니다. 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.
- "emitConsumerLagMetrics": (선택 사항) 옵션을 'true'로 설정하면 각 배치에 대해 주제가 수신한 가장 오래된 레코드와 해당 레코드가 도착한 시간 사이의 기간 동안 메트릭이 내보내집니다. AWS Glue CloudWatch 메트릭의 이름은 "glue.driver.streaming.maxConsumerLagInMs"입니다. 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.

쓸 때는 "connectionType": "kafka"를 사용한 다음 연결 옵션을 사용합니다.

- "connectionName"(필수) Kafka 클러스터에 연결하는 데 사용되는 AWS Glue 연결 이름 (Kafka 소스와 유사).
- "topic"(필수) 주제 열이 존재하는 경우 주제 구성 옵션이 설정되지 않은 한 Kafka에 지정된 행을 쓸 때 해당 값이 주제로 사용됩니다. 즉, topic 구성 옵션이 주제 열을 재정의합니다.
- "partition"(선택 사항) 유효한 파티션 번호가 지정되면 레코드를 보낼 때 해당 partition이 사용됩니다.

파티션을 지정하지 않았지만 key가 있는 경우 키의 해시를 사용하여 파티션이 선택됩니다.

key와 partition이 모두 존재하지 않으면 파티션에 최소 batch.size 바이트가 생성될 때 변경 사항을 고정 파티셔닝하여 파티션이 선택됩니다.

- "key"(선택 사항) partition이 null인 경우 파티셔닝에 사용됩니다.
- "classification"(선택 사항) 레코드의 데이터에서 사용하는 파일 형식입니다. JSON, CSV, Avro만 지원합니다.

Avro 형식을 사용하면 직렬화할 사용자 지정 avroSchema를 제공할 수 있지만 역직렬화를 위해서는 소스에서도 이를 제공해야 한다는 점에 유의하세요. 그렇지 않으면 기본적으로 직렬화에 Apache를 사용합니다. AvroSchema

또한 필요에 따라 [Kafka 프로듀서 구성 파라미터](#)를 업데이트하여 Kafka 싱크를 미세 튜닝할 수 있습니다. 연결 옵션에는 허용 목록이 없으며 모든 키값 쌍은 싱크에 그대로 유지됩니다.

그러나 적용되지 않는 일부 거부 옵션의 목록이 있습니다. 자세한 내용은 [Kafka 특정 구성](#)을 참조하세요.

Azure Cosmos DB 연결

AWS Glue for Spark를 이용하여 Azure Cosmos DB의 기존 컨테이너에서 읽고 쓸 수 있습니다. 이 때 AWS Glue 4.0 이상에서 NoSQL API를 이용합니다. SQL 쿼리를 사용하여 Azure Cosmos DB에서 읽을 내용을 정의할 수 있습니다. AWS Glue 연결을 통해 AWS Secrets Manager에 저장된 Azure Cosmos DB 키를 사용하여 Azure Cosmos DB에 연결합니다.

NoSQL용 Azure Cosmos DB에 대한 자세한 내용은 [Azure 설명서](#)를 참조하십시오.

Azure Cosmos DB 연결 구성

AWS Glue에서 Azure Cosmos DB에 연결하려면 Azure Cosmos DB 키를 만들어 AWS Secrets Manager 암호에 저장한 다음 해당 암호를 Azure Cosmos DB AWS Glue 연결에 연결해야 합니다.

사전 조건:

- Azure에서는 AWS Glue cosmosKey에서 사용할 Azure Cosmos DB 키를 식별하거나 생성해야 합니다. 자세한 내용은 Azure 설명서의 [Azure Cosmos DB의 데이터에 대한 보안 액세스](#)를 참조하십시오.

Azure Cosmos DB에 대한 연결 구성 방법:

1. AWS Secrets Manager에서 Azure Cosmos DB 키를 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키값 페어를 선택하면 값 *cosmosKey*가 포함된 키 spark.cosmos.accountKey에 대한 페어를 생성합니다.

2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때는 Azure Cosmos DB를 선택합니다.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.

AWS Glue Azure Cosmos DB 연결을 생성한 후에는 AWS Glue 작업을 실행하기 전에 다음 단계를 수행해야 합니다.

- AWS Glue 작업과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여합니다.
- AWS Glue 작업 구성에서 추가 네트워크 연결로 *connectionName*을 제공합니다.

NoSQL 컨테이너용 Azure Cosmos DB에서 읽기

사전 조건:

- 읽으려는 NoSQL 컨테이너용 Azure Cosmos 데이터베이스. 컨테이너의 식별 정보가 필요합니다.

NoSQL용 Azure Cosmos 컨테이너는 해당 데이터베이스 및 컨테이너로 식별됩니다.

NoSQL API용 Azure Cosmos에 연결할 때 데이터베이스, *cosmosDBName*, 및 컨테이너, *cosmosContainerName*, 이름을 제공해야 합니다.

- 인증 및 네트워크 위치 정보를 제공하도록 구성된 AWS Glue Azure Cosmos DB 연결입니다. 이 정보를 얻으려면 앞 절차인 Azure Cosmos DB에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
azurecosmos_read = glueContext.create_dynamic_frame.from_options(
    connection_type="azurecosmos",
    connection_options={
        "connectionName": connectionName,
        "spark.cosmos.database": cosmosDBName,
        "spark.cosmos.container": cosmosContainerName,
    }
)
```

SELECT SQL 쿼리에 반환되는 결과를 필터링하는 쿼리를 제공할 수도 있습니다. *query*을 구성해야 합니다.

예:

```
azurecosmos_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="azurecosmos",
    connection_options={
        "connectionName": "connectionName",
        "spark.cosmos.database": cosmosDBName,
        "spark.cosmos.container": cosmosContainerName,
        "spark.cosmos.read.customQuery": "query"
    }
)
```

NoSQL 컨테이너용 Azure Cosmos DB에 쓰기

이 예제에서는 기존 DynamicFrame, *dynamicFrame*의 정보를 Azure Cosmos DB에 씁니다. 컨테이너에 이미 정보가 있는 경우 AWS Glue는 DynamicFrame의 데이터를 추가합니다. 컨테이너의 정보가 작성한 정보와 다른 스키마를 사용하는 경우 오류가 발생합니다.

사전 조건:

- 쓰려는 Azure Cosmos DB 테이블. 컨테이너의 식별 정보가 필요합니다. 연결 방법을 호출하기 전에 컨테이너를 만들어야 합니다.

NoSQL용 Azure Cosmos 컨테이너는 해당 데이터베이스 및 컨테이너로 식별됩니다.

NoSQL API용 Azure Cosmos에 연결할 때 데이터베이스, *cosmosDBName*, 및 컨테이너, *cosmosContainerName*, 이름을 제공해야 합니다.

- 인증 및 네트워크 위치 정보를 제공하도록 구성된 AWS Glue Azure Cosmos DB 연결입니다. 이 정보를 얻으려면 앞 절차인 Azure Cosmos DB에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
azurecosmos_write = glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="azurecosmos",
    connection_options={
        "connectionName": connectionName,
        "spark.cosmos.database": cosmosDBName,
        "spark.cosmos.container": cosmosContainerName
    }
)
```

Azure Cosmos DB 연결 옵션 참조

- `connectionName` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 연결 방법에 인증 및 네트워크 위치 정보를 제공하도록 구성된 AWS Glue Azure Cosmos DB 연결의 이름입니다.
- `spark.cosmos.database` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 유효한 값: 데이터베이스 이름. NoSQL용 Azure Cosmos DB 데이터베이스 이름입니다.
- `spark.cosmos.container` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 유효한 값: 컨테이너 이름. NoSQL용 Azure Cosmos DB 컨테이너 이름입니다.
- `spark.cosmos.read.customQuery` — 읽기에 사용됩니다. 유효한 값: SQL 쿼리를 선택합니다. 읽을 문서를 선택하기 위한 사용자 지정 쿼리.

Azure SQL 연결

AWS Glue for Spark를 사용하여 AWS Glue 4.0 이상 버전에서 Azure SQL Managed Instances의 테이블에서 읽고 쓸 수 있습니다. SQL 쿼리를 사용하여 Azure SQL에서 읽을 내용을 정의할 수 있습니다. AWS Glue 연결을 통해 AWS Secrets Manager에 저장된 사용자 이름 및 암호 보안 인증 정보를 사용하여 Azure SQL에 연결할 수 있습니다.

Azure SQL에 대한 자세한 내용은 [Azure SQL 설명서](#)를 참조하십시오.

Azure SQL 연결 구성

AWS Glue에서 Azure SQL에 연결하려면 Azure SQL 보안 인증 정보를 만들어 AWS Secrets Manager 암호에 저장한 다음 해당 암호를 Azure SQL AWS Glue 연결에 연결해야 합니다.

Azure SQL에 대한 연결을 구성하는 방법:

1. AWS Secrets Manager에서 Azure SQL 보안 인증을 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, `secretName`을 유지합니다.
 - 키/값 페어를 선택하면 값 `azuresqlUsername`이 포함된 키 `user`에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 `azuresqlPassword`가 포함된 키 `password`에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 `connectionName`을 유지합니다.
 - 연결 유형을 선택할 때 Azure SQL를 선택합니다.

- Azure SQL URL을 제공할 때는 JDBC 엔드포인트 URL을 제공하십시오.

목록은

`jdbc:sqlserver://databaseServerName:databasePort;databaseName=azuresqlDBName`
형식이어야 합니다.

AWS Glue에는 다음과 같은 URL 속성이 필요합니다.

- `databaseName` - 연결할 Azure SQL의 기본 데이터베이스입니다.

Azure SQL 관리형 인스턴스용 JDBC URL에 대한 자세한 내용은 [Microsoft 설명서](#)를 참조하십시오.

- AWS 보안 암호를 선택할 때 `secretName`을 입력합니다.

AWS Glue Azure SQL 연결을 생성한 후에는 AWS Glue 작업을 실행하기 전에 다음 단계를 수행해야 합니다.

- AWS Glue 작업과 연결된 IAM 역할에 `secretName`을 읽을 수 있는 권한을 부여합니다.
- AWS Glue 작업 구성에서 추가 네트워크 연결로 `connectionName`을 제공합니다.

Azure SQL 테이블에서 읽는 중

사전 조건:

- 읽으려는 Azure SQL 테이블. 테이블, `databaseName`, `tableIdentifier`에 대한 식별 정보가 필요합니다.

Azure SQL 테이블은 데이터베이스, 스키마 및 테이블 이름으로 식별됩니다. Azure SQL에 연결할 때 데이터베이스 이름과 테이블 이름을 제공해야 합니다. 스키마가 기본값인 "public"이 아닌 경우에도 스키마를 제공해야 합니다. 데이터베이스는 `connectionName`의 URL 속성을, `dbtable`을 통해 스키마 및 테이블 이름을 제공 받습니다.

- 인증 정보를 제공하도록 구성된 AWS Glue Azure SQL 연결입니다. 인증 정보를 구성하려면 앞 절 차인 Azure SQL에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 `connectionName`이 필요합니다.

예:

```
azuresql_read_table = glueContext.create_dynamic_frame.from_options(
```

```

connection_type="azuresql",
connection_options={
    "connectionName": "connectionName",
    "dbtable": "tableIdentifier"
}
)

```

SELECT SQL 쿼리에 반환되는 결과를 필터링하는 쿼리를 제공할 수도 있습니다. query를 구성해야 합니다.

예:

```

azuresql_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="azuresql",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)

```

Azure SQL 테이블에 쓰기

이 예제에서는 기존 DynamicFrame, *dynamicFrame*의 정보를 Azure SQL에 씁니다. 테이블에 이미 정보가 있는 경우 AWS Glue는 DynamicFrame의 데이터를 추가합니다.

사전 조건:

- 쓰려는 Azure SQL 테이블. 테이블, *databaseName*, *tableIdentifier*에 대한 식별 정보가 필요합니다.

Azure SQL 테이블은 데이터베이스, 스키마 및 테이블 이름으로 식별됩니다. Azure SQL에 연결할 때 데이터베이스 이름과 테이블 이름을 제공해야 합니다. 스키마가 기본값인 "public"이 아닌 경우에도 스키마를 제공해야 합니다. 데이터베이스는 *connectionName*의 URL 속성을, dbtable을 통해 스키마 및 테이블 이름을 제공 받습니다.

- Azure SQL 인증 정보. 인증 정보를 구성하려면 앞 절차인 Azure SQL에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```

azuresql_write = glueContext.write_dynamic_frame.from_options(

```

```

connection_type="azuresql",
connection_options={
  "connectionName": "connectionName",
  "dbtable": "tableIdentifier"
}
)

```

Azure SQL 연결 옵션 참조

- `connectionName` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 연결 방법에 인증 정보를 제공하도록 구성된 AWS Glue Azure SQL 연결의 이름입니다.
- `databaseName` - 읽기/쓰기에 사용됩니다. 유효한 값: Azure SQL 데이터베이스 이름. 연결할 Azure SQL 데이터베이스의 이름입니다.
- `dbtable` - 쓰기 시 필수, `query`가 제공되지 않는 한 읽기 전용. 읽기 및 쓰기에 사용됩니다. 유효한 값: Azure SQL 테이블의 이름 또는 마침표로 구분된 스키마/테이블 이름 조합. 연결할 테이블을 식별하는 테이블과 스키마를 지정하는 데 사용됩니다. 기본 스키마는 "public"입니다. 테이블이 기본 스키마가 아닌 스키마를 사용하는 경우 이 정보를 양식 `schemaName.tableName`에 입력하십시오.
- `query` — 읽기에 사용됩니다. Azure SQL에서 읽을 때 검색해야 하는 내용을 정의하는 Transact-SQL SELECT 쿼리입니다. 자세한 내용은 [Microsoft 설명서](#)를 참조하십시오.

BigQuery 연결

AWS Glue for Spark를 사용하여 AWS Glue 4.0 이상 버전에서 Google BigQuery의 테이블에서 읽고 쓸 수 있습니다. Google SQL 쿼리를 사용하여 BigQuery에서 읽을 수 있습니다. AWS Glue 연결을 통해 AWS Secrets Manager에 저장된 보안 인증 정보를 사용하여 BigQuery에 연결합니다.

Google BigQuery에 대한 자세한 내용은 [Google Cloud BigQuery 웹사이트](#)를 참조하세요.

BigQuery 연결 구성

AWS Glue에서 Google BigQuery에 연결하려면 AWS Secrets Manager 보안 암호에서 Google Cloud Platform 보안 인증 정보를 생성하고 저장한 다음 해당 보안 암호를 Google BigQuery AWS Glue 연결에 연결해야 합니다.

BigQuery에 대한 연결 구성하기:

1. Google Cloud Platform에서 관련 리소스를 생성하고 식별합니다.
 - 연결하려는 BigQuery 테이블이 포함된 GCP 프로젝트를 생성하거나 식별합니다.

- BigQuery API를 활성화합니다. 자세한 내용은 [BigQuery Storage Read API를 사용하여 테이블 데이터 읽기](#)를 참조하세요.
2. Google Cloud Platform에서 서비스 계정 보안 인증 정보를 생성하고 내보냅니다.

BigQuery 보안 인증 마법사를 사용하여 [보안 인증 정보 생성하기](#) 단계를 신속하게 처리할 수 있습니다.

GCP에서 서비스 계정을 생성하려면 [서비스 계정 생성하기](#)에서 제공되는 튜토리얼을 따르세요.

- 프로젝트를 선택할 때 BigQuery 테이블이 포함된 프로젝트를 선택합니다.
- 서비스 계정의 GCP IAM 역할을 선택할 때 BigQuery 작업을 실행하여 BigQuery 테이블을 읽고, 쓰고, 생성할 수 있는 적절한 권한을 부여하는 역할을 추가하거나 생성하세요.

서비스 계정의 보안 인증 정보를 생성하려면 [서비스 계정 키 생성하기](#)에서 제공되는 튜토리얼을 따르세요.

- 키 유형을 선택할 때 JSON을 선택합니다.

이제 서비스 계정의 보안 인증 정보가 포함된 JSON 파일을 다운로드했어야 합니다. 예를 들면 다음과 같아야 합니다.

```
{
  "type": "service_account",
  "project_id": "*****",
  "private_key_id": "*****",
  "private_key": "*****",
  "client_email": "*****",
  "client_id": "*****",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "*****",
  "universe_domain": "googleapis.com"
}
```

3. base64는 다운로드한 보안 인증 파일을 인코딩합니다. AWS CloudShell 세션 또는 유사한 사항인 경우 명령줄에서 `cat credentialsFile.json | base64 -w 0`(를) 실행하여 이 작업을 수행할 수 있습니다. 이 명령의 출력인 *credentialString*을 유지합니다.

4. AWS Secrets Manager에서 Google Cloud Platform 보안 인증 정보를 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *credentialString*이 포함된 키 credentials에 대한 페어를 생성합니다.
5. AWS Glue 데이터 카탈로그에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 Google BigQuery를 선택합니다.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.
6. AWS Glue 작업과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여합니다.
7. AWS Glue 작업 구성에서 추가 네트워크 연결로 *connectionName*을 제공합니다.

BigQuery 테이블에서 읽기

사전 조건:

- 읽으려는 BigQuery 테이블. 양식 [dataset].[table]에 있는 BigQuery 테이블 및 데이터 세트 이름이 필요합니다. 이 *tableName*이라고 부르겠습니다.
- BigQuery 테이블의 청구 프로젝트. 프로젝트 이름인 *parentProject*가 필요합니다. 결제 상위 프로젝트가 없는 경우 테이블이 포함된 프로젝트를 사용합니다.
- BigQuery 인증 정보. AWS Glue로 연결 보안 인증 관리하기 단계를 완료하여 인증 정보를 구성합니다. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
bigquery_read = glueContext.create_dynamic_frame.from_options(
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "sourceType": "table",
        "table": "tableName",
    }
)
```

DynamicFrame에 반환되는 결과를 필터링하는 쿼리를 제공할 수도 있습니다. `query`, `sourceType`, `viewsEnabled` 및 `materializationDataset`을(를) 구성해야 합니다.

예:

추가 사전 조건:

BigQuery가 쿼리에 대한 구체화된 뷰를 작성할 수 있는 BigQuery 데이터 세트인 `materializationDataset`를 생서하거나 식별해야 합니다.

`materializationDataset`에서 테이블을 생성하려면 서비스 계정에 적절한 GCP IAM 권한을 부여해야 합니다.

```
glueContext.create_dynamic_frame.from_options(
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "materializationDataset": materializationDataset,
        "parentProject": "parentProject",
        "viewsEnabled": "true",
        "sourceType": "query",
        "query": "select * from bqtest.test"
    }
)
```

BigQuery 테이블에 쓰기

이 예시는 BigQuery 서비스에 직접 작성합니다. BigQuery는 '간접' 쓰기 메서드도 지원합니다. 간접 쓰기 구성에 대한 자세한 내용은 [the section called “Google BigQuery를 통한 간접 쓰기 사용”](#)을(를) 참조하세요.

사전 조건:

- 쓰려는 BigQuery 테이블. 양식 `[dataset].[table]`에 있는 BigQuery 테이블 및 데이터 세트 이름이 필요합니다. 자동으로 생성되는 새 테이블 이름을 제공할 수도 있습니다. 이 `tableName`이라고 부르겠습니다.
- BigQuery 테이블의 청구 프로젝트. 프로젝트 이름인 `parentProject`가 필요합니다. 결제 상위 프로젝트가 없는 경우 테이블이 포함된 프로젝트를 사용합니다.

- BigQuery 인증 정보. AWS Glue로 연결 보안 인증 관리하기 단계를 완료하여 인증 정보를 구성합니다. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
bigquery_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="bigquery",
    connection_options={
        "connectionName": "connectionName",
        "parentProject": "parentProject",
        "writeMethod": "direct",
        "table": "tableName",
    }
)
```

BigQuery연결 옵션 참조

- **project**— 기본값: Google Cloud 서비스 계정 기본값. 읽기 및 쓰기에 사용됩니다. 테이블과 연결된 Google Cloud 프로젝트의 이름.
- **table** -- (필수) 읽기 및 쓰기에 사용됩니다. [[project:]dataset.] 형식에 따른 BigQuery 테이블의 이름.
- **dataset**— **table** 옵션을 통해 정의되지 않은 경우 필수입니다. 읽기 및 쓰기에 사용됩니다. BigQuery 테이블을 포함하는 데이터 세트의 이름.
- **parentProject**— 기본값: Google Cloud 서비스 계정 기본값. 읽기 및 쓰기에 사용됩니다. 청구에 사용되는 **project**와(과) 연결된 Google Cloud 프로젝트의 이름.
- **sourceType** — 읽기에 사용됩니다. 읽을 때 필요합니다. 유효한 값: **table**, **query**은(는) 테이블로 읽을 것인지 쿼리로 읽을 것인지 AWS Glue에 알립니다.
- **materializationDataset** — 읽기에 사용됩니다. 유효한 값: 문자열. 보기의 구체화를 저장하는 데 사용되는 BigQuery 데이터 세트의 이름.
- **viewsEnabled** — 읽기에 사용됩니다. 기본값: **false**. 유효한 값: 참, 거짓. BigQuery에서 보기를 사용할지 여부를 구성합니다.
- **query** — 읽기에 사용됩니다. **viewsEnabled**이(가) 참일 때 사용됩니다. GoogleSQL DQL 쿼리.
- **temporaryGcsBucket**— 쓰기에 사용됩니다. **writeMethod**이(가) 기본값(**indirect**)으로 설정된 경우에 필요합니다. BigQuery에 쓰는 동안 중간 형식의 데이터를 저장하는 데 사용되는 Google Cloud Storage 버킷의 이름.

- `writeMethod` - 기본값: `indirect`. 유효한 값: `direct`, `indirect`. 쓰기에 사용됩니다. 데이터를 쓰는 데 사용되는 메서드를 지정합니다.
- `direct`(으)로 설정하면 커넥터가 BigQuery Storage 쓰기 API를 사용하여 데이터를 작성합니다.
- `indirect`(으)로 설정하면 커넥터가 Google Cloud Storage에 쓴 다음 로드 작업을 사용하여 BigQuery로 전송합니다. Google Cloud 서비스 계정에는 적절한 GCS 권한이 필요합니다.

Google BigQuery를 통한 간접 쓰기 사용

이 예시에서는 Google Cloud Storage에 데이터를 쓰고 Google BigQuery에 데이터를 복사하는 간접 쓰기를 사용합니다.

사전 조건:

임시 Google Cloud Storage 버킷인 `temporaryBucket`이 필요합니다.

AWS Glue GCP 서비스 계정의 GCP IAM 역할에는 `temporaryBucket`에 액세스하는 적절한 GCS 권한이 필요합니다.

추가 구성:

BigQuery를 사용하여 간접 쓰기 구성하기:

1. [the section called “BigQuery 구성”](#)을(를) 평가하고, GCP 보안 인증 JSON 파일을 찾거나 다시 다운로드합니다. 작업에 사용되는 Google BigQuery AWS Glue 연결의 AWS Secrets Manager 보안 암호인 `secretName`을(를) 식별합니다.
2. 보안 인증 JSON 파일을 적절하게 안전한 Amazon S3 위치에 업로드합니다. 향후 단계를 위해 파일 경로인 `s3secretpath`를 유지합니다.
3. `secretName`을(를) 편집하고 `spark.hadoop.google.cloud.auth.service.account.json.keyfile` 키를 추가합니다. 값을 `s3secretpath`에 설정합니다.
4. AWSGlue 작업 Amazon S3 IAM에 `s3secretpath`에 액세스할 수 있는 권한을 부여합니다.

이제 쓰기 메서드에 임시 GCS 버킷 위치를 제공할 수 있습니다. `indirect`은(는) 이전에는 기본값이었기 때문에 `writeMethod`을(를) 제공하지 않아도 됩니다.

```
bigquery_write = glueContext.write_dynamic_frame.from_options(
    frame=frameToWrite,
    connection_type="bigquery",
```

```

connection_options={
  "connectionName": "connectionName",
  "parentProject": "parentProject",
  "temporaryGcsBucket": "temporaryBucket",
  "table": "tableName",
}
)

```

JDBC 연결

일반적으로 특정 관계형 데이터베이스 유형은 JDBC 표준을 통한 연결을 지원합니다. JDBC에 대한 자세한 내용은 [Java JDBC API](#) 설명서를 참조하세요. AWS Glue는 기본적으로 JDBC 커넥터를 통해 특정 데이터베이스 연결을 지원합니다. JDBC 라이브러리는 AWS Glue Spark 작업에서 제공됩니다. AWS Glue 라이브러리를 사용하여 이러한 데이터베이스 유형에 연결할 때 표준 옵션 세트에 액세스할 수 있습니다.

JDBC connectionType 값은 다음과 같습니다.

- "connectionType": "sqlserver": Microsoft SQL Server 데이터베이스에 연결을 지정합니다.
- "connectionType": "mysql": MySQL 데이터베이스에 대한 연결을 지정합니다.
- "connectionType": "oracle": Oracle Database에 연결을 지정합니다.
- "connectionType": "postgresql": PostgreSQL 데이터베이스에 대한 연결을 지정합니다.
- "connectionType": "redshift": Amazon Redshift 데이터베이스에 대한 연결을 지정합니다. 자세한 내용은 [the section called “Redshift 연결”](#) 단원을 참조하십시오.

다음 표에는 AWS Glue에서 지원되는 JDBC 드라이버 버전 목록이 있습니다.

제품	Glue 4.0용 JDBC 드라이버 버전	Glue 3.0용 JDBC 드라이버 버전	Glue 0.9, 1.0, 2.0용 JDBC 드라이버 버전
Microsoft SQL Server	9.4.0	7.x	6.x
MySQL	8.0.23	8.0.23	5.1
Oracle Database	21.7	21.1	11.2
PostgreSQL	42.3.6	42.2.18	42.1.x

제품	Glue 4.0용 JDBC 드라이버 버전	Glue 3.0용 JDBC 드라이버 버전	Glue 0.9, 1.0, 2.0용 JDBC 드라이버 버전
MongoDB	4.7.2	4.0.0	2.0.0
Amazon Redshift *	redshift-jdbc42-2.1.0.16	redshift-jdbc41-1.2.12.1017	redshift-jdbc41-1.2.12.1017

* Amazon Redshift 연결 유형의 경우 형식 지정 옵션을 비롯하여 JDBC 연결에 대한 연결 옵션에 포함된 다른 모든 옵션 이름 및 값 페어는 기본 SparkSQL DataSource에 직접 전달됩니다. AWS Glue 4.0 이상 버전의 Spark 작업을 포함하는 AWS Glue에서 Amazon Redshift용 AWS Glue 네이티브 커넥터는 Apache Spark용 Amazon Redshift 통합을 사용합니다. 자세한 내용은 [Apache Spark용 Amazon Redshift 통합](#)을 참조하세요. 이전 버전인 경우 [Amazon Redshift data source for Spark](#)를 참조하세요.

JDBC를 사용하여 Amazon RDS 데이터 스토어에 연결하도록 Amazon VPC를 구성하려면 [the section called “Amazon RDS 데이터 스토어에 연결하도록 Amazon VPC 설정”](#) 섹션을 참조하세요.

Note

AWS Glue 작업은 실행 중에 하나의 서브넷에만 연결됩니다. 이로 인해 동일한 작업을 통해 여러 데이터 소스에 연결하는 기능이 영향을 받을 수 있습니다. 이 동작은 JDBC 소스에만 국한되지 않습니다.

주제

- [JDBC 연결 옵션 참조](#)
- [sampleQuery 사용](#)
- [사용자 지정 JDBC 드라이버 사용](#)
- [JDBC 테이블을 병렬로 읽기](#)
- [AWS Glue에서 Amazon RDS 데이터 스토어에 대해 JDBC를 연결하도록 Amazon VPC 설정](#)

JDBC 연결 옵션 참조

이미 JDBC AWS Glue 연결을 정의한 경우 URL, 사용자 및 암호와 같은 JDBC 연결에 정의된 구성 속성을 재사용할 수 있으므로 코드에서 연결 옵션으로 지정할 필요가 없습니다. 이 기능은 AWS Glue 3.0 이상 버전에서 사용할 수 있습니다. 이렇게 하려면 다음 연결 속성을 사용합니다.

- "useConnectionProperties": 연결에서 구성을 사용하려는 경우 이 값을 'true'로 설정합니다.
- "connectionName": 구성을 검색할 연결 이름을 입력합니다. 연결은 작업과 동일한 영역에 정의되어 있어야 합니다.

이 연결 옵션을 JDBC 연결에 사용하십시오.

- "url": (필수) 데이터베이스의 JDBC URL.
- "dbtable": (필수) 읽을 데이터베이스 테이블입니다. 데이터베이스 내의 스키마를 지원하는 JDBC 데이터 스토어의 경우 schema.table-name에 대해 지정합니다. 스키마가 제공되지 않으면 기본 "퍼블릭" 스키마가 사용됩니다.
- "user": (필수 사항) 연결할 때 사용할 사용자 이름입니다.
- "password": (필수) 연결할 때 사용할 암호.
- (선택 사항) 다음 옵션을 사용하면 사용자 정의 JDBC 드라이버를 제공할 수 있습니다. AWS Glue에서 기본적으로 지원하지 않는 드라이버를 사용해야 한다면 이 옵션을 사용하십시오.

ETL 작업은 원본과 대상이 같은 데이터베이스 제품이어도 데이터 원본과 대상에 여러 JDBC 드라이버 버전을 사용할 수 있습니다. 따라서 버전이 다른 원본 데이터베이스와 대상 데이터베이스 간에 데이터를 마이그레이션할 수 있습니다. 이 옵션을 사용하려면 먼저 Amazon S3에 JDBC 드라이버의 JAR 파일을 업로드해야 합니다.

- "customJdbcDriverS3Path": 사용자 정의 JDBC 드라이버의 Amazon S3 경로입니다.
- "customJdbcDriverClassName": JDBC 드라이버의 클래스 이름입니다.
- "bulkSize": (선택 사항) JDBC 대상에 신속하게 대량 로드하기 위해 병렬 삽입을 구성하는 데 사용됩니다. 데이터를 쓰거나 삽입할 때 사용할 병렬 처리 수준의 정수 값을 지정합니다. 이 옵션은 Arch User Repository(AUR)와 같은 데이터베이스에 대한 쓰기 성능을 향상시키는 데 유용합니다.
- "hashfield"(선택 사항) JDBC 테이블에서 병렬로 읽을 때 데이터를 파티션으로 구분하는 데 사용할 JDBC 테이블의 열 이름을 지정하기 위해 사용하는 문자열입니다. 'hashfield' 또는 'hashexpression'을 제공합니다. 자세한 내용은 [the section called "JDBC를 병렬로 읽기"](#) 단원을 참조하십시오.
- "hashexpression"(선택 사항) 정수를 반환하는 SQL select 절입니다. JDBC 테이블에서 병렬로 읽을 때 JDBC 테이블의 데이터를 파티션으로 구분하는 데 사용됩니다. 'hashfield' 또는 'hashexpression'을 제공합니다. 자세한 내용은 [the section called "JDBC를 병렬로 읽기"](#) 단원을 참조하십시오.

- "hashpartitions"(선택 사항) 양의 정수입니다. JDBC 테이블에서 병렬로 읽을 때 JDBC 테이블의 병렬 읽기 수를 지정하는 데 사용됩니다. 기본값: 7. 자세한 내용은 [the section called “JDBC를 병렬로 읽기”](#) 단원을 참조하십시오.
- "sampleQuery": (선택 사항) 사용자 지정 SQL 쿼리 문입니다. 테이블의 정보 하위 세트를 지정하여 테이블 콘텐츠의 샘플을 검색하는 데 사용됩니다. 데이터를 고려하지 않고 구성하면 DynamicFrame 메서드보다 효율성이 떨어져 시간 초과나 메모리 부족 오류가 발생할 수 있습니다. 자세한 내용은 [the section called “sampleQuery 사용”](#) 단원을 참조하십시오.
- "enablePartitioningForSampleQuery": (선택 사항) 부울입니다. 기본값: false. sampleQuery를 지정할 때 JDBC 테이블에서 병렬 읽기를 활성화하는 데 사용됩니다. true로 설정된 경우 AWS Glue에서 파티셔닝 조건을 추가하려면 **sampleQuery**가 'where' 또는 'and'로 끝나야 합니다. 자세한 내용은 [the section called “sampleQuery 사용”](#) 단원을 참조하십시오.
- "sampleSize": (선택 사항) 양의 정수입니다. 샘플 쿼리에서 반환되는 행의 수를 제한합니다. enablePartitioningForSampleQuery가 true인 경우에만 작동합니다. 파티셔닝이 활성화되지 않은 경우 대신 sampleQuery에서 "limit x"를 직접 추가하여 크기를 제한해야 합니다. 자세한 내용은 [the section called “sampleQuery 사용”](#) 단원을 참조하십시오.

sampleQuery 사용

이 섹션에서는 sampleQuery, enablePartitioningForSampleQuery 및 sampleSize 사용 방법을 설명합니다.

sampleQuery는 데이터 세트의 몇 개 행을 샘플링할 때 효율적입니다. 기본적으로 쿼리는 단일 실행기에 의해 실행됩니다. 데이터를 고려하지 않고 구성하면 DynamicFrame 메서드보다 효율성이 떨어져 시간 초과나 메모리 부족 오류가 발생할 수 있습니다. ETL 파이프라인의 일부로 기본 데이터베이스에서 SQL을 실행하는 방법은 일반적으로 성능 목적으로만 필요합니다. 데이터 세트의 몇 개 행을 미리 보려는 경우 [the section called “show”](#) 사용을 고려합니다. SQL을 사용하여 데이터 세트를 변환하려는 경우 DataFrame 양식의 데이터에 대해 SparkSQL 변환을 정의하는 데 [the section called “toDF”](#) 사용을 고려합니다.

쿼리가 다양한 테이블을 조작할 수 있지만 dbtable이 여전히 필요합니다.

sampleQuery를 사용하여 테이블 샘플 검색

기본 sampleQuery 동작을 사용하여 데이터 샘플을 검색할 때 AWS Glue는 뛰어난 처리량을 기대하지 않으므로 단일 실행기에서 쿼리를 실행합니다. 제공하는 데이터를 제한하고 성능 문제를 일으키지 않으려면 SQL에 LIMIT 절을 제공하는 것이 좋습니다.

Example 분할 없이 SampleQuery 사용

다음 코드 예제에서는 분할 없이 sampleQuery를 사용하는 방법을 보여줍니다.

```
//A full sql query statement.
val query = "select name from $tableName where age > 0 limit 1"
val connectionOptions = JsonOptions(Map(
  "url" -> url,
  "dbtable" -> tableName,
  "user" -> user,
  "password" -> password,
  "sampleQuery" -> query ))
val dyf = glueContext.getSource("mysql", connectionOptions)
  .getDynamicFrame()
```

대규모 데이터 세트에서 sampleQuery 사용

대규모 데이터 세트를 읽는 경우 JDBC 분할을 활성화하여 테이블을 병렬로 쿼리해야 할 수 있습니다. 자세한 내용은 [the section called “JDBC를 병렬로 읽기”](#) 단원을 참조하십시오. JDBC 파티셔닝과 함께 sampleQuery를 사용하려면 enablePartitioningForSampleQuery를 true로 설정합니다. 이 기능을 활성화하려면 sampleQuery를 일부 변경해야 합니다.

sampleQuery와 함께 JDBC 파티셔닝을 사용하는 경우 AWS Glue에서 파티셔닝 조건을 추가하려면 쿼리가 'where' 또는 'and'로 끝나야 합니다.

JDBC 테이블에서 병렬로 읽을 때 sampleQuery의 결과를 제한하려면 LIMIT 절을 지정하는 대신, "sampleSize" 파라미터를 설정합니다.

Example JDBC 분할과 함께 sampleQuery 사용

다음 코드 예제는 JDBC 분할과 함께 sampleQuery를 사용하는 방법을 보여줍니다.

```
//note that the query should end with "where" or "and" if use with JDBC partitioning.
val query = "select name from $tableName where age > 0 and"

//Enable JDBC partitioning by setting hashfield.
//to use sampleQuery with partitioning, set enablePartitioningForSampleQuery.
//use sampleSize to limit the size of returned data.
val connectionOptions = JsonOptions(Map(
  "url" -> url,
  "dbtable" -> tableName,
  "user" -> user,
  "password" -> password,
```

```

    "hashfield" -> primaryKey,
    "sampleQuery" -> query,
    "enablePartitioningForSampleQuery" -> true,
    "sampleSize" -> "1" ))
val dyf = glueContext.getSource("mysql", connectionOptions)
    .getDynamicFrame()

```

규칙 및 제한:

샘플 쿼리는 작업 북마크와 함께 사용할 수 없습니다. 두 가지 모두에 대한 구성이 제공되면 북마크 상태는 무시됩니다.

사용자 지정 JDBC 드라이버 사용

다음 코드 예제에서는 사용자 지정 JDBC 드라이버로 JDBC 데이터베이스에서 읽고 쓰는 방법을 보여줍니다. 한 버전의 데이터베이스 제품에서 읽고 같은 제품의 나중 버전에 쓰는 작업을 볼 수 있습니다.

Python

```

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

# Construct JDBC connection options
connection_mysql5_options = {
    "url": "jdbc:mysql://<jdbc-host-name>:3306/db",
    "dbtable": "test",
    "user": "admin",
    "password": "pwd"}

connection_mysql8_options = {
    "url": "jdbc:mysql://<jdbc-host-name>:3306/db",
    "dbtable": "test",
    "user": "admin",

```

```
"password": "pwd",
"customJdbcDriverS3Path": "s3://DOC-EXAMPLE-BUCKET/mysql-connector-
java-8.0.17.jar",
"customJdbcDriverClassName": "com.mysql.cj.jdbc.Driver"}

connection_oracle11_options = {
  "url": "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL",
  "dbtable": "test",
  "user": "admin",
  "password": "pwd"}

connection_oracle18_options = {
  "url": "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL",
  "dbtable": "test",
  "user": "admin",
  "password": "pwd",
  "customJdbcDriverS3Path": "s3://DOC-EXAMPLE-BUCKET/ojdbc10.jar",
  "customJdbcDriverClassName": "oracle.jdbc.OracleDriver"}

# Read from JDBC databases with custom driver
df_mysql8 = glueContext.create_dynamic_frame.from_options(connection_type="mysql",
  connection_options=connection_mysql8_options)

# Read DynamicFrame from MySQL 5 and write to MySQL 8
df_mysql5 = glueContext.create_dynamic_frame.from_options(connection_type="mysql",
  connection_options=connection_mysql5_options)
glueContext.write_from_options(frame_or_dfc=df_mysql5, connection_type="mysql",
  connection_options=connection_mysql8_options)

# Read DynamicFrame from Oracle 11 and write to Oracle 18
df_oracle11 =
  glueContext.create_dynamic_frame.from_options(connection_type="oracle",
  connection_options=connection_oracle11_options)
glueContext.write_from_options(frame_or_dfc=df_oracle11, connection_type="oracle",
  connection_options=connection_oracle18_options)
```

Scala

```
import com.amazonaws.services.glue.GlueContext
```

```
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  val MYSQL_5_URI: String = "jdbc:mysql://<jdbc-host-name>:3306/db"
  val MYSQL_8_URI: String = "jdbc:mysql://<jdbc-host-name>:3306/db"
  val ORACLE_11_URI: String = "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL"
  val ORACLE_18_URI: String = "jdbc:oracle:thin:@//<jdbc-host-name>:1521/ORCL"

  // Construct JDBC connection options
  lazy val mysql5JsonOption = jsonOptions(MYSQL_5_URI)
  lazy val mysql8JsonOption = customJDBCDriverJsonOptions(MYSQL_8_URI, "s3://DOC-EXAMPLE-BUCKET/mysql-connector-java-8.0.17.jar", "com.mysql.cj.jdbc.Driver")
  lazy val oracle11JsonOption = jsonOptions(ORACLE_11_URI)
  lazy val oracle18JsonOption = customJDBCDriverJsonOptions(ORACLE_18_URI, "s3://DOC-EXAMPLE-BUCKET/ojdbc10.jar", "oracle.jdbc.OracleDriver")

  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Read from JDBC database with custom driver
    val df_mysql8: DynamicFrame = glueContext.getSource("mysql",
mysql8JsonOption).getDynamicFrame()

    // Read DynamicFrame from MySQL 5 and write to MySQL 8
    val df_mysql5: DynamicFrame = glueContext.getSource("mysql",
mysql5JsonOption).getDynamicFrame()
    glueContext.getSink("mysql", mysql8JsonOption).writeDynamicFrame(df_mysql5)

    // Read DynamicFrame from Oracle 11 and write to Oracle 18
    val df_oracle11: DynamicFrame = glueContext.getSource("oracle",
oracle11JsonOption).getDynamicFrame()
    glueContext.getSink("oracle", oracle18JsonOption).writeDynamicFrame(df_oracle11)
```

```

    Job.commit()
  }

  private def jsonOptions(url: String): JsonOptions = {
    new JsonOptions(
      s""""{"url": "${url}",
        |"dbtable":"test",
        |"user": "admin",
        |"password": "pwd"}"""".stripMargin)
  }

  private def customJDBCdriverJsonOptions(url: String, customJdbcDriverS3Path:
String, customJdbcDriverClassName: String): JsonOptions = {
    new JsonOptions(
      s""""{"url": "${url}",
        |"dbtable":"test",
        |"user": "admin",
        |"password": "pwd",
        |"customJdbcDriverS3Path": "${customJdbcDriverS3Path}",
        |"customJdbcDriverClassName" :
"${customJdbcDriverClassName}"}"""".stripMargin)
  }
}

```

JDBC 테이블을 병렬로 읽기

JDBC 테이블의 속성을 AWS Glue에서 분할된 데이터를 병렬로 읽도록 설정할 수 있습니다. 특정 속성을 설정할 때 AWS Glue에게 데이터의 논리적 파티션에 대해 병렬 SQL 쿼리를 실행하도록 지시합니다. 해시 필드 또는 해시 표현식을 설정하여 파티셔닝을 제어할 수 있습니다. 데이터에 액세스할 때 사용되는 병렬 읽기의 수도 제어할 수 있습니다.

JDBC 테이블에서 병렬로 읽는 것은 성능을 향상시킬 수 있는 최적화 기법입니다. 이 기법이 적절한 시기를 식별하는 프로세스에 대한 자세한 내용은 AWS 권장 가이드의 [Apache Spark용 AWS Glue 작업 성능 조정 모범 사례 가이드](#)에 있는 [데이터 스캔량 감소](#)를 참조하세요.

병렬 읽기를 활성화하려면 테이블 구조의 파라미터 필드에서 키 값 페어를 설정합니다. JSON 표기법을 사용하여 테이블 파라미터 필드 값을 설정합니다. 테이블 속성을 편집하는 방법에 대한 더 자세한 정보는 [테이블 세부 정보 보기 및 편집](#)를 참조하십시오. ETL(추출, 변환 및 로드) 방법 `create_dynamic_frame_from_options` 및 `create_dynamic_frame_from_catalog`을 호출할 때도 병렬 읽기를 활성화할 수 있습니다 이러한 방법에서 옵션을 지정하는 방법은 [from_options](#) 및 [from_catalog](#) 단원을 참조하십시오.

이 방법을 JDBC 테이블, 즉 베이스 데이터가 JDBC 데이터 스토어인 대부분의 테이블에 사용할 수 있습니다. Amazon Redshift 및 Amazon S3 테이블을 읽을 때 이러한 속성은 무시됩니다.

hashfield

hashfield를 사용할 JDBC 테이블의 열 이름으로 설정하여 데이터를 파티션으로 분할합니다. 파티션 사이로 데이터를 분배할 값이 이 열에 고르게 분포하는 것이 가장 좋습니다. 이 열은 데이터 형식이 될 수 있습니다. AWS Glue는 이 열로 분할된 데이터를 읽기 위해 병렬로 실행되는 비-중첩 쿼리를 생성합니다. 예를 들어, 데이터가 월별로 고르게 분포할 경우 month 열을 사용하여 각 월의 데이터를 병렬로 읽을 수 있습니다.

```
'hashfield': 'month'
```

AWS Glue은 쿼리를 생성하여 필드 값을 파티션 번호로 해싱하고 전체 파티션의 질의를 병렬로 실행합니다. 사용자의 쿼리를 사용하여 테이블 읽기를 분할하려면 hashfield 대신 hashexpression를 제공합니다.

hashexpression

hashexpression을 자연수를 반환하는 SQL 표현식(JDBC 데이터베이스 엔진 문법 준수)으로 설정합니다. 간단한 표현식은 테이블의 숫자 열 이름입니다. AWS Glue는 WHERE 절의 hashexpression를 사용하여 JDBC 데이터를 병렬로 읽어 데이터를 분할할 SQL 쿼리를 생성합니다.

예를 들어, 숫자 열 customerID를 사용하여 고객 번호로 분할된 데이터를 읽습니다.

```
'hashexpression': 'customerID'
```

AWS Glue이 파티셔닝을 제어하게 하려면 hashexpression 대신 hashfield를 제공합니다.

hashpartitions

hashpartitions을 JDBC 테이블 병렬 읽기 번호로 설정합니다. 이 속성을 설정하지 않을 경우 기본값은 7입니다.

예를 들어, 병렬 읽기 번호를 5로 설정하면 AWS Glue는 쿼리가 5개(또는 이하)인 데이터를 읽습니다.

```
'hashpartitions': '5'
```

AWS Glue에서 Amazon RDS 데이터 스토어에 대해 JDBC를 연결하도록 Amazon VPC 설정

JDBC를 사용하여 Amazon RDS의 데이터베이스에 연결하는 경우 추가 설정을 수행해야 합니다. AWS Glue 구성 요소가 Amazon RDS와 통신할 수 있도록 하려면 Amazon VPC에서 Amazon RDS 데이터 스토어에 대한 액세스를 설정해야 합니다. AWS Glue가 구성 요소 간에 통신하려면 보안 그룹이 모든 TCP 포트에 자기 참조 인바운드 규칙을 지정해야 합니다. 자체 참조 규칙을 생성하면 소스를 VPC의 동일한 보안 그룹으로 제한할 수 있습니다. 자체 참조 규칙을 사용한다고 해서 VPC가 모든 네트워크에 개방되는 것은 아닙니다. VPC의 기본 보안 그룹은 ALL Traffic의 자기 참조 인바운드 액세스 규칙이 있습니다.

AWS Glue와 Amazon RDS 데이터 스토어 간에 액세스를 설정하려면

1. AWS Management Console 로그인하고 <https://console.aws.amazon.com/rds/> 에서 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 콘솔에서 Amazon RDS 데이터베이스에 대한 액세스를 제어하는 데 사용되는 보안 그룹을 식별합니다.

왼쪽 탐색 창에서 데이터베이스를 선택한 다음 기본 창의 목록에서 연결하려는 인스턴스를 선택합니다.

데이터베이스 세부 정보 페이지의 연결 및 보안 탭에서 VPC 보안 그룹을 찾습니다.

3. 네트워크 아키텍처를 기반으로 AWS Glue 서비스에 대한 액세스를 허용하기 위해 수정하는 것이 가장 좋은 관련 보안 그룹을 식별하십시오. 나중에 참조할 수 *database-security-group* 있도록 이름을 저장하십시오. 적절한 보안 그룹이 없는 경우 Amazon RDS 설명서의 [보안 그룹을 생성하여 VPC에서 DB 인스턴스에 대한 액세스 권한 제공](#) 지침을 따르세요.

4. AWS Management Console [로그인하고 https://console.aws.amazon.com/vpc/](https://console.aws.amazon.com/vpc/) 에서 Amazon VPC 콘솔을 엽니다.

5. Amazon VPC 콘솔에서 업데이트 방법을 식별합니다. *database-security-group*

왼쪽 탐색 창에서 보안 그룹을 선택한 다음 기본 창의 *database-security-group* 목록에서 선택합니다.

6. *database-security-group*, 의 보안 그룹 ID를 *database-sg-id* 식별하십시오. 나중에 참조할 수 있도록 저장합니다.

보안 그룹 세부 정보 페이지에서 보안 그룹 ID를 찾으세요.

7. 의 인바운드 규칙을 변경하고 구성 요소가 *database-security-group* 통신할 수 있도록 AWS Glue 자체 참조 규칙을 추가하십시오. 구체적으로, 유형은 이고, 프로토콜은 이고, 포트 범위는 모든 포트를 **All TCP** 포함하며, 소스는 **TCP** 인 규칙이 있는지 추가하거나 확인하십시오. *database-sg-id* 소스에 입력한 보안 그룹이 편집 중인 보안 그룹과 동일한지 확인합니다.

보안 그룹 세부 정보 페이지에서 인바운드 규칙 편집을 선택합니다.

인바운드 규칙은 다음과 비슷하게 보입니다.

유형	프로토콜	포트 범위	소스
모든 TCP	TCP	0~65535	<i>database-sg-id</i>

8. 아웃바운드 트래픽에 대한 규칙을 추가합니다.

보안 그룹 세부 정보 페이지에서 아웃바운드 규칙 편집을 선택합니다.

보안 그룹에서 모든 아웃바운드 트래픽을 허용하는 경우에는 별도의 규칙이 필요하지 않습니다. 예:

유형	프로토콜	포트 범위	대상
모든 트래픽	ALL	ALL	0.0.0.0/0

네트워크 아키텍처가 아웃바운드 트래픽을 제한하도록 설계된 경우 다음과 같은 아웃바운드 규칙을 생성하세요.

유형은 이고 All TCP, 프로토콜은 이고, 포트 범위는 모든 포트를 포함하며 TCP, 대상은 인 자체 참조 규칙을 생성합니다. *database-sg-id* 대상에 입력한 보안 그룹이 편집 중인 보안 그룹과 동일한지 확인합니다.

Amazon S3 VPC 엔드포인트를 사용하는 경우에는 HTTPS 규칙을 추가하여 VPC에서 Amazon S3로의 트래픽을 허용하세요. Amazon S3 게이트웨이 엔드포인트에 대한 관리형 접두사 목록 *prefix-list-ids3-#* ID로 유형 HTTPS, 프로토콜, 포트 범위, 목적지가 인 규칙을 생성합니다. TCP 443 접두사 목록 및 Amazon S3 게이트웨이 엔드포인트에 대한 자세한 내용은 Amazon VPC 설명서에서 [Amazon S3에 대한 게이트웨이 엔드포인트](#)를 참조하세요.

예:

유형	프로토콜	포트 범위	대상
모든 TCP	TCP	0~65535	<i>database-sg-id</i>
HTTPS	TCP	443	<i>s3- prefix-li st-id</i>

MongoDB 연결

AWS Glue for Spark를 사용하여 AWS Glue 4.0 이상 버전에서 MongoDB와 MongoDB Atlas의 테이블에서 읽고 쓸 수 있습니다. AWS Glue 연결을 통해 AWS Secrets Manager에 저장된 사용자 이름 및 암호 보안 인증 정보를 사용하여 MongoDB에 연결할 수 있습니다.

MongoDB 대한 자세한 내용은 [MongoDB 설명서](#)를 참조하십시오.

MongoDB 연결 구성

AWS Glue에서 MongoDB에 연결하려면 MongoDB 보안 인증 정보와 *mongodbUser*, *mongodbPass*가 필요합니다.

AWS Glue에서 MongoDB에 연결하려면 몇 가지 전제 조건이 필요할 수 있습니다.

- MongoDB 인스턴스가 Amazon VPC에 있는 경우, 퍼블릭 인터넷을 통과하는 트래픽 없이 AWS Glue 작업이 MongoDB 인스턴스와 통신할 수 있도록 Amazon VPC를 구성하십시오.

Amazon VPC에서 AWS Glue가 작업을 실행하는 동안 사용할 VPC, 서브넷 및 보안 그룹을 식별하거나 생성합니다. 또한 MongoDB 인스턴스와 이 위치 간의 네트워크 트래픽을 허용하도록 Amazon VPC를 구성해야 합니다. 네트워크 레이아웃에 따라 보안 그룹 규칙, 네트워크 ACL, NAT 게이트웨이 및 피어링 연결을 변경해야 할 수도 있습니다.

그런 다음 MongoDB와 함께 사용할 수 있도록 AWS Glue를 구성할 수 있습니다.

MongoDB 연결 구성 방법:

1. 대안으로 AWS Secrets Manager에서 MongoDB 보안 인증을 사용하여 보안 암호를 생성할 수 있습니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create](#)

an [AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.

- 키/값 페어를 선택하면 값 *mongodbUser*이 포함된 키 username에 대한 페어를 생성합니다.

키/값 페어를 선택하면 값 *mongodbPass*가 포함된 키 password에 대한 페어를 생성합니다.

2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.

- 연결 유형을 선택할 때에는 MongoDB 또는 MongoDB Atlas를 선택합니다.
- MongoDB URL 또는 MongoDB Atlas URL을 선택할 때에는 MongoDB 인스턴스의 호스트 이름을 제공하십시오.

MongoDB URL은 `mongodb://mongoHost:mongoPort/mongoDBName` 형식으로 제공됩니다.

MongoDB Atlas URL은 `mongodb+srv://mongoHost:mongoPort/mongoDBName` 형식으로 제공됩니다.

연결을 위한 기본 데이터베이스를 제공하는 *mongoDBName*은 선택 사항입니다.

- Secrets Manager 암호를 생성하기로 선택한 경우 AWS Secrets Manager 보안 인증 정보 유형을 선택합니다.

그런 다음 AWS 암호에 *secretName*을 입력합니다.

- 사용자 이름과 비밀번호를 제공하기로 선택한 경우 *mongodbUser*와 *mongodbPass*를 제공하십시오.

3. 다음과 같은 상황에서는 추가 구성이 필요할 수도 있습니다.

- Amazon VPC에서 AWS에 호스팅된 MongoDB 인스턴스의 경우
 - MongoDB 보안 인증 정보를 정의하는 AWS Glue 연결에 Amazon VPC 연결 정보를 제공해야 합니다. 연결을 만들거나 업데이트할 때 네트워크 옵션에서 VPC, 서브넷 및 보안 그룹을 설정합니다.

AWS Glue MongoDB 연결을 생성한 후에는 연결 방법을 호출하기 전에 다음 조치를 수행해야 합니다.

- Secrets Manager 암호를 생성하기로 선택한 경우, AWS Glue 작업과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여하십시오.
- AWS Glue 작업 구성에서 추가 네트워크 연결로 *connectionName*을 제공합니다.

AWS Glue for Spark에서 AWS Glue MongoDB 연결을 사용하려면 연결 방법 호출에서 `connectionName` 옵션을 제공하십시오. 또는, [the section called “MongoDB와 통합”](#)의 단계에 따라 AWS Glue 데이터 카탈로그와 함께 연결을 사용할 수도 있습니다.

Glue AWS 연결을 사용하여 MongoDB에서 읽기

사전 조건:

- 읽으려는 MongoDB 컬렉션. 컬렉션에 대한 식별 정보가 필요합니다.

MongoDB 컬렉션은 데이터베이스 이름 *mongodbName* 및 컬렉션 이름 *mongodbCollection*으로 식별됩니다.

- 인증 정보를 제공하도록 구성된 AWS Glue MongoDB 연결입니다. 인증 정보를 구성하려면 앞 절 차인 MongoDB에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
mongodb_read = glueContext.create_dynamic_frame.from_options(
    connection_type="mongodb",
    connection_options={
        "connectionName": "connectionName",
        "database": "mongodbName",
        "collection": "mongodbCollection",
        "partitioner":
            "com.mongodb.spark.sql.connector.read.partitionner.SinglePartitionPartitioner",
        "partitionerOptions.partitionSizeMB": "10",
        "partitionerOptions.partitionKey": "_id",
        "disableUpdateUri": "false",
    }
)
```

MongoDB 테이블에 쓰기

이 예제에서는 기존 `DynamicFrame`, *dynamicFrame*의 정보를 MongoDB에 씁니다.

사전 조건:

- 쓰려는 MongoDB 컬렉션. 컬렉션에 대한 식별 정보가 필요합니다.

MongoDB 컬렉션은 데이터베이스 이름 *mongodbName* 및 컬렉션 이름 *mongodbCollection*으로 식별됩니다.

- 인증 정보를 제공하도록 구성된 AWS Glue MongoDB 연결입니다. 인증 정보를 구성하려면 앞 절 차인 MongoDB에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="mongodb",
    connection_options={
        "connectionName": "connectionName",
        "database": "mongodbName",
        "collection": "mongodbCollection",
        "disableUpdateUri": "false",
        "retryWrites": "false",
    },
)
```

MongoDB에서 테이블에 쓰고 쓰는 중

이 예제에서는 기존 DynamicFrame, *dynamicFrame*의 정보를 MongoDB에 씁니다.

사전 조건:

- 읽으려는 MongoDB 컬렉션. 컬렉션에 대한 식별 정보가 필요합니다.

쓰려는 MongoDB 컬렉션. 컬렉션에 대한 식별 정보가 필요합니다.

MongoDB 컬렉션은 데이터베이스 이름 *mongodbName* 및 컬렉션 이름 *mongodbCollection*으로 식별됩니다.

- MongoDB 인증 정보, *mongodbUser*, *mongodbPassword*.

예:

Python

```
import sys
```

```
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext, SparkConf
from awsglue.context import GlueContext
from awsglue.job import Job
import time

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'], args)

output_path = "s3://some_bucket/output/" + str(time.time()) + "/"
mongo_uri = "mongodb://<mongo-instanced-ip-address>:27017"
mongo_ssl_uri = "mongodb://<mongo-instanced-ip-address>:27017"
write_uri = "mongodb://<mongo-instanced-ip-address>:27017"

read_mongo_options = {
    "uri": mongo_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
    "username": "mongodbUsername",
    "password": "mongodbPassword",
    "partitioner": "MongoSamplePartitioner",
    "partitionerOptions.partitionSizeMB": "10",
    "partitionerOptions.partitionKey": "_id"}

ssl_mongo_options = {
    "uri": mongo_ssl_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
    "ssl": "true",
    "ssl.domain_match": "false"
}

write_mongo_options = {
    "uri": write_uri,
    "database": "mongodbName",
    "collection": "mongodbCollection",
```

```

    "username": "mongodbUsername",
    "password": "mongodbPassword",
  }

# Get DynamicFrame from MongoDB
dynamic_frame =
glueContext.create_dynamic_frame.from_options(connection_type="mongodb",

connection_options=read_mongo_options)

# Write DynamicFrame to MongoDB
glueContext.write_dynamic_frame.from_options(dynamicFrame,
connection_type="mongodb", connection_options=write_mongo_options)

job.commit()

```

Scala

```

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.DynamicFrame
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  val DEFAULT_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  val WRITE_URI: String = "mongodb://<mongo-instanced-ip-address>:27017"
  lazy val defaultJsonOption = jsonOptions(DEFAULT_URI)
  lazy val writeJsonOption = jsonOptions(WRITE_URI)
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)

    // Get DynamicFrame from MongoDB
    val dynamicFrame: DynamicFrame = glueContext.getSource("mongodb",
defaultJsonOption).getDynamicFrame()

```

```

// Write DynamicFrame to MongoDB
glueContext.getSink("mongodb", writeJsonOption).writeDynamicFrame(dynamicFrame)

Job.commit()
}

private def jsonOptions(uri: String): JsonOptions = {
  new JsonOptions(
    s""""{uri": "${uri}",
      |"database": "mongodbName",
      |"collection": "mongodbCollection",
      |"username": "mongodbUsername",
      |"password": "mongodbPassword",
      |"ssl": "true",
      |"ssl.domain_match": "false",
      |"partitioner": "MongoSamplePartitioner",
      |"partitionerOptions.partitionSizeMB": "10",
      |"partitionerOptions.partitionKey": "_id"}"""".stripMargin)
  }
}

```

MongoDB 연결 옵션 참조

MongoDB에 대한 연결을 지정합니다. 연결 옵션은 소스 연결 및 싱크 연결에 따라 다릅니다.

다음과 같은 연결 속성은 소스 연결과 싱크 연결 간에 공유됩니다.

- `connectionName` - 읽기/쓰기에 사용됩니다. 연결 방법에 인증 및 네트워킹 정보를 제공하도록 구성된 AWS Glue MongoDB 연결의 이름입니다. [the section called “MongoDB 구성”](#)에서 설명한 대로 AWS Glue 연결을 구성하면 `connectionName`을 제공했을 때 `"uri"`, `"username"` 및 `"password"` 연결 옵션을 제공할 필요가 없어집니다.
- `"uri"`: (필수 사항) 읽을 소스 MongoDB 호스트로, `mongodb://<host>:<port>` 형식입니다. AWS Glue 4.0 이전의 AWS Glue 버전에서 사용됩니다.
- `"connection.uri"`: (필수 사항) 읽을 소스 MongoDB 호스트로, `mongodb://<host>:<port>` 형식입니다. AWS Glue 4.0 이상 버전에서 사용됩니다.
- `"username"`: (필수 사항) MongoDB 사용자 이름입니다.
- `"password"`: (필수 사항) MongoDB 암호입니다.

- "database": (필수 사항) 읽을 MongoDB 데이터베이스입니다. 이 옵션은 작업 스크립트에서 `glue_context.create_dynamic_frame_from_catalog`를 호출할 때 `additional_options`로 전달할 수도 있습니다.
- "collection": (필수 사항) 읽을 MongoDB 컬렉션입니다. 이 옵션은 작업 스크립트에서 `glue_context.create_dynamic_frame_from_catalog`를 호출할 때 `additional_options`로 전달할 수도 있습니다.

소스로서의 "connectionType": "mongodb"

소스로서의 "connectionType": "mongodb"에는 다음 연결 옵션을 사용합니다.

- "ssl": (선택 사항) true이면 SSL 연결을 시작합니다. 기본값은 false입니다.
- "ssl.domain_match": (선택 사항) true 및 ssl이 true이면 도메인 일치 검사가 수행됩니다. 기본값은 true입니다.
- "batchSize": (선택 사항) 내부 배치의 커서 내에서 사용되는 배치당 반환할 문서 수입니다.
- "partitioner": (선택 사항) MongoDB에서 입력 데이터를 읽는 파티셔너의 클래스 이름입니다. 커넥터는 다음 파티셔너를 제공합니다.
 - MongoDefaultPartitioner(기본값)(AWS Glue 4.0에서는 지원되지 않음)
 - MongoSamplePartitioner(MongoDB 3.2 이상 필요)(AWS Glue 4.0에서는 지원되지 않음)
 - MongoShardedPartitioner(AWS Glue 4.0에서는 지원되지 않음)
 - MongoSplitVectorPartitioner(AWS Glue 4.0에서는 지원되지 않음)
 - MongoPaginateByCountPartitioner(AWS Glue 4.0에서는 지원되지 않음)
 - MongoPaginateBySizePartitioner(AWS Glue 4.0에서는 지원되지 않음)
 - `com.mongodb.spark.sql.connector.read.partitioners.SinglePartitionPartitioner`
 - `com.mongodb.spark.sql.connector.read.partitioners.ShardedPartitioner`
 - `com.mongodb.spark.sql.connector.read.partitioners.PaginateIntoPartitionsPartitioner`
- "partitionerOptions": (선택 사항) 지정된 파티셔너에 대한 옵션입니다. 각 파티셔너에 대해 다음 옵션이 지원됩니다.
 - MongoSamplePartitioner: `partitionKey`, `partitionSizeMB`, `samplesPerPartition`
 - MongoShardedPartitioner: `shardkey`
 - MongoSplitVectorPartitioner: `partitionKey`, `partitionSizeMB`
 - MongoPaginateByCountPartitioner: `partitionKey`, `numberOfPartitions`
 - MongoPaginateBySizePartitioner: `partitionKey`, `partitionSizeMB`

이러한 옵션에 대한 자세한 내용은 MongoDB 설명서의 [파티셔너 구성](#)을 참조하십시오.

싱크로서의 "connectionType": "mongodb"

싱크로서의 "connectionType": "mongodb"에는 다음 연결 옵션을 사용합니다.

- "ssl": (선택 사항) true이면 SSL 연결을 시작합니다. 기본값은 false입니다.
- "ssl.domain_match": (선택 사항) true 및 ssl이 true이면 도메인 일치 검사가 수행됩니다. 기본값은 true입니다.
- "extendedBsonTypes": (선택 사항) true이면 데이터를 MongoDB에 쓸 때 확장 BSON 유형을 허용합니다. 기본값은 true입니다.
- "replaceDocument": (선택 사항) true이면 _id 필드가 포함된 데이터 세트를 저장할 때 전체 문서를 대체합니다. false이면 데이터 세트의 필드와 일치하는 문서의 필드만 업데이트됩니다. 기본값은 true입니다.
- "maxBatchSize": (선택 사항) 데이터를 저장할 때 대량 작업의 최대 배치 크기입니다. 기본값은 512입니다.
- "retryWrites": (선택 사항): AWS Glue에서 네트워크 오류가 발생하는 경우 특정 쓰기 작업을 한번 자동으로 재시도합니다.

SAP HANA 커넥션

AWS Glue for Spark를 사용하여 AWS Glue 4.0 이상 버전에서 SAP HANA의 테이블에서 읽고 쓸 수 있습니다. SQL 쿼리를 사용하여 SAP HANA에서 읽을 내용을 정의할 수 있습니다. AWS Glue SAP HANA 연결을 통해 AWS Secrets Manager에 저장된 JDBC 보안 인증 정보를 사용하여 SAP HANA에 연결합니다.

SAP HANA JDBC에 대한 자세한 내용은 [SAP HANA 설명서](#)를 참조하십시오.

SAP HANA 연결 구성

AWS Glue에서 SAP HANA에 연결하려면 AWS Secrets Manager 보안 암호에서 SAP HANA 보안 인증 정보를 생성하고 저장한 다음 해당 보안 암호를 SAP HANA AWS 연결에 연결해야 합니다. SAP HANA 서비스와 AWS Glue 간의 네트워크 연결을 구성해야 합니다.

SAP HANA에 연결하려면 몇 가지 사전 요구 사항이 필요할 수 있습니다.

- SAP HANA 서비스가 Amazon VPC에 있는 경우, 퍼블릭 인터넷을 통과하는 트래픽 없이 AWS Glue 작업이 SAP HANA 서비스와 통신할 수 있도록 Amazon VPC를 구성하십시오.

Amazon VPC에서 AWS Glue가 작업을 실행하는 동안 사용할 VPC, 서브넷 및 보안 그룹을 식별하거나 생성합니다. 또한 Amazon VPC가 SAP HANA 엔드포인트와 이 위치 간의 네트워크 트래픽을 허용하도록 구성되어 있는지 확인해야 합니다. 작업을 수행하려면 SAP HANA JDBC 포트와의 TCP 연결을 설정해야 합니다. SAP HANA 포트에 대한 자세한 내용은 [SAP HANA 설명서](#)를 참조하십시오. 네트워크 레이아웃에 따라 보안 그룹 규칙, 네트워크 ACL, NAT 게이트웨이 및 피어링 연결을 변경해야 할 수도 있습니다.

- SAP HANA 엔드포인트가 인터넷에 액세스할 수 있는 경우 추가 전제 조건은 없습니다.

SAP HANA에 대한 연결 구성하는 방법:

1. AWS Secrets Manager에서 SAP HANA 보안 인증을 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *saphanaUsername*이 포함된 키 user에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *saphanaPassword*가 포함된 키 password에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 AWS Glue에서 이용하기 위해 연결 이름 *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 SAP HANA를 선택합니다.
 - SAP HANA URL을 제공할 때는 인스턴스의 URL을 제공하십시오.

SAP HANA JDBC URL은

```
jdbc:sap://saphanaHostname:saphanaPort/?databaseName=saphanaDBname,ParameterName
```

형식입니다

AWS Glue에는 다음과 같은 JDBC URL 매개변수가 필요합니다.

- databaseName - 연결할 SAP HANA의 기본 데이터베이스입니다.
- AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.

AWS Glue SAP HANA 연결을 생성한 후에는 AWS Glue 작업을 실행하기 전에 다음 단계를 수행해야 합니다.

- AWS Glue 작업과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여합니다.

- AWS Glue 작업 구성에서 추가 네트워크 연결로 *connectionName*을 제공합니다.

SAP HANA 테이블에서 읽는 중

사전 조건:

- 읽으려는 SAP HANA 테이블. 테이블에 대한 식별 정보가 필요합니다.

SAP HANA 테이블 이름 및 스키마 이름을 *schemaName.tableName* 형식으로 테이블을 지정할 수 있습니다. 테이블이 기본 스키마 "public"에 있는 경우 스키마 이름과 "." 구분 기호는 필요하지 않습니다. 이것을 *tableIdentifier*라고 합니다. 데이터베이스는 *connectionName*에서 JDBC URL 매개변수로 제공된다는 점에 유의하십시오.

- 인증 정보를 제공하도록 구성된 AWS Glue SAP HANA 연결입니다. 인증 정보를 구성하려면 앞 절 차인 SAP HANA에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
saphana_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="saphana",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableIdentifier",
    }
)
```

SELECT SQL 쿼리에 반환되는 결과를 필터링하는 쿼리를 제공할 수도 있습니다. *query*을 구성해야 합니다.

예:

```
saphana_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="saphana",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

SAP HANA 테이블에 쓰기

이 예제에서는 기존 `DynamicFrame`, `dynamicFrame`의 정보를 SAP HANA에 기록합니다. 테이블에 이미 정보가 있는 경우 AWS Glue에서 오류가 발생합니다.

사전 조건:

- 쓰려고 하는 SAP HANA 테이블.

SAP HANA 테이블 이름 및 스키마 이름을 `schemaName.tableName` 형식으로 테이블을 지정할 수 있습니다. 테이블이 기본 스키마 "public"에 있는 경우 스키마 이름과 "." 구분 기호는 필요하지 않습니다. 이것을 `tableIdentifier`라고 합니다. 데이터베이스는 `connectionName`에서 JDBC URL 매개변수로 제공된다는 점에 유의하십시오.

- SAP HANA 인증 정보. 인증 정보를 구성하려면 앞 절차인 SAP HANA에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 `connectionName`이 필요합니다.

예:

```
options = {
    "connectionName": "connectionName",
    "dbtable": 'tableIdentifier'
}

saphana_write = glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="saphana",
    connection_options=options
)
```

SAP HANA 연결 옵션 참조

- `connectionName` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 연결 방법에 인증 및 네트워킹 정보를 제공하도록 구성된 AWS Glue SAP HANA 연결의 이름입니다.
- `databaseName` - 읽기/쓰기에 사용됩니다. 유효한 값: SAP HANA의 데이터베이스 이름. 연결할 데이터베이스의 이름입니다.
- `dbtable` - 쓰기 시 필수, `query`가 제공되지 않는 한 읽기 전용. 읽기 및 쓰기에 사용됩니다. 유효한 값: SAP HANA SQL FROM 절의 내용입니다. 연결할 SAP HANA의 테이블을 식별합니다. 테이블 이름 이외의 다른 SQL(예: 하위 쿼리)을 제공할 수도 있습니다. 자세한 내용은 SAP HANA 설명서의 [From 절](#)을 참조하십시오.

- query — 읽기에 사용됩니다. SAP HANA에서 읽을 때 검색해야 하는 내용을 정의하는 SAP HANA SQL SELECT 쿼리입니다.

Snowflake 연결

AWS Glue for Spark를 사용하여 AWS Glue 4.0 이상 버전에서 Snowflake의 테이블에서 읽고 쓸 수 있습니다. SQL 쿼리를 사용하여 Snowflake에서 읽을 수 있습니다. 사용자와 암호를 사용하여 Snowflake에 연결할 수 있습니다. AWS Glue 데이터 카탈로그를 통해 AWS Secrets Manager에 저장된 Snowflake 보안 인증을 참조할 수 있습니다. AWS Glue for Spark용 데이터 카탈로그 Snowflake 보안 인증은 크롤러용 데이터 카탈로그 Snowflake 보안 인증과 별도로 저장됩니다. Snowflake에 연결하도록 구성된 JDBC 유형 연결이 아니라 SNOWFLAKE 유형 연결을 선택해야 합니다.

Snowball에 대한 자세한 내용은 [Snowflake 웹 사이트](#)를 참조하세요. AWS에서 Snowflake에 대한 자세한 내용은 [Snowflake Data Warehouse on Amazon Web Services](#)를 참조하세요.

Snowflake 연결 구성

인터넷을 통해 사용할 수 있는 Snowflake 데이터베이스에 연결하기 위한 AWS의 필수 조건은 없습니다.

선택적으로 다음 구성을 수행하여 AWS Glue로 연결 보안 인증을 관리할 수 있습니다.

AWS Glue로 연결 보안 인증을 관리하려면

1. Snowflake에서 사용자, *snowflakeUser* 및 암호, *snowflakePassword*를 생성합니다.
2. AWS Secrets Manager에서 Snowflake 보안 인증을 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 키가 sfUser인 *snowflakeUser*에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 키가 sfPassword인 *snowflakePassword*에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 키가 sfWarehouse인 Snowflake 웨어하우스를 제공할 수 있습니다.
3. AWS Glue 데이터 카탈로그에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 Snowflake를 선택합니다.

- Snowflake URL을 선택할 때 Snowflake 인스턴스의 URL을 제공합니다. URL은 `account_identifier.snowflakecomputing.com` 양식의 호스트 이름을 사용합니다.
 - AWS 보안 암호를 선택할 때 `secretName`을 입력합니다.
4. AWS Glue 작업 구성에서 추가 네트워크 연결로 `connectionName`을 제공합니다.

다음과 같은 상황에서 다음이 필요할 수 있습니다.

- Amazon VPC에서 AWS에 호스팅된 Snowflake의 경우
 - Snowflake에 적합한 Amazon VPC 구성이 필요합니다. Amazon VPC를 구성하는 방법에 대한 자세한 내용은 Snowflake 설명서의 [AWS PrivateLink & Snowflake](#)를 참조하세요.
 - AWS Glue에 적합한 Amazon VPC 구성이 필요합니다. [the section called “VPC 엔드포인트\(AWS PrivateLink\)”](#).
 - Snowflake 보안 인증을 정의하는 AWS Secrets Manager 보안 암호의 ID 외에도 Amazon VPC 연결 정보를 제공하는 AWS Glue Data Catalog 연결을 생성해야 합니다. 이전 항목에 링크된 Snowflake 설명서에 설명된 대로 AWS PrivateLink 사용 시 URL이 변경됩니다.
 - 데이터 카탈로그 연결을 추가 네트워크 연결로 포함하는 작업 구성이 필요합니다.

Snowflake 테이블에서 읽기

사전 조건: 읽으려는 Snowflake 테이블. Snowflake 테이블 이름, `tableName`이 필요합니다.

Snowflake URL `snowflakeUrl`, 사용자 이름 `snowflakeUser` 및 암호 `snowflakePassword`가 필요합니다. Snowflake 사용자에게 기본 네임스페이스가 설정되어 있지 않은 경우 Snowflake 데이터베이스 이름, `databaseName` 및 스키마 이름, `schemaName`이 필요합니다. 또한 Snowflake 사용자에게 기본 웨어하우스가 설정되어 있지 않은 경우 웨어하우스 이름, `warehouseName`이 필요합니다.

예:

추가 사전 조건: AWS Glue로 연결을 관리하려면 단계를 수행하여 `snowflakeUrl`, `snowflakeUsername` 및 `snowflakePassword`를 구성합니다. 이 단계를 검토하려면 이전 [the section called “Snowflake 구성”](#) 섹션을 참조하세요. 연결할 추가 네트워크 연결을 선택하기 위해 `connectionName` 파라미터를 사용합니다.

```
snowflake_read = glueContext.create_dynamic_frame.from_options(
    connection_type="snowflake",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName",
```

```

    "sfDatabase": "databaseName",
    "sfSchema": "schemaName",
    "sfWarehouse": "warehouseName",
  }
)

```

또한 autopushdown 및 query 파라미터를 사용하여 Snowflake 테이블의 일부를 읽을 수 있습니다. 이 방법은 결과를 Spark에 로드한 후 필터링하는 것보다 훨씬 더 효율적일 수 있습니다. 모든 판매가 동일한 테이블에 저장되지만 연휴에 특정 매장의 매출만 분석하려는 예제를 살펴봅니다. 해당 정보가 테이블에 저장된 경우 다음과 같이 조건자 푸시다운을 사용하여 결과를 가져올 수 있습니다.

```

snowflake_node = glueContext.create_dynamic_frame.from_options(
    connection_type="snowflake",
    connection_options={
        "autopushdown": "on",
        "query": "select * from sales where store='1' and IsHoliday='TRUE'",
        "connectionName": "snowflake-glue-conn",
        "sfDatabase": "databaseName",
        "sfSchema": "schemaName",
        "sfWarehouse": "warehouseName",
    }
)

```

Snowflake 테이블에 쓰기

사전 조건: 쓰려는 Snowflake 데이터베이스. 현재 또는 원하는 테이블 이름, *TableName*이 필요합니다. Snowflake URL *snowflakeUrl*, 사용자 이름 *snowflakeUser* 및 암호 *snowflakePassword*가 필요합니다. Snowflake 사용자에게 기본 네임스페이스가 설정되어 있지 않은 경우 Snowflake 데이터베이스 이름, *databaseName* 및 스키마 이름, *schemaName*이 필요합니다. 또한 Snowflake 사용자에게 기본 웨어하우스가 설정되어 있지 않은 경우 웨어하우스 이름, *warehouseName*이 필요합니다.

예:

추가 사전 조건: AWS Glue로 연결을 관리하려면 단계를 수행하여 *snowflakeUrl*, *snowflakeUsername* 및 *snowflakePassword*를 구성합니다. 이 단계를 검토하려면 이전 [the section called “Snowflake 구성”](#) 섹션을 참조하세요. 연결할 추가 네트워크 연결을 선택하기 위해 `connectionName` 파라미터를 사용합니다.

```

glueContext.write_dynamic_frame.from_options(
    connection_type="snowflake",

```

```

connection_options={
  "connectionName": "connectionName",
  "dbtable": "tableName",
  "sfDatabase": "databaseName",
  "sfSchema": "schemaName",
  "sfWarehouse": "warehouseName",
},
)

```

Snowflake 연결 옵션 참조

Snowflake 연결 유형에서는 다음과 같은 연결 옵션을 사용합니다.

이 섹션의 일부 파라미터는 데이터 카탈로그 연결(sfUrl, sfUser, sfPassword)에서 검색할 수 있으며, 이 경우 사용자가 해당 파라미터를 제공하지 않아도 됩니다. connectionName 파라미터를 제공하여 이 작업을 수행할 수 있습니다.

이 섹션의 일부 파라미터는 AWS Secrets Manager 보안 암호(sfUser, sfPassword)에서 검색할 수 있으며, 이 경우 사용자가 해당 파라미터를 제공하지 않아도 됩니다. 보안 암호는 sfUser 및 sfPassword 키 아래에서 해당 콘텐츠를 제공해야 합니다. secretId 파라미터를 제공하여 이 작업을 수행할 수 있습니다.

다음 파라미터는 보통 Snowflake에 연결할 때 사용됩니다.

- sfDatabase - Snowflake에 사용자 기본값이 설정되어 있지 않은 경우 필요합니다. 읽기 및 쓰기에 사용됩니다. 연결 후 세션에 사용할 데이터베이스입니다.
- sfSchema - Snowflake에 사용자 기본값이 설정되어 있지 않은 경우 필요합니다. 읽기 및 쓰기에 사용됩니다. 연결 후 세션에 사용할 스키마입니다.
- sfWarehouse - Snowflake에 사용자 기본값이 설정되어 있지 않은 경우 필요합니다. 읽기 및 쓰기에 사용됩니다. 연결 후 세션에 사용할 기본 가상 웨어하우스입니다.
- sfRole - Snowflake에 사용자 기본값이 설정되어 있지 않은 경우 필요합니다. 읽기 및 쓰기에 사용됩니다. 연결 후 세션에 사용할 기본 보안 암호 역할입니다.
- sfUrl -- (필수) 읽기 및 쓰기에 사용됩니다. 다음 `account_identifier.snowflakecomputing.com` 형식으로 계정의 호스트 이름을 지정합니다. 계정 식별자에 대한 자세한 내용은 Snowflake 설명서의 [Account Identifiers](#)를 참조하세요.
- sfUser -- (필수) 읽기 및 쓰기에 사용됩니다. Snowflake 사용자의 로그인 이름입니다.
- sfPassword - (pem_private_key가 제공되지 않은 경우 필수) 읽기/쓰기에 사용됩니다. Snowflake 사용자의 암호입니다.

- `dbtable` - 전체 테이블에 대한 작업을 수행할 때 필요합니다. 읽기 및 쓰기에 사용됩니다. 읽을 테이블 또는 데이터를 쓸 테이블의 이름입니다. 읽을 때 모든 열과 레코드가 검색됩니다.
- `pem_private_key` - 읽기/쓰기에 사용됩니다. 암호화되지 않은 b64 인코딩된 프라이빗 키 문자열입니다. Snowflake 사용자의 프라이빗 키입니다. 일반적으로 이를 PEM 파일에서 복사합니다. 자세한 내용은 Snowflake 설명서의 [키 페어 인증 & 키 페어 순환](#)을 참조하세요.
- `query` - 쿼리로 읽을 때 필요합니다. 읽기에 사용됩니다. 실행할 정확한 쿼리(SELECT 문)입니다.

다음 옵션은 Snowflake에 연결하는 과정에서 특정 동작을 구성하는 데 사용됩니다.

- `preactions` - 읽기/쓰기에 사용됩니다. 유효한 값: 세미콜론으로 구분된 SQL 문 목록(문자열). SQL 문은 AWS Glue와 Snowflake 사이에서 데이터가 전송되기 전에 실행됩니다. 명령문에 %s가 포함된 경우 %s는 작업에서 참조하는 테이블 이름으로 바뀝니다.
- `postactions` - 읽기/쓰기에 사용됩니다. SQL 문은 AWS Glue와 Snowflake 사이에서 데이터가 전송된 후에 실행됩니다. 명령문에 %s가 포함된 경우 %s는 작업에서 참조하는 테이블 이름으로 바뀝니다.
- `autopushdown` - 기본값: "on". 유효한 값: "on", "off". 이 파라미터는 자동 쿼리 푸시다운의 활성화 여부를 제어합니다. 푸시다운이 활성화된 경우 Spark에서 쿼리를 실행할 때 쿼리의 일부를 Snowflake 서버로 '푸시다운'할 수 있으면 해당 쿼리가 푸시다운됩니다. 이렇게 하면 일부 쿼리의 성능이 향상됩니다. 쿼리의 푸시다운 여부에 대한 자세한 내용은 Snowflake 설명서의 [Pushdown](#)을 참조하세요.

또한 Snowflake Spark 커넥터에서 사용할 수 있는 일부 옵션이 AWS Glue에서 지원될 수도 있습니다. Snowflake Spark 커넥터에서 사용할 수 있는 옵션에 대한 자세한 내용은 Snowflake 설명서의 [Setting Configuration Options for the Connector](#)를 참조하세요.

Snowflake 커넥터 제한

AWS Glue for Spark를 사용하여 Snowflake에 연결하는 경우 다음과 같은 제한 사항이 적용됩니다.

- 이 커넥터는 작업 북마크를 지원하지 않습니다. 작업 북마크에 대한 자세한 내용을 알아보려면 [the section called “처리된 데이터를 작업 북마크로 추적”](#)을(를) 참조하세요.
- 이 커넥터는 `create_dynamic_frame.from_catalog` 및 `write_dynamic_frame.from_catalog` 메서드를 사용하여 AWS Glue 데이터 카탈로그의 테이블을 통한 Snowflake 읽기 및 쓰기를 지원하지 않습니다.
- 이 커넥터는 사용자 및 암호 이외의 보안 인증으로 Snowflake에 연결하는 것을 지원하지 않습니다.
- 이 커넥터는 스트리밍 작업 내에서 지원되지 않습니다.

- 이 커넥터는 정보를 검색할 때(예: query 파라미터 사용) SELECT 문 기반 쿼리를 지원합니다. 다른 종류의 쿼리(예: SHOW, DESC 또는 DML 문)는 지원되지 않습니다.
- Snowflake는 Snowflake 클라이언트를 통해 제출되는 쿼리 텍스트(즉, SQL 문)의 크기를 명령문당 1MB로 제한합니다. 자세한 내용은 [Limits on Query Text Size](#)를 참조하세요.

Teradata Vantage 연결

AWS Glue for Spark를 이용하여 AWS Glue 4.0 이상 버전에서 Teradata Vantage의 테이블에서 읽고 쓸 수 있습니다. SQL 쿼리를 사용하여 Teradata에서 읽을 내용을 정의할 수 있습니다. AWS Glue 연결을 통해 AWS Secrets Manager에 저장된 사용자 이름 및 암호 보안 인증 정보를 사용하여 Teradata에 연결할 수 있습니다.

Teradata에 대한 자세한 내용은 [Teradata 설명서](#)를 참조하십시오.

Teradata 연결 구성

AWS Glue에서 Teradata에 연결하려면 Teradata 보안 인증 정보를 생성하여 AWS Secrets Manager 암호에 저장한 다음 해당 암호를 AWS Glue Teradata 연결에 연결해야 합니다. Teradata 인스턴스가 Amazon VPC에 있는 경우 AWS Glue Teradata 연결에 네트워킹 옵션도 제공해야 합니다.

AWS Glue에서 Teradata에 연결하려면 몇 가지 전제 조건이 필요할 수 있습니다.

- Amazon VPC를 통해 Teradata 환경에 액세스하는 경우, AWS Glue 작업이 Teradata 환경과 통신할 수 있도록 Amazon VPC를 구성하십시오. 퍼블릭 인터넷을 통해 Teradata 환경에 액세스하는 것은 권장하지 않습니다.

Amazon VPC에서 AWS Glue가 작업을 실행하는 동안 사용할 VPC, 서브넷 및 보안 그룹을 식별하거나 생성합니다. 또한 Amazon VPC가 Teradata 인스턴스와 이 위치 간의 네트워크 트래픽을 허용하도록 구성되어 있는지 확인해야 합니다. 작업을 수행하려면 Teradata 클라이언트 포트와 TCP 연결을 설정해야 합니다. Teradata 포트에 대한 자세한 내용은 [Teradata 설명서](#)를 참조하십시오.

네트워크 레이아웃에 따라 보안 VPC 연결에는 Amazon VPC 및 기타 네트워킹 서비스를 변경해야 할 수 있습니다. AWS 연결에 대한 자세한 내용은 Teradata 설명서의 [AWS연결 옵션](#)을 참조하십시오.

AWS Glue Teradata 연결을 구성하는 방법:

1. Teradata 구성에서 AWS Glue가 *teradataUser* 및 *teradataPassword*와 연결할 사용자 및 암호를 식별하거나 생성합니다. 자세한 내용은 Teradata 설명서의 [Vantage 보안 개요](#)를 참조하십시오.
2. AWS Secrets Manager에서 Teradata 보안 인증 정보를 사용하여 보안 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *teradataUsername*이 포함된 키 user에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *teradataPassword*가 포함된 키 password에 대한 페어를 생성합니다.
3. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 Teradata를 선택합니다.
 - JDBC URL을 제공할 때는 인스턴스의 URL을 제공하십시오. 또한 JDBC URL에 심표로 구분된 특정 연결 매개변수를 하드코딩할 수 있습니다. URL의 형식: `jdbc:teradata://teradataHostname/ParameterName=ParameterValue,ParameterName`
지원되는 URL 파라미터는 다음과 같습니다.
 - DATABASE - 기본으로 액세스하는 호스트의 데이터베이스 이름입니다.
 - DBS_PORT - 비표준 포트에서 실행할 때 사용되는 데이터베이스 포트입니다.
 - 보안 인증 정보 유형을 선택할 때에는 AWS Secrets Manager을 선택한 다음 AWS보안 암호를 *secretName*으로 설정합니다.
4. 다음과 같은 상황에서는 추가 구성이 필요할 수도 있습니다.
 - Amazon VPC에서 AWS에 호스팅된 Teradata 인스턴스의 경우
 - Teradata 보안 인증 정보를 정의하는 AWS Glue 연결에 Amazon VPC 연결 정보를 제공해야 합니다. 연결을 만들거나 업데이트할 때 네트워크 옵션에서 VPC, 서브넷 및 보안 그룹을 설정합니다.

AWS Glue Teradata 연결을 생성한 후에는 연결 방법을 호출하기 전에 다음 단계를 수행해야 합니다.

- AWS Glue 작업과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여합니다.

- AWS Glue 작업 구성에서 추가 네트워크 연결로 *connectionName*을 제공합니다.

Teradata에서 읽는 중

사전 조건:

- 읽으려는 Teradata 테이블. 테이블 이름, *tableName*이 필요합니다.
- 인증 정보를 제공하도록 구성된 AWS Glue Teradata 연결입니다. 인증 정보를 구성하려면 Teradata에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.

예:

```
teradata_read_table = glueContext.create_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName"
    }
)
```

SELECT SQL 쿼리에 반환되는 결과를 필터링하는 쿼리를 제공할 수도 있습니다. *query*을 구성해야 합니다.

예:

```
teradata_read_query = glueContext.create_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "query": "query"
    }
)
```

Teradata 테이블에 쓰기

전제 조건: 쓰고 싶은 Teradata 테이블, *tableName*. 연결 방법을 호출하기 전에 테이블을 생성해야 합니다.

예:

```

teradata_write = glueContext.write_dynamic_frame.from_options(
    connection_type="teradata",
    connection_options={
        "connectionName": "connectionName",
        "dbtable": "tableName"
    }
)

```

Teradata 연결 옵션 참조

- `connectionName` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 연결 방법에 인증 및 네트워킹 정보를 제공하도록 구성된 AWS Glue Teradata 연결의 이름입니다.
- `dbtable` - 쓰기 시 필수, `query`가 제공되지 않는 한 읽기 전용. 읽기 및 쓰기에 사용됩니다. 연결 방법이 상호 작용할 테이블의 이름.
- `query` — 읽기에 사용됩니다. Teradata에서 읽을 때 검색해야 하는 내용을 정의하는 SELECT SQL 쿼리입니다.

수직 연결

AWS Glue for Spark를 사용하여 AWS Glue 4.0 이상 버전에서 Vertica의 테이블에서 읽고 쓸 수 있습니다. SQL 쿼리를 사용하여 Vertica에서 읽을 내용을 정의할 수 있습니다. AWS Glue 연결을 통해 AWS Secrets Manager에 저장된 사용자 이름 및 암호 보안 인증 정보를 사용하여 Vertica에 연결할 수 있습니다.

Vertica에 대한 자세한 내용은 [Vertica 설명서](#)를 참조하십시오.

Vertica 연결 구성

AWS Glue에서 Vertica에 연결하려면 AWS Secrets Manager 보안 암호에서 Vertica 보안 인증 정보를 생성하고 저장한 다음 해당 보안 암호를 Vertica AWS Glue 연결에 연결해야 합니다. Vertica 인스턴스가 Amazon VPC에 있는 경우 AWS Glue Vertica 연결에 네트워킹 옵션도 제공해야 합니다. 데이터베이스에서 읽고 쓸 때 임시 스토리지로 이용할 Amazon S3 버킷이나 폴더가 필요합니다.

AWS Glue에서 Vertica에 연결하려면 몇 가지 필수 조건이 필요합니다.

- `tempS3Path`에서 참조하여 데이터베이스에서 읽고 쓸 때 임시 스토리지로 사용하는 Amazon S3 버킷 또는 폴더.

Note

AWS Glue 작업 데이터 미리 보기의 Vertica를 사용하는 경우 임시 파일은 *tempS3Path*에서 자동으로 제거되지 않을 수 있습니다. 임시 파일을 제거하려면 데이터 미리 보기 창에서 세션 종료를 선택하여 데이터 미리 보기 세션을 바로 종료합니다.

데이터 미리 보기 세션이 바로 종료되도록 보장할 수 없는 경우 이전 데이터를 제거하도록 Amazon S3 수명 주기 구성을 설정하는 것이 좋습니다. 최대 작업 런타임에 여백을 더한 기준으로 49시간이 지난 데이터는 제거하는 것이 좋습니다. Amazon S3 수명 주기 구성에 대한 자세한 내용은 Amazon S3 설명서에서 [스토리지 수명 주기 관리](#)를 참조하십시오.

- Amazon S3 경로에 대한 적절한 권한이 있는 IAM 정책을 AWS Glue 작업 역할과 연결할 수 있습니다.
- Vertica 인스턴스가 Amazon VPC에 있는 경우, 퍼블릭 인터넷을 통과하는 트래픽 없이 AWS Glue 작업이 Vertica 인스턴스와 통신할 수 있도록 Amazon VPC를 구성하십시오.

Amazon VPC에서 AWS Glue가 작업을 실행하는 동안 사용할 VPC, 서브넷 및 보안 그룹을 식별하거나 생성합니다. 또한 Vertica 인스턴스와 이 위치 간의 네트워크 트래픽을 허용하도록 Amazon VPC를 구성해야 합니다. 작업을 수행하려면 Vertica 클라이언트 포트(기본값 5433)와의 TCP 연결을 설정해야 합니다. 네트워크 레이아웃에 따라 보안 그룹 규칙, 네트워크 ACL, NAT 게이트웨이 및 피어링 연결을 변경해야 할 수도 있습니다.

그런 다음 Vertica와 함께 사용할 AWS Glue를 구성할 수 있습니다.

Vertica에 대한 연결 구성하는 방법:

1. AWS Secrets Manager에서 Vertica 보안 인증 정보, *verticaUsername*, *verticaPassword*을 사용하여 암호를 생성합니다. Secrets Manager에서 보안 암호를 생성하려면 AWS Secrets Manager 설명서의 [Create an AWS Secrets Manager secret](#)에서 제공하는 자습서를 따릅니다. 보안 암호를 생성한 후에는 다음 단계를 위해 보안 암호 이름, *secretName*을 유지합니다.
 - 키/값 페어를 선택하면 값 *verticaUsername*이 포함된 키 user에 대한 페어를 생성합니다.
 - 키/값 페어를 선택하면 값 *verticaPassword*이 포함된 키 password에 대한 페어를 생성합니다.
2. AWS Glue 콘솔에서 [the section called “AWS Glue 연결 추가”](#)의 단계에 따라 연결을 생성합니다. 연결을 생성한 후에는 다음 단계를 위해 연결 이름, *connectionName*을 유지합니다.
 - 연결 유형을 선택할 때 Vertica를 선택합니다.

- Vertica 호스트를 선택할 때 Vertica 설치의 호스트 이름을 제공합니다.
 - Vertica 포트를 선택하면 해당 포트를 통해 Vertica 설치를 사용할 수 있습니다.
 - AWS 보안 암호를 선택할 때 *secretName*을 입력합니다.
3. 다음과 같은 상황에서는 추가 구성이 필요할 수도 있습니다.
- Amazon VPC에서 AWS에 호스팅된 Vertica 인스턴스의 경우
 - Vertica 보안 인증 정보를 정의하는 Amazon VPC 연결 정보를 AWS Glue 연결에 제공하십시오. 연결을 만들거나 업데이트할 때 네트워크 옵션에서 VPC, 서브넷 및 보안 그룹을 설정합니다.

AWS Glue Vertica 연결을 생성한 후에는 연결 방법을 호출하기 전에 다음 단계를 수행해야 합니다.

- AWS Glue 작업과 권한과 연결된 IAM 역할을 *tempS3Path*에 부여합니다.
- AWS Glue 작업과 연결된 IAM 역할에 *secretName*을 읽을 수 있는 권한을 부여합니다.
- AWS Glue 작업 구성에서 추가 네트워크 연결로 *connectionName*을 제공합니다.

Vertica에서 읽는 중

사전 조건:

- 읽으려는 Vertica 테이블. Vertica 데이터베이스 이름인 *dbName*과 테이블 이름인 *tableName*이 필요합니다.
- 인증 정보를 제공하도록 구성된 AWS Glue Vertica 연결. 인증 정보를 구성하려면 앞 절차인 Vertica에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.
- 앞서 언급한 임시 스토리지로 사용할 Amazon S3 버킷 또는 폴더. *tempS3Path*라는 이름이 필요합니다. s3a 프로토콜을 사용하여 이 위치에 연결해야 합니다.

예:

```
dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="vertica",
    connection_options={
        "connectionName": "connectionName",
        "staging_fs_url": "s3a://tempS3Path",
        "db": "dbName",
```

```

        "table": "tableName",
    }
)

```

SELECT SQL 쿼리를 제공하여 DynamicFrame에 반환되는 결과를 필터링할 수도 있고 여러 테이블의 데이터 세트에 액세스할 수도 있습니다.

예:

```

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="vertica",
    connection_options={
        "connectionName": "connectionName",
        "staging_fs_url": "s3a://tempS3Path",
        "db": "dbName",
        "query": "select * FROM tableName",
    },
)

```

Vertica 테이블에 쓰기

이 예제에서는 기존 DynamicFrame, *dynamicFrame*의 정보를 Vertica에 씁니다. 테이블에 이미 정보가 있는 경우 AWS Glue는 DynamicFrame의 데이터를 추가합니다.

사전 조건:

- 쓰려는 현재 또는 원하는 테이블 이름, *tableName*. 또한 해당하는 Vertica 데이터베이스 이름인 *dbName*도 필요합니다.
- 인증 정보를 제공하도록 구성된 AWS Glue Vertica 연결. 인증 정보를 구성하려면 앞 절차인 Vertica에 대한 연결을 구성하는 방법의 단계를 완료하십시오. AWS Glue 연결의 이름인 *connectionName*이 필요합니다.
- 앞서 언급한 임시 스토리지로 사용할 Amazon S3 버킷 또는 폴더. *tempS3Path*라는 이름이 필요합니다. s3a 프로토콜을 사용하여 이 위치에 연결해야 합니다.

예:

```

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="vertica",
    connection_options={

```



```

    "connectionName": "connectionName",
    "staging_fs_url": "s3a://tempS3Path",
    "db": "dbName",
    "table": "tableName",
  }
)

```

Vertica 연결 옵션 참조

- `connectionName` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 연결 방법에 인증 및 네트워킹 정보를 제공하도록 구성된 AWS Glue Vertica 연결의 이름입니다.
- `db` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 연결 방법이 상호 작용하는 Vertica의 데이터베이스 이름.
- `dbSchema` - 테이블을 식별하는 데 필요한 경우 필요합니다. 읽기 및 쓰기에 사용됩니다. 기본값: `public`. 연결 방법이 상호 작용할 스키마의 이름.
- `table` - 쓰기 시 필수, `query`가 제공되지 않는 한 읽기 전용. 읽기 및 쓰기에 사용됩니다. 연결 방법이 상호 작용할 테이블의 이름.
- `query` — 읽기에 사용됩니다. Teradata에서 읽을 때 검색해야 하는 내용을 정의하는 SELECT SQL 쿼리입니다.
- `staging_fs_url` — 필수입니다. 읽기 및 쓰기에 사용됩니다. 유효한 값: s3a URL. 임시 스토리지로 사용할 Amazon S3 버킷 또는 폴더의 URL입니다.

사용자 정의 및 AWS Marketplace 연결 유형 값

여기에는 다음이 포함됩니다.

- `"connectionType": "marketplace.athena"`: Amazon Athena 데이터 스토어에 대한 연결을 지정합니다. 연결은 AWS Marketplace의 커넥터를 사용합니다.
- `"connectionType": "marketplace.spark"`: Apache Spark 데이터 스토어에 대한 연결을 지정합니다. 연결은 AWS Marketplace의 커넥터를 사용합니다.
- `"connectionType": "marketplace.jdbc"`: JDBC 데이터 스토어에 대한 연결을 지정합니다. 연결은 AWS Marketplace의 커넥터를 사용합니다.
- `"connectionType": "custom.athena"`: Amazon Athena 데이터 스토어에 대한 연결을 지정합니다. 연결은 AWS Glue Studio에 업로드하는 사용자 지정 커넥터를 사용합니다.
- `"connectionType": "custom.spark"`: Apache Spark 데이터 스토어에 대한 연결을 지정합니다. 연결은 AWS Glue Studio에 업로드하는 사용자 지정 커넥터를 사용합니다.

- "connectionType": "custom.jdbc": JDBC 데이터 스토어에 대한 연결을 지정합니다. 연결은 AWS Glue Studio에 업로드하는 사용자 지정 커넥터를 사용합니다.

custom.jdbc 또는 marketplace.jdbc 유형에 대한 연결 옵션

- className - 문자열, 필수, 드라이버 클래스 이름입니다.
- connectionName - 문자열, 필수, 커넥터와 연결된 연결 이름입니다.
- url- 문자열, 필수, 데이터 원본에 대한 연결을 구축하는 데 사용되는 자리 표시자({})가 있는 JDBC URL입니다. 자리 표시자 \${secretKey}는 AWS Secrets Manager에서 같은 이름의 보안 암호로 대체됩니다. URL 구성에 대한 자세한 내용은 데이터 스토어 설명서를 참조하세요.
- secretId 또는 user/password - 문자열, 필수, URL에 대한 자격 증명을 검색하는 데 사용됩니다.
- dbTable 또는 query - 문자열, 필수, 데이터를 가져올 테이블 또는 SQL 쿼리입니다. dbTable 또는 query를 지정할 수 있지만 둘 다 함께 지정할 수는 없습니다.
- partitionColumn- 문자열, 선택 사항, 파티셔닝에 사용되는 정수 열의 이름입니다. 이 옵션은 lowerBound, upperBound 및 numPartitions에 포함되는 경우에만 작동합니다. 이 옵션은 Spark SQL JDBC 리더에서와 같은 방식으로 작동합니다. 자세한 내용은 Apache Spark SQL, DataFrames and Datasets Guide의 [JDBC To Other Databases](#)를 참조하세요.

lowerBound 및 upperBound 값은 테이블의 행을 필터링하는 것이 아니라 파티션 스트라이드를 결정하는 데 사용됩니다. 테이블의 모든 행이 분할되어 반환됩니다.

Note

테이블 이름 대신 쿼리를 사용하는 경우 쿼리가 지정된 분할 조건에서 작동하는지 확인해야 합니다. 예:

- 쿼리 포맷이 "SELECT col1 FROM table1"이면 파티션 열을 사용하는 쿼리 끝에 WHERE 절을 추가하여 쿼리를 테스트합니다.
- 쿼리 포맷이 "SELECT col1 FROM table1 WHERE col2=val"이면 AND와 파티션 열을 사용하는 표현식으로 WHERE 절을 확장하여 쿼리를 테스트합니다.

- lowerBound - 정수, 선택 사항, 파티션 스트라이드를 결정하는 데 사용되는 partitionColumn의 최소값입니다.
- upperBound - 정수, 선택 사항, 파티션 스트라이드를 결정하는 데 사용되는 partitionColumn의 최대값입니다.

- numPartitions- 정수, 선택 사항, 파티션 수입니다. 이 값은 lowerBound(포함) 및 upperBound(배타)와 함께 partitionColumn을 분할하는 데 사용되는 생성된 WHERE 절 표현에 대한 파티션 스트라이드를 형성합니다.

Important

파티션이 너무 많으면 외부 데이터베이스 시스템에 문제가 발생할 수 있으므로 파티션 수에 주의합니다.

- filterPredicate - 문자열, 선택 사항, 소스에서 데이터를 필터링하기 위한 추가 조건 절입니다. 예:

```
BillingCity='Mountain View'
```

테이블 이름 대신 쿼리를 사용하는 경우 쿼리가 지정된 filterPredicate에서 작동하는지 검증해야 합니다. 예:

- 쿼리 포맷이 "SELECT col1 FROM table1"이면 필터 조건자를 사용하는 쿼리 끝에 WHERE 절을 추가하여 쿼리를 테스트합니다.
- 쿼리 포맷이 "SELECT col1 FROM table1 WHERE col2=val"이면 AND로 WHERE 절을 확장하고 필터 조건자를 사용하는 표현식을 사용하여 쿼리를 테스트합니다.
- dataTypeMapping - 딕셔너리, 선택 사항, JDBC 데이터 유형에서 Glue 데이터 유형으로의 매핑을 구축하는 사용자 정의 데이터 유형 매핑입니다. 예를 들어 "dataTypeMapping": {"FLOAT": "STRING"} 옵션은 드라이버의 ResultSet.getString() 메서드를 호출하여 JDBC 유형 FLOAT의 데이터 필드를 Java String 유형으로 매핑하고 이를 AWS Glue 레코드를 구축하는 데 사용합니다. ResultSet 객체는 각 드라이버에 의해 구현되므로 동작은 사용하는 드라이버에 따라 다릅니다. 드라이버가 변환을 수행하는 방법을 이해하려면 JDBC 드라이버에 대한 설명서를 참조하세요.
- 현재 지원되는 AWS Glue 데이터 유형은 다음과 같습니다.
 - DATE
 - STRING
 - TIMESTAMP
 - INT
 - FLOAT
 - LONG
 - BIGDECIMAL

- BYTE
- SHORT
- DOUBLE

지원되는 JDBC 데이터 유형은 [Java8 java.sql.types](#)입니다.

기본 데이터 유형 매핑(JDBC에서 AWS Glue로)은 다음과 같습니다.

- DATE -> DATE
- VARCHAR -> STRING
- CHAR -> STRING
- LONGNVARCHAR -> STRING
- TIMESTAMP -> TIMESTAMP
- INTEGER -> INT
- FLOAT -> FLOAT
- REAL -> FLOAT
- BIT -> BOOLEAN
- BOOLEAN -> BOOLEAN
- BIGINT -> LONG
- DECIMAL -> BIGDECIMAL
- NUMERIC -> BIGDECIMAL
- TINYINT -> SHORT
- SMALLINT -> SHORT
- DOUBLE -> DOUBLE

옵션 `dataTypeMapping`과 함께 사용자 정의 데이터 유형 매핑을 사용하는 경우 기본 데이터 유형 매핑을 재정의할 수 있습니다. `dataTypeMapping` 옵션에 나열된 JDBC 데이터 유형만 영향을 받습니다. 기본 매핑은 다른 모든 JDBC 데이터 유형에 사용됩니다. 필요한 경우 추가 JDBC 데이터 유형에 대한 매핑을 추가할 수 있습니다. JDBC 데이터 유형이 기본 매핑이나 사용자 지정 매핑에 포함되지 않은 경우 데이터 유형은 기본값으로 AWS Glue STRING 데이터 유형으로 변환됩니다.

다음 Python 코드 예제는 AWS Marketplace JDBC 드라이버를 사용하여 JDBC 데이터베이스에서 읽는 방법을 보여줍니다. 데이터베이스에서 읽고 S3 위치에 쓰는 방법을 보여줍니다.

```
import sys
```

```

from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [connection_type = "marketplace.jdbc", connection_options =
  {"dataTypeMapping":{"INTEGER":"STRING"},"upperBound":"200","query":"select id,
    name, department from department where id < 200","numPartitions":"4",
    "partitionColumn":"id","lowerBound":"0","connectionName":"test-connection-
jdbc"},
  transformation_ctx = "DataSource0"]
## @return: DataSource0
## @inputs: []
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
  "marketplace.jdbc", connection_options = {"dataTypeMapping":{"INTEGER":"STRING"},
  "upperBound":"200","query":"select id, name, department from department where
  id < 200","numPartitions":"4","partitionColumn":"id","lowerBound":"0",
  "connectionName":"test-connection-jdbc"}, transformation_ctx = "DataSource0")
## @type: ApplyMapping
## @args: [mappings = [("department", "string", "department", "string"), ("name",
"string",
  "name", "string"), ("id", "int", "id", "int")], transformation_ctx =
"Transform0"]
## @return: Transform0
## @inputs: [frame = DataSource0]
Transform0 = ApplyMapping.apply(frame = DataSource0, mappings = [("department",
"string",
  "department", "string"), ("name", "string", "name", "string"), ("id", "int",
"id", "int")],
  transformation_ctx = "Transform0")
## @type: DataSink
## @args: [connection_type = "s3", format = "json", connection_options = {"path":
  "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
## @return: DataSink0

```

```
## @inputs: [frame = Transform0]
DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
    connection_type = "s3", format = "json", connection_options = {"path":
    "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
job.commit()
```

custom.athena 또는 marketplace.athena 유형에 대한 연결 옵션

- `className` - 문자열, 필수, 드라이버 클래스 이름입니다. Athena-CloudWatch 커넥터를 사용하는 경우 이 파라미터 값은 클래스 이름의 접두사(예: "com.amazonaws.athena.connectors")입니다. Athena-CloudWatch 커넥터는 메타데이터 핸들러와 레코드 핸들러의 두 가지 클래스로 구성됩니다. 여기에 공통 접두사를 제공하면 API가 해당 접두사를 기반으로 올바른 클래스를 로드합니다.
- `tableName` - 문자열, 필수, 읽을 CloudWatch 로그 스트림의 이름입니다. 이 코드 조각은 특수 보기 이름 `all_log_streams`을 사용합니다. 즉, 반환된 동적 데이터 프레임에는 로그 그룹의 모든 로그 스트림 데이터가 포함됩니다.
- `schemaName` - 문자열, 필수, 읽을 CloudWatch 로그 그룹의 이름입니다. 예를 들면 `/aws-glue/jobs/output`입니다.
- `connectionName` - 문자열, 필수, 커넥터와 연결된 연결 이름입니다.

이 커넥터에 대한 추가 옵션은 GitHub의 [Amazon Athena CloudWatch 커넥터 README](#)를 참조하세요.

다음 Python 코드 예제는 AWS Marketplace 커넥터를 사용하여 Athena 데이터 스토어에서 읽는 방법을 보여줍니다. Athena에서 읽고 S3 위치에 쓰는 방법을 보여줍니다.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
```

```

## @args: [connection_type = "marketplace.athena", connection_options =
  {"tableName":"all_log_streams","schemaName":"/aws-glue/jobs/output",
  "connectionName":"test-connection-athena"}, transformation_ctx = "DataSource0"]
## @return: DataSource0
## @inputs: []
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
  "marketplace.athena", connection_options = {"tableName":"all_log_streams",,
  "schemaName":"/aws-glue/jobs/output","connectionName":
  "test-connection-athena"}, transformation_ctx = "DataSource0")
## @type: ApplyMapping
## @args: [mappings = [("department", "string", "department", "string"), ("name",
"string",
  "name", "string"), ("id", "int", "id", "int")], transformation_ctx =
"Transform0"]
## @return: Transform0
## @inputs: [frame = DataSource0]
Transform0 = ApplyMapping.apply(frame = DataSource0, mappings = [("department",
"string",
  "department", "string"), ("name", "string", "name", "string"), ("id", "int",
"id", "int")],
  transformation_ctx = "Transform0")
## @type: DataSink
## @args: [connection_type = "s3", format = "json", connection_options = {"path":
  "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
## @return: DataSink0
## @inputs: [frame = Transform0]
DataSink0 = glueContext.write_dynamic_frame.from_options(frame = Transform0,
  connection_type = "s3", format = "json", connection_options = {"path":
  "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
job.commit()

```

custom.spark 또는 marketplace.spark 유형에 대한 연결 옵션

- className - 문자열, 필수, 커넥터 클래스 이름입니다.
- secretId - 문자열, 선택 사항, 커넥터 연결에 대한 자격 증명을 검색하는 데 사용됩니다.
- connectionName - 문자열, 필수, 커넥터와 연결된 연결 이름입니다.
- 다른 옵션은 데이터 스토어에 따라 다릅니다. 예를 들어 OpenSearch 구성 옵션은 [Apache Hadoop 용 Elasticsearch](#) 설명서에 설명된 대로 접두사 es로 시작합니다. Snowflake에 대한 Spark 연결은 [Connecting to Snowflake 가이드의 Using the Spark Connector](#)에 설명된 대로 sfUser 및 sfPassword와 같은 옵션을 사용합니다.

다음 Python 코드 예제는 marketplace.spark 연결을 사용하여 OpenSearch 데이터 스토어에서 읽는 방법을 보여줍니다.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [connection_type = "marketplace.spark", connection_options =
{"path":"test",
  "es.nodes.wan.only":"true","es.nodes":"https://<AWS endpoint>",
  "connectionName":"test-spark-es","es.port":"443"}, transformation_ctx =
"DataSource0"]
## @return: DataSource0
## @inputs: []
DataSource0 = glueContext.create_dynamic_frame.from_options(connection_type =
  "marketplace.spark", connection_options = {"path":"test","es.nodes.wan.only":
  "true","es.nodes":"https://<AWS endpoint>","connectionName":
  "test-spark-es","es.port":"443"}, transformation_ctx = "DataSource0")
## @type: DataSink
## @args: [connection_type = "s3", format = "json", connection_options = {"path":
  "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0"]
## @return: DataSink0
## @inputs: [frame = DataSource0]
DataSink0 = glueContext.write_dynamic_frame.from_options(frame = DataSource0,
  connection_type = "s3", format = "json", connection_options = {"path":
  "s3://<S3 path>/", "partitionKeys": []}, transformation_ctx = "DataSink0")
job.commit()
```


일반 옵션

이 섹션의 옵션은 `connection_options`로 제공되지만 특별히 하나의 커넥터에만 적용되지는 않습니다.

다음 파라미터는 보통 북마크를 구성할 때 사용됩니다. Amazon S3 또는 JDBC 워크플로에 적용될 수도 있습니다. 자세한 내용은 [the section called “작업 북마크 사용”](#) 단원을 참조하십시오.

- `jobBookmarkKeys` - 열 이름의 배열입니다.
- `jobBookmarkKeysSortOrder` - 정렬 순서에 따라 값을 비교하는 방법을 정의하는 문자열입니다. 유효한 값: "asc", "desc".
- `useS3ListImplementation` - Amazon S3 버킷 콘텐츠를 나열할 때 메모리 성능을 관리하는 데 사용됩니다. 자세한 내용은 [Optimize memory management in AWS Glue](#)를 참조하세요.

AWS Glue for Spark에서 입력 및 출력의 데이터 형식 옵션

이 페이지는 AWS Glue for Spark에서 지원하는 데이터 형식에 대한 기능 지원 및 구성 파라미터에 대한 정보를 제공합니다. 이 정보의 사용 및 적용 가능성에 대한 설명은 다음을 참조하세요.

AWS Glue의 데이터 형식 전반에 걸친 기능 지원

각 데이터 형식은 다른 AWS Glue 기능을 지원할 수 있습니다. 다음 공통 기능은 해당 형식 유형에 따라 지원되거나 지원되지 않을 수 있습니다. 요구 사항을 충족하기 위해 기능을 활용하는 방법을 이해하려면 데이터 형식에 대한 설명서를 참조하세요.

읽기	AWS Glue는 커넥터와 같은 추가 리소스 없이 이 데이터 형식을 인식하고 해석할 수 있습니다.
쓰기	AWS Glue는 추가 리소스 없이 이 형식으로 데이터를 쓸 수 있습니다. 다른 Spark 환경에서와 같이 작업에 타사 라이브러리를 포함하고 표준 Apache Spark 기능을 사용하여 데이터를 작성할 수 있습니다. 라이브러리를 포함한 자세한 내용은 the section called “Python 라이브러리” 단원을 참조하세요.
스트리밍 읽기	AWS Glue는 Apache Kafka, Amazon Managed Streaming for Apache Kafka 또는 Amazon Kinesis 메시지 스트림에서 이 데이터 형식을 인

식하고 해석할 수 있습니다. 스트림이 일관된 형식으로 데이터를 표시할 것으로 예상하므로 DataFrames 로 읽습니다.

작은 파일 그룹화 AWS Glue는 AWS Glue 변환을 수행할 때 각 노드로 전송되는 일괄 작업으로 파일을 그룹화할 수 있습니다. 이렇게 하면 많은 양의 작은 파일이 포함된 워크로드의 성능이 크게 향상될 수 있습니다. 자세한 내용은 [the section called “입력 파일 그룹화”](#) 단원을 참조하십시오.

작업 북마크 AWS Glue는 작업 북마크를 사용하여 작업 실행 전반에 걸쳐 동일한 데이터 세트에서 동일한 작업을 수행하는 변환의 진행 상황을 추적할 수 있습니다. 이렇게 하면 마지막 작업 실행 이후 새 데이터에 대해서만 작업을 수행해야 하는 데이터 세트와 관련된 워크로드의 성능을 향상시킬 수 있습니다. 자세한 내용은 [the section called “처리된 데이터를 작업 북마크로 추적”](#) 단원을 참조하십시오.

AWS Glue의 데이터 형식과 상호 작용하는 데 사용되는 파라미터

특정 AWS Glue 연결 유형은 여러 format 유형을 지원하므로 `GlueContext.write_dynamic_frame.from_options`와 같은 방법을 사용할 때 `format_options` 객체로 데이터 형식에 대한 정보를 지정해야 합니다.

- s3 - 자세한 내용은 AWS Glue: [S3 연결 파라미터](#)의 ETL 연결 유형 및 옵션을 참조하세요. 이 연결 유형(Python에서 [the section called “create_dynamic_frame_from_options”](#) 및 [the section called “write_dynamic_frame_from_options”](#), 그에 해당하는 스칼라 메서드 [the section called “getSourceWith포맷”](#) 및 [the section called “getSinkWith포맷”](#))을 지원하는 메서드에 대한 문서도 볼 수 있습니다.
- kinesis - 자세한 내용은 AWS Glue: [Kinesis 연결 파라미터](#)의 ETL 연결 유형 및 옵션을 참조하세요. 이 연결 유형([the section called “create_data_frame_from_options”](#) 및 그에 해당하는 스칼라 메서드 [the section called “createDataFrameFromOptions”](#))을 지원하는 메서드에 대한 문서도 볼 수 있습니다.

- kafka - 자세한 내용은 AWS Glue: [Kafka 연결 파라미터](#)의 ETL 연결 유형 및 옵션을 참조하세요. 이 연결 유형([the section called “create_data_frame_from_options”](#) 및 그에 해당하는 스칼라 메서드 [the section called “createDataFrameFromOptions”](#))을 지원하는 메서드에 대한 문서도 볼 수 있습니다.

일부 연결 유형은 `format_options`가 필요하지 않습니다. 예를 들어 일반적인 사용 과정에서 관계형 데이터베이스에 대한 JDBC 연결은 일관된 테이블형 데이터 형식으로 데이터를 검색하므로 JDBC 연결에서 읽을 때 `format_options`가 필요하지 않습니다.

Glue에서 데이터를 읽고 쓰는 일부 방법은 `format_options`가 필요하지 않습니다. 예를 들어, Glue 크롤러에서 `GlueContext.create_dynamic_frame.from_catalog` 및 AWS를 사용하는 경우, 크롤러는 데이터의 형태를 결정합니다. 크롤러를 사용할 때 AWS Glue 분류기는 데이터를 검사하여 데이터 형식을 표현하는 방법에 대한 현명한 결정을 내립니다. 그런 다음 AWS Glue ETL 스크립트 내에서 `GlueContext.create_dynamic_frame.from_catalog` 메서드로 데이터를 검색하는 데 사용할 수 있는 AWS Glue 데이터 카탈로그에 데이터 표현을 저장합니다. 크롤러를 사용하면 데이터 형식에 대한 정보를 수동으로 지정할 필요가 없습니다.

AWS Lake Formation 관리형 테이블에 액세스하는 작업의 경우 AWS Glue는 Lake Formation 관리형 테이블에서 지원되는 모든 포맷을 읽고 쓸 수 있도록 지원합니다. AWS Lake Formation 관리형 테이블에서 현재 지원되는 포맷 목록은 [AWS Lake Formation 개발자 가이드](#)의 관리형 테이블에 대한 참고 사항 및 제한 사항을 참조하세요.

Note

Apache Parquet 쓰기의 경우 AWS Glue ETL은 Dynamic Frames에 최적화된 사용자 지정 Parquet 라이터 유형에 대한 옵션을 지정하여 관리형 테이블에 쓰는 기능만을 지원합니다. parquet 포맷을 사용하는 관리형 테이블에 쓰는 경우 테이블 파라미터의 `true` 값과 함께 `useGlueParquetWriter` 키를 추가해야 합니다.

주제

- [AWS Glue에서 CSV 형식 사용](#)
- [AWS Glue에서 Parquet 형식 사용](#)
- [AWS Glue에서 XML 형식 사용](#)
- [AWS Glue에서 Avro 형식 사용](#)
- [AWS Glue에서 grokLog 형식 사용](#)
- [AWS Glue에서 Ion 형식 사용](#)

- [Glue에서 AWS JSON 포맷 사용하기](#)
- [AWS Glue에서 ORC 형식 사용](#)
- [AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용](#)
- [공유 구성 참조](#)

AWS Glue에서 CSV 형식 사용

AWS Glue는 소스에서 데이터를 검색하고 다양한 데이터 형식으로 저장 및 전송되는 대상에 데이터를 씁니다. 데이터가 CSV 데이터 형식으로 저장 또는 전송되는 경우 이 문서에서는 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능을 소개합니다.

AWS Glue는 CSV(쉼표로 분리된 값) 형식 사용만 지원합니다. 이 형식은 최소 행 기반 데이터 형식입니다. CSV는 표준을 엄격하게 준수하지 않는 경우가 많지만 [RFC 4180](#)과 [RFC 7111](#)에서 자세한 정보를 참조할 수 있습니다.

AWS Glue를 사용하여 Amazon S3와 스트리밍 소스에서 CSV를 읽을 수 있을 뿐만 아니라 Amazon S3에 CSV를 쓸 수 있습니다. S3에서 CSV 파일이 포함된 bzip 및 gzip 아카이브를 읽고 쓸 수 있습니다. 이 페이지에서 설명하는 구성 대신 [S3 연결 파라미터](#)에서 압축 동작을 구성할 수 있습니다.

다음 표에서는 CSV 형식 옵션을 지원하는 일반적인 AWS Glue 기능을 보여줍니다.

읽기	쓰기	스트리밍 읽기	작은 파일 그룹화	작업 복마크
지원	지원	지원	지원	지원

예: S3에서 CSV 파일 또는 폴더 읽기

사전 조건: 읽고자 하는 CSV 파일 또는 폴더에 대한 S3 경로(s3path)가 필요합니다.

구성: 함수 옵션에서 format="csv"를 지정합니다. connection_options에서 paths 키를 사용하여 s3path를 지정합니다. connection_options에서 리더와 S3가 상호 작용하는 방식을 구성할 수 있습니다. 자세한 내용은 AWS Glue에서 ETL 관련 연결 유형 및 옵션 참조: [S3 연결 파라미터](#) 리더에서 format_options의 CSV 파일을 해석하는 방법을 구성할 수 있습니다. 자세한 내용은 [CSV 구성 참조](#)를 참조하십시오.

다음 AWS Glue ETL 스크립트는 S3에서 CSV 파일 또는 폴더를 읽는 프로세스를 보여줍니다.

optimizePerformance 구성 키를 통해 사용자 지정 CSV 리더에 일반적인 워크플로우에 대한 성능 최적화가 제공됩니다. 이 리더가 워크로드에 적합한지 확인하려면 [the section called “최적화된 CSV 리더 사용”](#) 단원을 참조하십시오.

Python

이 예에서는 [create_dynamic_frame.from_options](#) 메서드를 사용합니다.

```
# Example: Read CSV from S3
# For show, we handle a CSV with a header row. Set the withHeader option.
# Consider whether optimizePerformance is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="csv",
    format_options={
        "withHeader": True,
        # "optimizePerformance": True,
    },
)
```

또한 스크립트(`pyspark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
dataFrame = spark.read\
    .format("csv")\
    .option("header", "true")\
    .load("s3://s3path")
```

Scala

이 예에서는 [getSourceWithFormat](#) 작업을 사용합니다.

```
// Example: Read CSV from S3
// For show, we handle a CSV with a header row. Set the withHeader option.
```

```
// Consider whether optimizePerformance is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"withHeader": true}"""),
      connectionType="s3",
      format="csv",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

또한 스크립트(`org.apache.spark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
val dataframe = spark.read
  .option("header", "true")
  .format("csv")
  .load("s3://s3path")
```

예: S3에 CSV 파일 및 폴더 쓰기

사전 조건: 초기화된 `DataFrame(dataFrame)` 또는 `DynamicFrame(dynamicFrame)`이 필요합니다. 예상되는 S3 출력 경로(`s3path`)도 필요합니다.

구성: 함수 옵션에서 `format="csv"`를 지정합니다. `connection_options`에서 `paths` 키를 사용하여 `s3path`를 지정합니다. `connection_options`에서 라이터와 S3가 상호 작용하는 방식을 구성할 수 있습니다. 자세한 내용은 AWS Glue에서 ETL 관련 연결 유형 및 옵션 참조: [S3 연결 파라미터](#) 작업에서 `format_options`에 있는 파일의 내용을 쓰는 방법을 구성할 수 있습니다. 자세한 내용은 [CSV 구성 참조](#)를 참조하십시오. 다음 AWS Glue ETL 스크립트는 S3로 CSV 파일 및 폴더를 쓰는 프로세스를 보여줍니다.

Python

이 예에서는 [write_dynamic_frame.from_options](#) 메서드를 사용합니다.

```
# Example: Write CSV to S3
# For show, customize how we write string type values. Set quoteChar to -1 so our
values are not quoted.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    connection_options={"path": "s3://s3path"},
    format="csv",
    format_options={
        "quoteChar": -1,
    },
)
```

또한 스크립트(`pyspark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
dataFrame.write\
    .format("csv")\
    .option("quote", None)\
    .mode("append")\
    .save("s3://s3path")
```

Scala

이 예에서는 [getSinkWithFormat](#) 메서드를 사용합니다.

```
// Example: Write CSV to S3
// For show, customize how we write string type values. Set quoteChar to -1 so our
values are not quoted.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
```

```
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
      format="csv"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

또한 스크립트(org.apache.spark.sql.DataFrame)에서 DataFrame을 사용합니다.

```
dataFrame.write
  .format("csv")
  .option("quote", null)
  .mode("Append")
  .save("s3://s3path")
```

CSV 구성 참조

AWS Glue 라이브러리가 format="csv"를 지정한 곳이라면 어디에서든 다음 format_options을 사용할 수 있습니다.

- separator - 구분 기호 문자열을 지정합니다. 기본값은 쉼표(,)이지만, 다른 문자도 지정할 수 있습니다.
 - 유형: 텍스트, 기본값: ","
- escaper - 이스케이프 처리에 사용할 문자를 지정합니다. 이 옵션은 CSV 파일을 읽을 때만 사용되며 쓸 때는 사용되지 않습니다. 활성화된 경우 바로 다음에 나오는 문자가 잘 알려진 이스케이프 세트(\n, \r, \t 및 \0)를 제외하고는 있는 그대로 사용됩니다.
 - 유형: 텍스트, 기본값: 없음
- quoteChar - 인용에 사용할 문자를 지정합니다. 기본값은 큰 따옴표(")입니다. 전체 인용을 해제하려면 이 값을 -1로 설정합니다.
 - 유형: 텍스트, 기본값: '"'

- `multiLine` - 단일 기록이 다양한 라인을 포괄할 수 있는지 여부를 지정합니다. 필드가 인용된 새로운 라인 문자를 포함할 때 발생합니다. 이 옵션을 `True`로 설정해야 기록이 여러 라인을 포괄할 수 있습니다. `multiLine`을 활성화하면 구문 분석하는 동안 더 신중한 파일 분할이 필요하므로 성능이 저하될 수 있습니다.
 - 유형: 부울, 기본값: `false`
- `withHeader` - 첫 번째 라인을 헤더로 취급할지 여부를 지정합니다. 이 옵션은 `DynamicFrameReader` 클래스에서 사용할 수 있습니다.
 - 유형: 부울, 기본값: `false`
- `writeHeader` - 헤더를 작성하여 출력할지 여부를 지정합니다. 이 옵션은 `DynamicFrameWriter` 클래스에서 사용할 수 있습니다.
 - 유형: 부울, 기본값: `true`
- `skipFirst` - 첫 번째 데이터 라인을 건너뛰는지 여부를 지정합니다.
 - 유형: 부울, 기본값: `false`
- `optimizePerformance` - Apache Arrow 기반 열 포맷 메모리 포맷과 함께 고급 SIMD CSV 리더를 사용할지 여부를 지정합니다. AWS Glue 3.0 이상에서만 사용 가능합니다.
 - 유형: 부울, 기본값: `false`
- `strictCheckForQuoting` - CSV를 쓸 때 Glue는 문자열로 해석되는 값에 따옴표를 추가할 수 있습니다. 이는 기록된 내용이 모호하지 않도록 하기 위해서입니다. 기록할 내용을 결정할 때 시간을 절약하기 위해 Glue는 따옴표가 필요하지 않은 특정 상황에서 인용할 수 있습니다. 엄격한 검사를 활성화하면 보다 컴퓨팅 집약적인 작업을 수행하고 필요한 경우에만 인용합니다. AWS Glue 3.0 이상에서만 사용 가능합니다.
 - 유형: 부울, 기본값: `false`

벡터화된 SIMD CSV 리더로 읽기 성능 최적화

AWS Glue 버전 3.0에는 행 기반 CSV 리더에 비해 전반적인 작업 속도를 크게 높일 수 있는 최적화된 CSV 리더가 추가되었습니다.

최적화된 리더:

- CPU SIMD 명령을 사용하여 디스크에서 읽기
- 레코드를 열 기반 형식으로 메모리에 즉시 쓰기(Apache Arrow)
- 레코드를 배치로 분할

이렇게 하면 나중에 레코드가 일괄 처리되거나 열 기반 형식으로 변환될 때 처리 시간이 절약됩니다. 스키마를 변경하거나 열별로 데이터를 검색하는 경우를 예로 들 수 있습니다.

최적화된 리더를 사용하려면 `format_options` 또는 테이블 속성에서 `"optimizePerformance"`를 `true`로 설정합니다.

```
glueContext.create_dynamic_frame.from_options(
    frame = datasource1,
    connection_type = "s3",
    connection_options = {"paths": ["s3://s3path"]},
    format = "csv",
    format_options={
        "optimizePerformance": True,
        "separator": ",",
    },
    transformation_ctx = "datasink2")
```

벡터화된 CSV 리더에 대한 제한 사항

벡터화된 CSV 리더의 제한 사항:

- `multiLine` 및 `escaper` 포맷 옵션은 지원되지 않습니다. 큰따옴표 문자('')의 기본값 `escaper`가 사용됩니다. 이러한 옵션을 설정하면 AWS Glue는 행 기반 CSV 리더를 사용하는 상태로 자동으로 돌아갑니다.
- [ChoiceType](#)으로 `DynamicFrame` 생성이 지원되지 않습니다.
- [오류 레코드](#)로 `DynamicFrame` 생성이 지원되지 않습니다.
- 일본어 또는 중국어와 같은 멀티바이트 문자가 포함된 CSV 파일 읽기가 지원되지 않습니다.

AWS Glue에서 Parquet 형식 사용

AWS Glue는 소스에서 데이터를 검색하고 다양한 데이터 형식으로 저장 및 전송되는 대상에 데이터를 씁니다. 데이터가 Parquet 데이터 형식으로 저장 또는 전송되는 경우 이 문서에서는 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능을 소개합니다.

AWS Glue는 Parquet 형식 사용을 지원합니다. 이 형식은 성능 중심의 열 기반 데이터 형식입니다. 표준 기관의 형식에 대한 소개는 [Apache Parquet 설명서 개요](#)를 참조하세요.

AWS Glue를 사용하여 Amazon S3와 스트리밍 소스에서 Parquet 파일을 읽을 수 있을 뿐만 아니라 Amazon S3에 Parquet 파일을 쓸 수 있습니다. S3에서 Parquet 파일이 포함된 bzip 및 gzip 아카이

브를 읽고 쓸 수 있습니다. 이 페이지에서 설명하는 구성 대신 [S3 연결 파라미터](#)에서 압축 동작을 구성할 수 있습니다.

다음 표에서는 Parquet 형식 옵션을 지원하는 일반적인 AWS Glue 기능을 보여줍니다.

읽기	쓰기	스트리밍 읽기	작은 파일 그룹화	작업 복마크
지원	지원	지원	지원되지 않음	지원*

* AWS Glue 버전 1.0 이상에서 지원

예: S3에서 Parquet 파일 또는 폴더 읽기

사전 조건: 읽고자 하는 Parquet 파일 또는 폴더에 대한 S3 경로(s3path)가 필요합니다.

구성: 함수 옵션에서 format="parquet"를 지정합니다. connection_options에서 paths 키를 사용하여 s3path를 지정합니다.

connection_options에서 리더와 S3가 상호 작용하는 방식을 구성할 수 있습니다. 자세한 내용은 AWS Glue에서 ETL 관련 연결 유형 및 옵션 참조: [S3 연결 파라미터](#)

리더에서 format_options의 Parquet 파일을 해석하는 방법을 구성할 수 있습니다. 자세한 내용은 [Parquet 구성 참조](#)를 참조하십시오.

다음 AWS Glue ETL 스크립트는 S3에서 Parquet 파일 또는 폴더를 읽는 프로세스를 보여줍니다.

Python

이 예에서는 [create_dynamic_frame.from_options](#) 메서드를 사용합니다.

```
# Example: Read Parquet from S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type = "s3",
```

```

    connection_options = {"paths": ["s3://s3path/"]},
    format = "parquet"
)

```

또한 스크립트(`pyspark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
dataFrame = spark.read.parquet("s3://s3path/")
```

Scala

이 예에서는 [getSourceWithFormat](#) 메서드를 사용합니다.

```

// Example: Read Parquet from S3

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="parquet",
      options=JsonOptions("""{"paths": ["s3://s3path"]}""")
    ).getDynamicFrame()
  }
}

```

또한 스크립트(`org.apache.spark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
spark.read.parquet("s3://s3path/")
```

예: S3에 Parquet 파일 및 폴더 쓰기

사전 조건: 초기화된 `DataFrame(dataFrame)` 또는 `DynamicFrame(dynamicFrame)`이 필요합니다. 예상되는 S3 출력 경로(`s3path`)도 필요합니다.

구성: 함수 옵션에서 `format="parquet"`를 지정합니다. `connection_options`에서 `paths` 키를 사용하여 `s3path`를 지정합니다.

`connection_options`에서 라이터가 S3와 상호 작용하는 방식을 추가로 변경할 수 있습니다. 자세한 내용은 AWS Glue에서 ETL 관련 연결 유형 및 옵션 참조: [S3 연결 파라미터](#) 작업에서 `format_options`에 있는 파일의 내용을 쓰는 방법을 구성할 수 있습니다. 자세한 내용은 [Parquet 구성 참조](#)를 참조하십시오.

다음 AWS Glue ETL 스크립트는 S3로 Parquet 파일 및 폴더를 쓰는 프로세스를 보여줍니다.

`useGlueParquetWriter` 구성 키를 통해 사용자 지정 Parquet 라이터에 `DynamicFrame`에 대한 성능 최적화가 제공됩니다. 이 라이터가 워크로드에 적합한지 확인하려면 [Glue Parquet 라이터](#)를 참조하십시오.

Python

이 예에서는 [write_dynamic_frame.from_options](#) 메서드를 사용합니다.

```
# Example: Write Parquet to S3
# Consider whether useGlueParquetWriter is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="parquet",
    connection_options={
        "path": "s3://s3path",
    },
    format_options={
        # "useGlueParquetWriter": True,
    },
)
```

또한 스크립트(`pyspark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
df.write.parquet("s3://s3path/")
```

Scala

이 예에서는 [getSinkWithFormat](#) 메서드를 사용합니다.

```
// Example: Write Parquet to S3
// Consider whether useGlueParquetWriter is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
      format="parquet"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

또한 스크립트(org.apache.spark.sql.DataFrame)에서 DataFrame을 사용합니다.

```
df.write.parquet("s3://s3path/")
```

Parquet 구성 참조

AWS Glue 라이브러리가 format="parquet"을 지정한 어디에서나 다음 format_options를 사용할 수 있습니다.

- useGlueParquetWriter - DynamicFrame 워크플로에 대한 성능 최적화가 포함된 사용자 지정 Parquet 라이터의 사용을 지정합니다. 자세한 내용은 [Glue Parquet 라이터](#)를 참조하세요.
 - 유형: 부울, 기본값:false
- compression - 사용되는 압축 코덱을 지정합니다. 값은 org.apache.parquet.hadoop.metadata.CompressionCodecName과 완벽하게 호환됩니다.
 - 유형: 열거 텍스트, 기본값: "snappy"

- 값: "uncompressed", "snappy", "gzip" 및 "lzo"
- blockSize - 메모리에 버퍼링되는 행 그룹의 크기(바이트 단위)를 지정합니다. 성능 튜닝에 사용됩니다. 크기는 정확히 메가바이트 수로 나뉘어야 합니다.
 - 유형: 숫자, 기본값:134217728
 - 기본값은 128MB입니다.
- pageSize - 페이지의 크기를 바이트 단위로 지정합니다. 성능 튜닝에 사용됩니다. 페이지는 단일 레코드에 액세스하기 위해 완전하게 읽어야 하는 가장 작은 단위입니다.
 - 유형: 숫자, 기본값:1048576
 - 기본값은 1MB입니다.

Note

또한 SparkSQL 기초 코드에 의해 수락된 옵션은 connection_options 맵 파라미터를 거쳐 SparkSQL 기초 코드로 넘겨질 수 있습니다. 예를 들어 AWS Glue Spark 리더에 대해 [mergeSchema](#)와 같은 Spark 구성을 설정하여 모든 파일의 스키마를 병합할 수 있습니다.

AWS Glue Parquet 라이터를 사용한 쓰기 성능 최적화

Note

AWS Glue Parquet 라이터는 예전에 glueparquet 형식 유형을 통해 액세스되었습니다. 이 액세스 패턴은 더 이상 지원되지 않습니다. 대신 useGlueParquetWriter이 활성화된 parquet 유형을 사용합니다.

AWS Glue Parquet 라이터는 더 빠른 Parquet 파일 쓰기가 가능하도록 성능이 향상되었습니다. 기존 라이터는 쓰기 전에 스키마를 컴퓨팅합니다. Parquet 형식은 빠르게 검색할 수 있는 방식으로 스키마를 저장하지 않으므로 시간이 걸릴 수 있습니다. AWS Glue Parquet 라이터를 사용하는 경우 사전 컴퓨팅된 스키마가 필요하지 않습니다. 데이터가 들어오면 라이터는 스키마를 동적으로 컴퓨팅 및 수정합니다.

useGlueParquetWriter를 지정하는 경우 다음 제한에 유의하세요.

- 라이터는 열 추가 또는 제거와 같은 스키마 개선만 지원하고 ResolveChoice와 같은 열 유형 변경은 지원하지 않습니다.

- 작성기는 빈 DataFrames 작성(예: 스키마 전용 파일 작성)을 지원하지 않습니다. `enableUpdateCatalog=True`를 설정하여 AWS Glue 데이터 카탈로그와 통합할 때 빈 DataFrame을 작성하려고 해도 데이터 카탈로그가 업데이트되지 않습니다. 그러면 스키마 없이 데이터 카탈로그에 테이블이 생성됩니다.

변형에 이러한 제한이 필요하지 않은 경우 AWS Glue Parquet 라이터를 켜면 성능이 향상됩니다.

AWS Glue에서 XML 형식 사용

AWS Glue는 소스에서 데이터를 검색하고 다양한 데이터 형식으로 저장 및 전송되는 대상에 데이터를 씁니다. 데이터가 XML 데이터 형식으로 저장 또는 전송되는 경우 이 문서에서는 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능을 소개합니다.

AWS Glue는 XML 형식 사용을 지원합니다. 이 형식은 행이나 열을 기반으로 하지 않는 고도로 구성 가능하고 엄격하게 정의된 데이터 구조를 나타냅니다. XML은 고도로 표준화되어 있습니다. 표준 기관의 형식에 대한 소개는 [XML 필수 사항](#)을 참조하십시오.

AWS Glue를 사용하여 Amazon S3는 물론 XML 파일이 포함된 bzip 및 gzip 아카이브에서 XML 파일을 읽을 수 있습니다. 이 페이지에서 설명하는 구성 대신 [S3 연결 파라미터](#)에서 압축 동작을 구성할 수 있습니다.

다음 표에서는 XML 형식 옵션을 지원하는 일반적인 AWS Glue 기능을 보여줍니다.

읽기	쓰기	스트리밍 읽기	작은 파일 그룹화	작업 복마크
지원	지원되지 않음	지원되지 않음	지원	지원

예: S3에서 XML 읽기

XML 리더는 XML 태그 이름을 사용합니다. 입력 내에 해당 태그가 있는 요소를 검사하여 스키마를 추론하고 DynamicFrame을 해당 값으로 채웁니다. AWS Glue XML 기능은 [Apache Spark용 XML 데이터 소스](#)와 비슷하게 작동합니다. 이 리더를 해당 프로젝트의 문서와 비교하면 기본 동작에 대한 통찰력을 얻을 수 있습니다.

사전 조건: 읽고자 하는 CSV 파일 또는 폴더에 대한 S3 경로(s3path), 그리고 XML 파일에 관한 약간의 정보가 필요합니다. 읽으려는 XML 요소에 대한 태그(xmlTag)도 필요합니다.

구성: 함수 옵션에서 `format="xml"`를 지정합니다. `connection_options`에서 `paths` 키를 사용하여 `s3path`를 지정합니다. `connection_options`에서 리더와 S3가 상호 작용하는 방식을

추가로 구성할 수 있습니다. 자세한 내용은 AWS Glue에서 ETL 관련 연결 유형 및 옵션 참조: [S3 연결 파라미터](#) `format_options`에서 `rowTag` 키를 사용하여 `xmlTag`를 지정합니다. 리더에서 `format_options`의 XML 파일을 해석하는 방법을 추가로 구성할 수 있습니다. 자세한 내용은 [XML 구성 참조](#)를 참조하십시오.

다음 AWS Glue ETL 스크립트는 S3에서 XML 파일 또는 폴더를 읽는 프로세스를 보여줍니다.

Python

이 예에서는 [create_dynamic_frame_from_options](#) 메서드를 사용합니다.

```
# Example: Read XML from S3
# Set the rowTag option to configure the reader.

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="xml",
    format_options={"rowTag": "xmlTag"},
)
```

또한 스크립트(`pyspark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
dataFrame = spark.read\
    .format("xml")\
    .option("rowTag", "xmlTag")\
    .load("s3://s3path")
```

Scala

이 예에서는 [getSourceWithFormat](#) 작업을 사용합니다.

```
// Example: Read XML from S3
// Set the rowTag option to configure the reader.
```

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkSession

val glueContext = new GlueContext(SparkContext.getOrCreate())
val sparkSession: SparkSession = glueContext.getSparkSession

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"rowTag": "xmlTag"}"""),
      connectionType="s3",
      format="xml",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

또한 스크립트(`org.apache.spark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
val dataframe = spark.read
  .option("rowTag", "xmlTag")
  .format("xml")
  .load("s3://s3path")
```

XML 구성 참조

AWS Glue 라이브러리가 `format="xml"`를 지정한 곳이라면 어디에서든 다음 `format_options`를 사용할 수 있습니다.

- `rowTag` - 파일의 XML 태그를 지정하여 태그를 열로 취급합니다. 열 태그는 자기 복제를 할 수 없습니다.
 - 유형: 텍스트, 필수
- `encoding` - 문자 인코딩을 지정합니다. 런타임 환경에서 지원하는 [Charset](#)의 이름 또는 별칭이 될 수 있습니다. 인코딩 지원에 대해 구체적으로 보장하지는 않지만 주요 인코딩은 작동할 것입니다.
 - 유형: 텍스트, 기본값: "UTF-8"
- `excludeAttribute` - 요소에서 속성을 제외할지 여부를 지정합니다.
 - 유형: 부울, 기본값: `false`
- `treatEmptyValuesAsNulls` - 공백을 null 값으로 취급할지 여부를 지정합니다.

- 유형: 부울, 기본값: false
- attributePrefix - 하위 요소 텍스트와 구별하기 위한 속성용 접두사. 이 접두사는 필드 이름으로 사용됩니다.
 - 유형: 텍스트, 기본값: "_"
- valueTag - 하위 요소가 없는 요소에 속성이 있는 값에 사용되는 태그.
 - 유형: 텍스트, 기본값: "_VALUE"
- ignoreSurroundingSpaces - 값을 둘러싸는 공백을 무시할지 여부를 지정합니다.
 - 유형: 부울, 기본값: false
- withSchema - 추론된 스키마를 재정의하려는 상황에서 예상되는 스키마를 포함합니다. 이 옵션을 사용하지 않으면 AWS Glue는 XML 데이터에서 스키마를 유추합니다.
 - 유형: 텍스트, 기본값: 해당 사항 없음
 - 값은 StructType을 나타내는 JSON 객체여야 합니다.

XML 스키마를 수동으로 지정

수동 XML 스키마 예

withSchema 포맷 옵션을 사용하여 XML 데이터에 대한 스키마를 지정하는 예입니다.

```
from awsglue.gluetypes import *

schema = StructType([
    Field("id", IntegerType()),
    Field("name", StringType()),
    Field("nested", StructType([
        Field("x", IntegerType()),
        Field("y", StringType()),
        Field("z", ChoiceType([IntegerType(), StringType()]))
    ]))
])

datasource0 = create_dynamic_frame_from_options(
    connection_type,
    connection_options={"paths": ["s3://xml_bucket/someprefix"]},
    format="xml",
    format_options={"withSchema": json.dumps(schema.jsonValue())},
    transformation_ctx = ""
)
```

AWS Glue에서 Avro 형식 사용

AWS Glue는 소스에서 데이터를 검색하고 다양한 데이터 형식으로 저장 및 전송되는 대상에 데이터를 씁니다. 데이터가 Avro 데이터 형식으로 저장 또는 전송되는 경우 이 문서에서는 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능을 소개합니다.

AWS Glue는 Avro 형식 사용을 지원합니다. 이 형식은 성능 중심의 행 기반 데이터 형식입니다. 표준 기관의 형식에 대한 소개는 [Apache Avro 1.8.2 설명서 개요](#)를 참조하세요.

AWS Glue를 사용하여 Amazon S3와 스트리밍 소스에서 Avro 파일을 읽을 수 있을 뿐만 아니라 Amazon S3에 Avro 파일을 쓸 수 있습니다. S3에서 Avro 파일이 포함된 bzip2 및 gzip 아카이브를 읽고 쓸 수 있습니다. 또한 Avro 파일이 포함된 deflate, snappy, xz 아카이브를 쓸 수 있습니다. 이 페이지에서 설명하는 구성 대신 [S3 연결 파라미터](#)에서 압축 동작을 구성할 수 있습니다.

다음 표에서는 Avro 형식 옵션을 지원하는 일반적인 AWS Glue 기능을 보여 줍니다.

읽기	쓰기	스트리밍 읽기	작은 파일 그룹화	작업 북마크
지원	지원	지원*	지원되지 않음	지원

* 제한적으로 지원됩니다. 자세한 내용은 [the section called “Avro 스트리밍 소스에 대한 참고 사항 및 제한 사항”](#) 단원을 참조하십시오.

예: S3에서 Avro 파일 또는 폴더 읽기

사전 조건: 읽으려는 Avro 파일 또는 폴더에 대한 S3 경로(s3path)가 필요합니다.

구성: 함수 옵션에서 format="avro"를 지정합니다. connection_options에서 paths 키를 사용하여 s3path를 지정합니다. connection_options에서 리더와 S3가 상호 작용하는 방식을 구성할 수 있습니다. 자세한 내용은 AWS Glue: [the section called “S3 연결 파라미터”](#)의 ETL 입력 및 출력에 대한 데이터 형식 옵션을 참조하세요. 리더에서 format_options의 Avro 파일을 해석하는 방법을 구성할 수 있습니다. 자세한 내용은 [Avro 구성 참조](#)를 참조하세요.

다음 AWS Glue ETL 스크립트는 S3에서 Avro 파일 또는 폴더를 읽는 프로세스를 보여 줍니다.

Python

이 예에서는 [create_dynamic_frame_from_options](#) 메서드를 사용합니다.

```
from pyspark.context import SparkContext
```

```

from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="avro"
)

```

Scala

이 예에서는 [getSourceWithFormat](#) 작업을 사용합니다.

```

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      connectionType="s3",
      format="avro",
      options=JsonOptions("""{"paths": ["s3://s3path"]}""")
    ).getDynamicFrame()
  }
}

```

예: S3에 Avro 파일 및 폴더 쓰기

사전 조건: 초기화된 DataFrame(dataFrame) 또는 DynamicFrame(dynamicFrame)이 필요합니다. 예상되는 S3 출력 경로(s3path)도 필요합니다.

구성: 함수 옵션에서 format="avro"를 지정합니다. connection_options에서 paths 키를 사용하여 s3path를 지정합니다. connection_options에서 라이터가 S3와 상호 작용하는 방식을 추가로 변경할 수 있습니다. 자세한 내용은 AWS Glue: [the section called "S3 연결 파라미터"](#)의 ETL 입력 및 출력에 대한 데이터 형식 옵션을 참조하세요. 작가가 format_options에서 Avro 파일을 해석하는 방식을 변경할 수 있습니다. 자세한 내용은 [Avro 구성 참조](#)를 참조하세요.

다음 AWS Glue ETL 스크립트는 S3로 Avro 파일 및 폴더를 쓰는 프로세스를 보여줍니다.

Python

이 예에서는 [write_dynamic_frame_from_options](#) 메서드를 사용합니다.

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="avro",
    connection_options={
        "path": "s3://s3path"
    }
)
```

Scala

이 예에서는 [getSinkWithFormat](#) 메서드를 사용합니다.

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
      format="avro"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

Avro 구성 참조

AWS Glue 라이브러리가 `format="avro"`를 지정한 곳이라면 어디에서든 다음 `format_options` 값을 사용할 수 있습니다.

- `version` - 지원하는 Apache Avro 리더/라이터 포맷의 버전을 지정합니다. 기본값은 "1.7"입니다. `format_options={"version": "1.8"}`를 지정해 Avro 논리적 유형 읽기 및 쓰기를 활성화할 수 있습니다. 자세한 내용은 [Apache Avro 1.7.7 사양](#) 및 [Apache Avro 1.8.2 사양](#)을 참조하십시오.

Apache Avro 1.8 커넥터는 다음과 같은 논리적 유형 변환을 지원합니다.

리더의 경우: 다음 테이블은 Avro 리더 1.7 및 1.8을 위한 Avro 데이터 유형(논리적 유형, Avro 기본 유형)과 AWS Glue `DynamicFrame` 데이터 유형 간 변환을 보여줍니다.

Avro 데이터 형식: 논리적 유형	Avro 데이터 형식: Avro 기본 유형	GlueDynamicFrame 데이터 유형: Avro 리더 1.7	GlueDynamicFrame 데이터 유형: Avro 리더 1.8
10진수	bytes	BINARY	10진수
10진수	고정	BINARY	10진수
날짜	int	INT	날짜
시간(밀리초)	int	INT	INT
시간(마이크로초)	long	LONG	LONG
타임스탬프(밀리초)	long	LONG	Timestamp
타임스탬프(마이크로초)	long	LONG	LONG
기간(논리적 유형 아님)	고정(12)	BINARY	BINARY

라이터의 경우: 다음 테이블은 Avro 라이터 1.7 및 1.8을 위한 AWS Glue `DynamicFrame` 데이터 유형과 Avro 데이터 유형 간 변환을 보여줍니다.

AWS Glue DynamicFrame 데이터 유형	Avro 데이터 형식: Avro 라이터 1.7	Avro 데이터 형식: Avro 라이터 1.8
10진수	String	decimal
날짜	String	date
Timestamp	String	timestamp-micros

Avro Spark DataFrame 지원

Spark DataFrame API에서 Avro를 사용하려면 해당 Spark 버전에 대한 Spark Avro 플러그인을 설치해야 합니다. 작업에서 사용할 수 있는 Spark 버전은 AWS Glue 버전에 따라 결정됩니다. Spark 버전에 대한 자세한 내용은 [the section called “AWS Glue 버전”](#) 섹션을 참조하세요. 이 플러그인은 Apache에서 유지 관리하며 구체적인 지원을 보장하지는 않습니다.

AWS Glue 2.0에서 - Spark Avro 플러그인의 버전 2.4.3을 사용하세요. 이 JAR는 Maven Central에서 찾을 수 있습니다. org.apache.spark:spark-avro_2.12:2.4.3을 참조하세요.

AWS Glue 3.0에서 - Spark Avro 플러그인의 버전 3.1.1을 사용하세요. 이 JAR는 Maven Central에서 찾을 수 있습니다. org.apache.spark:spark-avro_2.12:3.1.1을 참조하세요.

추가 JAR을 AWS Glue ETL 작업에 포함하려면 `--extra-jars` 작업 파라미터를 사용하세요. 작업 파라미터에 대한 자세한 내용을 알아보려면 [the section called “작업 파라미터”](#) 섹션을 참조하세요. AWS Management Console에서 이 매개 변수를 구성할 수도 있습니다.

AWS Glue에서 grokLog 형식 사용

AWS Glue는 소스에서 데이터를 검색하고 다양한 데이터 형식으로 저장 및 전송되는 대상에 데이터를 씁니다. 데이터가 느슨하게 구조화된 일반 텍스트 형식으로 저장되거나 전송되는 경우 이 문서에서는 Grok 패턴을 통해 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능을 소개합니다.

AWS Glue는 Grok 패턴 사용을 지원합니다. Grok 패턴은 정규 표현식 캡처 그룹과 유사합니다. 이들은 일반 텍스트 파일에서 문자 시퀀스의 패턴을 인식하여 유형과 목적을 부여합니다. AWS Glue에서 주요 목적은 로그를 읽는 것입니다. 저자의 Grok에 대한 소개는 [Logstash 참조: Grok 필터 플러그인](#)을 참조하세요.

읽기	쓰기	스트리밍 읽기	작은 파일 그룹화	작업 북마크
지원	해당 사항 없음	지원	지원	지원되지 않음

grokLog 구성 참조

format="grokLog"으로 다음 format_options 값을 사용할 수 있습니다.

- logFormat - 로그 포맷과 일치하는 Grok 패턴을 지정합니다.
- customPatterns - 여기서 사용된 Grok 패턴을 지정합니다.
- MISSING - 누락된 값을 식별하는 데 사용된 신호를 지정합니다. 기본값은 '-'입니다.
- LineCount - 각 로그 기록의 라인 수를 지정합니다. 기본값은 '1'이며 현재 단일 라인 기록만 지원합니다.
- StrictMode - 제한 모드를 설정할지 여부를 지정하는 부울 값. 제한 모드에서 리더는 자동적으로 유형을 전환하거나 복구하지 않습니다. 기본 값은 "false"입니다.

AWS Glue에서 Ion 형식 사용

AWS Glue는 소스에서 데이터를 검색하고 다양한 데이터 형식으로 저장 및 전송되는 대상에 데이터를 씁니다. 데이터가 Ion 데이터 형식으로 저장 또는 전송되는 경우 이 문서에서는 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능을 소개합니다.

AWS Glue는 Ion 형식 사용을 지원합니다. 이 형식은 교환 가능한 이진 및 일반 텍스트 표현으로 데이터 구조(행 또는 열 기반이 아님)를 나타냅니다. 저자의 형식에 대한 소개는 [Amazon Ion](#)을 참조하세요. (자세한 내용은 [Amazon Ion 상세 정보](#)를 참조하십시오.)

AWS Glue를 사용하여 Amazon S3에서 Ion 파일을 읽을 수 있습니다. S3에서 Ion 파일이 포함된 bzip 및 gzip 아카이브를 읽을 수 있습니다. 이 페이지에서 설명하는 구성 대신 [S3 연결 파라미터](#)에서 압축 동작을 구성할 수 있습니다.

다음 표에서는 Ion 형식 옵션을 지원하는 일반적인 AWS Glue 기능을 보여 줍니다.

읽기	쓰기	스트리밍 읽기	작은 파일 그룹화	작업 북마크
지원	지원되지 않음	지원되지 않음	지원	지원되지 않음

예: S3에서 Ion 파일 및 폴더 읽기

사전 조건: 읽으려는 Ion 파일 또는 폴더에 대한 S3 경로(s3path)가 필요합니다.

구성: 함수 옵션에서 format="json"를 지정합니다. connection_options에서 paths 키를 사용하여 s3path를 지정합니다. connection_options에서 리더와 S3가 상호 작용하는 방식을 구성할 수 있습니다. 자세한 내용은 AWS Glue에서 ETL 관련 연결 유형 및 옵션 참조: [the section called "S3 연결 파라미터"](#)

다음 AWS Glue ETL 스크립트는 S3에서 Ion 파일 또는 폴더를 읽는 프로세스를 보여 줍니다.

Python

이 예에서는 [create_dynamic_frame.from_options](#) 메서드를 사용합니다.

```
# Example: Read ION from S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="ion"
)
```

Scala

이 예에서는 [getSourceWithFormat](#) 작업을 사용합니다.

```
// Example: Read ION from S3

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
  }
}
```

```

val dynamicFrame = glueContext.getSourceWithFormat(
  connectionType="s3",
  format="ion",
  options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
).getDynamicFrame()
}
}

```

Ion 구성 참조

format="ion"을 위한 format_options 값은 없습니다.

Glue에서 AWS JSON 포맷 사용하기

AWS Glue는 소스에서 데이터를 검색하고 다양한 데이터 형식으로 저장 및 전송되는 대상에 데이터를 씁니다. 데이터가 JSON 데이터 형식으로 저장되거나 전송되는 경우 이 문서에서는 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능을 소개합니다.

AWS Glue는 JSON 형식 사용을 지원합니다. 이 형식은 행이나 열을 기반으로 하지 않는 일관된 모양이지만 유연한 내용의 데이터 구조를 나타냅니다. JSON은 여러 기관에서 발행한 병렬 표준으로 정의되며 그 중 하나는 ECMA-404입니다. 일반적으로 참조되는 소스의 형식에 대한 소개는 [JSON 소개](#)를 참조하세요.

AWS Glue를 사용하여 Amazon S3에서 JSON 파일과 gzip 압축된 JSON 파일을 읽을 수 있습니다. bzip 이 페이지에서 설명하는 구성 대신 [S3 연결 파라미터](#)에서 압축 동작을 구성할 수 있습니다.

읽기	쓰기	스트리밍 읽기	작은 파일 그룹화	작업 북마크	
지원	지원	지원	지원	지원	

예: S3에서 JSON 파일 또는 폴더 읽기

사전 조건: 읽으려는 JSON 파일 또는 폴더에 대한 S3 경로(s3path)가 필요합니다.

구성: 함수 옵션에서 format="json"를 지정합니다. connection_options에서 paths 키를 사용하여 s3path를 지정합니다. 연결 옵션에서 읽기 작업이 s3를 통과하는 방법을 추가로 변경할 수 있습니다. 자세한 내용은 [the section called "S3 연결 파라미터"](#)를 참조하세요. 리더에서

`format_options`의 JSON 파일을 해석하는 방법을 구성할 수 있습니다. 자세한 내용은 [JSON 구성 참조](#)를 참조하세요.

다음 AWS Glue ETL 스크립트는 S3에서 JSON 파일 또는 폴더를 읽는 프로세스를 보여줍니다.

Python

이 예에서는 [create_dynamic_frame.from_options](#) 메서드를 사용합니다.

```
# Example: Read JSON from S3
# For show, we handle a nested JSON file that we can limit with the JsonPath
parameter
# For show, we also handle a JSON where a single entry spans multiple lines
# Consider whether optimizePerformance is right for your workflow.

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="json",
    format_options={
        "jsonPath": "$.id",
        "multiline": True,
        # "optimizePerformance": True, -> not compatible with jsonPath, multiline
    }
)
```

스크립트 () DataFrames `pyspark.sql.DataFrame` 에서도 사용할 수 있습니다.

```
dataFrame = spark.read\
    .option("multiLine", "true")\
    .json("s3://s3path")
```

Scala

이 예제에서는 [getSourceWithFormat](#) 작업을 사용합니다.

```
// Example: Read JSON from S3
// For show, we handle a nested JSON file that we can limit with the JsonPath
// parameter
// For show, we also handle a JSON where a single entry spans multiple lines
// Consider whether optimizePerformance is right for your workflow.

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    val dynamicFrame = glueContext.getSourceWithFormat(
      formatOptions=JsonOptions("""{"jsonPath": "$.id", "multiline": true,
"optimizePerformance":false}"""),
      connectionType="s3",
      format="json",
      options=JsonOptions("""{"paths": ["s3://s3path"], "recurse": true}""")
    ).getDynamicFrame()
  }
}
```

스크립트 (pyspark.sql.DataFrame) DataFrames 에서도 사용할 수 있습니다.

```
val dataframe = spark.read
  .option("multiLine", "true")
  .json("s3://s3path")
```

예: S3에 JSON 파일 및 폴더 쓰기

필수 조건: 초기화된 DataFrame (dataFrame) 또는 DynamicFrame () 가 필요합니다.

dynamicFrame 예상되는 S3 출력 경로(s3path)도 필요합니다.

구성: 함수 옵션에서 format="json"를 지정합니다. connection_options에서 paths 키를 사용하여 s3path를 지정합니다. connection_options에서 라이터가 S3와 상호 작용하는 방식을 추가로 변경할 수 있습니다. 자세한 내용은 AWS [the section called "S3 연결 파라미터"](#) Glue:의 ETL 입력 및 출력에 대한 데이터 형식 옵션을 참조하십시오. 작성기에서 format_options의 JSON 파일을 해석하는 방법을 구성할 수 있습니다. 자세한 내용은 [JSON 구성 참조](#)를 참조하세요.

다음 AWS Glue ETL 스크립트는 S3에서 JSON 파일 또는 폴더를 작성하는 프로세스를 보여줍니다.

Python

이 예에서는 [write_dynamic_frame.from_options](#) 메서드를 사용합니다.

```
# Example: Write JSON to S3

from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    connection_options={"path": "s3://s3path"},
    format="json"
)
```

스크립트 () DataFrames pyspark.sql.DataFrame 에서도 사용할 수 있습니다.

```
df.write.json("s3://s3path/")
```

Scala

이 예제에서는 [getSinkWithFormat](#) 메서드를 사용합니다.

```
// Example: Write JSON to S3

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
    )
  }
}
```

```

        format="json"
    ).writeDynamicFrame(dynamicFrame)
}
}

```

스크립트 (`pyspark.sql.DataFrame`) `DataFrames` 에서도 사용할 수 있습니다.

```
df.write.json("s3://s3path")
```

JSON 구성 참조

`format="json"`으로 다음 `format_options` 값을 사용할 수 있습니다.

- `jsonPath`— 레코드로 읽어올 객체를 식별하는 [JsonPath](#) 표현식. 이것은 외부 배열 안에 중첩된 레코드가 파일에 있을 때 특히 유용합니다. 예를 들어, 다음 `JsonPath` 표현식은 JSON `id` 객체의 필드를 대상으로 합니다.

```
format="json", format_options={"jsonPath": "$.id"}
```

- `multiLine` - 단일 기록이 다양한 라인을 포괄할 수 있는지 여부를 지정하는 부울 값. 필드가 인용된 새로운 라인 문자를 포함할 때 발생합니다. 이 옵션을 `"true"`로 설정해야 기록이 여러 라인을 포괄할 수 있습니다. 기본값은 `"false"`이어서 파싱 동안 더 많은 공격적 파일 쪼개기가 가능합니다.
- `optimizePerformance` - Apache Arrow 기반 열 포맷 메모리 포맷과 함께 고급 SIMD JSON 리더를 사용할지 여부를 지정하는 부울 값입니다. AWS Glue 3.0에서만 사용 가능합니다. `multiLine` 또는 `jsonPath`와 호환되지 않습니다. 이러한 옵션 중 하나를 제공하면 AWS Glue가 표준 리더로 대체하도록 지시합니다.
- `withSchema` - [the section called “XML 스키마 지정”](#)에 설명된 형식으로 테이블 스키마를 지정하는 문자열 값입니다. 비카탈로그 연결에서 읽을 때 `optimizePerformance`에만 사용됩니다.

Apache Arrow 열 포맷으로 벡터화된 SIMD JSON 리더 사용

AWS Glue 버전 3.0은 JSON 데이터에 대한 벡터화된 리더를 추가합니다. 이 리더는 특정 조건에서 표준 리더에 비해 2배 더 빠른 성능을 발휘합니다. 이 리더에는 사용하기 전에 사용자가 알아야 할 특정 제한 사항이 있습니다(이 섹션에 설명되어 있음).

최적화된 리더를 사용하려면 `format_options` 또는 테이블 속성에서 `"optimizePerformance"`를 `True`로 설정합니다. 또한 카탈로그에서 읽지 않는 한 `withSchema`를 제공해야 합니다. `withSchema`는 [the section called “XML 스키마 지정”](#)에 설명된 대로 입력을 기대합니다.

```
// Read from S3 data source
glueContext.create_dynamic_frame.from_options(
  connection_type = "s3",
  connection_options = {"paths": ["s3://s3path"]},
  format = "json",
  format_options={
    "optimizePerformance": True,
    "withSchema": SchemaString
  })

// Read from catalog table
glueContext.create_dynamic_frame.from_catalog(
  database = database,
  table_name = table,
  additional_options = {
    // The vectorized reader for JSON can read your schema from a catalog table
    // property.
    "optimizePerformance": True,
  })
```

AWS Glue 라이브러리의 건물 *SchemaString*에 대한 자세한 내용은 [을 참조하십시오](#) [the section called “타입”](#).

벡터화된 CSV 리더에 대한 제한 사항

다음과 같은 제한 사항이 있습니다.

- 중첩된 객체 또는 배열 값이 있는 JSON 요소는 지원되지 않습니다. 제공된 경우 AWS Glue는 표준 리더로 대체됩니다.
- 카탈로그에서 또는 `withSchema`를 사용하여 스키마를 제공해야 합니다.
- `multiLine` 또는 `jsonPath`와 호환되지 않습니다. 이러한 옵션 중 하나를 제공하면 AWS Glue가 표준 리더로 대체하도록 지시합니다.
- 입력 스키마와 일치하지 않는 입력 레코드를 제공하면 리더가 실패할 수 있습니다.
- [오류 레코드](#)는 생성되지 않습니다.
- 일본어 또는 중국어와 같은 멀티바이트 문자가 포함된 JSON 파일은 지원되지 않습니다.

AWS Glue에서 ORC 형식 사용

AWS Glue는 소스에서 데이터를 검색하고 다양한 데이터 형식으로 저장 및 전송되는 대상에 데이터를 씁니다. 데이터가 ORC 데이터 형식으로 저장 또는 전송되는 경우 이 문서에서는 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능을 소개합니다.

AWS Glue는 ORC 형식 사용을 지원합니다. 이 형식은 성능 중심의 열 기반 데이터 형식입니다. 표준 기관의 형식에 대한 소개는 [Apache Orc](#)를 참조하세요.

AWS Glue를 사용하여 Amazon S3와 스트리밍 소스에서 ORC 파일을 읽을 수 있을 뿐만 아니라 Amazon S3에 ORC 파일을 쓸 수 있습니다. S3에서 ORC 파일이 포함된 bzip 및 gzip 아카이브를 읽고 쓸 수 있습니다. 이 페이지에서 설명하는 구성 대신 [S3 연결 파라미터](#)에서 압축 동작을 구성할 수 있습니다.

다음 표에서는 ORC 형식 옵션을 지원하는 일반적인 AWS Glue 기능을 보여 줍니다.

읽기	쓰기	스트리밍 읽기	작은 파일 그룹화	작업 복마크
지원	지원	지원	지원되지 않음	지원*

* AWS Glue 버전 1.0 이상에서 지원

예: S3에서 ORC 파일 또는 폴더 읽기

사전 조건: 읽으려는 ORC 파일 또는 폴더에 대한 S3 경로(s3path)가 필요합니다.

구성: 함수 옵션에서 format="orc"를 지정합니다. connection_options에서 paths 키를 사용하여 s3path를 지정합니다. connection_options에서 리더와 S3가 상호 작용하는 방식을 구성할 수 있습니다. 자세한 내용은 AWS Glue에서 ETL 관련 연결 유형 및 옵션 참조: [the section called "S3 연결 파라미터"](#)

다음 AWS Glue ETL 스크립트는 S3에서 ORC 파일 또는 폴더를 읽는 프로세스를 보여 줍니다.

Python

이 예에서는 [create_dynamic_frame_from_options](#) 메서드를 사용합니다.

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext
```

```

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

dynamicFrame = glueContext.create_dynamic_frame.from_options(
    connection_type="s3",
    connection_options={"paths": ["s3://s3path"]},
    format="orc"
)

```

또한 스크립트(`pyspark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```

dataFrame = spark.read\
    .orc("s3://s3path")

```

Scala

이 예에서는 [getSourceWithFormat](#) 작업을 사용합니다.

```

import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.GlueContext
import org.apache.spark.sql.SparkContext

object GlueApp {
    def main(sysArgs: Array[String]): Unit = {
        val spark: SparkContext = new SparkContext()
        val glueContext: GlueContext = new GlueContext(spark)

        val dynamicFrame = glueContext.getSourceWithFormat(
            connectionType="s3",
            format="orc",
            options=JsonOptions("""{"paths": ["s3://s3path"]}""")
        ).getDynamicFrame()
    }
}

```

또한 스크립트(`pyspark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```

val dataFrame = spark.read
    .orc("s3://s3path")

```

예: S3에 ORC 파일 및 폴더 쓰기

사전 조건: 초기화된 DataFrame(`dataFrame`) 또는 DynamicFrame(`dynamicFrame`)이 필요합니다. 예상되는 S3 출력 경로(`s3path`)도 필요합니다.

구성: 함수 옵션에서 `format="orc"`를 지정합니다. 연결 옵션에서 `paths` 키를 사용하여 `s3path`를 지정합니다. `connection_options`에서 라이터가 S3와 상호 작용하는 방식을 추가로 변경할 수 있습니다. 자세한 내용은 AWS Glue: [the section called "S3 연결 파라미터"](#)의 ETL 입력 및 출력에 대한 데이터 형식 옵션을 참조하세요. 다음 코드 예제는 프로세스를 보여 줍니다.

Python

이 예에서는 [write_dynamic_frame_from_options](#) 메서드를 사용합니다.

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext

sc = SparkContext.getOrCreate()
glueContext = GlueContext(sc)

glueContext.write_dynamic_frame.from_options(
    frame=dynamicFrame,
    connection_type="s3",
    format="orc",
    connection_options={
        "path": "s3://s3path"
    }
)
```

또한 스크립트(`pyspark.sql.DataFrame`)에서 DataFrame을 사용합니다.

```
df.write.orc("s3://s3path/")
```

Scala

이 예에서는 [getSinkWithFormat](#) 메서드를 사용합니다.

```
import com.amazonaws.services.glue.util.JsonOptions
import com.amazonaws.services.glue.{DynamicFrame, GlueContext}
import org.apache.spark.SparkContext
```

```
object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)

    glueContext.getSinkWithFormat(
      connectionType="s3",
      options=JsonOptions("""{"path": "s3://s3path"}"""),
      format="orc"
    ).writeDynamicFrame(dynamicFrame)
  }
}
```

또한 스크립트(`pyspark.sql.DataFrame`)에서 `DataFrame`을 사용합니다.

```
df.write.orc("s3://s3path/")
```

XML 구성 참조

`format="orc"`을 위한 `format_options` 값은 없습니다. 그러나 SparkSQL 기초 코드에 의해 수락된 옵션은 `connection_options` 맵 파라미터를 거쳐 SparkSQL 기초 코드로 넘겨질 수 있습니다.

AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용

오픈 소스 데이터 레이크 프레임워크는 Amazon S3에 빌드된 데이터 레이크에 저장된 파일의 증분 데이터 처리를 간소화합니다. AWS Glue 3.0 이상에서는 다음과 같은 오픈 소스 데이터 레이크 프레임워크를 지원합니다.

- Apache Hudi
- Linux Foundation Delta Lake
- Apache Iceberg

Amazon S3에 저장된 데이터를 트랜잭션적으로 일관된 방식으로 읽고 쓸 수 있도록 이러한 프레임워크에 대한 기본 지원을 제공합니다. AWS Glue ETL 작업에 이러한 프레임워크를 사용하기 위해 별도의 커넥터를 설치하거나 추가 구성 단계를 완료할 필요가 없습니다.

AWS Glue Data Catalog를 통해 데이터 세트를 관리하는 경우 Spark DataFrames에서 AWS Glue 메서드를 사용하여 데이터 레이크 테이블을 읽고 쓸 수 있습니다. Spark DataFrame API를 사용하여 Amazon S3 데이터를 읽고 쓸 수도 있습니다.

이 비디오에서는 Apache Hudi, Apache Iceberg 및 Delta Lake work의 작동 방식에 대한 기본 사항을 살펴봅니다. 데이터 레이크에서 데이터를 삽입, 업데이트 및 삭제하는 방법과 각 프레임워크의 작동 방식을 확인할 수 있습니다.

주제

- [제한 사항](#)
- [AWS Glue에서 Hudi 프레임워크 사용](#)
- [AWS Glue에서 Delta Lake 프레임워크 사용](#)
- [AWS Glue에서 Iceberg 프레임워크 사용](#)

제한 사항

데이터 레이크 프레임워크를 사용하기 전에 다음 제한 사항을 고려하십시오. AWS Glue

- 에 대한 다음 AWS Glue GlueContext 메서드는 데이터 레이크 프레임워크 테이블 읽기 및 쓰기를 DynamicFrame 지원하지 않습니다. DataFrame 또는 Spark DataFrame API의 GlueContext 메서드를 대신 사용하십시오.
 - Lake Formation 권한 DynamicFrame 제어에는 다음과 같은 GlueContext 방법이 지원되지 않습니다.
 - `create_dynamic_frame.from_catalog`
 - `write_dynamic_frame.from_catalog`
 - `getDynamicFrame`
 - `writeDynamicFrame`
 - Lake Formation 권한 제어에는 다음과 같은 GlueContext 방법이 DataFrame 지원됩니다.
 - `create_data_frame.from_catalog`
 - `write_data_frame.from_catalog`
 - `getDataFrame`
 - `writeDataFrame`
- [작은 파일 그룹화](#)는 지원되지 않습니다.
- [작업 북마크](#)는 지원되지 않습니다.
- AWS Glue 3.0용 아파치 휴디 0.10.1은 MoR (읽기 시 휴디 머지) 테이블을 지원하지 않습니다.
- ALTER TABLE ... RENAME T03.0용 아파치 아이스버그 0.13.1에는 사용할 수 없습니다. AWS Glue

Lake Formation 권한으로 관리되는 데이터 레이크 형식 테이블에 대한 제한

데이터 레이크 형식은 Lake Formation 권한을 통해 AWS Glue ETL과 통합됩니다. DynamicFrame 사용 `create_dynamic_frame` 생성은 지원되지 않습니다. 자세한 내용은 다음 예를 참조하세요.

- [예: Lake Formation 권한 제어가 포함된 Iceberg 테이블 읽기 및 쓰기](#)
- [예: Lake Formation 권한 제어를 사용하여 Hudi 테이블 읽기 및 쓰기](#)
- [예: Lake Formation 권한 제어가 포함된 Delta Lake 테이블 읽기 및 쓰기](#)

Note

아파치 후디, 아파치 아이스버그, 델타 레이크에 대한 AWS Glue Lake Formation 권한을 통한 ETL과의 통합은 버전 4.0에서만 지원됩니다. AWS Glue

아파치 아이스버그는 Lake Formation 권한을 통해 AWS Glue ETL과 가장 잘 통합됩니다. 거의 모든 작업을 지원하며 SQL 지원을 포함합니다.

Hudi는 관리 작업을 제외한 대부분의 기본 작업을 지원합니다. 이러한 옵션은 일반적으로 데이터 프레임 작업을 통해 수행되고 `additional_options`를 통해 지정되기 때문입니다. AWS Glue SparkSQL은 지원되지 않으므로 API를 사용하여 작업을 DataFrames 생성해야 합니다.

Delta Lake는 테이블 데이터의 읽기, 추가, 덮어쓰기만 지원합니다. Delta Lake에서는 업데이트와 같은 다양한 작업을 수행하려면 자체 라이브러리를 사용해야 합니다.

Lake Formation 권한으로 관리되는 Iceberg 테이블에는 다음 기능을 사용할 수 없습니다.

- ETL을 사용한 압축 AWS Glue
- ETL을 통한 스파크 SQL 지원 AWS Glue

Lake Formation 권한으로 관리되는 Hudi 테이블의 제한 사항은 다음과 같습니다.

- 분리된 파일 제거

Lake Formation 권한으로 관리되는 Delta Lake 테이블의 제한 사항은 다음과 같습니다.

- Delta Lake 테이블에서 삽입 및 읽기 이외의 모든 기능.

AWS Glue에서 Hudi 프레임워크 사용

AWS Glue 3.0 이상에서는 데이터 레이크에 대한 Apache Hudi 프레임워크를 지원합니다. Hudi는 증분 데이터 처리 및 데이터 파이프라인 개발을 간소화하는 오픈 소스 데이터 레이크 스토리지 프레임워크입니다. 이 항목에서는 Hudi 테이블에서 데이터를 전송하거나 저장할 때 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능에 대해 설명합니다. Hudi에 대한 자세한 내용은 공식 [Apache Hudi 설명서](#)를 참조하세요.

AWS Glue를 사용하여 Amazon S3의 Hudi 테이블에서 읽기 및 쓰기 작업을 수행하거나 AWS Glue 데이터 카탈로그를 사용하여 Hudi 테이블로 작업할 수 있습니다. 삽입, 업데이트, 모든 [Apache Spark 작업](#)을 포함한 추가 작업도 지원됩니다.

Note

Apache Hudi 0.10.1 for AWS Glue 3.0은 Hudi 읽을 때 병합(MoR) 테이블을 지원하지 않습니다.

다음 표에는 각 AWS Glue 버전에 포함된 Hudi 버전이 나와 있습니다.

AWS Glue 버전	지원되는 Hudi 버전
4.0	0.12.1
3.0	0.10.1

AWS Glue가 지원하는 데이터 레이크 프레임워크에 대한 자세한 내용은 [AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용](#) 섹션을 참조하세요.

Hudi 활성화

Hudi for AWS Glue를 활성화하려면 다음 작업을 완료합니다.

- hudi를 `--datalake-formats` 작업 파라미터의 값으로 지정합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#) 단원을 참조하십시오.
- AWS Glue 작업에 대한 `--conf` 키를 생성하고 다음 값으로 설정합니다. 또는 스크립트에서 SparkConf를 사용하여 다음 구성을 설정할 수 있습니다. 이러한 설정은 Apache Spark에서 Hudi 테이블을 올바르게 처리하는 데 도움이 됩니다.

```
spark.serializer=org.apache.spark.serializer.KryoSerializer --conf
spark.sql.hive.convertMetastoreParquet=false
```

- Hudi에 대한 Lake Formation 권한 지원은 AWS Glue 4.0에서 기본으로 활성화되어 있습니다. Lake Formation에 등록된 Hudi 테이블을 읽고 쓰는 데 추가 구성이 필요하지 않습니다. 등록된 Hudi 테이블을 읽으려면 AWS Glue 작업 IAM 역할에 SELECT 권한이 있어야 합니다. 등록된 Hudi 테이블에 글을 쓰려면 AWS Glue 작업 IAM 역할에 SUPER 권한이 있어야 합니다. Lake Formation 권한 관리에 대해 자세히 알아보려면 [Data Catalog 리소스에 대한 권한 부여 및 취소](#)를 참조하십시오.

다른 Hudi 버전 사용

AWS Glue에서 지원하지 않는 Hudi 버전을 사용하려면 `--extra-jars` 작업 파라미터를 사용하여 고유한 Hudi JAR 파일을 지정하세요. `--datalake-formats` 작업 파라미터의 값으로 hudi를 포함하지 마세요.

예: Amazon S3에 Hudi 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록

이 예제 스크립트는 Amazon S3에 Hudi 테이블을 작성하고 AWS Glue 데이터 카탈로그에 테이블을 등록하는 방법을 보여줍니다. 이 예제에서는 Hudi [Hive Sync 도구](#)를 사용하여 테이블을 등록합니다.

Note

이 예에서는 AWS Glue 데이터 카탈로그를 Apache Spark Hive 메타스토어로 사용하기 위해 `--enable-glue-datacatalog` 작업 파라미터를 설정해야 합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#)를 참조하십시오.

Python

```
# Example: Create a Hudi table from a DataFrame
# and register the table to Glue Data Catalog

additional_options={
    "hoodie.table.name": "<your_table_name>",
    "hoodie.datasource.write.storage.type": "COPY_ON_WRITE",
    "hoodie.datasource.write.operation": "upsert",
    "hoodie.datasource.write.recordkey.field": "<your_recordkey_field>",
    "hoodie.datasource.write.precombine.field": "<your_precombine_field>",
    "hoodie.datasource.write.partitionpath.field": "<your_partitionkey_field>",
    "hoodie.datasource.write.hive_style_partitioning": "true",
```



```

    "hoodie.datasource.hive_sync.enable": "true",
    "hoodie.datasource.hive_sync.database": "<your_database_name>",
    "hoodie.datasource.hive_sync.table": "<your_table_name>",
    "hoodie.datasource.hive_sync.partition_fields": "<your_partitionkey_field>",
    "hoodie.datasource.hive_sync.partition_extractor_class":
"org.apache.hudi.hive.MultiPartKeyValueExtractor",
    "hoodie.datasource.hive_sync.use_jdbc": "false",
    "hoodie.datasource.hive_sync.mode": "hms",
    "path": "s3://<s3Path/>"
}

dataFrame.write.format("hudi") \
  .options(**additional_options) \
  .mode("overwrite") \
  .save()

```

Scala

```

// Example: Example: Create a Hudi table from a DataFrame
// and register the table to Glue Data Catalog

val additionalOptions = Map(
  "hoodie.table.name" -> "<your_table_name>",
  "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
  "hoodie.datasource.write.operation" -> "upsert",
  "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
  "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
  "hoodie.datasource.write.partitionpath.field" -> "<your_partitionkey_field>",
  "hoodie.datasource.write.hive_style_partitioning" -> "true",
  "hoodie.datasource.hive_sync.enable" -> "true",
  "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
  "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
  "hoodie.datasource.hive_sync.partition_fields" -> "<your_partitionkey_field>",
  "hoodie.datasource.hive_sync.partition_extractor_class" ->
"org.apache.hudi.hive.MultiPartKeyValueExtractor",
  "hoodie.datasource.hive_sync.use_jdbc" -> "false",
  "hoodie.datasource.hive_sync.mode" -> "hms",
  "path" -> "s3://<s3Path/>")

dataFrame.write.format("hudi")
  .options(additionalOptions)
  .mode("append")
  .save()

```

예: AWS Glue 데이터 카탈로그를 사용하여 Amazon S3에서 Hudi 테이블 읽기

이 예에서는 Amazon S3의 [예: Amazon S3에 Hudi 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록](#)에서 생성한 Hudi 테이블을 읽습니다.

Note

이 예에서는 AWS Glue 데이터 카탈로그를 Apache Spark Hive 메타스토어로 사용하기 위해 `--enable-glue-datacatalog` 작업 파라미터를 설정해야 합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#)를 참조하십시오.

Python

이 예에서는 [GlueContext.create_data_frame.from_catalog\(\)](#) 메서드를 사용합니다.

```
# Example: Read a Hudi table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

dataFrame = glueContext.create_data_frame.from_catalog(
    database = "<your_database_name>",
    table_name = "<your_table_name>"
)
```

Scala

이 예에서는 [getCatalogSource](#) 메서드를 사용합니다.

```
// Example: Read a Hudi table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
```

```

val glueContext: GlueContext = new GlueContext(spark)

val dataframe = glueContext.getCatalogSource(
  database = "<your_database_name>",
  tableName = "<your_table_name>"
).getDataFrame()
}
}

```

예: Amazon S3의 Hudi 테이블에서 **DataFrame** 업데이트 및 삽입

이 예제에서는 AWS Glue 데이터 카탈로그를 사용하여 [예: Amazon S3에 Hudi 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록](#)에서 생성한 Hudi 테이블에 DataFrame을 삽입합니다.

Note

이 예에서는 AWS Glue 데이터 카탈로그를 Apache Spark Hive 메타스토어로 사용하기 위해 `--enable-glue-datacatalog` 작업 파라미터를 설정해야 합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#)를 참조하십시오.

Python

이 예에서는 [GlueContext.write_data_frame.from_catalog\(\)](#) 메서드를 사용합니다.

```

# Example: Upsert a Hudi table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
  frame = dataframe,
  database = "<your_database_name>",
  table_name = "<your_table_name>",
  additional_options={
    "hoodie.table.name": "<your_table_name>",
    "hoodie.datasource.write.storage.type": "COPY_ON_WRITE",
    "hoodie.datasource.write.operation": "upsert",
  }
)

```

```

"hoodie.datasource.write.recordkey.field": "<your_recordkey_field>",
"hoodie.datasource.write.precombine.field": "<your_precombine_field>",
"hoodie.datasource.write.partitionpath.field": "<your_partitionkey_field>",
"hoodie.datasource.write.hive_style_partitioning": "true",
"hoodie.datasource.hive_sync.enable": "true",
"hoodie.datasource.hive_sync.database": "<your_database_name>",
"hoodie.datasource.hive_sync.table": "<your_table_name>",
"hoodie.datasource.hive_sync.partition_fields": "<your_partitionkey_field>",
"hoodie.datasource.hive_sync.partition_extractor_class":
"org.apache.hudi.hive.MultiPartKeysValueExtractor",
"hoodie.datasource.hive_sync.use_jdbc": "false",
"hoodie.datasource.hive_sync.mode": "hms"
}
)

```

Scala

이 예에서는 [getCatalogSink](#) 메서드를 사용합니다.

```

// Example: Upsert a Hudi table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = JsonOptions(Map(
        "hoodie.table.name" -> "<your_table_name>",
        "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
        "hoodie.datasource.write.operation" -> "upsert",
        "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
        "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
        "hoodie.datasource.write.partitionpath.field" ->
"<your_partitionkey_field>",
        "hoodie.datasource.write.hive_style_partitioning" -> "true",
        "hoodie.datasource.hive_sync.enable" -> "true",
        "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
        "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
        "hoodie.datasource.hive_sync.partition_fields" ->
"<your_partitionkey_field>",

```

```

        "hoodie.datasource.hive_sync.partition_extractor_class" ->
"org.apache.hudi.hive.MultiPartKeyValueExtractor",
        "hoodie.datasource.hive_sync.use_jdbc" -> "false",
        "hoodie.datasource.hive_sync.mode" -> "hms"
    )))
    .writeDataFrame(dataFrame, glueContext)
}
}

```

예: Spark를 사용하여 Amazon S3에서 Hudi 테이블 읽기

이 예에서는 Spark DataFrame API를 사용하여 Amazon S3에서 Hudi 테이블을 읽습니다.

Python

```

# Example: Read a Hudi table from S3 using a Spark DataFrame

dataFrame = spark.read.format("hudi").load("s3://<s3path/>")

```

Scala

```

// Example: Read a Hudi table from S3 using a Spark DataFrame

val dataFrame = spark.read.format("hudi").load("s3://<s3path/>")

```

예: Spark를 사용하여 Amazon S3에 Hudi 테이블 쓰기

이 예에서는 Spark를 사용하여 Amazon S3에 Hudi 테이블을 씁니다.

Python

```

# Example: Write a Hudi table to S3 using a Spark DataFrame

dataFrame.write.format("hudi") \
    .options(**additional_options) \
    .mode("overwrite") \
    .save("s3://<s3Path/>")

```

Scala

```

// Example: Write a Hudi table to S3 using a Spark DataFrame

```

```
dataFrame.write.format("hudi")
  .options(additionalOptions)
  .mode("overwrite")
  .save("s3://<s3path/>")
```

예: Lake Formation 권한 제어를 사용하여 Hudi 테이블 읽기 및 쓰기

이 예제에서는 Lake Formation 권한 제어가 있는 Hudi 테이블을 읽고 씁니다.

1. Hudi 테이블을 생성하여 Lake Formation에 등록합니다.

- a. Lake Formation 권한 제어를 활성화하려면 먼저 Lake Formation에 Amazon S3 경로를 등록해야 합니다. 자세한 내용을 알아보려면 [Registering an Amazon S3 location](#)(Amazon S3 위치 등록)을 참조하세요. Lake Formation 콘솔에서 등록하거나 AWS CLI를 사용하여 등록할 수 있습니다.

```
aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-
folder> --use-service-linked-role --region <REGION>
```

Amazon S3 위치를 등록하면 해당 위치(또는 하위 위치)를 가리키는 AWS Glue 테이블이 GetTable 호출 시 IsRegisteredWithLakeFormation 파라미터 값을 true로 반환합니다.

- b. Spark 데이터프레임 API를 통해 등록된 Amazon S3 경로를 가리키는 Hudi 테이블을 생성합니다.

```
hudi_options = {
  'hoodie.table.name': table_name,
  'hoodie.datasource.write.storage.type': 'COPY_ON_WRITE',
  'hoodie.datasource.write.recordkey.field': 'product_id',
  'hoodie.datasource.write.table.name': table_name,
  'hoodie.datasource.write.operation': 'upsert',
  'hoodie.datasource.write.precombine.field': 'updated_at',
  'hoodie.datasource.write.hive_style_partitioning': 'true',
  'hoodie.upsert.shuffle.parallelism': 2,
  'hoodie.insert.shuffle.parallelism': 2,
  'path': <S3_TABLE_LOCATION>,
  'hoodie.datasource.hive_sync.enable': 'true',
  'hoodie.datasource.hive_sync.database': database_name,
  'hoodie.datasource.hive_sync.table': table_name,
  'hoodie.datasource.hive_sync.use_jdbc': 'false',
  'hoodie.datasource.hive_sync.mode': 'hms'
}
```

```
df_products.write.format("hudi") \
    .options(**hudi_options) \
    .mode("overwrite") \
    .save()
```

2. AWSGlue 작업 IAM 역할에 Lake Formation 권한을 부여하십시오. Lake Formation 콘솔에서 권한을 부여하거나 AWS CLI를 사용하여 권한을 부여할 수 있습니다. 자세한 내용을 알아보려면 [Lake Formation 콘솔 및 명명된 리소스 방법을 사용하여 데이터베이스 권한 부여](#)를 참조하십시오
3. Lake Formation에 등록된 Hudi 테이블을 읽어보십시오. 코드는 등록되지 않은 Hudi 테이블을 읽는 것과 같습니다. 읽기에 성공하려면 AWS Glue 작업 IAM 역할에 SELECT 권한이 있어야 한다는 점에 유의하십시오.

```
val dataframe = glueContext.getCatalogSource(
    database = "<your_database_name>",
    tableName = "<your_table_name>"
).getDataFrame()
```

4. Lake Formation에 등록된 Hudi 테이블에 글을 작성하십시오. 코드는 등록되지 않은 Hudi 테이블에 쓰는 것과 같습니다. 참고로 AWS Glue 작업 IAM 역할에 SUPER 권한이 있어야 쓰기가 성공합니다.

```
glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
    additionalOptions = JsonOptions(Map(
        "hoodie.table.name" -> "<your_table_name>",
        "hoodie.datasource.write.storage.type" -> "COPY_ON_WRITE",
        "hoodie.datasource.write.operation" -> "<write_operation>",
        "hoodie.datasource.write.recordkey.field" -> "<your_recordkey_field>",
        "hoodie.datasource.write.precombine.field" -> "<your_precombine_field>",
        "hoodie.datasource.write.partitionpath.field" -> "<your_partitionkey_field>",
        "hoodie.datasource.write.hive_style_partitioning" -> "true",
        "hoodie.datasource.hive_sync.enable" -> "true",
        "hoodie.datasource.hive_sync.database" -> "<your_database_name>",
        "hoodie.datasource.hive_sync.table" -> "<your_table_name>",
        "hoodie.datasource.hive_sync.partition_fields" ->
"<your_partitionkey_field>",
        "hoodie.datasource.hive_sync.partition_extractor_class" ->
"org.apache.hudi.hive.MultiPartKeyValueExtractor",
        "hoodie.datasource.hive_sync.use_jdbc" -> "false",
        "hoodie.datasource.hive_sync.mode" -> "hms"
    )))
.writeDataFrame(dataFrame, glueContext)
```

AWS Glue에서 Delta Lake 프레임워크 사용

AWS Glue 3.0 이상은 Linux Foundation Delta Lake 프레임워크를 지원합니다. Delta Lake는 ACID 트랜잭션을 수행하고, 메타데이터 처리를 확장하고, 스트리밍 및 배치 데이터 처리를 통합하는 데 도움이 되는 오픈 소스 데이터 레이크 스토리지 프레임워크입니다. 이 항목에서는 Delta Lake 테이블에서 데이터를 전송하거나 저장할 때 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능에 대해 설명합니다. Delta Lake에 대한 자세한 내용은 공식 [Delta Lake 설명서](#)를 참조하세요.

AWS Glue를 사용하여 Amazon S3의 Delta Lake 테이블에서 읽기 및 쓰기 작업을 수행하거나 AWS Glue 데이터 카탈로그를 사용하여 Delta Lake 테이블로 작업할 수 있습니다. 삽입, 업데이트, [테이블 배치 읽기 및 쓰기](#)와 같은 추가 작업도 지원됩니다. Delta Lake 테이블을 사용할 경우 Delta Lake Python 라이브러리(예: `DeltaTable.forPath`)의 메서드를 사용할 수도 있습니다. Delta Lake Python 라이브러리에 대한 자세한 내용은 Delta Lake의 Python 설명서를 참조하세요.

다음 표에는 각 AWS Glue 버전에 포함된 Delta Lake의 버전이 나와 있습니다.

AWS Glue 버전	지원되는 Delta Lake 버전
4.0	2.1.0
3.0	1.0.0

AWS Glue가 지원하는 데이터 레이크 프레임워크에 대한 자세한 내용은 [AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용](#) 섹션을 참조하세요.

Delta Lake for AWS Glue 활성화

Delta Lake for AWS Glue를 활성화하려면 다음 작업을 완료합니다.

- `delta`를 `--datalake-formats` 작업 파라미터의 값으로 지정합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#) 단원을 참조하십시오.
- AWS Glue 작업에 대한 `--conf` 키를 생성하고 다음 값으로 설정합니다. 또는 스크립트에서 `SparkConf`를 사용하여 다음 구성을 설정할 수 있습니다. 이러한 설정은 Apache Spark에서 Delta Lake 테이블을 올바르게 처리하는 데 도움이 됩니다.

```
spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog --conf
spark.delta.logStore.class=org.apache.spark.sql.delta.storage.S3SingleDriverLogStore
```


- Delta 테이블에 대한 Lake Formation 권한 지원은 AWS Glue 4.0에서 기본적으로 활성화되어 있습니다. Lake Formation에 등록된 Delta 테이블을 읽고 쓰는 데 추가 구성이 필요하지 않습니다. 등록된 Delta 테이블을 읽으려면 AWS Glue 작업 IAM 역할에 SELECT 권한이 있어야 합니다. 등록된 Delta 테이블에 쓰려면 AWS Glue 작업 IAM 역할에 SUPER 권한이 있어야 합니다. Lake Formation 권한 관리에 대해 자세히 알아보려면 [Data Catalog 리소스에 대한 권한 부여 및 취소](#)를 참조하십시오.

다른 Delta Lake 버전 사용

AWS Glue에서 지원하지 않는 Delta Lake 버전을 사용하려면 `--extra-jars` 작업 파라미터를 사용하여 고유한 Delta Lake JAR 파일을 지정하세요. `--datalake-formats` 작업 파라미터의 값으로 `delta`를 포함하지 마세요. 이 경우에 Delta Lake Python 라이브러리를 사용하려면 `--extra-py-files` 작업 파라미터를 사용하여 라이브러리 JAR 파일을 지정해야 합니다. Python 라이브러리는 Delta Lake JAR 파일에 패키징되어 제공됩니다.

예: Amazon S3에 Delta Lake 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록

다음 AWS Glue ETL 스크립트는 Amazon S3에 Delta Lake 테이블을 작성하고 AWS Glue 데이터 카탈로그에 테이블을 등록하는 방법을 보여줍니다.

Python

```
# Example: Create a Delta Lake table from a DataFrame
# and register the table to Glue Data Catalog

additional_options = {
    "path": "s3://<s3Path>"
}
dataFrame.write \
    .format("delta") \
    .options(**additional_options) \
    .mode("append") \
    .partitionBy("<your_partitionkey_field>") \
    .saveAsTable("<your_database_name>.<your_table_name>")
```

Scala

```
// Example: Example: Create a Delta Lake table from a DataFrame
// and register the table to Glue Data Catalog
```

```

val additional_options = Map(
  "path" -> "s3://<s3Path>"
)
dataFrame.write.format("delta")
  .options(additional_options)
  .mode("append")
  .partitionBy("<your_partitionkey_field>")
  .saveAsTable("<your_database_name>.<your_table_name>")

```

예: AWS Glue 데이터 카탈로그를 사용하여 Amazon S3에서 Delta Lake 테이블 읽기

다음 AWS Glue ETL 스크립트는 [예: Amazon S3에 Delta Lake 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록](#)에서 생성한 Delta Lake 테이블을 읽습니다.

Python

이 예에서는 [create_data_frame.from_catalog](#) 메서드를 사용합니다.

```

# Example: Read a Delta Lake table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
  database="<your_database_name>",
  table_name="<your_table_name>",
  additional_options=additional_options
)

```

Scala

이 예에서는 [getCatalogSource](#) 메서드를 사용합니다.

```

// Example: Read a Delta Lake table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {

```

```
def main(sysArgs: Array[String]): Unit = {
  val spark: SparkContext = new SparkContext()
  val glueContext: GlueContext = new GlueContext(spark)
  val df = glueContext.getCatalogSource("<your_database_name>",
"<your_table_name>",
  additionalOptions = additionalOptions)
  .getDataFrame()
}
}
```

예: AWS Glue 데이터 카탈로그를 사용하여 Amazon S3에서 Delta Lake 테이블에 **DataFrame** 삽입

이 예에서는 [예: Amazon S3에 Delta Lake 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록](#)에서 생성한 Delta Lake 테이블에 데이터를 삽입합니다.

Note

이 예에서는 AWS Glue 데이터 카탈로그를 Apache Spark Hive 메타스토어로 사용하기 위해 `--enable-glue-datacatalog` 작업 파라미터를 설정해야 합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#)를 참조하십시오.

Python

이 예에서는 [write_data_frame.from_catalog](#) 메서드를 사용합니다.

```
# Example: Insert into a Delta Lake table in S3 using Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

이 예에서는 [getCatalogSink](#) 메서드를 사용합니다.

```
// Example: Insert into a Delta Lake table in S3 using Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
      additionalOptions = additionalOptions)
      .writeDataFrame(dataFrame, glueContext)
  }
}
```

예: Spark API를 사용하여 Amazon S3에서 Delta Lake 테이블 읽기

이 예에서는 Spark API를 사용하여 Amazon S3에서 Delta Lake 테이블을 읽습니다.

Python

```
# Example: Read a Delta Lake table from S3 using a Spark DataFrame

dataFrame = spark.read.format("delta").load("s3://<s3path/>")
```

Scala

```
// Example: Read a Delta Lake table from S3 using a Spark DataFrame

val dataFrame = spark.read.format("delta").load("s3://<s3path/>")
```

예: Spark를 사용하여 Amazon S3에 Delta Lake 테이블 쓰기

이 예에서는 Spark를 사용하여 Amazon S3에 Delta Lake 테이블을 씁니다.

Python

```
# Example: Write a Delta Lake table to S3 using a Spark DataFrame

dataFrame.write.format("delta") \
    .options(**additional_options) \
    .mode("overwrite") \
    .partitionBy("<your_partitionkey_field>") \
    .save("s3://<s3Path>")
```

Scala

```
// Example: Write a Delta Lake table to S3 using a Spark DataFrame

dataFrame.write.format("delta")
    .options(additionalOptions)
    .mode("overwrite")
    .partitionBy("<your_partitionkey_field>")
    .save("s3://<s3path/>")
```

예: Lake Formation 권한 제어가 포함된 Delta Lake 테이블 읽기 및 쓰기

이 예제에서는 Lake Formation 권한 제어를 사용하여 Delta Lake 테이블을 읽고 씁니다.

1. Delta 테이블을 생성하여 Lake Formation에 등록하기

- a. Lake Formation 권한 제어를 활성화하려면 먼저 Lake Formation에 Amazon S3 경로를 등록해야 합니다. 자세한 내용을 알아보려면 [Registering an Amazon S3 location](#)(Amazon S3 위치 등록)을 참조하세요. Lake Formation 콘솔에서 등록하거나 AWS CLI를 사용하여 등록할 수 있습니다.

```
aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-
folder> --use-service-linked-role --region <REGION>
```

Amazon S3 위치를 등록하면 해당 위치(또는 하위 위치)를 가리키는 AWS Glue 테이블이 GetTable 호출 시 IsRegisteredWithLakeFormation 파라미터 값을 true로 반환합니다.

- b. Spark를 통해 등록된 Amazon S3 경로를 가리키는 Delta 테이블을 생성합니다.

Note

Python의 예를 들면 다음과 같습니다.

```
dataFrame.write \
  .format("delta") \
  .mode("overwrite") \
  .partitionBy("<your_partitionkey_field>") \
  .save("s3://<the_s3_path>")
```

Amazon S3에 데이터를 기록한 후 AWS Glue 크롤러를 사용하여 새 Delta 카탈로그 테이블을 생성합니다. 자세한 내용은 [AWSGlue 크롤러를 통한 네이티브 Delta Lake 테이블 지원 소개](#)를 참조하십시오.

AWSGlue CreateTable API를 통해 테이블을 수동으로 생성할 수도 있습니다.

2. AWSGlue 작업 IAM 역할에 Lake Formation 권한을 부여하십시오. Lake Formation 콘솔에서 권한을 부여하거나 AWS CLI를 사용하여 권한을 부여할 수 있습니다. 자세한 내용을 알아보려면 [Lake Formation 콘솔 및 명명된 리소스 방법을 사용하여 데이터베이스 권한 부여](#)를 참조하십시오.
3. Lake Formation에 등록된 Delta 테이블을 읽어보십시오. 코드는 등록되지 않은 Delta 테이블을 읽는 것과 같습니다. 읽기에 성공하려면 AWS Glue 작업 IAM 역할에 SELECT 권한이 있어야 한다는 점에 유의하십시오.

```
# Example: Read a Delta Lake table from Glue Data Catalog

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

4. Lake Formation에 등록된 Delta 테이블에 글을 씁니다. 코드는 등록되지 않은 Delta 테이블에 쓰는 것과 같습니다. 참고로 AWS Glue 작업 IAM 역할에 SUPER 권한이 있어야 쓰기가 성공합니다.

기본 설정상 AWS Glue는 Append를 saveMode로 사용합니다. additional_options에서 saveMode 옵션을 설정하여 변경할 수 있습니다. Delta 테이블의 saveMode 지원에 대한 자세한 내용은 [테이블에 쓰기](#)를 참조하십시오.

```
glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
```

)

AWS Glue에서 Iceberg 프레임워크 사용

AWS Glue 3.0 이상에서는 데이터 레이크에 대한 Apache Iceberg 프레임워크를 지원합니다. Iceberg는 SQL 테이블처럼 작동하는 고성능 테이블 형식을 제공합니다. 이 항목에서는 Iceberg 테이블에서 데이터를 전송하거나 저장할 때 AWS Glue에서 데이터를 사용하는 데 사용할 수 있는 기능에 대해 설명합니다. Iceberg에 대한 자세한 내용은 공식 [Apache Iceberg 설명서](#)를 참조하세요.

AWS Glue를 사용하여 Amazon S3의 Iceberg 테이블에서 읽기 및 쓰기 작업을 수행하거나 AWS Glue 데이터 카탈로그를 사용하여 Iceberg 테이블로 작업할 수 있습니다. 삽입, 업데이트, 모든 [Spark 쿼리](#) [Spark 쓰기](#)를 포함한 추가 작업도 지원됩니다.

Note

ALTER TABLE ... RENAME T0는 Apache Iceberg 0.13.1 for AWS Glue 3.0에서 사용할 수 없습니다.

다음 표에는 각 AWS Glue 버전에 포함된 Iceberg의 버전이 나와 있습니다.

AWS Glue 버전	지원되는 Iceberg 버전
4.0	1.0.0
3.0	0.13.1

AWS Glue가 지원하는 데이터 레이크 프레임워크에 대한 자세한 내용은 [AWS Glue ETL 작업에서 데이터 레이크 프레임워크 사용](#) 섹션을 참조하세요.

Iceberg 프레임워크 활성화

Iceberg for AWS Glue를 활성화하려면 다음 작업을 완료합니다.

- iceberg를 --datalake-formats 작업 파라미터의 값으로 지정합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#) 단원을 참조하십시오.

- AWS Glue 작업에 대한 `--conf` 키를 생성하고 다음 값으로 설정합니다. 또는 스크립트에서 `SparkConf`를 사용하여 다음 구성을 설정할 수 있습니다. 이러한 설정은 Apache Spark에서 Iceberg 테이블을 올바르게 처리하는 데 도움이 됩니다.

```
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
--conf spark.sql.catalog.glue_catalog=org.apache.iceberg.spark.SparkCatalog
--conf spark.sql.catalog.glue_catalog.warehouse=s3://<your-warehouse-dir>/
--conf spark.sql.catalog.glue_catalog.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog
--conf spark.sql.catalog.glue_catalog.io-impl=org.apache.iceberg.aws.s3.S3FileIO
```

Lake Formation에 등록된 Iceberg 테이블을 읽거나 쓰는 경우 다음 구성을 추가하여 Lake Formation 지원을 활성화하십시오. 참고로 Lake Formation에 등록된 Iceberg 테이블을 지원하는 것은 AWS Glue 4.0뿐입니다.

```
--conf spark.sql.catalog.glue_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.glue_catalog.glue.id=<table-catalog-id>
```

AWS Glue 3.0을 Iceberg 0.13.1과 함께 사용하는 경우 Amazon DynamoDB 잠금 관리자를 사용하여 원자성 트랜잭션을 보장하려면 다음과 같은 추가 구성을 설정해야 합니다. AWS Glue 4.0은 기본적으로 낙관적 잠금을 사용합니다. 자세한 내용은 공식 Apache Iceberg 설명서의 [Iceberg AWS 통합](#)을 참조하세요.

```
--conf spark.sql.catalog.glue_catalog.lock-impl=org.apache.iceberg.aws.glue.DynamoLockManager
--conf spark.sql.catalog.glue_catalog.lock.table=<your-dynamodb-table-name>
```

다른 Iceberg 버전 사용

AWS Glue에서 지원하지 않는 Iceberg 버전을 사용하려면 `--extra-jars` 작업 파라미터를 사용하여 고유한 Iceberg JAR 파일을 지정하세요. `--datalake-formats` 파라미터의 값으로 `iceberg`를 포함하지 마세요.

Iceberg 테이블에 대한 암호화 활성화

Note

Iceberg 테이블에는 서버 측 암호화를 활성화하는 자체 메커니즘이 있습니다. AWS Glue의 보안 구성 외에도 이 구성을 활성화해야 합니다.

Iceberg 테이블에서 서버 측 암호화를 활성화하려면 [Iceberg 설명서](#)의 지침을 검토하세요.

예: Amazon S3에 Iceberg 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록

이 예제 스크립트는 Amazon S3에 Iceberg 테이블을 쓰는 방법을 보여줍니다. 이 예제에서는 [Iceberg AWS 통합](#)을 사용하여 테이블을 AWS Glue 데이터 카탈로그에 등록합니다.

Python

```
# Example: Create an Iceberg table from a DataFrame
# and register the table to Glue Data Catalog

dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

query = f"""
CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
USING iceberg
TBLPROPERTIES ("format-version"="2")
AS SELECT * FROM tmp_<your_table_name>
"""
spark.sql(query)
```

Scala

```
// Example: Example: Create an Iceberg table from a DataFrame
// and register the table to Glue Data Catalog

dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

val query = """CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
USING iceberg
TBLPROPERTIES ("format-version"="2")
AS SELECT * FROM tmp_<your_table_name>
"""
spark.sql(query)
```

대신에 Spark 메서드를 사용하여 Amazon S3 및 데이터 카탈로그에 Iceberg 테이블을 작성할 수 있습니다.

사전 조건: Iceberg 라이브러리에서 사용할 카탈로그를 프로비저닝해야 합니다. AWS Glue 데이터 카탈로그를 사용할 때 AWS Glue를 사용하면 이 작업을 간편하게 수행할 수 있습니다. AWS Glue 데이터 카탈로그는 Spark 라이브러리에서 `glue_catalog`로 사용하도록 미리 구성되어 있습니다. 데이터 카탈로그 테이블은 `databaseName` 및 `tableName`으로 식별됩니다. AWS Glue 데이터 카탈로그에 대한 자세한 내용은 [데이터 검색 및 카탈로그 작성](#) 섹션을 참조하세요.

AWS Glue 데이터 카탈로그를 사용하지 않는 경우 Spark API를 통해 카탈로그를 프로비저닝해야 합니다. 자세한 내용은 Iceberg 설명서의 [Spark Configuration](#)을 참조하세요.

이 예시에서는 Spark를 사용하여 데이터 카탈로그에서 Amazon S3의 Iceberg 테이블을 작성합니다.

Python

```
# Example: Write an Iceberg table to S3 on the Glue Data Catalog

# Create (equivalent to CREATE TABLE AS SELECT)
dataFrame.writeTo("glue_catalog.databaseName.tableName") \
    .tableProperty("format-version", "2") \
    .create()

# Append (equivalent to INSERT INTO)
dataFrame.writeTo("glue_catalog.databaseName.tableName") \
    .tableProperty("format-version", "2") \
    .append()
```

Scala

```
// Example: Write an Iceberg table to S3 on the Glue Data Catalog

// Create (equivalent to CREATE TABLE AS SELECT)
dataFrame.writeTo("glue_catalog.databaseName.tableName")
    .tableProperty("format-version", "2")
    .create()

// Append (equivalent to INSERT INTO)
dataFrame.writeTo("glue_catalog.databaseName.tableName")
    .tableProperty("format-version", "2")
    .append()
```

예: AWS Glue 데이터 카탈로그를 사용하여 Amazon S3에서 Iceberg 테이블 읽기

이 예제에서는 [예: Amazon S3에 Iceberg 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록](#)에서 생성한 Iceberg 테이블을 읽습니다.

Python

이 예에서는 [GlueContext.create_data_frame.from_catalog\(\)](#) 메서드를 사용합니다.

```
# Example: Read an Iceberg table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

이 예에서는 [getCatalogSource](#) 메서드를 사용합니다.

```
// Example: Read an Iceberg table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
  def main(sysArgs: Array[String]): Unit = {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val df = glueContext.getCatalogSource("<your_database_name>",
"<your_table_name>",
      additionalOptions = additionalOptions)
      .getDataFrame()
  }
}
```

예: AWS Glue 데이터 카탈로그를 사용하여 Amazon S3에서 Iceberg 테이블에 **DataFrame** 삽입

이 예에서는 [예: Amazon S3에 Iceberg 테이블을 작성하고 AWS Glue 데이터 카탈로그에 등록](#)에서 생성한 Iceberg 테이블에 데이터를 삽입합니다.

Note

이 예에서는 AWS Glue 데이터 카탈로그를 Apache Spark Hive 메타스토어로 사용하기 위해 `--enable-glue-datacatalog` 작업 파라미터를 설정해야 합니다. 자세한 내용은 [AWS Glue 작업 파라미터](#)를 참조하십시오.

Python

이 예에서는 [GlueContext.write_data_frame.from_catalog\(\)](#) 메서드를 사용합니다.

```
# Example: Insert into an Iceberg table from Glue Data Catalog

from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
glueContext = GlueContext(sc)

glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

Scala

이 예에서는 [getCatalogSink](#) 메서드를 사용합니다.

```
// Example: Insert into an Iceberg table from Glue Data Catalog

import com.amazonaws.services.glue.GlueContext
import org.apache.spark.SparkContext

object GlueApp {
```

```
def main(sysArgs: Array[String]): Unit = {
  val spark: SparkContext = new SparkContext()
  val glueContext: GlueContext = new GlueContext(spark)
  glueContext.getCatalogSink("<your_database_name>", "<your_table_name>",
    additionalOptions = additionalOptions)
    .writeDataFrame(dataFrame, glueContext)
}
}
```

예: Spark를 사용하여 Amazon S3에서 Iceberg 테이블 읽기

사전 조건: Iceberg 라이브러리에서 사용할 카탈로그를 프로비저닝해야 합니다. AWS Glue 데이터 카탈로그를 사용할 때 AWS Glue를 사용하면 이 작업을 간편하게 수행할 수 있습니다. AWS Glue 데이터 카탈로그는 Spark 라이브러리에서 `glue_catalog`로 사용하도록 미리 구성되어 있습니다. 데이터 카탈로그 테이블은 `databaseName` 및 `tableName`으로 식별됩니다. AWS Glue 데이터 카탈로그에 대한 자세한 내용은 [데이터 검색 및 카탈로그 작성](#) 섹션을 참조하세요.

AWS Glue 데이터 카탈로그를 사용하지 않는 경우 Spark API를 통해 카탈로그를 프로비저닝해야 합니다. 자세한 내용은 Iceberg 설명서의 [Spark Configuration](#)을 참조하세요.

이 예제에서는 Spark를 사용하여 데이터 카탈로그에서 Amazon S3의 Iceberg 테이블을 읽습니다.

Python

```
# Example: Read an Iceberg table on S3 as a DataFrame from the Glue Data Catalog
dataFrame = spark.read.format("iceberg").load("glue_catalog.<databaseName>.<tableName>")
```

Scala

```
// Example: Read an Iceberg table on S3 as a DataFrame from the Glue Data Catalog
val dataFrame =
  spark.read.format("iceberg").load("glue_catalog.<databaseName>.<tableName>")
```

예: Lake Formation 권한 제어가 포함된 Iceberg 테이블 읽기 및 쓰기

이 예제에서는 Lake Formation 권한 제어를 사용하여 Iceberg 테이블을 읽고 씁니다.

1. Iceberg 테이블을 만들고 Lake Formation에 등록하십시오.

- a. Lake Formation 권한 제어를 활성화하려면 먼저 Lake Formation에 Amazon S3 경로를 등록해야 합니다. 자세한 내용을 알아보려면 [Registering an Amazon S3 location](#)(Amazon S3 위치 등록)을 참조하세요. Lake Formation 콘솔에서 등록하거나 AWS CLI를 사용하여 등록할 수 있습니다.

```
aws lakeformation register-resource --resource-arn arn:aws:s3:::<s3-bucket>/<s3-folder> --use-service-linked-role --region <REGION>
```

Amazon S3 위치를 등록하면 해당 위치(또는 하위 위치)를 가리키는 AWS Glue 테이블이 GetTable 호출 시 IsRegisteredWithLakeFormation 파라미터 값을 true로 반환합니다.

- b. Spark SQL을 통해 등록된 경로를 가리키는 Iceberg 테이블을 생성하십시오.

Note

Python의 예를 들면 다음과 같습니다.

```
dataFrame.createOrReplaceTempView("tmp_<your_table_name>")

query = f"""
CREATE TABLE glue_catalog.<your_database_name>.<your_table_name>
USING iceberg
AS SELECT * FROM tmp_<your_table_name>
"""
spark.sql(query)
```

AWS Glue CreateTable API를 통해 테이블을 수동으로 생성할 수도 있습니다. 자세한 내용은 [Apache Iceberg 테이블 생성](#)을 참조하십시오.

2. 작업 IAM 역할에 Lake Formation 권한을 부여하십시오. Lake Formation 콘솔에서 권한을 부여하거나 AWS CLI를 사용하여 권한을 부여할 수 있습니다. 자세한 내용은 <https://docs.aws.amazon.com/lake-formation/latest/dg/granting-table-permissions.html> 단원을 참조하십시오.
3. Lake Formation에 등록된 Iceberg 테이블을 참조하십시오. 코드는 등록되지 않은 Iceberg 테이블을 읽는 것과 같습니다. 읽기에 성공하려면 AWS Glue Job IAM 역할에 SELECT 권한이 있어야 한다는 점에 유의하십시오.

```
# Example: Read an Iceberg table from the AWS Glue Data Catalog
from awsglue.context import GlueContext
from pyspark.context import SparkContext

sc = SparkContext()
```

```
glueContext = GlueContext(sc)

df = glueContext.create_data_frame.from_catalog(
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

4. Lake Formation에 등록된 Iceberg 테이블에 씁니다. 코드는 등록되지 않은 Iceberg 테이블에 쓰는 것과 같습니다. 글쓰기가 성공하려면 AWS Glue 작업 IAM 역할에 SUPER 권한이 있어야 한다는 점을 참고하십시오.

```
glueContext.write_data_frame.from_catalog(
    frame=dataFrame,
    database="<your_database_name>",
    table_name="<your_table_name>",
    additional_options=additional_options
)
```

공유 구성 참조

모든 형식 유형의 다음과 같은 `format_options` 값을 사용할 수 있습니다.

- `attachFilename` - 열 이름으로 사용할 적절한 형식의 문자열입니다. 이 옵션을 제공하면 레코드의 소스 파일 이름이 레코드에 추가됩니다. 파라미터 값이 열 이름으로 사용됩니다.
- `attachTimestamp` - 열 이름으로 사용할 적절한 형식의 문자열입니다. 이 옵션을 제공하면 레코드의 소스 파일 수정 시간이 레코드에 추가됩니다. 파라미터 값이 열 이름으로 사용됩니다.

Spark SQL 작업에 대해 AWS Glue Data Catalog 지원

AWS Glue Data Catalog는 Apache Hive 메타스토어와 호환되는 카탈로그입니다. Data Catalog를 외부 Apache Hive 메타스토어로 사용하도록 AWS Glue 작업과 개발 엔드포인트를 구성할 수 있습니다. 그런 다음, Data Catalog에 저장된 테이블에 대해 Apache Spark SQL 쿼리를 직접 실행할 수 있습니다. AWS Glue 동적 프레임은 기본적으로 Data Catalog와 통합됩니다. 한편, Spark SQL 작업은 이 기능을 통해 외부 Hive 메타스토어로 Data Catalog를 사용하기 시작할 수 있습니다.

이 기능을 사용하려면 AWS Glue API 엔드포인트에 대한 네트워크 액세스가 필요합니다. 프라이빗 서브넷에 있는 연결을 사용하는 AWS Glue 작업의 경우 네트워크 액세스를 제공하도록 VPC 엔드포인트 또는 NAT 게이트웨이를 구성해야 합니다. VPC 엔드포인트 구성 방법에 대한 자세한 내용은 [데이터 스](#)

[토어에 대한 네트워크 액세스 설정](#) 섹션을 참조하세요. NAT 게이트웨이를 생성하려면 Amazon VPC 사용 설명서의 [NAT 게이트웨이](#)를 참조하세요.

작업 인수 및 개발 엔드포인트 인수에 각각 "--enable-glue-datacatalog": ""를 추가하여 AWS Glue 작업 및 개발 엔드포인트를 구성할 수 있습니다. 이 인수를 전달하면 외부 Hive 메타스토어로 Data Catalog에 액세스 할 수 있도록 Spark에서 특정 구성이 설정됩니다. 또한 AWS Glue 작업 또는 개발 엔드포인트에서 생성된 SparkSession 객체에서 [Hive 지원을 사용](#)합니다.

Data Catalog 액세스를 사용하려면 콘솔의 작업 추가(Add job) 또는 엔드포인트 추가(Add endpoint) 페이지에 있는 카탈로그 옵션(Catalog options) 그룹에서 AWS Glue Data Catalog를 Hive 메타스토어로 사용(Use Glue Data Catalog as the Hive metastore) 확인란을 선택합니다. 작업 또는 개발 엔드포인트에 사용되는 IAM 역할은 glue:CreateDatabase 권한을 가져야 합니다. "default"라는 데이터베이스가 없는 경우에 Data Catalog에서 생성됩니다.

Spark SQL 작업에서 이 기능을 사용할 수 있는 방법의 예제를 살펴보십시오. 다음 예에서는 s3://awsglue-datasets/examples/us-legislators에서 사용 가능한 미국 법률 기관의 데이터 세트를 크롤링했다고 가정합니다.

AWS Glue Data Catalog에 정의된 테이블에서 데이터를 직렬화/역직렬화하려면 Spark 작업의 클래스 경로의 AWS Glue Data Catalog에 정의된 포맷을 위한 [Hive SerDe](#) 클래스가 Spark SQL에 있어야 합니다.

특정 공통 형식을 위한 SerDe는 AWS Glue에 의해 배포됩니다. 다음은 이들에 대한 Amazon S3 링크입니다.

- [JSON](#)
- [XML](#)
- [Grok](#)

[개발 엔드포인트에 대한 외부 JAR](#)로서 JSON SerDe를 추가합니다. 작업 시, 인수 필드의 --extra-jars 인수를 사용하여 SerDe를 추가할 수 있습니다. 자세한 내용은 [AWS Glue 작업 파라미터](#) 단원을 참조하십시오.

다음은 Spark SQL에서 Data Catalog를 사용할 수 있도록 설정된 개발 엔드포인트를 생성하기 위한 입력 JSON의 예입니다.

```
{
  "EndpointName": "Name",
```



```

"RoleArn": "role_ARN",
"PublicKey": "public_key_contents",
"NumberOfNodes": 2,
"Arguments": {
  "--enable-glue-datacatalog": ""
},
"ExtraJarsS3Path": "s3://crawler-public/json/serde/json-serde.jar"
}

```

이제 Spark SQL을 사용하여 미국 법률 기관 데이터 세트에서 생성된 테이블을 쿼리합니다.

```

>>> spark.sql("use legislators")
DataFrame[]
>>> spark.sql("show tables").show()
+-----+-----+-----+
| database|      tableName|isTemporary|
+-----+-----+-----+
|legislators|      areas_json|      false|
|legislators|  countries_json|      false|
|legislators|      events_json|      false|
|legislators|  memberships_json|      false|
|legislators|  organizations_json|      false|
|legislators|      persons_json|      false|
+-----+-----+-----+
>>> spark.sql("describe memberships_json").show()
+-----+-----+-----+
|      col_name|data_type|      comment|
+-----+-----+-----+
|      area_id|  string|from deserializer|
|  on_behalf_of_id|  string|from deserializer|
|  organization_id|  string|from deserializer|
|      role|  string|from deserializer|
|      person_id|  string|from deserializer|
|legislative_perio...|  string|from deserializer|
|      start_date|  string|from deserializer|
|      end_date|  string|from deserializer|
+-----+-----+-----+

```

이 형식에 대한 SerDe 클래스를 작업의 클래스 경로에서 사용할 수 없는 경우에는 아래와 비슷한 오류가 나타납니다.

```

>>> spark.sql("describe memberships_json").show()

```

```

Caused by: MetaException(message:java.lang.ClassNotFoundException Class
org.openx.data.jsonserde.JsonSerDe not found)
    at
org.apache.hadoop.hive.metastore.MetaStoreUtils.getDeserializer(MetaStoreUtils.java:399)
    at
org.apache.hadoop.hive.ql.metadata.Table.getDeserializerFromMetaStore(Table.java:276)
    ... 64 more

```

memberships 테이블에서 별도의 organization_id만 보려면 다음 SQL 쿼리를 실행합니다.

```

>>> spark.sql("select distinct organization_id from memberships_json").show()
+-----+
| organization_id|
+-----+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+-----+

```

동적 프레임에서 똑같이 하려면 다음을 실행합니다.

```

>>> memberships = glueContext.create_dynamic_frame.from_catalog(database="legislators",
table_name="memberships_json")
>>> memberships.toDF().createOrReplaceTempView("memberships")
>>> spark.sql("select distinct organization_id from memberships").show()
+-----+
| organization_id|
+-----+
|d56acebe-8fdc-47b...|
|8fa6c3d2-71dc-478...|
+-----+

```

DynamicFrame이 ETL 작업에 최적화된 상태에서 Spark SQL이 Data Catalog에 액세스할 수 있도록 하면 손쉽게 복잡한 SQL 문을 실행하거나 기존 애플리케이션을 포팅할 수 있습니다.

작업 북마크 사용

AWS Glue for Spark는 작업 북마크를 사용하여 이미 진행된 데이터를 추적합니다. 작업 북마크 기능에 대한 요약 및 지원 내용은 [the section called “처리된 데이터를 작업 북마크로 추적”](#) 섹션을 참조하세요. 북마크를 사용해 AWS Glue 작업을 프로그래밍하면 시각적 작업에서 얻을 수 없었던 유연성을 활용할 수 있습니다.

- JDBC에서 읽을 때 AWS Glue 스크립트에서 북마크 키로 사용할 열을 지정할 수 있습니다.
- 각 메서드의 직접 호출에 적용할 `transformation_ctx`를 선택할 수 있습니다.

스크립트 시작 부분에서는 항상 `job.init`를 호출하고 스크립트 끝 부분에서는 적절하게 구성된 파라미터를 사용하여 `job.commit`을 호출하세요. 이 두 함수는 북마크 서비스를 초기화하고 서비스에 대한 상태 변경을 업데이트합니다. 북마크는 호출하지 않으면 작동하지 않습니다.

북마크 키 지정

JDBC 워크플로의 경우 북마크는 키 필드의 값을 북마크된 값과 비교하여 작업에서 읽은 행을 추적합니다. 이는 Amazon S3 워크플로에 필요하지 않거나 적용될 수 없습니다. 시각적 편집기 없이 AWS Glue 스크립트를 작성할 때 북마크로 추적할 열을 지정할 수 있습니다. 열을 여러 개 지정할 수도 있습니다. 사용자 정의 북마크 키를 지정할 때는 값 순서에 간격을 둘 수 있습니다.

Warning

사용자 정의 북마크 키를 사용하는 경우 각각 엄격하게 단순 증가 또는 감소해야 합니다. 복합 키에 대해 추가 필드를 선택할 때 '마이너 버전' 또는 '개정 번호'와 같은 개념에 대한 필드는 해당 값이 데이터 세트 전체에서 재사용되므로 이 기준을 충족하지 않습니다.

다음과 같은 방법으로 `jobBookmarkKeys` 및 `jobBookmarkKeysSortOrder`를 지정할 수 있습니다.

- `create_dynamic_frame.from_catalog - additional_options`를 사용합니다.
- `create_dynamic_frame.from_options - connection_options`를 사용합니다.

변환 컨텍스트

대부분의 AWS Glue PySpark 동적 프레임 메서드에는 ETL 연산자 인스턴스의 고유한 식별자인 조건부 파라미터 `transformation_ctx`가 들어 있습니다. `transformation_ctx` 파라미터는 작업 북마크 내에서 지정된 연산자의 상태 정보를 식별하는 데 사용됩니다. 특히 AWS Glue는 `transformation_ctx`를 사용해 북마크 상태에 키를 인덱싱합니다.

Warning

이 `transformation_ctx`는 스크립트에서 특정 소스에 대한 책갈피 상태를 검색하는 키 역할을 합니다. 북마크가 제대로 작동하려면 항상 소스와 관련 `transformation_ctx`를 일치시켜야 합니다. 소스 속성이나 `transformation_ctx`의 이름을 변경하면 이전 북마크를 유

호하지 않게 만들 수 있으며 타임스탬프 기반 필터링으로 인해 올바른 결과가 나오지 않을 수 있습니다.

작업 북마크가 제대로 작동하려면 작업 북마크 파라미터를 활성화하고 `transformation_ctx` 파라미터를 설정해야 합니다. `transformation_ctx` 파라미터를 전달하지 않으면 메서드에서 사용하는 테이블이나 동적 프레임에 대해 작업 북마크가 활성화되지 않습니다. 예를 들어, ETL 작업이 2개의 Amazon S3 소스를 읽고 조인하면 `transformation_ctx` 파라미터를 사용하고자 하는 북마크의 방법에만 전달할 수도 있습니다. 작업의 작업 북마크를 재설정하면 어떤 `transformation_ctx`를 사용하든 간에 해당 작업과 연관된 모든 변환이 재설정됩니다.

`DynamicFrameReader` 클래스에 대한 자세한 내용은 [DynamicFrameReader 수업](#)을 참조하십시오. PySpark 확장에 대한 자세한 내용은 [AWS Glue PySpark 확장 참조](#) 단원을 참조하십시오.

예제

Example

다음은 Amazon S3 데이터 원본에 대해 생성된 스크립트의 예입니다. 작업 북마크를 사용하는 데 필요한 스크립트 부분은 기울임꼴로 표시됩니다. 이러한 요소에 대한 자세한 내용은 [GlueContext 수업](#) API 및 [DynamicFrameWriter 클래스](#) API를 참조하세요.

```
# Sample Script
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = "database",
    table_name = "relatedqueries_csv",
    transformation_ctx = "datasource0"
)
```

```

applymapping1 = ApplyMapping.apply(
    frame = datasource0,
    mappings = [("col0", "string", "name", "string"), ("col1", "string", "number",
"string")],
    transformation_ctx = "applymapping1"
)

datasink2 = glueContext.write_dynamic_frame.from_options(
    frame = applymapping1,
    connection_type = "s3",
    connection_options = {"path": "s3://input_path"},
    format = "json",
    transformation_ctx = "datasink2"
)

job.commit()

```

Example

다음은 JDBC 소스에 대해 생성된 스크립트의 예입니다. 소스 테이블은 기본 키로 empno 열이 있는 직원 테이블입니다. 북마크 키가 지정되지 않은 경우 작업은 기본적으로 순차 기본 키를 북마크 키로 사용하지 않지만 empno는 반드시 순차적이지 않기 때문에(값에 공백이 있을 수 있음) 기본 북마크 키로 적합하지 않습니다. 따라서 스크립트는 empno를 북마크 키로 명시적으로 지정합니다. 코드의 해당 부분은 기울임꼴로 표시됩니다.

```

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

datasource0 = glueContext.create_dynamic_frame.from_catalog(

```

```

    database = "hr",
    table_name = "emp",
    transformation_ctx = "datasource0",
    additional_options = {"jobBookmarkKeys":["empno"],"jobBookmarkKeysSortOrder":"asc"}
)

applymapping1 = ApplyMapping.apply(
    frame = datasource0,
    mappings = [("ename", "string", "ename", "string"), ("hrly_rate", "decimal(38,0)",
"hrly_rate", "decimal(38,0)"), ("comm", "decimal(7,2)", "comm", "decimal(7,2)"),
("hiredate", "timestamp", "hiredate", "timestamp"), ("empno", "decimal(5,0)", "empno",
"decimal(5,0)"), ("mgr", "decimal(5,0)", "mgr", "decimal(5,0)"), ("photo", "string",
"photo", "string"), ("job", "string", "job", "string"), ("deptno", "decimal(3,0)",
"deptno", "decimal(3,0)"), ("ssn", "decimal(9,0)", "ssn", "decimal(9,0)"), ("sal",
"decimal(7,2)", "sal", "decimal(7,2)"]],
    transformation_ctx = "applymapping1"
)

datasink2 = glueContext.write_dynamic_frame.from_options(
    frame = applymapping1,
    connection_type = "s3",
    connection_options = {"path": "s3://hr/employees"},
    format = "csv",
    transformation_ctx = "datasink2"
)

job.commit()

```

AWS Glue Studio 외부에서 민감한 데이터 탐지 사용

AWS Glue Studio를 사용하면 민감한 데이터를 탐지할 수 있지만 AWS Glue Studio 외부에서 민감한 데이터 탐지 기능을 사용할 수도 있습니다.

민감한 관리형 데이터 형식의 전체 목록은 [관리형 데이터 형식](#)을 참조하세요.

AWS 관리형 PII 유형을 사용하여 민감한 데이터 탐지 탐지

AWS Glue AWS Glue ETL 작업에서 두 개의 API를 제공합니다. 이러한 항목은 `detect()` 및 `classifyColumns()`입니다.

```

detect(frame: DynamicFrame,
    entityTypeToDetect: Seq[String],

```

```

outputColumnName: String = "DetectedEntities",
detectionSensitivity: String = "LOW"): DynamicFrame

detect(frame: DynamicFrame,
detectionParameters: JsonOptions,
outputColumnName: String = "DetectedEntities",
detectionSensitivity: String = "LOW"): DynamicFrame

classifyColumns(frame: DynamicFrame,
entityTypesToDetect: Seq[String],
sampleFraction: Double = 0.1,
thresholdFraction: Double = 0.1,
detectionSensitivity: String = "LOW")

```

`detect()` API를 사용하여 AWS 관리형 PII 유형과 사용자 지정 엔티티 유형을 식별할 수 있습니다. 새로운 열이 감지 결과와 함께 자동 생성됩니다. `classifyColumns()` API는 키가 열 이름이고 값이 감지된 엔티티 유형 목록인 맵을 반환합니다. `SampleFraction`은 PII 엔티티를 검색할 때 샘플링할 데이터의 비율을 나타내는 반면, `ThresholdFraction`은 열이 PII 데이터로 식별되기 위해 충족되어야 하는 데이터의 비율을 나타냅니다.

행 수준 탐지

이 예제에서는 작업이 `detect()` 및 `classifyColumns()` API를 사용하여 다음 작업을 수행합니다.

- Amazon S3 버킷에서 데이터를 읽고 이를 `DynamicFrame`으로 변환합니다.
- `DynamicFrame`에서 '이메일' 및 '신용 카드' 인스턴스 검색
- 원래 값과 각 행에 대한 감지 결과를 포함하는 하나의 열이 포함된 `DynamicFrame` 반환
- 반환된 동적 프레임을 다른 경로에 쓰기 Amazon S3

```

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.ml.EntityDetector

```

```

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame=
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
"withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
transformationContext="AmazonS3_node1650160158526").getDynamicFrame()

    val frameWithDetectedPII = EntityDetector.detect(frame, Seq("EMAIL",
"CREDIT_CARD"))

    glueContext.getSinkWithFormat(connectionType="s3",
options=JsonOptions("""{"path": "s3://pathToOutput/", "partitionKeys": []}"""),
transformationContext="someCtx",
format="json").writeDynamicFrame(frameWithDetectedPII)

    Job.commit()
  }
}

```

세분화된 동작을 통한 행 수준 탐지

이 예제에서는 작업이 `detect()` API를 사용하여 다음 작업을 수행합니다.

- Amazon S3 버킷에서 데이터를 읽고 이를 `dynamicFrame`으로 변환
- `dynamicFrame`에서 “USA_PTIN”, “BANK_ACCOUNT”, “USA_SSN”, “USA_PASSPORT_NUMBER”, “PHONE_NUMBER”에 대한 민감한 데이터 유형 탐지
- 수정된 마스크 값과 각 행에 대한 감지 결과를 포함하는 하나의 열이 포함된 `dynamicFrame` 반환
- 반환된 `dynamicFrame`을 다른 Amazon S3 경로에 쓰기

위의 `detect()` API와 달리 이 API에서는 개체 유형이 탐지할 수 있도록 세분화된 동작을 사용합니다. 자세한 정보는 [detect\(\) 사용을 위한 탐지 파라미터](#)를 참조하세요.

```

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec

```



```
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame =
      glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
"withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
transformationContext="AmazonS3_node_source").getDynamicFrame()

    val detectionParameters = JsonOptions(
      """
      {
        "USA_DRIVING_LICENSE": [{
          "action": "PARTIAL_REDACT",
          "sourceColumns": ["Driving License"],
          "actionOptions": {
            "matchPattern": "[0-9]",
            "redactChar": "*"
          }
        }
      ]],
      "BANK_ACCOUNT": [{
        "action": "DETECT",
        "sourceColumns": ["*"]
      }],
      "USA_SSN": [{
        "action": "SHA256_HASH",
        "sourceColumns": ["SSN"]
      }],
      "IP_ADDRESS": [{
        "action": "REDACT",
        "sourceColumns": ["IP Address"],
        "actionOptions": {"redactText": "*****"}
      }],
    """
  )
}
```

```

        "PHONE_NUMBER": [{
            "action": "PARTIAL_REDACT",
            "sourceColumns": ["Phone Number"],
            "actionOptions": {
                "numLeftCharsToExclude": 1,
                "numRightCharsToExclude": 0,
                "redactChar": "*"
            }
        }
    ]
}
}
)

val frameWithDetectedPII = EntityDetector.detect(frame, detectionParameters,
"DetectedEntities", "HIGH")

glueContext.getSinkWithFormat(connectionType="s3", options=JsonOptions("""{"path":
"s3://pathToOutput/", "partitionKeys": []}"""),
transformationContext="AmazonS3_node_target",
format="json").writeDynamicFrame(frameWithDetectedPII)

    Job.commit()
}
}

```

열 수준 탐지

이 예제에서는 작업이 `classifyColumns()` API를 사용하여 다음 작업을 수행합니다.

- Amazon S3 버킷에서 데이터를 읽고 이를 `dynamicFrame`으로 변환
- `DynamicFrame`에서 '이메일' 및 '신용 카드' 인스턴스 검색
- 열을 100% 샘플링하고, 셀의 10%에 존재하며, 민감도가 "LOW"이면 개체를 탐지된 것으로 표시하도록 파라미터를 설정합니다
- 키가 열 이름이고 값이 탐지된 개체 유형 목록인 맵을 반환합니다
- 반환된 `dynamicFrame`을 다른 Amazon S3 경로에 쓰기

```

import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite

```

```
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.DynamicFrame
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame =
      glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar":
"\\", "withHeader": true, "separator": ",", "optimizePerformance": false}"""),
      connectionType="s3", format="csv", options=JsonOptions("""{"paths": ["s3://
pathToSource"], "recurse": true}"""), transformationContext="frame").getDynamicFrame()

    import glueContext.sparkSession.implicits._

    val detectedDataFrame = EntityDetector.classifyColumns(
      frame,
      entityTypeToDetect = Seq("CREDIT_CARD", "PHONE_NUMBER"),
      sampleFraction = 1.0,
      thresholdFraction = 0.1,
      detectionSensitivity = "LOW"
    )
    val detectedDF = (detectedDataFrame).toSeq.toDF("columnName", "entityTypes")
    val DetectSensitiveData_node = DynamicFrame(detectedDF, glueContext)

    glueContext.getSinkWithFormat(connectionType="s3", options=JsonOptions("""{"path":
"s3://pathToOutput", "partitionKeys": []}"""), transformationContext="someCtx",
    format="json").writeDynamicFrame(DetectSensitiveData_node)

    Job.commit()
  }
}
```

PII 유형을 사용한 AWS CustomEntityType 민감한 데이터 탐지 탐지

AWS Studio를 통해 사용자 지정 엔티티를 정의할 수 있습니다. 하지만 AWS Studio 외부에서 이 기능을 사용하려면 먼저 사용자 지정 엔티티 유형을 정의한 다음 정의된 사용자 지정 엔티티 유형을 목록에 추가해야 entityTypesToDetect 합니다.

데이터에 특정 민감한 데이터 유형(예: '직원 ID')이 있는 경우에는 CreateCustomEntityType() API를 호출하여 사용자 지정 엔티티를 만들 수 있습니다. 다음 예제에서는 요청 파라미터를 사용하여 CreateCustomEntityType() API에 사용자 지정 엔티티 유형 'EMPLOYEE_ID'를 정의합니다.

```
{
  "name": "EMPLOYEE_ID",
  "regexString": "\\d{4}-\\d{3}",
  "contextWords": ["employee"]
}
```

그리고 나서 EntityDetector() API에 사용자 지정 엔티티 유형(EMPLOYEE_ID)을 추가하여 새로운 사용자 지정 민감한 데이터 유형을 사용하도록 작업을 수정합니다.

```
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._
import com.amazonaws.services.glue.ml.EntityDetector

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    val frame=
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar": "\"",
"withHeader": true, "separator": ","}"""), connectionType="s3", format="csv",
```

```

options=JsonOptions("""{"paths": ["s3://pathToSource"], "recurse": true}"""),
transformationContext="AmazonS3_node1650160158526").getDynamicFrame()

    val frameWithDetectedPII = EntityDetector.detect(frame, Seq("EMAIL",
"CREDIT_CARD", "EMPLOYEE_ID"))

    glueContext.getSinkWithFormat(connectionType="s3",
options=JsonOptions("""{"path": "s3://pathToOutput/", "partitionKeys": []}"""),
transformationContext="someCtx",
format="json").writeDynamicFrame(frameWithDetectedPII)

    Job.commit()
}
}

```

Note

사용자 지정 민감한 데이터 유형이 기존 관리형 엔터티 유형과 동일한 이름을 사용하여 정의된 경우 사용자 지정 민감한 데이터 유형이 우선 적용되고 관리형 엔터티 유형의 논리를 덮어씁니다.

detect() 사용을 위한 탐지 파라미터

이 메서드는 a에서 엔티티를 탐지하는 데 사용됩니다 DynamicFrame. 원래 값이 DataFrame 포함된 새 값과 PII 탐지 메타데이터가 outputColumnName 있는 추가 열을 반환합니다. 사용자 지정 DynamicFrame 마스킹은 AWS Glue 스크립트 내에서 반환된 후에 수행할 수 있으며, 세분화된 작업을 포함하는 detect () API를 대신 사용할 수 있습니다.

```

detect(frame: DynamicFrame,
    entityTypeToDetect: Seq[String],
    outputColumnName: String = "DetectedEntities",
    detectionSensitivity: String = "LOW"): DynamicFrame

```

파라미터:

- 프레임 — (유형:DynamicFrame) 처리할 데이터가 DynamicFrame 포함된 입력입니다.
- entityTypeTo감지 — (유형:[Seq[String]) 탐지할 개체 유형 목록입니다. 관리 개체 유형 또는 사용자 지정 개체 유형일 수 있습니다.

- `outputColumnName`— (유형: `String`, 기본값: `"DetectedEntities"`) 탐지된 개체가 저장될 열의 이름. 지정하지 않은 경우 기본 열 이름은 `DetectedEntities`입니다.
- `detectionSensitivity` - (유형: `String`, 옵션: `"LOW"` or `"HIGH"`, 기본: `"LOW"`) 탐지 프로세스의 민감도를 지정합니다. 유효한 옵션은 `"LOW"` 또는 `"HIGH"`입니다. 제공되지 않은 경우 기본 민감도는 `"LOW"`으로 설정됩니다.

`outputColumnName` 설정:

탐지될 열 이름입니다. 지정하지 않은 경우 기본 열 이름은 `DetectedEntities`입니다. 출력 열의 각 행에 대해 보조 열에는 탐지된 개체 메타데이터에 대한 열 이름 맵이 다음과 같은 키값 쌍과 함께 포함됩니다.

- `entityType` - 탐지된 엔티티 유형입니다.
- `start` - 원본 데이터에서 탐지된 개체의 시작 위치입니다.
- `end` - 원본 데이터에서 탐지된 개체의 끝 위치입니다.
- `actionUsed` - 탐지된 개체에 대해 수행된 작업 (예: `"DETECT"`, `"REDACT"`, `"PARTIAL_REDACT"`, `"SHA256_HASH"`).

예제

```
{
  "DetectedEntities":{
    "SSN Col":[
      {
        "entityType":"USA_SSN",
        "actionUsed":"DETECT",
        "start":4,
        "end":15
      }
    ],
    "Random Data col":[
      {
        "entityType":"BANK_ACCOUNT",
        "actionUsed":"PARTIAL_REDACT",
        "start":4,
        "end":13
      },
      {
        "entityType":"IP_ADDRESS",
```

```

        "actionUsed":"REDACT",
        "start":4,
        "end":13
    }
]
}
}

```

세분화된 동작이 포함된 **detect()**의 탐지 파라미터

이 메서드는 지정된 매개변수를 `DynamicFrame` 사용하여 `a`에서 개체를 탐지하는 데 사용됩니다. 원래 값이 마스크된 민감한 `DataFrame` 데이터로 대체되고 PII 탐지 메타데이터가 `outputColumnName` 있는 추가 열이 포함된 새 값이 반환됩니다.

```

detect(frame: DynamicFrame,
        detectionParameters: JsonObject,
        outputColumnName: String = "DetectedEntities",
        detectionSensitivity: String = "LOW"): DynamicFrame

```

파라미터:

- 프레임 — (유형: `DynamicFrame`): 처리할 데이터가 `DynamicFrame` 포함된 입력입니다.
- `detectionParameters` - (유형: `JsonObject`): 탐지 프로세스의 매개 변수를 지정하는 JSON 옵션입니다.
- `outputColumnName`— (유형: `String`, 기본값: "DetectedEntities"): 탐지된 개체가 저장될 열의 이름입니다. 지정하지 않은 경우 기본 열 이름은 `DetectedEntities`입니다.
- `detectionSensitivity` - (유형: `String`, 옵션: "LOW" or "HIGH", 기본: "LOW"): 탐지 프로세스의 민감도를 지정합니다. 유효한 옵션은 "LOW" 또는 "HIGH"입니다. 제공되지 않은 경우 기본 민감도는 "LOW"으로 설정됩니다.

`detectionParameters` 설정

설정이 포함되지 않은 경우 기본값이 사용됩니다.

- `action` - (유형: `String`, 옵션: "DETECT", "REDACT", "PARTIAL_REDACT", "SHA256_HASH") 엔티티에 대해 수행할 작업을 지정합니다. 필수 사항입니다. 마스킹을 수행하는 작업 ("DETECT"을 제

외한 모든 작업)은 열당 하나의 작업만 수행할 수 있다는 점에 유의하십시오. 이는 합쳐진 개체를 마스킹하기 위한 예방 조치입니다.

- `sourceColumns` - (유형: `List[String]`, 기본값: `["*"]`) 개체에 대해 탐지를 수행할 소스 열 이름 목록입니다. 없는 경우 기본값은 `["*"]`입니다. 잘못된 열 이름을 사용하면 `IllegalArgumentException`이 발생합니다.
- `sourceColumnsToExclude` — (type: `List[String]`) 개체에 대한 탐지를 수행할 소스 열 이름 목록입니다. `sourceColumns` 또는 `sourceColumnsToExclude`를 사용합니다. 잘못된 열 이름을 사용하면 `IllegalArgumentException`이 발생합니다.
- `actionOptions` - 지정된 작업에 따른 추가 옵션:
 - "DETECT"와 "SHA256_HASH"는 옵션이 불가합니다.
 - "REDACT"의 경우:
 - `redactText` - (유형: `String`, 기본값: `*****`) 탐지된 개체를 대체할 텍스트입니다.
 - "PARTIAL_REDACT"의 경우:
 - `redactChar` - (유형: `String`, 기본값: `"*`") 개체에서 감지된 각 문자를 대체할 문자입니다.
 - `matchPattern` - (유형: `String`) 부분 수정을 위한 정규식 패턴입니다. `numLeftCharsToExclude` 또는 `numRightCharsToExclude`와 함께 사용할 수 없습니다.
 - `numLeftCharsToExclude` — (유형: `String`, `integer`) 제외할 왼쪽 문자 수입니다. `matchPattern`과 함께 사용할 수는 없지만 `numRightCharsToExclude`와 함께 사용할 수는 있습니다.
 - `numRightCharsToExclude` — (유형: `String`, `integer`) 제외할 오른쪽 문자 수입니다. `matchPattern`과 함께 사용할 수는 없지만 `numRightCharsToExclude`와 함께 사용할 수는 있습니다.

outputColumnName 설정

[outputColumnName 설정 참조](#)

`classifyColumns()`의 탐지 파라미터

이 메서드는 a에서 개체를 탐지하는 데 사용됩니다 DynamicFrame. 키가 열 이름이고 값이 탐지된 개체 유형 목록인 맵을 반환합니다. 사용자 지정 마스킹은 AWS Glue 스크립트 내에서 반환된 후에 수행할 수 있습니다.

```
classifyColumns(frame: DynamicFrame,
                entityTypeToDetect: Seq[String],
```



```
sampleFraction: Double = 0.1,
thresholdFraction: Double = 0.1,
detectionSensitivity: String = "LOW")
```

파라미터:

- 프레임 — (유형:DynamicFrame) 처리할 데이터가 DynamicFrame 포함된 입력입니다.
- entityTypeTo감지 — (유형:Seq[String]) 탐지할 개체 유형 목록입니다. 관리 개체 유형 또는 사용자 지정 개체 유형일 수 있습니다.
- sampleFraction - (유형: Double, 기본값: 10%) PII 엔터티를 스캔할 때 샘플링할 데이터의 이름입니다.
- thresholdFraction- (유형: Double, 기본값: 10%): 열을 PII 데이터로 식별하기 위해 충족되어야 하는 데이터의 비율입니다.
- detectionSensitivity - (유형: String, 옵션: "LOW" or "HIGH", 기본: "LOW") 탐지 프로세스의 민감도를 지정합니다. 유효한 옵션은 "LOW" 또는 "HIGH"입니다. 제공되지 않은 경우 기본 민감도는 "LOW"으로 설정됩니다.

관리형 민감한 데이터 유형

글로벌 엔터티

데이터 형식	범주	설명
PERSON_NAME	일반	사람의 이름.
EMAIL	개인	이메일 주소.
IP_ADDRESS	컴퓨터	IP 주소
MAC_ADDRESS	개인	MAC 주소.

미국 데이터 유형

데이터 형식	설명
BANK_ACCOUNT	은행 계좌 번호. 특정 국가나 지역에 국한되지 않으나 미국 및 캐나다 계정 형식만 검색됩니다.
CREDIT_CARD	신용카드 번호.
PHONE_NUMBER	전화번호 특정 국가나 지역에 국한되지 않으나 현재 미국 및 캐나다 전화번호만 검색됩니다.
USA_ATIN	미국 국세청에서 발급한 미국 Adoption Taxpayer Identification Number(ATIN).
USA_CPT_CODE	CPT 코드(미국만 해당).
USA_DEA_NUMBER	DEA 번호(미국만 해당).
USA_DRIVING_LICENSE	운전면허증 번호(미국만 해당).
USA_HCPCS_CODE	HCPCS 코드(미국만 해당).
USA_HEALTH_INSURANCE_CLAIM_NUMBER	건강 보험 청구 번호(미국만 해당).
USA_ITIN	ITIN(미국 개인 또는 법인의 경우).
USA_MEDICARE_BENEFICIARY_IDENTIFIER	메디케어 수혜자 식별자(미국만 해당).
USA_NATIONAL_DRUG_CODE	NDC 코드(미국만 해당).
USA_NATIONAL_PROVIDER_IDENTIFIER	국가 공급자 식별 번호(미국만 해당).
USA_PASSPORT_NUMBER	여권 번호(미국인의 경우).
USA_PTIN	미국 국세청에서 발급한 미국 Preparer Tax Identification Number(PTIN).
USA_SSN	사회보장번호(미국인의 경우).

아르헨티나 데이터 형식

데이터 형식	설명
ARGENTINA_TAX_IDENTIFICATION_NUMBER	아르헨티나 세금 식별 번호. CUIT 또는 CUIL이라고도 합니다.

오스트레일리아 데이터 형식

데이터 형식	설명
AUSTRALIA_BUSINESS_NUMBER	호주 사업자 번호(ABN). 정부 및 지역 사회가 기업을 식별하기 위해 호주 사업자 등록부(ABN)에서 발급하는 고유 식별자.
AUSTRALIA_COMPANY_NUMBER	호주 회사 번호(ACN). 호주 증권투자위원회에서 발행한 고유 식별자.
AUSTRALIA_DRIVING_LICENSE	호주의 운전면허증 번호.
AUSTRALIA_MEDICARE_NUMBER	호주 메디케어 번호. 호주 건강보험위원회에서 발급한 개인 식별자.
AUSTRALIA_PASSPORT_NUMBER	호주 여권 번호.
AUSTRALIA_TAX_FILE_NUMBER	호주 납세자 번호(TFN). 세금 납부를 위해 호주 국세청(ATO)에서 납세자(개인, 회사 등)에게 발급하는 번호.

오스트리아 데이터 형식

데이터 형식	설명
AUSTRIA_DRIVING_LICENSE	운전면허증 번호(오스트리아만 해당).
AUSTRIA_PASSPORT_NUMBER	여권 번호(오스트리아만 해당).
AUSTRIA_SSN	사회보장번호(호주인의 경우).

데이터 형식	설명
AUSTRIA_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(오스트리아만 해당).
AUSTRIA_VALUE_ADDED_TAX	부가가치세(오스트리아만 해당).

발칸 반도 데이터 형식

데이터 형식	설명
BOSNIA_UNIQUE_MASTER_CITIZEN_NUMBER	보스니아-헤르체고비나 시민의 고유 마스터 시민 번호(JMBG).
KOSOVO_UNIQUE_MASTER_CITIZEN_NUMBER	코소보의 고유 마스터 시민 번호(JMBG).
MACEDONIA_UNIQUE_MASTER_CITIZEN_NUMBER	마케도니아의 고유 마스터 시민 번호.
MONTENEGRO_UNIQUE_MASTER_CITIZEN_NUMBER	몬테네그로의 고유 마스터 시민 번호(JMBG).
SERBIA_UNIQUE_MASTER_CITIZEN_NUMBER	세르비아의 고유 마스터 시민 번호(JMBG).
SERBIA_VALUE_ADDED_TAX	부가가치세(세르비아만 해당).
VOJVODINA_UNIQUE_MASTER_CITIZEN_NUMBER	보이보디나의 고유 마스터 시민 번호(JMBG).

벨기에 데이터 형식

데이터 형식	설명
BELGIUM_DRIVING_LICENSE	운전면허증 번호(벨기에만 해당).

데이터 형식	설명
BELGIUM_NATIONAL_IDENTIFICATION_NUMBER	벨기에 국가 번호(BNN).
BELGIUM_PASSPORT_NUMBER	여권 번호(벨기에만 해당).
BELGIUM_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(벨기에만 해당).
BELGIUM_VALUE_ADDED_TAX	부가가치세(벨기에만 해당).

브라질 데이터 형식

데이터 형식	설명
BRAZIL_BANK_ACCOUNT	은행 계좌 번호(브라질만 해당).
BRAZIL_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자(브라질만 해당).
BRAZIL_NATIONAL_REGISTRY_OF_LEGAL_ENTITIES_NUMBER	회사에 발급되는 식별 번호(CNPJ라고도 함, 브라질만 해당).
BRAZIL_NATURAL_PERSON_REGISTRY_NUMBER	자연인 등록 번호(CPF라고도 함).

불가리아 데이터 형식

데이터 형식	설명
BULGARIA_DRIVING_LICENSE	운전면허증 번호(불가리아만 해당).
BULGARIA_UNIFORM_CIVIL_NUMBER	국가 식별 번호 역할을 하는 통합 시민 번호(EGN).
BULGARIA_VALUE_ADDED_TAX	부가가치세(불가리아만 해당).

캐나다 데이터 형식

데이터 형식	설명
CANADA_DRIVING_LICENSE	운전면허증 번호(캐나다만 해당).
CANADA_GOVERNMENT_IDENTIFICATION_CARD_NUMBER	국가 식별자(캐나다만 해당).
CANADA_PASSPORT_NUMBER	여권 번호(캐나다만 해당).
CANADA_PERMANENT_RESIDENCE_NUMBER	영주권 번호(PR 카드 번호).
CANADA_PERSONAL_HEALTH_NUMBER	의료용 고유 식별자(PHN 번호).
CANADA_SOCIAL_INSURANCE_NUMBER	캐나다의 사회보험 번호(SIN).

칠레 데이터 형식

데이터 형식	설명
CHILE_DRIVING_LICENSE	운전면허증 번호(칠레만 해당).
CHILE_NATIONAL_IDENTIFICATION_NUMBER	칠레 국가 식별자(RUT 또는 RUN이라고도 함).

중국, 홍콩, 마카오, 대만 데이터 형식

데이터 형식	설명
CHINA_IDENTIFICATION	중국 식별자.
CHINA_LICENSE_PLATE_NUMBER	운전면허증 번호(중국만 해당).
CHINA_MAINLAND_TRAVEL_PERMIT_ID_HONG_KONG_MACAU	홍콩 및 마카오 거주자를 위한 본토 여행 허가증.

데이터 형식	설명
CHINA_MAINLAND_TRAVEL_PERMIT_ID_TAIWAN	중화인민공화국 정부(PRC)에서 발급한 대만 거주자를 위한 본토 여행 허가증.
CHINA_PASSPORT_NUMBER	여권 번호(중국만 해당).
CHINA_PHONE_NUMBER	전화번호(캐나다만 해당).
HONG_KONG_IDENTITY_CARD	홍콩 이민국에서 발급한 공식 신분증.
MACAU_RESIDENT_IDENTITY_CARD	마카오 거주자 신분증 또는 BIR은 마카오 신원 확인국에서 발급하는 공식 신분증.
TAIWAN_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자(대만만 해당).
TAIWAN_PASSPORT_NUMBER	여권 번호(대만만 해당).

콜롬비아 데이터 형식

데이터 형식	설명
COLOMBIA_PERSONAL_IDENTIFICATION_NUMBER	콜롬비아 출생 시 부여되는 고유 식별자.
COLOMBIA_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(콜롬비아만 해당).

크로아티아 데이터 형식

데이터 형식	설명
CROATIA_DRIVING_LICENSE	운전면허증 번호(크로아티아만 해당).
CROATIA_IDENTITY_NUMBER	국가 식별자(크로아티아에만 해당).
CROATIA_PASSPORT_NUMBER	여권 번호(크로아티아만 해당).

데이터 형식	설명
CROATIA_PERSONAL_IDENTIFICATION_NUMBER	개인 식별 번호(OIB).

키프로스 데이터 형식

데이터 형식	설명
CYPRUS_DRIVING_LICENSE	운전면허증 번호(키프로스만 해당).
CYPRUS_NATIONAL_IDENTIFICATION_NUMBER	키프로스 신분증.
CYPRUS_PASSPORT_NUMBER	여권 번호(키프로스만 해당).
CYPRUS_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(키프로스만 해당).
CYPRUS_VALUE_ADDED_TAX	부가가치세(키프로스만 해당).

체코 데이터 형식

데이터 형식	설명
CZECHIA_DRIVING_LICENSE	운전면허증 번호(체코만 해당).
CZECHIA_PERSONAL_IDENTIFICATION_NUMBER	개인 식별자 번호(체코만 해당).
CZECHIA_VALUE_ADDED_TAX	부가가치세(체코만 해당).

덴마크 데이터 형식

데이터 형식	설명
DENMARK_DRIVING_LICENSE	운전면허증 번호(덴마크만 해당).

데이터 형식	설명
DENMARK_PERSONAL_IDENTIFICATION_NUMBER	개인 식별자 번호(덴마크만 해당).
DENMARK_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(덴마크만 해당).
DENMARK_VALUE_ADDED_TAX	부가가치세(덴마크만 해당).

에스토니아 데이터 형식

데이터 형식	설명
ESTONIA_DRIVING_LICENSE	운전면허증 번호(에스토니아만 해당).
ESTONIA_PASSPORT_NUMBER	여권 번호(에스토니아만 해당).
ESTONIA_PERSONAL_IDENTIFICATION_CODE	개인 식별자 번호(에스토니아만 해당).
ESTONIA_VALUE_ADDED_TAX	부가가치세(에스토니아만 해당).

핀란드 데이터 형식

데이터 형식	설명
FINLAND_DRIVING_LICENSE	운전면허증 번호(핀란드만 해당).
FINLAND_HEALTH_INSURANCE_NUMBER	건강 보험 번호(핀란드만 해당).
FINLAND_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자 번호(핀란드만 해당).
FINLAND_PASSPORT_NUMBER	여권 번호(핀란드만 해당).
FINLAND_VALUE_ADDED_TAX	부가가치세(핀란드만 해당).

프랑스 데이터 형식

데이터 형식	설명
FRANCE_BANK_ACCOUNT	은행 계좌 번호(프랑스만 해당).
FRANCE_DRIVING_LICENSE	운전면허증 번호(프랑스만 해당).
FRANCE_HEALTH_INSURANCE_NUMBER	프랑스 건강보험 번호.
FRANCE_INSEE_CODE	프랑스 사회보장번호, SSN 번호 또는 NIR 번호.
FRANCE_NATIONAL_IDENTIFICATION_NUMBER	프랑스 국가 식별자 번호(CNI).
FRANCE_PASSPORT_NUMBER	여권 번호(프랑스만 해당).
FRANCE_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(프랑스만 해당).
FRANCE_VALUE_ADDED_TAX	부가가치세(프랑스만 해당).

독일 데이터 형식

데이터 형식	설명
GERMANY_BANK_ACCOUNT	은행 계좌 번호(독일만 해당).
GERMANY_DRIVING_LICENSE	운전면허증 번호(독일만 해당).
GERMANY_PASSPORT_NUMBER	여권 번호(독일만 해당).
GERMANY_PERSONAL_IDENTIFICATION_NUMBER	개인 식별 번호(독일만 해당).
GERMANY_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(독일만 해당).
GERMANY_VALUE_ADDED_TAX	부가가치세(독일만 해당).

그리스 데이터 형식

데이터 형식	설명
GREECE_DRIVING_LICENSE	운전면허증 번호(그리스만 해당).
GREECE_PASSPORT_NUMBER	여권 번호(그리스만 해당).
GREECE_SSN	사회보장번호(그리스인의 경우).
GREECE_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(그리스만 해당).
GREECE_VALUE_ADDED_TAX	부가가치세(그리스만 해당).

헝가리 데이터 형식

데이터 형식	설명
HUNGARY_DRIVING_LICENSE	운전면허증 번호(헝가리만 해당).
HUNGARY_PASSPORT_NUMBER	여권 번호(헝가리만 해당).
HUNGARY_SSN	사회보장번호(헝가리인의 경우).
HUNGARY_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(헝가리만 해당).
HUNGARY_VALUE_ADDED_TAX	부가가치세(헝가리만 해당).

아이슬란드 데이터 형식

데이터 형식	설명
ICELAND_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자(아이슬란드만 해당).
ICELAND_PASSPORT_NUMBER	여권 번호(아이슬란드만 해당).
ICELAND_VALUE_ADDED_TAX	부가가치세(아이슬란드만 해당).

인도 데이터 형식

데이터 형식	설명
INDIA_AADHAAR_NUMBER	인도 Unique Identification Authority에서 발급한 아드하르 식별 번호.
INDIA_PERMANENT_ACCOUNT_NUMBER	인도의 소득세 번호(PAN).

인도네시아 데이터 형식

데이터 형식	설명
INDONESIA_IDENTITY_CARD_NUMBER	국가 식별자(인도네시아만 해당).

아일랜드 데이터 형식

데이터 형식	설명
IRELAND_DRIVING_LICENSE	운전면허증 번호(아일랜드만 해당).
IRELAND_PASSPORT_NUMBER	여권 번호(아일랜드만 해당).
IRELAND_PERSONAL_PUBLIC_SERVICE_NUMBER	아일랜드 개인 공공 서비스 번호(PPS).
IRELAND_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(아일랜드만 해당).
IRELAND_VALUE_ADDED_TAX	부가가치세(아일랜드만 해당).

이스라엘 데이터 형식

데이터 형식	설명
ISRAEL_IDENTIFICATION_NUMBER	국가 식별자(이스라엘만 해당).

이탈리아 데이터 형식

데이터 형식	설명
ITALY_BANK_ACCOUNT	은행 계좌 번호(이탈리아만 해당).
ITALY_DRIVING_LICENSE	운전면허증 번호(이탈리아만 해당).
ITALY_FISCAL_CODE	식별자 번호(이탈리아 코디체 피스칼레(Codice Fiscale)라고도 함).
ITALY_PASSPORT_NUMBER	여권 번호(이탈리아만 해당).
ITALY_VALUE_ADDED_TAX	부가가치세(이탈리아만 해당).

일본 데이터 유형

데이터 형식	설명
JAPAN_BANK_ACCOUNT	일본 은행 계좌.
JAPAN_DRIVING_LICENSE	일본의 운전면허증 번호.
JAPAN_MY_NUMBER	조세 행정, 사회 보장 행정 및 재해 대응을 위해 사용하는 일본 시민 또는 법인의 고유 식별자
JAPAN_PASSPORT_NUMBER	일본 여권 번호.

한국 데이터 형식

데이터 형식	설명
KOREA_PASSPORT_NUMBER	여권 번호(한국만 해당).
KOREA_RESIDENCE_REGISTRATION_NUMBER_FOR_CITIZENS	한국 주민등록번호.

데이터 형식	설명
KOREA_RESIDENCE_REGISTRATION_NUMBER_FOR_FOREIGNERS	외국인을 위한 한국의 외국인 등록 번호.

라트비아 데이터 형식

데이터 형식	설명
LATVIA_DRIVING_LICENSE	운전면허증 번호(라트비아만 해당).
LATVIA_PASSPORT_NUMBER	여권 번호(라트비아만 해당).
LATVIA_PERSONAL_IDENTIFICATION_NUMBER	개인 식별자 번호(라트비아만 해당).
LATVIA_VALUE_ADDED_TAX	부가가치세(라트비아만 해당).

리히텐슈타인 데이터 형식

데이터 형식	설명
LIECHTENSTEIN_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자(리히텐슈타인만 해당).
LIECHTENSTEIN_PASSPORT_NUMBER	여권 번호(리히텐슈타인만 해당).
LIECHTENSTEIN_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(리히텐슈타인만 해당).

리투아니아 데이터 형식

데이터 형식	설명
LITHUANIA_DRIVING_LICENSE	운전면허증 번호(리투아니아만 해당).

데이터 형식	설명
LITHUANIA_PERSONAL_IDENTIFICATION_NUMBER	개인 식별자 번호(리투아니아만 해당).
LITHUANIA_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(리투아니아만 해당).
LITHUANIA_VALUE_ADDED_TAX	부가가치세(리투아니아만 해당).

룩셈부르크 데이터 형식

데이터 형식	설명
LUXEMBOURG_DRIVING_LICENSE	운전면허증 번호(룩셈부르크만 해당).
LUXEMBOURG_NATIONAL_INDIVIDUAL_NUMBER	국가 식별자(룩셈부르크만 해당).
LUXEMBOURG_PASSPORT_NUMBER	여권 번호(룩셈부르크만 해당).
LUXEMBOURG_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(룩셈부르크만 해당).
LUXEMBOURG_VALUE_ADDED_TAX	부가가치세(룩셈부르크만 해당).

말레이시아 데이터 형식

데이터 형식	설명
MALAYSIA_MYKAD_NUMBER	국가 식별자(말레이시아만 해당).
MALAYSIA_PASSPORT_NUMBER	여권 번호(말레이시아만 해당).

몰타 데이터 형식

데이터 형식	설명
MALTA_DRIVING_LICENSE	운전면허증 번호(몰타만 해당).
MALTA_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자(몰타만 해당).
MALTA_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(몰타만 해당).
MALTA_VALUE_ADDED_TAX	부가가치세(몰타만 해당).

멕시코 데이터 형식

데이터 형식	설명
MEXICO_CLABE_NUMBER	멕시코 Clave Bancaria Estandarizada(CLABE) 은행 번호.
MEXICO_DRIVING_LICENSE	운전면허증 번호(멕시코만 해당).
MEXICO_PASSPORT_NUMBER	여권 번호(멕시코만 해당).
MEXICO_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(멕시코만 해당).
MEXICO_UNIQUE_POPULATION_REGISTER_CODE	멕시코의 고유 식별 코드(Clave Única de Registro de Población(CURP)).

네덜란드 데이터 형식

데이터 형식	설명
NETHERLANDS_CITIZEN_SERVICE_NUMBER	네덜란드의 시민 번호(BSN, burgerservicenummer).
NETHERLANDS_DRIVING_LICENSE	운전면허증 번호(네덜란드만 해당).
NETHERLANDS_PASSPORT_NUMBER	여권 번호(네덜란드만 해당).

데이터 형식	설명
NETHERLANDS_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(네덜란드만 해당).
NETHERLANDS_VALUE_ADDED_TAX	부가가치세(네덜란드만 해당).
NETHERLANDS_BANK_ACCOUNT	은행 계좌 번호(네덜란드만 해당).

뉴질랜드 데이터 형식

데이터 형식	설명
NEW_ZEALAND_DRIVING_LICENSE	운전면허증 번호(뉴질랜드만 해당).
NEW_ZEALAND_NATIONAL_HEALTH_INDEX_NUMBER	뉴질랜드 국가 건강 지수 번호.
NEW_ZEALAND_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(내국세 번호라고도 함, 뉴질랜드만 해당).

노르웨이 데이터 형식

데이터 형식	설명
NORWAY_BIRTH_NUMBER	노르웨이 국가 식별 번호.
NORWAY_DRIVING_LICENSE	운전면허증 번호(노르웨이만 해당).
NORWAY_HEALTH_INSURANCE_NUMBER	노르웨이 건강보험 번호.
NORWAY_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자 번호(노르웨이만 해당).
NORWAY_VALUE_ADDED_TAX	부가가치세(노르웨이만 해당).

필리핀 데이터 형식

데이터 형식	설명
PHILIPPINES_DRIVING_LICENSE	운전면허증 번호(필리핀만 해당).
PHILIPPINES_PASSPORT_NUMBER	여권 번호(필리핀만 해당).

폴란드 데이터 형식

데이터 형식	설명
POLAND_DRIVING_LICENSE	운전면허증 번호(폴란드만 해당).
POLAND_IDENTIFICATION_NUMBER	폴란드 식별자.
POLAND_PASSPORT_NUMBER	여권 번호(폴란드만 해당).
POLAND_REGON_NUMBER	REGON 식별자 번호(통계 식별 번호라고도 함).
POLAND_SSN	사회보장번호(폴란드인의 경우).
POLAND_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(폴란드만 해당).
POLAND_VALUE_ADDED_TAX	부가가치세(폴란드만 해당).

포르투갈 데이터 형식

데이터 형식	설명
PORTUGAL_DRIVING_LICENSE	운전면허증 번호(포르투갈만 해당).
PORTUGAL_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자 번호(포르투갈만 해당).
PORTUGAL_PASSPORT_NUMBER	여권 번호(포르투갈만 해당).
PORTUGAL_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(포르투갈만 해당).
PORTUGAL_VALUE_ADDED_TAX	부가가치세(포르투갈만 해당).

루마니아 데이터 형식

데이터 형식	설명
ROMANIA_DRIVING_LICENSE	운전면허증 번호(루마니아만 해당).
ROMANIA_NUMERICAL_PERSONAL_CODE	개인 식별자 번호(루마니아만 해당).
ROMANIA_PASSPORT_NUMBER	여권 번호(루마니아만 해당).
ROMANIA_VALUE_ADDED_TAX	부가가치세(루마니아만 해당).

싱가포르 데이터 형식

데이터 형식	설명
SINGAPORE_DRIVING_LICENSE	운전면허증 번호(싱가포르만 해당).
SINGAPORE_NATIONAL_REGISTRY_IDENTIFICATION_NUMBER	싱가포르 국가 등록 신분증.
SINGAPORE_PASSPORT_NUMBER	여권 번호(싱가포르만 해당).
SINGAPORE_UNIQUE_ENTITY_NUMBER	싱가포르 고유 법인 번호.

슬로바키아 데이터 형식

데이터 형식	설명
SLOVAKIA_DRIVING_LICENSE	운전면허증 번호(슬로바키아만 해당).
SLOVAKIA_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자 번호(슬로바키아만 해당).
SLOVAKIA_PASSPORT_NUMBER	여권 번호(슬로바키아만 해당).
SLOVAKIA_VALUE_ADDED_TAX	부가가치세(슬로바키아만 해당).

슬로베니아 데이터 형식

데이터 형식	설명
SLOVENIA_DRIVING_LICENSE	운전면허증 번호(슬로베니아만 해당).
SLOVENIA_PASSPORT_NUMBER	여권 번호(슬로베니아만 해당).
SLOVENIA_TAX_IDENTIFICATION_NUMBER	세금 식별 번호(슬로베니아만 해당).
SLOVENIA_UNIQUE_MASTER_CITIZEN_NUMBER	슬로베니아 시민의 고유 마스터 시민 번호(JMBG).
SLOVENIA_VALUE_ADDED_TAX	부가가치세(슬로베니아만 해당).

남아프리카 데이터 형식

데이터 형식	설명
SOUTH_AFRICA_PERSONAL_IDENTIFICATION_NUMBER	개인 식별자 번호(남아프리카 공화국만 해당).

일본 데이터 형식

데이터 형식	설명
SPAIN_BANK_ACCOUNT	은행 계좌 번호(스페인만 해당).
SPAIN_DNI	스페인의 국가 신분증(Documento Nacional de Identidad).
SPAIN_DRIVING_LICENSE	운전면허증 번호(스페인만 해당).
SPAIN_NIE	외국인 신분증 번호(NIE라고도 함, 스페인만 해당).
SPAIN_NIF	세금 식별 번호(NIF라고도 함, 스페인만 해당).

데이터 형식	설명
SPAIN_PASSPORT_NUMBER	여권 번호(스페인만 해당).
SPAIN_SSN	사회보장번호(스페인의 경우).
SPAIN_VALUE_ADDED_TAX	부가가치세(스페인만 해당).

스리랑카 데이터 형식

데이터 형식	설명
SRI_LANKA_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자(스리랑카만 해당).

스웨덴 데이터 형식

데이터 형식	설명
SWEDEN_DRIVING_LICENSE	운전면허증 번호(스웨덴만 해당).
SWEDEN_PASSPORT_NUMBER	여권 번호(스웨덴만 해당).
SWEDEN_PERSONAL_IDENTIFICATION_NUMBER	국가 식별자 번호(스웨덴만 해당).
SWEDEN_TAX_IDENTIFICATION_NUMBER	스웨덴 세금 식별 번호(personnummer).
SWEDEN_VALUE_ADDED_TAX	부가가치세(스웨덴만 해당).

스위스 데이터 형식

데이터 형식	설명
SWITZERLAND_AHV	스위스인의 사회보장번호(AHV).

데이터 형식	설명
SWITZERLAND_HEALTH_INSURANCE_NUMBER	스위스 건강보험 번호.
SWITZERLAND_PASSPORT_NUMBER	여권 번호(스위스만 해당).
SWITZERLAND_VALUE_ADDED_TAX	부가가치세(스위스만 해당).

태국 데이터 형식

데이터 형식	설명
THAILAND_PASSPORT_NUMBER	여권 번호(태국만 해당).
THAILAND_PERSONAL_IDENTIFICATION_NUMBER	개인 식별자 번호(태국만 해당).

터키 데이터 형식

데이터 형식	설명
TURKEY_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자 번호(튀르키예만 해당).
TURKEY_PASSPORT_NUMBER	여권 번호(튀르키예만 해당).
TURKEY_VALUE_ADDED_TAX	부가가치세(튀르키예만 해당).

우크라이나 데이터 형식

데이터 형식	설명
UKRAINE_INDIVIDUAL_IDENTIFICATION_NUMBER	고유 식별자(우크라이나만 해당).

데이터 형식	설명
UKRAINE_PASSPORT_NUMBER_DOMESTIC	국내 여권 번호(우크라이나만 해당).
UKRAINE_PASSPORT_NUMBER_INTERNATIONAL	국제 여권 번호(우크라이나만 해당).

아랍에미리트(UAE) 데이터 형식

데이터 형식	설명
UNITED_ARAB_EMIRATES_PERSONAL_NUMBER	개인 식별자 번호(UAE만 해당).

영국 데이터 유형

데이터 형식	설명
UK_BANK_ACCOUNT	영국 은행 계좌.
UK_BANK_SORT_CODE	영국 은행 지점 코드. 지점 코드(sort code)는 해당 통관 기관을 통해 각 국가 안의 은행 간에 송금을 라우팅하기 위해 사용하는 은행 코드입니다.
UK_DRIVING_LICENSE	영국 및 북아일랜드의 운전 면허증 번호(영국만 해당)
UK_ELECTORAL_ROLL_NUMBER	선거인단 번호(ERN)는 영국 선거 등록을 위해 개인에게 발급하는 식별 번호입니다. 이 번호의 형식은 영국 내각 사무처의 영국 정부 표준에 명시되어 있습니다.
UK_NATIONAL_HEALTH_SERVICE_NUMBER	국민 보건 서비스(NHS) 번호는 영국의 등록된 공중 보건 서비스 사용자에게 할당하는 고유 번호입니다.

데이터 형식	설명
UK_NATIONAL_INSURANCE_NUMBER	국민 보험 번호(NINO)는 영국(UK)에서 국민 보험 프로그램 또는 사회 보장 제도를 위해 개인을 식별하는 데 사용하는 번호입니다. 일부 경우에 NI No 또는 NINO라고도 합니다.
UK_PASSPORT_NUMBER	영국 여권 번호.
UK_UNIQUE_TAXPAYER_REFERENCE_NUMBER	영국(UK) 고유 납세자 참조(UTR) 번호. 영국 정부가 과세 제도를 관리하기 위해 사용하는 식별자.
UK_VALUE_ADDED_TAX	VAT는 최종 소비자가 부담하는 소비세입니다. VAT는 제조 및 유통 과정에서 이뤄지는 각 거래에 대해 지불합니다. 영국의 경우 VAT 번호는 사업체가 설립된 지역의 VAT 사무소에서 발급합니다.
UK_PHONE_NUMBER	영국 전화번호.

베네수엘라 데이터 형식

데이터 형식	설명
VENEZUELA_DRIVING_LICENSE	운전면허증 번호(베네수엘라만 해당).
VENEZUELA_NATIONAL_IDENTIFICATION_NUMBER	국가 식별자 번호(베네수엘라만 해당).
VENEZUELA_VALUE_ADDED_TAX	부가가치세(베네수엘라만 해당).

세분화된 데이터 탐지 사용하기

Note

세분화된 동작은 AWS Glue 3.0 및 4.0에서만 사용할 수 있습니다. 여기에는 AWS Glue Studio 경험도 포함됩니다. 2.0에서는 영구 감사 로그 변경도 사용할 수 없습니다.

모든 AWS Glue Studio 3.0 및 4.0 비주얼 작업에는 세분화된 작업 API를 자동으로 사용하는 스크립트가 생성됩니다.

민감 데이터 탐지 변환은 사용자가 정의하거나 AWS Glue에서 사전 정의한 엔터티를 탐지, 마스크 또는 제거하는 기능을 제공합니다. 또한 세분화된 액션을 통해 엔터티별로 특정 액션을 적용할 수 있습니다. 추가 혜택은 다음과 같습니다.

- 데이터가 감지되는 즉시 조치가 적용되므로 성능이 개선되었습니다.
- 특정 열을 포함하거나 제외하는 옵션.
- 부분 마스킹을 사용할 수 있습니다. 이렇게 하면 전체 문자열을 마스킹하는 대신 감지된 민감한 데이터 엔터티를 부분적으로 마스킹할 수 있습니다. 오프셋과 정규식이 있는 단순 매개 변수가 모두 지원됩니다.

다음은 다음 섹션에서 참조할 샘플 작업에 사용되는 민감한 데이터 탐지 API와 세분화된 조치의 코드 스니펫입니다.

Detect API- 세분화된 작업에는 새 `detectionParameters` 파라미터가 사용됩니다.

```
def detect(
  frame: DynamicFrame,
  detectionParameters: JsonObject,
  outputColumnName: String = "DetectedEntities",
  detectionSensitivity: String = "LOW"
): DynamicFrame = {}
```

민감한 데이터 탐지 API와 세밀한 조치 사용

`detect`를 사용하는 민감한 데이터 탐지 API는 제공된 데이터를 분석하고, 행 또는 열이 민감한 데이터 항목 유형인지 확인하고 각 엔터티 유형에 대해 사용자가 지정한 작업을 실행합니다.

탐지 API를 사용하여 세부적인 조치를 취하십시오

`detect` API를 사용하고 `outputColumnName` 및 `detectionParameters`를 지정합니다.

```
object GlueApp {
  def main(sysArgs: Array[String]) {
```

```

val spark: SparkContext = new SparkContext()
val glueContext: GlueContext = new GlueContext(spark)

// @params: [JOB_NAME]
val args = GlueArgParser.getResolvedOptions(sysArgs, Seq("JOB_NAME").toArray)
Job.init(args("JOB_NAME"), glueContext, args.asJava)

// Script generated for node S3 bucket. Creates DataFrame from data stored in
S3.
val S3bucket_node1 =
glueContext.getSourceWithFormat(formatOptions=JsonOptions("""{"quoteChar":
"\\"", "withHeader": true, "separator": ",", "optimizePerformance": false}"""),
connectionType="s3", format="csv", options=JsonOptions("""{"paths":
["s3://189657479688-ddevansh-pii-test-bucket/tiny_pii.csv"], "recurse": true}"""),
transformationContext="S3bucket_node1").getDynamicFrame()

// Script generated for node Detect Sensitive Data. Will run detect API for the
DataFrame
// detectionParameter contains information on which EntityType are being
detected
// and what actions are being applied to them when detected.
val DetectSensitiveData_node2 = EntityDetector.detect(
  frame = S3bucket_node1,
  detectionParameters = JsonOptions(
    """
    {
      "PHONE_NUMBER": [
        {
          "action": "PARTIAL_REDACT",
          "actionOptions": {
            "numLeftCharsToExclude": "3",
            "numRightCharsToExclude": "4",
            "redactChar": "#"
          },
          "sourceColumnsToExclude": [ "Passport No", "DL NO#" ]
        }
      ],
      "USA_PASSPORT_NUMBER": [
        {
          "action": "SHA256_HASH",
          "sourceColumns": [ "Passport No" ]
        }
      ],
      "USA_DRIVING_LICENSE": [

```

```

        {
            "action": "REDACT",
            "actionOptions": {
                "redactText": "USA_DL"
            },
            "sourceColumns": [ "DL NO#" ]
        }
    ]

    }
    """
),
outputColumnName = "DetectedEntities"
)

// Script generated for node S3 bucket. Store Results of detect to S3 location
val S3bucket_node3 = glueContext.getSinkWithFormat(connectionType="s3",
options=JsonOptions("""{"path": "s3://189657479688-ddevansh-pii-test-bucket/
test-output/", "partitionKeys": []}"""), transformationContext="S3bucket_node3",
format="json").writeDynamicFrame(DetectSensitiveData_node2)

Job.commit()
}

```

위 스크립트는 Amazon S3의 위치에서 DataFrame을 생성한 다음 detect API를 실행합니다. detect API는 Glue 객체로 표시되는 필드 detectionParameters(해당 엔터티에 사용할 모든 작업 설정을 나열하는 엔터티 이름 맵)를 AWS Glue의 JsonOptions 객체로 표시해야 하므로 API의 기능을 확장할 수도 있습니다.

엔터티별로 지정된 각 작업에 대해 엔터티/액션 조합을 적용할 모든 열 이름 목록을 입력합니다. 이렇게 하면 데이터 세트의 모든 열을 감지하도록 항목을 사용자 지정하고 특정 열에 없는 것으로 알려진 항목은 건너뛴 수 있습니다. 또한 이러한 엔터티에 대해 불필요한 탐지 호출을 수행하지 않아도 되므로 작업 성능이 향상되고 각 열과 엔터티 조합에 고유한 작업을 수행할 수 있습니다.

detectionParameters를 자세히 살펴보면 샘플 작업에는 세 가지 엔터티 유형이 있습니다. Phone Number, USA_PASSPORT_NUMBER, USA_DRIVING_LICENSE입니다. 이 같은 각 개체 유형에 대해 AWS Glue는 PARTIAL_REDACT, SHA256_HASH, REDACT, DETECT 같은 서로 다른 작업을 실행합니다. 각 개체 유형도 sourceColumns에 적용하거나 탐지된 경우 sourceColumnsToExclude에 적용해야 합니다.

Note

열당 내부 편집 작업(PARTIAL_REDACT, SHA256_HASH, 또는 REDACT) 하나만 사용할 수 있지만 해당 DETECT 작업은 이러한 모든 작업에 사용할 수 있습니다.

detectionParameters 필드 레이아웃은 다음과 같습니다.

```
ENTITY_NAME -> List[Actions]
{
  "ENTITY_NAME": [{
    Action, // required
    ColumnSpecs,
    ActionOptionsMap
  }],
  "ENTITY_NAME2": [{
    ...
  }]
}
```

actions 및 actionOptions의 유형은 다음과 같습니다.

```
DETECT
{
  # Required
  "action": "DETECT",
  # Optional, depending on action chosen
  "actionOptions": {
    // There are no actionOptions for DETECT
  },
  # 1 of below required, both can also used
  "sourceColumns": [
    "COL_1", "COL_2", ..., "COL_N"
  ],
  "sourceColumnsToExclude": [
    "COL_5"
  ]
}
```

SHA256_HASH

```
{
  # Required
  "action": "SHA256_HASH",
  # Required or optional, depending on action chosen
  "actionOptions": {
    // There are no actionOptions for SHA256_HASH
  },

  # 1 of below required, both can also used
  "sourceColumns": [
    "COL_1", "COL_2", ..., "COL_N"
  ],
  "sourceColumnsToExclude": [
    "COL_5"
  ]
}

REDACT
{
  # Required
  "action": "REDACT",
  # Required or optional, depending on action chosen
  "actionOptions": {
    // The text that is being replaced
    "redactText": "USA_DL"
  },

  # 1 of below required, both can also used
  "sourceColumns": [
    "COL_1", "COL_2", ..., "COL_N"
  ],
  "sourceColumnsToExclude": [
    "COL_5"
  ]
}

PARTIAL_REDACT
{
  # Required
  "action": "PARTIAL_REDACT",
  # Required or optional, depending on action chosen
  "actionOptions": {
    // number of characters to not redact from the left side
    "numLeftCharsToExclude": "3",
```

```

    // number of characters to not redact from the right side
    "numRightCharsToExclude": "4",
    // the partial redact will be made with this redacted character
    "redactChar": "#",
    // regex pattern for partial redaction
    "matchPattern": "[0-9]"
  },

  # 1 of below required, both can also used
  "sourceColumns": [
    "COL_1", "COL_2", ..., "COL_N"
  ],
  "sourceColumnsToExclude": [
    "COL_5"
  ]
}

```

스크립트가 실행되면 결과가 지정된 Amazon S3 위치에 출력됩니다. Amazon S3에서 데이터를 볼 수 있지만 선택한 엔터티 유형은 선택한 작업에 따라 구분됩니다. 이 경우 다음과 같은 행이 있을 것입니다.

```

{
  "Name": "Colby Schuster",
  "Address": "39041 Antonietta Vista, South Rodgerside, Nebraska 24151",
  "Car Owned": "Fiat",
  "Email": "Kitty46@gmail.com",
  "Company": "O'Reilly Group",
  "Job Title": "Dynamic Functionality Facilitator",
  "ITIN": "991-22-2906",
  "Username": "Cassandre.Kub43",
  "SSN": "914-22-2906",
  "DOB": "2020-08-27",
  "Phone Number": "1-2#####1718",
  "Bank Account No": "69741187",
  "Credit Card Number": "6441-6289-6867-2162-2711",
  "Passport No": "94f311e93a623c72ccb6fc46cf5f5b0265ccb42c517498a0f27fd4c43b47111e",
  "DL NO#": "USA_DL"
}

```

위 스크립트에서는 Phone Number를 사용하여 부분적으로 #으로 수정되었습니다. Passport No가 SHA256 해시로 변경되었습니다. DL NO# 는 미국 운전면허증 번호로 인식되어 detectionParameters에 명시된 대로 "USA_DL"로 수정되었습니다.

Note

classifyColumns API는 API의 특성상 세분화된 작업에는 사용할 수 없습니다. 이 API는 열 샘플링(사용자가 조정 가능하지만 기본값 사용)을 수행하여 탐지를 더 빠르게 수행합니다. 이러한 이유로 세밀한 작업을 수행하려면 모든 값을 반복해야 합니다.

영구 감사 로그

세분화된 작업과 함께 도입된 새로운 기능(일반 API를 사용할 때도 사용 가능)은 영구 감사 로그의 존재입니다. 현재 detect API를 실행하면 PII 탐지 메타데이터가 포함된 추가 열(DetectedEntities가 기본값이지만 outputColumnName을 통해 사용자 지정 가능) 매개변수가 추가됩니다. 이제 여기에는, DETECT, PARTIAL_REDACT, SHA256_HASH, REDACT 중 하나인 "actionUsed" 메타데이터 키가 있습니다.

```
"DetectedEntities": {
  "Credit Card Number": [
    {
      "entityType": "CREDIT_CARD",
      "actionUsed": "DETECT",
      "start": 0,
      "end": 19
    }
  ],
  "Phone Number": [
    {
      "entityType": "PHONE_NUMBER",
      "actionUsed": "REDACT",
      "start": 0,
      "end": 14
    }
  ]
}
```

예를 들어 detect(entityTypesToDetect, outputColumnName)와 같이 세밀한 조치 없이 API를 사용하는 고객도 결과 데이터 프레임에서 이 영구 감사 로그를 볼 수 있습니다.

세분화된 조치가 포함된 API를 사용하는 고객은 수정 여부에 관계없이 모든 작업을 볼 수 있습니다. 예제

```
+-----+-----
+-----+-----
+
| Credit Card Number | Phone Number |
|                               | DetectedEntities |
+-----+-----
+-----+-----
+
| 622126741306XXXX | +12#####7890 | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":16}], "Phone
Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":12}]} |
| 6221 2674 1306 XXXX | +12#####7890 | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":19}], "Phone
Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":14}]} |
| 6221-2674-1306-XXXX | 22#####7890 | {"Credit Card Number":
[{"entityType":"CREDIT_CARD","actionUsed":"PARTIAL_REDACT","start":0,"end":19}], "Phone
Number":
[{"entityType":"PHONE_NUMBER","actionUsed":"PARTIAL_REDACT","start":0,"end":14}]} |
+-----+-----
+-----+-----
+
```

DetectedEntities 열을 보지 않으려면 사용자 지정 스크립트에 추가 열을 놓기만 하면 됩니다.

AWS Glue Visual Job API

AWS Glue는 고객이 시각적 단계 워크플로를 나타내는 JSON 객체의 AWS Glue API를 사용하여 데이터 통합 작업을 생성할 수 있게 해주는 API를 제공합니다. 그러면 고객은 AWS Glue Studio의 시각적 편집기를 사용하여 이러한 작업을 수행할 수 있습니다.

시각적 작업 API 데이터 형식에 대한 자세한 내용은 [Visual Job API](#)를 참조하세요.

주제

- [API 설계 및 CRUD API](#)
- [시작하기](#)

- [시각적 작업 제한 사항](#)

API 설계 및 CRUD API

이제 CreateJob 및 UpdateJob [API](#) 에서 선택적 추가 파라미터인 codeGenConfigurationNodes를 지원합니다. 이 필드에 비어 있지 않은 JSON 구조를 제공하면 생성된 작업에 대한 DAG가 AWS Glue Studio에 등록되고 관련 코드가 생성됩니다. 작업 생성 시 이 필드의 Null 값이나 빈 문자열은 무시됩니다.

codegenConfigurationNodes 필드에 대한 업데이트는 CreateJob과 비슷한 방식으로 UpdateJob AWS Glue API를 통해 수행됩니다. DAG가 원하는 대로 변경된 UpdateJob에 전체 필드를 지정해야 합니다. 제공된 Null 값은 무시되며 DAG에 대한 업데이트가 수행되지 않습니다. 빈 구조나 문자열을 제공하면 codeGenConfigurationNodes가 비어 있는 것으로 설정되고 이전 DAG는 모두 제거됩니다. GetJob API는 DAG를 반환합니다(존재하는 경우). DeleteJob API는 연결된 모든 DAG도 삭제합니다.

시작하기

작업을 생성하려면 [CreateJob 작업](#)을 사용합니다. CreateJob 요청 입력에는 JSON에서 DAG 객체를 지정할 수 있는 추가 필드인 'codegenConfigurationNodes'가 있습니다.

다음과 같은 사항에 유의하세요.

- 'codegenConfigurationNodes' 필드는 nodeId를 노드에 매핑한 것입니다.
- 각 노드는 노드 종류를 식별하는 키로 시작합니다.
- 노드는 단일 유형만 가능하므로 하나의 키만 지정할 수 있습니다.
- 입력 필드에는 현재 노드의 상위 노드가 포함됩니다.

다음은 CreateJob 입력의 JSON 표현입니다.

```
{
  "node-1": {
    "S3CatalogSource": {
      "Table": "csvFormattedTable",
      "PartitionPredicate": "",
      "Name": "S3 bucket",
      "AdditionalOptions": {},
      "Database": "myDatabase"
    }
  },
  "node-3": {
```

```
"S3DirectTarget": {
  "Inputs": ["node-2"],
  "PartitionKeys": [],
  "Compression": "none",
  "Format": "json",
  "SchemaChangePolicy": { "EnableUpdateCatalog": false },
  "Path": "",
  "Name": "S3 bucket"
},
"node-2": {
  "ApplyMapping": {
    "Inputs": ["node-1"],
    "Name": "ApplyMapping",
    "Mapping": [
      {
        "FromType": "long",
        "ToType": "long",
        "Dropped": false,
        "ToKey": "myheader1",
        "FromPath": ["myheader1"]
      },
      {
        "FromType": "long",
        "ToType": "long",
        "Dropped": false,
        "ToKey": "myheader2",
        "FromPath": ["myheader2"]
      },
      {
        "FromType": "long",
        "ToType": "long",
        "Dropped": false,
        "ToKey": "myheader3",
        "FromPath": ["myheader3"]
      }
    ]
  }
}
```

작업 업데이트 및 가져오기

UpdateJob에도 'codegenConfigurationNodes' 필드가 있으므로 입력 형식은 동일합니다. [UpdateJob](#) 작업을 참조하세요.

GetJob 작업도 동일한 형식으로 'codegenConfigurationNodes' 필드를 반환합니다. [GetJob](#) 작업을 참조하세요.

시각적 작업 제한 사항

'codegenConfigurationNodes' 파라미터가 기존 API에 추가되었으므로 해당 API의 모든 제한 사항이 상속됩니다. 또한 codegenConfigurationNodes와 일부 노드는 크기가 제한됩니다. 자세한 내용은 [작업 구조](#)를 참조하세요.

Ray 스크립트 프로그래밍

AWS Glue를 사용하면 Ray 스크립트를 쉽게 쓰고 실행할 수 있습니다. 이 섹션에서는 AWS Glue for Ray에서 사용할 수 있는 지원되는 Ray 기능을 설명합니다. Python에서 Ray 스크립트를 프로그래밍합니다.

사용자 정의 스크립트는 작업 정의의 Runtime 필드에 정의된 Ray 버전과 호환되어야 합니다. 작업 API에서 Runtime에 대한 자세한 내용은 [the section called “작업”](#) 섹션을 참조하세요. 각 런타임 환경에 대한 자세한 내용은 [the section called “지원되는 Ray 런타임 환경”](#) 섹션을 참조하세요.

주제

- [자습서: AWS Glue for Ray에서 ETL 스크립트 작성](#)
- [AWS Glue for Ray에서 Ray Core 및 Ray Data 사용](#)
- [Ray 작업에 파일 및 Python 라이브러리 제공](#)
- [Ray 작업의 데이터에 연결](#)

자습서: AWS Glue for Ray에서 ETL 스크립트 작성

Ray는 기본적으로 Python에서 배포된 작업을 작성하고 규모를 조정할 수 있는 기능을 제공합니다. AWS Glue for Ray는 작업과 대화형 세션 모두에서 액세스할 수 있는 서버리스 Ray 환경을 제공합니다 (Ray 대화형 세션은 평가판 단계임). AWS Glue 작업 시스템은 일정에 따라, 트리거 또는 AWS Glue 콘솔을 통해 작업을 관리하고 실행할 수 있는 일관된 방법을 제공합니다.

이러한 AWS Glue 도구를 결합하면 추출, 전환, 적재(ETL) 워크로드에 사용할 수 있는 강력한 도구 모음이 생성되며, AWS Glue의 사용 사례에 많이 사용됩니다. 이 자습서에서는 이 솔루션을 구성하는 기본 사항을 학습합니다.

또한 ETL 워크로드에서 AWS Glue for Spark를 사용할 수 있도록 지원합니다. AWS Glue for Spark 스크립트 작성에 대한 자습서는 [the section called “자습서: Spark 스크립트 작성”](#) 섹션을 참조하세요. 사용 가능한 엔진에 대한 자세한 내용은 [the section called “AWS Glue for Spark 및 AWS Glue for Ray”](#) 섹션을 참조하세요. Ray는 분석, 기계 학습(ML) 및 애플리케이션 개발 분야에서 다양한 종류의 작업을 처리할 수 있습니다.

이 자습서에서는 Amazon Simple Storage Service(S3)에 호스팅된 CSV 데이터 세트를 추출, 변환 및 적재합니다. 먼저 퍼블릭 Amazon S3 버킷에 저장된 New York City Taxi and Limousine Commission (TLC) Trip Record Data 데이터 세트로 시작합니다. 이 데이터 세트에 대한 자세한 내용은 [AWS의 오픈 데이터 레지스트리](#)를 참조하세요.

Ray Data 라이브러리에서 사용할 수 있는 미리 정의된 변환을 사용하여 데이터를 변환합니다. Ray Data는 Ray에서 설계한 데이터 세트 준비 라이브러리이며 AWS Glue for Ray 환경에 기본적으로 포함되어 있습니다. 기본적으로 포함된 라이브러리에 대한 자세한 내용은 [the section called “Ray 작업과 함께 제공되는 모듈”](#) 섹션을 참조하세요. 그런 다음 사용자가 제어하는 Amazon S3 버킷에 변환된 데이터를 작성합니다.

사전 조건 - 이 자습서에서는 AWS Glue 및 Amazon S3에 액세스할 수 있는 AWS 계정이 필요합니다.

1단계: Amazon S3에 출력 데이터를 보관할 버킷 생성

이 자습서에서 생성한 데이터의 싱크 역할을 하기 위해 사용자가 제어하는 Amazon S3 버킷이 필요합니다. 다음 절차에 따라 이 버킷을 생성할 수 있습니다.

Note

사용자가 제어하는 기존 버킷에 데이터를 쓰려면 이 단계를 건너뛰면 됩니다. 이후 단계에서 사용하기 위해 기존 버킷 이름(*yourBucketName*)을 기록해둡니다.

Ray 작업 출력을 위한 버킷을 생성하려면

- Amazon S3 사용 설명서의 [버킷 생성](#)에 나와 있는 단계에 따라 버킷을 생성합니다.
 - 버킷 이름을 선택할 때는 *yourBucketName*을 기록해둡니다. 이 이름은 이후 단계에서 참조합니다.
 - 다른 구성의 경우 Amazon S3 콘솔에서 제공하는 제안된 설정도 이 자습서에서 작동할 수 있습니다.

예를 들어 Amazon S3 콘솔의 버킷 생성 대화 상자는 다음과 비슷할 수 있습니다.

Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name

Bucket name must be globally unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

AWS Region

2단계: Ray 작업에 대한 IAM 역할 생성

작업에는 다음을 포함하는 AWS Identity and Access Management(IAM) 역할이 필요합니다.

- AWSGlueServiceRole 관리형 정책에서 부여한 권한. 이는 AWS Glue 작업을 실행하는 데 필요한 기본 권한입니다.
- `nyc-tlc/*` Amazon S3 리소스에 대한 Read 액세스 수준 권한.
- `yourBucketName/*` Amazon S3 리소스에 대한 Write 액세스 수준 권한.
- `glue.amazonaws.com` 보안 주체가 역할을 수임할 수 있도록 하는 신뢰 관계.

다음 절차에 따라 이 역할을 생성할 수 있습니다.

AWS Glue for Ray 작업에 대한 IAM 역할을 생성하려면

Note

다양한 절차에 따라 IAM 역할을 생성할 수 있습니다. IAM 리소스를 프로비저닝하는 방법에 대한 자세한 내용이나 옵션은 [AWS Identity and Access Management 설명서](#)를 참조하세요.

1. IAM 사용 설명서의 [시각적 편집기를 사용하여 IAM 정책\(콘솔\) 생성](#)에 나온 단계를 수행하여 이전에 설명한 Amazon S3 권한을 정의하는 정책을 생성합니다.
 - 서비스를 선택할 때 Amazon S3를 선택합니다.
 - 정책에 대한 권한을 선택할 때 앞서 언급한 다음 리소스에 대해 다음 작업 세트를 연결합니다.
 - `nyc-tlc/*` Amazon S3 리소스에 대한 읽기 액세스 수준 권한.
 - `yourBucketName/*` Amazon S3 리소스에 대한 쓰기 액세스 수준 권한.
 - 정책 이름을 선택할 때는 `YourPolicyName`을 기록해둡니다. 이 이름은 이후 단계에서 참조합니다.
2. IAM 사용 설명서의 [AWS 서비스에 대한 역할 생성\(콘솔\)](#)에 나온 단계에 따라 AWS Glue for Ray 작업의 역할을 생성합니다.
 - 신뢰할 수 있는 AWS 서비스 엔터티를 선택할 때 Glue를 선택합니다. 그러면 작업에 필요한 신뢰 관계가 자동으로 채워집니다.
 - 권한 정책에 대한 정책을 선택할 때 다음 정책을 연결합니다.
 - `AWSGlueServiceRole`
 - `YourPolicyName`
 - 역할 이름을 선택할 때는 `YourRoleName`을 기록해둡니다. 이 이름은 이후 단계에서 참조합니다.

3단계: AWS Glue for Ray 작업 생성 및 실행

이 단계에서는 AWS Management Console을 사용하여 AWS Glue 작업을 생성하고 샘플 스크립트를 함께 제공하며 작업을 실행합니다. 작업을 생성하면 콘솔에 Ray 스크립트를 저장, 구성 및 편집할 수 있는 위치가 생성됩니다. 작업 생성에 대한 자세한 내용은 [the section called “콘솔로 로그인합니다”](#) 섹션을 참조하세요.

이 자습서에서는 다음과 같은 ETL 시나리오를 다룹니다. New York City TLC Trip Record 데이터 세트에서 2022년 1월 기록을 읽고, 기존 열의 데이터를 결합하여 데이터 세트에 새 열(`tip_rate`)을 추가한 다음, 현재 분석과 관련이 없는 열을 여러 개 제거하고 결과를 `yourBucketName`에 쓰려고 합니다. 다음 Ray 스크립트에서 다음 단계를 수행합니다.

```
import ray
import pandas
from ray import data

ray.init('auto')
```

```

ds = ray.data.read_csv("s3://nyc-tlc/opendata_repo/opendata_webconvert/yellow/
yellow_tripdata_2022-01.csv")

# Add the given new column to the dataset and show the sample record after adding a new
column
ds = ds.add_column( "tip_rate", lambda df: df["tip_amount"] / df["total_amount"])

# Dropping few columns from the underlying Dataset
ds = ds.drop_columns(["payment_type", "fare_amount", "extra", "tolls_amount",
"improvement_surcharge"])

ds.write_parquet("s3://yourBucketName/ray/tutorial/output/")

```

AWS Glue for Ray 작업을 생성하고 실행하려면

1. AWS Management Console에서 AWS Glue 랜딩 페이지로 이동합니다.
2. 측면 탐색 창에서 ETL 작업을 선택합니다.
3. 다음 그림과 같이 작업 생성에서 Ray 스크립트 편집기를 선택한 다음 생성을 선택합니다.

The screenshot shows the 'Create job' page in AWS Glue Studio. It features a 'Create' button in the top right corner. Below the header, there are six options for creating a job, each with a radio button and a description:

- Visual with a source and target: Start with a source, ApplyMapping transform, and target.
- Visual with a blank canvas: Author using an interactive visual interface.
- Spark script editor: Write or upload your own Spark code.
- Python Shell script editor: Write or upload your own Python shell script.
- Jupyter Notebook: Write your own code in a Jupyter Notebook for interactive development.
- Ray script editor: Write your own code to run on Ray.

Below these options is an 'Options' section with two radio buttons:

- Create a new script with boilerplate code
- Upload and edit an existing script: Choose a local file.

4. 스크립트의 전체 텍스트를 스크립트 창에 붙여넣고 기존 텍스트를 모두 바꿉니다.
5. 작업 세부 정보로 이동하여 IAM 역할 속성을 *YourRoleName*으로 설정합니다.
6. 저장을 선택한 다음 실행을 선택합니다.

4단계: 출력 검사

AWS Glue 작업을 실행한 후에는 출력이 이 시나리오의 예상과 일치하는지 확인해야 합니다. 다음 구문을 사용하면 이 작업을 수행할 수 있습니다.

Ray 작업이 성공적으로 실행되었는지 확인하려면

1. 작업 세부 정보 페이지에서 실행으로 이동합니다.
2. 몇 분 후 실행 상태가 성공인 실행이 표시됩니다.
3. Amazon S3 콘솔(<https://console.aws.amazon.com/s3/>)로 이동하여 *yourBucketName*을 검사합니다. 출력 버킷에 쓴 파일이 표시됩니다.
4. Parquet 파일을 읽고 해당 콘텐츠를 확인합니다. 기존 도구를 사용하여 이 작업을 수행할 수 있습니다. Parquet 파일을 검증하는 프로세스가 없는 경우 Spark 또는 Ray(평가판)를 사용하여 AWS Glue 대화형 세션이 있는 AWS Glue 콘솔에서 이 작업을 수행할 수 있습니다.

대화형 세션에서는 선택한 엔진에 따라 기본적으로 제공되는 Ray Data, Spark 또는 pandas 라이브러리에 액세스할 수 있습니다. 파일 콘텐츠를 확인하려면 해당 라이브러리에서 사용할 수 있는 일반적인 검사 방법(예: count, schema 및 show)을 사용할 수 있습니다. 콘솔에서의 대화형 세션에 대한 자세한 내용은 [AWS Glue Studio 및 AWS Glue에서 노트북 사용](#)을 참조하세요.

파일이 버킷에 기록된 것을 확인했으므로 출력에 문제가 있는 경우 IAM 구성과 관련이 없다고 비교적 확신할 수 있습니다. 관련 파일에 액세스할 수 있도록 *yourRoleName*으로 세션을 구성합니다.

예상된 결과를 얻지 못하면 이 안내서의 문제 해결 콘텐츠를 살펴보고 오류의 원인을 파악하고 해결합니다. 작업 실행 오류 상태를 해석하려면 [the section called “작업 실행 상태”](#) 섹션을 참조하세요. [AWS Glue 문제 해결](#) 장에서 문제 해결 콘텐츠를 찾을 수 있습니다. Ray 작업과 관련된 특정 오류에 대해서는 문제 해결 장의 [the section called “Ray 오류 해결”](#) 섹션을 참조하세요.

다음 단계

지금까지 AWS Glue for Ray를 사용하여 ETL 프로세스를 처음부터 끝까지 확인하고 수행했습니다. 다음 리소스를 통해 AWS Glue for Ray가 대규모로 데이터를 변환하고 해석하기 위해 제공하는 도구를 이해할 수 있습니다.

- Ray의 작업 모델에 대한 자세한 내용은 [the section called “AWS Glue for Ray에서 Ray Core 및 Ray Data 사용”](#) 섹션을 참조하세요. Ray 작업 사용에 대한 더 많은 경험을 쌓으려면 Ray Core 설명서의 예제를 따르세요. Ray 설명서의 [Ray Core: Ray Tutorials and Examples \(2.4.0\)](#)를 참조하세요.

- AWS Glue for Ray에서 사용 가능한 데이터 관리 라이브러리에 대한 지침은 [the section called “데이터에 연결”](#) 섹션을 참조하세요. Ray Data를 사용하여 데이터 세트를 변환하고 작성하는 데 더 많은 경험을 쌓으려면 Ray Data 설명서의 예제를 따르세요. [Ray Data: Examples \(2.4.0\)](#)를 참조하세요.
- AWS Glue for Ray 작업 구성에 대한 자세한 내용은 [the section called “Ray 작업 사용”](#) 섹션을 참조하세요.
- AWS Glue for Ray 스크립트 작성에 대한 자세한 내용은 이 섹션의 설명서를 계속 읽어보세요.

AWS Glue for Ray에서 Ray Core 및 Ray Data 사용

Ray는 클러스터 전체에 작업을 분산하여 Python 스크립트 규모를 스케일 업하는 프레임워크입니다. Ray는 다양한 문제에 대한 솔루션으로 사용할 수 있으므로 Ray는 특정 작업을 최적화하는 라이브러리를 제공합니다. AWS Glue에서는 Ray를 사용하여 대규모 데이터 세트를 변환하는 데 중점을 둡니다. AWS Glue에서는 이 작업을 용이하게 하기 위해 Ray Data 및 Ray Core의 일부에 대한 지원을 제공합니다.

Ray Core란 무엇인가요?

분산 애플리케이션을 구축하는 첫 번째 단계는 동시에 수행할 수 있는 작업을 식별하고 정의하는 것입니다. Ray Core에는 동시에 수행할 수 있는 작업을 정의하는 데 사용하는 Ray의 일부 기능이 포함되어 있습니다. Ray는 Ray에서 제공하는 도구를 익히는 데 사용할 수 있는 참조 정보 및 빠른 시작 정보를 제공합니다. 자세한 내용은 [What is Ray Core?](#) 및 [Ray Core Quick Start](#)를 참조하세요. Ray에서 동시 작업을 효과적으로 정의하는 방법에 대한 자세한 내용은 [Tips for first-time users](#)를 참조하세요.

Ray 작업 및 액터

AWS Glue for Ray 설명서에서는 Ray의 핵심 개념인 작업 및 액터를 언급할 수 있습니다.

Ray는 Python 함수와 클래스를 분산 컴퓨팅 시스템의 구성 요소로 사용합니다. Python 함수와 변수가 클래스에서 사용될 때 '메서드'와 '속성' 역할을 하는 것과 마찬가지로, Ray에서 작업자에게 코드를 보낼 때 사용되는 함수는 '작업'이 되고 클래스는 '액터'가 됩니다. `@ray.remote` 주석을 통해 Ray에서 사용할 수 있는 함수와 클래스를 식별할 수 있습니다.

작업과 액터는 구성이 가능하고, 수명 주기가 있으며, 수명 주기 동안 컴퓨팅 리소스를 차지합니다. 문제의 근본 원인을 찾을 때 작업이나 액터 수준까지 오류를 발생시키는 코드를 추적할 수 있습니다. 따라서 이러한 용어는 AWS Glue for Ray 작업의 구성, 모니터링 또는 디버깅 방법을 배울 때 언급될 수 있습니다.

작업과 액터를 효과적으로 사용하여 분산 애플리케이션을 구축하는 방법을 배우려면 Ray 설명서의 [Key Concepts](#)를 참조하세요.

AWS Glue for Ray의 Ray Core

AWS Glue for Ray 환경은 클러스터 형성 및 규모 조정뿐만 아니라 로그 수집 및 시각화를 관리합니다. AWS에서도 이러한 항목을 관리하기 때문에 오픈 소스 클러스터에서 이러한 항목을 해결하는 데 사용되는 Ray Core의 API에 대한 액세스 및 지원이 제한됩니다.

관리형 Ray 2.4 런타임 환경에서 AWS는 다음을 지원하지 않습니다.

- [Ray Core CLI](#)
- [Ray State CLI](#)
- ray.util.metrics Prometheus 지표 유틸리티 메서드:
 - [Counter](#)
 - [Gauge](#)
 - [Histogram](#)
- 기타 디버깅 도구:
 - [ray.util.pdb.set_trace](#)
 - [ray.util.inspect_serializability](#)
 - [ray.timeline](#)

Ray Data란 무엇인가요?

데이터 소스 및 대상에 연결하고, 데이터 세트를 처리하며, 일반적인 변환을 시작할 때 Ray Data는 Ray를 사용하여 Ray 데이터 세트 변환 관련 문제를 해결할 수 있는 간단한 방법입니다. Ray Data 사용에 대한 자세한 내용은 [Ray Datasets: Distributed Data Preprocessing](#)을 참조하세요.

Ray Data 또는 기타 도구를 사용하여 데이터에 액세스할 수 있습니다. Ray에서 데이터에 액세스하는 방법에 대한 자세한 내용은 [the section called “데이터에 연결”](#) 섹션을 참조하세요.

AWS Glue for Ray의 Ray Data

Ray Data는 관리형 Ray 2.4 런타임 환경에서 기본적으로 지원되고 제공됩니다. 제공된 모듈에 대한 자세한 내용은 [the section called “Ray 작업과 함께 제공되는 모듈”](#) 섹션을 참조하세요.

Ray 작업에 파일 및 Python 라이브러리 제공

이 섹션에서는 AWS Glue Ray 작업에서 Python 라이브러리를 사용하는 데 필요한 정보를 제공합니다. 모든 Ray 작업에 기본적으로 포함된 특정 공용 라이브러리를 사용할 수 있습니다. Ray 작업에 자체 Python 라이브러리를 제공할 수도 있습니다.

Ray 작업과 함께 제공되는 모듈

다음과 같은 제공된 패키지를 사용하여 Ray 작업에서 데이터 통합 워크플로를 수행할 수 있습니다. 이러한 패키지는 Ray 작업에서 기본적으로 사용할 수 있습니다.

AWS Glue version 4.0

AWS Glue 4.0에서 Ray(Ray2.4 런타임) 환경은 다음 패키지를 제공합니다.

- boto3 == 1.26.133
- ray == 2.4.0
- pyarrow == 11.0.0
- pandas == 1.5.3
- numpy == 1.24.3
- fsspec == 2023.4.0

이 목록에는 ray[data] == 2.4.0과 함께 설치할 모든 패키지가 포함되어 있습니다. Ray Data는 기본적으로 지원됩니다.

Ray 작업에 파일 제공

--working-dir 파라미터를 사용하여 Ray 작업에 파일을 제공할 수 있습니다. Amazon S3에 호스팅된 .zip 파일의 경로를 이 파라미터에 제공합니다. .zip 파일 내에서 파일은 단일 최상위 디렉터리에 포함되어야 합니다. 최상위 수준에 다른 파일은 있을 수 없습니다.

스크립트를 실행하기 전에 파일이 각 Ray 노드에 배포됩니다. 이때 각 Ray 노드에서 사용할 수 있는 디스크 공간에 미치는 영향을 고려합니다. 사용 가능한 디스크 공간은 작업 구성에 설정된 WorkerType에 따라 결정됩니다. 작업 데이터를 대규모로 제공하려는 경우 이 메커니즘은 올바른 솔루션이 아닙니다. 작업에 데이터를 제공하는 방법에 대한 자세한 내용은 [the section called “데이터에 연결”](#) 섹션을 참조하세요.

working_dir 파라미터를 통해 Ray에 디렉터리를 제공한 것처럼 파일에도 액세스할 수 있습니다. 예를 들어 .zip 파일의 최상위 디렉터리에서 이름이 sample.txt인 파일을 읽으려면 다음을 직접 호출할 수 있습니다.

```
@ray.remote
def do_work():
```

```
f = open("sample.txt", "r")
print(f.read())
```

`working_dir`에 대한 자세한 내용은 [Ray 설명서](#)를 참조하세요. 이 기능은 Ray의 기본 기능과 유사하게 작동합니다.

Ray 작업을 위한 추가 Python 모듈

PyPI의 추가 모듈

Ray 작업에서는 Python 패키지 설치 프로그램(`pip3`)을 사용하여 Ray 스크립트에서 사용할 추가 모듈을 설치합니다. `--pip-install` 파라미터를 쉼표로 구분된 Python 모듈 목록과 함께 사용하여 새 모듈을 추가하거나 기존 모듈의 버전을 변경할 수 있습니다.

예를 들어 새 `scikit-learn` 모듈을 업데이트하거나 추가하려면 다음 키 값 페어를 사용합니다.

```
"--pip-install", "scikit-learn==0.21.3"
```

사용자 지정 모듈이나 사용자 지정 패치가 있는 경우, Amazon S3에서 `--s3-py-modules` 파라미터를 사용하여 자체 라이브러리를 배포할 수 있습니다. 배포를 업로드하기 전에 다시 패키징하고 다시 구축해야 할 수도 있습니다. [the section called "Ray 작업에 Python 코드 포함"](#)의 지침을 따릅니다.

Amazon S3의 사용자 지정 배포

사용자 지정 배포는 종속성에 대한 Ray 패키징 지침을 준수해야 합니다. 다음 섹션에서 이러한 배포를 빌드하는 방법을 확인할 수 있습니다. Ray에서 종속성을 설정하는 방법에 대한 자세한 내용은 Ray 설명서의 [환경 종속성](#)을 참조하세요.

콘텐츠를 평가한 후 사용자 지정 배포 파일을 포함하려면 작업의 IAM 역할에서 사용할 수 있는 버킷에 배포 가능 파일을 업로드합니다. 파라미터 구성에서 Python zip 아카이브에 대한 Amazon S3 경로를 지정합니다. 배포 파일을 여러 개 제공하는 경우 쉼표로 구분합니다. 예:

```
"--s3-py-modules", "s3://s3bucket/pythonPackage.zip"
```

제한 사항

Ray 작업에서는 작업 환경의 기본 코드 컴파일을 지원하지 않습니다. Python 종속성이 컴파일된 기본 코드에 전이적으로 종속되는 경우 이로 인해 제한될 수 있습니다. Ray 작업에서는 제공된 바이너리를 실행할 수 있지만 ARM64 기반 Linux용으로 컴파일해야 합니다. 즉, `aarch64manylinux Wheel`의 콘텐츠를 사용할 수 있습니다. Wheel을 Ray 표준에 따라 다시 패키징하여 컴파일된 형식으로 기본 종속

성을 제공할 수 있습니다. 일반적으로 이는 아카이브의 루트에 단 하나의 폴더만 남도록 `dist-info` 폴더를 제거하는 것을 의미합니다.

이 파라미터를 사용하여 `ray` 또는 `ray[data]` 버전을 업그레이드할 수 없습니다. 새 버전의 Ray를 사용하려면 지원이 출시된 후 작업의 런타임 필드를 변경해야 합니다. 지원되는 Ray 버전에 대한 자세한 내용은 [the section called “AWS Glue 버전”](#) 섹션을 참조하세요.

Ray 작업에 Python 코드 포함

Python Software Foundation은 다양한 런타임에서 사용할 Python 파일을 패키징하기 위한 표준화된 동작을 제공합니다. Ray는 사용자가 숙지해야 하는 패키징 표준에 대한 제한 사항을 도입했습니다. AWS Glue에서는 Ray에 지정된 표준 이외의 패키징 표준을 지정하지 않습니다. 다음 지침은 간단한 Python 패키지 패키징 관련 표준 지침을 제공합니다.

파일을 `.zip` 아카이브에 패키징합니다. 디렉터리는 아카이브의 루트에 있어야 합니다. 아카이브의 루트 수준에 다른 파일이 없어야 합니다. 그렇지 않으면 예기치 않은 동작이 발생합니다. 루트 디렉터리는 패키지이며 패키지 이름은 패키지를 가져올 때 Python 코드를 참조하는 데 사용됩니다.

`--s3-py-modules`를 사용하여 Ray 작업에 이 양식의 배포를 제공하는 경우 Ray 스크립트의 패키지 파일에서 Python 코드를 가져올 수 있습니다.

패키지는 몇 가지 Python 파일을 포함하는 단일 Python 모듈을 제공하거나 여러 모듈을 함께 패키징할 수 있습니다. PyPI의 라이브러리와 같은 종속성을 다시 패키징할 때 해당 패키지 내에 숨겨진 파일 및 메타데이터 디렉터리가 있는지 확인합니다.

Warning

특정 OS 동작으로 인해 이러한 패키징 지침을 제대로 수행하지 못할 수 있습니다.

- OSX는 최상위 수준의 zip 파일에 숨겨진 파일(예: `__MACOSX`)을 추가할 수 있습니다.
- Windows에서는 파일을 zip 내의 폴더에 자동으로 추가하여 의도치 않게 중첩된 폴더가 생성될 수 있습니다.

다음 절차에서는 Amazon Linux 2 또는 Info-ZIP `zip` 및 `zipinfo` 유틸리티 배포를 제공하는 유사한 OS에서 파일과 상호 작용하는 경우를 가정합니다. 예상치 못한 동작을 방지하려면 이러한 도구를 사용하는 것이 좋습니다.

Ray에서 사용할 Python 파일을 패키징하려면

1. 패키지 이름으로 임시 디렉터리를 생성하고 작업 디렉터리가 상위 디렉터리인지 확인합니다. 다음 명령으로 이를 수행할 수 있습니다.

```
cd parent_directory
mkdir temp_dir
```

2. 파일을 임시 디렉터리에 복사한 다음 디렉터리 구조를 확인합니다. 이 디렉터리의 콘텐츠는 Python 모듈로 직접 액세스됩니다. 이러한 권한 부여는 다음 명령을 사용하여 가능합니다.

```
ls -AR temp_dir
# my_file_1.py
# my_file_2.py
```

3. zip을 사용하여 임시 폴더를 압축합니다. 다음 명령으로 이를 수행할 수 있습니다.

```
zip -r zip_file.zip temp_dir
```

4. 파일이 제대로 패키징되었는지 확인합니다. 이제 작업 디렉터리에서 *zip_file.zip*을 찾을 수 있습니다. 다음 명령을 사용하여 검사할 수 있습니다.

```
zipinfo -1 zip_file.zip
# temp_dir/
# temp_dir/my_file_1.py
# temp_dir/my_file_2.py
```

Ray에서 사용할 Python 패키지를 다시 패키징하려면

1. 패키지 이름으로 임시 디렉터리를 생성하고 작업 디렉터리가 상위 디렉터리인지 확인합니다. 다음 명령으로 이를 수행할 수 있습니다.

```
cd parent_directory
mkdir temp_dir
```

2. 패키지의 압축을 풀고 콘텐츠를 임시 디렉터리에 복사합니다. 모듈의 콘텐츠만 남기고 이전 패키징 표준과 관련된 파일을 제거합니다. 다음 명령을 사용하여 파일 구조가 올바른지 확인합니다.

```
ls -AR temp_dir
# my_module
# my_module/__init__.py
```

```
# my_module/my_file_1.py
# my_module/my_submodule/__init__.py
# my_module/my_submodule/my_file_2.py
# my_module/my_submodule/my_file_3.py
```

3. zip을 사용하여 임시 폴더를 압축합니다. 다음 명령으로 이를 수행할 수 있습니다.

```
zip -r zip_file.zip temp_dir
```

4. 파일이 제대로 패키징되었는지 확인합니다. 이제 작업 디렉터리에서 *zip_file.zip*을 찾을 수 있습니다. 다음 명령을 사용하여 검사할 수 있습니다.

```
zipinfo -1 zip_file.zip
# temp_dir/my_module/
# temp_dir/my_module/__init__.py
# temp_dir/my_module/my_file_1.py
# temp_dir/my_module/my_submodule/
# temp_dir/my_module/my_submodule/__init__.py
# temp_dir/my_module/my_submodule/my_file_2.py
# temp_dir/my_module/my_submodule/my_file_3.py
```

Ray 작업의 데이터에 연결

AWS Glue Ray 작업에서는 데이터를 빠르게 통합하도록 설계된 다양한 Python 패키지를 사용할 수 있습니다. 사용자 환경의 혼란이 가중되지 않도록 최소한의 종속성 세트가 제공됩니다. 기본적으로 포함되는 항목에 대한 자세한 정보는 [the section called “Ray 작업과 함께 제공되는 모듈”](#) 섹션을 참조하세요.

Note

AWS Glue 추출, 변환, 적재(ETL)에서는 DynamicFrame 추상화를 제공하여 데이터 세트의 행 간 스키마 차이를 해결하는 ETL 워크플로를 간소화합니다. AWS Glue ETL은 추가 기능(예: 작업 북마크 및 입력 파일 그룹화)을 제공합니다. 현재 Ray 작업에서는 해당 기능을 제공하지 않습니다.

AWS Glue for Spark는 특정 데이터 형식, 소스 및 싱크와의 연결을 직접 지원합니다. Ray에서 AWS SDK for pandas 및 최신 서드 파티 라이브러리가 실질적으로 이러한 요구 사항을 충족합니다. 사용 가능한 기능을 알아보려면 해당 라이브러리를 참조해야 합니다.

AWS Glue for Ray 및 Amazon VPC의 통합은 현재 사용할 수 없습니다. Amazon VPC 내 리소스는 퍼블릭 경로로만 액세스할 수 있습니다. Amazon VPC에서 AWS Glue를 사용하는 방법에 대한 자세한 내용은 [the section called “VPC 엔드포인트\(AWS PrivateLink\)”](#) 섹션을 참조하세요.

Ray의 데이터 작업을 위한 공통 라이브러리

Ray Data - Ray Data는 일반적인 데이터 형식, 소스 및 싱크를 처리하는 메서드를 제공합니다. Ray Data에서 지원되는 형식 및 소스에 대한 자세한 내용은 Ray Data 설명서의 [Input/Output](#)을 참조하세요. Ray Data는 데이터 세트를 처리하기 위한 범용 라이브러리가 아닌, 독자적인 라이브러리입니다.

Ray는 Ray Data가 작업에 가장 적합한 솔루션일 수 있는 사용 사례에 대한 특정 지침을 제공합니다. 자세한 내용은 Ray 설명서에서 [Ray 사용 사례](#)를 참조하세요.

AWS SDK for pandas (awswrangler) – AWS SDK for pandas는 변환을 통해 pandas DataFrames로 데이터를 관리할 때 AWS 서비스에서 읽고 쓸 수 있는 테스트된 깔끔한 솔루션을 제공하는 AWS 제품입니다. AWS SDK for pandas에서 지원되는 형식 및 소스에 대한 자세한 내용은 AWS SDK for pandas 설명서의 [API Reference](#)를 참조하세요.

AWS SDK for pandas를 사용하여 데이터를 읽고 쓰는 방법에 대한 예제는 AWS SDK for pandas 설명서의 [Quick Start](#)를 참조하세요. AWS SDK for pandas에서는 데이터에 대한 변환을 제공하지 않습니다. 소스에서의 읽기 및 쓰기만 지원합니다.

Modin – Modin은 일반적인 pandas 연산을 배포 가능한 방식으로 구현하는 Python 라이브러리입니다. Modin에 대한 자세한 내용은 [Modin 설명서](#)를 참조하세요. Modin 자체는 소스에서의 읽기 및 쓰기를 지원하지 않습니다. 공통 변환의 분산 구현을 제공합니다. Modin은 AWS SDK for pandas에서 지원됩니다.

Ray 환경에서 Modin 및 AWS SDK for pandas를 함께 실행하면 공통 ETL 작업을 수행하여 더 뛰어난 결과를 얻을 수 있습니다. AWS SDK for pandas와 함께 Modin을 사용하는 방법에 대한 자세한 내용은 AWS SDK for pandas 설명서의 [At scale](#)을 참조하세요.

기타 프레임워크 - Ray가 지원하는 프레임워크에 대한 자세한 내용은 Ray 설명서의 [Ray 에코시스템](#)을 참조하세요. AWS Glue for Ray에서는 다른 프레임워크에 대한 지원을 제공하지 않습니다.

데이터 카탈로그를 통해 데이터에 연결

Ray 작업과 함께 데이터 카탈로그를 통해 데이터를 관리하는 기능은 AWS SDK for pandas에서 지원됩니다. 자세한 내용은 AWS SDK for pandas 웹사이트의 [Glue 카탈로그](#)를 참조하세요.

AWS SDK와 함께 이 서비스 사용

AWS 소프트웨어 개발 키트 (SDK) 는 널리 사용되는 여러 프로그래밍 언어에 사용할 수 있습니다. 각 SDK는 개발자가 선호하는 언어로 애플리케이션을 쉽게 구축할 수 있도록 하는 API, 코드 예시 및 설명서를 제공합니다.

SDK 설명서	코드 예시
AWS SDK for C++	AWS SDK for C++ 코드 예제
AWS CLI	AWS CLI 코드 예제
AWS SDK for Go	AWS SDK for Go 코드 예제
AWS SDK for Java	AWS SDK for Java 코드 예제
AWS SDK for JavaScript	AWS SDK for JavaScript 코드 예제
AWS SDK for Kotlin	AWS SDK for Kotlin 코드 예제
AWS SDK for .NET	AWS SDK for .NET 코드 예제
AWS SDK for PHP	AWS SDK for PHP 코드 예제
AWS Tools for PowerShell	PowerShell 코드 예제를 위한 도구
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 코드 예제
AWS SDK for Ruby	AWS SDK for Ruby 코드 예제
AWS SDK for Rust	AWS SDK for Rust 코드 예제
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP 코드 예제
AWS SDK for Swift	AWS SDK for Swift 코드 예제

이 서비스 관련 예시는 [AWS GlueAWS SDK를 사용한 API 코드 예제](#)를 참조하세요.

i 예제 사용 가능 여부

필요한 예제를 찾을 수 없습니까? 이 페이지 하단의 피드백 제공 링크를 사용하여 코드 예시를 요청하세요.

AWS Glue API

이 섹션에서는 AWS Glue SDK 및 도구에서 사용하는 데이터 유형과 프리미티브에 대해 설명합니다. 외부에서 AWS Glue 프로그래밍 방식으로 상호 작용하는 세 가지 일반적인 방법이 있으며 AWS Management Console, 각 방법마다 고유한 설명서가 있습니다.

- 언어 SDK 라이브러리를 사용하면 일반적인 프로그래밍 언어의 AWS 리소스에 액세스할 수 있습니다. [AWS를 기반으로 빌드할 도구](#)에서 자세한 정보를 찾으십시오.
- 명령줄에서 AWS 리소스에 액세스할 수 있습니다. AWS CLI [AWS CLI 명령 참조](#)에서 자세한 정보를 찾으십시오.
- AWS CloudFormation 일관되게 함께 프로비저닝할 AWS 리소스 세트를 정의할 수 있습니다. 자세한 내용은 [AWS Glue 리소스 AWS CloudFormation 유형](#) 참조에서 확인할 수 있습니다.

이 섹션에서는 이러한 SDK 및 도구와는 별도로 공유 기본 요소에 대해 설명합니다. 도구는 [AWS Glue 웹 API 참조](#)를 사용하여 AWS 통신합니다.

목차

- [AWS Glue의 보안 API](#)
 - [데이터 타입](#)
 - [DataCatalogEncryptionSettings 구조](#)
 - [EncryptionAtRest 구조](#)
 - [ConnectionPasswordEncryption 구조](#)
 - [EncryptionConfiguration 구조](#)
 - [S3Encryption 구조](#)
 - [CloudWatchEncryption 구조](#)
 - [JobBookmarksEncryption 구조](#)
 - [SecurityConfiguration 구조](#)
 - [GluePolicy 구조](#)
 - [운영](#)
 - [GetDataCatalogEncryptionSettings 작업\(Python: \[get_data_catalog_encryption_settings\]\(#\)\)](#)
 - [PutDataCatalogEncryptionSettings 작업\(Python: \[put_data_catalog_encryption_settings\]\(#\)\)](#)
 - [PutResourcePolicy 작업\(Python: \[put_resource_policy\]\(#\)\)](#)
 - [GetResourcePolicy 작업\(Python: \[get_resource_policy\]\(#\)\)](#)

- [DeleteResourcePolicy](#) 작업(Python: delete_resource_policy)
- [CreateSecurityConfiguration](#) 작업(Python: create_security_configuration)
- [DeleteSecurityConfiguration](#) 작업(Python: delete_security_configuration)
- [GetSecurityConfiguration](#) 작업(Python: get_security_configuration)
- [GetSecurityConfigurations](#) 작업(Python: get_security_configurations)
- [GetResourcePolicies](#) 작업(Python: get_resource_policies)
- [카탈로그 API](#)
 - [데이터베이스 API](#)
 - [데이터 타입](#)
 - [데이터베이스 구조](#)
 - [DatabaseInput](#) 구조
 - [PrincipalPermissions](#) 구조
 - [DataLakePrincipal](#) 구조
 - [DatabaseIdentifier](#) 구조
 - [FederatedDatabase](#) 구조
 - [운영](#)
 - [CreateDatabase](#) 작업(Python: create_database)
 - [UpdateDatabase](#) 작업(Python: update_database)
 - [DeleteDatabase](#) 작업(Python: delete_database)
 - [GetDatabase](#) 작업(Python: get_database)
 - [GetDatabases](#) 작업(Python: get_database)
 - [표 API](#)
 - [데이터 타입](#)
 - [테이블 구조](#)
 - [TableInput](#) 구조
 - [FederatedTable](#) 구조
 - [열 구조](#)
 - [StorageDescriptor](#) 구조
 - [SchemaReference](#) 구조
 - [SerDeInfo](#) 구조

- [Order 구조](#)
- [SkewedInfo 구조체](#)
- [TableVersion 구조](#)
- [TableError 구조](#)
- [TableVersionError 구조](#)
- [SortCriterion 구조](#)
- [TableIdentifier 구조](#)
- [KeySchemaElement 구조](#)
- [PartitionIndex 구조](#)
- [PartitionIndexDescriptor 구조](#)
- [BackfillError 구조](#)
- [IcebergInput 구조](#)
- [OpenTableFormatInput 구조](#)
- [ViewDefinition 구조](#)
- [ViewDefinitionInput 구조](#)
- [ViewRepresentation 구조](#)
- [ViewRepresentationInput 구조](#)
- [운영](#)
- [CreateTable 액션 \(파이썬: create_table\)](#)
- [UpdateTable 액션 \(Python: 업데이트_테이블\)](#)
- [DeleteTable 액션 \(파이썬: 삭제_테이블\)](#)
- [BatchDeleteTable 액션 \(Python: 배치_삭제_테이블\)](#)
- [GetTable 액션 \(파이썬: get_table\)](#)
- [GetTables 액션 \(파이썬: get_tables\)](#)
- [GetTableVersion 액션 \(파이썬: get_table_version\)](#)
- [GetTableVersions 액션 \(파이썬: get_table_version\)](#)
- [DeleteTableVersion 액션 \(Python: 테이블_버전_삭제\)](#)
- [BatchDeleteTableVersion 액션 \(Python: 배치_삭제_테이블_버전\)](#)
- [SearchTables 액션 \(Python: 검색_테이블\)](#)
- [GetPartitionIndexes 액션 \(Python: get_partition_index\)](#)

- [CreatePartitionIndex](#) 액션 (Python: 파티션_인덱스_생성하기)
- [DeletePartitionIndex](#) 액션 (Python: 파티션_인덱스_삭제)
- [GetColumnStatisticsForTable](#) 액션 (Python: get_column_statistics_for_table)
- [UpdateColumnStatisticsForTable](#) 액션 (Python: 업데이트_컬럼_통계_포터_테이블)
- [DeleteColumnStatisticsForTable](#) 액션 (Python: 테이블_열_통계_삭제_)
- [파티션 API](#)
 - [데이터 타입](#)
 - [파티션 구조](#)
 - [PartitionInput](#) 구조
 - [PartitionSpecWithSharedStorageDescriptor](#) 구조
 - [PartitionListComposingSpec](#) 구조
 - [PartitionSpecProxy](#) 구조
 - [PartitionValueList](#) 구조
 - [세그먼트 구조](#)
 - [PartitionError](#) 구조
 - [BatchUpdatePartitionFailureEntry](#) 구조
 - [BatchUpdatePartitionRequestEntry](#) 구조
 - [StorageDescriptor](#) 구조
 - [SchemaReference](#) 구조
 - [SerDelInfo](#) 구조
 - [SkewedInfo](#) 구조
 - [운영](#)
 - [CreatePartition](#) 작업(Python: create_partition)
 - [BatchCreatePartition](#) 작업(Python: batch_create_partition)
 - [UpdatePartition](#) 작업(Python: update_partition)
 - [DeletePartition](#) 작업(Python: delete_partition)
 - [BatchDeletePartition](#) 작업(Python: batch_delete_partition)
 - [GetPartition](#) 작업(Python: get_partition)
 - [GetPartitions](#) 작업(Python: get_partitions)
 - [BatchGetPartition](#) 작업(Python: batch_get_partition)

- [BatchUpdatePartition](#) 작업(Python: `batch_update_partition`)
- [GetColumnStatisticsForPartition](#) 작업(Python: `get_column_statistics_for_partition`)
- [UpdateColumnStatisticsForPartition](#) 작업(Python: `update_column_statistics_for_partition`)
- [DeleteColumnStatisticsForPartition](#) 작업(Python: `delete_column_statistics_for_partition`)
- [연결 API](#)
 - [데이터 타입](#)
 - [연결 구조](#)
 - [ConnectionInput](#) 구조
 - [PhysicalConnectionRequirements](#) 구조
 - [GetConnectionsFilter](#) 구조
 - [운영](#)
 - [CreateConnection](#) 작업(Python: `create_connection`)
 - [DeleteConnection](#) 작업(Python: `delete_connection`)
 - [GetConnection](#) 작업(Python: `get_connection`)
 - [GetConnections](#) 작업(Python: `get_connections`)
 - [UpdateConnection](#) 작업(Python: `update_connection`)
 - [BatchDeleteConnection](#) 작업(Python: `batch_delete_connection`)
 - [인증 구성](#)
 - [AuthenticationConfiguration](#) 구조
 - [AuthenticationConfigurationInput](#) 구조
 - [OAuth2Properties](#) 구조
 - [OAuth2PropertiesInput](#) 구조
 - [OAuth2ClientApplication](#) 구조
 - [AuthorizationCodeProperties](#) 구조
- [사용자 정의 함수 API](#)
 - [데이터 타입](#)
 - [UserDefinedFunction](#) 구조
 - [UserDefinedFunctionInput](#) 구조
 - [운영](#)
 - [CreateUserDefinedFunction](#) 작업(Python: `create_user_defined_function`)

- [UpdateUserDefinedFunction 작업 \(Python: update_user_defined_function\)](#)
- [DeleteUserDefinedFunction 작업 \(Python: delete_user_defined_function\)](#)
- [GetUserDefinedFunction 작업 \(Python: get_user_defined_function\)](#)
- [GetUserDefinedFunctions 작업 \(Python: get_user_defined_functions\)](#)
- [Athena 카탈로그를 AWS Glue로 가져오기](#)
 - [데이터 타입](#)
 - [CatalogImportStatus 구조](#)
 - [운영](#)
 - [ImportCatalogToGlue 작업\(Python: import_catalog_to_glue\)](#)
 - [GetCatalogImportStatus 작업\(Python: get_catalog_import_status\)](#)
- [테이블 옵티마이저 API](#)
 - [데이터 타입](#)
 - [TableOptimizer 구조](#)
 - [TableOptimizerConfiguration 구조](#)
 - [TableOptimizerRun 구조](#)
 - [RunMetrics 구조](#)
 - [BatchGetTableOptimizerEntry 구조](#)
 - [BatchTableOptimizer 구조](#)
 - [BatchGetTableOptimizerError 구조](#)
 - [운영](#)
 - [GetTableOptimizer 액션 \(파이썬: get_table_optimizer\)](#)
 - [BatchGetTableOptimizer 액션 \(Python: batch_get_table_optimizer\)](#)
 - [ListTableOptimizerRuns 액션 \(Python: 목록_테이블_옵티마이저_실행\)](#)
 - [CreateTableOptimizer 액션 \(Python: 생성_테이블_옵티마이저\)](#)
 - [DeleteTableOptimizer 액션 \(Python: 삭제_테이블_옵티마이저\)](#)
 - [UpdateTableOptimizer 액션 \(Python: 업데이트_테이블_옵티마이저\)](#)
- [크롤러 및 분류자 API](#)
 - [분류자 API](#)
 - [데이터 타입](#)
 - [분류자 구조](#)

- [GrokClassifier 구조](#)
- [XMLClassifier 구조](#)
- [JsonClassifier 구조](#)
- [CsvClassifier 구조](#)
- [CreateGrokClassifierRequest 구조](#)
- [UpdateGrokClassifierRequest 구조](#)
- [CreateXMLClassifierRequest 구조](#)
- [UpdateXMLClassifierRequest 구조](#)
- [CreateJsonClassifierRequest 구조](#)
- [UpdateJsonClassifierRequest 구조](#)
- [CreateCsvClassifierRequest 구조](#)
- [UpdateCsvClassifierRequest 구조](#)
- [운영](#)
- [CreateClassifier 작업\(Python: create_classifier\)](#)
- [DeleteClassifier 작업\(Python: delete_classifier\)](#)
- [GetClassifier 작업\(Python: get_classifier\)](#)
- [GetClassifiers 작업\(Python: get_classifiers\)](#)
- [UpdateClassifier 작업\(Python: update_classifier\)](#)
- [크롤러 API](#)
 - [데이터 타입](#)
 - [크롤러 구조](#)
 - [일정 구조](#)
 - [CrawlerTargets 구조](#)
 - [S3Target 구조](#)
 - [S3 DeltaCatalogTarget 구조](#)
 - [S3 DeltaDirectTarget 구조](#)
 - [JdbcTarget 구조](#)
 - [MongoDBTarget 구조](#)
 - [DynamoDBTarget 구조](#)
 - [DeltaTarget 구조](#)

- [IcebergTarget 구조](#)
- [HudiTarget 구조](#)
- [CatalogTarget 구조](#)
- [CrawlerMetrics 구조](#)
- [CrawlerHistory 구조](#)
- [CrawlsFilter 구조](#)
- [SchemaChangePolicy 구조](#)
- [LastCrawlInfo 구조](#)
- [RecrawlPolicy 구조](#)
- [LineageConfiguration 구조](#)
- [LakeFormationConfiguration 구조](#)
- [운영](#)
- [CreateCrawler 액션 \(Python: create_crawler\)](#)
- [DeleteCrawler 액션 \(Python: 삭제_크롤러\)](#)
- [GetCrawler 액션 \(파이썬: get_crawler\)](#)
- [GetCrawlers 액션 \(파이썬: get_crawlers\)](#)
- [GetCrawlerMetrics 액션 \(Python: get_crawler_metrics\)](#)
- [UpdateCrawler 액션 \(Python: 업데이트_크롤러\)](#)
- [StartCrawler 액션 \(Python: start_crawler\)](#)
- [StopCrawler 액션 \(Python: stop_crawler\)](#)
- [BatchGetCrawlers 액션 \(파이썬: batch_get_crawlers\)](#)
- [ListCrawlers 액션 \(Python: 리스트_크롤러\)](#)
- [ListCrawls 액션 \(Python: 리스트_크롤링\)](#)
- [열 통계의 API](#)
 - [데이터 타입](#)
 - [ColumnStatisticsTaskRun 구조](#)
 - [ColumnStatisticsTaskRunningException 구조](#)
 - [ColumnStatisticsTaskNotRunningException 구조](#)
 - [ColumnStatisticsTaskStoppingException 구조](#)
- [운영](#)

- [StartColumnStatisticsTaskRun 작업 \(Python: start_column_statistics_task_run\)](#)
- [GetColumnStatisticsTaskRun 작업 \(Python: get_column_statistics_task_run\)](#)
- [GetColumnStatisticsTaskRuns 작업 \(Python: get_column_statistics_task_runs\)](#)
- [ListColumnStatisticsTaskRuns 작업 \(Python: list_column_statistics_task_runs\)](#)
- [StopColumnStatisticsTaskRun 작업 \(Python: stop_column_statistics_task_run\)](#)
- [크롤러 스케줄러 API](#)
 - [데이터 타입](#)
 - [일정 구조](#)
 - [운영](#)
 - [UpdateCrawlerSchedule 작업\(Python: start_crawler_schedule\)](#)
 - [StartCrawlerSchedule 작업\(Python: start_crawler_schedule\)](#)
 - [StopCrawlerSchedule 작업\(Python: stop_crawler_schedule\)](#)
- [ETL 스크립트 API 자동 생성](#)
 - [데이터 타입](#)
 - [CodeGenNode 구조](#)
 - [CodeGenNodeArg 구조](#)
 - [CodeGenEdge 구조](#)
 - [위치 구조](#)
 - [CatalogEntry 구조](#)
 - [MappingEntry 구조](#)
 - [운영](#)
 - [CreateScript 작업\(Python: create_script\)](#)
 - [GetDataflowGraph 작업\(Python: get_dataflow_graph\)](#)
 - [GetMapping 작업\(Python: get_mapping\)](#)
 - [GetPlan 작업\(Python: get_plan\)](#)
- [시각적 작업 API](#)
 - [데이터 타입](#)
 - [CodeGenConfigurationNode 구조](#)
 - [JDBC 구조 ConnectorOptions](#)
 - [StreamingDataPreviewOptions 구조체](#)

- [AthenaConnectorSource 구조](#)
- [JDBC 구조 ConnectorSource](#)
- [SparkConnectorSource 구조](#)
- [CatalogSource 구조](#)
- [MySQL 구조 CatalogSource](#)
- [PostgreSQL 구조 CatalogSource](#)
- [오라클 SQL 구조 CatalogSource](#)
- [마이크로소프트/SQL 구조 ServerCatalogSource](#)
- [CatalogKinesisSource 구조](#)
- [DirectKinesisSource 구조](#)
- [KinesisStreamingSourceOptions 구조](#)
- [CatalogKafkaSource 구조](#)
- [DirectKafkaSource 구조](#)
- [KafkaStreamingSourceOptions 구조](#)
- [RedshiftSource 구조](#)
- [AmazonRedshiftSource 구조](#)
- [AmazonRedshiftNodeData 구조](#)
- [AmazonRedshiftAdvancedOption 구조](#)
- [옵션 구조](#)
- [S3 CatalogSource 구조](#)
- [S3 SourceAdditionalOptions 구조](#)
- [S3 CsvSource 구조](#)
- [DirectJDBCSource 구조](#)
- [S3 DirectSourceAdditionalOptions 구조](#)
- [S3 JsonSource 구조](#)
- [S3 구조 ParquetSource](#)
- [S3 구조 DeltaSource](#)
- [S3 CatalogDeltaSource 구조](#)
- [CatalogDeltaSource 구조](#)
- [S3 HudiSource 구조](#)

- [S3 구조 CatalogHudiSource](#)
- [CatalogHudiSource 구조](#)
- [DynamoDB 구조 CatalogSource](#)
- [RelationalCatalogSource 구조](#)
- [JDBC 구조 ConnectorTarget](#)
- [SparkConnectorTarget 구조](#)
- [BasicCatalogTarget 구조](#)
- [MySQL 구조 CatalogTarget](#)
- [PostgreSQL 구조 CatalogTarget](#)
- [오라클 SQL 구조 CatalogTarget](#)
- [마이크로소프트 SQL 구조 ServerCatalogTarget](#)
- [RedshiftTarget 구조](#)
- [AmazonRedshiftTarget 구조](#)
- [UpsertRedshiftTargetOptions 구조](#)
- [S3 CatalogTarget 구조](#)
- [S3 GlueParquetTarget 구조](#)
- [CatalogSchemaChangePolicy 구조](#)
- [S3 DirectTarget 구조](#)
- [S3 HudiCatalogTarget 구조](#)
- [S3 구조 HudiDirectTarget](#)
- [S3 구조 DeltaCatalogTarget](#)
- [S3 DeltaDirectTarget 구조](#)
- [DirectSchemaChangePolicy 구조](#)
- [ApplyMapping 구조](#)
- [Mapping 구조](#)
- [SelectFields 구조](#)
- [DropFields 구조](#)
- [RenameField 구조](#)
- [Spigot 구조](#)
- [조인 구조](#)

- [JoinColumn 구조](#)
- [SplitFields 구조](#)
- [SelectFromCollection 구조](#)
- [FillMissingValues 구조](#)
- [Filter 구조](#)
- [FilterExpression 구조](#)
- [FilterValue 구조](#)
- [CustomCode 구조](#)
- [SparkSQL 구조](#)
- [SqlAlias 구조](#)
- [DropNullFields 구조](#)
- [NullCheckBoxList 구조](#)
- [NullValueField 구조](#)
- [데이터 형식 구조](#)
- [병합 구조](#)
- [결합 구조](#)
- [PIIDetection 구조](#)
- [집계 구조](#)
- [DropDuplicates 구조](#)
- [GovernedCatalogTarget 구조](#)
- [GovernedCatalogSource 구조](#)
- [AggregateOperation 구조](#)
- [GlueSchema 구조](#)
- [GlueStudioSchemaColumn 구조](#)
- [GlueStudioColumn 구조](#)
- [DynamicTransform 구조](#)
- [TransformConfigParameter 구조](#)
- [EvaluateDataQuality 구조](#)
- [DQ 구조 ResultsPublishingOptions](#)
- [DQ 구조 StopJobOnFailureOptions](#)

- [EvaluateDataQualityMultiFrame 구조](#)
- [레시피 구조](#)
- [RecipeReference 구조](#)
- [SnowflakeNodeData 구조](#)
- [SnowflakeSource 구조](#)
- [SnowflakeTarget 구조](#)
- [ConnectorDataSource 구조](#)
- [ConnectorDataTarget 구조](#)
- [작업 API](#)
 - [작업](#)
 - [데이터 타입](#)
 - [작업 구조](#)
 - [ExecutionProperty 구조](#)
 - [NotificationProperty 구조](#)
 - [JobCommand 구조](#)
 - [ConnectionsList 구조](#)
 - [JobUpdate 구조](#)
 - [SourceControlDetails 구조](#)
 - [운영](#)
 - [CreateJob 액션 \(Python: create_job\)](#)
 - [UpdateJob 액션 \(Python: 업데이트_작업\)](#)
 - [GetJob 액션 \(파이썬: get_job\)](#)
 - [GetJobs 액션 \(파이썬: get_jobs\)](#)
 - [DeleteJob 액션 \(Python: 삭제_작업\)](#)
 - [ListJobs 액션 \(파이썬: 리스트_잡\)](#)
 - [BatchGetJobs 액션 \(파이썬: batch_get_jobs\)](#)
 - [작업 실행](#)
 - [데이터 타입](#)
 - [JobRun 구조](#)
 - [이전 구조](#)

- [JobBookmarkEntry 구조](#)
- [BatchStopJobRunSuccessfulSubmission 구조](#)
- [BatchStopJobRunError 구조](#)
- [NotificationProperty 구조](#)
- [운영](#)
- [StartJobRun 액션 \(Python: 시작_작업_실행\)](#)
- [BatchStopJobRun 액션 \(Python: 배치_중지_작업_실행\)](#)
- [GetJobRun 액션 \(파이썬: get_job_run\)](#)
- [GetJobRuns 액션 \(파이썬: get_job_runs\)](#)
- [GetJobBookmark 액션 \(파이썬: get_job_bookmark\)](#)
- [GetJobBookmarks 액션 \(Python: get_job_bookmark\)](#)
- [ResetJobBookmark 액션 \(Python: reset_job_bookmark\)](#)
- [트리거](#)
 - [데이터 타입](#)
 - [트리거 구조](#)
 - [TriggerUpdate 구조](#)
 - [조건자 구조](#)
 - [조건 구조](#)
 - [작업 구조](#)
 - [EventBatchingCondition 구조](#)
 - [운영](#)
 - [CreateTrigger 작업\(Python: create_trigger\)](#)
 - [StartTrigger 작업\(Python: start_trigger\)](#)
 - [GetTrigger 작업\(Python: get_trigger\)](#)
 - [GetTrigger 작업\(Python: get_triggers\)](#)
 - [UpdateTrigger 작업\(Python: update_trigger\)](#)
 - [StopTrigger 작업\(Python: stop_trigger\)](#)
 - [DeleteTrigger 작업\(Python: delete_trigger\)](#)
 - [ListTriggers 작업\(Python: list_triggers\)](#)
 - [BatchGetTriggers 작업\(Python: batch_get_triggers\)](#)

- [대화형 세션 API](#)
 - [데이터 타입](#)
 - [세션 구조](#)
 - [SessionCommand 구조](#)
 - [명령문 구조](#)
 - [StatementOutput 구조체](#)
 - [StatementOutputData 구조](#)
 - [ConnectionsList 구조](#)
 - [운영](#)
 - [CreateSession 액션 \(Python: create_session\)](#)
 - [StopSession 액션 \(파이썬: 스탑_세션\)](#)
 - [DeleteSession 액션 \(Python: 삭제_세션\)](#)
 - [GetSession 액션 \(파이썬: get_session\)](#)
 - [ListSessions 액션 \(Python: 리스트_세션\)](#)
 - [RunStatement 액션 \(Python: run_statement\)](#)
 - [CancelStatement 액션 \(Python: 취소_명령문\)](#)
 - [GetStatement 액션 \(파이썬: get_statement\)](#)
 - [ListStatements 액션 \(Python: 리스트_스테이트먼트\)](#)
- [개발 엔드포인트 API](#)
 - [데이터 타입](#)
 - [DevEndpoint 구조](#)
 - [DevEndpointCustomLibraries 구조](#)
 - [운영](#)
 - [CreateDevEndpoint 작업\(Python: create_dev_endpoint\)](#)
 - [UpdateDevEndpoint 작업\(Python: update_dev_endpoint\)](#)
 - [DeleteDevEndpoint 작업\(Python: delete_dev_endpoint\)](#)
 - [GetDevEndpoint 작업\(Python: get_dev_endpoint\)](#)
 - [GetDevEndpoints 작업\(Python: get_dev_endpoints\)](#)
 - [BatchGetDevEndpoints 작업\(Python: batch_get_dev_endpoints\)](#)
 - [ListDevEndpoints 작업\(Python: list_dev_endpoints\)](#)

- [Schema Registry](#)
 - [데이터 타입](#)
 - [RegistryId 구조](#)
 - [RegistryListItem 구조](#)
 - [MetadataInfo 구조](#)
 - [OtherMetadataValueListItem 구조](#)
 - [SchemaListItem 구조](#)
 - [SchemaVersionListItem 구조](#)
 - [MetadataKeyValuePair 구조](#)
 - [SchemaVersionErrorItem 구조](#)
 - [ErrorDetails 구조](#)
 - [SchemaVersionNumber 구조](#)
 - [Schemald 구조](#)
 - [운영](#)
 - [CreateRegistry](#) 액션 (Python: 레지스트리 만들기)
 - [CreateSchema](#) 액션 (Python: 생성_스키마)
 - [GetSchema](#) 액션 (파이썬: get_schema)
 - [ListSchemaVersions](#) 액션 (Python: 목록_스키마_버전)
 - [GetSchemaVersion](#) 액션 (Python: get_schema_version)
 - [GetSchemaVersionsDiff](#) 액션 (Python: get_schema_versions_diff)
 - [ListRegistries](#) 액션 (Python: 리스트_레지스트리)
 - [ListSchemas](#) 액션 (Python: 리스트_스키마)
 - [RegisterSchemaVersion](#) 액션 (Python: 레지스터_스키마_버전)
 - [UpdateSchema](#) 액션 (Python: 업데이트_스키마)
 - [CheckSchemaVersionValidity](#) 조치 (Python: 검사_스키마_버전_유효성 검사)
 - [UpdateRegistry](#) 액션 (Python: 업데이트_레지스트리)
 - [GetSchemaByDefinition](#) 액션 (Python: get_schema_by_definition)
 - [GetRegistry](#) 액션 (파이썬: get_registry)
 - [PutSchemaVersionMetadata](#) 액션 (Python: put_schema_version_메타데이터)
 - [QuerySchemaVersionMetadata](#) 액션 (Python: 쿼리_스키마_버전_메타데이터)

- [RemoveSchemaVersionMetadata](#) 액션 (Python: [스키마_버전_메타데이터 제거](#))
- [DeleteRegistry](#) 액션 (Python: [레지스트리 삭제](#))
- [DeleteSchema](#) 액션 (Python: [삭제_스키마](#))
- [DeleteSchemaVersions](#) 액션 (Python: [스키마_버전 삭제](#))
- [워크플로](#)
 - [데이터 타입](#)
 - [JobNodeDetails](#) 구조
 - [CrawlerNodeDetails](#) 구조
 - [TriggerNodeDetails](#) 구조
 - [크롤 구조](#)
 - [노드 구조](#)
 - [엡지 구조](#)
 - [워크플루 구조](#)
 - [WorkflowGraph](#) 구조
 - [WorkflowRun](#) 구조
 - [WorkflowRunStatistics](#) 구조
 - [StartingEventBatchCondition](#) 구조
 - [블루프린트 구조](#)
 - [BlueprintDetails](#) 구조
 - [LastActiveDefinition](#) 구조
 - [BlueprintRun](#) 구조
 - [운영](#)
 - [CreateWorkflow](#) 작업(Python: [create_workflow](#))
 - [UpdateWorkflow](#) 작업(Python: [update_workflow](#))
 - [DeleteWorkflow](#) 작업(Python: [delete_workflow](#))
 - [GetWorkflow](#) 작업(Python: [get_workflow](#))
 - [ListWorkflows](#) 작업(Python: [list_workflows](#))
 - [BatchGetWorkflows](#) 작업(Python: [batch_get_workflows](#))
 - [GetWorkflowRun](#) 작업(Python: [get_workflow_run](#))
 - [GetWorkflowRuns](#) 작업(Python: [get_workflow_runs](#))

- [GetWorkflowRunProperties](#) 작업(Python: [get_workflow_run_properties](#))
- [PutWorkflowRunProperties](#) 작업(Python: [put_workflow_run_properties](#))
- [CreateBlueprint](#) 작업(Python: [create_blueprint](#))
- [UpdateBlueprint](#) 작업(Python: [update_blueprint](#))
- [DeleteBlueprint](#) 작업(Python: [delete_blueprint](#))
- [ListBlueprints](#) 작업(Python: [list_blueprints](#))
- [BatchGetBlueprints](#) 작업(Python: [batch_get_blueprints](#))
- [StartBlueprintRun](#) 작업(Python: [start_blueprint_run](#))
- [GetBlueprintRun](#) 작업(Python: [get_blueprint_run](#))
- [GetBlueprintRuns](#) 작업(Python: [get_blueprint_runs](#))
- [StartWorkflowRun](#) 작업(Python: [start_workflow_run](#))
- [StopWorkflowRun](#) 작업(Python: [stop_workflow_run](#))
- [ResumeWorkflowRun](#) 작업(Python: [resume_workflow_run](#))
- [사용 프로파일](#)
 - [데이터 타입](#)
 - [ProfileConfiguration](#) 구조
 - [ConfigurationObject](#) 구조
 - [UsageProfileDefinition](#) 구조
 - [운영](#)
 - [CreateUsageProfile](#) 액션 (Python: [사용_프로파일_생성하기](#))
 - [GetUsageProfile](#) 액션 (Python: [get_usage_profile](#))
 - [UpdateUsageProfile](#) 액션 (Python: [업데이트_사용_프로파일](#))
 - [DeleteUsageProfile](#) 액션 (Python: [사용_프로파일_삭제](#))
 - [ListUsageProfiles](#) 액션 (Python: [목록_사용_프로파일](#))
- [기계 학습 API](#)
 - [데이터 타입](#)
 - [TransformParameters](#) 구조
 - [EvaluationMetrics](#) 구조
 - [ML Transform](#) 구조
- [FindMatchesParameters](#) 구조

- [FindMatchesMetrics 구조](#)
- [ConfusionMatrix 구조](#)
- [GlueTable 구조](#)
- [TaskRun 구조](#)
- [TransformFilterCriteria 구조](#)
- [TransformSortCriteria 구조](#)
- [TaskRunFilterCriteria 구조](#)
- [TaskRunSortCriteria 구조](#)
- [TaskRunProperties 구조](#)
- [FindMatchesTaskRunProperties 구조](#)
- [ImportLabelsTaskRunProperties 구조](#)
- [ExportLabelsTaskRunProperties 구조](#)
- [LabelingSetGenerationTaskRunProperties 구조](#)
- [SchemaColumn 구조](#)
- [TransformEncryption 구조](#)
- [MLUserDataEncryption 구조](#)
- [ColumnImportance 구조](#)
- [운영](#)
- [CreateMLTransform 작업\(Python: create_ml_transform\)](#)
- [UpdateMLTransform 작업\(Python: update_ml_transform\)](#)
- [DeleteMLTransform 작업\(Python: delete_ml_transform\)](#)
- [GetMLTransform 작업\(Python: get_ml_transform\)](#)
- [GetMLTransforms 작업\(Python: get_ml_transforms\)](#)
- [ListMLTransforms 작업\(Python: list_ml_transforms\)](#)
- [StartMLEvaluationTaskRun 작업\(Python: start_ml_evaluation_task_run\)](#)
- [StartMLLabelingSetGenerationTaskRun 작업\(Python: start_ml_labeling_set_generation_task_run\)](#)
- [GetMLTaskRun 작업\(Python: get_ml_task_run\)](#)
- [GetMLTaskRuns 작업\(Python: get_ml_task_runs\)](#)
- [CancelMLTaskRun 작업\(Python: cancel_ml_task_run\)](#)

- [StartExportLabelsTaskRun 작업\(Python: start_export_labels_task_run\)](#)
- [StartImportLabelsTaskRun 작업\(Python: start_import_labels_task_run\)](#)
- [데이터 품질 API](#)
 - [데이터 타입](#)
 - [DataSource 구조](#)
 - [DataQualityRulesetListDetails 구조](#)
 - [DataQualityTargetTable 구조](#)
 - [DataQualityRulesetEvaluationRunDescription 구조](#)
 - [DataQualityRulesetEvaluationRunFilter 구조](#)
 - [DataQualityEvaluationRunAdditionalRunOptions 구조](#)
 - [DataQualityRuleRecommendationRunDescription 구조](#)
 - [DataQualityRuleRecommendationRunFilter 구조](#)
 - [DataQualityResult 구조](#)
 - [DataQualityAnalyzerResult 구조](#)
 - [DataQualityObservation 구조](#)
 - [MetricBasedObservation 구조](#)
 - [DataQualityMetricValues 구조](#)
 - [DataQualityRuleResult 구조](#)
 - [DataQualityResultDescription 구조](#)
 - [DataQualityResultFilterCriteria 구조](#)
 - [DataQualityRulesetFilterCriteria 구조](#)
- [운영](#)
 - [StartDataQualityRulesetEvaluationRun 액션 \(Python: 시작_데이터_품질_규칙 세트_평가_실행\)](#)
 - [CancelDataQualityRulesetEvaluationRun 액션 \(Python: 취소_데이터_품질_규칙 세트_평가_실행\)](#)
 - [GetDataQualityRulesetEvaluationRun 액션 \(Python: get_data_quality_ruleset_valuation_run\)](#)
 - [ListDataQualityRulesetEvaluationRuns 액션 \(Python: 목록_데이터_품질_규칙 세트_평가_실행\)](#)
 - [StartDataQualityRuleRecommendationRun 액션 \(Python: 시작_데이터_품질_규칙_권장_실행\)](#)
 - [CancelDataQualityRuleRecommendationRun 조치 \(Python: 취소_데이터_품질_규칙_권장_실행\)](#)
 - [GetDataQualityRuleRecommendationRun 액션 \(Python: get_data_품질_규칙_권장_실행\)](#)
 - [ListDataQualityRuleRecommendationRuns 액션 \(Python: 목록_데이터_품질_규칙_권장_실행\)](#)

- [GetDataQualityResult](#) 액션 (Python: [데이터_품질_결과_가져오기](#))
- [BatchGetDataQualityResult](#) 액션 (Python: [배치_가져오기_데이터_품질_결과](#))
- [ListDataQualityResults](#) 액션 (Python: [목록_데이터_품질_결과](#))
- [CreateDataQualityRuleset](#) 액션 (Python: [생성_데이터_품질_규칙_세트](#))
- [DeleteDataQualityRuleset](#) 액션 (Python: [데이터_품질_규칙_세트_삭제](#))
- [GetDataQualityRuleset](#) 액션 (Python: [get_data_quality_ruleset](#))
- [ListDataQualityRulesets](#) 액션 (Python: [목록_데이터_품질_규칙_세트](#))
- [UpdateDataQualityRuleset](#) 액션 (Python: [업데이트_데이터_품질_규칙_세트](#))
- [민감한 데이터 감지 API](#)
 - [데이터 타입](#)
 - [CustomEntityType](#) 구조
 - [운영](#)
 - [CreateCustomEntityType](#) 작업(Python: [create_custom_entity_type](#))
 - [DeleteCustomEntityType](#) 작업(Python: [delete_custom_entity_type](#))
 - [GetCustomEntityType](#) 작업(Python: [get_custom_entity_type](#))
 - [BatchGetCustomEntityTypes](#) 작업(Python: [batch_get_custom_entity_types](#))
 - [ListCustomEntityTypes](#) 작업(Python: [list_custom_entity_types](#))
- [AWS Glue에서 API 태그 지정](#)
 - [데이터 타입](#)
 - [태그 구조](#)
 - [운영](#)
 - [TagResource](#) 작업(Python: [tag_resource](#))
 - [UntagResource](#) 작업(Python: [untag_resource](#))
 - [GetTags](#) 작업(Python: [get_tags](#))
- [공통 데이터 형식](#)
 - [태그 구조](#)
 - [DecimalNumber](#) 구조
 - [ErrorDetail](#) 구조
 - [PropertyPredicate](#) 구조
 - [ResourceUri](#) 구조

- [ColumnStatistics 구조](#)
- [ColumnStatisticsError 구조](#)
- [ColumnError 구조](#)
- [ColumnStatisticsData 구조](#)
- [BooleanColumnStatisticsData 구조](#)
- [DateColumnStatisticsData 구조](#)
- [DecimalColumnStatisticsData 구조](#)
- [DoubleColumnStatisticsData 구조](#)
- [LongColumnStatisticsData 구조](#)
- [StringColumnStatisticsData 구조](#)
- [BinaryColumnStatisticsData 구조](#)
- [문자열 패턴](#)
- [예외](#)
 - [AccessDeniedException 구조](#)
 - [AlreadyExistsException 구조](#)
 - [ConcurrentModificationException 구조](#)
 - [ConcurrentRunsExceededException 구조](#)
 - [CrawlerNotRunningException 구조](#)
 - [CrawlerRunningException 구조](#)
 - [CrawlerStoppingException 구조](#)
 - [EntityNotFoundException 구조](#)
 - [FederationSourceException 구조](#)
 - [FederationSourceRetryableException 구조](#)
 - [GlueEncryptionException 구조](#)
 - [IdempotentParameterMismatchException 구조](#)
 - [IllegalWorkflowStateException 구조](#)
 - [InternalServiceException 구조](#)
 - [InvalidExecutionEngineException 구조](#)
 - [InvalidInputException 구조](#)
 - [InvalidStateException 구조](#)

- [InvalidTaskStatusTransitionException 구조](#)
- [JobDefinitionErrorException 구조](#)
- [JobRunInTerminalStateException 구조](#)
- [JobRunInvalidStateTransitionException 구조](#)
- [JobRunNotInTerminalStateException 구조](#)
- [LateRunnerException 구조](#)
- [NoScheduleException 구조](#)
- [OperationTimeoutException 구조](#)
- [ResourceNotReadyException 구조](#)
- [ResourceNumberLimitExceededException 구조](#)
- [SchedulerNotRunningException 구조](#)
- [SchedulerRunningException 구조](#)
- [SchedulerTransitioningException 구조](#)
- [UnrecognizedRunnerException 구조](#)
- [ValidationException 구조](#)
- [VersionMismatchException 구조](#)

AWS Glue의 보안 API

보안 API는 보안 데이터 유형 및 AWS Glue의 보안과 관련된 API에 대해 설명합니다.

데이터 타입

- [DataCatalogEncryptionSettings 구조](#)
- [EncryptionAtRest 구조](#)
- [ConnectionPasswordEncryption 구조](#)
- [EncryptionConfiguration 구조](#)
- [S3Encryption 구조](#)
- [CloudWatchEncryption 구조](#)
- [JobBookmarksEncryption 구조](#)
- [SecurityConfiguration 구조](#)
- [GluePolicy 구조](#)

DataCatalogEncryptionSettings 구조

데이터 카탈로그 보안을 유지 관리하기 위한 구성 정보가 포함되어 있습니다.

필드

- EncryptionAtRest – [EncryptionAtRest](#) 객체입니다.

데이터 카탈로그에 대한 저장 데이터 암호화 구성을 지정합니다.

- ConnectionPasswordEncryption – [ConnectionPasswordEncryption](#) 객체입니다.

연결 암호 보호가 활성화된 경우 데이터 카탈로그는 고객 제공 키를 사용하여 CreateConnection 또는 UpdateConnection의 일부로 암호를 암호화하고 이를 연결 속성의 ENCRYPTED_PASSWORD 필드에 저장합니다. 카탈로그 암호화를 활성화하거나 암호 암호화만 활성화할 수 있습니다.

EncryptionAtRest 구조

데이터 카탈로그에 대한 저장 데이터 암호화 구성을 지정합니다.

필드

- CatalogEncryptionMode – 필수: UTF-8 문자열입니다(유효한 값: DISABLED | SSE-KMS="SSEKMS" | SSE-KMS-WITH-SERVICE-ROLE="SSEKMSWITHSERVICEROLE").

데이터 카탈로그 데이터를 암호화하기 위한 저장 데이터 암호화 모드입니다.

- SseAwsKmsKeyId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

저장된 데이터 암호화에 사용할 AWS KMS 키의 ID입니다.

- CatalogEncryptionServiceRole – [Custom string pattern #24](#)과(와) 일치하는 UTF-8 문자열입니다.

AWS Glue가 호출자를 대신하여 데이터 카탈로그 객체를 암호화 및 해독하는 역할을 맡습니다.

ConnectionPasswordEncryption 구조

데이터 카탈로그가 CreateConnection 또는 UpdateConnection의 일부로 암호를 암호화하고 이를 연결 속성의 ENCRYPTED_PASSWORD 필드에 저장하기 위해 사용하는 데이터 구조입니다. 카탈로그 암호화를 활성화하거나 암호 암호화만 활성화할 수 있습니다.

암호를 포함하는 CreationConnection 요청이 도착하면 Data Catalog는 먼저 AWS KMS 키를 사용하여 암호를 암호화합니다. 그런 다음 카탈로그 암호화도 활성화되어 있으면 전체 연결 객체를 다시 암호화합니다.

이 암호화를 사용하려면 보안 요구 사항에 따라 암호 키에 대한 액세스를 사용하거나 제한하도록 AWS KMS 키 권한을 설정해야 합니다. 예를 들어 관리자만 암호 키에 대한 암호화 해제 권한을 갖도록 하고자 할 수 있습니다.

필드

- ReturnConnectionPasswordEncrypted – 필수(Required): 부울.

ReturnConnectionPasswordEncrypted 플래그가 "true"로 설정된 경우, 암호는 GetConnection 및 GetConnections의 응답에서 암호화된 상태로 유지됩니다. 이 암호화는 카탈로그 암호화와는 독립적으로 적용됩니다.

- AwsKmsKeyId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

연결 암호를 암호화하는 데 사용되는 AWS KMS 키입니다.

연결 암호 보호가 사용되는 경우 CreateConnection 및 UpdateConnection의 호출자는 Data Catalog에 암호를 저장하기 전에 암호화하려면 지정된 AWS KMS 키에 대한 최소한 kms:Encrypt 권한이 필요합니다.

보안 요구 사항에 따라 암호 키에 대한 액세스를 허용하거나 제한하도록 암호화 해제 권한을 설정할 수 있습니다.

EncryptionConfiguration 구조

암호화 구성을 지정합니다.

필드

- S3Encryption – [S3Encryption](#) 객체의 배열입니다.

Amazon Simple Storage Service(Amazon S3) 데이터에 대한 암호화 구성.

- CloudWatchEncryption – [CloudWatchEncryption](#) 객체입니다.

Amazon CloudWatch에 대한 암호화 구성.

- JobBookmarksEncryption – [JobBookmarksEncryption](#) 객체입니다.

작업 북마크에 대한 암호화 구성.

S3Encryption 구조

Amazon Simple Storage Service(Amazon S3) 데이터를 암호화하는 방법을 지정합니다.

필드

- S3EncryptionMode – UTF-8 문자열입니다(유효한 값: DISABLED | SSE-KMS="SSEKMS" | SSE-S3="SSES3").

Amazon S3 데이터에 사용할 암호화 모드입니다.

- KmsKeyArn – [Custom string pattern #25](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터를 암호화하는 데 사용되는 KMS 키의 Amazon 리소스 이름(ARN).

CloudWatchEncryption 구조

Amazon CloudWatch 데이터를 암호화하는 방법을 지정합니다.

필드

- CloudWatchEncryptionMode – UTF-8 문자열입니다(유효한 값: DISABLED | SSE-KMS="SSEKMS").

CloudWatch 데이터에 사용할 암호화 모드입니다.

- KmsKeyArn – [Custom string pattern #25](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터를 암호화하는 데 사용되는 KMS 키의 Amazon 리소스 이름(ARN).

JobBookmarksEncryption 구조

작업 북마크 데이터를 암호화하는 방법을 지정합니다.

필드

- JobBookmarksEncryptionMode – UTF-8 문자열입니다(유효한 값: DISABLED | CSE-KMS="CSEKMS").

작업 북마크 데이터에 사용할 암호화 모드입니다.

- KmsKeyArn – [Custom string pattern #25](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터를 암호화하는 데 사용되는 KMS 키의 Amazon 리소스 이름(ARN).

SecurityConfiguration 구조

보안 구성을 지정합니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

보안 구성의 이름입니다.

- CreatedTimeStamp – 타임스탬프입니다.

이 보안 구성이 생성된 시간입니다.

- EncryptionConfiguration – [EncryptionConfiguration](#) 객체입니다.

이 보안 구성과 관련된 암호화 구성입니다.

GluePolicy 구조

리소스 정책을 반환하기 위한 구조입니다.

필드

- PolicyInJson – UTF-8 문자열입니다(최소 2바이트).

요청된 정책 문서(JSON 형식)가 포함되어 있습니다.

- PolicyHash – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 정책과 연관된 해시 값을 포함하고 있습니다.

- CreateTime – 타임스탬프입니다.

정책이 생성된 날짜와 시간입니다.

- UpdateTime – 타임스탬프입니다.

정책이 업데이트된 날짜와 시간입니다.

운영

- [GetDataCatalogEncryptionSettings](#) 작업(Python: `get_data_catalog_encryption_settings`)
- [PutDataCatalogEncryptionSettings](#) 작업(Python: `put_data_catalog_encryption_settings`)
- [PutResourcePolicy](#) 작업(Python: `put_resource_policy`)
- [GetResourcePolicy](#) 작업(Python: `get_resource_policy`)
- [DeleteResourcePolicy](#) 작업(Python: `delete_resource_policy`)
- [CreateSecurityConfiguration](#) 작업(Python: `create_security_configuration`)
- [DeleteSecurityConfiguration](#) 작업(Python: `delete_security_configuration`)
- [GetSecurityConfiguration](#) 작업(Python: `get_security_configuration`)
- [GetSecurityConfigurations](#) 작업(Python: `get_security_configurations`)
- [GetResourcePolicies](#) 작업(Python: `get_resource_policies`)

GetDataCatalogEncryptionSettings 작업(Python: `get_data_catalog_encryption_settings`)

지정된 카탈로그에 대한 보안 구성을 검색합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

보안 구성을 검색할 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

응답

- `DataCatalogEncryptionSettings` – [DataCatalogEncryptionSettings](#) 객체입니다.

요청된 보안 구성입니다.

Errors

- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

PutDataCatalogEncryptionSettings 작업(Python: `put_data_catalog_encryption_settings`)

지정된 카탈로그에 대한 보안 구성을 설정합니다. 구성이 설정되면 지정된 암호화가 이후의 모든 카탈로그 쓰기에 적용됩니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

보안 구성을 설정할 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `DataCatalogEncryptionSettings` – 필수(Required): [DataCatalogEncryptionSettings](#) 객체입니다.

설정할 보안 구성입니다.

응답

- 무응답 파라미터.

Errors

- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

PutResourcePolicy 작업(Python: put_resource_policy)

액세스 제어를 위한 데이터 카탈로그 리소스 정책을 설정합니다.

요청

- PolicyInJson – 필수: UTF-8 문자열입니다(최소 2바이트).

설정할 정책 문서(JSON 형식)가 포함되어 있습니다.

- ResourceArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

사용하지 않습니다. 내부용입니다.

- PolicyHashCondition – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이전 정책이 PutResourcePolicy를 사용하여 설정된 경우 반환되는 해시 값입니다. 정책의 동시 수정을 방지하기 위해 사용됩니다. 설정된 이전 정책이 없는 경우에는 이 파라미터를 사용하지 마십시오.

- PolicyExistsCondition – UTF-8 문자열입니다(유효 값: MUST_EXIST | NOT_EXIST | NONE).

MUST_EXIST의 값은 정책을 업데이트하는 데 사용됩니다. NOT_EXIST의 값은 새 정책을 생성하는 데 사용됩니다. NONE의 값 또는 null 값이 사용되는 경우 호출은 정책의 존재 여부에 종속되지 않습니다.

- EnableHybrid – UTF-8 문자열입니다(유효한 값: TRUE | FALSE).

'TRUE'이면 Data Catalog 리소스에 대한 교차 계정 액세스 권한을 부여하기 위해 두 가지 방법을 모두 사용하고 있음을 나타냅니다.

- PutResourcePolicy로 리소스 정책을 직접 업데이트하여
- AWS Management Console에서 권한 부여 명령을 사용합니다.

이미 관리 콘솔을 사용하여 교차 계정 액세스 권한을 부여한 경우 'TRUE'로 설정해야 합니다. 그렇지 않으면 호출이 실패합니다. 기본값은 'FALSE'입니다.

응답

- PolicyHash – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

방금 설정한 정책의 해시입니다. 이 정책을 덮어쓰거나 업데이트하는 후속 호출에 반드시 포함되어 있어야 합니다.

Errors

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ConditionCheckFailureException

GetResourcePolicy 작업(Python: get_resource_policy)

지정된 리소스 정책을 검색합니다.

요청

- ResourceArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

리소스 정책을 검색할 AWS Glue 리소스의 ARN입니다. 제공하지 않으면 Data Catalog 리소스 정책이 반환됩니다. GetResourcePolicies를 사용하여 기존 리소스 정책을 모두 봅니다. 자세한 내용은 [AWS Glue 리소스 ARN 지정](#)을 참조하세요.

응답

- PolicyInJson – UTF-8 문자열입니다(최소 2바이트).

요청된 정책 문서(JSON 형식)가 포함되어 있습니다.

- PolicyHash – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 정책과 연관된 해시 값을 포함하고 있습니다.

- CreateTime – 타임스탬프입니다.

정책이 생성된 날짜와 시간입니다.

- UpdateTime – 타임스탬프입니다.

정책이 업데이트된 날짜와 시간입니다.

Errors

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

DeleteResourcePolicy 작업(Python: delete_resource_policy)

지정된 정책을 삭제합니다.

요청

- PolicyHashCondition – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 정책이 설정되었을 때 반환된 해시 값입니다.

- ResourceArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

삭제할 리소스 정책에 대한 AWS Glue 리소스의 ARN입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ConditionCheckFailureException

CreateSecurityConfiguration 작업(Python: create_security_configuration)

새로운 보안 구성을 생성합니다. 보안 구성은 AWS Glue에서 사용할 수 있는 보안 속성 세트입니다. 보안 구성을 사용하여 미사용 데이터를 암호화할 수 있습니다. AWS Glue에서 보안 구성 사용에 대한 자세한 내용은 [크롤러, 작업 및 개발 엔드포인트로 기록된 데이터 암호화](#)를 참조하세요.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

새 보안 구성의 이름입니다.

- EncryptionConfiguration – 필수(Required): [EncryptionConfiguration](#) 객체입니다.

새 보안 구성을 위한 암호화 구성입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

새 보안 구성에 할당된 이름입니다.

- CreatedTimestamp – 타임스탬프입니다.

새 보안 구성이 생성된 시간입니다.

Errors

- AlreadyExistsException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException

DeleteSecurityConfiguration 작업(Python: delete_security_configuration)

지정된 보안 구성을 삭제합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 보안 구성의 이름입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetSecurityConfiguration 작업(Python: `get_security_configuration`)

지정된 보안 구성을 검색합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

검색할 보안 구성의 이름입니다.

응답

- SecurityConfiguration – [SecurityConfiguration](#) 객체입니다.

요청된 보안 구성입니다.

Errors

- EntityNotFoundException

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetSecurityConfigurations 작업(Python: `get_security_configurations`)

모든 보안 구성 목록을 검색합니다.

요청

- `MaxResults` – 1~1,000의 숫자(정수)입니다.
반환할 최대 결과 수입니다.
- `NextToken` – UTF-8 문자열입니다.
이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- `SecurityConfigurations` – [SecurityConfiguration](#) 객체의 배열입니다.
보안 구성의 목록입니다.
- `NextToken` – UTF-8 문자열입니다.
반환할 보안 구성이 더 있는 경우 연속 토큰입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetResourcePolicies 작업(Python: `get_resource_policies`)

교차 계정 권한 부여 중에 AWS Resource Access Manager가 개별 리소스에 설정한 리소스 정책을 검색합니다. 또한 Data Catalog 리소스 정책을 검색합니다.

Data Catalog 설정에서 메타데이터 암호화를 사용하고 AWS KMS 키에 대한 권한이 없으면 작업에서 Data Catalog 리소스 정책을 반환할 수 없습니다.

요청

- `NextToken` – UTF-8 문자열입니다.
이것이 지속적인 요청이라면 지속적인 토큰입니다.
- `MaxResults` – 1~1,000의 숫자(정수)입니다.
반환할 목록의 최대 크기.

응답

- `GetResourcePoliciesResponseList` – [GluePolicy](#) 객체의 배열입니다.
개별 리소스 정책 및 계정 수준 리소스 정책 목록입니다.
- `NextToken` – UTF-8 문자열입니다.
반환된 목록이 사용 가능한 마지막 리소스 정책을 포함하지 경우의 연속 토큰입니다.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

카탈로그 API

카탈로그 API는 AWS Glue에서의 카탈로그 작업과 관련된 API 및 데이터 유형에 대해 설명합니다.

주제

- [데이터베이스 API](#)
- [표 API](#)
- [파티션 API](#)

- [연결 API](#)
- [사용자 정의 함수 API](#)
- [Athena 카탈로그를 AWS Glue로 가져오기](#)

데이터베이스 API

데이터베이스 API는 데이터베이스 데이터 형식을 설명하며, 데이터베이스를 생성, 삭제, 위치 지정, 업데이트 및 목록화하는 API를 포함합니다.

데이터 타입

- [데이터베이스 구조](#)
- [DatabaseInput 구조](#)
- [PrincipalPermissions 구조](#)
- [DataLakePrincipal 구조](#)
- [DatabaseIdentifier 구조](#)
- [FederatedDatabase 구조](#)

데이터베이스 구조

Database 객체는 Hive 메타스토어 혹은 RDBMS에 있는 테이블의 논리적 그룹을 나타냅니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터베이스의 이름입니다. 저장될 때 소문자로 저장되어 Hive 호환성을 유지합니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

데이터베이스에 대한 설명입니다.

- LocationUri – [URI address multi-line string pattern](#)과(와) 일치하는 1~1,024바이트 길이의 URI(Uniform Resource Identifier)입니다.

데이터베이스의 위치(예: HDFS 경로)입니다.

- **Parameters** – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 키-값 쌍은 데이터베이스의 파라미터와 속성을 정의합니다.

- **CreateTime** – 타임스탬프입니다.

메타 데이터베이스가 카탈로그에 생성된 시간입니다.

- **CreateTableDefaultPermissions** – [PrincipalPermissions](#) 객체의 배열입니다.

보안 주체에 대한 테이블에서 기본 권한 세트를 생성합니다. AWS Lake Formation에서 사용됩니다. AWS Glue 작업의 일반적인 과정에서는 사용되지 않습니다.

- **TargetDatabase** – [DatabaseIdentifier](#) 객체입니다.

리소스 링크에 대한 대상 데이터베이스를 설명하는 DatabaseIdentifier 구조입니다.

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

데이터베이스가 있는 데이터 카탈로그의 ID입니다.

- **FederatedDatabase** – [FederatedDatabase](#) 객체입니다.

AWS Glue Data Catalog 외부의 엔티티를 참조하는 FederatedDatabase 구조입니다.

DatabaseInput 구조

데이터베이스를 생성 및 업데이트할 때 사용되는 구조입니다.

필드

- **Name** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터베이스의 이름입니다. 저장될 때 소문자로 저장되어 Hive 호환성을 유지합니다.

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

데이터베이스에 대한 설명입니다.

- `LocationUri` – [URI address multi-line string pattern](#)과(와) 일치하는 1~1,024바이트 길이의 URI(Uniform Resource Identifier)입니다.

데이터베이스의 위치(예: HDFS 경로)입니다.

- `Parameters` – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 키-값 쌍은 데이터베이스의 파라미터와 속성을 정의합니다.

이러한 키-값 쌍은 데이터베이스의 파라미터와 속성을 정의합니다.

- `CreateTableDefaultPermissions` – [PrincipalPermissions](#) 객체의 배열입니다.

보안 주체에 대한 테이블에서 기본 권한 세트를 생성합니다. AWS Lake Formation에서 사용됩니다. AWS Glue 작업의 일반적인 과정에서는 사용되지 않습니다.

- `TargetDatabase` – [DatabaseIdentifier](#) 객체입니다.

리소스 링크에 대한 대상 데이터베이스를 설명하는 `DatabaseIdentifier` 구조입니다.

- `FederatedDatabase` – [FederatedDatabase](#) 객체입니다.

AWS Glue Data Catalog 외부의 엔터티를 참조하는 `FederatedDatabase` 구조입니다.

PrincipalPermissions 구조

보안 주체에 부여된 권한입니다.

필드

- `Principal` – [DataLakePrincipal](#) 객체입니다.

권한을 부여받는 보안 주체입니다.

- `Permissions` – UTF-8 문자열의 배열입니다.

보안 주체에 부여되는 권한입니다.

DataLakePrincipal 구조

AWS Lake Formation 보안 주체입니다.

필드

- `DataLakePrincipalIdentifier` – 1~255바이트 길이의 UTF-8 문자열입니다.
AWS Lake Formation 보안 주체의 식별자입니다.

Databaselfentifier 구조

리소스 링크에 대한 대상 데이터베이스를 설명하는 구조입니다.

필드

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.
데이터베이스가 있는 데이터 카탈로그의 ID입니다.
- `DatabaseName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
카탈로그 데이터베이스의 이름입니다.
- `Region` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
대상 데이터베이스의 리전입니다.

FederatedDatabase 구조

AWS Glue Data Catalog 외부의 엔터티를 가리키는 데이터베이스입니다.

필드

- `Identifier` – [Single-line string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.
페더레이션된 데이터베이스의 고유 식별자입니다.
- `ConnectionName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

외부 메타스토어에 대한 연결 이름입니다.

운영

- [CreateDatabase 작업\(Python: create_database\)](#)
- [UpdateDatabase 작업\(Python: update_database\)](#)
- [DeleteDatabase 작업\(Python: delete_database\)](#)
- [GetDatabase 작업\(Python: get_database\)](#)
- [GetDatabases 작업\(Python: get_database\)](#)

CreateDatabase 작업(Python: create_database)

데이터 카탈로그에 새로운 데이터베이스를 생성합니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

데이터베이스를 생성할 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseInput – 필수(Required): [DatabaseInput](#) 객체입니다.

데이터베이스의 메타데이터입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

데이터베이스에 할당하는 태그입니다.

응답

- 무응답 파라미터.

Errors

- `InvalidInputException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `ConcurrentModificationException`
- `FederatedResourceAlreadyExistsException`

UpdateDatabase 작업(Python: `update_database`)

데이터 카탈로그에서 기존 데이터베이스를 업데이트합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

메타데이터 데이터베이스가 있는 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `Name` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

카탈로그에서 업데이트할 데이터베이스의 이름입니다. 저장될 때 소문자로 저장되어 Hive 호환성을 유지합니다.

- `DatabaseInput` – 필수(Required): [DatabaseInput](#) 객체입니다.

`DatabaseInput` 객체는 카탈로그에서 메타데이터 데이터베이스의 새로운 정의를 지정합니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException

DeleteDatabase 작업(Python: delete_database)

데이터 카탈로그에서 지정된 데이터베이스를 제거합니다.

Note

이 작업을 완료한 후에는 테이블(및 테이블에 속하는 모든 테이블 버전과 파티션)은 물론 삭제된 데이터베이스에 있는 사용자 정의 함수에 더 이상 액세스하지 못합니다. AWS Glue는 이러한 "분리된" 리소스를 서비스 재량에 따라 적시에 비동기로 삭제합니다. 관련된 모든 리소스가 즉시 삭제되도록 DeleteDatabase 호출 전에 DeleteTableVersion 또는 BatchDeleteTableVersion, DeletePartition 또는 BatchDeletePartition, DeleteUserDefinedFunction 및 DeleteTable 또는 BatchDeleteTable를 사용하여 데이터베이스에 속하는 모든 리소스를 삭제하십시오.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

데이터베이스가 있는 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 데이터베이스의 이름입니다. 반드시 모두 소문자로 저장하여 Hive 호환성을 유지하도록 합니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

GetDatabase 작업(Python: get_database)

지정된 데이터베이스의 정의를 검색합니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

데이터베이스가 있는 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

검색할 데이터베이스의 이름입니다. 반드시 모두 소문자로 저장하여 Hive 호환성을 유지하도록 합니다.

응답

- Database – [데이터베이스](#) 객체입니다.

데이터 카탈로그에서 지정된 데이터베이스의 정의입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `FederationSourceException`

GetDatabases 작업(Python: `get_database`)

주어진 데이터 카탈로그에 정의된 모든 데이터베이스를 검색합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

`Databases`를 검색할 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

- `MaxResults` – 1~100의 숫자(정수)입니다.

한 번의 응답으로 반환될 최대 데이터베이스 수입니다.

- `ResourceShareType` – UTF-8 문자열입니다(유효한 값: `FOREIGN` | `ALL` | `FEDERATED`).

계정과 공유된 데이터베이스를 나열하도록 지정할 수 있습니다. 허용 가능한 값은 `FEDERATED`, `FOREIGN` 또는 `ALL`입니다.

- `FEDERATED`로 설정하면 계정과 공유된 페더레이션된 데이터베이스(외부 엔터티라고 함)가 나열됩니다.
- `FOREIGN`으로 설정하면 계정과 공유된 데이터베이스가 나열됩니다.
- `ALL`로 설정하면 계정과 공유된 데이터베이스와 로컬 계정의 데이터베이스가 나열됩니다.

응답

- DatabaseList – 필수(Required): [데이터베이스](#) 객체의 배열입니다.

지정된 카탈로그의 Database 목록입니다.

- NextToken – UTF-8 문자열입니다.

목록의 현재 세그먼트가 마지막이 아니면 반환된 토큰 목록에 페이지를 매기는 지속적인 토큰은 반환됩니다.

Errors

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

표 API

테이블 API는 테이블과 관련된 데이터 형식 및 작업에 대해 설명합니다.

데이터 타입

- [테이블 구조](#)
- [TableInput 구조](#)
- [FederatedTable 구조](#)
- [열 구조](#)
- [StorageDescriptor 구조](#)
- [SchemaReference 구조](#)
- [SerDelInfo 구조](#)
- [Order 구조](#)
- [SkewedInfo 구조체](#)
- [TableVersion 구조](#)
- [TableError 구조](#)

- [TableVersionError 구조](#)
- [SortCriterion 구조](#)
- [TableIdentifier 구조](#)
- [KeySchemaElement 구조](#)
- [PartitionIndex 구조](#)
- [PartitionIndexDescriptor 구조](#)
- [BackfillError 구조](#)
- [IcebergInput 구조](#)
- [OpenTableFormatInput 구조](#)
- [ViewDefinition 구조](#)
- [ViewDefinitionInput 구조](#)
- [ViewRepresentation 구조](#)
- [ViewRepresentationInput 구조](#)

테이블 구조

행과 열에 조직된 관련 데이터 모음을 보여줍니다.

필드

- Name – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 이름. 반드시 모두 소문자로 저장하여 Hive 호환성을 유지하도록 합니다.

- DatabaseName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 메타데이터가 있는 데이터베이스의 이름입니다. 반드시 모두 소문자로 저장하여 Hive 호환성을 유지하도록 합니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

테이블에 대한 설명입니다.

- Owner – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 소유자입니다.

- `CreateTime` – 타임스탬프입니다.

테이블 정의가 데이터 카탈로그에 생성된 시간입니다.

- `UpdateTime` – 타임스탬프입니다.

테이블이 업데이트된 마지막 시간입니다.

- `LastAccessTime` – 타임스탬프입니다.

테이블이 액세스된 마지막 시간입니다. 보통 HDFS에서 가져왔고 대다수는 신뢰할 만하지 않습니다.

- `LastAnalyzedTime` – 타임스탬프입니다.

이 테이블에 대한 열 통계가 계산된 마지막 시간입니다.

- `Retention` – None 이하의 숫자(정수)입니다.

이 테이블의 보관 기간입니다.

- `StorageDescriptor` – [StorageDescriptor](#) 객체입니다.

이 테이블의 물리적 스토리지에 대한 정보를 포함하는 스토리지 서술자입니다.

- `PartitionKeys` – [열](#) 객체의 배열입니다.

테이블을 분할할 열의 목록입니다. 초기 유형만 파티션 키로써 지원됩니다.

Amazon Athena에서 사용하는 테이블을 생성할 때 `partitionKeys`를 지정하지 않은 경우, 최소한 값 `partitionKeys`를 빈 목록으로 설정해야 합니다. 예:

```
"PartitionKeys": []
```

- `ViewOriginalText` – 409,600바이트 이하 길이의 UTF-8 문자열입니다.

Apache Hive 호환성을 위해 포함되었습니다. 정상적인 AWS Glue 작업 과정에서는 사용되지 않습니다. 테이블이 a인 경우 `VIRTUAL_VIEW`, base64로 인코딩된 특정 Athena 구성입니다.

- `ViewExpandedText` – 409,600바이트 이하 길이의 UTF-8 문자열입니다.

Apache Hive 호환성을 위해 포함되었습니다. 일반적인 작업 과정에서는 사용되지 않습니다. AWS Glue

- `TableType` – 255바이트 이하 길이의 UTF-8 문자열입니다.

이 테이블의 유형. AWS Glue 다음과 같은 `EXTERNAL_TABLE` 유형으로 테이블을 생성합니다. 와 같은 다른 Athena서비스에서는 추가 테이블 유형을 사용하여 테이블을 생성할 수 있습니다.

AWS Glue 관련 테이블 유형:

EXTERNAL_TABLE

Hive 호환 속성 - Hive 관리형이 아닌 테이블을 나타냅니다.

GOVERNED

에서 사용 AWS Lake Formation. AWS Glue 데이터 카탈로그는 GOVERNED 이해합니다.

- Parameters – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 키-값 쌍은 테이블과 관련된 속성을 정의합니다.

- CreatedBy – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블을 만든 사용자 혹은 개체.

- IsRegisteredWithLakeFormation – 부울입니다.

테이블이 에 등록되었는지 여부를 나타냅니다. AWS Lake Formation

- TargetTable – [TableIdentifier](#) 객체입니다.

리소스 링크에 대한 대상 테이블을 설명하는 TableIdentifier 구조입니다.

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 위치한 데이터 카탈로그의 ID입니다.

- VersionId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 버전의 ID입니다.

- FederatedTable – [FederatedTable](#) 객체입니다.

AWS Glue Data Catalog외부의 엔터티를 참조하는 FederatedTable 구조입니다.

- ViewDefinition – [ViewDefinition](#) 객체입니다.

뷰에 대한 하나 이상의 언어와 쿼리 등 뷰를 정의하는 모든 정보를 포함하는 구조입니다.

- IsMultiDialectView – 부울입니다.

뷰가 여러 다양한 쿼리 엔진의 SQL 언어의 지원하며 해당 엔진에서 읽을 수 있는지 여부를 지정합니다.

TableInput 구조

테이블을 정의하는 데 사용된 구조입니다.

필드

- Name – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 이름. 저장될 때 소문자로 저장되어 Hive 호환성을 유지합니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

테이블에 대한 설명입니다.

- Owner – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 소유자입니다. Apache Hive 호환성을 위해 포함되었습니다. 정상적인 AWS Glue 작업 과정에서는 사용되지 않습니다.

- LastAccessTime – 타임스탬프입니다.

테이블이 액세스된 마지막 시간입니다.

- LastAnalyzedTime – 타임스탬프입니다.

이 테이블에 대한 열 통계가 계산된 마지막 시간입니다.

- Retention – None 이하의 숫자(정수)입니다.

이 테이블의 보관 기간입니다.

- StorageDescriptor – [StorageDescriptor](#) 객체입니다.

이 테이블의 물리적 스토리지에 대한 정보를 포함하는 스토리지 서술자입니다.

- PartitionKeys – [열](#) 객체의 배열입니다.

테이블을 분할할 열의 목록입니다. 초기 유형만 파티션 키로써 지원됩니다.

Amazon Athena에서 사용하는 테이블을 생성할 때 partitionKeys를 지정하지 않은 경우, 최소한 값 partitionKeys를 빈 목록으로 설정해야 합니다. 예:

"PartitionKeys": []

- ViewOriginalText – 409,600바이트 이하 길이의 UTF-8 문자열입니다.

Apache Hive 호환성을 위해 포함되었습니다. 정상적인 AWS Glue 작업 과정에서는 사용되지 않습니다. 테이블이 a인 경우 VIRTUAL_VIEW, base64로 인코딩된 특정 Athena 구성입니다.

- ViewExpandedText – 409,600바이트 이하 길이의 UTF-8 문자열입니다.

Apache Hive 호환성을 위해 포함되었습니다. 일반적인 작업 과정에서는 사용되지 않습니다. AWS Glue

- TableType – 255바이트 이하 길이의 UTF-8 문자열입니다.

이 테이블의 유형. AWS Glue 다음과 같은 EXTERNAL_TABLE 유형으로 테이블을 생성합니다. 와 같은 다른 Athena 서비스에서는 추가 테이블 유형을 사용하여 테이블을 생성할 수 있습니다.

AWS Glue 관련 테이블 유형:

EXTERNAL_TABLE

Hive 호환 속성 - Hive 관리형이 아닌 테이블을 나타냅니다.

GOVERNED

에서 사용 AWS Lake Formation. AWS Glue 데이터 카탈로그는 GOVERNED 이해합니다.

- Parameters – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 키-값 쌍은 테이블과 관련된 속성을 정의합니다.

- TargetTable – [TableIdentifier](#) 객체입니다.

리소스 링크에 대한 대상 테이블을 설명하는 TableIdentifier 구조입니다.

- ViewDefinition – [ViewDefinitionInput](#) 객체입니다.

뷰에 대한 하나 이상의 언어와 쿼리 등 뷰를 정의하는 모든 정보를 포함하는 구조입니다.

FederatedTable 구조

AWS Glue Data Catalog 외부의 엔터티를 가리키는 테이블입니다.

필드

- Identifier – [Single-line string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.
페더레이션된 테이블의 고유 식별자입니다.
- DatabaseIdentifier – [Single-line string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.
페더레이션된 데이터베이스의 고유 식별자입니다.
- ConnectionName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
외부 메타스토어에 대한 연결 이름입니다.

열 구조

Table의 열.

필드

- Name – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
Column의 이름입니다.
- Type – [Single-line string pattern](#)과(와) 일치하는 131,072바이트 이하 길이의 UTF-8 문자열입니다.
Column의 데이터 형식입니다.
- Comment – [Single-line string pattern](#)과(와) 일치하는 255바이트 이하 길이의 주석 문자열입니다.
자유 형식의 텍스트 설명.
- Parameters – 키-값 페어의 맵 배열입니다.
각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.
각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.
이러한 키-값 페어는 열과 관련된 속성을 정의합니다.

StorageDescriptor 구조

테이블 데이터의 물리적 스토리지를 설명합니다.

필드

- Columns – [열](#) 객체의 배열입니다.

테이블의 Columns 목록입니다.

- Location – [URI address multi-line string pattern](#)과(와) 일치하는 2,056바이트 이하 길이의 위치 문자열입니다.

테이블의 물리적 위치입니다. 기본적으로 웨어하우스 위치, 웨어하우스의 데이터베이스 위치, 테이블 이름 순으로 나타납니다.

- AdditionalLocations – UTF-8 문자열의 배열입니다.

Delta 테이블이 위치한 경로를 가리키는 위치 목록입니다.

- InputFormat – [Single-line string pattern](#)과(와) 일치하는 128바이트 이하 길이의 포맷 문자열입니다.

입력 형식: SequenceFileInputFormat(이진), TextInputFormat 또는 사용자 지정 형식입니다.

- OutputFormat – [Single-line string pattern](#)과(와) 일치하는 128바이트 이하 길이의 포맷 문자열입니다.

출력 형식: SequenceFileOutputFormat(이진), IgnoreKeyTextOutputFormat 또는 사용자 지정 형식입니다.

- Compressed – 부울입니다.

테이블의 데이터가 압축되면 True이고 그렇지 않으면 False입니다.

- NumberOfBuckets - 숫자(정수)입니다.

테이블에 차원 열이 포함되어 있는 경우 이 속성을 지정해야 합니다.

- SerdeInfo – [SerDeInfo](#) 객체입니다.

직렬화/역직렬화 () 정보. SerDe

- BucketColumns – UTF-8 문자열의 배열입니다.

테이블의 열, 클러스터링 열 및 버킷 열을 지정하는 그룹화하는 reducer 목록입니다.

- `SortColumns` – [Order](#) 객체의 배열입니다.

테이블에 있는 각 버킷의 정렬 순서를 지정하는 목록입니다.

- `Parameters` – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

키 값 형식의 사용자 제공 속성입니다.

- `SkewedInfo` – [SkewedInfo](#) 객체입니다.

열에 자주 표시되는 값에 대한 정보입니다(편향된 값).

- `StoredAsSubDirectories` – 부울입니다.

테이블 데이터가 하위 디렉터리에 저장되면 `True`이고 그렇지 않으면 `False`입니다.

- `SchemaReference` – [SchemaReference](#) 객체입니다.

스키마 레지스트리에 저장된 스키마를 참조하는 객체입니다. AWS Glue

테이블을 생성할 때 스키마에 대한 빈 열 목록을 전달하고, 대신 스키마 참조를 사용할 수 있습니다.

SchemaReference 구조

스키마 레지스트리에 저장된 스키마를 참조하는 객체입니다. AWS Glue

필드

- `SchemaId` – [Schemald](#) 객체입니다.

스키마 ID 필드를 포함하는 구조입니다. 이것 또는 `SchemaVersionId`가 제공되어야 합니다.

- `SchemaVersionId` – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전에 할당된 고유 ID입니다. 이것 또는 `SchemaId`가 제공되어야 합니다.

- `SchemaVersionNumber` - 1~100,000의 숫자(정수)입니다.

스키마의 버전 번호입니다.

SerDeInfo 구조

추출기 및 로더 역할을 하는 직렬화/역직렬화 프로그램 (SerDe) 에 대한 정보.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

SerDe의 이름.

- SerializationLibrary – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

일반적으로 를 SerDe 구현하는 클래스입니다. 예를 들면, `org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe`입니다.

- Parameters – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 키-값 쌍은 의 초기화 매개변수를 정의합니다. SerDe

Order 구조

분류된 열의 정렬 순서를 지정합니다.

필드

- Column – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

열의 이름입니다.

- SortOrder – 필수(Required): 1 이하의 숫자(정수)입니다.

열이 오름차순(== 1) 또는 내림차순(==0)으로 정렬된 것을 나타냅니다.

SkewedInfo 구조체

테이블에 왜곡된 값을 지정합니다. 왜곡된 값은 매우 높은 빈도를 통해 발생한 값입니다.

필드

- `SkewedColumnNames` – UTF-8 문자열의 배열입니다.
왜곡된 값이 포함된 열의 이름 목록입니다.
- `SkewedColumnValues` – UTF-8 문자열의 배열입니다.
너무 자주 나타나서 왜곡된 것으로 간주되는 값의 목록입니다.
- `SkewedColumnValueLocationMaps` – 키-값 페어의 맵 배열입니다.
각 키는 UTF-8 문자열입니다.
각 값은 UTF-8 문자열입니다.
왜곡된 값을 이 값을 포함하는 열에 매핑.

TableVersion 구조

테이블 버전을 지정합니다.

필드

- `Table` – [표](#) 객체입니다.
문제의 테이블입니다.
- `VersionId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 테이블 버전을 식별하는 ID 값. `VersionId`는 정수의 문자열 표현입니다. 각 버전은 1씩 증가합니다.

TableError 구조

테이블 작업의 오류 기록.

필드

- `TableName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
테이블의 이름 반드시 모두 소문자로 저장하여 Hive 호환성을 유지하도록 합니다.
- `ErrorDetail` – [ErrorDetail](#) 객체입니다.

오류에 대한 세부 정보입니다.

TableVersionError 구조

테이블 버전 작업의 오류 기록.

필드

- **TableName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
문제의 테이블 이름입니다.
- **VersionId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
문제의 버전 ID 값입니다. VersionID는 정수의 문자열 표현입니다. 각 버전은 1씩 증가합니다.
- **ErrorDetail** – [ErrorDetail](#) 객체입니다.
오류에 대한 세부 정보입니다.

SortCriterion 구조

정렬 기준 필드와 정렬 순서를 지정합니다.

필드

- **FieldName** – 1,024바이트 이하 길이의 값 문자열입니다.
정렬할 필드의 이름입니다.
- **Sort** – UTF-8 문자열입니다(유효 값: ASC="ASCENDING" | DESC="DESCENDING").
오름차순 또는 내림차순 정렬.

TableIdentifier 구조

리소스 링크에 대한 대상 테이블을 설명하는 구조입니다.

필드

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 위치한 데이터 카탈로그의 ID입니다.

- DatabaseName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

대상 테이블을 포함하는 카탈로그 데이터베이스의 이름입니다.

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

대상 테이블의 이름입니다.

- Region – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

대상 테이블의 리전입니다.

KeySchemaElement 구조

이름과 유형으로 구성된 파티션 키 페어입니다.

필드

- Name – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 키의 이름입니다.

- Type – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 131,072바이트 이하 길이의 UTF-8 문자열입니다.

파티션 키의 유형입니다.

PartitionIndex 구조

파티션 인덱스의 구조입니다.

필드

- Keys – 필수(Required): UTF-8 문자열의 배열이며 문자열은 1개 이상입니다.

파티션 인덱스의 키입니다.

- IndexName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 인덱스의 이름입니다.

PartitionIndexDescriptor 구조

테이블의 파티션 인덱스에 대한 설명자입니다.

필드

- **IndexName** – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 인덱스의 이름입니다.

- **Keys** – 필수(Required): [KeySchemaElement](#) 객체의 배열로 구조가 1개 이상입니다.

파티션 인덱스에 대한 하나 이상의 키 목록(KeySchemaElement 구조)입니다.

- **IndexStatus** – 필수(Required): UTF-8 문자열입니다(유효 값: CREATING | ACTIVE | DELETING | FAILED).

파티션 인덱스의 상태입니다.

가능한 상태는 다음과 같습니다.

- [생성 중(CREATING)]: 인덱스를 생성 중입니다. 인덱스가 [생성 중(CREATING)] 상태인 경우 인덱스 또는 해당 테이블을 삭제할 수 없습니다.
- [활성(ACTIVE)]: 인덱스 생성에 성공했습니다.
- [실패(FAILED)]: 인덱스 생성에 실패했습니다.
- [삭제 중(DELETING)]: 인덱스 목록에서 인덱스를 삭제합니다.
- **BackfillErrors** – [BackfillError](#) 객체의 배열입니다.

기존 테이블에 대한 파티션 인덱스를 등록할 때 발생할 수 있는 오류 목록입니다.

BackfillError 구조

기존 테이블에 대한 파티션 인덱스를 등록할 때 발생할 수 있는 오류 목록입니다.

이러한 오류는 인덱스 등록이 실패한 이유에 대한 세부 정보를 제공하고 응답에 제한된 수의 파티션을 제공하므로 결함이 있는 파티션을 수정하고 인덱스 등록을 다시 시도할 수 있습니다. 발생할 수 있는 가장 일반적인 오류 집합은 다음과 같이 분류됩니다.

- **EncryptedPartitionError**: 파티션이 암호화됩니다.
- **InvalidPartitionTypeDataError**: 파티션 값이 해당 파티션 열의 데이터 유형과 일치하지 않습니다.

- `MissingPartitionValueError`: 파티션이 암호화됩니다.
- `UnsupportedPartitionCharacterError`: 파티션 값 내의 문자는 지원되지 않습니다. 예: U+0000 , U+0001, U+0002.
- `InternalError`: 다른 오류 코드에 속하지 않는 모든 오류

필드

- `Code` – UTF-8 문자열입니다(유효한 값: `ENCRYPTED_PARTITION_ERROR` | `INTERNAL_ERROR` | `INVALID_PARTITION_TYPE_DATA_ERROR` | `MISSING_PARTITION_VALUE_ERROR` | `UNSUPPORTED_PARTITION_CHARACTER_ERROR`).

기존 테이블에 대한 파티션 인덱스를 등록할 때 발생한 오류에 대한 오류 코드입니다.

- `Partitions` – [PartitionValueList](#) 객체의 배열입니다.

응답의 제한된 수의 파티션 목록입니다.

IcebergInput 구조

카탈로그에서 생성할 Apache Iceberg 메타데이터 테이블을 정의하는 구조입니다.

필드

- `MetadataOperation` – 필수: UTF-8 문자열입니다(유효한 값: `CREATE`).
- 필수 메타데이터 작업입니다. `CREATE`로만 설정할 수 있습니다.
- `Version` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Iceberg 테이블의 테이블 버전입니다. 기본값은 2입니다.

OpenTableFormatInput 구조

오픈 형식 테이블을 나타내는 구조입니다.

필드

- `IcebergInput` – [IcebergInput](#) 객체입니다.

Apache Iceberg 메타데이터 테이블을 정의하는 `IcebergInput` 구조를 지정합니다.

ViewDefinition 구조

표현에 대한 세부 정보를 포함하는 구조입니다.

필드

- IsProtected – 부울입니다.

이 플래그를 true로 설정하면 쿼리 계획 중에 사용자가 제공한 작업을 뷰의 논리적 계획으로 푸시하지 않도록 엔진에 지시할 수 있습니다. 하지만 이 플래그를 설정한다고 해서 엔진이 지시를 따른다는 보장은 없습니다. 제공되는 보장에 대해 알아보려면 엔진 설명서를 참조하세요(있는 경우).

- Definer – [Single-line string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.

SQL에서 뷰의 정의자입니다.

- SubObjects – 10개 이하의 문자열로 구성된 UTF-8 문자열입니다.

Amazon 리소스 이름(ARN) 테이블 목록입니다.

- Representations – [ViewRepresentation](#) 객체의 배열이며 구조는 1~1,000개입니다.

표현 목록입니다.

ViewDefinitionInput 구조

AWS Glue 뷰를 만들거나 업데이트하기 위한 세부 정보가 포함된 구조입니다.

필드

- IsProtected – 부울입니다.

이 플래그를 true로 설정하면 쿼리 계획 중에 사용자가 제공한 작업을 뷰의 논리적 계획으로 푸시하지 않도록 엔진에 지시할 수 있습니다. 하지만 이 플래그를 설정한다고 해서 엔진이 지시를 따른다는 보장은 없습니다. 제공되는 보장에 대해 알아보려면 엔진 설명서를 참조하세요(있는 경우).

- Definer – [Single-line string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.

SQL에서 뷰의 정의자입니다.

- Representations – [ViewRepresentationInput](#) 객체의 배열이며 구조는 1~10개입니다.

뷰의 언어와 뷰를 정의하는 쿼리를 포함하는 구조의 목록입니다.

- SubObjects – 10개 이하의 문자열로 구성된 UTF-8 문자열입니다.

뷰를 구성하는 기본 테이블 ARN의 목록입니다.

ViewRepresentation 구조

뷰의 언어와 뷰를 정의하는 쿼리를 포함하는 구조입니다.

필드

- `Dialect` – UTF-8 문자열입니다(유효한 값: REDSHIFT | ATHENA | SPARK).

쿼리 엔진의 언어입니다.

- `DialectVersion` – 1~255바이트 길이의 UTF-8 문자열입니다.

쿼리 엔진의 언어 버전입니다. 예를 들어 3.0.0입니다.

- `ViewOriginalText` – 409,600바이트 이하 길이의 UTF-8 문자열입니다.

CREATE VIEW DDL 중에 고객이 제공한 SELECT 쿼리입니다. 이 SQL은 뷰에서 쿼리를 수행하는 동안에는 사용되지 않습니다(대신 `ViewExpandedText` 사용). `ViewOriginalText`는 사용자가 뷰를 생성한 원본 DDL 명령을 보려고 하는 SHOW CREATE VIEW와 같은 경우에 사용됩니다.

- `ViewExpandedText` – 409,600바이트 이하 길이의 UTF-8 문자열입니다.

뷰를 위한 확장된 SQL입니다. 이 SQL은 엔진이 뷰에서 쿼리를 처리하는 동안 사용됩니다. 엔진은 뷰 생성 중에 `ViewOriginalText`를 `ViewExpandedText`로 변환하기 위해 작업을 수행할 수 있습니다. 예:

- 정규화된 식별자: `SELECT * from table1 -> SELECT * from db1.table1`
- `ValidationConnection` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

뷰의 특정 표현을 검증하는 데 사용할 연결의 이름입니다.

- `IsStale` – 부울입니다.

기한 경과로 표시된 언어는 더 이상 유효하지 않으므로 해당 쿼리 엔진에서 쿼리하려면 먼저 업데이트해야 합니다.

ViewRepresentationInput 구조

Lake Formation 뷰를 업데이트하거나 생성하기 위한 표현의 세부 정보를 포함하는 구조입니다.

필드

- `Dialect` – UTF-8 문자열입니다(유효한 값: REDSHIFT | ATHENA | SPARK).
특정 표현의 엔진 유형을 지정하는 파라미터입니다.
- `DialectVersion` – 1~255바이트 길이의 UTF-8 문자열입니다.
특정 표현의 엔진 버전을 지정하는 파라미터입니다.
- `ViewOriginalText` – 409,600바이트 이하 길이의 UTF-8 문자열입니다.
뷰를 설명하는 원본 SQL 쿼리를 나타내는 문자열입니다.
- `ValidationConnection` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
뷰의 특정 표현을 검증하는 데 사용할 연결의 이름입니다.
- `ViewExpandedText` – 409,600바이트 이하 길이의 UTF-8 문자열입니다.
확장된 리소스 ARN으로 뷰를 설명하는 SQL 쿼리를 나타내는 문자열

운영

- [CreateTable](#) 액션 (파이썬: `create_table`)
- [UpdateTable](#) 액션 (Python: 업데이트 테이블)
- [DeleteTable](#) 액션 (파이썬: 삭제 테이블)
- [BatchDeleteTable](#) 액션 (Python: 배치 삭제 테이블)
- [GetTable](#) 액션 (파이썬: `get_table`)
- [GetTables](#) 액션 (파이썬: `get_tables`)
- [GetTableVersion](#) 액션 (파이썬: `get_table_version`)
- [GetTableVersions](#) 액션 (파이썬: `get_table_version`)
- [DeleteTableVersion](#) 액션 (Python: 테이블 버전 삭제)
- [BatchDeleteTableVersion](#) 액션 (Python: 배치 삭제 테이블 버전)
- [SearchTables](#) 액션 (Python: 검색 테이블)
- [GetPartitionIndexes](#) 액션 (Python: `get_partition_index`)
- [CreatePartitionIndex](#) 액션 (Python: 파티션 인덱스 생성하기)
- [DeletePartitionIndex](#) 액션 (Python: 파티션 인덱스 삭제)

- [GetColumnStatisticsForTable](#) 액션 (Python: `get_column_statistics_for_table`)
- [UpdateColumnStatisticsForTable](#) 액션 (Python: 업데이트_컬럼_통계_포터_테이블)
- [DeleteColumnStatisticsForTable](#) 액션 (Python: 테이블_열_통계_삭제_)

CreateTable 액션 (파이썬: `create_table`)

데이터 카탈로그에서 새로운 테이블 정의를 생성합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

`Table`을 생성할 데이터 카탈로그의 ID입니다. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

새로운 테이블을 생성할 카탈로그 데이터베이스입니다. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- `TableInput` – 필수(Required): [TableInput](#) 객체입니다.

`TableInput` 객체는 카탈로그에 생성할 메타데이터 테이블을 정의합니다.

- `PartitionIndexes` – [PartitionIndex](#) 객체의 배열이며 구조는 3개 이하입니다.

테이블에 생성할 파티션 인덱스(`PartitionIndex` 구조)의 목록입니다.

- `TransactionId` – [Custom string pattern #16](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

트랜잭션의 ID입니다.

- `OpenTableFormatInput` – [OpenTableFormatInput](#) 객체입니다.

오픈 형식 테이블을 생성하는 경우 `OpenTableFormatInput` 구조를 지정합니다.

응답

- 무응답 파라미터.

Errors

- AlreadyExistsException
- InvalidInputException
- EntityNotFoundException
- ResourceNumberLimitExceededException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ConcurrentModificationException
- ResourceNotReadyException

UpdateTable 액션 (Python: 업데이트_테이블)

데이터 카탈로그에서 메타데이터 테이블을 업데이트합니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 데이터 데이터베이스의 이름입니다. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- TableInput – 필수(Required): [TableInput](#) 객체입니다.

업데이트된 TableInput 객체는 카탈로그에 생성할 메타데이터 테이블을 정의합니다.

- SkipArchive – 부울입니다.

기본적으로 UpdateTable는 항상 업데이트하기 전에 테이블 보관 버전을 생성합니다. 그러나 skipArchive이 true이면 UpdateTable는 보관된 버전을 생성하지 않습니다.

- TransactionId – [Custom string pattern #16](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 콘텐츠를 업데이트할 트랜잭션 ID입니다.

- VersionId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 콘텐츠를 업데이트할 버전 ID입니다.

- ViewUpdateAction – UTF-8 문자열입니다(유효한 값: ADD | REPLACE | ADD_OR_REPLACE | DROP).

뷰를 업데이트할 때 수행할 작업입니다.

- Force – 부울입니다.

이 플래그를 true로 설정하면 일치하는 스토리지 서술자와 하위 객체 매칭 요구 사항을 무시하도록 지정할 수 있습니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException
- ResourceNumberLimitExceededException
- GlueEncryptionException
- ResourceNotReadyException

DeleteTable 액션 (파이썬: 삭제_테이블)

데이터 카탈로그에서 테이블 정의를 제거합니다.

Note

이 작업을 완료하면 삭제된 테이블에 속한 테이블 버전 및 파티션에 더 이상 액세스할 수 없습니다. AWS Glue 는 이러한 "분리된" 리소스를 서비스 재량에 따라 적시에 비동기로 삭제합니다.

관련된 모든 리소스가 즉시 삭제되도록 DeleteTable 호출 전에 DeleteTableVersion 또는 BatchDeleteTableVersion과 DeletePartition 또는 BatchDeletePartition을 사용하여 테이블에 속한 모든 리소스를 삭제하십시오.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 데이터 데이터베이스의 이름입니다. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제된 테이블 이름. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- TransactionId – [Custom string pattern #16](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 콘텐츠를 삭제할 트랜잭션 ID입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`
- `ResourceNotReadyException`

BatchDeleteTable 액션 (Python: 배치_삭제_테이블)

한 번에 여러 테이블 삭제

Note

이 작업을 완료하면 삭제된 테이블에 속한 테이블 버전 및 파티션에 더 이상 액세스할 수 없습니다. AWS Glue 는 이러한 "분리된" 리소스를 서비스 재량에 따라 적시에 비동기로 삭제합니다.

관련된 모든 리소스가 즉시 삭제되도록 `BatchDeleteTable` 호출 전에 `DeleteTableVersion` 또는 `BatchDeleteTableVersion`과 `DeletePartition` 또는 `BatchDeletePartition`을 사용하여 테이블에 속한 모든 리소스를 삭제하십시오.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 테이블이 있는 카탈로그 데이터베이스의 이름입니다. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- `TablesToDelete` – 필수(Required): 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.

삭제할 테이블 목록.

- `TransactionId` – [Custom string pattern #16](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 콘텐츠를 삭제할 트랜잭션 ID입니다.

응답

- Errors – [TableError](#) 객체의 배열입니다.

지정된 테이블을 삭제하는 중 발생한 오류 목록입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `ResourceNotReadyException`

GetTable 액션 (파이썬: `get_table`)

지정된 테이블의 데이터 카탈로그에서 Table정의를 가져옵니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스 이름입니다. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- `Name` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

정의를 가져오는 테이블의 이름입니다. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- TransactionId – [Custom string pattern #16](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 콘텐츠를 읽을 트랜잭션 ID입니다.

- QueryAsOfTime – 타임스탬프입니다.

테이블 콘텐츠를 읽을 기준 시간입니다. 설정하지 않으면 가장 최근의 트랜잭션 커밋 시간이 사용됩니다. TransactionId과(와) 함께 지정할 수 없습니다.

응답

- Table – [표](#) 객체입니다.

지정한 테이블을 정의하는 Table 객체.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

GetTables 액션 (파이썬: get_tables)

주어진 Database의 테이블 중 몇 개 혹은 모두의 정의를 가져옵니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기록할 테이블이 있는 카탈로그의 데이터베이스입니다. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- Expression – [Single-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 UTF-8 문자열입니다.

정규 표현식 패턴 테이블이 있으면 이름이 패턴과 일치하는 테이블만 반환됩니다.

- NextToken – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰을 포함합니다.

- MaxResults – 1~100의 숫자(정수)입니다.

한 번의 응답으로 반환할 최대 테이블 수.

- TransactionId – [Custom string pattern #16](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 콘텐츠를 읽을 트랜잭션 ID입니다.

- QueryAsOfTime – 타임스탬프입니다.

테이블 콘텐츠를 읽을 기준 시간입니다. 설정하지 않으면 가장 최근의 트랜잭션 커밋 시간이 사용됩니다. TransactionId과(와) 함께 지정할 수 없습니다.

응답

- TableList – [표](#) 객체의 배열입니다.

요청한 Table 객체의 목록입니다.

- NextToken – UTF-8 문자열입니다.

현재 목록 부분이 유지가 되지 않으면 연속 토큰이 존재합니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- FederationSourceException
- FederationSourceRetryableException

GetTableVersion 액션 (파이썬: get_table_version)

테이블의 지정된 버전을 가져옵니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- VersionId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

가져올 테이블 버전의 ID값입니다. VersionID는 정수의 문자열 표현입니다. 각 버전은 1씩 증가합니다.

응답

- `TableVersion` – [TableVersion](#) 객체입니다.

요청한 테이블 버전

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

GetTableVersions 액션 (파이썬: `get_table_version`)

문자열 목록을 가져와 지정된 테이블의 사용 가능한 버전을 식별합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- `NextToken` – UTF-8 문자열입니다.

이것이 첫 번째 호출이 아니면 지속적인 토큰입니다.

- `MaxResults` – 1~100의 숫자(정수)입니다.

한 번의 응답으로 반환될 최대 테이블 버전 수입니다.

응답

- `TableVersions` – [TableVersion](#) 객체의 배열입니다.

문자열 목록은 지정된 테이블의 사용 가능한 버전을 식별합니다.

- `NextToken` – UTF-8 문자열입니다.

사용 가능한 버전 목록이 마지막 버전을 포함하지 않은 경우의 연속 토큰입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeleteTableVersion 액션 (Python: 테이블_버전 삭제)

테이블의 지정된 버전을 삭제합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- **TableName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- **VersionId** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제될 테이블 버전의 ID입니다. VersionID는 정수의 문자열 표현입니다. 각 버전은 1씩 증가합니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchDeleteTableVersion 액션 (Python: 배치_삭제_테이블_버전)

테이블 버전의 지정된 배치를 삭제합니다.

요청

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 아무 것도 제공하지 않으면 기본적으로 계정 ID가 사용됩니다. AWS

- **DatabaseName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스. 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- **TableName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름 이름은 모두 소문자로 되어 있어야 Hive 호환성을 유지합니다.

- **VersionIds** – 필수(Required): 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.

삭제될 ID 버전 목록입니다. **VersionId**는 정수의 문자열 표현입니다. 각 버전은 1씩 증가합니다.

응답

- **Errors** – [TableVersionError](#) 객체의 배열입니다.

지정된 테이블 버전을 삭제하고자 할 때 발생한 오류 목록입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

SearchTables 액션 (Python: 검색_테이블)

상위 데이터베이스 뿐만 아니라 테이블 메타데이터의 속성을 기반으로 테이블 집합을 검색합니다. 텍스트 또는 필터 조건으로 검색할 수 있습니다.

Lake Formation에 정의된 보안 정책을 기반으로 액세스할 수 있는 테이블만 가져올 수 있습니다. 테이블이 반환되기 위해서는 최소한 읽기 전용 액세스 권한이 필요합니다. 테이블의 모든 열에 액세스할 수 없으면, 테이블 목록이 다시 반환될 때 이러한 열이 검색되지 않습니다. 열에 액세스할 수 있지만 열의 데이터에는 액세스할 수 없는 경우 해당 열 및 해당 열의 관련 메타데이터가 검색에 포함됩니다.

요청

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

`account_id`로 구성된 고유한 식별자입니다.

- **NextToken** – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰을 포함합니다.

- **Filters** – [PropertyPredicate](#) 객체의 배열입니다.

키-값 페어의 목록 및 검색 결과를 필터링하는 데 사용되는 비교기입니다. 조건자와 일치하는 모든 엔터티를 반환합니다.

PropertyPredicate 구조체의 Comparator 멤버는 시간 필드에만 사용되며 다른 필드 유형에서는 생략할 수 있습니다. 또한 Key=Name과 같이 문자열 값을 비교할 때 퍼지 일치 알고리즘을 사용합니다. Key 필드(예: Name 필드 값)는 특정 구두점 문자(예: -, :, # 등)에서 토큰으로 분할됩니다. 그런 다음 각 토큰은 PropertyPredicate의 Value 멤버와 정확히 일치합니다. 예를 들어 Key=Name과 Value=link가 있으면 customer-link 및 xx-link-yy라는 테이블은 반환되지만 xxlinkyy는 반환되지 않는다.

- **SearchText** – 1,024바이트 이하 길이의 값 문자열입니다.

텍스트 검색에 사용되는 문자열입니다.

인용 부호로 값을 지정하면, 정확도를 기준으로 필터링합니다.

- **SortCriteria** – [SortCriterion](#) 객체의 배열이며 구조는 1개 이하입니다.

필드 이름을 기준으로 결과를 오름차순 또는 내림차순으로 정렬하는 기준 목록입니다.

- **MaxResults** – 1~1,000의 숫자(정수)입니다.

한 번의 응답으로 반환할 최대 테이블 수.

- **ResourceShareType** – UTF-8 문자열입니다(유효한 값: FOREIGN | ALL | FEDERATED).

계정과 공유된 테이블을 검색하도록 지정할 수 있습니다. 허용 가능 값은 FOREIGN 또는 ALL입니다.

- FOREIGN으로 설정하면 계정과 공유된 테이블이 검색됩니다.
- ALL로 설정하면 계정과 공유된 테이블과 로컬 계정의 테이블이 검색됩니다.

응답

- **NextToken** – UTF-8 문자열입니다.

현재 목록 부분이 유지가 되지 않으면 연속 토큰이 존재합니다.

- **TableList** – [표](#) 객체의 배열입니다.

요청한 Table 객체의 목록입니다. SearchTables 응답은 액세스할 수 있는 테이블만 반환합니다.

Errors

- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`

GetPartitionIndexes 액션 (Python: `get_partition_index`)

테이블과 연결된 파티션 인덱스를 검색합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 있는 카탈로그 ID입니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 인덱스를 검색하려는 데이터베이스의 이름을 지정합니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 인덱스를 검색할 테이블의 이름을 지정합니다.

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰을 포함합니다.

응답

- `PartitionIndexDescriptorList` – [PartitionIndexDescriptor](#) 객체의 배열입니다.

인덱스 설명자의 목록입니다.

- `NextToken` – UTF-8 문자열입니다.

현재 목록 부분이 유지가 되지 않으면 연속 토큰이 존재합니다.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ConflictException`

CreatePartitionIndex 액션 (Python: 파티션_인덱스 생성하기)

기존 테이블에 지정된 파티션 인덱스를 생성합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 있는 카탈로그 ID입니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 인덱스를 생성하려는 데이터베이스의 이름을 지정합니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 인덱스를 생성하려는 테이블의 이름을 지정합니다.

- `PartitionIndex` – 필수(Required): [PartitionIndex](#) 객체입니다.

`PartitionIndex` 구조를 지정하여 기존 테이블에 파티션 인덱스를 생성합니다.

응답

- 무응답 파라미터.

Errors

- `AlreadyExistsException`

- `InvalidInputException`
- `EntityNotFoundException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

DeletePartitionIndex 액션 (Python: 파티션_인덱스 삭제)

기존 테이블에서 지정된 파티션 인덱스를 삭제합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 있는 카탈로그 ID입니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 인덱스를 삭제하려는 데이터베이스의 이름을 지정합니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 인덱스를 삭제하려는 테이블의 이름을 지정합니다.

- `IndexName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 파티션 인덱스의 이름입니다.

응답

- 무응답 파라미터.

Errors

- `InternalServiceException`

- `OperationTimeoutException`
- `InvalidInputException`
- `EntityNotFoundException`
- `ConflictException`
- `GlueEncryptionException`

GetColumnStatisticsForTable 액션 (Python: `get_column_statistics_for_table`)

열의 테이블 통계를 검색합니다.

이 작업에 필요한 Identity and Access Management(IAM) 권한은 `GetTable`입니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 아무 것도 제공하지 않으면 기본적으로 계정 ID가 사용됩니다. AWS

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 테이블의 이름입니다.

- `ColumnNames` – 필수(Required): 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.

열 이름의 목록입니다.

응답

- `ColumnStatisticsList` – [ColumnStatistics](#) 객체의 배열입니다.

목록 `ColumnStatistics`.

- `Errors` – [ColumnError](#) 객체의 배열입니다.

ColumnStatistics 해당 목록을 가져오지 못했습니다.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

UpdateColumnStatisticsForTable 액션 (Python: 업데이트_컬럼_통계_포터_테이블)

열의 테이블 통계를 생성하거나 업데이트합니다.

이 작업에 필요한 Identity and Access Management(IAM) 권한은 UpdateTable입니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 아무 것도 제공하지 않으면 기본적으로 계정 ID가 사용됩니다. AWS

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 테이블의 이름입니다.

- ColumnStatisticsList – 필수(Required): [ColumnStatistics](#) 객체의 배열이며 구조는 25개 이하입니다.

열 통계의 목록입니다.

응답

- Errors – [ColumnStatisticsError](#) 객체의 배열입니다.

목록 ColumnStatisticsErrors.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

DeleteColumnStatisticsForTable 액션 (Python: 테이블_열_통계_삭제_)

열의 테이블 통계를 검색합니다.

이 작업에 필요한 Identity and Access Management(IAM) 권한은 DeleteTable입니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 아무 것도 제공하지 않으면 기본적으로 계정 ID가 사용됩니다. AWS

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 테이블의 이름입니다.

- ColumnName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

열의 이름입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

파티션 API

파티션 API는 파티션을 실행할 데이터 형식 및 작업에 대해 설명합니다.

데이터 타입

- [파티션 구조](#)
- [PartitionInput 구조](#)
- [PartitionSpecWithSharedStorageDescriptor 구조](#)
- [PartitionListComposingSpec 구조](#)
- [PartitionSpecProxy 구조](#)
- [PartitionValueList 구조](#)
- [세그먼트 구조](#)
- [PartitionError 구조](#)
- [BatchUpdatePartitionFailureEntry 구조](#)
- [BatchUpdatePartitionRequestEntry 구조](#)
- [StorageDescriptor 구조](#)
- [SchemaReference 구조](#)
- [SerDelInfo 구조](#)

- [SkewedInfo 구조](#)

파티션 구조

테이블 데이터의 조각을 나타냅니다.

필드

- Values – UTF-8 문자열의 배열입니다.

파티션 값입니다.

- DatabaseName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 생성되는 카탈로그 데이터베이스의 이름입니다.

- TableName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 생성되는 데이터베이스 테이블의 이름입니다.

- CreationTime – 타임스탬프입니다.

파티션이 생성된 시간.

- LastAccessTime – 타임스탬프입니다.

파티션이 액세스된 마지막 시간.

- StorageDescriptor – [StorageDescriptor](#) 객체입니다.

파티션이 저장된 물리적 위치에 대한 정보를 제공합니다.

- Parameters – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 키-값 쌍은 파티션 파라미터를 정의합니다.

- LastAnalyzedTime – 타임스탬프입니다.

이 파티션을 위해 계산된 열 통계 마지막 시간.

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

파티션이 있는 Data Catalog의 ID입니다.

PartitionInput 구조

파티션을 생성 및 업데이트할 때 사용되는 구조입니다.

필드

- Values – UTF-8 문자열의 배열입니다.

파티션 값입니다. 이 파라미터는 SDK에 필수는 아니지만 유효한 입력을 위해 이 파라미터를 지정해야 합니다.

새 파티션의 키 값은 Amazon S3 접두사에 표시되는 파티션 키와 동일한 순서로 정렬되어야 하는 문자열 객체의 배열로 전달되어야 합니다. 그렇지 않으면 AWS Glue가 잘못된 키에 값을 추가합니다.

- LastAccessTime – 타임스탬프입니다.

파티션이 액세스된 마지막 시간.

- StorageDescriptor – [StorageDescriptor](#) 객체입니다.

파티션이 저장된 물리적 위치에 대한 정보를 제공합니다.

- Parameters – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 키-값 쌍은 파티션 파라미터를 정의합니다.

- LastAnalyzedTime – 타임스탬프입니다.

이 파티션을 위해 계산된 열 통계 마지막 시간.

PartitionSpecWithSharedStorageDescriptor 구조

물리적 위치를 공유하는 파티션의 파티션 스펙.

필드

- StorageDescriptor – [StorageDescriptor](#) 객체입니다.

공유된 물리적 스토리지 정보.

- Partitions – [Partition](#) 객체의 배열입니다.

이 물리적 위치를 공유하는 파티션 목록.

PartitionListComposingSpec 구조

관련된 파티션을 나열합니다.

필드

- Partitions – [Partition](#) 객체의 배열입니다.

구성 스펙의 파티션 목록.

PartitionSpecProxy 구조

지정된 파티션으로의 루트 경로를 제공합니다.

필드

- DatabaseName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스입니다.

- TableName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 포함된 테이블의 이름입니다.

- RootPath – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 설명되어 있는 프록시 루트 경로.

- PartitionSpecWithSharedSD – [PartitionSpecWithSharedStorageDescriptor](#) 객체입니다.

동일한 스토리지 위치를 공유하는 파티션 스펙.

- PartitionListComposingSpec – [PartitionListComposingSpec](#) 객체입니다.

파티션 목록을 지정합니다.

PartitionValueList 구조

파티션을 정의하는 값 목록을 포함합니다.

필드

- `Values` – 필수(Required): UTF-8 문자열의 배열입니다.

값 목록.

세그먼트 구조

테이블 파티션의 중첩되지 않은 리전을 정의하여 다양한 요청이 병렬로 실행되도록 허용합니다.

필드

- `SegmentNumber` – 필수(Required): None 이하의 숫자(정수)입니다.

세그먼트의 0 기반 인덱스 숫자입니다. 예를 들어 총 세그먼트 수가 4개인 경우, `SegmentNumber` 값의 범위는 0~3입니다.

- `TotalSegments` – 필수(Required): 1~10의 숫자(정수)입니다.

총 세그먼트 수입니다.

PartitionError 구조

파티션 오류에 관한 정보를 포함합니다.

필드

- `PartitionValues` – UTF-8 문자열의 배열입니다.

파티션을 정의하는 값입니다.

- `ErrorDetail` – [ErrorDetail](#) 객체입니다.

파티션 오류에 대한 세부 정보입니다.

BatchUpdatePartitionFailureEntry 구조

배치 업데이트 파티션 오류에 관한 정보를 포함합니다.

필드

- `PartitionValueList` – 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.
파티션을 정의하는 값의 목록입니다.
- `ErrorDetail` – [ErrorDetail](#) 객체입니다.
배치 업데이트 파티션 오류에 대한 세부 정보입니다.

BatchUpdatePartitionRequestEntry 구조

파티션을 업데이트하는 데 사용되는 값과 구조를 포함하는 구조입니다.

필드

- `PartitionValueList` – 필수(Required): 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.
파티션을 정의하는 값의 목록입니다.
- `PartitionInput` – 필수(Required): [PartitionInput](#) 객체입니다.
파티션을 업데이트할 때 사용되는 구조입니다.

StorageDescriptor 구조

테이블 데이터의 물리적 스토리지를 설명합니다.

필드

- `Columns` – [열](#) 객체의 배열입니다.
테이블의 `Columns` 목록입니다.
- `Location` – [URI address multi-line string pattern](#)과(와) 일치하는 2,056바이트 이하 길이의 위치 문자열입니다.
테이블의 물리적 위치입니다. 기본적으로 웨어하우스 위치, 웨어하우스의 데이터베이스 위치, 테이블 이름 순으로 나타납니다.
- `AdditionalLocations` – UTF-8 문자열의 배열입니다.
Delta 테이블이 위치한 경로를 가리키는 위치 목록입니다.

- InputFormat – [Single-line string pattern](#)과(와) 일치하는 128바이트 이하 길이의 포맷 문자열입니다.

입력 형식: SequenceFileInputFormat(이진), TextInputFormat 또는 사용자 지정 형식입니다.

- OutputFormat – [Single-line string pattern](#)과(와) 일치하는 128바이트 이하 길이의 포맷 문자열입니다.

출력 형식: SequenceFileOutputFormat(이진), IgnoreKeyTextOutputFormat 또는 사용자 지정 형식입니다.

- Compressed – 부울입니다.

테이블의 데이터가 압축되면 True이고 그렇지 않으면 False입니다.

- NumberOfBuckets - 숫자(정수)입니다.

테이블에 차원 열이 포함되어 있는 경우 이 속성을 지정해야 합니다.

- SerdeInfo – [SerDeInfo](#) 객체입니다.

직렬화/역직렬화(SerDe) 정보입니다.

- BucketColumns – UTF-8 문자열의 배열입니다.

테이블의 열, 클러스터링 열 및 버킷 열을 지정하는 그룹화하는 reducer 목록입니다.

- SortColumns – [Order](#) 객체의 배열입니다.

테이블에 있는 각 버킷의 정렬 순서를 지정하는 목록입니다.

- Parameters – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

키 값 형식의 사용자 제공 속성입니다.

- SkewedInfo – [SkewedInfo](#) 객체입니다.

열에 자주 표시되는 값에 대한 정보입니다(편향된 값).

- StoredAsSubDirectories – 부울입니다.

테이블 데이터가 하위 디렉터리에 저장되면 True이고 그렇지 않으면 False입니다.

- SchemaReference – [SchemaReference](#) 객체입니다.

AWS Glue Schema Registry에 저장된 스키마를 참조하는 객체입니다.

테이블을 생성할 때 스키마에 대한 빈 열 목록을 전달하고, 대신 스키마 참조를 사용할 수 있습니다.

SchemaReference 구조

AWS Glue Schema Registry에 저장된 스키마를 참조하는 객체입니다.

필드

- SchemaId – [SchemaId](#) 객체입니다.

스키마 ID 필드를 포함하는 구조입니다. 이것 또는 SchemaVersionId가 제공되어야 합니다.

- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전에 할당된 고유 ID입니다. 이것 또는 SchemaId가 제공되어야 합니다.

- SchemaVersionNumber - 1~100,000의 숫자(정수)입니다.

스키마의 버전 번호입니다.

SerdeInfo 구조

추출기 및 로더 역할을 하는 직렬화/역직렬화 프로그램(Serde)에 대한 정보입니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Serde의 이름입니다.

- SerializationLibrary – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

일반적으로 SerDe를 구현하는 클래스입니다. 예를 들면, `org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe`입니다.

- Parameters – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 키 문자열입니다.

각 값은 512000 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 키-값 쌍은 SerDe의 초기화 파라미터를 정의합니다.

SkewedInfo 구조

테이블에 왜곡된 값을 지정합니다. 왜곡된 값은 매우 높은 빈도를 통해 발생한 값입니다.

필드

- `SkewedColumnNames` – UTF-8 문자열의 배열입니다.

왜곡된 값이 포함된 열의 이름 목록입니다.

- `SkewedColumnValues` – UTF-8 문자열의 배열입니다.

너무 자주 나타나서 왜곡된 것으로 간주되는 값의 목록입니다.

- `SkewedColumnValueLocationMaps` – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

왜곡된 값을 이 값을 포함하는 열에 매핑.

운영

- [CreatePartition 작업\(Python: create_partition\)](#)
- [BatchCreatePartition 작업\(Python: batch_create_partition\)](#)
- [UpdatePartition 작업\(Python: update_partition\)](#)
- [DeletePartition 작업\(Python: delete_partition\)](#)
- [BatchDeletePartition 작업\(Python: batch_delete_partition\)](#)
- [GetPartition 작업\(Python: get_partition\)](#)
- [GetPartitions 작업\(Python: get_partitions\)](#)
- [BatchGetPartition 작업\(Python: batch_get_partition\)](#)
- [BatchUpdatePartition 작업\(Python: batch_update_partition\)](#)
- [GetColumnStatisticsForPartition 작업\(Python: get_column_statistics_for_partition\)](#)

- [UpdateColumnStatisticsForPartition](#) 작업(Python: `update_column_statistics_for_partition`)
- [DeleteColumnStatisticsForPartition](#) 작업(Python: `delete_column_statistics_for_partition`)

CreatePartition 작업(Python: `create_partition`)

새 파티션을 생성합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

파티션이 생성될 카탈로그의 AWS 계정 ID입니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

생성될 파티션이 있는 메타데이터 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

생성될 파티션이 있는 메타데이터 테이블 이름입니다.

- `PartitionInput` – 필수(Required): [PartitionInput](#) 객체입니다.

`PartitionInput` 구조는 생성될 파티션을 정의합니다.

응답

- 무응답 파라미터.

Errors

- `InvalidInputException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`
- `InternalServiceException`
- `EntityNotFoundException`

- `OperationTimeoutException`
- `GlueEncryptionException`

BatchCreatePartition 작업(Python: `batch_create_partition`)

배치 작업에서 하나 이상의 파티션을 만듭니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

파티션이 생성될 카탈로그 ID입니다. 현재는 AWS 계정 ID여야 합니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

생성될 파티션이 있는 메타데이터 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

생성될 파티션이 있는 메타데이터 테이블 이름입니다.

- `PartitionInputList` – 필수(Required): [PartitionInput](#) 객체의 배열이며 구조는 100개 이하입니다.

`PartitionInput` 구조 목록은 생성될 파티션을 정의합니다.

응답

- `Errors` – [PartitionError](#) 객체의 배열입니다.

요청된 파티션을 생성하고자 할 때 발생한 오류입니다.

Errors

- `InvalidInputException`
- `AlreadyExistsException`
- `ResourceNumberLimitExceededException`

- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `GlueEncryptionException`

UpdatePartition 작업(Python: `update_partition`)

파티션 업데이트

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

업데이트되는 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

문제의 테이블이 있는 카탈로그 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트되는 파티션이 있는 테이블 이름입니다.

- `PartitionValueList` – 필수(Required): 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.

업데이트할 파티션을 정의하는 파티션 키 값 목록입니다.

- `PartitionInput` – 필수(Required): [PartitionInput](#) 객체입니다.

파티션을 업데이트할 새로운 파티션 객체.

`Values` 속성을 변경할 수 없습니다. 파티션의 파티션 키 값을 변경하려면 파티션을 삭제하고 다시 생성합니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

DeletePartition 작업(Python: delete_partition)

지정된 파티션을 삭제합니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

삭제되는 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

문제의 테이블이 있는 카탈로그 데이터베이스 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제될 파티션이 포함된 테이블 이름입니다.

- PartitionValues – 필수(Required): UTF-8 문자열의 배열입니다.

파티션을 정의하는 값입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchDeletePartition 작업(Python: batch_delete_partition)

배치 작업에서 하나 이상의 파티션을 삭제합니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

삭제되는 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

문제의 테이블이 있는 카탈로그 데이터베이스 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제될 파티션이 포함된 테이블 이름입니다.

- PartitionsToDelete – 필수(Required): [PartitionValueList](#) 객체의 배열이며 구조는 25개 이하입니다.

PartitionInput 구조 목록은 삭제되는 파티션을 정의합니다.

응답

- Errors – [PartitionError](#) 객체의 배열입니다.

요청된 파티션을 삭제하고자 할 때 발생한 오류입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetPartition 작업(Python: `get_partition`)

지정된 파티션에 대한 정보를 가져옵니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 테이블의 이름입니다.

- `PartitionValues` – 필수(Required): UTF-8 문자열의 배열입니다.

파티션을 정의하는 값입니다.

응답

- `Partition` – [Partition](#) 객체입니다.

`Partition` 객체의 형식으로 요청된 정보입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`
- `FederationSourceException`
- `FederationSourceRetryableException`

GetPartitions 작업(Python: `get_partitions`)

테이블의 파티션에 대한 정보를 가져옵니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 테이블의 이름입니다.

- `Expression` – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 조건자 문자열입니다.

반환할 파티션을 필터링하는 표현식입니다.

이 표현식은 SQL WHERE 필터 절과 유사한 SQL 구문을 사용합니다. SQL 문 파서 [JSQLParser](#)는 이 표현식을 구문 분석합니다.

연산자: 다음은 Expression API 호출에서 사용할 수 있는 연산자입니다.

=

두 피연산자의 값이 동일한지 여부를 확인하고, 동일한 경우에는 이 조건이 true가 됩니다.

예제: '변수 a'에는 10이 들어 있고 '변수 b'에는 20이 들어 있다고 가정해 보겠습니다.

(a = b)는 true가 아닙니다.

<>

두 피연산자의 값이 동일한지 여부를 확인하고, 값이 동일하지 않으면 이 조건이 true가 됩니다.

예제: (a <> b)는 true입니다.

>

왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 큰지 여부를 확인하고, 크면 이 조건이 true가 됩니다.

예제: (a > b)는 true가 아닙니다.

<

왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작은지 여부를 확인하고, 크면 이 조건이 true가 됩니다.

예제: (a < b)는 true입니다.

>=

왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 크거나 같은지 여부를 확인하고, 크거나 같으면 이 조건이 true가 됩니다.

예제: (a >= b)는 true가 아닙니다.

<=

왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작거나 같은지 여부를 확인하고, 작거나 같으면 이 조건이 true가 됩니다.

예제: (a <= b)는 true입니다.

AND, OR, IN, BETWEEN, LIKE, NOT, IS NULL

논리 연산자

지원되는 파티션 키 유형: 다음은 지원되는 파티션 키입니다.

- string
- date
- timestamp
- int
- bigint
- long
- tinyint
- smallint
- decimal

유효하지 않은 유형이 있으면 예외가 발생합니다.

다음 목록에는 각 유형에 대한 유효 연산자가 표시됩니다. 크롤러를 정의할 때 `partitionKey` 유형이 카탈로그 파티션과 호환될 `STRING`으로 생성됩니다.

단순 API 호출:

Example

`twitter_partition` 표에는 파티션 세 개가 있습니다.

```
year = 2015
  year = 2016
  year = 2017
```

Example

Get partition `year`가 2015와 같음

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
  --expression "year*='2015'"
```

Example

Get Partition `year`가 2016~2018(제외)

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
--expression "year>'2016' AND year<'2018'"
```

Example

Get Partition year가 2015~2018(포함) 다음 API 호출은 서로에 상응합니다.

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
--expression "year>='2015' AND year<='2018'"

aws glue get-partitions --database-name dbname --table-name
twitter_partition
--expression "year BETWEEN 2015 AND 2018"

aws glue get-partitions --database-name dbname --table-name
twitter_partition
--expression "year IN (2015,2016,2017,2018)"
```

Example

와일드카드 파티션 필터. 여기서는 다음 호출 출력이 partition year=2017입니다. 정규식은 LIKE에서 지원되지 않습니다.

```
aws glue get-partitions --database-name dbname --table-name twitter_partition
--expression "year LIKE '%7'"
```

- NextToken – UTF-8 문자열입니다.

이것이 파티션을 가져오기 위한 첫 번째 호출이 아니면 지속적인 토큰입니다.

- Segment – [세그먼트](#) 객체입니다.

이 요청에 따라 스캔할 테이블 파티션의 세그먼트입니다.

- MaxResults – 1~1,000의 숫자(정수)입니다.

한 번의 응답으로 반환할 최대 파티션 수.

- ExcludeColumnSchema – 부울입니다.

true이면 파티션 열 스키마를 반환하지 않도록 지정합니다. 파티션 값이나 위치와 같은 다른 파티션 속성에만 관심이 있을 때 유용합니다. 이 접근 방식을 사용하면 중복 데이터가 반환되지 않으므로 큰 응답 문제를 피할 수 있습니다.

- TransactionId – [Custom string pattern #16](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 콘텐츠를 읽을 트랜잭션 ID입니다.

- QueryAsOfTime – 타임스탬프입니다.

파티션 콘텐츠를 읽을 기준 시간입니다. 설정하지 않으면 가장 최근의 트랜잭션 커밋 시간이 사용됩니다. TransactionId과(와) 함께 지정할 수 없습니다.

응답

- Partitions – [Partition](#) 객체의 배열입니다.

요청된 파티션 목록.

- NextToken – UTF-8 문자열입니다.

반환된 파티션 목록에 마지막 항목이 포함되지 않은 경우의 연속 토큰입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException
- InvalidStateException
- ResourceNotReadyException
- FederationSourceException
- FederationSourceRetryableException

BatchGetPartition 작업(Python: batch_get_partition)

배치 요청에 따라 파티션을 가져옵니다.

요청

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- **DatabaseName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스 이름입니다.

- **TableName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 테이블의 이름입니다.

- **PartitionsToGet** – 필수(Required): [PartitionValueList](#) 객체의 배열이며 구조는 1,000개 이하입니다.

파티션 값의 목록은 가져올 파티션을 식별합니다.

응답

- **Partitions** – [Partition](#) 객체의 배열입니다.

요청한 파티션의 목록입니다.

- **UnprocessedKeys** – [PartitionValueList](#) 객체의 배열이며 구조는 1,000개 이하입니다.

파티션이 반환되지 않은 요청의 파티션 값 목록입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`

- `InternalServiceException`
- `GlueEncryptionException`
- `InvalidStateException`
- `FederationSourceException`
- `FederationSourceRetryableException`

BatchUpdatePartition 작업(Python: `batch_update_partition`)

배치 작업에서 하나 이상의 파티션을 업데이트합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

파티션이 업데이트될 카탈로그의 ID입니다. 현재는 AWS 계정 ID여야 합니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트될 파티션이 있는 메타데이터 데이터베이스의 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트될 파티션이 있는 메타데이터 테이블의 이름입니다.

- `Entries` – 필수(Required): [BatchUpdatePartitionRequestEntry](#) 객체의 배열이며 구조는 1~100개입니다.

업데이트할 최대 100개의 `BatchUpdatePartitionRequestEntry` 객체 목록입니다.

응답

- `Errors` – [BatchUpdatePartitionFailureEntry](#) 객체의 배열입니다.

요청된 파티션을 업데이트하고자 할 때 발생한 오류입니다.

`BatchUpdatePartitionFailureEntry` 객체의 목록.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InternalServiceException`
- `GlueEncryptionException`

GetColumnStatisticsForPartition 작업(Python: `get_column_statistics_for_partition`)

열의 파티션 통계를 검색합니다.

이 작업에 필요한 Identity and Access Management(IAM) 권한은 `GetPartition`입니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 테이블의 이름입니다.

- `PartitionValues` – 필수(Required): UTF-8 문자열의 배열입니다.

파티션을 식별하는 파티션 값의 목록입니다.

- `ColumnNames` – 필수(Required): 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.

열 이름의 목록입니다.

응답

- ColumnStatisticsList – [ColumnStatistics](#) 객체의 배열입니다.

검색에 실패한 ColumnStatistics 목록입니다.

- Errors – [ColumnError](#) 객체의 배열입니다.

열 통계 데이터를 검색하는 동안 오류가 발생했습니다.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

UpdateColumnStatisticsForPartition 작업(Python: update_column_statistics_for_partition)

열의 파티션 통계를 생성하거나 업데이트합니다.

이 작업에 필요한 Identity and Access Management(IAM) 권한은 UpdatePartition입니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션이 있는 카탈로그 데이터베이스 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

파티션 테이블의 이름입니다.

- `PartitionValues` – 필수(Required): UTF-8 문자열의 배열입니다.

파티션을 식별하는 파티션 값의 목록입니다.

- `ColumnStatisticsList` – 필수(Required): [ColumnStatistics](#) 객체의 배열이며 구조는 25개 이하입니다.

열 통계의 목록입니다.

응답

- `Errors` – [ColumnStatisticsError](#) 객체의 배열입니다.

열 통계 데이터를 업데이트하는 동안 오류가 발생했습니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `GlueEncryptionException`

`DeleteColumnStatisticsForPartition` 작업(Python: `delete_column_statistics_for_partition`)

열의 파티션 열 통계를 삭제합니다.

이 작업에 필요한 Identity and Access Management(IAM) 권한은 `DeletePartition`입니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

문제의 파티션이 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- **DatabaseName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
파티션이 있는 카탈로그 데이터베이스 이름입니다.
- **TableName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
파티션 테이블의 이름입니다.
- **PartitionValues** – 필수(Required): UTF-8 문자열의 배열입니다.
파티션을 식별하는 파티션 값의 목록입니다.
- **ColumnName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
열의 이름입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

연결 API

연결 API는 AWS Glue 연결 데이터 유형과 연결을 생성, 삭제, 업데이트 및 나열하기 위한 API에 대해 설명합니다.

데이터 타입

- [연결 구조](#)
- [ConnectionInput 구조](#)

- [PhysicalConnectionRequirements](#) 구조
- [GetConnectionsFilter](#) 구조

연결 구조

데이터 원본으로 연결을 정의합니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

연결 정의 이름입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

연결에 대한 설명입니다.

- ConnectionType – UTF-8 문자열입니다(유효한 값: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE).

연결 유형입니다. 현재 SFTP는 지원되지 않습니다.

- MatchCriteria – 10개 이하의 문자열로 구성된 UTF-8 문자열입니다.

이 연결을 선택할 때 사용할 수 있는 기준입니다.

- ConnectionProperties – 100개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다(유효한 값: HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME | JDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD | KAFKA_CLIENT_KEY_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | SECRET_ID | CONNECTOR_URL | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME | KAFKA_SASL_MECHANISM | KAFKA_SASL_PLAIN_USERNAME | KAFKA_SASL_PLAIN_PASSWORD | ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD | KAFKA_SASL_SCRAM_USERNAME

| KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_SCRAM_SECRETS_ARN |
 ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB
 | KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE |
 KAFKA_SASL_GSSAPI_PRINCIPAL | ROLE_ARN).

각 값은 1,024 바이트 이하 길이의 값 문자열입니다.

이러한 키-값 쌍은 연결용 파라미터를 정의합니다.

- HOST - 호스트 URI: 데이터베이스 호스트의 IPv4 주소 또는 FQDN(fully qualified domain name: 정규화된 도메인 이름).
- PORT - 데이터베이스 호스트가 데이터베이스 연결을 수신 중인 포트의 포트 번호(1,024~65,535)입니다.
- USER_NAME- 데이터베이스에 로그인할 이름. USER_NAME의 값 문자열은 "USERNAME"입니다.
- PASSWORD- 사용자 이름에 해당하는 암호(사용할 경우).
- ENCRYPTED_PASSWORD - 데이터 카탈로그 암호화 설정에서 ConnectionPasswordEncryption을 설정하여 연결 암호 보호를 활성화하면 이 필드가 암호화된 암호를 저장합니다.
- JDBC_DRIVER_JAR_URI - 사용할 JDBC 드라이버가 포함된 JAR 파일의 Amazon Simple Storage Service(Amazon S3) 경로.
- JDBC_DRIVER_CLASS_NAME- 사용할 JDBC 드라이버의 클래스 이름.
- JDBC_ENGINE- 사용할 JDBC 엔진의 이름.
- JDBC_ENGINE_VERSION - 사용할 JDBC 엔진의 버전.
- CONFIG_FILES - (추후 사용 예약.)
- INSTANCE_ID- 사용할 인스턴스 ID.
- JDBC_CONNECTION_URL - JDBC 데이터 원본에 연결하기 위한 URL입니다.
- JDBC_ENFORCE_SSL - 호스트 이름이 일치하는 Secure Sockets Layer(SSL)를 클라이언트의 JDBC 연결용으로 적용할지 여부를 지정하는 부울 문자열(true, false) 기본값은 false입니다.
- CUSTOM_JDBC_CERT - 고객의 루트 인증서를 지정하는 Amazon S3 위치입니다. AWS Glue는 이 루트 인증서를 사용하여 고객 데이터베이스에 연결할 때 고객의 인증서를 검증합니다. AWS Glue는 X.509 인증서만 처리합니다. 인증서는 DER로 인코딩되어 Base64 인코딩 PEM 형식으로 제공되어야 합니다.
- SKIP_CUSTOM_JDBC_CERT_VALIDATION - 기본적으로 false입니다. AWS Glue는 고객 인증서의 서명 알고리즘 및 주제 퍼블릭 키 알고리즘을 검증합니다. 서명 알고리즘에 허용되는 유일한

연결 알고리즘은 SHA256withRSA, SHA384withRSA 또는 SHA512withRSA. 주제 퍼블릭 키 알고리즘

의 경우, 키 길이는 2048 이상이어야 합니다. 이 속성 값을 true로 설정하여 AWS Glue의 고객 인증서 검증을 건너뛸 수 있습니다.

- CUSTOM_JDBC_CERT_STRING - 중간자 (man-in-the-middle) 공격을 방지하기 위해 도메인 일치 또는 고유 이름 일치에 사용되는 사용자 지정 JDBC 인증서 문자열입니다. Oracle Database에서는 SSL_SERVER_CERT_DN으로, Microsoft SQL Server에서는 hostNameInCertificate로 사용됩니다.
- CONNECTION_URL - 일반(비JDBC) 데이터 원본에 연결하기 위한 URL입니다.
- SECRET_ID - 자격 증명의 보안 관리자에 사용되는 보안 암호 ID입니다.
- CONNECTOR_URL - MARKETPLACE 또는 CUSTOM 연결을 위한 커넥터 URL입니다.
- CONNECTOR_TYPE - MARKETPLACE 또는 CUSTOM 연결을 위한 커넥터 유형입니다.
- CONNECTOR_CLASS_NAME - MARKETPLACE 또는 CUSTOM 연결을 위한 커넥터 클래스 이름입니다.
- KAFKA_BOOTSTRAP_SERVERS - Kafka 클라이언트에서 연결하고 자체 부트스트랩하는 Kafka 클러스터에 있는 Apache Kafka 브로커의 주소입니다 호스트 및 포트 쌍의 쉼표로 구분된 목록입니다.
- KAFKA_SSL_ENABLED - Apache Kafka 연결에서 SSL을 사용할지 아니면 사용 중지할지 여부입니다. 기본값은 "true"입니다.
- KAFKA_CUSTOM_CERT - 프라이빗 CA 인증서 파일(.pem 포맷)의 Amazon S3 URL입니다. 기본값은 빈 문자열입니다.
- KAFKA_SKIP_CUSTOM_CERT_VALIDATION - CA 인증서 파일의 검증을 건너뛸지 여부입니다. AWS Glue는 SHA256withRSA, SHA384withRSA 및 SHA512withRSA의 세 가지 알고리즘에 대해 검증합니다. 기본값은 "false"입니다.
- KAFKA_CLIENT_KEYSTORE - Kafka 클라이언트 측 인증을 위한 클라이언트 키 스토어 파일의 Amazon S3 위치입니다(선택 사항).
- KAFKA_CLIENT_KEYSTORE_PASSWORD - 제공된 키 스토어에 액세스하기 위한 암호입니다(선택 사항).
- KAFKA_CLIENT_KEY_PASSWORD - 키 스토어는 여러 키로 구성 될 수 있으므로 Kafka 서버 측 키와 함께 사용할 클라이언트 키에 액세스하기 위한 암호입니다(선택 사항).
- ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD - Kafka 클라이언트 키 스토어 암호의 암호화된 버전입니다(사용자가 AWS Glue 암호 암호화 설정을 선택한 경우).
- ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD - Kafka 클라이언트 키 암호의 암호화된 버전입니다(사용자가 AWS Glue 암호 암호화 설정을 선택한 경우).

- KAFKA_SASL_MECHANISM - "SCRAM-SHA-512", "GSSAPI", "AWS_MSK_IAM" 또는 "PLAIN". 다음은 지원되는 [SASL 메커니즘](#)입니다.
- KAFKA_SASL_PLAIN_USERNAME - "PLAIN" 메커니즘으로 인증하는 데 사용되는 일반 텍스트 사용자 이름입니다.
- KAFKA_SASL_PLAIN_PASSWORD - "PLAIN" 메커니즘으로 인증하는 데 사용되는 일반 텍스트 암호입니다.
- ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD - Kafka SASL PLAIN 암호의 암호화된 버전입니다(사용자가 AWS Glue 암호 암호화 설정을 선택한 경우).
- KAFKA_SASL_SCRAM_USERNAME - "SCRAM-SHA-512" 메커니즘으로 인증하는 데 사용되는 일반 텍스트 사용자 이름입니다.
- KAFKA_SASL_SCRAM_PASSWORD - "SCRAM-SHA-512" 메커니즘으로 인증하는 데 사용되는 일반 텍스트 암호입니다.
- ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD - Kafka SASL SCRAM 암호의 암호화된 버전입니다(사용자가 AWS Glue 암호 암호화 설정을 선택한 경우).
- KAFKA_SASL_SCRAM_SECRETS_ARN - AWS Secrets Manager에서 보안 암호의 Amazon 리소스 이름(ARN)입니다.
- KAFKA_SASL_GSSAPI_KEYTAB - Kerberos keytab 파일의 S3 위치입니다. keytab은 하나 이상의 보안 주체에 대한 장기 키를 저장합니다. 자세한 내용은 [MIT Kerberos Documentation: Keytab](#)(MIT Kerberos 설명서: Keytab)을 참조하세요.
- KAFKA_SASL_GSSAPI_KRB5_CONF - Kerberos krb5.conf 파일의 S3 위치입니다. krb5.conf는 KDC 서버의 위치와 같은 Kerberos 구성 정보를 저장합니다. 자세한 내용은 [MIT Kerberos Documentation: krb5.conf](#)(MIT Kerberos 설명서: krb5.conf)를 참조하세요.
- KAFKA_SASL_GSSAPI_SERVICE - [Kafka 구성](#)에서 `sasl.kerberos.service.name`으로 설정된 Kerberos 서비스 이름입니다.
- KAFKA_SASL_GSSAPI_PRINCIPAL - AWS Glue에서 사용하는 Kerberos 보안 주체의 이름입니다. 자세한 내용은 [Kafka Documentation: Configuring Kafka Brokers](#)(Kafka 설명서: Kafka 브로커 구성)를 참조하세요.
- PhysicalConnectionRequirements - [PhysicalConnectionRequirements](#) 객체입니다.
Virtual Private Cloud(VPC) 및 SecurityGroup과 같이 이 연결을 설정하는 데 필요한 물리적 연결 요구 사항입니다.
- CreationTime - 타임스탬프입니다.

이 연결 정의가 생성된 시간의 타임스탬프입니다.

- **LastUpdatedTime** – 타임스탬프입니다.
이 연결 정의가 업데이트된 마지막 시간의 타임스탬프입니다.
- **LastUpdatedBy** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 연결 정의를 마지막으로 업데이트한 사용자, 그룹 혹은 역할입니다.
- **Status** – UTF-8 문자열입니다(유효한 값: READY | IN_PROGRESS | FAILED).
연결 상태입니다. READY, IN_PROGRESS 또는 FAILED 중 하나일 수 있습니다.
- **StatusReason** – UTF-8 문자열입니다(1~16384바이트).
연결 상태의 이유입니다.
- **LastConnectionValidationTime** – 타임스탬프입니다.
이 연결이 마지막으로 검증된 시간의 타임스탬프입니다.
- **AuthenticationConfiguration** – [AuthenticationConfiguration](#) 객체입니다.
연결의 인증 속성입니다.

ConnectionInput 구조

생성 혹은 업데이트를 위한 연결을 지정할 때 사용되는 구조입니다.

필드

- **Name** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
연결의 이름입니다.
- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.
연결에 대한 설명입니다.
- **ConnectionType** – 필수: UTF-8 문자열입니다(유효한 값: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE).
연결 유형입니다. 현재 이러한 유형이 지원됩니다.
 - JDBC - JDBC(Java Database Connectivity)를 통해 데이터베이스에 대한 연결을 지정합니다.

JDBC 연결은 다음 ConnectionParameters를 사용합니다.

- 필수 항목: JDBC_CONNECTION_URL 또는 (HOST, PORT, JDBC_ENGINE) 모두.
- 필수 항목: SECRET_ID 또는 (USERNAME, PASSWORD) 모두.
- 선택 사항: JDBC_ENFORCE_SSL, CUSTOM_JDBC_CERT, CUSTOM_JDBC_CERT_STRING, SKIP_CUSTOM_JDBC_CERT_VALIDATION. 이러한 파라미터는 JDBC로 SSL을 구성하는 데 사용됩니다.
- KAFKA - Apache Kafka 스트리밍 플랫폼에 대한 연결을 지정합니다.

KAFKA 연결은 다음 ConnectionParameters를 사용합니다.

- 필수 항목: KAFKA_BOOTSTRAP_SERVERS.
- 선택 사항: KAFKA_SSL_ENABLED, KAFKA_CUSTOM_CERT, KAFKA_SKIP_CUSTOM_CERT_VALIDATION. 이러한 파라미터는 KAFKA로 SSL을 구성하는 데 사용됩니다.
- 선택 사항: KAFKA_CLIENT_KEYSTORE, KAFKA_CLIENT_KEYSTORE_PASSWORD, KAFKA_CLIENT_KEY_PASSWORD, ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD, ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD. 이러한 파라미터는 KAFKA에서 SSL로 TLS 클라이언트 구성을 지정하는 데 사용됩니다.
- 선택 사항: KAFKA_SASL_MECHANISM SCRAM-SHA-512, GSSAPI 또는 AWS_MSK_IAM으로 지정할 수 있습니다.
- 선택 사항: KAFKA_SASL_SCRAM_USERNAME, KAFKA_SASL_SCRAM_PASSWORD, ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD. 이러한 파라미터는 KAFKA로 SASL/SCRAM-SHA-512 인증을 구성하는 데 사용됩니다.
- 선택 사항: KAFKA_SASL_GSSAPI_KEYTAB, KAFKA_SASL_GSSAPI_KRB5_CONF, KAFKA_SASL_GSSAPI_SERVICE, KAFKA_SASL_GSSAPI_PRINCIPAL. 이러한 파라미터는 KAFKA로 SASL/GSSAPI 인증을 구성하는 데 사용됩니다.
- MONGODB - MongoDB 문서 데이터베이스에 대한 연결을 지정합니다.

MONGODB 연결은 다음 ConnectionParameters를 사용합니다.

- 필수 항목: CONNECTION_URL.
- 필수 항목: SECRET_ID 또는 (USERNAME, PASSWORD) 모두.
- SALESFORCE - OAuth 인증을 사용하여 Salesforce 연결을 지정합니다.
- AuthenticationConfiguration 멤버를 구성해야 합니다.

- NETWORK - Amazon Virtual Private Cloud(Amazon VPC) 환경 내의 데이터 원본에 대한 네트워크 연결을 지정합니다.

NETWORK 연결에는 ConnectionParameters가 필요하지 않습니다. 대신 PhysicalConnectionRequirements를 제공합니다.

- MARKETPLACE - AWS Marketplace에서 구입한 커넥터에 포함된 구성 설정을 사용하여 AWS Glue에서 기본적으로 지원하지 않는 데이터 스토어에서 읽고 씁니다.

MARKETPLACE 연결은 다음 ConnectionParameters를 사용합니다.

- 필수 항목: CONNECTOR_TYPE, CONNECTOR_URL, CONNECTOR_CLASS_NAME, CONNECTION_URL.
- JDBC CONNECTOR_TYPE 연결에 필요: SECRET_ID 또는 (USERNAME, PASSWORD) 모두.
- CUSTOM - AWS Glue에서 기본적으로 지원하지 않는 데이터 스토어에서 읽고 쓰기 위해 사용자 정의 커넥터에 포함된 구성 설정을 사용합니다.

SFTP는 지원되지 않습니다.

선택적 ConnectionProperties를 사용하여 AWS Glue의 기능을 구성하는 방법에 대한 자세한 내용을 알아보려면 [AWS Glue 연결 속성](#)을 참조하세요.

선택적 ConnectionProperties를 사용하여 AWS Glue Studio의 기능을 구성하는 방법에 대한 자세한 내용을 알아보려면 [커넥터 및 연결 사용](#)을 참조하세요.

- MatchCriteria - 10개 이하의 문자열로 구성된 UTF-8 문자열입니다.

이 연결을 선택할 때 사용할 수 있는 기준입니다.

- ConnectionProperties - 필수(Required): 100개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다(유효한 값: HOST | PORT | USERNAME="USER_NAME" | PASSWORD | ENCRYPTED_PASSWORD | JDBC_DRIVER_JAR_URI | JDBC_DRIVER_CLASS_NAME | JDBC_ENGINE | JDBC_ENGINE_VERSION | CONFIG_FILES | INSTANCE_ID | JDBC_CONNECTION_URL | JDBC_ENFORCE_SSL | CUSTOM_JDBC_CERT | SKIP_CUSTOM_JDBC_CERT_VALIDATION | CUSTOM_JDBC_CERT_STRING | CONNECTION_URL | KAFKA_BOOTSTRAP_SERVERS | KAFKA_SSL_ENABLED | KAFKA_CUSTOM_CERT | KAFKA_SKIP_CUSTOM_CERT_VALIDATION | KAFKA_CLIENT_KEYSTORE | KAFKA_CLIENT_KEYSTORE_PASSWORD | KAFKA_CLIENT_KEY_PASSWORD | ENCRYPTED_KAFKA_CLIENT_KEYSTORE_PASSWORD

| ENCRYPTED_KAFKA_CLIENT_KEY_PASSWORD | SECRET_ID | CONNECTOR_URL
 | CONNECTOR_TYPE | CONNECTOR_CLASS_NAME | KAFKA_SASL_MECHANISM
 | KAFKA_SASL_PLAIN_USERNAME | KAFKA_SASL_PLAIN_PASSWORD |
 ENCRYPTED_KAFKA_SASL_PLAIN_PASSWORD | KAFKA_SASL_SCRAM_USERNAME
 | KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_SCRAM_SECRETS_ARN |
 ENCRYPTED_KAFKA_SASL_SCRAM_PASSWORD | KAFKA_SASL_GSSAPI_KEYTAB
 | KAFKA_SASL_GSSAPI_KRB5_CONF | KAFKA_SASL_GSSAPI_SERVICE |
 KAFKA_SASL_GSSAPI_PRINCIPAL | ROLE_ARN).

각 값은 1,024 바이트 이하 길이의 값 문자열입니다.

이러한 키-값 쌍은 연결용 파라미터를 정의합니다.

- PhysicalConnectionRequirements – [PhysicalConnectionRequirements](#) 객체입니다.

Virtual Private Cloud(VPC) 및 SecurityGroup과 같이 이 연결을 설정하는 데 필요한 물리적 연결 요구 사항입니다.

- AuthenticationConfiguration – [AuthenticationConfigurationInput](#) 객체입니다.

연결의 인증 속성입니다. Salesforce 연결에 사용됩니다.

- ValidateCredentials – 부울입니다.

연결 생성 중에 자격 증명을 검증하기 위한 플래그입니다. Salesforce 연결에 사용됩니다. 기본값은 true입니다.

PhysicalConnectionRequirements 구조

GetConnection 응답의 OAuth 클라이언트 앱입니다.

필드

- SubnetId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

연결에서 사용하는 서브넷 ID입니다.

- SecurityGroupIdList – 50개 이하의 문자열로 구성된 UTF-8 문자열입니다.

연결에서 사용하는 보안 그룹 ID 목록입니다.

- AvailabilityZone – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

연결의 가용 영역입니다.

GetConnectionsFilter 구조

GetConnections API 작업에서 반환하는 연결 정의를 필터링합니다.

필드

- **MatchCriteria** – 10개 이하의 문자열로 구성된 UTF-8 문자열입니다.

연결 정의가 반환되도록 연결 정의에 기록된 기준과 반드시 일치해야 하는 기준 문자열입니다.

- **ConnectionType** – UTF-8 문자열입니다(유효한 값: JDBC | SFTP | MONGODB | KAFKA | NETWORK | MARKETPLACE | CUSTOM | SALESFORCE).

반환할 연결 유형입니다. 현재 SFTP는 지원되지 않습니다.

운영

- [CreateConnection 작업\(Python: create_connection\)](#)
- [DeleteConnection 작업\(Python: delete_connection\)](#)
- [GetConnection 작업\(Python: get_connection\)](#)
- [GetConnections 작업\(Python: get_connections\)](#)
- [UpdateConnection 작업\(Python: update_connection\)](#)
- [BatchDeleteConnection 작업\(Python: batch_delete_connection\)](#)

CreateConnection 작업(Python: create_connection)

데이터 카탈로그에서 연결 정의를 생성합니다.

페더레이션 리소스를 생성하는 데 사용되는 연결에는 IAM `glue:PassConnection` 권한이 필요합니다.

요청

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

연결을 생성할 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `ConnectionInput` – 필수(Required): [ConnectionInput](#) 객체입니다.

생성할 연결을 정의하는 `ConnectionInput` 객체입니다.

- `Tags` – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

연결에 할당하는 태그입니다.

응답

- `CreateConnectionStatus` – UTF-8 문자열입니다(유효한 값: `READY` | `IN_PROGRESS` | `FAILED`).

연결 생성 요청의 상태입니다. VPC를 통해 토큰을 교환하는 OAuth 연결을 생성하는 경우와 같이, 특정 인증 유형의 경우 요청에 다소 시간이 걸릴 수 있습니다.

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `GlueEncryptionException`

DeleteConnection 작업(Python: `delete_connection`)

데이터 카탈로그에서 연결을 삭제합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

연결이 지속될 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- **ConnectionName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 연결의 이름입니다.

응답

- 무응답 파라미터.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`

GetConnection 작업(Python: `get_connection`)

데이터 카탈로그에서 연결 정의를 검색합니다.

요청

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

연결이 지속될 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- **Name** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

검색할 연결 정의 이름입니다.

- **HidePassword** – 부울입니다.

암호를 반환하지 않고 연결 메타데이터를 검색할 수 있습니다. 예를 들어 AWS Glue 콘솔은 이 플래그를 사용하여 연결을 검색하되 암호를 표시하지 않습니다. 호출자에게 AWS KMS 키를 사용하여 암호를 복호화할 권한이 없지만 연결 속성의 나머지에 액세스할 권한이 있는 경우 이 파라미터를 설정합니다.

응답

- `Connection` – [연결](#) 객체입니다.

요청한 연결 정의입니다.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

GetConnections 작업(Python: `get_connections`)

데이터 카탈로그에서 연결 정의의 목록을 검색합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

연결이 지속될 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `Filter` – [GetConnectionsFilter](#) 객체입니다.

반환되는 연결을 관리하는 필터입니다.

- `HidePassword` – 부울입니다.

암호를 반환하지 않고 연결 메타데이터를 검색할 수 있습니다. 예를 들어 AWS Glue 콘솔은 이 플래그를 사용하여 연결을 검색하되 암호를 표시하지 않습니다. 호출자에게 AWS KMS 키를 사용하여 암호를 복호화할 권한이 없지만 연결 속성의 나머지에 액세스할 권한이 있는 경우 이 파라미터를 설정합니다.

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

- `MaxResults` – 1~1,000의 숫자(정수)입니다.

한 번의 응답으로 반환될 최대 결과 수입니다.

응답

- `ConnectionList` – [연결](#) 객체의 배열입니다.

요청한 연결 정의의 목록입니다.

- `NextToken` – UTF-8 문자열입니다.

반환된 연결 목록이 필터링된 연결의 마지막을 포함하지 않는 경우의 연속 토큰입니다.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

UpdateConnection 작업(Python: `update_connection`)

데이터 카탈로그에서 연결 정의를 업데이트합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

연결이 지속될 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `Name` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트할 연결 정의 이름입니다.

- `ConnectionInput` – 필수(Required): [ConnectionInput](#) 객체입니다.

질의를 따른 연결을 재정의하는 `ConnectionInput` 객체입니다.

응답

- 무응답 파라미터.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`
- `GlueEncryptionException`

BatchDeleteConnection 작업(Python: `batch_delete_connection`)

데이터 카탈로그에서 연결 정의의 목록을 삭제합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

연결이 지속될 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `ConnectionNameList` – 필수(Required): 25개 이하의 문자열로 구성된 UTF-8 문자열입니다.

삭제할 연결 이름의 목록입니다.

응답

- `Succeeded` – UTF-8 문자열의 배열입니다.

성공적으로 삭제된 연결 정의 이름의 목록입니다.

- `Errors` – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 [ErrorDetail](#) 객체입니다.

오류 세부 정보에서 성공적으로 삭제된 연결 이름의 맵입니다.

Errors

- `InternalServiceException`
- `OperationTimeoutException`

인증 구성

- [AuthenticationConfiguration 구조](#)
- [AuthenticationConfigurationInput 구조](#)
- [OAuth2Properties 구조](#)
- [OAuth2PropertiesInput 구조](#)
- [OAuth2ClientApplication 구조](#)
- [AuthorizationCodeProperties 구조](#)

AuthenticationConfiguration 구조

인증 구성을 포함하는 구조입니다.

필드

- `AuthenticationType` – UTF-8 문자열입니다(유효한 값: BASIC | OAUTH2 | CUSTOM).

인증 구성을 포함하는 구조입니다.

- `SecretArn` – [Custom string pattern #11](#)과(와) 일치하는 UTF-8 문자열입니다.

자격 증명을 저장할 보안 관리자 ARN입니다.

- `OAuth2Properties` – [OAuth2Properties](#) 객체입니다.

OAuth2 인증을 위한 속성입니다.

AuthenticationConfigurationInput 구조

CreateConnection 요청에서 인증 구성을 포함하는 구조입니다.

필드

- `AuthenticationType` – UTF-8 문자열입니다(유효한 값: BASIC | OAUTH2 | CUSTOM).

`CreateConnection` 요청에서 인증 구성을 포함하는 구조입니다.

- `SecretArn` – [Custom string pattern #11](#)과(와) 일치하는 UTF-8 문자열입니다.

`CreateConnection` 요청에서 자격 증명을 저장할 보안 관리자 ARN입니다.

- `OAuth2Properties` – [OAuth2PropertiesInput](#) 객체입니다.

`CreateConnection` 요청에서 OAuth2 인증의 속성입니다.

OAuth2Properties 구조

OAuth2 인증을 위한 속성을 포함하는 구조입니다.

필드

- `OAuth2GrantType` – UTF-8 문자열입니다(유효한 값: AUTHORIZATION_CODE | CLIENT_CREDENTIALS | JWT_BEARER).

OAuth2 권한 부여 유형입니다. 예: AUTHORIZATION_CODE, JWT_BEARER 또는 CLIENT_CREDENTIALS.

- `OAuth2ClientApplication` – [OAuth2ClientApplication](#) 객체입니다.

클라이언트 애플리케이션 유형입니다. 예: AWS_MANAGED 또는 USER_MANAGED.

- `TokenUrl` – [Custom string pattern #12](#)과(와) 일치하는 256바이트 이하 길이의 UTF-8 문자열입니다.

인증 코드를 액세스 토큰으로 교환하기 위한 제공업체 인증 서버 URL입니다.

- `TokenUrlParametersMap` – 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 길이가 1~512바이트인 UTF-8 문자열입니다.

토큰 GET 요청에 추가된 파라미터의 맵입니다.

OAuth2PropertiesInput 구조

CreateConnection 요청에서 OAuth2를 위한 속성을 포함하는 구조입니다.

필드

- OAuth2GrantType – UTF-8 문자열입니다(유효한 값: AUTHORIZATION_CODE | CLIENT_CREDENTIALS | JWT_BEARER).

CreateConnection 요청에서 OAuth2 권한 부여 유형입니다. 예: AUTHORIZATION_CODE, JWT_BEARER 또는 CLIENT_CREDENTIALS.

- OAuth2ClientApplication – [OAuth2ClientApplication](#) 객체입니다.

CreateConnection 요청에서 클라이언트 애플리케이션 유형입니다. 예: AWS_MANAGED 또는 USER_MANAGED.

- TokenUrl – [Custom string pattern #12](#)과(와) 일치하는 256바이트 이하 길이의 UTF-8 문자열입니다.

인증 코드를 액세스 토큰으로 교환하기 위한 제공업체 인증 서버 URL입니다.

- TokenUrlParametersMap – 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 길이가 1~512바이트인 UTF-8 문자열입니다.

토큰 GET 요청에 추가된 파라미터의 맵입니다.

- AuthorizationCodeProperties – [AuthorizationCodeProperties](#) 객체입니다.

OAuth2 AUTHORIZATION_CODE 권한 부여 유형에 필요한 속성 세트입니다.

OAuth2ClientApplication 구조

연결에 사용된 OAuth2 클라이언트 앱입니다.

필드

- UserManagedClientApplicationClientId – [Custom string pattern #13](#)과(와) 일치하는 2,048 바이트 이하 길이의 UTF-8 문자열입니다.

ClientAppType이 USER_MANAGED인 경우 클라이언트 애플리케이션의 clientID입니다.

- `AWSTManagedClientApplicationReference` – [Custom string pattern #13](#)과(와) 일치하는 2,048 바이트 이하 길이의 UTF-8 문자열입니다.

AWS 관리형인 SaaS 측 클라이언트 앱에 대한 참조입니다.

AuthorizationCodeProperties 구조

OAuth2 AUTHORIZATION_CODE 권한 부여 유형 워크플로에 필요한 속성 세트입니다.

필드

- `AuthorizationCode` – [Custom string pattern #13](#)과(와) 일치하는 1~4096바이트 길이의 UTF-8 문자열입니다.

AUTHORIZATION_CODE 권한 부여 워크플로의 세 번째 레그에서 사용할 인증 코드입니다. 이 코드는 액세스 토큰으로 교환되면 유효하지 않게 되는 일회용 코드이므로 이 값을 요청 파라미터로 사용하는 것이 허용됩니다.

- `RedirectUri` – [Custom string pattern #14](#)과(와) 일치하는 512바이트 이하 길이의 UTF-8 문자열입니다.

인증 코드를 발급할 때 권한 부여 서버가 사용자를 리디렉션하는 리디렉션 URI입니다. 계속해서 인증 코드를 액세스 토큰으로 교환할 때 이 URI가 사용됩니다.

사용자 정의 함수 API

사용자 지정 함수 API는 함수로 작업할 때 사용되는 AWS Glue 데이터 유형 및 작업에 대해 설명합니다.

데이터 타입

- [UserDefinedFunction 구조](#)
- [UserDefinedFunctionInput 구조](#)

UserDefinedFunction 구조

Hive 사용자 정의 함수(UDF) 정의와 동등한 수준을 보여줍니다.

필드

- **FunctionName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수의 이름입니다.

- **DatabaseName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수를 포함하는 카탈로그 데이터베이스의 이름입니다.

- **ClassName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수 코드가 포함된 Java 클래스입니다.

- **OwnerName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수의 소유자입니다.

- **OwnerType** – UTF-8 문자열입니다(유효 값: USER | ROLE | GROUP).

소유자 유형입니다.

- **CreateTime** – 타임스탬프입니다.

함수가 생성된 시간.

- **ResourceUri** – [ResourceUri](#) 객체의 배열이며 구조는 1,000개 이하입니다.

함수 리소스 URI

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

함수가 있는 Data Catalog의 ID입니다.

UserDefinedFunctionInput 구조

사용자 정의 함수를 생성하거나 업데이트할 때 사용되는 구조입니다.

필드

- **FunctionName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수의 이름입니다.

- `ClassName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수 코드가 포함된 Java 클래스입니다.

- `OwnerName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수의 소유자입니다.

- `OwnerType` – UTF-8 문자열입니다(유효 값: USER | ROLE | GROUP).

소유자 유형입니다.

- `ResourceUris` – [ResourceUri](#) 객체의 배열이며 구조는 1,000개 이하입니다.

함수 리소스 URI

운영

- [CreateUserDefinedFunction](#) 작업(Python: `create_user_defined_function`)
- [UpdateUserDefinedFunction](#) 작업 (Python: `update_user_defined_function`)
- [DeleteUserDefinedFunction](#) 작업 (Python: `delete_user_defined_function`)
- [GetUserDefinedFunction](#) 작업 (Python: `get_user_defined_function`)
- [GetUserDefinedFunctions](#) 작업 (Python: `get_user_defined_functions`)

CreateUserDefinedFunction 작업(Python: `create_user_defined_function`)

데이터 카탈로그에서 새로운 함수 정의를 생성합니다.

요청

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

함수를 생성할 데이터 카탈로그의 ID입니다. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수가 생성되는 카탈로그 데이터베이스의 이름입니다.

- FunctionInput – 필수(Required): [UserDefinedFunctionInput](#) 객체입니다.

FunctionInput 객체는 데이터 카탈로그에 생성할 함수를 정의합니다.

응답

- 무응답 파라미터.

Errors

- AlreadyExistsException
- InvalidInputException
- InternalServiceException
- EntityNotFoundException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- GlueEncryptionException

UpdateUserDefinedFunction 작업 (Python: update_user_defined_function)

데이터 카탈로그에서 기존 함수를 업데이트합니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

업데이트되는 함수가 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트되는 함수가 존재하는 카탈로그 데이터베이스 이름

- FunctionName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수의 이름입니다.

- FunctionInput – 필수(Required): [UserDefinedFunctionInput](#) 객체입니다.

FunctionInput 객체는 데이터 카탈로그의 함수를 재정의합니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

DeleteUserDefinedFunction 작업 (Python: delete_user_defined_function)

데이터 카탈로그에서 기존 함수를 삭제합니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

삭제되는 함수가 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수가 존재하는 카탈로그 데이터베이스 이름

- FunctionName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제된 함수 정의 이름입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetUserDefinedFunction 작업 (Python: get_user_defined_function)

데이터 카탈로그에서 지정된 함수를 가져옵니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

가져올 함수가 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용 됩니다.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수가 존재하는 카탈로그 데이터베이스 이름

- FunctionName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수의 이름입니다.

응답

- UserDefinedFunction – [UserDefinedFunction](#) 객체입니다.

요청한 함수 정의입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- GlueEncryptionException

GetUserDefinedFunctions 작업 (Python: get_user_defined_functions)

데이터 카탈로그에서 다양한 함수 정의를 가져옵니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

가져올 함수가 존재하는 데이터 카탈로그 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- DatabaseName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

함수가 존재하는 카탈로그 데이터베이스 이름 아무 것도 제공되지 않으면 카탈로그에 있는 모든 데이터베이스의 함수가 반환됩니다.

- Pattern – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

반환된 함수 정의를 필터링하는 조건부 함수 이름 패턴 문자열입니다.

- NextToken – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

- MaxResults – 1~100의 숫자(정수)입니다.

한 번의 응답으로 반환될 최대 함수 수입니다.

응답

- UserDefinedFunctions – [UserDefinedFunction](#) 객체의 배열입니다.

요청한 함수 정의의 목록입니다.

- NextToken – UTF-8 문자열입니다.

반환된 함수 목록이 마지막으로 요청된 함수가 아닌 경우의 연속 토큰입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- GlueEncryptionException

Athena 카탈로그를 AWS Glue로 가져오기

마이그레이션 API는 Athena 데이터 카탈로그를 AWS Glue로 마이그레이션하는 것과 관련된 AWS Glue 데이터 유형 및 작업에 대해 설명합니다.

데이터 타입

- [CatalogImportStatus 구조](#)

CatalogImportStatus 구조

이송 상태 정보를 포함한 구조.

필드

- ImportCompleted – 부울입니다.

마이그레이션이 완료되면 True이고 그렇지 않으면 False입니다.

- ImportTime – 타임스탬프입니다.

이송이 시작된 시간.

- ImportedBy – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이송을 시작한 사람의 이름입니다.

운영

- [ImportCatalogToGlue 작업\(Python: import_catalog_to_glue\)](#)
- [GetCatalogImportStatus 작업\(Python: get_catalog_import_status\)](#)

ImportCatalogToGlue 작업(Python: import_catalog_to_glue)

기존 Amazon Athena Data Catalog를 AWS Glue로 가져옵니다.

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

가져올 객체의 ID입니다. 현재는 AWS 계정 ID여야 합니다.

응답

- 무응답 파라미터.

Errors

- `InternalServiceException`
- `OperationTimeoutException`

GetCatalogImportStatus 작업(Python: get_catalog_import_status)

이송 작업 상태 검색

요청

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

이송할 카탈로그의 ID입니다. 현재는 AWS 계정 ID여야 합니다.

응답

- ImportStatus – [CatalogImportStatus](#) 객체입니다.

지정된 카탈로그 이송 상태.

Errors

- InternalServiceException
- OperationTimeoutException

테이블 옵티마이저 API

테이블 옵티마이저 API는 컴팩션을 활성화하여 읽기 성능을 향상시키는 AWS Glue API를 설명합니다.

데이터 타입

- [TableOptimizer 구조](#)
- [TableOptimizerConfiguration 구조](#)
- [TableOptimizerRun 구조](#)
- [RunMetrics 구조](#)
- [BatchGetTableOptimizerEntry 구조](#)
- [BatchTableOptimizer 구조](#)
- [BatchGetTableOptimizerError 구조](#)

TableOptimizer 구조

테이블과 관련된 옵티마이저에 대한 세부 정보가 들어 있습니다.

필드

- type – UTF-8 문자열입니다(유효한 값: compaction="COMPACTION").

테이블 옵티마이저 유형. 현재 유일한 유효 값은 compaction입니다.

- configuration – [TableOptimizerConfiguration](#) 객체입니다.

테이블 옵티마이저를 만들거나 업데이트할 때 지정된 `TableOptimizerConfiguration` 객체입니다.

- `lastRun` – [TableOptimizerRun](#) 객체입니다.

테이블 옵티마이저의 마지막 실행을 나타내는 `TableOptimizerRun` 객체입니다.

TableOptimizerConfiguration 구조

테이블 옵티마이저의 구성에 대한 세부 정보가 들어 있습니다. 테이블 옵티마이저를 만들거나 업데이트할 때 이 구성을 전달합니다.

필드

- `roleArn` – [Single-line string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.
호출자가 전달하는 역할로, 호출자를 대신하여 최적기와 관련된 리소스를 업데이트할 수 있는 권한을 서비스에 부여합니다.
- `enabled` – 부울입니다.
테이블 최적화가 활성화되었는지 여부.

TableOptimizerRun 구조

테이블 옵티마이저 실행에 대한 세부 정보가 들어 있습니다.

필드

- `eventType` – UTF-8 문자열입니다(유효한 값: `starting="STARTING" | completed="COMPLETED" | failed="FAILED" | in_progress="IN_PROGRESS"`).
테이블 옵티마이저 실행 상태를 나타내는 이벤트 유형입니다.
- `startTimestamp` – 타임스탬프입니다.
Lake Formation 내에서 압축 작업이 시작된 시점의 에포크 타임스탬프를 나타냅니다.
- `endTimestamp` – 타임스탬프입니다.
컴팩션 작업이 종료된 에포크 타임스탬프를 나타냅니다.
- `metrics` – [RunMetrics](#) 객체입니다.

옵티마이저 실행에 대한 메트릭이 포함된 RunMetrics 객체입니다.

- error – UTF-8 문자열입니다.

옵티마이저 실행 중에 발생한 오류입니다.

RunMetrics 구조

옵티마이저 실행에 대한 지표.

필드

- NumberOfBytesCompacted – UTF-8 문자열입니다.

압축 작업 실행으로 제거된 바이트 수입니다.

- NumberOfFilesCompacted – UTF-8 문자열입니다.

압축 작업 실행으로 제거된 파일 수입니다.

- NumberOfDpus – UTF-8 문자열입니다.

작업에 사용된 DPU 시간.

- JobDurationInHour – UTF-8 문자열입니다.

작업 기간(시간).

BatchGetTableOptimizerEntry 구조

BatchGetTableOptimizer 작업에서 검색할 테이블 옵티마이저를 나타냅니다.

필드

- catalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- databaseName – UTF-8 문자열입니다(최소 1바이트).

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- tableName – UTF-8 문자열입니다(최소 1바이트).

테이블의 이름

- type – UTF-8 문자열입니다(유효한 값: compaction="COMPACTION").

테이블 옵티마이저 유형.

BatchTableOptimizer 구조

BatchGetTableOptimizer 작업에서 반환된 테이블 옵티마이저 중 하나에 대한 세부 정보가 들어 있습니다.

필드

- catalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- databaseName – UTF-8 문자열입니다(최소 1바이트).

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- tableName – UTF-8 문자열입니다(최소 1바이트).

테이블의 이름

- tableOptimizer – [TableOptimizer](#) 객체입니다.

테이블 옵티마이저의 구성 및 마지막 실행에 대한 세부 정보가 포함된 TableOptimizer 객체입니다.

BatchGetTableOptimizerError 구조

BatchGetTableOptimizer 작업에서 반환된 오류 목록의 오류 중 하나에 대한 세부 정보가 들어 있습니다.

필드

- error – [ErrorDetail](#) 객체입니다.

오류에 대한 코드 및 메시지 세부 정보가 포함된 ErrorDetail 객체입니다.

- `catalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- `databaseName` – UTF-8 문자열입니다(최소 1바이트).

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- `tableName` – UTF-8 문자열입니다(최소 1바이트).

테이블의 이름

- `type` – UTF-8 문자열입니다(유효한 값: `compaction="COMPACTION"`).

테이블 옵티마이저 유형.

운영

- [GetTableOptimizer](#) 액션 (파이썬: `get_table_optimizer`)
- [BatchGetTableOptimizer](#) 액션 (Python: `batch_get_table_optimizer`)
- [ListTableOptimizerRuns](#) 액션 (Python: `목록_테이블_옵티마이저_실행`)
- [CreateTableOptimizer](#) 액션 (Python: `생성_테이블_옵티마이저`)
- [DeleteTableOptimizer](#) 액션 (Python: `삭제_테이블_옵티마이저`)
- [UpdateTableOptimizer](#) 액션 (Python: `업데이트_테이블_옵티마이저`)

GetTableOptimizer 액션 (파이썬: `get_table_optimizer`)

지정된 테이블과 관련된 모든 옵티마이저의 구성을 반환합니다.

요청

- `CatalogId` – 필수: [Single-line string pattern](#)과 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

- `Type` – 필수: UTF-8 문자열입니다(유효한 값: `compaction="COMPACTION"`).

테이블 옵티마이저 유형.

응답

- `CatalogId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- `DatabaseName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- `TableName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

- `TableOptimizer` – [TableOptimizer](#) 객체입니다.

지정된 테이블에 연결된 옵티마이저입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `AccessDeniedException`
- `InternalServiceException`

BatchGetTableOptimizer 액션 (Python: `batch_get_table_optimizer`)

지정된 테이블 옵티마이저의 구성을 반환합니다.

요청

- Entries – 필수: [BatchGetTableOptimizerEntry](#) 객체의 배열입니다.

검색할 테이블 옵티마이저를 지정하는 BatchGetTableOptimizerEntry 객체 목록입니다.

응답

- TableOptimizers – [BatchTableOptimizer](#) 객체의 배열입니다.

BatchTableOptimizer 객체의 목록.

- Failures – [BatchGetTableOptimizerError](#) 객체의 배열입니다.

작업의 오류 목록.

Errors

- InternalServiceException

ListTableOptimizerRuns 액션 (Python: 목록_테이블_옵티마이저_실행)

특정 테이블에 대한 이전 옵티마이저 실행 기록을 나열합니다.

요청

- CatalogId – 필수: [Single-line string pattern](#)과 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

- Type – 필수: UTF-8 문자열입니다(유효한 값: compaction="COMPACTION").

테이블 옵티마이저 유형. 현재 유일한 유효 값은 compaction입니다.

- MaxResults - 숫자(정수)입니다.

각 호출에서 반환되는 옵티마이저 실행의 최대 수입니다.

- NextToken - UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- CatalogId - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- DatabaseName - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- TableName - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

- NextToken - UTF-8 문자열입니다.

목록의 현재 세그먼트가 마지막이 아니면 반환된 옵티마이저 실행 목록에 페이지를 매기는 지속적인 토큰은 반환됩니다.

- TableOptimizerRuns - [TableOptimizerRun](#) 객체의 배열입니다.

테이블에 연결된 옵티마이저 실행 목록입니다.

Errors

- EntityNotFoundException
- AccessDeniedException
- InvalidInputException
- InternalServiceException

CreateTableOptimizer 액션 (Python: 생성_테이블_옵티마이저)

특정 함수를 위한 새 테이블 옵티마이저를 만듭니다. `compaction`은 현재 지원되는 유일한 옵티마이저 유형입니다.

요청

- `CatalogId` – 필수: [Single-line string pattern](#)과 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- `DatabaseName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

- `Type` – 필수: UTF-8 문자열입니다(유효한 값: `compaction="COMPACTION"`).

테이블 옵티마이저 유형. 현재 유일한 유효 값은 `compaction`입니다.

- `TableOptimizerConfiguration` – 필수: [TableOptimizerConfiguration](#) 객체입니다.

테이블 옵티마이저의 구성을 나타내는 `TableOptimizerConfiguration` 객체입니다.

응답

- 무응답 파라미터.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `AccessDeniedException`
- `AlreadyExistsException`
- `InternalServiceException`

DeleteTableOptimizer 액션 (Python: 삭제_테이블_옵티마이저)

테이블의 옵티마이저 및 모든 관련 메타데이터를 삭제합니다. 최적화는 더 이상 테이블에서 수행되지 않습니다.

요청

- CatalogId – 필수: [Single-line string pattern](#)과 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

- Type – 필수: UTF-8 문자열입니다(유효한 값: compaction="COMPACTION").

테이블 옵티마이저 유형.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- InternalServiceException

UpdateTableOptimizer 액션 (Python: 업데이트_테이블_옵티마이저)

기존 테이블 옵티마이저의 구성을 업데이트합니다.

요청

- CatalogId – 필수: [Single-line string pattern](#)과 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블의 카탈로그 ID.

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 카탈로그의 데이터베이스 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

- Type – 필수: UTF-8 문자열입니다(유효한 값: compaction="COMPACTION").

테이블 옵티마이저 유형. 현재 유일한 유효 값은 compaction입니다.

- TableOptimizerConfiguration – 필수: [TableOptimizerConfiguration](#) 객체입니다.

테이블 옵티마이저의 구성을 나타내는 TableOptimizerConfiguration 객체입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- AccessDeniedException
- InternalServiceException

크롤러 및 분류자 API

크롤러 및 분류자 API는 AWS Glue 크롤러 및 분류자 데이터 유형에 대해 설명하며 크롤러 또는 분류자를 생성, 삭제, 업데이트 및 나열하기 위한 API를 포함합니다.

주제

- [분류자 API](#)
- [크롤러 API](#)
- [열 통계의 API](#)
- [크롤러 스케줄러 API](#)

분류자 API

분류자 API는 AWS Glue 분류자 데이터 유형에 대해 설명하며 분류자를 생성, 삭제, 업데이트 및 나열하기 위한 API를 포함합니다.

데이터 타입

- [분류자 구조](#)
- [GrokClassifier 구조](#)
- [XMLClassifier 구조](#)
- [JsonClassifier 구조](#)
- [CsvClassifier 구조](#)
- [CreateGrokClassifierRequest 구조](#)
- [UpdateGrokClassifierRequest 구조](#)
- [CreateXMLClassifierRequest 구조](#)
- [UpdateXMLClassifierRequest 구조](#)
- [CreateJsonClassifierRequest 구조](#)
- [UpdateJsonClassifierRequest 구조](#)
- [CreateCsvClassifierRequest 구조](#)
- [UpdateCsvClassifierRequest 구조](#)

분류자 구조

분류자는 크롤링 작업 중에 시작됩니다. 분류자는 지정된 파일이 처리할 수 있는 형식인지 여부를 확인합니다. 처리할 수 있는 형식인 경우, 분류자는 해당 데이터 형식과 일치하는 StructType 객체의 형태로 스키마를 생성합니다.

AWS Glue가 제공하는 스탠더드 분류자를 사용하거나 자체 분류자를 작성하여 데이터 원본을 분류하고 분류자를 고려해 사용할 적절한 스키마를 지정합니다. 분류자는 grok 분류자이거나, XML 분류자이거나, JSON 분류자이거나, Classifier 객체의 필드 중 하나에 지정된 사용자 지정 CSV 분류자일 수 있습니다.

필드

- GrokClassifier – [GrokClassifier](#) 객체입니다.

grok을 사용하는 분류자입니다.

- XMLClassifier – [XMLClassifier](#) 객체입니다.

XML 콘텐츠의 분류자입니다.

- JsonClassifier – [JsonClassifier](#) 객체입니다.

JSON 콘텐츠의 분류자입니다.

- CsvClassifier – [CsvClassifier](#) 객체입니다.

쉼표로 구분된 값(CSV)의 분류자입니다.

GrokClassifier 구조

grok 패턴을 사용하는 분류자입니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- Classification – 필수(Required): UTF-8 문자열입니다.

Twitter, JSON, Omniture 로그, 등과 같이 분류자가 일치하는 데이터 양식의 분류자입니다.

- CreationTime – 타임스탬프입니다.

이 분류자가 등록된 시간입니다.

- LastUpdated – 타임스탬프입니다.

이 분류자가 마지막으로 업데이트된 시간입니다.

- **Version** - 숫자(정수)입니다.

이 분류자 버전

- **GrokPattern** – 필수(Required): [A Logstash Grok string pattern](#)과(와) 일치하는 1~2,048바이트 길이의 UTF-8 문자열입니다.

이 분류자에 의해 grok 패턴이 데이터 스토어로 적용됩니다. 자세한 내용은 [사용자 지정 분류자 작성](#)에서 기본 설정 패턴을 참조하십시오.

- **CustomPatterns** – [URI address multi-line string pattern](#)과(와) 일치하는 16,000바이트 이하 길이의 UTF-8 문자열입니다.

이 분류자가 정의한 조건부 사용자 지정 grok 패턴입니다. 자세한 내용은 [사용자 지정 분류자 작성](#)에서 사용자 지정 패턴을 참조하십시오.

XMLClassifier 구조

XML 내용의 분류자

필드

- **Name** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- **Classification** – 필수(Required): UTF-8 문자열입니다.

분류자가 일치하는 데이터 양식의 분류자입니다.

- **CreationTime** – 타임스탬프입니다.

이 분류자가 등록된 시간입니다.

- **LastUpdated** – 타임스탬프입니다.

이 분류자가 마지막으로 업데이트된 시간입니다.

- **Version** - 숫자(정수)입니다.

이 분류자 버전

- **RowTag** – UTF-8 문자열입니다.

각 기록을 구문 분석된 XML 문서에 포함하는 요소를 설계하는 XML 태그입니다. 이 태그는 자기 닫기 요소(</>에 의해 닫힌 요소)를 식별할 수 없습니다. 요소가 닫는 태그를 통해 종료되는 한 구문 분석된 속성만 포함하는 빈 행 요소입니다 (예를 들어, <row item_a="A" item_b="B"></row>는 괜찮지만 <row item_a="A" item_b="B" />는 괜찮지 않습니다).

JsonClassifier 구조

JSON 내용의 분류자

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- CreationTime – 타임스탬프입니다.

이 분류자가 등록된 시간입니다.

- LastUpdated – 타임스탬프입니다.

이 분류자가 마지막으로 업데이트된 시간입니다.

- Version - 숫자(정수)입니다.

이 분류자 버전

- JsonPath – 필수(Required): UTF-8 문자열입니다.

분류자가 분류할 JSON 데이터를 정의하는 JsonPath 문자열입니다. AWS Glue는 [Writing JsonPath 사용자 정의 분류자 작성](#)에 설명된 대로 JsonPath의 하위 집합을 지원합니다.

CsvClassifier 구조

사용자 지정 CSV 콘텐츠의 분류자입니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- `CreationTime` - 타임스탬프입니다.

이 분류자가 등록된 시간입니다.

- `LastUpdated` - 타임스탬프입니다.

이 분류자가 마지막으로 업데이트된 시간입니다.

- `Version` - 숫자(정수)입니다.

이 분류자 버전

- `Delimiter` - [Custom string pattern #10](#)과(와) 일치하는 1~1바이트 길이의 UTF-8 문자열입니다.

행의 열 입력 항목 각각을 구분하는 것을 나타내기 위한 사용자 지정 기호입니다.

- `QuoteSymbol` - [Custom string pattern #10](#)과(와) 일치하는 1~1바이트 길이의 UTF-8 문자열입니다.

단일 열 값에 내용을 결합하는 것을 나타내기 위한 사용자 지정 기호입니다. 열 구분 기호와 달라야 합니다.

- `ContainsHeader` - UTF-8 문자열입니다(유효 값: UNKNOWN | PRESENT | ABSENT).

CSV 파일에 헤더가 포함되어 있는지 여부를 나타냅니다.

- `Header` - UTF-8 문자열의 배열입니다.

열 이름을 나타내는 문자열 목록입니다.

- `DisableValueTrimming` - 부울입니다.

열 값의 유형을 식별하기 전에 값을 트리밍하지 않도록 지정합니다. 기본 값은 `true`입니다.

- `AllowSingleColumn` - 부울입니다.

오직 하나의 열만 포함하는 파일을 처리할 수 있도록 합니다.

- `CustomDatatypeConfigured` - 부울입니다.

사용자 지정 데이터 유형을 구성할 수 있습니다.

- `CustomDatatypes` - UTF-8 문자열의 배열입니다.

사용자 지정 데이터 유형 목록에는 “바이너리”, “부울”, “날짜”, “십진수”, “더블”, “플로트”, “INT”, “롱”, “쇼트”, “문자열”, “타임스탬프” 등이 포함됩니다.

- Serde – UTF-8 문자열입니다(유효한 값: OpenCSVSerde | LazySimpleSerde | None).

분류자에서 CSV를 처리하기 위한 Serde를 설정합니다. 이는 데이터 카탈로그에서 적용됩니다. 유효한 값은 OpenCSVSerde, LazySimpleSerde, None입니다. 크롤러에서 감지하려는 경우 None 값을 지정할 수 있습니다.

CreateGrokClassifierRequest 구조

생성할 CreateClassifier를 위해 grok 분류자를 지정합니다.

필드

- Classification – 필수(Required): UTF-8 문자열입니다.

Twitter, JSON, Omniture Logs, Amazon CloudWatch Logs 등과 같이 분류자가 일치하는 데이터 양식의 분류자입니다.

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

새로운 분류자의 이름입니다.

- GrokPattern – 필수(Required): [A Logstash Grok string pattern](#)과(와) 일치하는 1~2,048바이트 길이의 UTF-8 문자열입니다.

이 분류자가 사용하는 grok 패턴입니다.

- CustomPatterns – [URI address multi-line string pattern](#)과(와) 일치하는 16,000바이트 이하 길이의 UTF-8 문자열입니다.

이 분류자가 사용하는 조건부 사용자 지정 grok 패턴입니다.

UpdateGrokClassifierRequest 구조

UpdateClassifier로 전달되면 업데이트될 grok 분류자를 지정합니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

GrokClassifier의 이름입니다.

- **Classification** – UTF-8 문자열입니다.

Twitter, JSON, Omniture Logs, Amazon CloudWatch Logs 등과 같이 분류자가 일치하는 데이터 양식의 분류자입니다.

- **GrokPattern** – [A Logstash Grok string pattern](#)과(와) 일치하는 1~2,048바이트 길이의 UTF-8 문자열입니다.

이 분류자가 사용하는 grok 패턴입니다.

- **CustomPatterns** – [URI address multi-line string pattern](#)과(와) 일치하는 16,000바이트 이하 길이의 UTF-8 문자열입니다.

이 분류자가 사용하는 조건부 사용자 지정 grok 패턴입니다.

CreateXMLClassifierRequest 구조

생성할 CreateClassifier를 위해 XML 분류자를 지정합니다.

필드

- **Classification** – 필수(Required): UTF-8 문자열입니다.

분류자가 일치하는 데이터 양식의 분류자입니다.

- **Name** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- **RowTag** – UTF-8 문자열입니다.

각 기록을 구문 분석된 XML 문서에 포함하는 요소를 설계하는 XML 태그입니다. 이 태그는 자기 닫기 요소(</>에 의해 닫힌 요소)를 식별할 수 없습니다. 요소가 닫는 태그를 통해 종료되는 한 구문 분석된 속성만 포함하는 빈 행 요소입니다 (예를 들어, <row item_a="A" item_b="B"></row>는 괜찮지만 <row item_a="A" item_b="B" />는 괜찮지 않습니다).

UpdateXMLClassifierRequest 구조

업데이트될 XML 분류자를 지정합니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- Classification – UTF-8 문자열입니다.

분류자가 일치하는 데이터 양식의 분류자입니다.

- RowTag – UTF-8 문자열입니다.

각 기록을 구문 분석된 XML 문서에 포함하는 요소를 설계하는 XML 태그입니다. 이 태그는 자기 닫기 요소(</>에 의해 닫힌 요소)를 식별할 수 없습니다. 요소가 닫는 태그를 통해 종료되는 한 구문 분석된 속성만 포함하는 빈 행 요소입니다 (예를 들어, <row item_a="A" item_b="B"></row>는 괜찮지만 <row item_a="A" item_b="B" />는 괜찮지 않습니다).

CreateJsonClassifierRequest 구조

생성할 CreateClassifier를 위해 JSON 분류자를 지정합니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- JsonPath – 필수(Required): UTF-8 문자열입니다.

분류자가 분류할 JSON 데이터를 정의하는 JsonPath 문자열입니다. AWS Glue는 [Writing JsonPath 사용자 정의 분류자 작성](#)에 설명된 대로 JsonPath의 하위 집합을 지원합니다.

UpdateJsonClassifierRequest 구조

업데이트될 JSON 분류자를 지정합니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- JsonPath – UTF-8 문자열입니다.

분류자가 분류할 JSON 데이터를 정의하는 JsonPath 문자열입니다. AWS Glue는 [Writing JsonPath 사용자 정의 분류자 작성](#)에 설명된 대로 JsonPath의 하위 집합을 지원합니다.

CreateCsvClassifierRequest 구조

생성할 CreateClassifier를 위해 사용자 지정 CSV 분류자를 지정합니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- Delimiter – [Custom string pattern #10](#)과(와) 일치하는 1~1바이트 길이의 UTF-8 문자열입니다.

행의 열 입력 항목 각각을 구분하는 것을 나타내기 위한 사용자 지정 기호입니다.

- QuoteSymbol – [Custom string pattern #10](#)과(와) 일치하는 1~1바이트 길이의 UTF-8 문자열입니다.

단일 열 값에 내용을 결합하는 것을 나타내기 위한 사용자 지정 기호입니다. 열 구분 기호와 달라야 합니다.

- ContainsHeader – UTF-8 문자열입니다(유효 값: UNKNOWN | PRESENT | ABSENT).

CSV 파일에 헤더가 포함되어 있는지 여부를 나타냅니다.

- Header – UTF-8 문자열의 배열입니다.

열 이름을 나타내는 문자열 목록입니다.

- DisableValueTrimming – 부울입니다.

열 값의 유형을 식별하기 전에 값을 트리밍하지 않도록 지정합니다. 기본값은 true입니다.

- AllowSingleColumn – 부울입니다.

오직 하나의 열만 포함하는 파일을 처리할 수 있도록 합니다.

- CustomDatatypeConfigured – 부울입니다.

사용자 지정 데이터 유형을 구성할 수 있습니다.

- CustomDatatypes – UTF-8 문자열의 배열입니다.

지원되는 사용자 지정 데이터 유형 목록을 만듭니다.

- Serde – UTF-8 문자열입니다(유효한 값: OpenCSVSerDe | LazySimpleSerDe | None).

분류자에서 CSV를 처리하기 위한 Serde를 설정합니다. 이는 데이터 카탈로그에서 적용됩니다. 유효한 값은 OpenCSVSerDe, LazySimpleSerDe, None입니다. 크롤러에서 감지하려는 경우 None 값을 지정할 수 있습니다.

UpdateCsvClassifierRequest 구조

업데이트될 사용자 지정 CSV 분류자를 지정합니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

분류자의 이름입니다.

- Delimiter – [Custom string pattern #10](#)과(와) 일치하는 1~1바이트 길이의 UTF-8 문자열입니다.

행의 열 입력 항목 각각을 구분하는 것을 나타내기 위한 사용자 지정 기호입니다.

- QuoteSymbol – [Custom string pattern #10](#)과(와) 일치하는 1~1바이트 길이의 UTF-8 문자열입니다.

단일 열 값에 내용을 결합하는 것을 나타내기 위한 사용자 지정 기호입니다. 열 구분 기호와 달라야 합니다.

- ContainsHeader – UTF-8 문자열입니다(유효 값: UNKNOWN | PRESENT | ABSENT).

CSV 파일에 헤더가 포함되어 있는지 여부를 나타냅니다.

- Header – UTF-8 문자열의 배열입니다.

열 이름을 나타내는 문자열 목록입니다.

- DisableValueTrimming – 부울입니다.

열 값의 유형을 식별하기 전에 값을 트리밍하지 않도록 지정합니다. 기본값은 true입니다.

- AllowSingleColumn – 부울입니다.

오직 하나의 열만 포함하는 파일을 처리할 수 있도록 합니다.

- CustomDatatypeConfigured – 부울입니다.

사용자 지정 데이터 유형의 구성을 지정합니다.

- CustomDatatypes – UTF-8 문자열의 배열입니다.

지원되는 사용자 지정 데이터 유형 목록을 지정합니다.

- Serde – UTF-8 문자열입니다(유효한 값: OpenCSVSerDe | LazySimpleSerDe | None).

분류자에서 CSV를 처리하기 위한 Serde를 설정합니다. 이는 데이터 카탈로그에서 적용됩니다. 유효한 값은 OpenCSVSerDe, LazySimpleSerDe, None입니다. 크롤러에서 감지하려는 경우 None 값을 지정할 수 있습니다.

운영

- [CreateClassifier 작업\(Python: create_classifier\)](#)
- [DeleteClassifier 작업\(Python: delete_classifier\)](#)
- [GetClassifier 작업\(Python: get_classifier\)](#)
- [GetClassifiers 작업\(Python: get_classifiers\)](#)
- [UpdateClassifier 작업\(Python: update_classifier\)](#)

CreateClassifier 작업(Python: create_classifier)

사용자 계정에 분류자를 만듭니다. 이 분류자는 어떤 요청 필드가 존재하는지에 따라 GrokClassifier, XMLClassifier, JsonClassifier 또는 CsvClassifier일 수 있습니다.

요청

- GrokClassifier – [CreateGrokClassifierRequest](#) 객체입니다.

GrokClassifier 객체는 생성할 분류자를 지정합니다.

- XMLClassifier – [CreateXMLClassifierRequest](#) 객체입니다.

XMLClassifier 객체는 생성할 분류자를 지정합니다.

- JsonClassifier – [CreateJsonClassifierRequest](#) 객체입니다.

JsonClassifier 객체는 생성할 분류자를 지정합니다.

- `CsvClassifier` – [CreateCsvClassifierRequest](#) 객체입니다.

`CsvClassifier` 객체는 생성할 분류자를 지정합니다.

응답

- 무응답 파라미터.

Errors

- `AlreadyExistsException`
- `InvalidInputException`
- `OperationTimeoutException`

DeleteClassifier 작업(Python: `delete_classifier`)

데이터 카탈로그에서 분류자를 제거합니다.

요청

- `Name` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

제거할 분류자의 이름입니다.

응답

- 무응답 파라미터.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`

GetClassifier 작업(Python: `get_classifier`)

이름에 따라 분류자를 검색합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

검색할 분류자의 이름입니다.

응답

- Classifier – [분류자](#) 객체입니다.

요청된 분류자

Errors

- EntityNotFoundException
- OperationTimeoutException

GetClassifiers 작업(Python: get_classifiers)

데이터 카탈로그에서 분류자 객체를 열거합니다.

요청

- MaxResults – 1~1,000의 숫자(정수)입니다.

반환할 목록의 크기(선택 사항)입니다.

- NextToken – UTF-8 문자열입니다.

연속 토큰(선택 사항).

응답

- Classifiers – [분류자](#) 객체의 배열입니다.

분류자 객체의 요청한 목록입니다.

- NextToken – UTF-8 문자열입니다.

연속 토큰

Errors

- `OperationTimeoutException`

UpdateClassifier 작업(Python: `update_classifier`)

기존 분류자(어떤 필드가 존재하는지에 따라 `GrokClassifier`, `XMLClassifier`, `JsonClassifier` 또는 `CsvClassifier` 분류자)를 수정합니다.

요청

- `GrokClassifier` – [UpdateGrokClassifierRequest](#) 객체입니다.

업데이트도니 필드와 `GrokClassifier` 객체.

- `XMLClassifier` – [UpdateXMLClassifierRequest](#) 객체입니다.

업데이트도니 필드와 `XMLClassifier` 객체.

- `JsonClassifier` – [UpdateJsonClassifierRequest](#) 객체입니다.

업데이트도니 필드와 `JsonClassifier` 객체.

- `CsvClassifier` – [UpdateCsvClassifierRequest](#) 객체입니다.

업데이트도니 필드와 `CsvClassifier` 객체.

응답

- 무응답 파라미터.

Errors

- `InvalidInputException`
- `VersionMismatchException`
- `EntityNotFoundException`
- `OperationTimeoutException`

크롤러 API

Crawler API는 AWS Glue 크롤러 생성, 삭제, 업데이트 및 목록 작성을 위한 API와 함께 크롤러 데이터 유형을 설명합니다.

데이터 타입

- [크롤러 구조](#)
- [일정 구조](#)
- [CrawlerTargets 구조](#)
- [S3Target 구조](#)
- [S3 DeltaCatalogTarget 구조](#)
- [S3 DeltaDirectTarget 구조](#)
- [JdbcTarget 구조](#)
- [MongoDBTarget 구조](#)
- [DynamoDBTarget 구조](#)
- [DeltaTarget 구조](#)
- [IcebergTarget 구조](#)
- [HudiTarget 구조](#)
- [CatalogTarget 구조](#)
- [CrawlerMetrics 구조](#)
- [CrawlerHistory 구조](#)
- [CrawlsFilter 구조](#)
- [SchemaChangePolicy 구조](#)
- [LastCrawlInfo 구조](#)
- [RecrawlPolicy 구조](#)
- [LineageConfiguration 구조](#)
- [LakeFormationConfiguration 구조](#)

크롤러 구조

데이터 원본을 검사하는 크롤러 프로그램을 지정하고 분류자를 사용하여 스키마를 결정합니다. 성공적이면 크롤러는 AWS Glue Data Catalog의 데이터 원본을 고려하여 메타데이터를 기록합니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

크롤러의 이름입니다.

- Role – UTF-8 문자열입니다.

Amazon Simple Storage Service(Amazon S3) 데이터 등의 고객 리소스에 액세스하는 데 사용되는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

- Targets – [CrawlerTargets](#) 객체입니다.

크롤할 대상 모음입니다.

- DatabaseName – UTF-8 문자열입니다.

크롤러의 출력이 저장되는 데이터베이스의 이름입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

크롤러에 대한 설명.

- Classifiers – UTF-8 문자열의 배열입니다.

크롤러와 연결된 사용자 지정 분류자를 지정하는 UTF-8 문자열 목록입니다.

- RecrawlPolicy – [RecrawlPolicy](#) 객체입니다.

전체 데이터 집합을 다시 크롤링할지 아니면 마지막 크롤러 실행 이후 추가된 폴더만 크롤링할지 지정하는 정책입니다.

- SchemaChangePolicy – [SchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 및 삭제 동작을 지정하는 정책입니다.

- LineageConfiguration – [LineageConfiguration](#) 객체입니다.

크롤러에 대해 데이터 계보가 사용되는지 여부를 지정하는 구성입니다.

- State – UTF-8 문자열입니다(유효 값: READY | RUNNING | STOPPING).

크롤러가 실행되거나 실행되지 않았는지 여부를 나타냅니다.

- TablePrefix – 128바이트 이하 길이의 UTF-8 문자열입니다.

생성된 테이블 이름에 추가된 접두사.

- Schedule – [일정](#) 객체입니다.

일정이 짜여진 크롤러를 위한 크롤러가 실행될 때의 일정.

- CrawlElapsedTime - 숫자(정수)입니다.

크롤러가 실행되면 마지막 크롤이 시작된 후부터 총 경과 시간.

- CreationTime - 타임스탬프입니다.

크롤러가 생성된 시간.

- LastUpdated - 타임스탬프입니다.

크롤러가 마지막으로 업데이트된 시간.

- LastCrawl – [LastCrawlInfo](#) 객체입니다.

마지막 크롤 상태 및 오류가 발생한 잠재적 오류 정보.

- Version - 숫자(정수)입니다.

크롤러 버전.

- Configuration – UTF-8 문자열입니다.

크롤러 구성 정보. 이 버전의 JSON 문자열은 사용자가 크롤러 동작을 지정할 수 있게 만듭니다. 자세한 내용을 알아보려면 [크롤러 구성 옵션 설정](#)을 참조하세요.

- CrawlerSecurityConfiguration – 128바이트 이하 길이의 UTF-8 문자열입니다.

이 크롤러가 사용할 SecurityConfiguration 구조의 이름입니다.

- LakeFormationConfiguration – [LakeFormationConfiguration](#) 객체입니다.

크롤러가 IAM 역할 자격 증명 대신 크롤러에 대한 AWS Lake Formation 자격 증명을 사용해야 하는지 여부를 지정합니다.

일정 구조

cron을 사용하여 객체의 일정을 정하여 이벤트의 일정을 정합니다.

필드

- ScheduleExpression – UTF-8 문자열입니다.

일정을 지정하는 데 사용되는 cron 표현식입니다([작업 및 크롤러의 시간 기반 일정 참조](#)). 예를 들어, 매일 오후 12시 15분(UTC)에 실행하려면 cron(15 12 * * ? *)을 지정합니다.

- State – UTF-8 문자열입니다(유효 값: SCHEDULED | NOT_SCHEDULED | TRANSITIONING).

일정 상태

CrawlerTargets 구조

크롤할 데이터 스토어 지정.

필드

- S3Targets – [S3Target](#) 객체의 배열입니다.

Amazon Simple Storage Service(Amazon S3) 대상을 지정합니다.

- JdbcTargets – [JdbcTarget](#) 객체의 배열입니다.

JDBC 대상 지정

- MongoDBTargets – [MongoDBTarget](#) 객체의 배열입니다.

Amazon DocumentDB 또는 MongoDB 대상을 지정합니다.

- DynamoDBTargets – [DynamoDBTarget](#) 객체의 배열입니다.

Amazon DynamoDB 대상을 지정합니다.

- CatalogTargets – [CatalogTarget](#) 객체의 배열입니다.

AWS Glue Data Catalog 대상을 지정합니다.

- DeltaTargets – [DeltaTarget](#) 객체의 배열입니다.

델타 데이터 스토어 대상을 지정합니다.

- IcebergTargets – [IcebergTarget](#) 객체의 배열입니다.

Apache Iceberg 데이터 스토어 대상을 지정합니다.

- HudiTargets – [HudiTarget](#) 객체의 배열입니다.

Apache Hudi 데이터 스토어 대상을 지정합니다.

S3Target 구조

Amazon Simple Storage Service(Amazon S3)의 데이터 스토어를 지정합니다.

필드

- Path – UTF-8 문자열입니다.

Amazon S3 대상에 대한 경로입니다.

- Exclusions – UTF-8 문자열의 배열입니다.

크롤링에서 제외하는 데 사용되는 glob 패턴 목록입니다. 자세한 내용은 [크롤러를 사용하여 테이블 분류](#)를 참조하십시오.

- ConnectionName – UTF-8 문자열입니다.

작업 또는 크롤러가 Amazon Virtual Private Cloud 환경(Amazon VPC) 내에서 Amazon S3의 데이터에 액세스할 수 있도록 허용하는 연결 이름입니다.

- SampleSize - 숫자(정수)입니다.

데이터 집합의 샘플 파일을 크롤링할 때 크롤링할 각 리프 폴더의 파일 수를 설정합니다. 설정하지 않으면 모든 파일이 크롤링됩니다. 유효한 값은 1~249의 정수입니다.

- EventQueueArn – UTF-8 문자열입니다.

유효한 Amazon SQS ARN입니다. 예를 들어 `arn:aws:sqs:region:account:sqs`입니다.

- D1qEventQueueArn – UTF-8 문자열입니다.

유효한 Amazon 배달 못한 편지 SQS ARN입니다. 예를 들어 `arn:aws:sqs:region:account:deadLetterQueue`입니다.

S3 DeltaCatalogTarget 구조

데이터 카탈로그의 Delta Lake 데이터 소스에 쓰는 대상을 지정합니다. AWS Glue

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- PartitionKeys – UTF-8 문자열의 배열입니다.

일련의 키를 사용하여 기본 분할을 지정합니다.

- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 데이터베이스 테이블의 이름입니다.

- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 데이터베이스의 이름입니다.

- AdditionalOptions – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터에 대한 추가 연결 옵션을 지정합니다.

- SchemaChangePolicy – [CatalogSchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

S3 DeltaDirectTarget 구조

델타 레이크 데이터 소스에 쓰는 대상을 지정합니다 Amazon S3.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- PartitionKeys – UTF-8 문자열의 배열입니다.

일련의 키를 사용하여 기본 분할을 지정합니다.

- Path – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 Delta Lake 데이터 소스의 Amazon S3 경로입니다.

- **Compression** – 필수: UTF-8 문자열입니다(유효한 값: `uncompressed="UNCOMPRESSED" | snappy="SNAPPY"`).

데이터 압축 방식을 지정합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip"입니다.

- **Format** – 필수: UTF-8 문자열입니다(유효한 값: `json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA"`).

대상에 대한 데이터 출력 포맷을 지정합니다.

- **AdditionalOptions** – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터에 대한 추가 연결 옵션을 지정합니다.

- **SchemaChangePolicy** – [DirectSchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

JdbcTarget 구조

크롤할 JDBC 데이터 스토어 지정.

필드

- **ConnectionName** – UTF-8 문자열입니다.

JDBC 대상에 연결할 연결 이름입니다.

- **Path** – UTF-8 문자열입니다.

JDBC 대상의 경로입니다.

- **Exclusions** – UTF-8 문자열의 배열입니다.

크롤링에서 제외하는 데 사용되는 glob 패턴 목록입니다. 자세한 내용은 [크롤러를 사용하여 테이블 분류](#)를 참조하십시오.

- **EnableAdditionalMetadata** – UTF-8 문자열의 배열입니다.

RAWTYPES 또는 COMMENTS 값을 지정하여 테이블 응답에서 추가 메타데이터를 활성화합니다. RAWTYPES는 기본 레벨 데이터 유형을 제공합니다. COMMENTS는 데이터베이스의 열 또는 테이블과 연결된 설명을 제공합니다.

추가 메타데이터가 필요하지 않은 경우 필드를 비워 두세요.

MongoDBTarget 구조

크롤링할 Amazon DocumentDB 또는 MongoDB 데이터 스토어를 지정합니다.

필드

- ConnectionName – UTF-8 문자열입니다.

Amazon DocumentDB 또는 MongoDB 대상에 연결하는 데 사용할 연결 이름입니다.

- Path – UTF-8 문자열입니다.

Amazon DocumentDB 또는 MongoDB 대상(데이터베이스/컬렉션)의 경로입니다.

- ScanAll – 부울입니다.

모든 레코드를 스캔할지 또는 테이블에서 행을 샘플링할지 여부를 나타냅니다. 테이블이 높은 처리량 테이블이 아닌 경우 모든 레코드를 스캔하는 데 시간이 오래 걸릴 수 있습니다.

true 값은 모든 레코드를 스캔하는 것을 의미하고 false 값은 레코드를 샘플링하는 것을 의미합니다. 값을 지정하지 않으면 기본값은 true입니다.

DynamoDBTarget 구조

크롤링할 Amazon DynamoDB 테이블을 지정합니다.

필드

- Path – UTF-8 문자열입니다.

크롤링할 DynamoDB 테이블의 이름입니다.

- scanAll – 부울입니다.

모든 레코드를 스캔할지 또는 테이블에서 행을 샘플링할지 여부를 나타냅니다. 테이블이 높은 처리량 테이블이 아닌 경우 모든 레코드를 스캔하는 데 시간이 오래 걸릴 수 있습니다.

`true` 값은 모든 레코드를 스캔하는 것을 의미하고 `false` 값은 레코드를 샘플링하는 것을 의미합니다. 값을 지정하지 않으면 기본값은 `true`입니다.

- `scanRate` - 숫자(double)입니다.

AWS Glue 크롤러가 사용하도록 구성된 읽기 용량 단위의 백분율. 읽기 용량 단위는 DynamoDB에서 정의한 용어이며, 초당 해당 테이블에서 수행할 수 있는 읽기 수에 대한 속도 제한기 역할을 하는 숫자 값입니다.

유효한 값은 `null` 또는 0.1~1.5의 값입니다. `null` 값은 사용자가 값을 제공하지 않을 때 사용되며, 기본값은 구성된 읽기 용량 단위의 0.5(프로비저닝된 테이블의 경우) 또는 구성된 최대 읽기 용량 단위의 0.25(온디맨드 모드를 사용하는 테이블의 경우)입니다.

DeltaTarget 구조

하나 이상의 델타 테이블을 크롤링할 델타 데이터 스토어를 지정합니다.

필드

- `DeltaTables` – UTF-8 문자열의 배열입니다.

델타 테이블에 대한 Amazon S3 경로 목록입니다.

- `ConnectionName` – UTF-8 문자열입니다.

델타 테이블 대상에 연결하는 데 사용할 연결 이름입니다.

- `WriteManifest` – 부울입니다.

매니페스트 파일을 델타 테이블 경로에 쓸지 지정합니다.

- `CreateNativeDeltaTable` – 부울입니다.

크롤러가 Delta 트랜잭션 로그의 쿼리를 직접 지원하는 쿼리 엔진과 통합할 수 있도록 기본 테이블을 생성할지 여부를 지정합니다.

IcebergTarget 구조

Amazon S3에서 Iceberg 테이블이 저장되는 Apache Iceberg 데이터 소스를 지정합니다.

필드

- Paths – UTF-8 문자열의 배열입니다.

Iceberg 메타데이터 폴더를 로 `s3://bucket/prefix` 포함하는 하나 이상의 Amazon S3 경로.

- ConnectionName – UTF-8 문자열입니다.

Iceberg 대상에 연결하는 데 사용할 연결 이름입니다.

- Exclusions – UTF-8 문자열의 배열입니다.

크롤링에서 제외하는 데 사용되는 glob 패턴 목록입니다. 자세한 내용은 [크롤러를 사용하여 테이블 분류](#)를 참조하십시오.

- MaximumTraversalDepth - 숫자(정수)입니다.

크롤러가 Amazon S3 경로에 있는 Iceberg 메타데이터 폴더를 발견하기 위해 탐색할 수 있는 최대 경로 깊이입니다. Amazon S3 크롤러 실행 시간을 제한하는 데 사용됩니다.

HudiTarget 구조

Apache Hudi 데이터 소스를 지정합니다.

필드

- Paths – UTF-8 문자열의 배열입니다.

Hudi의 Amazon S3 위치 문자열 배열로, 각 문자열은 Hudi 테이블의 메타데이터 파일이 있는 루트 폴더를 나타냅니다. Hudi 폴더는 루트 폴더의 하위 폴더에 있을 수 있습니다.

크롤러는 경로 아래에 있는 모든 폴더에서 Hudi 폴더를 스캔합니다.

- ConnectionName – UTF-8 문자열입니다.

Hudi 대상에 연결하는 데 사용할 연결 이름입니다. VPC 인증이 필요한 버킷에 Hudi 파일이 저장된 경우 여기에서 연결 속성을 설정할 수 있습니다.

- Exclusions – UTF-8 문자열의 배열입니다.

크롤링에서 제외하는 데 사용되는 glob 패턴 목록입니다. 자세한 내용은 [크롤러를 사용하여 테이블 분류](#)를 참조하십시오.

- MaximumTraversalDepth - 숫자(정수)입니다.

크롤러가 Amazon S3 경로에 있는 Hudi 메타데이터 폴더를 발견하기 위해 탐색할 수 있는 최대 경로 깊이입니다. Amazon S3 크롤러 실행 시간을 제한하는 데 사용됩니다.

CatalogTarget 구조

AWS Glue Data Catalog 대상을 지정합니다.

필드

- DatabaseName – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

동기화할 데이터베이스의 이름입니다.

- Tables – 필수(Required): UTF-8 문자열의 배열이며 문자열은 1개 이상입니다.

동기화할 테이블의 목록입니다.

- ConnectionName – UTF-8 문자열입니다.

NETWORK 연결 유형에 페어링된 Catalog 연결 유형을 사용할 때 Amazon S3 기반 데이터 카탈로그 테이블이 크롤링의 대상이 되도록 하는 연결의 이름입니다.

- EventQueueArn – UTF-8 문자열입니다.

유효한 Amazon SQS ARN입니다. 예를 들어 `arn:aws:sqs:region:account:sqs`입니다.

- D1qEventQueueArn – UTF-8 문자열입니다.

유효한 Amazon 배달 못한 편지 SQS ARN입니다. 예를 들어 `arn:aws:sqs:region:account:deadLetterQueue`입니다.

CrawlerMetrics 구조

지정한 크롤러에 대한 지표.

필드

- CrawlerName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

크롤러의 이름입니다.

- `TimeLeftSeconds` – None 이하의 숫자(실수)입니다.
크롤을 완료할 때까지 남은 예상 시간.
- `StillEstimating` – 부울입니다.
이 작업이 완료될 때까지 소요 시간을 여전히 추정하고 있다면 True입니다.
- `LastRuntimeSeconds` – None 이하의 숫자(실수)입니다.
초 단위의 크롤러의 최근 작업 실행 지속 시간.
- `MedianRuntimeSeconds` – None 이하의 숫자(실수)입니다.
초 단위의 크롤러의 평균 작업 실행 지속 시간.
- `TablesCreated` – None 이하의 숫자(정수)입니다.
이 크롤러가 생성한 테이블 수.
- `TablesUpdated` – None 이하의 숫자(정수)입니다.
이 크롤러가 업데이트한 테이블 수.
- `TablesDeleted` – None 이하의 숫자(정수)입니다.
이 크롤러가 삭제한 테이블 수.

CrawlerHistory 구조

크롤러 실행에 대한 정보가 포함되어 있습니다.

필드

- `CrawlId` – UTF-8 문자열입니다.
각 크롤링에 대한 UUID 식별자입니다.
- `State` – UTF-8 문자열입니다(유효한 값: RUNNING | COMPLETED | FAILED | STOPPED).
크롤의 상태.
- `StartTime` – 타임스탬프입니다.
크롤이 시작된 날짜와 시간입니다.
- `EndTime` – 타임스탬프입니다.

크롤이 시작된 날짜와 시간.

- Summary – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

JSON의 특정 크롤에 대한 실행 요약. 추가, 업데이트 또는 삭제된 카탈로그 테이블 및 파티션을 포함합니다.

- ErrorMessage – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

(오류가 발생한 경우) 크롤과 연결된 오류 메시지.

- LogGroup – [Log group string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.

크롤과 연결된 로그 그룹입니다.

- LogStream – [Log-stream string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.

크롤과 연결된 로그 스트림입니다.

- MessagePrefix – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 크롤링에 대한 CloudWatch 메시지의 접두사입니다.

- DPUHour – None 이하의 숫자(실수)입니다.

크롤에 사용된 DPU(데이터 처리 단위)의 수(시간 단위).

CrawlsFilter 구조

지정된 크롤러에 대한 크롤러 실행을 필터링하는 데 사용할 수 있는 필드, 비교기 및 값의 목록입니다.

필드

- `FieldName` – UTF-8 문자열입니다(유효한 값: `CRAWL_ID` | `STATE` | `START_TIME` | `END_TIME` | `DPU_HOUR`).

지정된 크롤러에 대한 크롤러 실행 필터링에 사용되는 키. 각 필드 이름에 유효한 값은 다음과 같습니다.

- `CRAWL_ID`: 크롤링의 UUID 식별자를 나타내는 문자열.
- `STATE`: 크롤의 상태를 나타내는 문자열.
- `START_TIME` 및 `END_TIME`: 밀리초 단위의 Epoch 타임스탬프.

- `DPU_HOUR`: 크롤에 사용된 DPU(데이터 처리 단위)의 수(시간 단위).
- `FilterOperator` – UTF-8 문자열입니다(유효 값: `GT` | `GE` | `LT` | `LE` | `EQ` | `NE`).

값의 연산을 수행하는 정의된 비교기. 사용 가능한 연산자는 다음과 같습니다.

- `GT`: 큼.
- `GE`: 크거나 같음.
- `LT`: 작음.
- `LE`: 작거나 같음.
- `EQ`: 같음.
- `NE`: 같지 않음.
- `FieldValue` – UTF-8 문자열입니다.

크롤 필드에서 비교하기 위해 제공된 값입니다.

SchemaChangePolicy 구조

크롤러에 대한 업데이트 및 삭제 동작을 지정하는 정책입니다.

필드

- `UpdateBehavior` – UTF-8 문자열입니다(유효 값: `LOG` | `UPDATE_IN_DATABASE`).
- 크롤러가 변화된 객체를 찾을 때 업데이트 동작.
- `DeleteBehavior` – UTF-8 문자열입니다(유효 값: `LOG` | `DELETE_FROM_DATABASE` | `DEPRECATE_IN_DATABASE`).

크롤러가 변화된 혹은 삭제된 객체를 찾을 때 삭제 동작.

LastCrawlInfo 구조

최신 크롤의 상태 및 오류 정보.

필드

- `Status` – UTF-8 문자열입니다(유효한 값: `SUCCEEDED` | `CANCELLED` | `FAILED`).

최종 크롤 상태

- ErrorMessage – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

오류가 발생할 때 마지막 크롤에 대한 오류 정보.

- LogGroup – [Log group string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.

마지막 크롤의 로그 그룹.

- LogStream – [Log-stream string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.

마지막 크롤의 로그 스트림.

- MessagePrefix – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 크롤 메시지에 대한 접두사.

- StartTime – 타임스탬프입니다.

크롤이 시작된 시간.

RecrawlPolicy 구조

첫 번째 크롤링이 완료된 후 Amazon S3 데이터 원본을 크롤링할 때 전체 데이터 집합을 다시 크롤링할지 아니면 마지막 크롤러 실행 이후에 추가된 폴더만 크롤링할지 지정합니다. 자세한 내용은 SageMaker 개발자 안내서의 [AWS Glue의 증분 크롤링](#)을 참조하세요.

필드

- RecrawlBehavior – UTF-8 문자열입니다(유효한 값: CRAWL_EVERYTHING | CRAWL_NEW_FOLDERS_ONLY | CRAWL_EVENT_MODE).

전체 데이터 집합을 다시 크롤링할지 아니면 마지막 크롤러 실행 이후 추가된 폴더만 크롤링할지 지정합니다.

값 CRAWL_EVERYTHING은 전체 데이터 집합을 다시 크롤링하도록 지정합니다.

값 CRAWL_NEW_FOLDERS_ONLY는 마지막 크롤러 실행 이후에 추가된 폴더만 크롤링하도록 지정합니다.

값 CRAWL_EVENT_MODE는 Amazon S3 이벤트에서 식별된 변경 사항만 크롤링하도록 지정합니다.

LineageConfiguration 구조

크롤러에 대한 데이터 계보 구성 설정을 지정합니다.

필드

- `CrawlerLineageSettings` – UTF-8 문자열입니다(유효 값: ENABLE | DISABLE).
크롤러에 데이터 계보가 사용되는지 여부를 지정합니다. 유효한 값은 다음과 같습니다.
 - [사용(ENABLE)]: 크롤러에 데이터 계보를 사용합니다.
 - [사용 중지(DISABLE)]: 크롤러에 데이터 계보 사용을 중지합니다.

LakeFormationConfiguration 구조

크롤러의 AWS Lake Formation 구성 설정을 지정합니다.

필드

- `UseLakeFormationCredentials` – 부울입니다.
IAM 역할 AWS Lake Formation 자격 증명 대신 크롤러에 대한 자격 증명을 사용할지 여부를 지정합니다.
- `AccountId` – UTF-8 문자열입니다(12바이트 이하).
교차 계정 크롤링에 필요합니다. 대상 데이터와 동일한 계정 크롤링의 경우 이 값을 null로 둘 수 있습니다.

운영

- [CreateCrawler 액션 \(Python: `create_crawler`\)](#)
- [DeleteCrawler 액션 \(Python: `삭제_크롤러`\)](#)
- [GetCrawler 액션 \(파이썬: `get_crawler`\)](#)
- [GetCrawlers 액션 \(파이썬: `get_crawlers`\)](#)
- [GetCrawlerMetrics 액션 \(Python: `get_crawler_metrics`\)](#)
- [UpdateCrawler 액션 \(Python: `업데이트_크롤러`\)](#)
- [StartCrawler 액션 \(Python: `start_crawler`\)](#)
- [StopCrawler 액션 \(Python: `stop_crawler`\)](#)

- [BatchGetCrawlers](#) 액션 (파이썬: `batch_get_crawlers`)
- [ListCrawlers](#) 액션 (Python: `리스트_크롤러`)
- [ListCrawls](#) 액션 (Python: `리스트_크롤링`)

CreateCrawler 액션 (Python: `create_crawler`)

지정된 대상, 역할, 구성 및 선택 일정을 통해 새로운 크롤러를 생성합니다. `s3Targets` 필드, `jdbcTargets` 필드 또는 `DynamoDBTargets` 필드에서 크롤 대상을 하나 이상 지정해야 합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

새로운 크롤러의 이름.

- Role – 필수(Required): UTF-8 문자열입니다.

새로운 크롤러를 사용하여 고객 리소스에 액세스하는 IAM 역할 또는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

- DatabaseName – UTF-8 문자열입니다.

결과가 기록되는 AWS Glue 데이터베이스 (예: `arn:aws:daylight:us-east-1::database/sometable/*`)

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

새로운 크롤러에 대한 설명.

- Targets – 필수(Required): [CrawlerTargets](#) 객체입니다.

크롤할 대상 모음의 기록.

- Schedule – UTF-8 문자열입니다.

일정을 지정하는 데 사용되는 cron 표현식입니다([작업 및 크롤러의 시간 기반 일정](#) 참조). 예를 들어, 매일 오후 12시 15분(UTC)에 실행하려면 `cron(15 12 * * ? *)`을 지정합니다.

- Classifiers – UTF-8 문자열의 배열입니다.

사용자가 등록한 사용자 지정 분류자 목록. 기본적으로 모든 기본 설정 분류자는 크롤러에 포함되지만 이 사용자 지정 분류자는 항상 주어진 분류에 대한 기본 분류자를 재정의합니다.

- `TablePrefix` – 128바이트 이하 길이의 UTF-8 문자열입니다.
생성된 카탈로그 테이블에 사용되는 테이블 접두사입니다.
- `SchemaChangePolicy` – [SchemaChangePolicy](#) 객체입니다.
크롤러의 업데이트 및 삭제 동작 정책입니다.
- `RecrawlPolicy` – [RecrawlPolicy](#) 객체입니다.
전체 데이터 집합을 다시 크롤링할지 아니면 마지막 크롤러 실행 이후 추가된 폴더만 크롤링할지 지정하는 정책입니다.
- `LineageConfiguration` – [LineageConfiguration](#) 객체입니다.
크롤러에 대한 데이터 계보 구성 설정을 지정합니다.
- `LakeFormationConfiguration` – [LakeFormationConfiguration](#) 객체입니다.
크롤러의 AWS Lake Formation 구성 설정을 지정합니다.
- `Configuration` – UTF-8 문자열입니다.
크롤러 구성 정보. 이 버전의 JSON 문자열은 사용자가 크롤러 동작을 지정할 수 있게 만듭니다. 자세한 내용을 알아보려면 [크롤러 구성 옵션 설정](#)을 참조하세요.
- `CrawlerSecurityConfiguration` – 128바이트 이하 길이의 UTF-8 문자열입니다.
이 크롤러가 사용할 `SecurityConfiguration` 구조의 이름입니다.
- `Tags` – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.
각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.
각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.
이 크롤러 요청에서 사용할 태그입니다. 태그를 사용하여 크롤러에 대한 액세스를 제한할 수 있습니다. 의 태그에 대한 자세한 내용은 개발자 AWS Glue 안내서의 [AWS Tags AWS Glue](#) in을 참조하십시오.

응답

- 무응답 파라미터.

Errors

- `InvalidInputException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteCrawler 액션 (Python: 삭제_크롤러)

크롤러 상태가 아닌 경우 에서 지정된 크롤러를 제거합니다. AWS Glue Data Catalog RUNNING

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자 열입니다.

제거할 크롤러의 이름입니다.

응답

- 무응답 파라미터.

Errors

- `EntityNotFoundException`
- `CrawlerRunningException`
- `SchedulerTransitioningException`
- `OperationTimeoutException`

GetCrawler 액션 (파이썬: get_crawler)

지정한 크롤러의 메타데이터 가져오기

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자 열입니다.

메타데이터를 검색할 크롤러 이름입니다.

응답

- Crawler – [크롤러](#) 객체입니다.

지정한 크롤러의 메타데이터

Errors

- EntityNotFoundException
- OperationTimeoutException

GetCrawlers 액션 (파이썬: `get_crawlers`)

사용자 계정에 정의된 모든 크롤러의 메타데이터를 가져옵니다.

요청

- MaxResults – 1~1,000의 숫자(정수)입니다.
각 호출에 따라 반환할 크롤러의 수입니다.
- NextToken – UTF-8 문자열입니다.
이것이 지속적인 요청이라면 지속적인 토큰입니다.

응답

- Crawlers – [크롤러](#) 객체의 배열입니다.
크롤러 메타데이터의 목록.
- NextToken – UTF-8 문자열입니다.
이 사용자 계정에 정의된 것들의 끝에 반환된 목록이 도달하지 못한 경우, 지속적인 토큰입니다.

Errors

- OperationTimeoutException

GetCrawlerMetrics 액션 (Python: `get_crawler_metrics`)

지정한 크롤러의 지표 가져오기

요청

- `CrawlerNameList` – 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.
지표를 가져올 크롤러의 이름 목록.
- `MaxResults` – 1~1,000의 숫자(정수)입니다.
반환할 목록의 최대 크기.
- `NextToken` – UTF-8 문자열입니다.
이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- `CrawlerMetricsList` – [CrawlerMetrics](#) 객체의 배열입니다.
지정한 크롤러에 대한 지표 목록.
- `NextToken` – UTF-8 문자열입니다.
반환된 목록이 사용가능한 마지막 지표를 포함하지 경우의 연속 토큰입니다.

Errors

- `OperationTimeoutException`

UpdateCrawler 액션 (Python: `업데이트_크롤러`)

크롤러 업데이트. 크롤러가 실행 중이면 업데이트하기 전에는 `StopCrawler`를 사용하여 중지해야 합니다.

요청

- `Name` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
새로운 크롤러의 이름.

- Role – UTF-8 문자열입니다.

새로운 크롤러를 사용하여 고객 리소스에 액세스하는 IAM 역할 또는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

- DatabaseName – UTF-8 문자열입니다.

결과가 저장되는 AWS Glue 데이터베이스 (예: `arn:aws:daylight:us-east-1::database/sometable/*`)

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 UTF-8 문자열입니다.

새로운 크롤러에 대한 설명.

- Targets – [CrawlerTargets](#) 객체입니다.

크롤할 대상 목록.

- Schedule – UTF-8 문자열입니다.

일정을 지정하는 데 사용되는 cron 표현식입니다([작업 및 크롤러의 시간 기반 일정](#) 참조). 예를 들어, 매일 오후 12시 15분(UTC)에 실행하려면 `cron(15 12 * * ? *)`을 지정합니다.

- Classifiers – UTF-8 문자열의 배열입니다.

사용자가 등록한 사용자 지정 분류자 목록. 기본적으로 모든 기본 설정 분류자는 크롤러에 포함되지만 이 사용자 지정 분류자는 항상 주어진 분류에 대한 기본 분류자를 재정의합니다.

- TablePrefix – 128바이트 이하 길이의 UTF-8 문자열입니다.

생성된 카탈로그 테이블에 사용되는 테이블 접두어입니다.

- SchemaChangePolicy – [SchemaChangePolicy](#) 객체입니다.

크롤러의 업데이트 및 삭제 동작 정책입니다.

- RecrawlPolicy – [RecrawlPolicy](#) 객체입니다.

전체 데이터 집합을 다시 크롤링할지 아니면 마지막 크롤러 실행 이후 추가된 폴더만 크롤링할지 지정하는 정책입니다.

- LineageConfiguration – [LineageConfiguration](#) 객체입니다.

크롤러에 대한 데이터 계보 구성 설정을 지정합니다.

- LakeFormationConfiguration – [LakeFormationConfiguration](#) 객체입니다.

크롤러의 AWS Lake Formation 구성 설정을 지정합니다.

- Configuration – UTF-8 문자열입니다.

크롤러 구성 정보. 이 버전의 JSON 문자열은 사용자가 크롤러 동작을 지정할 수 있게 만듭니다. 자세한 내용을 알아보려면 [크롤러 구성 옵션 설정](#)을 참조하세요.

- CrawlerSecurityConfiguration – 128바이트 이하 길이의 UTF-8 문자열입니다.

이 크롤러가 사용할 SecurityConfiguration 구조의 이름입니다.

응답

- 무응답 파라미터.

Errors

- InvalidInputException
- VersionMismatchException
- EntityNotFoundException
- CrawlerRunningException
- OperationTimeoutException

StartCrawler 액션 (Python: start_crawler)

어떤 일정이든지 지정된 크롤러를 사용하여 크롤러를 시작합니다. 크롤러가 이미 실행 중이면 a를 반환합니다. [CrawlerRunningException](#)

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

시작할 크롤러의 이름.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- CrawlerRunningException
- OperationTimeoutException

StopCrawler 액션 (Python: stop_crawler)

지정된 크롤러가 실행 중이면 크롤러를 중지합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

중지할 크롤러의 이름.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- CrawlerNotRunningException
- CrawlerStoppingException
- OperationTimeoutException

BatchGetCrawlers 액션 (파이썬: batch_get_crawlers)

주어진 크롤러 이름 목록에 대한 리소스 메타데이터 목록을 반환합니다. ListCrawlers 작업을 호출한 후에는 권한이 부여된 데이터에 액세스하기 위해 이 작업을 호출할 수 있습니다. 이 작업은 태그를 사용하는 권한 조건을 포함해 모든 IAM 권한을 지원합니다.

요청

- CrawlerNames – 필수(Required): 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.

크롤러 이름(ListCrawlers 작업에서 반환된 이름일 수 있음)의 목록입니다.

응답

- Crawlers – [크롤러](#) 객체의 배열입니다.

크롤러 정의 목록.

- CrawlersNotFound – 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.

찾을 수 없는 크롤러의 이름 목록입니다.

Errors

- InvalidInputException
- OperationTimeoutException

ListCrawlers 액션 (Python: 리스트_크롤러)

이 AWS 계정의 모든 크롤러 리소스 또는 지정된 태그가 있는 리소스의 이름을 검색합니다. 이 작업을 통해 계정에서 사용 가능한 리소스와 그 이름을 확인할 수 있습니다.

이 작업을 수행하면 응답에서 필터로 사용할 수 있는 선택 사항인 Tags 필드가 검색되기 때문에 태그가 지정된 리소스를 하나의 그룹으로 검색할 수 있습니다. 태그 필터링을 사용하기로 선택하면 태그가 포함된 리소스만 검색됩니다.

요청

- MaxResults – 1~1,000의 숫자(정수)입니다.

반환할 목록의 최대 크기.

- NextToken – UTF-8 문자열입니다.

이것이 지속적인 요청이라면 지속적인 토큰입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이렇게 태그가 지정된 리소스만 반환하도록 지정합니다.

응답

- `CrawlerNames` – 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.
계정의 모든 크롤러 또는 지정된 태그를 가진 크롤러의 이름.
- `NextToken` – UTF-8 문자열입니다.
반환된 목록이 사용가능한 마지막 지표를 포함하지 경우의 연속 토큰입니다.

Errors

- `OperationTimeoutException`

ListCrawls 액션 (Python: `리스트_크롤링`)

지정된 크롤러에 대한 모든 크롤이 반환됩니다. 크롤러 기록 기능의 시작 날짜 이후 발생한 크롤만 반환되고 최대 12개월의 크롤만 유지됩니다. 이전의 크롤은 반환되지 않습니다.

이 API를 사용하여 다음을 수행할 수 있습니다.

- 지정된 크롤러의 모든 크롤을 검색합니다.
- 제한된 수 내에서 지정된 크롤러의 모든 크롤을 검색합니다.
- 특정 시간 범위에서 지정된 크롤러의 모든 크롤을 검색합니다.
- 특정 상태, 크롤 ID 또는 DPU 시간 값을 사용하여 지정된 크롤러의 모든 크롤을 검색합니다.

요청

- `CrawlerName` – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
실행을 검색할 크롤러의 이름입니다.
- `MaxResults` – 1~1,000의 숫자(정수)입니다.
반환할 최대 결과 수입니다. 기본값은 20이고 최대값은 100입니다.
- `Filters` – [CrawlsFilter](#) 객체의 배열입니다.

`CrawlsFilter` 객체 목록에서 지정하는 기준에 따라 크롤을 필터링합니다.

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- `Crawls` – [CrawlerHistory](#) 객체의 배열입니다.

기준을 충족하는 크롤 실행을 나타내는 `CrawlerHistory` 객체의 목록입니다.

- `NextToken` – UTF-8 문자열입니다.

목록의 현재 세그먼트가 마지막이 아니면 반환된 토큰 목록에 페이지를 매기는 지속적인 토큰은 반환됩니다.

Errors

- `EntityNotFoundException`
- `OperationTimeoutException`
- `InvalidInputException`

열 통계의 API

열 통계 AWS Glue API는 테이블의 열에 대한 통계를 반환하기 위한 API를 설명합니다.

데이터 타입

- [ColumnStatisticsTaskRun 구조](#)
- [ColumnStatisticsTaskRunningException 구조](#)
- [ColumnStatisticsTaskNotRunningException 구조](#)
- [ColumnStatisticsTaskStoppingException 구조](#)

ColumnStatisticsTaskRun 구조

열 통계 실행의 세부 정보를 표시하는 객체입니다.

필드

- `CustomerId` – UTF-8 문자열입니다(12바이트 이하).

AWS 계정 ID입니다.

- `ColumnStatisticsTaskRunId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

특정 열 통계 작업 실행의 식별자입니다.

- `DatabaseName` – UTF-8 문자열입니다.

테이블이 상주하는 데이터베이스.

- `TableName` – UTF-8 문자열입니다.

열 통계가 생성되는 테이블의 이름입니다.

- `ColumnNameList` – UTF-8 문자열의 배열입니다.

열 이름의 목록입니다. 제공되지 않은 경우 기본적으로 테이블의 모든 열 이름이 사용됩니다.

- `CatalogID` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 카탈로그 ID 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `Role` – UTF-8 문자열입니다.

서비스가 통계를 생성하기 위해 맡을 IAM 역할.

- `SampleSize` – 100 이하의 숫자(실수)입니다.

통계 생성에 사용된 행의 비율. 제공되지 않은 경우 전체 테이블을 사용하여 통계를 생성합니다.

- `SecurityConfiguration` – 128바이트 이하 길이의 UTF-8 문자열입니다.

열 통계 작업 실행의 CloudWatch 로그를 암호화하는 데 사용되는 보안 구성의 이름입니다.

- `NumberOfWorkers` – 1 이상의 숫자(정수)입니다.

열 통계를 생성하는 데 사용된 작업자 수입니다. 작업은 인스턴스 25개까지 자동 확장되도록 사전 구성되어 있습니다.

- `WorkerType` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

통계 생성에 사용되는 작업자 유형. 기본값은 g.1x입니다.

- Status – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | SUCCEEDED | FAILED | STOPPED).

실행된 작업의 상태입니다.

- CreationTime – 타임스탬프입니다.

이 작업이 생성된 시각입니다.

- LastUpdated – 타임스탬프입니다.

이 작업이 마지막으로 수정된 시점.

- StartTime – 타임스탬프입니다.

이벤트의 시작 시각.

- EndTime – 타임스탬프입니다.

작업의 종료 시각.

- ErrorMessage – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

작업에 대한 오류 메시지입니다.

- DPUSeconds – None 이하의 숫자(실수)입니다.

자동 크기 조정된 모든 작업자의 계산된 DPU 사용량(초 단위).

ColumnStatisticsTaskRunningException 구조

열 통계 생성 작업을 실행하는 동안 다른 작업을 시작하려고 하면 예외가 발생합니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

ColumnStatisticsTaskNotRunningException 구조

실행 중인 작업이 없을 때 작업 실행을 중지하려고 하면 예외가 발생합니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

ColumnStatisticsTaskStoppingException 구조

작업 실행을 중지하려고 할 때 발생하는 예외입니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

운영

- [StartColumnStatisticsTaskRun 작업 \(Python: start_column_statistics_task_run\)](#)
- [GetColumnStatisticsTaskRun 작업 \(Python: get_column_statistics_task_run\)](#)
- [GetColumnStatisticsTaskRuns 작업 \(Python: get_column_statistics_task_runs\)](#)
- [ListColumnStatisticsTaskRuns 작업 \(Python: list_column_statistics_task_runs\)](#)
- [StopColumnStatisticsTaskRun 작업 \(Python: stop_column_statistics_task_run\)](#)

StartColumnStatisticsTaskRun 작업 (Python: start_column_statistics_task_run)

지정된 테이블 및 열에 대해 열 통계 작업을 실행을 시작합니다.

요청

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 있는 데이터베이스의 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

통계를 생성할 테이블의 이름입니다.

- `ColumnNameList` – UTF-8 문자열의 배열입니다.

통계를 생성할 열 이름의 목록입니다. 제공되지 않은 경우 기본적으로 테이블의 모든 열 이름이 사용됩니다.

- `Role` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

서비스가 통계를 생성하기 위해 맡을 IAM 역할.

- `SampleSize` – 100 이하의 숫자(실수)입니다.

통계 생성에 사용된 행의 비율. 제공되지 않은 경우 전체 테이블을 사용하여 통계를 생성합니다.

- `CatalogID` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블이 존재하는 데이터 카탈로그의 ID. 제공되지 않은 경우 기본적으로 AWS 계정 ID가 사용됩니다.

- `SecurityConfiguration` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

열 통계 작업 실행의 CloudWatch 로그를 암호화하는 데 사용되는 보안 구성의 이름입니다.

응답

- `ColumnStatisticsTaskRunId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

열 통계 작업 실행의 식별자입니다.

Errors

- `AccessDeniedException`
- `EntityNotFoundException`
- `ColumnStatisticsTaskRunningException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `InvalidInputException`

GetColumnStatisticsTaskRun 작업 (Python: get_column_statistics_task_run)

작업 실행 ID가 주어지면 작업 실행과 관련된 메타데이터/정보를 가져옵니다.

요청

- ColumnStatisticsTaskRunId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

특정 열 통계 작업 실행의 식별자입니다.

응답

- ColumnStatisticsTaskRun – [ColumnStatisticsTaskRun](#) 객체입니다.

열 통계 실행의 세부 정보를 나타내는 ColumnStatisticsTaskRun 객체입니다.

Errors

- EntityNotFoundException
- OperationTimeoutException
- InvalidInputException

GetColumnStatisticsTaskRuns 작업 (Python: get_column_statistics_task_runs)

지정된 테이블과 관련된 모든 실행에 대한 정보를 검색합니다.

요청

- DatabaseName – 필수: UTF-8 문자열입니다.

테이블이 있는 데이터베이스의 이름입니다.

- TableName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

- MaxResults – 1~1,000의 숫자(정수)입니다.

응답의 최대 크기입니다.

- NextToken – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- ColumnStatisticsTaskRuns – [ColumnStatisticsTaskRun](#) 객체의 배열입니다.

실행된 작업의 열 통계 목록입니다.

- NextToken – UTF-8 문자열입니다.

실행된 작업이 아직 모두 반환되지 않은 경우의 지속 토큰입니다.

Errors

- OperationTimeoutException

ListColumnStatisticsTaskRuns 작업 (Python: list_column_statistics_task_runs)

특정 계정에 대해 실행된 모든 작업을 나열합니다.

요청

- MaxResults – 1~1,000의 숫자(정수)입니다.

응답의 최대 크기입니다.

- NextToken – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- ColumnStatisticsTaskRunIds – 100개 이하의 문자열로 구성된 UTF-8 문자열입니다.

열 통계 작업 실행 ID 목록.

- NextToken – UTF-8 문자열입니다.

실행된 작업 ID가 아직 전부 반환되지 않은 경우의 지속 토큰입니다.

Errors

- `OperationTimeoutException`

StopColumnStatisticsTaskRun 작업 (Python: `stop_column_statistics_task_run`)

지정된 테이블에 대한 작업 실행을 중지합니다.

요청

- `DatabaseName` – 필수: UTF-8 문자열입니다.

테이블이 있는 데이터베이스의 이름입니다.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블의 이름

응답

- 무응답 파라미터.

Errors

- `EntityNotFoundException`
- `ColumnStatisticsTaskNotRunningException`
- `ColumnStatisticsTaskStoppingException`
- `OperationTimeoutException`

크롤러 스케줄러 API

크롤러 스케줄러 API는 크롤러를 생성, 삭제, 업데이트 및 나열하기 위한 API와 함께 AWS Glue 크롤러 데이터 유형에 대해 설명합니다.

데이터 타입

- [일정 구조](#)

일정 구조

cron을 사용하여 객체의 일정을 정하여 이벤트의 일정을 정합니다.

필드

- `ScheduleExpression` – UTF-8 문자열입니다.

일정을 지정하는 데 사용되는 cron 표현식입니다([작업 및 크롤러의 시간 기반 일정 참조](#)). 예를 들어, 매일 오후 12시 15분(UTC)에 실행하려면 `cron(15 12 * * ? *)`을 지정합니다.

- `State` – UTF-8 문자열입니다(유효 값: SCHEDULED | NOT_SCHEDULED | TRANSITIONING).

일정 상태

운영

- [UpdateCrawlerSchedule 작업\(Python: start_crawler_schedule\)](#)
- [StartCrawlerSchedule 작업\(Python: start_crawler_schedule\)](#)
- [StopCrawlerSchedule 작업\(Python: stop_crawler_schedule\)](#)

UpdateCrawlerSchedule 작업(Python: start_crawler_schedule)

cron 표현식을 사용하여 크롤러의 일정을 업데이트합니다.

요청

- `CrawlerName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트할 일정이 있는 크롤러 이름입니다.

- `Schedule` – UTF-8 문자열입니다.

일정을 지정하는 데 사용되는 업데이트된 cron 표현식입니다([작업 및 크롤러의 시간 기반 일정 참조](#)). 예를 들어, 매일 오후 12시 15분(UTC)에 실행하려면 `cron(15 12 * * ? *)`을 지정합니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- VersionMismatchException
- SchedulerTransitioningException
- OperationTimeoutException

StartCrawlerSchedule 작업(Python: start_crawler_schedule)

크롤러가 실행 중이거나 일정 상태가 이미 SCHEDULED로 되어있지 않는 한 지정된 크롤러의 일정 상태를 SCHEDULED로 변경합니다.

요청

- CrawlerName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

일정을 잡을 크롤러의 이름.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- SchedulerRunningException
- SchedulerTransitioningException
- NoScheduleException
- OperationTimeoutException

StopCrawlerSchedule 작업(Python: stop_crawler_schedule)

지정된 크롤러의 일정 상태를 NOT_SCHEDULED로 설정하지만 크롤러가 실행되고 있으면 크롤러를 중지하지 않습니다.

요청

- CrawlerName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

설정할 일정이 있는 크롤러 이름.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- SchedulerNotRunningException
- SchedulerTransitioningException
- OperationTimeoutException

ETL 스크립트 API 자동 생성

ETL 스크립트 생성 API는 AWS Glue에서 ETL 스크립트를 생성하는 API와 데이터 유형에 대해 설명합니다.

데이터 타입

- [CodeGenNode 구조](#)
- [CodeGenNodeArg 구조](#)
- [CodeGenEdge 구조](#)
- [위치 구조](#)
- [CatalogEntry 구조](#)
- [MappingEntry 구조](#)

CodeGenNode 구조

DAG(방향성 비순환 그래프)로 노드를 표시합니다

필드

- **Id** – 필수(Required): [Identifier string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

노드 그래프 내 고유한 노드 식별자입니다.

- **NodeType** – 필수(Required): UTF-8 문자열입니다.

노드 유형입니다.

- **Args** – 필수(Required): [CodeGenNodeArg](#) 객체의 배열이며 구조는 50개 이하입니다.

이름값 페어 형식인 노드의 속성

- **LineNumber** - 숫자(정수)입니다.

노드의 행 수입니다.

CodeGenNodeArg 구조

인수 또는 노드의 속성.

필드

- **Name** – 필수(Required): UTF-8 문자열입니다.

인수 또는 속성의 이름입니다.

- **Value** – 필수(Required): UTF-8 문자열입니다.

인수 또는 속성의 값입니다.

- **Param** – 부울입니다.

값이 파라미터로 사용되었다면 true입니다.

CodeGenEdge 구조

DAG(방향성 비순환 그래프)로 방향성 엣지를 표시합니다

필드

- Source – 필수(Required): [Identifier string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

엣지가 시작하는 노드의 ID입니다.

- Target – 필수(Required): [Identifier string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

엣지가 끝나는 노드의 ID입니다.

- TargetParameter – UTF-8 문자열입니다.

엣지의 대상입니다.

위치 구조

리소스의 위치입니다.

필드

- Jdbc – [CodeGenNodeArg](#) 객체의 배열이며 구조는 50개 이하입니다.

JDBC 위치

- S3 – [CodeGenNodeArg](#) 객체의 배열이며 구조는 50개 이하입니다.

Amazon Simple Storage Service(Amazon S3) 위치입니다.

- DynamoDB – [CodeGenNodeArg](#) 객체의 배열이며 구조는 50개 이하입니다.

Amazon DynamoDB 테이블 위치입니다.

CatalogEntry 구조

AWS Glue Data Catalog에서 테이블 정의를 지정합니다.

필드

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

테이블 메타데이터가 상주하는 데이터베이스.

- `TableName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

문제의 테이블 이름입니다.

MappingEntry 구조

매핑 정의

필드

- `SourceTable` – UTF-8 문자열입니다.

원본 테이블의 이름.

- `SourcePath` – UTF-8 문자열입니다.

소스 경로입니다.

- `SourceType` – UTF-8 문자열입니다.

소스 유형

- `TargetTable` – UTF-8 문자열입니다.

대상 테이블은 입니다.

- `TargetPath` – UTF-8 문자열입니다.

대상 경로

- `TargetType` – UTF-8 문자열입니다.

대상 유형입니다.

운영

- [CreateScript](#) 작업(Python: `create_script`)
- [GetDataflowGraph](#) 작업(Python: `get_dataflow_graph`)
- [GetMapping](#) 작업(Python: `get_mapping`)
- [GetPlan](#) 작업(Python: `get_plan`)

CreateScript 작업(Python: create_script)

DAG(방향성 비순환 그래프)를 코드로 변환합니다.

요청

- DagNodes – [CodeGenNode](#) 객체의 배열입니다.

DAG 노드 목록.

- DagEdges – [CodeGenEdge](#) 객체의 배열입니다.

DAG 엣지 목록.

- Language – UTF-8 문자열입니다(유효 값: PYTHON | SCALA).

DAG의 결과 코드 프로그래밍 언어.

응답

- PythonScript – UTF-8 문자열입니다.

DAG에서 생성된 Python 스크립트입니다.

- ScalaCode – UTF-8 문자열입니다.

DAG에서 생성된 Scala 코드입니다.

Errors

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetDataflowGraph 작업(Python: get_dataflow_graph)

DAG(방향성 비순환 그래프)로 Python 스크립트를 변환합니다.

요청

- PythonScript – UTF-8 문자열입니다.

변환할 Python 스크립트

응답

- DagNodes – [CodeGenNode](#) 객체의 배열입니다.

결과 DAG 노드 목록.

- DagEdges – [CodeGenEdge](#) 객체의 배열입니다.

결과 DAG 엣지 목록.

Errors

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetMapping 작업(Python: get_mapping)

매핑 생성

요청

- Source – 필수(Required): [CatalogEntry](#) 객체입니다.

원본 테입을 지정합니다.

- Sinks – [CatalogEntry](#) 객체의 배열입니다.

대상 테이블 목록

- Location – [위치](#) 객체입니다.

매핑 파라미터입니다.

응답

- Mapping – 필수(Required): [MappingEntry](#) 객체의 배열입니다.

지정한 대상 매핑 목록.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

GetPlan 작업(Python: `get_plan`)

지정한 매핑을 실행할 코드 얻기.

요청

- `Mapping` – 필수(Required): [MappingEntry](#) 객체의 배열입니다.

원본 테이블에서 대상 테이블로 매핑하는 목록

- `Source` – 필수(Required): [CatalogEntry](#) 객체입니다.

원본 테이블.

- `Sinks` – [CatalogEntry](#) 객체의 배열입니다.

대상 테이블

- `Location` – [위치](#) 객체입니다.

매핑 파라미터입니다.

- `Language` – UTF-8 문자열입니다(유효 값: PYTHON | SCALA).

매핑을 실행할 코드 프로그래밍 언어.

- `AdditionalPlanOptionsMap` – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

추가 선택적 키-값 파라미터를 보유하는 맵입니다.

현재 다음 키값 페어가 지원됩니다.

- `inferSchema` - AWS Glue 작업에 의해 생성된 기본 스크립트에 대해 `inferSchema`를 `true`로 설정할지 아니면 `false`로 설정할지 지정합니다. 예를 들어 `inferSchema`를 `true`로 설정하려면 다음 키 값 페어를 전달합니다.

```
--additional-plan-options-map '{"inferSchema":"true"}
```

응답

- `PythonScript` - UTF-8 문자열입니다.

매핑할 Python 스크립트

- `ScalaCode` - UTF-8 문자열입니다.

매핑을 실행할 Scala 코드.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

시각적 작업 API

시각적 작업 API를 사용하면 작업의 시각적 구성을 나타내는 JSON 개체의 AWS Glue API를 사용하여 데이터 통합 작업을 만들 수 있습니다. AWS Glue

AWS Glue Studio에서 생성된 작업에 대해 DAG를 등록하고 관련 코드를 생성하기 위한 작업 생성 또는 업데이트 API에 목록이 `CodeGenConfigurationNodes` 제공됩니다.

데이터 타입

- [CodeGenConfigurationNode 구조](#)
- [JDBC 구조 ConnectorOptions](#)
- [StreamingDataPreviewOptions 구조체](#)
- [AthenaConnectorSource 구조](#)

- [JDBC 구조 ConnectorSource](#)
- [SparkConnectorSource 구조](#)
- [CatalogSource 구조](#)
- [MySQL 구조 CatalogSource](#)
- [PostgreSQL 구조 CatalogSource](#)
- [오라클 SQL 구조 CatalogSource](#)
- [마이크로소프트/SQL 구조 ServerCatalogSource](#)
- [CatalogKinesisSource 구조](#)
- [DirectKinesisSource 구조](#)
- [KinesisStreamingSourceOptions 구조](#)
- [CatalogKafkaSource 구조](#)
- [DirectKafkaSource 구조](#)
- [KafkaStreamingSourceOptions 구조](#)
- [RedshiftSource 구조](#)
- [AmazonRedshiftSource 구조](#)
- [AmazonRedshiftNodeData 구조](#)
- [AmazonRedshiftAdvancedOption 구조](#)
- [옵션 구조](#)
- [S3 CatalogSource 구조](#)
- [S3 SourceAdditionalOptions 구조](#)
- [S3 CsvSource 구조](#)
- [DirectJDBCSource 구조](#)
- [S3 DirectSourceAdditionalOptions 구조](#)
- [S3 JsonSource 구조](#)
- [S3 구조 ParquetSource](#)
- [S3 구조 DeltaSource](#)
- [S3 CatalogDeltaSource 구조](#)
- [CatalogDeltaSource 구조](#)
- [S3 HudiSource 구조](#)
- [S3 구조 CatalogHudiSource](#)

- [CatalogHudiSource 구조](#)
- [DynamoDB 구조 CatalogSource](#)
- [RelationalCatalogSource 구조](#)
- [JDBC 구조 ConnectorTarget](#)
- [SparkConnectorTarget 구조](#)
- [BasicCatalogTarget 구조](#)
- [MySQL 구조 CatalogTarget](#)
- [PostgreSQL 구조 CatalogTarget](#)
- [오라클 SQL 구조 CatalogTarget](#)
- [마이크로소프트 SQL 구조 ServerCatalogTarget](#)
- [RedshiftTarget 구조](#)
- [AmazonRedshiftTarget 구조](#)
- [UpsertRedshiftTargetOptions 구조](#)
- [S3 CatalogTarget 구조](#)
- [S3 GlueParquetTarget 구조](#)
- [CatalogSchemaChangePolicy 구조](#)
- [S3 DirectTarget 구조](#)
- [S3 HudiCatalogTarget 구조](#)
- [S3 구조 HudiDirectTarget](#)
- [S3 구조 DeltaCatalogTarget](#)
- [S3 DeltaDirectTarget 구조](#)
- [DirectSchemaChangePolicy 구조](#)
- [ApplyMapping 구조](#)
- [Mapping 구조](#)
- [SelectFields 구조](#)
- [DropFields 구조](#)
- [RenameField 구조](#)
- [Spigot 구조](#)
- [조인 구조](#)
- [JoinColumn 구조](#)

- [SplitFields 구조](#)
- [SelectFromCollection 구조](#)
- [FillMissingValues 구조](#)
- [Filter 구조](#)
- [FilterExpression 구조](#)
- [FilterValue 구조](#)
- [CustomCode 구조](#)
- [SparkSQL 구조](#)
- [SqlAlias 구조](#)
- [DropNullFields 구조](#)
- [NullCheckBoxList 구조](#)
- [NullValueField 구조](#)
- [데이터 형식 구조](#)
- [병합 구조](#)
- [결합 구조](#)
- [PIIDetection 구조](#)
- [집계 구조](#)
- [DropDuplicates 구조](#)
- [GovernedCatalogTarget 구조](#)
- [GovernedCatalogSource 구조](#)
- [AggregateOperation 구조](#)
- [GlueSchema 구조](#)
- [GlueStudioSchemaColumn 구조](#)
- [GlueStudioColumn 구조](#)
- [DynamicTransform 구조](#)
- [TransformConfigParameter 구조](#)
- [EvaluateDataQuality 구조](#)
- [DQ 구조 ResultsPublishingOptions](#)
- [DQ 구조 StopJobOnFailureOptions](#)
- [EvaluateDataQualityMultiFrame 구조](#)

- [레시피 구조](#)
- [RecipeReference 구조](#)
- [SnowflakeNodeData 구조](#)
- [SnowflakeSource 구조](#)
- [SnowflakeTarget 구조](#)
- [ConnectorDataSource 구조](#)
- [ConnectorDataTarget 구조](#)

CodeGenConfigurationNode 구조

CodeGenConfigurationNode는 유효한 모든 노드 유형을 열거합니다. 멤버 변수 중 하나만 채울 수 있습니다.

필드

- AthenaConnectorSource – [AthenaConnectorSource](#) 객체입니다.

Amazon Athena 데이터 원본에 대한 커넥터를 지정합니다.

- JDBCConnectorSource – [JDBC ConnectorSource](#) 객체입니다.

JDBC 데이터 원본에 대한 커넥터를 지정합니다.

- SparkConnectorSource – [SparkConnectorSource](#) 객체입니다.

Apache Spark 데이터 원본에 대한 커넥터를 지정합니다.

- CatalogSource – [CatalogSource](#) 객체입니다.

데이터 카탈로그의 AWS Glue 데이터 저장소를 지정합니다.

- RedshiftSource – [RedshiftSource](#) 객체입니다.

Amazon Redshift 데이터 스토어를 지정합니다.

- S3CatalogSource – [S3 CatalogSource](#) 객체입니다.

데이터 카탈로그에 Amazon S3 데이터 스토어를 지정합니다. AWS Glue

- S3CsvSource – [S3 CsvSource](#) 객체입니다.

Amazon S3에 저장된 CSV(쉼표로 구분된 값) 데이터 스토어를 지정합니다.

- S3JsonSource – [S3 JsonSource](#) 객체입니다.

Amazon S3에 저장된 JSON 데이터 스토어를 지정합니다.

- S3ParquetSource – [S3 ParquetSource](#) 객체입니다.

Amazon S3에 저장된 Apache Parquet 데이터 스토어를 지정합니다.

- RelationalCatalogSource – [RelationalCatalogSource](#) 객체입니다.

데이터 카탈로그의 관계형 카탈로그 데이터 스토어를 지정합니다. AWS Glue

- DynamoDBCatalogSource – [DynamoDB CatalogSource](#) 객체입니다.

데이터 카탈로그의 DynamoDB 카탈로그 데이터 스토어를 지정합니다. AWS Glue

- JDBCConnectorTarget – [JDBC ConnectorTarget](#) 객체입니다.

Apache Parquet 열 형식 스토리지의 Amazon S3에 쓰는 데이터 대상을 지정합니다.

- SparkConnectorTarget – [SparkConnectorTarget](#) 객체입니다.

Apache Spark 커넥터를 사용하는 대상을 지정합니다.

- CatalogTarget – [BasicCatalogTarget](#) 객체입니다.

AWS Glue 데이터 카탈로그 테이블을 사용하는 대상을 지정합니다.

- RedshiftTarget – [RedshiftTarget](#) 객체입니다.

Amazon Redshift를 사용하는 대상을 지정합니다.

- S3CatalogTarget – [S3 CatalogTarget](#) 객체입니다.

데이터 카탈로그를 사용하여 Amazon S3에 쓰는 AWS Glue 데이터 대상을 지정합니다.

- S3GlueParquetTarget – [S3 GlueParquetTarget](#) 객체입니다.

Apache Parquet 열 형식 스토리지의 Amazon S3에 쓰는 데이터 대상을 지정합니다.

- S3DirectTarget – [S3 DirectTarget](#) 객체입니다.

Amazon S3에 쓰는 데이터 대상을 지정합니다.

- ApplyMapping – [ApplyMapping](#) 객체입니다.

데이터 원본의 데이터 속성 키를 데이터 대상의 데이터 속성 키에 매핑하는 변환을 지정합니다. 키의 이름을 바꾸고 키의 데이터 유형을 수정하고 데이터 집합에서 삭제할 키를 선택할 수 있습니다.

- SelectFields – [SelectFields](#) 객체입니다.

유지할 데이터 속성 키를 선택하는 변환을 지정합니다.

- DropFields – [DropFields](#) 객체입니다.

삭제할 데이터 속성 키를 선택하는 변환을 지정합니다.

- RenameField – [RenameField](#) 객체입니다.

단일 데이터 속성 키의 이름을 바꾸는 변환을 지정합니다.

- Spigot – [Spigot](#) 객체입니다.

Amazon S3 버킷에 데이터 샘플을 쓰는 변환을 지정합니다.

- Join – [조인](#) 객체입니다.

지정된 데이터 속성 키의 비교 구문을 사용하여 두 데이터 집합을 하나의 데이터 집합으로 조인하는 변환을 지정합니다. 내부, 외부, 왼쪽, 오른쪽, 왼쪽 반 및 왼쪽 안티 조인을 사용할 수 있습니다.

- SplitFields – [SplitFields](#) 객체입니다.

데이터 속성 키를 두 개의 DynamicFrames로 분할하는 변환을 지정합니다. 출력은 DynamicFrames 컬렉션입니다. 하나에는 선택한 데이터 속성 키가 있고 다른 하나에는 나머지 데이터 속성 키가 있습니다.

- SelectFromCollection – [SelectFromCollection](#) 객체입니다.

DynamicFrames 컬렉션에서 하나의 DynamicFrame을 선택하는 변환을 지정합니다. 출력은 선택한 DynamicFrame입니다.

- FillMissingValues – [FillMissingValues](#) 객체입니다.

데이터 집합에서 누락된 값이 있는 레코드를 찾고 대체를 통해 결정된 값으로 새 필드를 추가하는 변환을 지정합니다. 입력 데이터 집합은 누락 값을 결정하는 기계 학습 모델을 훈련하는 데 사용됩니다.

- Filter – [Filter](#) 객체입니다.

필터 조건에 따라 하나의 데이터 집합을 두 개로 분할하는 변환을 지정합니다.

- CustomCode – [CustomCode](#) 객체입니다.

제공한 사용자 지정 코드를 사용하여 데이터 변환을 수행하는 변환을 지정합니다. 출력은 DynamicFrames의 컬렉션입니다.

- SparkSQL – [SparkSQL](#) 객체입니다.

데이터를 변환하기 위해 Spark SQL 구문을 사용하여 SQL 쿼리를 입력하는 변환을 지정합니다. 출력은 단일 `DynamicFrame`입니다.

- `DirectKinesisSource` – [DirectKinesisSource](#) 객체입니다.

직접적인 Amazon Kinesis 데이터 원본을 지정합니다.

- `DirectKafkaSource` – [DirectKafkaSource](#) 객체입니다.

Apache Kafka 데이터 스토어를 지정합니다.

- `CatalogKinesisSource` – [CatalogKinesisSource](#) 객체입니다.

데이터 카탈로그의 Kinesis 데이터 소스를 지정합니다. AWS Glue

- `CatalogKafkaSource` – [CatalogKafkaSource](#) 객체입니다.

데이터 카탈로그의 Apache Kafka 데이터 스토어를 지정합니다.

- `DropNullFields` – [DropNullFields](#) 객체입니다.

열의 모든 값이 'null'인 경우 데이터 집합에서 열을 제거하는 변환을 지정합니다. 기본적으로 AWS Glue Studio는 null 객체를 인식하지만 빈 문자열, "null"인 문자열, -1 정수 또는 기타 자리 표시자 (예: 0)와 같은 일부 값은 자동으로 null로 인식되지 않습니다.

- `Merge` – [병합](#) 객체입니다.

레코드를 식별하기 위해 지정된 기본 키를 기준으로 `DynamicFrame`을 스테이징 `DynamicFrame`과 병합하는 변환을 지정합니다. 중복 레코드(기본 키가 동일한 레코드)는 중복 제거되지 않습니다.

- `Union` – [Union](#) 객체입니다.

둘 이상 데이터 집합의 행을 단일 결과로 결합하는 변환을 지정합니다.

- `PIIDetection` – [PIIDetection](#) 객체입니다.

PII 데이터를 식별, 제거 또는 마스킹하는 변환을 지정합니다.

- `Aggregate` – [Aggregate](#) 객체입니다.

선택한 필드별로 행을 그룹화하고 지정된 함수에 의해 집계된 값을 계산하는 변환을 지정합니다.

- `DropDuplicates` – [DropDuplicates](#) 객체입니다.

데이터 세트에서 반복 데이터의 행을 제거하는 변환을 지정합니다.

- `GovernedCatalogTarget` – [GovernedCatalogTarget](#) 객체입니다.

관리 카탈로그에 작성하는 데이터 대상을 지정합니다.

- GovernedCatalogSource – [GovernedCatalogSource](#) 객체입니다.

관리 데이터 카탈로그의 데이터 소스를 지정합니다.

- MicrosoftSQLServerCatalogSource – [마이크로소프트 SQL ServerCatalogSource](#) 객체입니다.

AWS Glue 데이터 카탈로그의 Microsoft SQL 서버 데이터 소스를 지정합니다.

- MySQLCatalogSource – [MySQL CatalogSource](#) 객체입니다.

데이터 카탈로그의 MySQL 데이터 원본을 지정합니다. AWS Glue

- OracleSQLCatalogSource – [오라클 SQL CatalogSource](#) 객체입니다.

데이터 카탈로그에서 Oracle 데이터 원본을 지정합니다. AWS Glue

- PostgreSQLCatalogSource – [PostgreSQL CatalogSource](#) 객체입니다.

데이터 카탈로그에서 PostgreSQL 데이터 원본을 지정합니다. AWS Glue

- MicrosoftSQLServerCatalogTarget – [마이크로소프트 SQL ServerCatalogTarget](#) 객체입니다.

Microsoft SQL을 사용하는 대상을 지정합니다.

- MySQLCatalogTarget – [MySQL CatalogTarget](#) 객체입니다.

MySQL을 사용하는 대상을 지정합니다.

- OracleSQLCatalogTarget – [오라클 SQL CatalogTarget](#) 객체입니다.

Oracle SQL을 사용하는 대상을 지정합니다.

- PostgreSQLCatalogTarget – [PostgreSQL CatalogTarget](#) 객체입니다.

Postgres SQL을 사용하는 대상을 지정합니다.

- DynamicTransform – [DynamicTransform](#) 객체입니다.

사용자가 생성한 사용자 지정 시각적 변환을 지정합니다.

- EvaluateDataQuality – [EvaluateDataQuality](#) 객체입니다.

데이터 품질 평가 기준을 지정합니다.

- S3CatalogHudiSource – [S3 CatalogHudiSource](#) 객체입니다.

데이터 카탈로그에 등록된 Hudi 데이터 소스를 지정합니다. AWS Glue 데이터 소스는 에 Amazon S3 저장되어야 합니다.

- CatalogHudiSource – [CatalogHudiSource](#) 객체입니다.

데이터 카탈로그에 등록된 Hudi 데이터 원본을 지정합니다. AWS Glue

- S3HudiSource – [S3 HudiSource](#) 객체입니다.

에 저장된 Hudi 데이터 원본을 지정합니다. Amazon S3

- S3HudiCatalogTarget – [S3 HudiCatalogTarget](#) 객체입니다.

데이터 카탈로그의 Hudi 데이터 원본에 AWS Glue 쓰는 대상을 지정합니다.

- S3HudiDirectTarget – [S3 HudiDirectTarget](#) 객체입니다.

에서 Hudi 데이터 원본에 쓰는 대상을 지정합니다. Amazon S3

- S3CatalogDeltaSource – [S3 CatalogDeltaSource](#) 객체입니다.

데이터 카탈로그에 등록된 Delta Lake AWS Glue 데이터 원본을 지정합니다. 데이터 원본은 에 저장되어야 합니다 Amazon S3.

- CatalogDeltaSource – [CatalogDeltaSource](#) 객체입니다.

데이터 카탈로그에 등록된 Delta Lake AWS Glue 데이터 원본을 지정합니다.

- S3DeltaSource – [S3 DeltaSource](#) 객체입니다.

에 저장된 델타 레이크 데이터 원본을 지정합니다 Amazon S3.

- S3DeltaCatalogTarget – [S3 DeltaCatalogTarget](#) 객체입니다.

데이터 카탈로그의 Delta Lake 데이터 원본에 AWS Glue 쓰는 대상을 지정합니다.

- S3DeltaDirectTarget – [S3 DeltaDirectTarget](#) 객체입니다.

델타 레이크 데이터 원본에 쓰는 대상을 지정합니다 Amazon S3.

- AmazonRedshiftSource – [AmazonRedshiftSource](#) 객체입니다.

Amazon Redshift에서 데이터 소스에 작성하는 대상을 지정합니다.

- AmazonRedshiftTarget – [AmazonRedshiftTarget](#) 객체입니다.

Amazon Redshift에서 데이터 대상에 작성하는 대상을 지정합니다.

- EvaluateDataQualityMultiFrame – [EvaluateDataQualityMultiFrame](#) 객체입니다.

데이터 품질 평가 기준을 지정합니다. 여러 입력 데이터를 허용하고 동적 프레임 컬렉션을 반환합니다.

- Recipe – [레시피](#) 객체입니다.

AWS Glue DataBrew 레시피 노드를 지정합니다.

- SnowflakeSource – [SnowflakeSource](#) 객체입니다.

Snowflake 데이터 소스를 지정합니다.

- SnowflakeTarget – [SnowflakeTarget](#) 객체입니다.

Snowflake 데이터 소스에 작성하는 대상을 지정합니다.

- ConnectorDataSource – [ConnectorDataSource](#) 객체입니다.

표준 연결 옵션으로 생성된 소스를 지정합니다.

- ConnectorDataTarget – [ConnectorDataTarget](#) 객체입니다.

표준 연결 옵션으로 생성된 대상을 지정합니다.

JDBC 구조 ConnectorOptions

커넥터에 대한 추가 연결 옵션입니다.

필드

- FilterPredicate – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

소스에서 데이터를 필터링하기 위한 추가 조건 절입니다. 예:

```
BillingCity='Mountain View'
```

테이블 이름 대신 쿼리를 사용하는 경우 쿼리가 지정된 filterPredicate에서 작동하는지 검증해야 합니다.

- PartitionColumn – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

분할에 사용되는 정수 열의 이름입니다. 이 옵션은 lowerBound, upperBound 및 numPartitions에 포함되는 경우에만 작동합니다. 이 옵션은 Spark SQL JDBC 리더에서와 같은 방식으로 작동합니다.

- LowerBound – None 이하의 숫자(정수)입니다.

파티션 스트라이드를 결정하는 데 사용되는 `partitionColumn`의 최소값입니다.

- `UpperBound` – None 이하의 숫자(정수)입니다.

파티션 스트라이드를 결정하는 데 사용되는 `partitionColumn`의 최대값입니다.

- `NumPartitions` – None 이하의 숫자(정수)입니다.

파티션 수입니다. 이 값은 `lowerBound`(포함) 및 `upperBound`(배타)와 함께 `partitionColumn`을 분할하는 데 사용되는 생성된 WHERE 절 표현에 대한 파티션 스트라이드를 형성합니다.

- `JobBookmarkKeys` – UTF-8 문자열의 배열입니다.

정렬할 작업 북마크 키의 이름입니다.

- `JobBookmarkKeysSortOrder` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

오름차순 또는 내림차순 정렬 순서를 지정합니다.

- `DataTypeMapping` – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다(유효한 값: ARRAY | BIGINT | BINARY | BIT | BLOB | BOOLEAN | CHAR | CLOB | DATALINK | DATE | DECIMAL | DISTINCT | DOUBLE | FLOAT | INTEGER | JAVA_OBJECT | LONGNVARCHAR | LONGVARBINARY | LONGVARCHAR | NCHAR | NCLOB | NULL | NUMERIC | NVARCHAR | OTHER | REAL | REF | REF_CURSOR | ROWID | SMALLINT | SQLXML | STRUCT | TIME | TIME_WITH_TIMEZONE | TIMESTAMP | TIMESTAMP_WITH_TIMEZONE | TINYINT | VARBINARY | VARCHAR).

각 값은 UTF-8 문자열입니다(유효한 값: DATE | STRING | TIMESTAMP | INT | FLOAT | LONG | BIGDECIMAL | BYTE | SHORT | DOUBLE).

JDBC 데이터 유형에서 AWS Glue 데이터 유형으로의 매핑을 구축하는 사용자 지정 데이터 유형 매핑입니다. 예를 들어, 옵션은 드라이버의 `ResultSet.getString()` 메서드를 호출하여 JDBC String 유형의 FLOAT 데이터 필드를 Java 유형으로 "dataTypeMapping": {"FLOAT": "STRING"} 매핑하고 이를 사용하여 레코드를 작성합니다. AWS Glue `ResultSet` 객체는 각 드라이버에 의해 구현되므로 동작은 사용하는 드라이버에 따라 다릅니다. 드라이버가 변환을 수행하는 방법을 이해하려면 JDBC 드라이버에 대한 설명서를 참조하세요.

StreamingDataPreviewOptions 구조체

데이터 샘플을 보기 위한 데이터 미리 보기와 관련된 옵션을 지정합니다.

필드

- `PollingTime` - 최소 10 이상의 숫자(long)입니다.
밀리초 단위의 폴링 시간입니다.
- `RecordPollingLimit` - 최소 1 이상의 숫자(long)입니다.
폴링되는 레코드 수에 대한 제한입니다.

AthenaConnectorSource 구조

Amazon Athena 데이터 원본에 대한 커넥터를 지정합니다.

필드

- `Name` - 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 원본의 이름입니다.
- `ConnectionName` - 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
커넥터와 연관된 연결 이름입니다.
- `ConnectorName` - 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
AWS Glue Studio의 데이터 저장소에 액세스하는 데 도움이 되는 커넥터의 이름.
- `ConnectionType` - 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
Amazon Athena 데이터 스토어에 대한 연결을 지정하는 `marketplace.athena` 또는 `custom.athena`와 같은 연결 유형입니다.
- `ConnectionTable` - [Custom string pattern #41](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 원본에 있는 테이블의 이름입니다.
- `SchemaName` - 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 CloudWatch 로그 그룹의 이름입니다. 예: `/aws-glue/jobs/output`.
- `OutputSchemas` - [GlueSchema](#) 객체의 배열입니다.
사용자 지정 Athena 소스에 대한 데이터 스키마를 지정합니다.

JDBC 구조 ConnectorSource

JDBC 데이터 원본에 대한 커넥터를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 원본의 이름입니다.

- ConnectionName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터와 연관된 연결 이름입니다.

- ConnectorName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Studio의 데이터 저장소에 액세스하는 데 도움이 되는 커넥터의 이름. AWS Glue

- ConnectionType – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

JDBC 데이터 스토어에 대한 연결을 지정하는 marketplace.jdbc 또는 custom.jdbc와 같은 연결 유형입니다.

- AdditionalOptions – [JDBC ConnectorOptions](#) 객체입니다.

커넥터에 대한 추가 연결 옵션입니다.

- ConnectionTable – [Custom string pattern #41](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 원본에 있는 테이블의 이름입니다.

- Query – [Custom string pattern #42](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터를 가져올 테이블 또는 SQL 쿼리입니다. ConnectionTable 또는 query를 지정할 수 있지만 둘 다 함께 지정할 수는 없습니다.

- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.

사용자 지정 JDBC 소스에 대한 데이터 스키마를 지정합니다.

SparkConnectorSource 구조

Apache Spark 데이터 원본에 대한 커넥터를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 원본의 이름입니다.
- ConnectionName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
커넥터와 연관된 연결 이름입니다.
- ConnectorName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
AWS Glue Studio의 데이터 저장소에 액세스하는 데 도움이 되는 커넥터의 이름.
- ConnectionType – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
Apache Spark 데이터 스토어에 대한 연결을 지정하는 marketplace.spark 또는 custom.spark와 같은 연결 유형입니다.
- AdditionalOptions – 키-값 페어의 맵 배열입니다.
각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
커넥터에 대한 추가 연결 옵션입니다.
- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.
사용자 지정 Spark 소스에 대한 데이터 스키마를 지정합니다.

CatalogSource 구조

데이터 카탈로그의 AWS Glue 데이터 저장소를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 스토어의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽을 데이터베이스 테이블의 이름입니다.

MySQL 구조 CatalogSource

데이터 카탈로그의 MySQL 데이터 원본을 지정합니다. AWS Glue

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 원본의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스 테이블의 이름입니다.

PostgreSQL 구조 CatalogSource

데이터 카탈로그에서 PostgreSQL 데이터 원본을 지정합니다. AWS Glue

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 원본의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스 테이블의 이름입니다.

오라클 SQL 구조 CatalogSource

데이터 카탈로그에서 Oracle 데이터 소스를 지정합니다. AWS Glue

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 원본의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스 테이블의 이름입니다.

마이크로소프트/SQL 구조 ServerCatalogSource

AWS Glue 데이터 카탈로그의 Microsoft SQL 서버 데이터 소스를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 원본의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스 테이블의 이름입니다.

CatalogKinesisSource 구조

데이터 카탈로그의 Kinesis 데이터 소스를 지정합니다. AWS Glue

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 원본의 이름입니다.
- WindowSize – None 이하의 숫자(정수)입니다.
각 마이크로 배치를 처리하는 데 사용할 시간입니다.

- DetectSchema – 부울입니다.

들어오는 데이터의 스키마를 자동으로 결정할지 여부입니다.

- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽을 데이터베이스 테이블의 이름입니다.

- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽을 데이터베이스의 이름입니다.

- StreamingOptions – [KinesisStreamingSourceOptions](#) 객체입니다.

Kinesis 스트리밍 데이터 원본에 대한 추가 옵션입니다.

- DataPreviewOptions – [StreamingDataPreviewOptions](#) 객체입니다.

데이터 미리 보기에 대한 추가 옵션입니다.

DirectKinesisSource 구조

직접적인 Amazon Kinesis 데이터 원본을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 원본의 이름입니다.

- WindowSize – None 이하의 숫자(정수)입니다.

각 마이크로 배치를 처리하는 데 사용할 시간입니다.

- DetectSchema – 부울입니다.

들어오는 데이터의 스키마를 자동으로 결정할지 여부입니다.

- StreamingOptions – [KinesisStreamingSourceOptions](#) 객체입니다.

Kinesis 스트리밍 데이터 원본에 대한 추가 옵션입니다.

- DataPreviewOptions – [StreamingDataPreviewOptions](#) 객체입니다.

데이터 미리 보기에 대한 추가 옵션입니다.

KinesisStreamingSourceOptions 구조

Amazon Kinesis 스트리밍 데이터 원본에 대한 추가 옵션입니다.

필드

- `EndpointUrl` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Kinesis 엔드포인트의 URL입니다.

- `StreamName` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Kinesis 데이터 스트림의 이름입니다.

- `Classification` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

선택적 분류입니다.

- `Delimiter` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

구분 기호 문자열을 지정합니다.

- `StartingPosition` – UTF-8 문자열입니다(유효한 값: `latest="LATEST"` | `trim_horizon="TRIM_HORIZON"` | `earliest="EARLIEST"` | `timestamp="TIMESTAMP"`).

데이터를 읽을 Kinesis 데이터 스트림의 시작 위치입니다. 가능한 값은 "latest", "trim_horizon", "earliest" 또는 yyyy-mm-ddTHH:MM:SSZ 패턴에서 UTC 형식의 타임스탬프 문자열입니다(여기서, Z는 UTC 시간대 오프셋(+/-)임, 예: '2023-04-04T08:00:00-04:00'). 기본 값은 "latest"입니다.

참고: “StartingPosition”에 UTC 형식의 타임스탬프 문자열 값을 사용하는 것은 AWS Glue 버전 4.0 이상에서만 지원됩니다.

- `MaxFetchTimeInMs` – None 이하의 숫자(정수)입니다.

작업 실행기가 Kinesis 데이터 스트림에서 현재 배치에 대한 레코드를 읽는 데 걸리는 최대 시간(밀리초(ms) 단위로 지정)입니다. 이 시간 내에 여러 개의 `GetRecords` API 호출을 할 수 있습니다. 기본 값은 1000입니다.

- `MaxFetchRecordsPerShard` – None 이하의 숫자(정수)입니다.

마이크로 배치에 따라 Kinesis 데이터 스트림에서 샤드당 가져올 최대 레코드 수입니다. 참고: 스트리밍 작업이 이미 Kinesis의 동일한 `get-records` 호출에서 추가 레코드를 읽은 경우 클라이언트가 이 제한을 초과할 수 있습니다. `MaxFetchRecordsPerShard`가 엄격해야 한다면 `MaxRecordPerRead`의 배수여야 합니다. 기본 값은 100000입니다.

- `MaxRecordPerRead` – None 이하의 숫자(정수)입니다.

각 `getRecords` 작업에서 Kinesis 데이터 스트림으로부터 가져올 최대 레코드 수입니다. 기본 값은 10000입니다.

- `AddIdleTimeBetweenReads` – 부울입니다.

두 개의 연속 `getRecords` 작업 사이에 시간 지연을 추가합니다. 기본 값은 "False"입니다. 이 옵션은 Glue 버전 2.0 이상에서만 구성할 수 있습니다.

- `IdleTimeBetweenReadsInMs` – None 이하의 숫자(정수)입니다.

두 개의 연속 `getRecords` 작업 사이의 최소 시간 지연으로, ms 단위로 지정됩니다. 기본 값은 1000입니다. 이 옵션은 Glue 버전 2.0 이상에서만 구성할 수 있습니다.

- `DescribeShardInterval` – None 이하의 숫자(정수)입니다.

스크립트에서 리샤딩을 고려하기 위한 두 `ListShards` API 호출 사이의 최소 시간 간격입니다. 기본 값은 1s입니다.

- `NumRetries` – None 이하의 숫자(정수)입니다.

Kinesis Data Streams API 요청의 최대 재시도 횟수입니다. 기본 값은 3입니다.

- `RetryIntervalMs` – None 이하의 숫자(정수)입니다.

Kinesis Data Streams API 호출을 재시도하기 전의 휴지 기간(ms 단위로 지정)입니다. 기본 값은 1000입니다.

- `MaxRetryIntervalMs` – None 이하의 숫자(정수)입니다.

Kinesis Data Streams API 호출을 두 번 재시도하는 사이의 최대 휴지 시간(ms 단위로 지정)입니다. 기본 값은 10000입니다.

- `AvoidEmptyBatches` – 부울입니다.

배치가 시작되기 전에 Kinesis 데이터 스트림에서 읽지 않은 데이터를 확인하여 빈 마이크로 배치 작업 생성을 방지합니다. 기본 값은 "False"입니다.

- `StreamArn` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Kinesis 데이터 스트림의 Amazon 리소스 이름(ARN)입니다.

- `RoleArn` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

AWS Security Token Service(AWS STS)를 사용하여 맡을 역할의 Amazon 리소스 이름(ARN)입니다. 이 역할에는 Kinesis 데이터 스트림에 대한 레코드 작업을 설명하거나 읽을 수 있는 권

한이 있어야 합니다. 다른 계정의 데이터 스트림에 액세스할 때 이 파라미터를 사용해야 합니다. "awsSTSSessionName"과(와) 함께 사용합니다.

- RoleSessionName – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

AWS STS를 사용하여 역할을 맡는 세션의 식별자입니다. 다른 계정의 데이터 스트림에 액세스할 때 이 파라미터를 사용해야 합니다. "awsSTSRoleARN"과(와) 함께 사용합니다.

- AddRecordTimestamp – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

이 옵션이 'true'로 설정되면 데이터 출력에는 이름이 '_src_timestamp'라는 추가 열이 포함됩니다. 이 열은 스트림에서 해당 레코드를 수신한 시간을 나타냅니다. 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.

- EmitConsumerLagMetrics – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

이 옵션을 'true'로 설정하면 각 배치에 대해 스트림이 수신한 가장 오래된 레코드와 스트림이 수신된 시간 사이의 기간 동안 메트릭이 내보내집니다. AWS Glue CloudWatch 메트릭의 이름은 "glue.driver.streaming.maxConsumerLagInMs". 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.

- StartingTimestamp – UTF-8 문자열입니다.

Kinesis 데이터 스트림에서 데이터 읽기를 시작하는 레코드의 타임스탬프입니다. 가능한 값은 yyyy-mm-ddTHH:MM:SSZ 패턴에서 UTC 형식의 타임스탬프 문자열입니다(여기서, Z는 UTC 시간 대 오프셋(+/-)임, 예: '2023-04-04T08:00:00+08:00').

CatalogKafkaSource 구조

데이터 카탈로그의 Apache Kafka 데이터 스토어를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 스토어의 이름입니다.

- WindowSize – None 이하의 숫자(정수)입니다.

각 마이크로 배치를 처리하는 데 사용할 시간입니다.

- DetectSchema – 부울입니다.

들어오는 데이터의 스키마를 자동으로 결정할지 여부입니다.

- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스 테이블의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- StreamingOptions – [KafkaStreamingSourceOptions](#) 객체입니다.
스트리밍 옵션을 지정합니다.
- DataPreviewOptions – [StreamingDataPreviewOptions](#) 객체입니다.
데이터 샘플을 보기 위한 데이터 미리 보기와 관련된 옵션을 지정합니다.

DirectKafkaSource 구조

Apache Kafka 데이터 스토어를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 스토어의 이름입니다.
- StreamingOptions – [KafkaStreamingSourceOptions](#) 객체입니다.
스트리밍 옵션을 지정합니다.
- WindowSize – None 이하의 숫자(정수)입니다.
각 마이크로 배치를 처리하는 데 사용할 시간입니다.
- DetectSchema – 부울입니다.
들어오는 데이터의 스키마를 자동으로 결정할지 여부입니다.
- DataPreviewOptions – [StreamingDataPreviewOptions](#) 객체입니다.
데이터 샘플을 보기 위한 데이터 미리 보기와 관련된 옵션을 지정합니다.

KafkaStreamingSourceOptions 구조

스트리밍에 대한 추가 옵션입니다.

필드

- **BootstrapServers** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
부트스트랩 서버 URL 목록입니다(예: b-1.vpc-test-2.o4q88o.c6.kafka.us-east-1.amazonaws.com:9094). 이 옵션은 API 호출에 지정하거나 데이터 카탈로그의 테이블 메타데이터에 정의해야 합니다.
- **SecurityProtocol** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
브로커와 통신하는 데 사용되는 프로토콜입니다. 가능한 값은 "SSL" 또는 "PLAINTEXT"입니다.
- **ConnectionName** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
연결의 이름입니다.
- **TopicName** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
Apache Kafka에 지정된 주제 이름입니다. "topicName", "assign" 또는 "subscribePattern" 중 하나 이상을 지정해야 합니다.
- **Assign** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
사용할 특정 TopicPartitions입니다. "topicName", "assign" 또는 "subscribePattern" 중 하나 이상을 지정해야 합니다.
- **SubscribePattern** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
구독할 주제 목록을 식별하는 Java 정규식 문자열입니다. "topicName", "assign" 또는 "subscribePattern" 중 하나 이상을 지정해야 합니다.
- **Classification** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
선택적 분류입니다.
- **Delimiter** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
구분 기호 문자열을 지정합니다.
- **StartingOffsets** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터를 읽을 Kafka 주제의 시작 위치입니다. 가능한 값은 "earliest" 또는 "latest"입니다. 기본값은 "latest"입니다.
- **EndingOffsets** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

배치 쿼리가 종료되는 엔드포인트입니다. 가능한 값은 "latest" 또는 각 TopicPartition의 끝 오프셋을 지정하는 JSON 문자열입니다.

- PollTimeoutMs – None 이하의 숫자(정수)입니다.

Spark 작업 실행기에서 Kafka의 데이터를 폴링하는 시간 제한(밀리초)입니다. 기본 값은 512입니다.

- NumRetries – None 이하의 숫자(정수)입니다.

Kafka 오프셋 가져오기에 실패하기 전에 재시도할 횟수입니다. 기본 값은 3입니다.

- RetryIntervalMs – None 이하의 숫자(정수)입니다.

Kafka 오프셋 가져오기를 재시도하기 전에 대기할 시간(밀리초)입니다. 기본 값은 10입니다.

- MaxOffsetsPerTrigger – None 이하의 숫자(정수)입니다.

트리거 간격당 처리되는 최대 오프셋 수에 대한 속도 제한입니다. 지정된 총 오프셋 수는 서로 다른 볼륨의 topicPartitions에 비례하여 분할됩니다. 기본값은 null입니다. 즉, 소비자가 알려진 최신 오프셋까지 모든 오프셋을 읽습니다.

- MinPartitions – None 이하의 숫자(정수)입니다.

Kafka에서 읽을 원하는 최소 파티션 수입니다. 기본값은 null이며 이는 Spark 파티션의 수가 Kafka 파티션의 수와 동일함을 의미합니다.

- IncludeHeaders – 부울입니다.

Kafka 헤더를 포함할지 여부입니다. 옵션이 "true"로 설정되면 데이터 출력에는 유형이 Array[Struct(key: String, value: String)]인 "glue_streaming_kafka_headers"라는 추가 열이 포함됩니다. 기본값은 "false"입니다. 이 옵션은 AWS Glue 버전 3.0 이상에서만 사용할 수 있습니다.

- AddRecordTimestamp – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

이 옵션이 'true'로 설정되면 데이터 출력에는 이름이 '_src_timestamp'라는 추가 열이 포함됩니다. 이 열은 주제에서 해당 레코드를 수신한 시간을 나타냅니다. 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.

- EmitConsumerLagMetrics – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

이 옵션을 'true'로 설정하면 각 배치에 대해 주제가 수신한 가장 오래된 레코드와 해당 레코드가 도착한 시간 사이의 기간 동안 메트릭이 내보내집니다. AWS Glue CloudWatch 메트릭의 이름은 "glue.driver.streaming입니다. maxConsumerLagInMs". 기본값은 'false'입니다. 이 옵션은 AWS Glue 버전 4.0 이상에서 지원됩니다.

- `StartingTimestamp` – UTF-8 문자열입니다.

Kafka 주제에서 데이터 읽기를 시작하는 레코드의 타임스탬프입니다. 가능한 값은 yyyy-mm-ddTHH:MM:SSZ 패턴에서 UTC 형식의 타임스탬프 문자열입니다(여기서, Z는 UTC 시간대 오프셋 (+/-)임, 예: '2023-04-04T08:00:00+08:00').

`StartingTimestamp` 또는 `StartingOffsets` 중 하나만 설정해야 합니다.

RedshiftSource 구조

Amazon Redshift 데이터 스토어를 지정합니다.

필드

- `Name` – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

Amazon Redshift 데이터 스토어의 이름입니다.

- `Database` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽어야 할 데이터베이스입니다.

- `Table` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽어야 할 데이터베이스 테이블입니다.

- `RedshiftTmpDir` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터베이스 외부에서 복사할 때 임시 데이터를 스테이징할 수 있는 Amazon S3 경로입니다.

- `TmpDirIAMRole` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

권한이 있는 IAM 역할입니다.

AmazonRedshiftSource 구조

Amazon Redshift 소스를 지정합니다.

필드

- `Name` – [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

Amazon Redshift 소스의 이름입니다.

- Data – [AmazonRedshiftNodeData](#) 객체입니다.

Amazon Redshift 소스 노드의 데이터를 지정합니다.

AmazonRedshiftNodeData 구조

Amazon Redshift 노드를 지정합니다.

필드

- AccessType – [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.
Redshift 연결을 위한 액세스 유형입니다. 직접 연결 또는 카탈로그 연결일 수 있습니다.
- SourceType – [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.
특정 테이블이 소스인지 또는 사용자 지정 쿼리인지를 지정하기 위한 소스 유형입니다.
- Connection – [옵션](#) 객체입니다.
Redshift 클러스터에 대한 AWS Glue 연결입니다.
- Schema – [옵션](#) 객체입니다.
직접 연결로 작업하는 경우 Redshift 스키마 이름입니다.
- Table – [옵션](#) 객체입니다.
직접 연결로 작업하는 경우 Redshift 테이블 이름입니다.
- CatalogDatabase – [옵션](#) 객체입니다.
데이터 카탈로그로 작업할 때 사용되는 AWS Glue 데이터 카탈로그 데이터베이스의 이름.
- CatalogTable – [옵션](#) 객체입니다.
AWS Glue 데이터 카탈로그로 작업할 때의 데이터 카탈로그 테이블 이름.
- CatalogRedshiftSchema – UTF-8 문자열입니다.
데이터 카탈로그로 작업하는 경우 Redshift 스키마 이름입니다.
- CatalogRedshiftTable – UTF-8 문자열입니다.
읽어야 할 데이터베이스 테이블입니다.
- TempDir – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터베이스 외부에서 복사할 때 임시 데이터를 스테이징할 수 있는 Amazon S3 경로입니다.

- `IamRole` - [옵션](#) 객체입니다.

선택 사항입니다. S3에 연결할 때 사용하는 역할 이름입니다. 비어 있는 경우 IAM 역할은 기본적으로 작업의 역할을 사용합니다.

- `AdvancedOptions` - [AmazonRedshiftAdvancedOption](#) 객체의 배열입니다.

Redshift 클러스터에 연결하는 경우 선택적 값입니다.

- `SampleQuery` - UTF-8 문자열입니다.

'쿼리'일 때 Redshift 소스에서 데이터를 가져오는 데 사용되는 SQL입니다 `SourceType`.

- `PreAction` - UTF-8 문자열입니다.

업서트와 함께 MERGE 또는 APPEND를 실행하기 전에 사용되는 SQL입니다.

- `PostAction` - UTF-8 문자열입니다.

업서트와 함께 MERGE 또는 APPEND를 실행하기 전에 사용되는 SQL입니다.

- `Action` - UTF-8 문자열입니다.

Redshift 클러스터에 쓰는 방법을 지정합니다.

- `TablePrefix` - [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

테이블의 접두사를 지정합니다.

- `Upsert` - 부울입니다.

APPEND를 수행하는 경우 Redshift 싱크에서 사용되는 작업입니다.

- `MergeAction` - [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

Redshift 싱크에서 MERGE 처리 방식을 결정할 때 사용되는 작업입니다.

- `MergeWhenMatched` - [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

기존 레코드가 새 레코드와 일치하는 경우 Redshift 싱크에서 MERGE 처리 방식을 결정할 때 사용되는 작업입니다.

- `MergeWhenNotMatched` - [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

기존 레코드가 새 레코드와 일치하지 않는 경우 Redshift 싱크에서 MERGE 처리 방식을 결정할 때 사용되는 작업입니다.

- MergeClause – UTF-8 문자열입니다.

일치하는 레코드를 처리하기 위해 사용자 지정 병합에 사용되는 SQL입니다.

- CrawlerConnection – UTF-8 문자열입니다.

사용된 카탈로그 테이블과 연관된 연결 이름을 지정합니다.

- TableSchema – [옵션](#) 객체의 배열입니다.

지정된 노드에 대한 스키마 출력 배열입니다.

- StagingTable – UTF-8 문자열입니다.

업서트와 함께 MERGE 또는 APPEND를 수행할 때 사용되는 임시 스테이징 테이블의 이름입니다.

- SelectedColumns – [옵션](#) 객체의 배열입니다.

업서트와 함께 MERGE 또는 APPEND를 수행할 때 일치하는 레코드를 결정하는 데 사용되는 열 이름 목록입니다.

AmazonRedshiftAdvancedOption 구조

Redshift 클러스터에 연결할 때 선택적 값을 지정합니다.

필드

- Key – UTF-8 문자열입니다.

추가 연결 옵션의 키입니다.

- Value – UTF-8 문자열입니다.

추가 연결 옵션의 값입니다.

옵션 구조

옵션 값을 지정합니다.

필드

- Value – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

옵션 값을 지정합니다.

- Label – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
옵션의 레이블을 지정합니다.
- Description – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
옵션에 대한 설명을 지정합니다.

S3 CatalogSource 구조

데이터 카탈로그에 Amazon S3 데이터 스토어를 지정합니다. AWS Glue

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 스토어의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽어야 할 데이터베이스입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽어야 할 데이터베이스 테이블입니다.
- PartitionPredicate – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
이 조건자를 충족하는 파티션이 삭제됩니다. 이러한 파티션에서 보존 기간 내에 있는 파일은 삭제되지 않습니다. 기본적으로 ""(비움)로 설정합니다.
- AdditionalOptions – [S3 SourceAdditionalOptions](#) 객체입니다.
추가 연결 옵션을 지정합니다.

S3 SourceAdditionalOptions 구조

Amazon S3 데이터 스토어에 대한 추가 연결 옵션을 지정합니다.

필드

- BoundedSize - 숫자(정수)입니다.
처리될 데이터 집합의 대상 크기에 대한 상한을 바이트 단위로 설정합니다.

- BoundedFiles - 숫자(정수)입니다.

처리될 대상 파일 수에 대한 상한을 설정합니다.

S3 CsvSource 구조

Amazon S3에 저장된 CSV(쉼표로 구분된 값) 데이터 스토어를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 스토어의 이름입니다.

- Paths – 필수: UTF-8 문자열의 배열입니다.

읽을 Amazon S3 경로 목록입니다.

- CompressionType – UTF-8 문자열입니다(유효 값: gzip="GZIP" | bzip2="BZIP2").

데이터 압축 방식을 지정합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip"입니다.

- Exclusions – UTF-8 문자열의 배열입니다.

제외할 Unix 스타일 glob 패턴의 JSON 목록이 포함된 문자열입니다. 예를 들어 "[\]**.pdf\]"는 모든 PDF 파일을 배제합니다.

- GroupSize – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

대상 그룹 크기(바이트)입니다. 입력 데이터 크기와 클러스터 크기에 따라 기본값을 계산합니다. 입력 파일이 50,000개 미만일 때는 "groupFiles"을 "inPartition"으로 설정해야 적용됩니다.

- GroupFiles – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

입력에 50,000개 이상의 파일이 포함된 경우 기본값으로 파일 그룹화가 설정됩니다. 50,000개 미만인 파일로 그룹화를 설정하려면 이 파라미터를 "inPartition"으로 설정합니다. 파일이 50,000개 이상일 때 그룹화를 비활성화하려면 이 파라미터를 "none"으로 설정합니다.

- Recurse – 부울입니다.

true로 설정할 경우 지정된 경로의 모든 하위 디렉터리에 있는 파일을 재귀적으로 읽습니다.

- MaxBand – None 이하의 숫자(정수)입니다.

이 옵션은 s3 목록이 일정하게 유지되기 시작할 가능성이 있는 기간(밀리초)을 제어합니다. 수정 타임스탬프가 마지막 MaxBand 밀리초 이내인 파일은 특히 Amazon S3의 최종 일관성을 고려하여 JobBookmarks 사용할 때 추적됩니다. 대부분의 사용자는 이 옵션을 설정할 필요가 없습니다. 기본 값은 900,000밀리초 또는 15분입니다.

- MaxFilesInBand – None 이하의 숫자(정수)입니다.

이 옵션은 마지막 maxBand초부터 저장할 최대 파일 수를 지정합니다. 이 수를 초과할 경우 추가 파일은 건너뛰고 다음 작업 실행에서만 처리됩니다.

- AdditionalOptions – [S3 DirectSourceAdditionalOptions](#) 객체입니다.

추가 연결 옵션을 지정합니다.

- Separator – 필수: UTF-8 문자열입니다(유효한 값: comma="COMMA" | ctrlA="CTRLA" | pipe="PIPE" | semicolon="SEMICOLON" | tab="TAB").

구분 기호 문자열을 지정합니다. 기본값은 쉼표(",")지만 다른 문자도 지정할 수 있습니다.

- Escaper – [Custom string pattern #41](#)과(와) 일치하는 UTF-8 문자열입니다.

이스케이프에 사용할 문자를 지정합니다. 이 옵션은 CSV 파일을 읽을 때만 사용됩니다. 기본 값은 none입니다. 활성화된 경우 바로 다음에 나오는 문자가 잘 알려진 이스케이프 세트(\n, \r, \t 및 \0)를 제외하고는 있는 그대로 사용됩니다.

- QuoteChar – 필수: UTF-8 문자열입니다(유효한 값: quote="QUOTE" | quillemet="QUILLET" | single_quote="SINGLE_QUOTE" | disabled="DISABLED").

인용에 사용할 문자를 지정합니다. 기본 문자는 큰 따옴표(")입니다: '"'. 전체 인용을 해제하려면 이 값을 -1로 설정합니다.

- Multiline – 부울입니다.

단일 기록이 다양한 라인을 포괄할 수 있는지 여부를 지정하는 부울 값입니다. 필드가 인용된 새로운 라인 문자를 포함할 때 발생합니다. 레코드가 여러 줄에 걸쳐 있는 경우 이 옵션을 True로 설정해야 합니다. 기본값은 False이어서 파싱 동안 더 많은 공격적 파일 쪼개기가 가능합니다.

- WithHeader – 부울입니다.

첫 번째 라인을 헤더로 취급할지 여부를 지정하는 부울 값입니다. 기본 값은 False입니다.

- WriteHeader – 부울입니다.

헤더를 작성하여 출력할지 여부를 지정하는 부울 값입니다. 기본 값은 True입니다.

- SkipFirst – 부울입니다.

첫 번째 데이터 라인을 건너뛴지 여부를 지정하는 부울 값입니다. 기본 값은 `False`입니다.

- `OptimizePerformance` – 부울입니다.

Apache Arrow 기반 열 형식 메모리 포맷과 함께 고급 SIMD CSV 리더를 사용할지 여부를 지정하는 부울 값입니다. 버전 3.0에서만 사용할 수 있습니다. AWS Glue

- `OutputSchemas` – [GlueSchema](#) 객체의 배열입니다.

S3 CSV 소스에 대한 데이터 스키마를 지정합니다.

DirectJDBCSource 구조

직접 JDBC 소스 연결을 지정합니다.

필드

- `Name` – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

JDBC 소스 연결의 이름입니다.

- `Database` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

JDBC 소스 연결의 데이터베이스입니다.

- `Table` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

JDBC 소스 연결의 테이블입니다.

- `ConnectionName` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

JDBC 소스의 연결 이름입니다.

- `ConnectionType` – 필수: UTF-8 문자열입니다(유효한 값: `sqlserver` | `mysql` | `oracle` | `postgresql` | `redshift`).

JDBC 소스의 연결 유형입니다.

- `RedshiftTmpDir` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

JDBC Redshift 소스의 임시 디렉터리입니다.

S3 DirectSourceAdditionalOptions 구조

Amazon S3 데이터 스토어에 대한 추가 연결 옵션을 지정합니다.

필드

- **BoundedSize** - 숫자(정수)입니다.
처리될 데이터 집합의 대상 크기에 대한 상한을 바이트 단위로 설정합니다.
- **BoundedFiles** - 숫자(정수)입니다.
처리될 대상 파일 수에 대한 상한을 설정합니다.
- **EnableSamplePath** - 부울입니다.
샘플 경로를 사용 설정하는 옵션을 설정합니다.
- **SamplePath** - [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
사용 설정된 경우 샘플 경로를 지정합니다.

S3 JsonSource 구조

Amazon S3에 저장된 JSON 데이터 스토어를 지정합니다.

필드

- **Name** - 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 스토어의 이름입니다.
- **Paths** - 필수: UTF-8 문자열의 배열입니다.
읽을 Amazon S3 경로 목록입니다.
- **CompressionType** - UTF-8 문자열입니다(유효 값: `gzip="GZIP" | bzip2="BZIP2"`).
데이터 압축 방식을 지정합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip"입니다.
- **Exclusions** - UTF-8 문자열의 배열입니다.
제외할 Unix 스타일 glob 패턴의 JSON 목록이 포함된 문자열입니다. 예를 들어 "[***.pdf\]"는 모든 PDF 파일을 배제합니다.

- **GroupSize** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

대상 그룹 크기(바이트)입니다. 입력 데이터 크기와 클러스터 크기에 따라 기본값을 계산합니다. 입력 파일이 50,000개 미만일 때는 "groupFiles"을 "inPartition"으로 설정해야 적용됩니다.

- **GroupFiles** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

입력에 50,000개 이상의 파일이 포함된 경우 기본값으로 파일 그룹화가 설정됩니다. 50,000개 미만인 파일로 그룹화를 설정하려면 이 파라미터를 "inPartition"으로 설정합니다. 파일이 50,000개 이상일 때 그룹화를 비활성화하려면 이 파라미터를 "none"으로 설정합니다.

- **Recurse** – 부울입니다.

true로 설정할 경우 지정된 경로의 모든 하위 디렉터리에 있는 파일을 재귀적으로 읽습니다.

- **MaxBand** – None 이하의 숫자(정수)입니다.

이 옵션은 s3 목록이 일정하게 유지되기 시작할 가능성이 있는 기간(밀리초)을 제어합니다. 수정 타임스탬프가 마지막 MaxBand 밀리초 이내인 파일은 특히 Amazon S3의 최종 일관성을 고려하여 JobBookmarks 사용할 때 추적됩니다. 대부분의 사용자는 이 옵션을 설정할 필요가 없습니다. 기본값은 900,000밀리초 또는 15분입니다.

- **MaxFilesInBand** – None 이하의 숫자(정수)입니다.

이 옵션은 마지막 maxBand초부터 저장할 최대 파일 수를 지정합니다. 이 수를 초과할 경우 추가 파일은 건너뛰고 다음 작업 실행에서만 처리됩니다.

- **AdditionalOptions** – [S3 DirectSourceAdditionalOptions](#) 객체입니다.

추가 연결 옵션을 지정합니다.

- **JsonPath** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

JSON 데이터를 정의하는 문자열. JsonPath

- **Multiline** – 부울입니다.

단일 기록이 다양한 라인을 포괄할 수 있는지 여부를 지정하는 부울 값입니다. 필드가 인용된 새로운 라인 문자를 포함할 때 발생합니다. 레코드가 여러 줄에 걸쳐 있는 경우 이 옵션을 True로 설정해야 합니다. 기본값은 False이어서 파싱 동안 더 많은 공격적 파일 쪼개기가 가능합니다.

- **OutputSchemas** – [GlueSchema](#) 객체의 배열입니다.

S3 JSON 소스에 대한 데이터 스키마를 지정합니다.

S3 구조 ParquetSource

Amazon S3에 저장된 Apache Parquet 데이터 스토어를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 스토어의 이름입니다.

- Paths – 필수: UTF-8 문자열의 배열입니다.

읽을 Amazon S3 경로 목록입니다.

- CompressionType – UTF-8 문자열입니다(유효한 값: snappy="SNAPPY" | lzo="LZO" | gzip="GZIP" | uncompressed="UNCOMPRESSED" | none="NONE").

데이터 압축 방식을 지정합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip"입니다.

- Exclusions – UTF-8 문자열의 배열입니다.

제외할 Unix 스타일 glob 패턴의 JSON 목록이 포함된 문자열입니다. 예를 들어 "[\]**.pdf]"는 모든 PDF 파일을 배제합니다.

- GroupSize – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

대상 그룹 크기(바이트)입니다. 입력 데이터 크기와 클러스터 크기에 따라 기본값을 계산합니다. 입력 파일이 50,000개 미만일 때는 "groupFiles"을 "inPartition"으로 설정해야 적용됩니다.

- GroupFiles – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

입력에 50,000개 이상의 파일이 포함된 경우 기본값으로 파일 그룹화가 설정됩니다. 50,000개 미만인 파일로 그룹화를 설정하려면 이 파라미터를 "inPartition"으로 설정합니다. 파일이 50,000개 이상일 때 그룹화를 비활성화하려면 이 파라미터를 "none"으로 설정합니다.

- Recurse – 부울입니다.

true로 설정할 경우 지정된 경로의 모든 하위 디렉터리에 있는 파일을 재귀적으로 읽습니다.

- MaxBand – None 이하의 숫자(정수)입니다.

이 옵션은 s3 목록이 일정하게 유지되기 시작할 가능성이 있는 기간(밀리초)을 제어합니다. 수정 타임스탬프가 마지막 MaxBand 밀리초 이내인 파일은 특히 Amazon S3의 최종 일관성을 고려하여 JobBookmarks 사용할 때 추적됩니다. 대부분의 사용자는 이 옵션을 설정할 필요가 없습니다. 기본 값은 900,000밀리초 또는 15분입니다.

- MaxFilesInBand – None 이하의 숫자(정수)입니다.

이 옵션은 마지막 maxBand로부터 저장할 최대 파일 수를 지정합니다. 이 수를 초과할 경우 추가 파일은 건너뛰고 다음 작업 실행에서만 처리됩니다.

- AdditionalOptions – [S3 DirectSourceAdditionalOptions](#) 객체입니다.

추가 연결 옵션을 지정합니다.

- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.

S3 Parquet 소스에 대한 데이터 스키마를 지정합니다.

S3 구조 DeltaSource

델타 레이크 데이터 원본을 저장하도록 지정합니다 Amazon S3.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

Delta Lake 소스의 이름입니다.

- Paths – 필수: UTF-8 문자열의 배열입니다.

읽을 Amazon S3 경로 목록입니다.

- AdditionalDeltaOptions – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

추가 연결 옵션을 지정합니다.

- AdditionalOptions – [S3 DirectSourceAdditionalOptions](#) 객체입니다.

커넥터의 추가 옵션을 지정합니다.

- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.

Delta Lake 소스에 대한 데이터 스키마를 지정합니다.

S3 CatalogDeltaSource 구조

데이터 카탈로그에 등록된 Delta Lake AWS Glue 데이터 소스를 지정합니다. 데이터 원본은 에 저장되어야 합니다 Amazon S3.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
Delta Lake 데이터 소스의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스 테이블의 이름입니다.
- AdditionalDeltaOptions – 키-값 페어의 맵 배열입니다.
각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
추가 연결 옵션을 지정합니다.
- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.
Delta Lake 소스에 대한 데이터 스키마를 지정합니다.

CatalogDeltaSource 구조

데이터 카탈로그에 등록된 Delta Lake AWS Glue 데이터 원본을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
Delta Lake 데이터 소스의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.

- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스 테이블의 이름입니다.
- AdditionalDeltaOptions – 키-값 페어의 맵 배열입니다.
각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
추가 연결 옵션을 지정합니다.
- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.
Delta Lake 소스에 대한 데이터 스키마를 지정합니다.

S3 HudiSource 구조

에 Amazon S3저장된 Hudi 데이터 소스를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
Hudi 테이블의 이름입니다.
- Paths – 필수: UTF-8 문자열의 배열입니다.
읽을 Amazon S3 경로 목록입니다.
- AdditionalHudiOptions – 키-값 페어의 맵 배열입니다.
각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
추가 연결 옵션을 지정합니다.
- AdditionalOptions – [S3 DirectSourceAdditionalOptions](#) 객체입니다.
커넥터의 추가 옵션을 지정합니다.
- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.

[Hudi 소스에 대한 데이터 스키마를 지정합니다.](#)

S3 구조 CatalogHudiSource

데이터 카탈로그에 등록된 Hudi AWS Glue 데이터 소스를 지정합니다. Hudi 데이터 소스는 에 저장되어야 합니다. Amazon S3

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
Hudi 데이터 소스의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스 테이블의 이름입니다.
- AdditionalHudiOptions – 키-값 페어의 맵 배열입니다.
각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
추가 연결 옵션을 지정합니다.
- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.
Hudi 소스에 대한 데이터 스키마를 지정합니다.

CatalogHudiSource 구조

데이터 카탈로그에 등록된 Hudi AWS Glue 데이터 소스를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
Hudi 데이터 소스의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽을 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽을 데이터베이스 테이블의 이름입니다.

- `AdditionalHudiOptions` – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

추가 연결 옵션을 지정합니다.

- `OutputSchemas` – [GlueSchema](#) 객체의 배열입니다.

Hudi 소스에 대한 데이터 스키마를 지정합니다.

DynamoDB 구조 CatalogSource

데이터 카탈로그의 DynamoDB 데이터 소스를 지정합니다. AWS Glue

필드

- `Name` – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 원본의 이름입니다.

- `Database` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽을 데이터베이스의 이름입니다.

- `Table` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽을 데이터베이스 테이블의 이름입니다.

RelationalCatalogSource 구조

AWS Glue 데이터 카탈로그의 관계형 데이터베이스 데이터 소스를 지정합니다.

필드

- `Name` – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 원본의 이름입니다.

- `Database` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽을 데이터베이스의 이름입니다.

- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

읽을 데이터베이스 테이블의 이름입니다.

JDBC 구조 ConnectorTarget

Apache Parquet 열 형식 스토리지의 Amazon S3에 쓰는 데이터 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- ConnectionName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터와 연관된 연결 이름입니다.

- ConnectionTable – 필수: [Custom string pattern #41](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상에 있는 테이블의 이름입니다.

- ConnectorName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

사용할 커넥터의 이름입니다.

- ConnectionType – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

JDBC 데이터 대상에 대한 연결을 지정하는 marketplace.jdbc 또는 custom.jdbc와 같은 연결 유형입니다.

- AdditionalOptions – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터에 대한 추가 연결 옵션입니다.

- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.

JDBC 대상의 데이터 스키마를 지정합니다.

SparkConnectorTarget 구조

Apache Spark 커넥터를 사용하는 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- ConnectionName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Apache Spark 커넥터에 대한 연결 이름입니다.

- ConnectorName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Apache Spark 커넥터의 이름입니다.

- ConnectionType – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Apache Spark 데이터 스토어에 대한 연결을 지정하는 marketplace.spark 또는 custom.spark와 같은 연결 유형입니다.

- AdditionalOptions – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터에 대한 추가 연결 옵션입니다.

- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.

사용자 지정 Spark 대상에 대한 데이터 스키마를 지정합니다.

BasicCatalogTarget 구조

AWS Glue 데이터 카탈로그 테이블을 사용하는 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
대상으로 사용할 테이블이 포함된 데이터베이스입니다. 이 데이터베이스가 데이터 카탈로그에 이미 존재해야 합니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
출력 데이터의 스키마를 정의하는 테이블입니다. 이 테이블이 데이터 카탈로그에 이미 존재해야 합니다.

MySQL 구조 CatalogTarget

MySQL을 사용하는 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스 테이블의 이름입니다.

PostgreSQL 구조 CatalogTarget

Postgres SQL을 사용하는 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스 테이블의 이름입니다.

오라클 SQL 구조 CatalogTarget

Oracle SQL을 사용하는 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스 테이블의 이름입니다.

마이크로소프트 SQL 구조 ServerCatalogTarget

Microsoft SQL을 사용하는 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스 테이블의 이름입니다.

RedshiftTarget 구조

Amazon Redshift를 사용하는 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스의 이름입니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스 테이블의 이름입니다.
- RedshiftTmpDir – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터베이스 외부에서 복사할 때 임시 데이터를 스테이징할 수 있는 Amazon S3 경로입니다.
- TmpDirIAMRole – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
권한이 있는 IAM 역할입니다.

- UpsertRedshiftOptions – [UpsertRedshiftTargetOptions](#) 객체입니다.

Redshift 대상에 쓸 때 업서트 작업을 구성하는 옵션 세트입니다.

AmazonRedshiftTarget 구조

Amazon Redshift 대상을 지정합니다.

필드

- Name – [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

Amazon Redshift 대상의 이름입니다.

- Data – [AmazonRedshiftNodeData](#) 객체입니다.

Amazon Redshift 대상 노드의 데이터를 지정합니다.

- Inputs – UTF-8 문자열의 배열입니다(1개의 문자열).

데이터 대상에 대한 입력인 노드입니다.

UpsertRedshiftTargetOptions 구조

Redshift 대상에 쓸 때 업서트 작업을 구성하는 옵션입니다.

필드

- TableLocation – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Redshift 테이블의 물리적 위치입니다.

- ConnectionName – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Redshift에 쓰는 데 사용할 연결 이름입니다.

- UpsertKeys – UTF-8 문자열의 배열입니다.

업데이트 또는 삽입 수행 여부를 결정하는 데 사용되는 키입니다.

S3 CatalogTarget 구조

데이터 카탈로그를 사용하여 Amazon S3에 쓰는 AWS Glue 데이터 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- PartitionKeys – UTF-8 문자열의 배열입니다.
일련의 키를 사용하여 기본 분할을 지정합니다.
- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스 테이블의 이름입니다.
- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스의 이름입니다.
- SchemaChangePolicy – [CatalogSchemaChangePolicy](#) 객체입니다.
크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

S3 GlueParquetTarget 구조

Apache Parquet 열 형식 스토리지의 Amazon S3에 쓰는 데이터 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- PartitionKeys – UTF-8 문자열의 배열입니다.
일련의 키를 사용하여 기본 분할을 지정합니다.
- Path – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 단일 Amazon S3 경로입니다.

- **Compression** – UTF-8 문자열입니다(유효한 값: snappy="SNAPPY" | lzo="LZO" | gzip="GZIP" | uncompressed="UNCOMPRESSED" | none="NONE").

데이터 압축 방식을 지정합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip"입니다).

- **SchemaChangePolicy** – [DirectSchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

CatalogSchemaChangePolicy 구조

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

필드

- **EnableUpdateCatalog** – 부울입니다.

크롤러가 변경된 스키마를 찾았을 때 지정된 업데이트 동작을 사용할지 여부입니다.

- **UpdateBehavior** – UTF-8 문자열입니다(유효 값: UPDATE_IN_DATABASE | LOG).

크롤러가 변화된 객체를 찾을 때 업데이트 동작.

S3 DirectTarget 구조

Amazon S3에 쓰는 데이터 대상을 지정합니다.

필드

- **Name** – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- **Inputs** – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- **PartitionKeys** – UTF-8 문자열의 배열입니다.

일련의 키를 사용하여 기본 분할을 지정합니다.

- **Path** – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 단일 Amazon S3 경로입니다.

- Compression – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 압축 방식을 지정합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip"입니다.

- Format – 필수: UTF-8 문자열입니다(유효한 값: json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA").

대상에 대한 데이터 출력 포맷을 지정합니다.

- SchemaChangePolicy – [DirectSchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

S3 HudiCatalogTarget 구조

데이터 카탈로그의 Hudi 데이터 소스에 AWS Glue 쓰는 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- PartitionKeys – UTF-8 문자열의 배열입니다.

일련의 키를 사용하여 기본 분할을 지정합니다.

- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 데이터베이스 테이블의 이름입니다.

- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 데이터베이스의 이름입니다.

- AdditionalOptions – 필수(Required): 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터에 대한 추가 연결 옵션을 지정합니다.

- SchemaChangePolicy – [CatalogSchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

S3 구조 HudiDirectTarget

에서 Hudi 데이터 소스에 쓰는 대상을 지정합니다. Amazon S3

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- Path – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 Hudi 데이터 소스의 Amazon S3 경로입니다.

- Compression – 필수: UTF-8 문자열입니다(유효한 값: `gzip="GZIP" | lzo="LZO" | uncompressed="UNCOMPRESSED" | snappy="SNAPPY"`).

데이터 압축 방식을 지정합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip"입니다).

- PartitionKeys – UTF-8 문자열의 배열입니다.

일련의 키를 사용하여 기본 분할을 지정합니다.

- Format – 필수: UTF-8 문자열입니다(유효한 값: `json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA"`).

대상에 대한 데이터 출력 포맷을 지정합니다.

- AdditionalOptions – 필수(Required): 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터에 대한 추가 연결 옵션을 지정합니다.

- SchemaChangePolicy – [DirectSchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

S3 구조 DeltaCatalogTarget

데이터 카탈로그의 Delta Lake 데이터 소스에 쓰는 대상을 지정합니다. AWS Glue

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- PartitionKeys – UTF-8 문자열의 배열입니다.

일련의 키를 사용하여 기본 분할을 지정합니다.

- Table – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 데이터베이스 테이블의 이름입니다.

- Database – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 데이터베이스의 이름입니다.

- AdditionalOptions – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터에 대한 추가 연결 옵션을 지정합니다.

- SchemaChangePolicy – [CatalogSchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

S3 DeltaDirectTarget 구조

델타 레이크 데이터 소스에 쓰는 대상을 지정합니다 Amazon S3.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 대상의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

데이터 대상에 대한 입력인 노드입니다.

- PartitionKeys – UTF-8 문자열의 배열입니다.

일련의 키를 사용하여 기본 분할을 지정합니다.

- Path – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

쓰기를 수행할 Delta Lake 데이터 소스의 Amazon S3 경로입니다.

- Compression – 필수: UTF-8 문자열입니다(유효한 값: uncompressed="UNCOMPRESSED" | snappy="SNAPPY").

데이터 압축 방식을 지정합니다. 이 작업은 데이터에 표준 파일 확장자가 있는 경우에는 필요하지 않습니다. 가능한 값은 "gzip" 및 "bzip"입니다).

- Format – 필수: UTF-8 문자열입니다(유효한 값: json="JSON" | csv="CSV" | avro="AVRO" | orc="ORC" | parquet="PARQUET" | hudi="HUDI" | delta="DELTA").

대상에 대한 데이터 출력 포맷을 지정합니다.

- AdditionalOptions – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

커넥터에 대한 추가 연결 옵션을 지정합니다.

- SchemaChangePolicy – [DirectSchemaChangePolicy](#) 객체입니다.

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

DirectSchemaChangePolicy 구조

크롤러에 대한 업데이트 동작을 지정하는 정책입니다.

필드

- `EnableUpdateCatalog` – 부울입니다.

크롤러가 변경된 스키마를 찾았을 때 지정된 업데이트 동작을 사용할지 여부입니다.

- `UpdateBehavior` – UTF-8 문자열입니다(유효 값: UPDATE_IN_DATABASE | LOG).

크롤러가 변화된 객체를 찾을 때 업데이트 동작.

- `Table` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

스키마 변경 정책이 적용되는 데이터베이스의 테이블을 지정합니다.

- `Database` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

스키마 변경 정책이 적용되는 데이터베이스를 지정합니다.

ApplyMapping 구조

데이터 원본의 데이터 속성 키를 데이터 대상의 데이터 속성 키에 매핑하는 변환을 지정합니다. 키의 이름을 바꾸고 키의 데이터 유형을 수정하고 데이터 집합에서 삭제할 키를 선택할 수 있습니다.

필드

- `Name` – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- `Inputs` – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

노드 이름으로 식별된 데이터 입력입니다.

- `Mapping` – 필수: [Mapping](#) 객체의 배열입니다.

데이터 원본의 데이터 속성 키를 데이터 대상의 데이터 속성 키에 매핑하도록 지정합니다.

Mapping 구조

데이터 속성 키의 매핑을 지정합니다.

필드

- ToKey – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
매핑을 적용한 후의 열 이름입니다. FromPath와 같을 수 있습니다.
- FromPath – UTF-8 문자열의 배열입니다.
수정할 테이블 또는 열입니다.
- FromType – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
수정할 데이터 유형입니다.
- ToType – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터가 수정되는 데이터 유형입니다.
- Dropped – 부울입니다.
true인 경우 열이 제거됩니다.
- Children – [Mapping](#) 객체의 배열입니다.

중첩된 데이터 구조에만 적용됩니다. 상위 구조뿐만 아니라 하위 구조 중 하나도 변경하려는 경우 이 데이터 구조를 작성할 수 있습니다. 마찬가지로 Mapping이지만 해당 FromPath도 상위 구조의 FromPath와 이 구조의 FromPath가 됩니다.

하위 부분의 경우 다음과 같은 구조가 있다고 가정합니다.

```
{ "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType":
"Struct", "Dropped": false, "Children": [{ "FromPath": "inner", "ToKey":
"inner", "ToType": "Double", "Dropped": false, }] }
```

다음과 같은 Mapping을 지정할 수 있습니다.

```
{ "FromPath": "OuterStructure", "ToKey": "OuterStructure", "ToType":
"Struct", "Dropped": false, "Children": [{ "FromPath": "inner", "ToKey":
"inner", "ToType": "Double", "Dropped": false, }] }
```

SelectFields 구조

유지할 데이터 속성 키를 선택하는 변환을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
노드 이름으로 식별된 데이터 입력입니다.
- Paths – 필수: UTF-8 문자열의 배열입니다.
데이터 구조의 변수에 대한 JSON 경로입니다.

DropFields 구조

삭제할 데이터 속성 키를 선택하는 변환을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
노드 이름으로 식별된 데이터 입력입니다.
- Paths – 필수: UTF-8 문자열의 배열입니다.
데이터 구조의 변수에 대한 JSON 경로입니다.

RenameField 구조

단일 데이터 속성 키의 이름을 바꾸는 변환을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
노드 이름으로 식별된 데이터 입력입니다.

- **SourcePath** – 필수: UTF-8 문자열의 배열입니다.
소스 데이터에 대한 데이터 구조의 변수에 대한 JSON 경로입니다.
- **TargetPath** – 필수: UTF-8 문자열의 배열입니다.
대상 데이터에 대한 데이터 구조의 변수에 대한 JSON 경로입니다.

Spigot 구조

Amazon S3 버킷에 데이터 샘플을 쓰는 변환을 지정합니다.

필드

- **Name** – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- **Inputs** – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
노드 이름으로 식별된 데이터 입력입니다.
- **Path** – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
변환이 데이터 집합의 레코드 하위 집합을 Amazon S3 버킷의 JSON 파일에 쓰는 Amazon S3의 경로입니다.
- **Topk** – 100 이하의 숫자(정수)입니다.
데이터 집합의 시작 부분부터 쓸 레코드 수를 지정합니다.
- **Prob** – 1 이하의 숫자(double)입니다.
지정된 레코드를 선택할 확률(최대값이 1인 소수 값)입니다. 값 1은 데이터 집합에서 읽은 각 행이 샘플 출력에 포함되어야 함을 나타냅니다.

조인 구조

지정된 데이터 속성 키의 비교 구문을 사용하여 두 데이터 집합을 하나의 데이터 집합으로 조인하는 변환을 지정합니다. 내부, 외부, 왼쪽, 오른쪽, 왼쪽 반 및 왼쪽 안티 조인을 사용할 수 있습니다.

필드

- **Name** – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(2개 이상)입니다.

노드 이름으로 식별된 데이터 입력입니다.

- JoinType – 필수: UTF-8 문자열입니다(유효한 값: equijoin="EQUIJOIN" | left="LEFT" | right="RIGHT" | outer="OUTER" | leftsemi="LEFT_SEMI" | leftanti="LEFT_ANTI").

데이터 집합에서 수행할 조인 유형을 지정합니다.

- Columns – 필수(Required): 2개 이상의 구조로 이루어진 [JoinColumn](#) 객체의 배열입니다.

조인할 두 열의 목록입니다.

JoinColumn 구조

조인할 열을 지정합니다.

필드

- From – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

조인할 열입니다.

- Keys – 필수: UTF-8 문자열의 배열입니다.

조인할 열의 키입니다.

SplitFields 구조

데이터 속성 키를 두 개의 DynamicFrames로 분할하는 변환을 지정합니다. 출력은 DynamicFrames 컬렉션입니다. 하나에는 선택한 데이터 속성 키가 있고 다른 하나에는 나머지 데이터 속성 키가 있습니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

노드 이름으로 식별된 데이터 입력입니다.

- Paths – 필수: UTF-8 문자열의 배열입니다.

데이터 구조의 변수에 대한 JSON 경로입니다.

SelectFromCollection 구조

DynamicFrames 컬렉션에서 하나의 DynamicFrame을 선택하는 변환을 지정합니다. 출력은 선택한 DynamicFrame입니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

노드 이름으로 식별된 데이터 입력입니다.

- Index – 필수(Required): None 이하의 숫자(정수)입니다.

선택할 인덱스입니다. DynamicFrame

FillMissingValues 구조

데이터 집합에서 누락된 값이 있는 레코드를 찾고 대체를 통해 결정된 값으로 새 필드를 추가하는 변환을 지정합니다. 입력 데이터 집합은 누락 값을 결정하는 기계 학습 모델을 훈련하는 데 사용됩니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

노드 이름으로 식별된 데이터 입력입니다.

- ImputedPath – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

대체된 데이터 집합에 대한 데이터 구조의 변수에 대한 JSON 경로입니다.

- FilledPath – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
작성된 데이터 집합에 대한 데이터 구조의 변수에 대한 JSON 경로입니다.

Filter 구조

필터 조건에 따라 하나의 데이터 집합을 두 개로 분할하는 변환을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
노드 이름으로 식별된 데이터 입력입니다.
- LogicalOperator – 필수: UTF-8 문자열입니다(유효한 값: AND | OR).
키 값을 지정된 값과 비교하여 행을 필터링하는 데 사용되는 연산자입니다.
- Filters – 필수: [FilterExpression](#) 객체의 배열입니다.
필터 표현식을 지정합니다.

FilterExpression 구조

필터 표현식을 지정합니다.

필드

- Operation – 필수: UTF-8 문자열입니다(유효한 값: EQ | LT | GT | LTE | GTE | REGEX | ISNULL).
표현식에서 수행할 작업의 유형입니다.
- Negated – 부울입니다.
표현식을 부정할지 여부입니다.
- Values – 필수: [FilterValue](#) 객체의 배열입니다.
필터 값 목록입니다.

FilterValue 구조

FilterExpression의 값 목록에 있는 단일 항목을 나타냅니다.

필드

- Type – 필수: UTF-8 문자열입니다(유효한 값: COLUMNEXTRACTED | CONSTANT).

필터 값 유형입니다.

- Value – 필수: UTF-8 문자열의 배열입니다.

연결할 값입니다.

CustomCode 구조

제공한 사용자 지정 코드를 사용하여 데이터 변환을 수행하는 변환을 지정합니다. 출력은 의 모음입니다 DynamicFrames.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- Inputs – 필수: UTF-8 문자열의 배열이며 문자열은 1개 이상입니다.

노드 이름으로 식별된 데이터 입력입니다.

- Code – 필수: [Custom string pattern #35](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 변환을 수행하는 데 사용되는 사용자 지정 코드입니다.

- ClassName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

사용자 지정 코드 노드 클래스에 대해 정의된 이름입니다.

- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.

사용자 지정 코드 변환에 대한 데이터 스키마를 지정합니다.

SparkSQL 구조

데이터를 변환하기 위해 Spark SQL 구문을 사용하여 SQL 쿼리를 입력하는 변환을 지정합니다. 출력은 단일 DynamicFrame입니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- Inputs – 필수: UTF-8 문자열의 배열이며 문자열은 1개 이상입니다.

노드 이름으로 식별된 데이터 입력입니다. SQL 쿼리에 사용할 각 입력 노드와 테이블 이름을 연결할 수 있습니다. 선택한 이름은 Spark SQL 이름 지정 제한을 충족해야 합니다.

- SqlQuery – 필수: [Custom string pattern #42](#)과(와) 일치하는 UTF-8 문자열입니다.

Spark SQL 구문을 사용하고 단일 데이터 집합을 반환해야 하는 SQL 쿼리입니다.

- SqlAliases – 필수: [SqlAlias](#) 객체의 배열입니다.

별칭 목록입니다. 별칭을 사용하면 지정된 입력에 대해 SQL에서 사용할 이름을 지정할 수 있습니다. 예를 들어 이름이 "MyDataSource"인 데이터 원본이 있습니다. as와 From MyDataSource Alias SqlName as를 지정하면 SQL에서 다음을 수행할 수 있습니다.

```
select * from SqlName
```

그리고 여기서 데이터를 가져옵니다 MyDataSource.

- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.

SparkSQL 변환에 대한 데이터 스키마를 지정합니다.

SqlAlias 구조

SqlAliases의 값 목록에 있는 단일 항목을 나타냅니다.

필드

- From – 필수: [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

테이블 또는 테이블의 열입니다.

- **Alias** – 필수: [Custom string pattern #41](#)과(와) 일치하는 UTF-8 문자열입니다.

테이블 또는 테이블의 열에 지정된 임시 이름입니다.

DropNullFields 구조

열의 모든 값이 'null'인 경우 데이터 집합에서 열을 제거하는 변환을 지정합니다. 기본적으로 AWS Glue Studio는 null 객체를 인식하지만 빈 문자열, "null"인 문자열, -1 정수 또는 0과 같은 기타 자리 표시자와 같은 일부 값은 자동으로 null로 인식되지 않습니다.

필드

- **Name** – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- **Inputs** – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.

노드 이름으로 식별된 데이터 입력입니다.

- **NullCheckBoxList** – [NullCheckBoxList](#) 객체입니다.

특정 값을 제거하기 위해 Null 값으로 인식할지 여부를 나타내는 구조입니다.

- **NullTextList** – [NullValueField](#) 객체의 배열이며 구조는 50개 이하입니다.

데이터셋에 고유한 null 자리 표시자로 사용되는 0이나 기타 값과 같은 사용자 지정 Null 값을 나타내는 NullValueField 구조 목록을 지정하는 구조체입니다.

DropNullFields 변환은 Null 자리 표시자의 값과 데이터 유형이 모두 데이터와 일치하는 경우에만 사용자 지정 Null 값을 제거합니다.

NullCheckBoxList 구조

제거를 위해 특정 값을 Null 값으로 인식할지 여부를 나타냅니다.

필드

- **IsEmpty** – 부울입니다.

빈 문자열이 Null 값으로 간주되도록 지정합니다.

- **IsNullString** – 부울입니다.

'null'이라는 단어의 철자를 사용하는 값이 Null 값으로 간주되도록 지정합니다.

- IsNegOne – 부울입니다.

정수 값 -1이 Null 값으로 간주되도록 지정합니다.

NullValueField 구조

0이나 데이터 집합에 고유한 Null 자리 표시자로 사용되는 다른 값과 같은 사용자 지정 Null 값을 나타냅니다.

필드

- Value – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Null 자리 표시자의 값입니다.

- Datatype – 필수(Required): [데이터 형식](#) 객체입니다.

값의 데이터 유형입니다.

데이터 형식 구조

값의 데이터 유형을 나타내는 구조입니다.

필드

- Id – 필수: [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

값의 데이터 유형입니다.

- Label – 필수: [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 유형에 할당된 레이블입니다.

병합 구조

레코드를 식별하기 위해 지정된 기본 키를 기준으로 DynamicFrame을 스테이징 DynamicFrame과 병합하는 변환을 지정합니다. 중복 레코드(기본 키가 동일한 레코드)는 중복 제거되지 않습니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(2개 이상)입니다.
노드 이름으로 식별된 데이터 입력입니다.
- Source – 필수: [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.
스테이징 DynamicFrame과 병합되는 소스 DynamicFrame입니다.
- PrimaryKeys – 필수: UTF-8 문자열의 배열입니다.
소스 및 스테이징 동적 프레임의 레코드와 일치시킬 기본 키 필드 목록입니다.

결합 구조

둘 이상 데이터 집합의 행을 단일 결과로 결합하는 변환을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(2개 이상)입니다.
노드 ID가 변환에 입력됩니다.
- UnionType – 필수: UTF-8 문자열입니다(유효한 값: ALL | DISTINCT).
Union 변환 유형을 나타냅니다.

데이터 소스의 모든 행을 결과 행에 ALL 결합하도록 지정합니다 DynamicFrame. 결과 union 구조는 중복 행을 제거하지 않습니다.

결과에서 중복된 행을 DISTINCT 제거하도록 지정합니다 DynamicFrame.

PIIDetection 구조

PII 데이터를 식별, 제거 또는 마스킹하는 변환을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

변환 노드의 이름입니다.

- Inputs – 필수: UTF-8 문자열의 배열(1개 이상)입니다.

노드 ID가 변환에 입력됩니다.

- PiiType – 필수: UTF-8 문자열입니다(유효한 값: RowAudit | RowMasking | ColumnAudit | ColumnMasking).

PIIDetection 변환 유형을 나타냅니다.

- EntityTypesToDetect – 필수: UTF-8 문자열의 배열입니다.

PIIDetection 변환이 PII 데이터로 식별할 엔터티 유형을 나타냅니다.

PII 유형 엔터티는 다음을 포함합니다. PERSON_NAME, DATE, USA_SNN, EMAIL, USA_ITIN, USA_PASSPORT_NUMBER, PHONE_NUMBER, BANK_ACCOUNT, IP_ADDRESS, MAC_ADDRESS, USA_CPT_CODE, USA_HCPCS_CODE, USA_NATIONAL_DRUG_CODE, USA_MEDICARE_BENEFICIARY_IDENTIFIER, USA_HEALTH_INSURANCE_CLAIM_NUMBER, CREDIT_CARD, USA_NATIONAL_PROVIDER_IDENTIFIER

- OutputColumnName – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

해당 행에서 감지된 모든 엔터티 유형을 포함할 출력 열 이름을 나타냅니다.

- SampleFraction – 1 이하의 숫자(실수)입니다.

PII 엔터티를 스캔할 때 샘플링할 데이터의 비율을 나타냅니다.

- ThresholdFraction – 1 이하의 숫자(실수)입니다.

열을 PII 데이터로 식별하기 위해 충족되어야 하는 데이터의 비율을 나타냅니다.

- MaskValue – [Custom string pattern #37](#)과(와) 일치하는 256바이트 이하 길이의 UTF-8 문자열입니다.

감지된 개체를 대체할 값을 나타냅니다.

집계 구조

선택한 필드별로 행을 그룹화하고 지정된 함수에 의해 집계된 값을 계산하는 변환을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- Inputs – 필수: UTF-8 문자열의 배열(1개 이상)입니다.
집계 변환에 대해 입력으로 사용할 필드와 행을 지정합니다.
- Groups – 필수: UTF-8 문자열의 배열입니다.
그룹화할 필드를 지정합니다.
- Aggs – 필수(Required): [AggregateOperation](#) 객체의 배열이며 구조는 1~30개입니다.
지정된 필드에서 수행할 집계 함수를 지정합니다.

DropDuplicates 구조

데이터 세트에서 반복 데이터의 행을 제거하는 변환을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
변환 노드의 이름입니다.
- Inputs – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
노드 이름으로 식별된 데이터 입력입니다.
- Columns – UTF-8 문자열의 배열입니다.
반복될 경우 병합하거나 제거할 열의 이름입니다.

GovernedCatalogTarget 구조

데이터 카탈로그를 사용하여 Amazon S3에 쓰는 AWS Glue 데이터 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 대상의 이름입니다.

- **Inputs** – 필수(Required): UTF-8 문자열의 배열(1개 이상)입니다.
데이터 대상에 대한 입력인 노드입니다.
- **PartitionKeys** – UTF-8 문자열의 배열입니다.
일련의 키를 사용하여 기본 분할을 지정합니다.
- **Table** – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스 테이블의 이름입니다.
- **Database** – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
쓰기를 수행할 데이터베이스의 이름입니다.
- **SchemaChangePolicy** – [CatalogSchemaChangePolicy](#) 객체입니다.
관리 카탈로그에 대한 업데이트 동작을 지정하는 정책입니다.

GovernedCatalogSource 구조

관리되는 데이터 카탈로그의 AWS Glue 데이터 저장소를 지정합니다.

필드

- **Name** – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
데이터 스토어의 이름입니다.
- **Database** – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽어야 할 데이터베이스입니다.
- **Table** – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
읽어야 할 데이터베이스 테이블입니다.
- **PartitionPredicate** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
이 조건자를 충족하는 파티션이 삭제됩니다. 이러한 파티션에서 보존 기간 내에 있는 파일은 삭제되지 않습니다. 기본적으로 ""(비움)로 설정합니다.
- **AdditionalOptions** – [S3 SourceAdditionalOptions](#) 객체입니다.
추가 연결 옵션을 지정합니다.

AggregateOperation 구조

집계 변환에서 집계를 수행하는 데 필요한 파라미터 세트를 지정합니다.

필드

- Column – 필수: UTF-8 문자열의 배열입니다.

집계 함수가 적용될 데이터 세트의 열을 지정합니다.

- AggFunc – 필수(Required): UTF-8 문자열입니다(유효 값: avg | countDistinct | count | first | last | kurtosis | max | min | skewness | stddev_samp | stddev_pop | sum | sumDistinct | var_samp | var_pop).

적용할 집계 함수를 지정합니다.

가능한 집계 함수로는 평균 개수별, 개수, 첫 번째, 마지막, kurtosis, 최대, 최소, 왜도, stddev_samp, stddev_pop, 합계, sumDistinct, var_samp, var_pop 등이 있습니다.

GlueSchema 구조

스키마를 AWS Glue에서 결정할 수 없는 경우 사용자 정의 스키마를 지정합니다.

필드

- Columns – [GlueStudioSchemaColumn](#) 객체의 배열입니다.

AWS Glue 스키마를 구성하는 열 정의를 지정합니다.

GlueStudioSchemaColumn 구조

AWS Glue 스키마 정의에서 단일 열을 지정합니다.

필드

- Name – 필수: [Single-line string pattern](#)과 일치하는 1,024바이트 이하 길이의 UTF-8 문자열입니다.

AWS Glue Studio 스키마의 열 이름.

- Type – [Single-line string pattern](#)과(와) 일치하는 131,072바이트 이하 길이의 UTF-8 문자열입니다.

AWS Glue Studio 스키마에서 이 열의 하이브 유형입니다.

GlueStudioColumn 구조

AWS Glue Studio에서 단일 열을 지정합니다.

필드

- Key – 필수: [Custom string pattern #41](#)과(와) 일치하는 UTF-8 문자열입니다.

AWS Glue Studio의 열 키입니다.

- FullPath – 필수: UTF-8 문자열의 배열입니다.

AWS Glue 스튜디오에 있는 컬럼의 전체 URL.

- Type – 필수: UTF-8 문자열(유효한 값: array="ARRAY" | bigint="BIGINT" | bigint array="BIGINT_ARRAY" | binary="BINARY" | binary array="BINARY_ARRAY" | boolean="BOOLEAN" | boolean array="BOOLEAN_ARRAY" | byte="BYTE" | byte array="BYTE_ARRAY" | char="CHAR" | char array="CHAR_ARRAY" | choice="CHOICE" | choice array="CHOICE_ARRAY" | date="DATE" | date array="DATE_ARRAY" | decimal="DECIMAL" | decimal array="DECIMAL_ARRAY" | double="DOUBLE" | double array="DOUBLE_ARRAY" | enum="ENUM" | enum array="ENUM_ARRAY" | float="FLOAT" | float array="FLOAT_ARRAY" | int="INT" | int array="INT_ARRAY" | interval="INTERVAL" | interval array="INTERVAL_ARRAY" | long="LONG" | long array="LONG_ARRAY" | object="OBJECT" | short="SHORT" | short array="SHORT_ARRAY" | smallint="SMALLINT" | smallint array="SMALLINT_ARRAY" | string="STRING" | string array="STRING_ARRAY" | timestamp="TIMESTAMP" | timestamp array="TIMESTAMP_ARRAY" | tinyint="TINYINT" | tinyint array="TINYINT_ARRAY" | varchar="VARCHAR" | varchar array="VARCHAR_ARRAY" | null="NULL" | unknown="UNKNOWN" | unknown array="UNKNOWN_ARRAY").

AWS Glue 스튜디오의 컬럼 유형.

- Children - 구조의 배열입니다.

AWS Glue Studio에 있는 상위 컬럼의 하위 항목입니다.

DynamicTransform 구조

동적 변환을 수행하는 데 필요한 파라미터 세트를 지정합니다.

필드

- Name – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
동적 변환의 이름을 지정합니다.
- TransformName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
AWS Glue Studio 비주얼 에디터에 나타나는 동적 변환의 이름을 지정합니다.
- Inputs – 필수: UTF-8 문자열의 배열(1개 이상)입니다.
필요한 동적 변환에 대한 입력을 지정합니다.
- Parameters – [TransformConfigParameter](#) 객체의 배열입니다.
동적 변환의 파라미터를 지정합니다.
- FunctionName – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
동적 변환의 함수 이름을 지정합니다.
- Path – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
동적 변환 소스 및 구성 파일의 경로를 지정합니다.
- Version – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
이 필드는 사용되지 않으며 향후 릴리스에서 사용 중단됩니다.
- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.
동적 변환에 대한 데이터 스키마를 지정합니다.

TransformConfigParameter 구조

동적 변환 구성 파일의 파라미터를 지정합니다.

필드

- Name – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
동적 변환 구성 파일의 파라미터 이름을 지정합니다.
- Type – 필수: UTF-8 문자열입니다(유효한 값: str="STR" | int="INT" | float="FLOAT" | complex="COMPLEX" | bool="BOOL" | list="LIST" | null="NULL").

동적 변환 구성 파일의 파라미터 유형을 지정합니다.

- `ValidationRule` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

동적 변환 구성 파일의 검증 규칙을 지정합니다.

- `ValidationMessage` – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

동적 변환 구성 파일의 검증 메시지를 지정합니다.

- `Value` – UTF-8 문자열의 배열입니다.

동적 변환 구성 파일의 파라미터 값을 지정합니다.

- `ListType` – UTF-8 문자열입니다(유효한 값: `str="STR" | int="INT" | float="FLOAT" | complex="COMPLEX" | bool="BOOL" | list="LIST" | null="NULL"`).

동적 변환 구성 파일의 파라미터 목록 유형을 지정합니다.

- `IsOptional` – 부울입니다.

파라미터가 동적 변환 구성 파일에서 선택 사항인지 여부를 지정합니다.

EvaluateDataQuality 구조

데이터 품질 평가 기준을 지정합니다.

필드

- `Name` – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 품질 평가의 이름입니다.

- `Inputs` – 필수: UTF-8 문자열의 배열(1개 이상)입니다.

데이터 품질 평가의 입력입니다.

- `Ruleset` – 필수: [Custom string pattern #38](#)과(와) 일치하는 1~65536바이트 길이의 UTF-8 문자열입니다.

데이터 품질 평가를 위한 규칙 세트입니다.

- `Output` – UTF-8 문자열입니다(유효한 값: `PrimaryInput | EvaluationResults`).

데이터 품질 평가의 출력입니다.

- `PublishingOptions` – [DQ ResultsPublishingOptions](#) 객체입니다.

결과 게시 방법을 구성하는 옵션입니다.

- StopJobOnFailureOptions – [DQ StopJobOnFailureOptions](#) 객체입니다.

데이터 품질 평가에 실패할 경우 작업을 중지하는 방법을 구성하는 옵션입니다.

DQ 구조 ResultsPublishingOptions

데이터 품질 평가 결과 게시 방법을 구성하는 옵션입니다.

필드

- EvaluationContext – [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

평가의 컨텍스트입니다.

- ResultsS3Prefix – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

결과 앞에 Amazon S3 접두사가 추가되었습니다.

- CloudWatchMetricsEnabled – 부울입니다.

데이터 품질 결과에 대한 지표를 활성화합니다.

- ResultsPublishingEnabled – 부울입니다.

데이터 품질 결과에 대한 게시를 활성화합니다.

DQ 구조 StopJobOnFailureOptions

데이터 품질 평가에 실패할 경우 작업을 중지하는 방법을 구성하는 옵션입니다.

필드

- StopJobOnFailureTiming – UTF-8 문자열입니다(유효한 값: Immediate | AfterDataLoad).

데이터 품질 평가에 실패할 경우에 작업을 중지할 시점입니다. 옵션은 즉시 또는 AfterDataLoad 입니다.

EvaluateDataQualityMultiFrame 구조

데이터 품질 평가 기준을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

데이터 품질 평가의 이름입니다.

- Inputs – 필수: UTF-8 문자열의 배열이며 문자열은 1개 이상입니다.

데이터 품질 평가의 입력입니다. 이 목록의 첫 번째 입력은 기본 데이터 소스입니다.

- AdditionalDataSources – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

기본을 제외한 모든 데이터 소스의 별칭입니다.

- Ruleset – 필수: [Custom string pattern #38](#)과(와) 일치하는 1~65536바이트 길이의 UTF-8 문자열입니다.

데이터 품질 평가를 위한 규칙 세트입니다.

- PublishingOptions – [DQ ResultsPublishingOptions](#) 객체입니다.

결과 게시 방법을 구성하는 옵션입니다.

- AdditionalOptions – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열(유효 값: performanceTuning.caching="CacheOption" | observations.scope="ObservationsOption")입니다.

각 값은 UTF-8 문자열입니다.

변환의 런타임 동작을 구성하는 옵션입니다.

- StopJobOnFailureOptions – [DQ StopJobOnFailureOptions](#) 객체입니다.

데이터 품질 평가에 실패할 경우 작업을 중지하는 방법을 구성하는 옵션입니다.

레시피 구조

AWS Glue 작업에서 AWS Glue DataBrew 레시피를 사용하는 AWS Glue 스튜디오 노트.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
AWS Glue 스튜디오 노드의 이름.
- Inputs – 필수: UTF-8 문자열의 배열(1개 이상)입니다.
레시피 노드에 대한 입력에 해당하는 노드로, ID로 식별됩니다.
- RecipeReference – 필수: [RecipeReference](#) 객체입니다.
노드에서 사용하는 DataBrew 레시피에 대한 참조입니다.

RecipeReference 구조

AWS Glue DataBrew 레시피에 대한 참조.

필드

- RecipeArn – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.
레시피의 ARN입니다. DataBrew
- RecipeVersion – 필수: 1~16바이트 길이의 UTF-8 문자열입니다.
RecipeVersion DataBrew 레시피의 일종입니다.

SnowflakeNodeData 구조

Studio의 스노우플레이크 노드 구성을 지정합니다. AWS Glue

필드

- SourceType – [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.
검색된 데이터를 지정하는 방법을 지정합니다. 유효한 값: "table", "query".
- Connection – [옵션](#) 객체입니다.
Snowflake 엔드포인트에 대한 AWS Glue 데이터 카탈로그 연결을 지정합니다.
- Schema – UTF-8 문자열입니다.
노드에서 사용할 Snowflake 데이터베이스 스키마를 지정합니다.

- **Table** – UTF-8 문자열입니다.

노드에서 사용할 Snowflake 테이블을 지정합니다.

- **Database** – UTF-8 문자열입니다.

노드에서 사용할 Snowflake 데이터베이스를 지정합니다.

- **TempDir** – [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

현재 사용 중이지 않습니다.

- **IamRole** – [옵션](#) 객체입니다.

현재 사용 중이지 않습니다.

- **AdditionalOptions** – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

Snowflake 커넥터에 전달되는 추가 옵션을 지정합니다. 이 노드의 다른 위치에서 옵션이 지정된 경우 이 옵션이 우선합니다.

- **SampleQuery** – UTF-8 문자열입니다.

query 소스 유형의 데이터를 검색하는 데 사용되는 SQL 문자열입니다.

- **PreAction** – UTF-8 문자열입니다.

Snowflake 커넥터가 표준 작업을 수행하기 전에 실행되는 SQL 문자열입니다.

- **PostAction** – UTF-8 문자열입니다.

Snowflake 커넥터가 표준 작업을 수행한 후에 실행되는 SQL 문자열입니다.

- **Action** – UTF-8 문자열입니다.

기존 데이터가 있는 테이블에 쓸 때 수행할 작업을 지정합니다. 유효한 값: `append`, `merge`, `truncate`, `drop`.

- **Upsert** – 부울입니다.

`append` 작업일 때 사용됩니다. 행이 이미 있는 경우 확인 동작을 지정합니다. `true`인 경우 기존 행이 업데이트됩니다. `false`인 경우 해당 행이 삽입됩니다.

- **MergeAction** – [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

병합 작업을 지정합니다. 유효한 값: simple, custom. simple인 경우 병합 동작은 MergeWhenMatched 및 MergeWhenNotMatched로 정의됩니다. custom인 경우 MergeClause로 정의됩니다.

- MergeWhenMatched – [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

병합 시 기존 데이터와 일치하는 레코드를 확인하는 방법을 지정합니다. 유효한 값: update, delete.

- MergeWhenNotMatched – [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

병합 시 기존 데이터와 일치하지 않는 레코드를 처리하는 방법을 지정합니다. 유효한 값: insert, none.

- MergeClause – UTF-8 문자열입니다.

사용자 지정 병합 동작을 지정하는 SQL 문입니다.

- StagingTable – UTF-8 문자열입니다.

merge 또는 업서트 append 작업을 수행할 때 사용되는 스테이징 테이블의 이름입니다. 데이터가 이 테이블에 기록된 후에는 생성된 사후 작업에 의해 table로 이동됩니다.

- SelectedColumns – [옵션](#) 객체의 배열입니다.

병합 및 업서트에서 일치하는 항목을 감지할 때 레코드를 식별하기 위해 결합된 열을 지정합니다. value, label 및 description 키가 있는 구조 목록입니다. 각 구조는 열을 설명합니다.

- AutoPushdown – 부울입니다.

자동 쿼리 푸시다운의 활성화 여부를 지정합니다. 푸시다운이 활성화된 경우 Spark에서 쿼리를 실행할 때 쿼리의 일부를 Snowflake 서버로 '푸시다운'할 수 있으면 해당 쿼리가 푸시다운됩니다. 이렇게 하면 일부 쿼리의 성능이 향상됩니다.

- TableSchema – [옵션](#) 객체의 배열입니다.

노드의 대상 스키마를 수동으로 정의합니다. value, label 및 description 키가 있는 구조 목록입니다. 각 구조는 열을 정의합니다.

SnowflakeSource 구조

Snowflake 데이터 소스를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
Snowflake 데이터 소스의 이름입니다.
- Data – 필수: [SnowflakeNodeData](#) 객체입니다.
Snowflake 데이터 소스의 구성입니다.
- OutputSchemas – [GlueSchema](#) 객체의 배열입니다.
출력 데이터에 대한 사용자 정의 스키마를 지정합니다.

SnowflakeTarget 구조

Snowflake 대상을 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
Snowflake 대상의 이름입니다.
- Data – 필수: [SnowflakeNodeData](#) 객체입니다.
Snowflake 대상 노드의 데이터를 지정합니다.
- Inputs – UTF-8 문자열의 배열입니다(1개의 문자열).
데이터 대상에 대한 입력인 노드입니다.

ConnectorDataSource 구조

표준 연결 옵션으로 생성된 소스를 지정합니다.

필드

- Name – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.
이 소스 노드의 이름입니다.
- ConnectionType – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

기본 AWS Glue 라이브러리에 제공된 대로입니다. `connectionType` 노드 유형은 다음과 같은 연결 유형을 지원합니다.

- `opensearch`
- `azuresql`
- `azurecosmos`
- `bigquery`
- `saphana`
- `teradata`
- `vertica`
- `Data` – 필수(Required): 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

노드에 대한 연결 옵션을 지정하는 맵입니다. AWS Glue 설명서의 연결 [매개변수 섹션에서 해당 연결 유형에 대한 표준 연결 옵션](#)을 찾을 수 있습니다.

- `OutputSchemas` – [GlueSchema](#) 객체의 배열입니다.

이 소스에 대한 데이터 스키마를 지정합니다.

ConnectorDataTarget 구조

표준 연결 옵션으로 생성된 대상을 지정합니다.

필드

- `Name` – 필수: [Custom string pattern #43](#)과(와) 일치하는 UTF-8 문자열입니다.

이 대상 노드의 이름입니다.

- `ConnectionType` – 필수: [Custom string pattern #40](#)과(와) 일치하는 UTF-8 문자열입니다.

기본 AWS Glue 라이브러리에 제공된 대로입니다. `connectionType` 노드 유형은 다음과 같은 연결 유형을 지원합니다.

- `opensearch`
- `azuresql`

- azurecosmos
- bigquery
- saphana
- teradata
- vertica
- Data – 필수(Required): 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

노드에 대한 연결 옵션을 지정하는 맵입니다. AWS Glue 설명서의 연결 [매개변수 섹션에서 해당 연결 유형에 대한 표준 연결](#) 옵션을 찾을 수 있습니다.

- Inputs – UTF-8 문자열의 배열입니다(1개의 문자열).

데이터 대상에 대한 입력인 노드입니다.

작업 API

작업 API는 작업 데이터 유형을 설명하며 AWS Glue에서 작업, 작업 실행, 트리거로 작업하기 위한 API를 포함합니다.

주제

- [작업](#)
- [작업 실행](#)
- [트리거](#)

작업

Jobs API는 에서 작업을 생성, 업데이트, 삭제 또는 보는 것과 관련된 데이터 유형 및 API를 설명합니다 AWS Glue.

데이터 타입

- [작업 구조](#)
- [ExecutionProperty 구조](#)

- [NotificationProperty 구조](#)
- [JobCommand 구조](#)
- [ConnectionsList 구조](#)
- [JobUpdate 구조](#)
- [SourceControlDetails 구조](#)

작업 구조

작업 정의를 지정합니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업 정의에 할당하는 이름입니다.

- JobMode – UTF-8 문자열입니다(유효한 값: SCRIPT="" | VISUAL="" | NOTEBOOK="").

작업이 생성된 방법을 설명하는 모드입니다. 유효한 값은 다음과 같습니다.

- SCRIPT- 작업은 AWS Glue Studio 스크립트 편집기를 사용하여 생성되었습니다.
- VISUAL- AWS Glue Studio 비주얼 에디터를 사용하여 작업을 생성했습니다.
- NOTEBOOK - 대화형 세션 노트북을 사용하여 작업을 생성했습니다.

JobMode 필드가 없거나 null인 경우 기본값으로 SCRIPT가 할당됩니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

작업 설명입니다.

- LogUri – UTF-8 문자열입니다.

이 필드는 향후 사용하기 위해 예약되어 있습니다.

- Role – UTF-8 문자열입니다.

이 작업과 연결된 IAM 역할의 이름 또는 Amazon 리소스 이름(ARN)입니다.

- CreatedOn – 타임스탬프입니다.

작업 정의가 생성된 날짜와 시간입니다.

- LastModifiedOn – 타임스탬프입니다.

이 작업 정의가 수정된 마지막 시점입니다.

- ExecutionProperty – [ExecutionProperty](#) 객체입니다.

ExecutionProperty는 이 작업에 허용된 최대 동시 실행 수를 지정합니다.

- Command – [JobCommand](#) 객체입니다.

이 작업을 실행하는 JobCommand입니다.

- DefaultArguments – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

이 작업의 모든 실행에서 기본 인수(이름 값 페어로 지정됨)입니다.

작업 실행 스크립트에서 사용하는 인수와 자체적으로 소비하는 AWS Glue 인수를 여기에 지정할 수 있습니다.

작업 인수가 로깅될 수 있습니다. 일반 텍스트 보안 암호를 인수로 전달하지 마세요. Job 내에 암호를 보관하려는 경우 AWS Glue Connection AWS Secrets Manager 또는 기타 비밀 관리 메커니즘에서 암호를 검색하십시오.

자체 작업 인수를 지정하고 사용하는 방법에 대한 자세한 내용은 개발자 가이드의 [Python에서 AWS Glue Glue API](#) 호출을 참조하세요.

Spark 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [AWS Glue에서 사용하는 특별 파라미터](#) 주제를 참조하세요.

Ray 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [Ray 작업에서 작업 파라미터 사용](#)을 참조하세요.

- NonOverridableArguments – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

작업 실행 시 작업 인수를 제공할 때 재정의되지 않는 이 작업의 인수(이름 값 페어로 지정됨)입니다.

- Connections – [ConnectionsList](#) 객체입니다.

이 작업에 사용된 연결입니다.

- `MaxRetries` - 숫자(정수)입니다.

`a`가 JobRun 실패한 후 이 작업을 재시도할 수 있는 최대 횟수입니다.

- `AllocatedCapacity` - 숫자(정수)입니다.

이 필드는 더 이상 사용되지 않습니다. 대신 `MaxCapacity`을 사용하세요.

이 작업 실행에 할당된 AWS Glue 데이터 처리 장치 (DPU) 수입니다. 최소 2DPU를 할당할 수 있습니다. 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

- `Timeout` - 1 이상의 숫자(정수)입니다.

작업 타임아웃(분)입니다. 작업을 실행하여 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 배치 작업의 경우 기본값은 2,880분(48시간)입니다.

스트리밍 작업의 시간 제한 값은 7일 또는 10,080분 미만이어야 합니다. 값을 비워 두면 유지 관리 기간을 설정하지 않았다고 가정하고 7일 후에 작업이 다시 시작됩니다. 유지 관리 기간을 설정한 경우 7일 후 유지 관리 기간 중에 작업이 다시 시작됩니다.

- `MaxCapacity` - 숫자(double)입니다.

Glue 버전 1.0 이하 작업의 경우 표준 작업자 유형을 사용하여 이 작업을 실행할 때 할당할 수 있는 AWS Glue 데이터 처리 장치 (DPU) 수를 계산합니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

Glue 버전 2.0 이상 작업의 경우 `Maximum capacity`를 지정할 수 없습니다. 그 대신 `Worker type` 및 `Number of workers`를 지정해야 합니다.

`WorkerType` 및 `NumberOfWorkers`를 사용하는 경우, `MaxCapacity`를 설정하지 마십시오.

`MaxCapacity`에 할당할 수 있는 값은 Python 셸 작업을 실행하는지 또는 Apache Spark ETL 작업 또는 Apache Spark 스트리밍 ETL 작업을 실행하는지에 따라 다릅니다.

- Python 셸 작업(`JobCommand.Name="pythonshell"`)을 지정하면 0.0625 또는 1 DPU를 할당할 수 있습니다. 기본값은 0.0625 DPU입니다.

- Apache Spark ETL 작업(JobCommand.Name="glueetl") 또는 Apache Spark 스트리밍 ETL 작업(JobCommand.Name="gluestreaming") 을 지정하면 2에서 100 DPU를 할당할 수 있습니다. 기본 값은 10 DPU입니다. 이 작업 유형에는 부분적인 DPU 할당을 사용할 수 없습니다.
- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

작업이 실행될 때 할당되는 미리 정의된 작업자 유형입니다. Spark 작업에 대해 G.1X, G.2X, G.4X, G.8X 또는 G.025X의 값을 허용합니다. Ray 작업에 대해 Z.2X 값을 허용합니다.

- G.1X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 1개의 DPU(vCPU 4개, 메모리 16GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 77GB의 여유 공간)에서 2개의 DPU(vCPU 8개, 메모리 32GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.4X 작업자 유형에서 각 작업자는 256GB 디스크(약 235GB의 여유 공간)에서 4개의 DPU(vCPU 16개, 메모리 64GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 미국 동부 (오하이오), 미국 동부 (버지니아 북부), 미국 서부 (오레곤), 아시아 태평양 (싱가포르), 아시아 태평양 (시드니), 아시아 태평양 (도쿄), 캐나다 (중부), 유럽 (프랑크푸르트), 유럽 (아일랜드) 및 유럽 (스톡홀름) 의 AWS Glue 버전 3.0 이상 Spark ETL 작업에만 사용할 수 있습니다. AWS
- G.8X 작업자 유형에서 각 작업자는 512GB 디스크(약 487GB의 여유 공간)에서 8개의 DPU(vCPU 32개, 메모리 128GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 작업자 유형에 지원되는 AWS 지역과 동일한 지역에서 AWS Glue 버전 3.0 이상의 Spark ETL G.4X 작업에만 사용할 수 있습니다.
- G.025X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 0.25개의 DPU(vCPU 2개, 메모리 4GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 볼륨이 낮은 스트리밍 작업에 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 AWS Glue 버전 3.0 스트리밍 작업에만 사용할 수 있습니다.
- Z.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 120GB의 여유 공간)에서 2개의 M-DPU(vCPU 8개, 메모리 64GB)에 매핑되고, Autoscaler에 따라 최대 8개의 Ray 작업자를 제공합니다.

- `NumberOfWorkers` - 숫자(정수)입니다.

작업이 실행될 때 할당되는 정의된 `workerType`의 작업자 수입니다.

- `SecurityConfiguration` - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업에 사용할 `SecurityConfiguration` 구조의 이름입니다.

- `NotificationProperty` - [NotificationProperty](#) 객체입니다.

작업 알림의 구성 속성을 지정합니다.

- `Running` - 부울입니다.

이 필드는 향후 사용하기 위해 예약되어 있습니다.

- `GlueVersion` - [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Spark 작업에서 작업에서 AWS Glue 사용할 수 있는 Apache Spark 및 Python 버전을 `GlueVersion` 결정합니다. Python의 버전으로 Spark 유형의 작업에 대해 지원되는 버전을 확인할 수 있습니다.

Ray 작업에서는 `GlueVersion`을 4.0 이상으로 설정해야 합니다. 그러나 Ray 작업에서 사용할 수 있는 Ray, Python 및 추가 라이브러리의 버전은 Job 명령의 Runtime 파라미터에 의해 결정됩니다.

사용 가능한 AWS Glue 버전과 해당 Spark 및 Python 버전에 대한 자세한 내용은 개발자 안내서의 [Glue 버전을](#) 참조하십시오.

Glue 버전 지정 없이 생성된 작업은 Glue 0.9로 기본 지정됩니다.

- `CodeGenConfigurationNodes` - 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [CodeGenConfigurationNode](#) 객체입니다.

Glue Studio 시각적 구성 요소 및 Glue Studio 코드 생성의 기반이 되는 방향성 비순환 그래프의 표현입니다.

- `ExecutionClass` - 16바이트 미만의 UTF-8 문자열입니다(유효한 값: FLEX="" | STANDARD="").

작업이 표준 또는 유연한 실행 클래스로 실행되는지 여부를 나타냅니다. 표준 실행 클래스는 빠른 작업 시작 및 전용 리소스가 필요한 시간에 민감한 워크로드에 적합합니다.

유연한 실행 클래스는 시작 및 완료 시간이 다를 수 있는 시간에 민감하지 않은 작업에 적합합니다.

AWS Glue 버전 3.0 이상의 작업과 명령 glueetl 유형만 ExecutionClass 로 FLEX 설정할 수 있습니다. 유연한 실행 클래스는 Spark 작업에 사용할 수 있습니다.

- SourceControlDetails – [SourceControlDetails](#) 객체입니다.

작업에 대한 소스 제어 구성에 대한 세부 정보로, 원격 리포지토리와 작업 아티팩트 동기화를 허용합니다.

- MaintenanceWindow – [Custom string pattern #30](#)과(와) 일치하는 UTF-8 문자열입니다.

이 필드는 스트리밍 작업에 대한 유지 관리 기간의 요일과 시간을 지정합니다. AWS Glue 정기적으로 유지 관리 활동을 수행합니다. 이러한 유지 관리 기간 AWS Glue 동안에는 스트리밍 작업을 다시 시작해야 합니다.

AWS Glue 지정된 유지 관리 기간으로부터 3시간 이내에 작업을 다시 시작합니다. 예를 들어, 유지 관리 기간을 GMT 기준 월요일 오전 10시로 설정하면 작업이 GMT 기준 오전 10시에서 오후 1시 사이에 다시 시작됩니다.

- ProfileName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업과 관련된 AWS Glue 사용 프로필의 이름.

ExecutionProperty 구조

작업의 실행 속성입니다.

필드

- MaxConcurrentRuns - 숫자(정수)입니다.

작업에 허용된 최대 동시 실행 수입니다. 기본 값은 1입니다. 이 임계값에 도달하면 오류가 반환됩니다. 지정할 수 있는 최대값은 서비스 제한에 따라 통제됩니다.

NotificationProperty 구조

알림의 구성 속성을 지정합니다.

필드

- `NotifyDelayAfter` – 1 이상의 숫자(정수)입니다.

작업 실행 시작 후 작업 실행 대기 알림을 전송하기 전까지 대기하는 시간(분)입니다.

JobCommand 구조

작업이 실행될 때 실행되는 코드를 지정합니다.

필드

- `Name` – UTF-8 문자열입니다.

작업 명령의 이름입니다. Apache Spark ETL 작업의 경우, `glueetl`이어야 합니다. Python 셸 작업의 경우, `pythonshell`이어야 합니다. Apache Spark 스트리밍 ETL 작업의 경우, `gluestreaming`이어야 합니다. Ray 작업의 경우 `glueray`이어야 합니다.

- `ScriptLocation` – 400,000바이트 이하 길이의 UTF-8 문자열입니다.

작업을 실행하는 스크립트의 Amazon Simple Storage Service(Amazon S3) 경로를 지정합니다.

- `PythonVersion` – [Custom string pattern #21](#)과(와) 일치하는 UTF-8 문자열입니다.

Python 셸 작업을 실행하는 데 사용되는 Python 버전입니다. 허용되는 값은 2 또는 3입니다.

- `Runtime` – [Custom string pattern #29](#)과 일치하는 64바이트 이하 길이의 UTF-8 문자열입니다.

Ray 작업에서 런타임은 사용자 환경에서 사용 가능한 Ray, Python 및 추가 라이브러리의 버전을 지정하는 데 사용됩니다. 이 필드는 다른 작업 유형에서 사용되지 않습니다. 지원되는 런타임 환경 값은 AWS Glue 개발자 안내서의 [지원되는 Ray 런타임 환경을](#) 참조하십시오.

ConnectionsList 구조

작업이 사용한 연결을 지정합니다.

필드

- `Connections` – UTF-8 문자열의 배열입니다.

작업이 사용한 연결 목록입니다.

JobUpdate 구조

기존 작업 정의 업데이트에 사용된 정보를 지정합니다. 이전 작업 정의를 이 정보로 완전히 덮어씁니다.

필드

- **JobMode** – UTF-8 문자열입니다(유효한 값: SCRIPT="" | VISUAL="" | NOTEBOOK="").

작업이 생성된 방법을 설명하는 모드입니다. 유효한 값은 다음과 같습니다.

- **SCRIPT**- AWS Glue Studio 스크립트 편집기를 사용하여 작업을 생성했습니다.
- **VISUAL**- AWS Glue Studio 비주얼 에디터를 사용하여 작업을 생성했습니다.
- **NOTEBOOK** - 대화형 세션 노트북을 사용하여 작업을 생성했습니다.

JobMode 필드가 없거나 null인 경우 기본값으로 SCRIPT가 할당됩니다.

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

정의된 작업에 대한 설명입니다.

- **LogUri** – UTF-8 문자열입니다.

이 필드는 향후 사용하기 위해 예약되어 있습니다.

- **Role** – UTF-8 문자열입니다.

이 작업과 연결된 IAM 역할의 이름 또는 Amazon 리소스 이름(ARN)입니다(필수).

- **ExecutionProperty** – [ExecutionProperty](#) 객체입니다.

ExecutionProperty는 이 작업에 허용된 최대 동시 실행 수를 지정합니다.

- **Command** – [JobCommand](#) 객체입니다.

이 작업을 실행하는 JobCommand입니다(필수).

- **DefaultArguments** – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

~~이 작업의 모든 실행에서 기본 인수(이름 값 페어로 지정됨)입니다.~~

작업 실행 스크립트에서 사용하는 인수와 자체적으로 소비하는 AWS Glue 인수를 여기에 지정할 수 있습니다.

작업 인수가 로깅될 수 있습니다. 일반 텍스트 보안 암호를 인수로 전달하지 마세요. Job 내에 암호를 보관하려는 경우 AWS Glue Connection AWS Secrets Manager 또는 기타 비밀 관리 메커니즘에서 암호를 검색하십시오.

자체 작업 인수를 지정하고 사용하는 방법에 대한 자세한 내용은 개발자 가이드의 [Python에서 AWS Glue Glue API](#) 호출을 참조하세요.

Spark 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [AWS Glue에서 사용하는 특별 파라미터](#) 주제를 참조하세요.

Ray 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [Ray 작업에서 작업 파라미터 사용](#)을 참조하세요.

- NonOverridableArguments – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

작업 실행 시 작업 인수를 제공할 때 재정의되지 않는 이 작업의 인수(이름 값 페어로 지정됨)입니다.

- Connections – [ConnectionsList](#) 객체입니다.

이 작업에 사용된 연결입니다.

- MaxRetries - 숫자(정수)입니다.

실패한 경우 이 작업을 다시 시도할 수 있는 최대 횟수입니다.

- AllocatedCapacity - 숫자(정수)입니다.

이 필드는 더 이상 사용되지 않습니다. 대신 MaxCapacity을 사용하세요.

이 작업에 할당할 AWS Glue 데이터 처리 장치 (DPU) 수. 최소 2DPU를 할당할 수 있습니다. 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

- Timeout – 1 이상의 숫자(정수)입니다.

작업 타임아웃(분)입니다. 작업을 실행하여 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 배치 작업의 경우 기본값은 2,880분(48시간)입니다.

스트리밍 작업의 시간 제한 값은 7일 또는 10,080분 미만이어야 합니다. 값을 비워 두면 유지 관리 기간을 설정하지 않았다고 가정하고 7일 후에 작업이 다시 시작됩니다. 유지 관리 기간을 설정한 경우 7일 후 유지 관리 기간 중에 작업이 다시 시작됩니다.

- MaxCapacity - 숫자(double)입니다.

Glue 버전 1.0 이하 작업의 경우 표준 작업자 유형을 사용하여 이 작업을 실행할 때 할당할 수 있는 AWS Glue 데이터 처리 장치 (DPU) 수를 계산합니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

Glue 버전 2.0 이상 작업의 경우 Maximum capacity를 지정할 수 없습니다. 그 대신 Worker type 및 Number of workers를 지정해야 합니다.

WorkerType 및 NumberOfWorkers를 사용하는 경우, MaxCapacity를 설정하지 마십시오.

MaxCapacity에 할당할 수 있는 값은 Python 셸 작업을 실행하는지 또는 Apache Spark ETL 작업 또는 Apache Spark 스트리밍 ETL 작업을 실행하는지에 따라 다릅니다.

- Python 셸 작업(JobCommand.Name="pythonshell")을 지정하면 0.0625 또는 1 DPU를 할당할 수 있습니다. 기본값은 0.0625 DPU입니다.
- Apache Spark ETL 작업(JobCommand.Name="glueetl") 또는 Apache Spark 스트리밍 ETL 작업(JobCommand.Name="gluestreaming")을 지정하면 2에서 100 DPU를 할당할 수 있습니다. 기본값은 10 DPU입니다. 이 작업 유형에는 부분적인 DPU 할당을 사용할 수 없습니다.
- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

작업이 실행될 때 할당되는 미리 정의된 작업자 유형입니다. Spark 작업에 대해 G.1X, G.2X, G.4X, G.8X 또는 G.025X의 값을 허용합니다. Ray 작업에 대해 Z.2X 값을 허용합니다.

- G.1X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 1개의 DPU(vCPU 4개, 메모리 16GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 77GB의 여유 공간)에서 2개의 DPU(vCPU 8개, 메모리 32GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.

- G.4X 작업자 유형에서 각 작업자는 256GB 디스크(약 235GB의 여유 공간)에서 4개의 DPU(vCPU 16개, 메모리 64GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 미국 동부 (오하이오), 미국 동부 (버지니아 북부), 미국 서부 (오레곤), 아시아 태평양 (싱가포르), 아시아 태평양 (시드니), 아시아 태평양 (도쿄), 캐나다 (중부), 유럽 (프랑크푸르트), 유럽 (아일랜드) 및 유럽 (스톡홀름) 의 AWS Glue 버전 3.0 이상 Spark ETL 작업에만 사용할 수 있습니다. AWS
- G.8X 작업자 유형에서 각 작업자는 512GB 디스크(약 487GB의 여유 공간)에서 8개의 DPU(vCPU 32개, 메모리 128GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 작업자 유형에 지원되는 AWS 지역과 동일한 지역에서 AWS Glue 버전 3.0 이상의 Spark ETL G.4X 작업에만 사용할 수 있습니다.
- G.025X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 0.25개의 DPU(vCPU 2개, 메모리 4GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 볼륨이 낮은 스트리밍 작업에 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 AWS Glue 버전 3.0 스트리밍 작업에만 사용할 수 있습니다.
- Z.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 120GB의 여유 공간)에서 2개의 M-DPU(vCPU 8개, 메모리 64GB)에 매핑되고, Autoscaler에 따라 최대 8개의 Ray 작업자를 제공합니다.
- NumberOfWorkers - 숫자(정수)입니다.

작업이 실행될 때 할당되는 정의된 workerType의 작업자 수입니다.

- SecurityConfiguration - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업에 사용할 SecurityConfiguration 구조의 이름입니다.

- NotificationProperty - [NotificationProperty](#) 객체입니다.

작업 알림의 구성 속성을 지정합니다.

- GlueVersion - [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Spark 작업에서 작업에서 AWS Glue 사용할 수 있는 Apache Spark 및 Python 버전을 GlueVersion 결정합니다. Python의 버전으로 Spark 유형의 작업에 대해 지원되는 버전을 확인할 수 있습니다.

Ray 작업에서는 GlueVersion을 4.0 이상으로 설정해야 합니다. 그러나 Ray 작업에서 사용할 수 있는 Ray, Python 및 추가 라이브러리의 버전은 Job 명령의 Runtime 파라미터에 의해 결정됩니다.

사용 가능한 AWS Glue 버전과 해당 Spark 및 Python 버전에 대한 자세한 내용은 개발자 안내서의 [Glue 버전을](#) 참조하십시오.

Glue 버전 지정 없이 생성된 작업은 Glue 0.9로 기본 지정됩니다.

- CodeGenConfigurationNodes – 카-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [CodeGenConfigurationNode](#) 객체입니다.

Glue Studio 시각적 구성 요소 및 Glue Studio 코드 생성의 기반이 되는 방향성 비순환 그래프의 표현입니다.

- ExecutionClass - 16바이트 미만의 UTF-8 문자열입니다(유효한 값: FLEX="" | STANDARD="").

작업이 표준 또는 유연한 실행 클래스로 실행되는지 여부를 나타냅니다. 표준 실행 클래스는 빠른 작업 시작 및 전용 리소스가 필요한 시간에 민감한 워크로드에 적합합니다.

유연한 실행 클래스는 시작 및 완료 시간이 다를 수 있는 시간에 민감하지 않은 작업에 적합합니다.

AWS Glue 버전 3.0 이상의 작업과 명령 glueetl 유형만 ExecutionClass 로 FLEX 설정할 수 있습니다. 유연한 실행 클래스는 Spark 작업에 사용할 수 있습니다.

- SourceControlDetails – [SourceControlDetails](#) 객체입니다.

작업에 대한 소스 제어 구성에 대한 세부 정보로, 원격 리포지토리와 작업 아티팩트 동기화를 허용합니다.

- MaintenanceWindow – [Custom string pattern #30](#)과(와) 일치하는 UTF-8 문자열입니다.

이 필드는 스트리밍 작업에 대한 유지 관리 기간의 요일과 시간을 지정합니다. AWS Glue 정기적으로 유지 관리 활동을 수행합니다. 이러한 유지 관리 기간 AWS Glue 동안에는 스트리밍 작업을 다시 시작해야 합니다.

AWS Glue 지정된 유지 관리 기간으로부터 3시간 이내에 작업을 다시 시작합니다. 예를 들어, 유지 관리 기간을 GMT 기준 월요일 오전 10시로 설정하면 작업이 GMT 기준 오전 10시에서 오후 1시 사이에 다시 시작됩니다.

- ProfileName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업과 관련된 AWS Glue 사용 프로필의 이름.

SourceControlDetails 구조

작업에 대한 소스 제어 구성에 대한 세부 정보로, 원격 리포지토리와 작업 아티팩트 동기화를 허용합니다.

필드

- Provider – UTF-8 문자열입니다.
원격 리포지토리의 공급자입니다.
- Repository – 1~512바이트 길이의 UTF-8 문자열입니다.
작업 아티팩트가 포함된 원격 리포지토리의 이름입니다.
- Owner – 1~512바이트 길이의 UTF-8 문자열입니다.
작업 아티팩트가 포함된 원격 리포지토리의 소유자입니다.
- Branch – 1~512바이트 길이의 UTF-8 문자열입니다.
원격 리포지토리의 선택적 브랜치입니다.
- Folder – 1~512바이트 길이의 UTF-8 문자열입니다.
원격 리포지토리의 선택적 폴더입니다.
- LastCommitId – 1~512바이트 길이의 UTF-8 문자열입니다.
원격 리포지토리의 커밋에 대한 마지막 커밋 ID입니다.
- LastSyncTimestamp – 1~512바이트 길이의 UTF-8 문자열입니다.
마지막 작업 동기화가 수행된 날짜와 시간입니다.
- AuthStrategy – UTF-8 문자열입니다.
인증 유형으로, AWS Secrets Manager에 저장된 인증 토큰 또는 개인용 액세스 토큰일 수 있습니다.
- AuthToken – 1~512바이트 길이의 UTF-8 문자열입니다.
인증 토큰의 값입니다.

운영

- [CreateJob 액션 \(Python: create_job\)](#)
- [UpdateJob 액션 \(Python: 업데이트_작업\)](#)
- [GetJob 액션 \(파이썬: get_job\)](#)
- [GetJobs 액션 \(파이썬: get_jobs\)](#)
- [DeleteJob 액션 \(Python: 삭제_작업\)](#)
- [ListJobs 액션 \(파이썬: 리스트_잡\)](#)
- [BatchGetJobs 액션 \(파이썬: batch_get_jobs\)](#)

CreateJob 액션 (Python: create_job)

새로운 작업 정의를 만듭니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업 정의에 할당하는 이름입니다. 계정에서 고유해야 합니다.

- JobMode – UTF-8 문자열입니다(유효한 값: SCRIPT="" | VISUAL="" | NOTEBOOK="").

작업이 생성된 방법을 설명하는 모드입니다. 유효한 값은 다음과 같습니다.

- SCRIPT- AWS Glue Studio 스크립트 편집기를 사용하여 작업을 생성했습니다.
- VISUAL- AWS Glue Studio 비주얼 에디터를 사용하여 작업을 생성했습니다.
- NOTEBOOK - 대화형 세션 노트북을 사용하여 작업을 생성했습니다.

JobMode 필드가 없거나 null인 경우 기본값으로 SCRIPT가 할당됩니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

정의된 작업에 대한 설명입니다.

- LogUri – UTF-8 문자열입니다.

이 필드는 향후 사용하기 위해 예약되어 있습니다.

- Role – 필수(Required): UTF-8 문자열입니다.

이 작업과 연결된 IAM 역할의 이름 또는 Amazon 리소스 이름(ARN)입니다.

- ExecutionProperty – [ExecutionProperty](#) 객체입니다.

ExecutionProperty는 이 작업에 허용된 최대 동시 실행 수를 지정합니다.

- Command – 필수(Required): [JobCommand](#) 객체입니다.

이 작업을 실행하는 JobCommand입니다.

- DefaultArguments – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

이 작업의 모든 실행에서 기본 인수(이름 값 페어로 지정됨)입니다.

작업 실행 스크립트에서 사용하는 인수와 자체적으로 소비하는 AWS Glue 인수를 여기에 지정할 수 있습니다.

작업 인수가 로깅될 수 있습니다. 일반 텍스트 보안 암호를 인수로 전달하지 마세요. Job 내에 암호를 보관하려는 경우 AWS Glue Connection AWS Secrets Manager 또는 기타 비밀 관리 메커니즘에서 암호를 검색하십시오.

자체 작업 인수를 지정하고 사용하는 방법에 대한 자세한 내용은 개발자 가이드의 [Python에서 AWS Glue API](#) 호출을 참조하세요.

Spark 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [AWS Glue에서 사용하는 특별 파라미터](#) 주제를 참조하세요.

Ray 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [Ray 작업에서 작업 파라미터 사용](#)을 참조하세요.

- NonOverridableArguments – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

작업 실행 시 작업 인수를 제공할 때 재정의되지 않는 이 작업의 인수(이름 값 페어로 지정됨)입니다.

- Connections – [ConnectionsList](#) 객체입니다.

이 작업에 사용된 연결입니다.

- `MaxRetries` - 숫자(정수)입니다.

실패한 경우 이 작업을 다시 시도할 수 있는 최대 횟수입니다.

- `AllocatedCapacity` - 숫자(정수)입니다.

이 파라미터는 이제 사용되지 않습니다. 대신 `MaxCapacity`을 사용하세요.

이 Job에 할당할 AWS Glue 데이터 처리 장치 (DPU) 수입니다. 최소 2DPU를 할당할 수 있습니다. 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

- `Timeout` - 1 이상의 숫자(정수)입니다.

작업 타임아웃(분)입니다. 작업을 실행하여 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 배치 작업의 경우 기본값은 2,880분(48시간)입니다.

스트리밍 작업의 시간 제한 값은 7일 또는 10,080분 미만이어야 합니다. 값을 비워 두면 유지 관리 기간을 설정하지 않았다고 가정하고 7일 후에 작업이 다시 시작됩니다. 유지 관리 기간을 설정한 경우 7일 후 유지 관리 기간 중에 작업이 다시 시작됩니다.

- `MaxCapacity` - 숫자(double)입니다.

Glue 버전 1.0 이하 작업의 경우 표준 작업자 유형을 사용하여 이 작업을 실행할 때 할당할 수 있는 AWS Glue 데이터 처리 장치 (DPU) 수를 계산합니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

Glue 버전 2.0 이상 작업의 경우 `Maximum capacity`를 지정할 수 없습니다. 그 대신 `Worker type` 및 `Number of workers`를 지정해야 합니다.

`WorkerType` 및 `NumberOfWorkers`를 사용하는 경우, `MaxCapacity`를 설정하지 마십시오.

`MaxCapacity`에 할당할 수 있는 값은 Python 셸 작업을 실행하는지 또는 Apache Spark ETL 작업 또는 Apache Spark 스트리밍 ETL 작업을 실행하는지에 따라 다릅니다.

- Python 셸 작업(`JobCommand.Name="pythonshell"`)을 지정하면 0.0625 또는 1 DPU를 할당할 수 있습니다. 기본값은 0.0625 DPU입니다.

- Apache Spark ETL 작업(JobCommand.Name="glueetl") 또는 Apache Spark 스트리밍 ETL 작업(JobCommand.Name="gluestreaming") 을 지정하면 2에서 100 DPU를 할당할 수 있습니다. 기본 값은 10 DPU입니다. 이 작업 유형에는 부분적인 DPU 할당을 사용할 수 없습니다.
- SecurityConfiguration – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업에 사용할 SecurityConfiguration 구조의 이름입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이 작업에서 사용할 태그입니다. 태그를 사용하여 작업에 대한 액세스를 제한할 수 있습니다. 태그에 대한 자세한 내용은 개발자 안내서의 [AWS 태그](#) 인을 참조하십시오. AWS Glue AWS Glue

- NotificationProperty – [NotificationProperty](#) 객체입니다.

작업 알림의 구성 속성을 지정합니다.

- GlueVersion – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Spark 작업에서 작업에서 AWS Glue 사용할 수 있는 Apache Spark 및 Python 버전을 GlueVersion 결정합니다. Python의 버전으로 Spark 유형의 작업에 대해 지원되는 버전을 확인할 수 있습니다.

Ray 작업에서는 GlueVersion을 4.0 이상으로 설정해야 합니다. 그러나 Ray 작업에서 사용할 수 있는 Ray, Python 및 추가 라이브러리의 버전은 Job 명령의 Runtime 파라미터에 의해 결정됩니다.

사용 가능한 AWS Glue 버전과 해당 Spark 및 Python 버전에 대한 자세한 내용은 개발자 안내서의 [Glue 버전을](#) 참조하십시오.

Glue 버전 지정 없이 생성된 작업은 Glue 0.9로 기본 지정됩니다.

- NumberOfWorkers - 숫자(정수)입니다.

작업이 실행될 때 할당되는 정의된 workerType의 작업자 수입니다.

- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

작업이 실행될 때 할당되는 미리 정의된 작업자 유형입니다. Spark 작업에 대해 G.1X, G.2X, G.4X, G.8X 또는 G.025X의 값을 허용합니다. Ray 작업에 대해 Z.2X 값을 허용합니다.

- G.1X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 1개의 DPU(vCPU 4개, 메모리 16GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 77GB의 여유 공간)에서 2개의 DPU(vCPU 8개, 메모리 32GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.4X 작업자 유형에서 각 작업자는 256GB 디스크(약 235GB의 여유 공간)에서 4개의 DPU(vCPU 16개, 메모리 64GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 미국 동부 (오하이오), 미국 동부 (버지니아 북부), 미국 서부 (오레곤), 아시아 태평양 (싱가포르), 아시아 태평양 (시드니), 아시아 태평양 (도쿄), 캐나다 (중부), 유럽 (프랑크푸르트), 유럽 (아일랜드) 및 유럽 (스톡홀름)의 AWS Glue 버전 3.0 이상 Spark ETL 작업에만 사용할 수 있습니다. AWS
- G.8X 작업자 유형에서 각 작업자는 512GB 디스크(약 487GB의 여유 공간)에서 8개의 DPU(vCPU 32개, 메모리 128GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 작업자 유형에 지원되는 AWS 지역과 동일한 지역에서 AWS Glue 버전 3.0 이상의 Spark ETL G.4X 작업에만 사용할 수 있습니다.
- G.025X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 0.25개의 DPU(vCPU 2개, 메모리 4GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 볼륨이 낮은 스트리밍 작업에 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 AWS Glue 버전 3.0 스트리밍 작업에만 사용할 수 있습니다.
- Z.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 120GB의 여유 공간)에서 2개의 M-DPU(vCPU 8개, 메모리 64GB)에 매핑되고, Autoscaler에 따라 최대 8개의 Ray 작업자를 제공합니다.
- CodeGenConfigurationNodes – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #39](#)과(와) 일치하는 UTF-8 문자열입니다.

각 값은 [CodeGenConfigurationNode](#) 객체입니다.

Glue Studio 시각적 구성 요소 및 Glue Studio 코드 생성의 기반이 되는 방향성 비순환 그래프의 표현입니다.

- `ExecutionClass` - 16바이트 미만의 UTF-8 문자열입니다(유효한 값: `FLEX=""` | `STANDARD=""`).

작업이 표준 또는 유연한 실행 클래스로 실행되는지 여부를 나타냅니다. 표준 실행 클래스는 빠른 작업 시작 및 전용 리소스가 필요한 시간에 민감한 워크로드에 적합합니다.

유연한 실행 클래스는 시작 및 완료 시간이 다를 수 있는 시간에 민감하지 않은 작업에 적합합니다.

AWS Glue 버전 3.0 이상 및 명령 유형의 `glueetl` 작업만 `ExecutionClass` 로 설정할 수 FLEX 있습니다. 유연한 실행 클래스는 Spark 작업에 사용할 수 있습니다.

- `SourceControlDetails` - [SourceControlDetails](#) 객체입니다.

작업에 대한 소스 제어 구성에 대한 세부 정보로, 원격 리포지토리와 작업 아티팩트 동기화를 허용합니다.

- `MaintenanceWindow` - [Custom string pattern #30](#)과(와) 일치하는 UTF-8 문자열입니다.

이 필드는 스트리밍 작업에 대한 유지 관리 기간의 요일과 시간을 지정합니다. AWS Glue 정기적으로 유지 관리 활동을 수행합니다. 이러한 유지 관리 기간 AWS Glue 동안에는 스트리밍 작업을 다시 시작해야 합니다.

AWS Glue 지정된 유지 관리 기간으로부터 3시간 이내에 작업을 다시 시작합니다. 예를 들어, 유지 관리 기간을 GMT 기준 월요일 오전 10시로 설정하면 작업이 GMT 기준 오전 10시에서 오후 1시 사이에 다시 시작됩니다.

- `ProfileName` - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업과 관련된 AWS Glue 사용 프로필의 이름.

응답

- `Name` - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업 정의를 위해 제공된 고유 이름입니다.

Errors

- `InvalidInputException`

- IdempotentParameterMismatchException
- AlreadyExistsException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

UpdateJob 액션 (Python: 업데이트_작업)

기존 작업 정의를 업데이트합니다. 이전 작업 정의를 이 정보로 완전히 덮어씁니다.

요청

- JobName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트할 작업 정의 이름입니다.

- JobUpdate – 필수(Required): [JobUpdate](#) 객체입니다.

작업 정의를 업데이트하여 값을 지정합니다. 지정되지 않은 구성은 제거되거나 기본값으로 재설정됩니다.

- ProfileName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업과 관련된 사용자 AWS Glue 프로필의 이름.

응답

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
업데이트된 작업 정의 이름을 반환합니다.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException

- `OperationTimeoutException`
- `ConcurrentModificationException`

GetJob 액션 (파이썬: `get_job`)

기존 작업 정의를 가져옵니다.

요청

- `JobName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

가져올 작업 정의 이름입니다.

응답

- `Job` – [작업](#) 객체입니다.

요청한 작업 정의.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

GetJobs 액션 (파이썬: `get_jobs`)

현재 모든 작업 정의를 가져옵니다.

요청

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

- `MaxResults` – 1~1,000의 숫자(정수)입니다.

응답의 최대 크기입니다.

응답

- Jobs – [작업](#) 객체의 배열입니다.

작업 정의 목록입니다.

- NextToken – UTF-8 문자열입니다.

모든 작업 정의가 반환하지 않은 경우의 지속 토큰입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

DeleteJob 액션 (Python: 삭제_작업)

지정한 작업 정의를 삭제합니다. 작업 정의를 못 찾으면 어떤 예외도 없습니다.

요청

- JobName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 작업 정의 이름입니다.

응답

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제된 작업 정의 이름입니다.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

ListJobs 액션 (파이썬: 리스트_잡)

이 AWS 계정의 모든 작업 리소스 또는 지정된 태그가 있는 리소스의 이름을 검색합니다. 이 작업을 통해 계정에서 사용 가능한 리소스와 그 이름을 확인할 수 있습니다.

이 작업을 수행하면 응답에서 필터로 사용할 수 있는 선택 사항인 `Tags` 필드가 검색되기 때문에 태그가 지정된 리소스를 하나의 그룹으로 검색할 수 있습니다. 태그 필터링을 사용하기로 선택하면 태그가 포함된 리소스만 검색됩니다.

요청

- `NextToken` – UTF-8 문자열입니다.
이것이 지속적인 요청이라면 지속적인 토큰입니다.
- `MaxResults` – 1~1,000의 숫자(정수)입니다.
반환할 목록의 최대 크기.
- `Tags` – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.
각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.
각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.
이렇게 태그가 지정된 리소스만 반환하도록 지정합니다.

응답

- `JobNames` – UTF-8 문자열의 배열입니다.
계정의 모든 작업 또는 지정된 태그를 가진 작업의 이름입니다.
- `NextToken` – UTF-8 문자열입니다.
반환된 목록이 사용가능한 마지막 지표를 포함하지 경우의 연속 토큰입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchGetJobs 액션 (파이썬: `batch_get_jobs`)

주어진 작업 이름 목록에 대한 리소스 메타데이터 목록을 반환합니다. `ListJobs` 작업을 호출한 후에는 권한이 부여된 데이터에 액세스하기 위해 이 작업을 호출할 수 있습니다. 이 작업은 태그를 사용하는 권한 조건을 포함해 모든 IAM 권한을 지원합니다.

요청

- `JobNames` – 필수(Required): UTF-8 문자열의 배열입니다.

작업 이름(`ListJobs` 작업에서 반환된 이름일 수 있음)의 목록입니다.

응답

- `Jobs` – [작업](#) 객체의 배열입니다.

작업 정의 목록입니다.

- `JobsNotFound` – UTF-8 문자열의 배열입니다.

찾을 수 없는 작업의 이름 목록입니다.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

작업 실행

Jobs Runs API는 에서 작업 실행 시작, 중지 또는 보기, 작업 북마크 재설정과 관련된 데이터 유형 및 API를 설명합니다. AWS Glue워크플로 및 작업 실행의 경우 작업 실행 기록을 90일 동안 액세스할 수 있습니다.

데이터 타입

- [JobRun 구조](#)
- [이전 구조](#)
- [JobBookmarkEntry 구조](#)
- [BatchStopJobRunSuccessfulSubmission 구조](#)
- [BatchStopJobRunError 구조](#)
- [NotificationProperty 구조](#)

JobRun 구조

작업 실행에 관한 정보를 포함합니다.

필드

- Id – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 작업 실행의 ID.
- Attempt - 숫자(정수)입니다.
이 작업을 실행하고자 시도했던 수입니다.
- PreviousRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 작업의 이전 실행 ID입니다. 예를 들어 StartJobRun 작업에 지정된 JobRunId입니다.
- TriggerName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 작업을 시작한 트리거의 이름입니다.
- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업에 사용된 작업 정의 이름입니다.

- JobMode – UTF-8 문자열입니다(유효한 값: SCRIPT="" | VISUAL="" | NOTEBOOK="").

작업이 생성된 방법을 설명하는 모드입니다. 유효한 값은 다음과 같습니다.

- SCRIPT- AWS Glue Studio 스크립트 편집기를 사용하여 작업을 생성했습니다.
- VISUAL- AWS Glue Studio 비주얼 에디터를 사용하여 작업을 생성했습니다.
- NOTEBOOK - 대화형 세션 노트북을 사용하여 작업을 생성했습니다.

JobMode 필드가 없거나 null인 경우 기본값으로 SCRIPT가 할당됩니다.

- StartedOn – 타임스탬프입니다.

이 작업이 시작된 날짜 및 시간.

- LastModifiedOn – 타임스탬프입니다.

이 작업 실행이 수정된 마지막 시간입니다.

- CompletedOn – 타임스탬프입니다.

이 작업 실행이 완료된 날짜 및 시간입니다.

- JobRunState – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED).

작업 실행의 현재 상태입니다. 비정상적으로 종료된 작업의 상태에 대한 자세한 내용은 [AWS Glue 작업 실행 상태](#)를 참조하세요.

- Arguments – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

작업과 연결되어 있는 작업 인수입니다. 이 작업 실행에서 작업 정의 자체에 설정된 기본 인수를 바꿉니다.

작업 실행 스크립트에서 사용하는 인수와 자체적으로 소비하는 AWS Glue 인수를 여기에 지정할 수 있습니다.

작업 인수가 로깅될 수 있습니다. 일반 텍스트 보안 암호를 인수로 전달하지 마세요. Job 내에 암호를 보관하려는 경우 AWS Glue Connection AWS Secrets Manager 또는 기타 비밀 관리 메커니즘에서 암호를 검색하십시오.

자체 작업 인수를 지정하고 사용하는 방법에 대한 자세한 내용은 개발자 가이드의 [Python에서 AWS Glue Glue API 호출을 참조하세요.](#)

Spark 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [AWS Glue에서 사용하는 특별 파라미터](#) 주제를 참조하세요.

Ray 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [Ray 작업에서 작업 파라미터 사용](#)을 참조하세요.

- ErrorMessage – UTF-8 문자열입니다.

이 작업의 실행과 연결된 오류 메시지입니다.

- PredecessorRuns – [이전 기록](#) 객체의 배열입니다.

이 작업을 실행하기 이전의 기록입니다.

- AllocatedCapacity - 숫자(정수)입니다.

이 필드는 더 이상 사용되지 않습니다. 대신 MaxCapacity을 사용하세요.

여기에 할당된 AWS Glue 데이터 처리 장치 (DPU) 수. JobRun 2~100DPU가 할당될 수 있습니다. 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

- ExecutionTime - 숫자(정수)입니다.

이 작업이 리소스를 소비한 시간(초).

- Timeout – 1 이상의 숫자(정수)입니다.

JobRun 제한 시간(분)입니다. 작업을 실행하여 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 이 값은 상위 작업에 설정된 제한 시간 값을 재정의합니다.

스트리밍 작업의 시간 제한 값은 7일 또는 10,080분 미만이어야 합니다. 값을 비워 두면 유지 관리 기간을 설정하지 않았다고 가정하고 7일 후에 작업이 다시 시작됩니다. 유지 관리 기간을 설정한 경우 7일 후 유지 관리 기간 중에 작업이 다시 시작됩니다.

- MaxCapacity - 숫자(double)입니다.

Glue 버전 1.0 이하 작업의 경우 표준 작업자 유형을 사용하여 이 작업을 실행할 때 할당할 수 있는 AWS Glue 데이터 처리 장치 (DPU) 수를 계산합니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

Glue 버전 2.0 이상 작업의 경우 Maximum capacity를 지정할 수 없습니다. 그 대신 Worker type 및 Number of workers를 지정해야 합니다.

WorkerType 및 NumberOfWorkers를 사용하는 경우, MaxCapacity를 설정하지 마십시오.

MaxCapacity에 할당할 수 있는 값은 Python 셸 작업을 실행하는지 또는 Apache Spark ETL 작업 또는 Apache Spark 스트리밍 ETL 작업을 실행하는지에 따라 다릅니다.

- Python 셸 작업(JobCommand.Name="pythonshell")을 지정하면 0.0625 또는 1 DPU를 할당할 수 있습니다. 기본값은 0.0625 DPU입니다.
- Apache Spark ETL 작업(JobCommand.Name="glueetl") 또는 Apache Spark 스트리밍 ETL 작업(JobCommand.Name="gluestreaming")을 지정하면 2에서 100 DPU를 할당할 수 있습니다. 기본값은 10 DPU입니다. 이 작업 유형에는 부분적인 DPU 할당을 사용할 수 없습니다.
- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

작업이 실행될 때 할당되는 미리 정의된 작업자 유형입니다. Spark 작업에 대해 G.1X, G.2X, G.4X, G.8X 또는 G.025X의 값을 허용합니다. Ray 작업에 대해 Z.2X 값을 허용합니다.

- G.1X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 1개의 DPU(vCPU 4개, 메모리 16GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 77GB의 여유 공간)에서 2개의 DPU(vCPU 8개, 메모리 32GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.4X 작업자 유형에서 각 작업자는 256GB 디스크(약 235GB의 여유 공간)에서 4개의 DPU(vCPU 16개, 메모리 64GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 미국 동부 (오하이오), 미국 동부 (버지니아 북부), 미국 서부 (오레곤), 아시아 태평양 (싱가포르), 아시아 태평양 (시드니), 아시아 태평양 (도쿄), 캐나다 (중부), 유럽 (프랑크푸르트), 유럽 (아일랜드) 및 유럽 (스톡홀름) 의 AWS Glue 버전 3.0 이상 Spark ETL 작업에만 사용할 수 있습니다. AWS
- G.8X 작업자 유형에서 각 작업자는 512GB 디스크(약 487GB의 여유 공간)에서 8개의 DPU(vCPU 32개, 메모리 128GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작

업자 유형은 작업자 유형에 지원되는 AWS 지역과 동일한 지역에서 AWS Glue 버전 3.0 이상의 Spark ETL G.4X 작업에만 사용할 수 있습니다.

- G.025X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 0.25개의 DPU(vCPU 2개, 메모리 4GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 볼륨이 낮은 스트리밍 작업에 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 AWS Glue 버전 3.0 스트리밍 작업에만 사용할 수 있습니다.
- Z.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 120GB의 여유 공간)에서 2개의 M-DPU(vCPU 8개, 메모리 64GB)에 매핑되고, Autoscaler에 따라 최대 8개의 Ray 작업자를 제공합니다.

- NumberOfWorkers - 숫자(정수)입니다.

작업이 실행될 때 할당되는 정의된 workerType의 작업자 수입니다.

- SecurityConfiguration - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업 실행에 사용할 SecurityConfiguration 구조의 이름입니다.

- LogGroupName - UTF-8 문자열입니다.

CloudWatch Amazon에서 사용하여 서버 측에서 암호화할 수 있는 보안 로깅용 로그 그룹의 이름입니다. AWS KMS이 이름은 /aws-glue/jobs/일 수 있으며, 그 기본 암호화는 NONE입니다. 역할 이름과 SecurityConfiguration 이름(즉, /aws-glue/jobs-yourRoleName-yourSecurityConfigurationName/)을 추가하면 로그 그룹 암호화에 이 보안 구성이 사용됩니다.

- NotificationProperty - [NotificationProperty](#) 객체입니다.

작업 실행 알림의 구성 속성을 지정합니다.

- GlueVersion - [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Spark 작업에서 작업에서 AWS Glue 사용할 수 있는 Apache Spark 및 Python 버전을 GlueVersion 결정합니다. Python의 버전으로 Spark 유형의 작업에 대해 지원되는 버전을 확인할 수 있습니다.

Ray 작업에서는 GlueVersion을 4.0 이상으로 설정해야 합니다. 그러나 Ray 작업에서 사용할 수 있는 Ray, Python 및 추가 라이브러리의 버전은 Job 명령의 Runtime 파라미터에 의해 결정됩니다.

사용 가능한 AWS Glue 버전과 해당 Spark 및 Python 버전에 대한 자세한 내용은 개발자 안내서의 [Glue 버전을](#) 참조하십시오.

Glue 버전 지정 없이 생성된 작업은 Glue 0.9로 기본 지정됩니다.

- DPUSeconds - 숫자(double)입니다.

이 필드는 실행 클래스가 FLEX인 작업 실행이거나 Auto Scaling이 활성화된 경우에 설정할 수 있으며, 작업 실행 수명 주기 동안 각 실행기가 실행된 총 시간(초)에 DPU 계수(작업자는 1(G.1X), 2(G.2X) 또는 0.25(G.025X))를 곱한 값을 나타냅니다. 이 값은 Auto Scaling 작업의 경우처럼 `executionEngineRuntime * MaxCapacity`와 다를 수 있습니다. 지정된 시간에 실행 중인 실행기 수가 `MaxCapacity`보다 작을 수 있기 때문입니다. 따라서, DPUSeconds 값이 `executionEngineRuntime * MaxCapacity`보다 작을 수 있습니다.

- ExecutionClass - 16바이트 미만의 UTF-8 문자열입니다(유효한 값: FLEX="" | STANDARD="").

작업이 표준 또는 유연한 실행 클래스로 실행되는지 여부를 나타냅니다. 표준 실행 클래스는 빠른 작업 시작 및 전용 리소스가 필요한 시간에 민감한 워크로드에 적합합니다.

유연한 실행 클래스는 시작 및 완료 시간이 다를 수 있는 시간에 민감하지 않은 작업에 적합합니다.

AWS Glue 버전 3.0 이상의 작업과 명령 `glueetl` 유형만 ExecutionClass 로 FLEX 설정할 수 있습니다. 유연한 실행 클래스는 Spark 작업에 사용할 수 있습니다.

- MaintenanceWindow - [Custom string pattern #30](#)과(와) 일치하는 UTF-8 문자열입니다.

이 필드는 스트리밍 작업에 대한 유지 관리 기간의 요일과 시간을 지정합니다. AWS Glue 정기적으로 유지 관리 활동을 수행합니다. 이러한 유지 관리 기간 AWS Glue 동안에는 스트리밍 작업을 다시 시작해야 합니다.

AWS Glue 지정된 유지 관리 기간으로부터 3시간 이내에 작업을 다시 시작합니다. 예를 들어, 유지 관리 기간을 GMT 기준 월요일 오전 10시로 설정하면 작업이 GMT 기준 오전 10시에서 오후 1시 사이에 다시 시작됩니다.

- ProfileName - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업 실행과 관련된 AWS Glue 사용 프로필의 이름.

이전 구조

이 작업을 실행할 수 있도록 조건적 트리거를 조건으로 한 작업입니다.

필드

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 이전 작업에 사용된 작업 정의 이름입니다.
- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 작업의 이전 작업 실행 ID입니다.

JobBookmarkEntry 구조

작업이 프로세싱을 다시 시작할 수 있는 포인트를 정의합니다.

필드

- JobName – UTF-8 문자열입니다.
문제의 작업 이름입니다.
- Version - 숫자(정수)입니다.
작업의 버전입니다.
- Run - 숫자(정수)입니다.
실행 ID 숫자입니다.
- Attempt - 숫자(정수)입니다.
시도 ID 숫자입니다.
- PreviousRunId – UTF-8 문자열입니다.
이전의 작업 실행과 연결된 고유의 실행 식별자입니다.
- RunId – UTF-8 문자열입니다.
실행 ID 숫자입니다.
- JobBookmark – UTF-8 문자열입니다.
그 자체를 즐겨찾습니다.

BatchStopJobRunSuccessfulSubmission 구조

지정된 JobRun을 중지한 성공적인 요청을 기록합니다.

필드

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 작업이 중지된 작업 정의 이름입니다.
- JobRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
중지된 작업 실행의 JobRunId입니다.

BatchStopJobRunError 구조

지정된 작업을 중지하고자 할 때 발생하는 오류를 기록합니다.

필드

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
문제의 작업 실행에 사용된 작업 정의 이름입니다.
- JobRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
문제의 작업 실행 JobRunId입니다.
- ErrorDetail – [ErrorDetail](#) 객체입니다.
발생한 오류에 대한 추가 세부 정보를 지정합니다.

NotificationProperty 구조

알림의 구성 속성을 지정합니다.

필드

- NotifyDelayAfter – 1 이상의 숫자(정수)입니다.
작업 실행 시작 후 작업 실행 대기 알림을 전송하기 전까지 대기하는 시간(분)입니다.

운영

- [StartJobRun 액션 \(Python: 시작_작업_실행\)](#)
- [BatchStopJobRun 액션 \(Python: 배치_중지_작업_실행\)](#)
- [GetJobRun 액션 \(파이썬: get_job_run\)](#)
- [GetJobRuns 액션 \(파이썬: get_job_runs\)](#)
- [GetJobBookmark 액션 \(파이썬: get_job_bookmark\)](#)
- [GetJobBookmarks 액션 \(Python: get_job_bookmark\)](#)
- [ResetJobBookmark 액션 \(Python: reset_job_bookmark\)](#)

StartJobRun 액션 (Python: 시작_작업_실행)

작업 정의를 사용한 작업 시작하기

요청

- JobName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

사용할 작업 정의의 이름입니다.

- JobRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

재시도할 이전 JobRun의 ID입니다.

- Arguments – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

작업과 연결되어 있는 작업 인수입니다. 이 작업 실행에서 작업 정의 자체에 설정된 기본 인수를 바꿉니다.

작업 실행 스크립트에서 사용하는 인수와 자체적으로 소비하는 인수를 여기에 지정할 수 있습니다.
AWS Glue

작업 인수가 로깅될 수 있습니다. 일반 텍스트 보안 암호를 인수로 전달하지 마세요. Job 내에 암호를 보관하려는 경우 AWS Glue Connection AWS Secrets Manager 또는 기타 비밀 관리 메커니즘에서 암호를 검색하십시오.

자체 작업 인수를 지정하고 사용하는 방법에 대한 자세한 내용은 개발자 가이드의 [Python에서 AWS Glue Glue API 호출을 참조](#)하세요.

Spark 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [AWS Glue에서 사용하는 특별 파라미터](#) 주제를 참조하세요.

Ray 작업을 구성할 때 이 필드에 제공할 수 있는 인수에 대한 자세한 내용은 개발자 안내서의 [Ray 작업에서 작업 파라미터 사용](#)을 참조하세요.

- `AllocatedCapacity` - 숫자(정수)입니다.

이 필드는 더 이상 사용되지 않습니다. 대신 `MaxCapacity`을 사용하세요.

여기에 할당할 AWS Glue 데이터 처리 장치 (DPU) 수. JobRun 최소 2DPU를 할당할 수 있습니다. 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

- `Timeout` - 1 이상의 숫자(정수)입니다.

JobRun 제한 시간(분)입니다. 작업을 실행하여 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 이 값은 상위 작업에 설정된 제한 시간 값을 재정의합니다.

스트리밍 작업의 시간 제한 값은 7일 또는 10,080분 미만이어야 합니다. 값을 비워 두면 유지 관리 기간을 설정하지 않았다고 가정하고 7일 후에 작업이 다시 시작됩니다. 유지 관리 기간을 설정한 경우 7일 후 유지 관리 기간 중에 작업이 다시 시작됩니다.

- `MaxCapacity` - 숫자(double)입니다.

Glue 버전 1.0 이하 작업의 경우 표준 작업자 유형을 사용하여 이 작업을 실행할 때 할당할 수 있는 AWS Glue 데이터 처리 장치 (DPU) 수를 계산합니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

Glue 버전 2.0 이상 작업의 경우 `Maximum capacity`를 지정할 수 없습니다. 그 대신 `Worker type` 및 `Number of workers`를 지정해야 합니다.

`WorkerType` 및 `NumberOfWorkers`를 사용하는 경우, `MaxCapacity`를 설정하지 마십시오.

`MaxCapacity`에 할당할 수 있는 값은 Python 셸 작업을 실행하는지 또는 Apache Spark ETL 작업 또는 Apache Spark 스트리밍 ETL 작업을 실행하는지에 따라 다릅니다.

- Python 셸 작업(`JobCommand.Name="pythonshell"`)을 지정하면 0.0625 또는 1 DPU를 할당할 수 있습니다. 기본값은 0.0625 DPU입니다.
- Apache Spark ETL 작업(`JobCommand.Name="glueetl"`) 또는 Apache Spark 스트리밍 ETL 작업(`JobCommand.Name="gluestreaming"`)을 지정하면 2에서 100 DPU를 할당할 수 있습니다. 기본값은 10 DPU입니다. 이 작업 유형에는 부분적인 DPU 할당을 사용할 수 없습니다.
- SecurityConfiguration – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업 실행에 사용할 SecurityConfiguration 구조의 이름입니다.

- NotificationProperty – [NotificationProperty](#) 객체입니다.

작업 실행 알림의 구성 속성을 지정합니다.

- WorkerType – UTF-8 문자열입니다(유효한 값: `Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X=""`).

작업이 실행될 때 할당되는 미리 정의된 작업자 유형입니다. Spark 작업에 대해 G.1X, G.2X, G.4X, G.8X 또는 G.025X의 값을 허용합니다. Ray 작업에 대해 Z.2X 값을 허용합니다.

- G.1X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 1개의 DPU(vCPU 4개, 메모리 16GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 77GB의 여유 공간)에서 2개의 DPU(vCPU 8개, 메모리 32GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.4X 작업자 유형에서 각 작업자는 256GB 디스크(약 235GB의 여유 공간)에서 4개의 DPU(vCPU 16개, 메모리 64GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 미국 동부 (오하이오), 미국 동부 (버지니아 북부), 미국 서부 (오레곤), 아시아 태평양 (싱가포르), 아시아 태평양 (시드니), 아시아 태평양 (도쿄), 캐나다 (중부), 유럽 (프랑크푸르트), 유럽 (아일랜드) 및 유럽 (스톡홀름)의 AWS Glue 버전 3.0 이상 Spark ETL 작업에만 사용할 수 있습니다. AWS
- G.8X 작업자 유형에서 각 작업자는 512GB 디스크(약 487GB의 여유 공간)에서 8개의 DPU(vCPU 32개, 메모리 128GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작

업자 유형은 작업자 유형에 지원되는 AWS 지역과 동일한 지역에서 AWS Glue 버전 3.0 이상의 Spark ETL G.4X 작업에만 사용할 수 있습니다.

- G.025X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 0.25개의 DPU(vCPU 2개, 메모리 4GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 볼륨이 낮은 스트리밍 작업에 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 AWS Glue 버전 3.0 스트리밍 작업에만 사용할 수 있습니다.
- Z.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 120GB의 여유 공간)에서 2개의 M-DPU(vCPU 8개, 메모리 64GB)에 매핑되고, Autoscaler에 따라 최대 8개의 Ray 작업자를 제공합니다.
- NumberOfWorkers - 숫자(정수)입니다.

작업이 실행될 때 할당되는 정의된 workerType의 작업자 수입니다.

- ExecutionClass - 16바이트 미만의 UTF-8 문자열입니다(유효한 값: FLEX="" | STANDARD="").

작업이 표준 또는 유연한 실행 클래스로 실행되는지 여부를 나타냅니다. 표준 실행 클래스는 빠른 작업 시작 및 전용 리소스가 필요한 시간에 민감한 워크로드에 적합합니다.

유연한 실행 클래스는 시작 및 완료 시간이 다를 수 있는 시간에 민감하지 않은 작업에 적합합니다.

AWS Glue 버전 3.0 이상 및 명령 유형의 glueetl 작업만 ExecutionClass 로 설정할 수 FLEX 있습니다. 유연한 실행 클래스는 Spark 작업에 사용할 수 있습니다.

- ProfileName - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업 실행과 관련된 AWS Glue 사용 프로필의 이름.

응답

- JobRunId - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업 실행에 할당된 ID.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException

- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentRunsExceededException`

BatchStopJobRun 액션 (Python: 배치_중지_작업_실행)

지정된 작업을 정의하기 위해서 하나 이상의 작업을 중지합니다.

요청

- `JobName` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업이 실행되지 않고 중지하기 위한 작업 정의 이름입니다.

- `JobRunIds` – 필수(Required): 1~25개 문자열의 UTF-8 문자열의 배열입니다.

이 작업을 정의하기 위해서 중지되어야 하는 `JobRunIds`의 목록입니다.

응답

- `SuccessfulSubmissions` – [BatchStopJobRunSuccessfulSubmission](#) 객체의 배열입니다.

중지를 위해 성공적으로 제출된 목록의 목록입니다. `JobRuns`

- `Errors` – [BatchStopJobRunError](#) 객체의 배열입니다.

`JobRuns`를 중지할 때 발생하는 오류 목록은 오류에 대한 상세 정보와 함께 각 오류가 발생한 `JobRunId`를 포함합니다.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

GetJobRun 액션 (파이썬: get_job_run)

작업 실행 시 메타데이터 가져오기 워크플로 및 작업 실행의 경우 작업 실행 기록을 90일 동안 액세스할 수 있습니다.

요청

- JobName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행 중인 작업 정의의 이름입니다.

- RunId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업 실행의 ID.

- PredecessorsIncluded – 부울입니다.

이전 작업 실행 목록이 반환되어야 하면 True입니다.

응답

- JobRun – [JobRun](#) 객체입니다.

요청한 작업 실행 메타데이터.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobRuns 액션 (파이썬: get_job_runs)

주어진 작업 정의의 모든 실행 시 메타데이터 가져오기

요청

- JobName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

모든 작업을 실행하기 위한 작업 정의 이름입니다.

- NextToken – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

- MaxResults - 1 이상 200 이하의 숫자(정수)입니다.

응답의 최대 크기입니다.

응답

- JobRuns – [JobRun](#) 객체의 배열입니다.

작업 실행 메타데이터 객체의 목록입니다.

- NextToken – UTF-8 문자열입니다.

요청된 모든 작업 실행이 반환하지 않은 경우의 지속 토큰입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetJobBookmark 액션 (파이썬: get_job_bookmark)

작업 북마크 항목에 대한 정보를 반환합니다.

작업 북마크에 대한 자세한 내용을 알아보려면 다음을 참조하세요.

- [처리된 데이터를 작업 북마크로 추적](#)
- [에서 사용하는 Job 매개변수 AWS Glue](#)

- [작업 구조](#)

요청

- JobName – 필수(Required): UTF-8 문자열입니다.

문제의 작업 이름입니다.

- Version - 숫자(정수)입니다.

작업의 버전입니다.

- RunId – UTF-8 문자열입니다.

이 작업 실행과 연결된 고유의 실행 식별자입니다.

응답

- JobBookmarkEntry – [JobBookmarkEntry](#) 객체입니다.

작업이 처리를 다시 시작할 수 있는 포인트를 정의하는 구조입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ValidationException

GetJobBookmarks 액션 (Python: get_job_bookmark)

작업 북마크 항목에 대한 정보를 반환합니다. 목록은 버전 번호가 감소되는 순서로 정렬됩니다.

작업 북마크에 대한 자세한 내용을 알아보려면 다음을 참조하세요.

- [처리된 데이터를 작업 북마크로 추적](#)
- [에서 사용하는 Job 매개변수 AWS Glue](#)
- [작업 구조](#)

요청

- JobName – 필수(Required): UTF-8 문자열입니다.

문제의 작업 이름입니다.

- MaxResults - 숫자(정수)입니다.

응답의 최대 크기입니다.

- NextToken - 숫자(정수)입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- JobBookmarkEntries – [JobBookmarkEntry](#) 객체의 배열입니다.

작업이 처리를 다시 시작할 수 있는 포인트를 정의하는 작업 북마크 항목의 목록입니다.

- NextToken - 숫자(정수)입니다.

모든 항목이 반환된 경우에는 1, 요청된 작업 실행 중 일부가 반환되지 않은 경우 1보다 큰 값이 있는 연속 토큰입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

ResetJobBookmark 액션 (Python: reset_job_bookmark)

북마크 입력 재설정

작업 북마크에 대한 자세한 내용을 알아보려면 다음을 참조하세요.

- [처리된 데이터를 작업 북마크로 추적](#)
- [에서 사용하는 Job 매개변수 AWS Glue](#)

- [작업 구조](#)

요청

- JobName – 필수(Required): UTF-8 문자열입니다.

문제의 작업 이름입니다.

- RunId – UTF-8 문자열입니다.

이 작업 실행과 연결된 고유의 실행 식별자입니다.

응답

- JobBookmarkEntry – [JobBookmarkEntry](#) 객체입니다.

북마크 입력 재설정

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

트리거

트리거 API는 AWS Glue에서의 작업 트리거 생성, 업데이트 또는 삭제 및 시작/종지와 관련된 API 및 데이터 유형에 대해 설명합니다.

데이터 타입

- [트리거 구조](#)
- [TriggerUpdate 구조](#)
- [조건자 구조](#)
- [조건 구조](#)
- [작업 구조](#)

- [EventBatchingCondition 구조](#)

트리거 구조

특정 트리거에 대한 정보

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
트리거의 이름입니다.
- WorkflowName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
트리거와 연결된 워크플로의 이름입니다.
- Id – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
추후 사용 예약.
- Type – UTF-8 문자열입니다(유효 값: SCHEDULED | CONDITIONAL | ON_DEMAND | EVENT).
트리거 유형입니다.
- State – UTF-8 문자열입니다(유효 값: CREATING | CREATED | ACTIVATING | ACTIVATED | DEACTIVATING | DEACTIVATED | DELETING | UPDATING).
트리거 현재 테이블 상태
- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.
이 트리거에 대한 설명입니다.
- Schedule – UTF-8 문자열입니다.
일정을 지정하는 데 사용되는 cron 표현식입니다([작업 및 크롤러의 시간 기반 일정](#) 참조). 예를 들어, 매일 오후 12시 15분(UTC)에 실행하려면 cron(15 12 * * ? *)을 지정합니다.
- Actions – [작업](#) 객체의 배열입니다.
이 트리거에 의해 시작되는 작업
- Predicate – [조건자](#) 객체입니다.
트리거가 실행되는 시기를 결정하는 이 트리거의 조건자입니다.

- EventBatchingCondition – [EventBatchingCondition](#) 객체입니다.

EventBridge 이벤트 트리거가 실행되기 전에 충족되어야 하는 배치 조건(수신된 이벤트의 지정된 수 또는 배치 기간이 만료됨).

TriggerUpdate 구조

트리거를 업데이트하기 위해 정보를 제공하는 데 사용되는 구조입니다. 이 객체는 이전 트리거 정의를 완전히 덮어써서 업데이트합니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

추후 사용 예약.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

이 트리거에 대한 설명입니다.

- Schedule – UTF-8 문자열입니다.

일정을 지정하는 데 사용되는 cron 표현식입니다([작업 및 크롤러의 시간 기반 일정](#) 참조). 예를 들어, 매일 오후 12시 15분(UTC)에 실행하려면 cron(15 12 * * ? *)을 지정합니다.

- Actions – [작업](#) 객체의 배열입니다.

이 트리거에 의해 시작되는 작업

- Predicate – [조건자](#) 객체입니다.

트리거가 실행되는 시기를 결정하는 이 트리거의 조건자입니다.

- EventBatchingCondition – [EventBatchingCondition](#) 객체입니다.

EventBridge 이벤트 트리거가 실행되기 전에 충족되어야 하는 배치 조건(수신된 이벤트의 지정된 수 또는 배치 기간이 만료됨).

조건자 구조

트리거가 실행되는 시기를 결정하는 트리거의 조건자를 정의합니다.

필드

- Logical – UTF-8 문자열입니다(유효한 값: AND | ANY).

오직 하나의 조건이 기록되어 있다면 조건부 필드입니다. 다수의 조건이 있을 때 이 필드가 필요합니다.

- Conditions – [Condition](#) 객체의 배열입니다.

트리거가 언제 발생하는지 확인하는 조건 목록입니다.

조건 구조

어떤 트리거가 시작되는지에 대한 조건을 결정합니다.

필드

- LogicalOperator – UTF-8 문자열입니다(유효한 값: EQUALS).

논리 연산자

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

JobRuns에 조건이 적용되고 트리거가 대기하는 작업의 이름입니다.

- State – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED).

조건 상태입니다. 현재 트리거가 수신할 수 있는 유일한 작업은 SUCCEEDED, STOPPED, FAILED 및 TIMEOUT입니다. 트리거가 수신할 수 있는 유일한 Crawler는 SUCCEEDED, FAILED 및 CANCELLED입니다.

- CrawlerName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 조건이 적용되는 크롤러의 이름입니다.

- CrawlState – UTF-8 문자열입니다(유효한 값: RUNNING | CANCELLING | CANCELLED | SUCCEEDED | FAILED | ERROR).

이 조건이 적용되는 크롤러의 상태입니다.

작업 구조

트리거에 의해 시작한 작업을 결정합니다.

필드

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행할 작업의 이름입니다.

- Arguments – 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

이 트리거가 작동할 때 사용되는 작업 인수입니다. 이 작업 실행에서 작업 정의 자체에 설정된 기본 인수를 바꿉니다.

AWS Glue 자체가 사용하는 인수는 물론 사용자의 작업 실행 스크립트가 사용하는 인수를 지정할 수 있습니다.

자체 작업 인수를 지정하고 사용하는 방법에 대한 자세한 내용은 개발자 가이드의 [Python에서 AWS Glue Glue API](#) 호출을 참조하세요.

AWS Glue가 작업을 설정하는 데 사용하는 키 값 페어에 대한 자세한 내용은 개발자 가이드의 [AWS Glue가 사용하는 특정 파라미터](#)를 참조하세요.

- Timeout – 1 이상의 숫자(정수)입니다.

JobRun 제한 시간(분)입니다. 작업을 실행하여 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 기본값은 2,880 분(48 시간)입니다. 상위 작업에 설정된 제한 시간 값을 재정의합니다.

- SecurityConfiguration – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업에 사용할 SecurityConfiguration 구조의 이름입니다.

- NotificationProperty – [NotificationProperty](#) 객체입니다.

작업 실행 알림의 구성 속성을 지정합니다.

- CrawlerName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업에 사용할 크롤러의 이름입니다.

EventBatchingCondition 구조

EventBridge 이벤트 트리거가 실행되기 전에 충족되어야 하는 배치 조건(수신된 이벤트의 지정된 수 또는 배치 기간이 만료됨).

필드

- **BatchSize** – 필수(Required): 1~100의 숫자(정수)입니다.

EventBridge 이벤트 트리거가 실행되기 전에 Amazon EventBridge에서 수신해야 하는 이벤트 수입니다.

- **BatchWindow** – 1~900의 숫자(정수)입니다.

EventBridge 이벤트 트리거가 실행된 후의 기간(초)입니다. 첫 번째 이벤트가 수신되면 기간이 시작됩니다.

운영

- [CreateTrigger 작업\(Python: create_trigger\)](#)
- [StartTrigger 작업\(Python: start_trigger\)](#)
- [GetTrigger 작업\(Python: get_trigger\)](#)
- [GetTrigger 작업\(Python: get_triggers\)](#)
- [UpdateTrigger 작업\(Python: update_trigger\)](#)
- [StopTrigger 작업\(Python: stop_trigger\)](#)
- [DeleteTrigger 작업\(Python: delete_trigger\)](#)
- [ListTriggers 작업\(Python: list_triggers\)](#)
- [BatchGetTriggers 작업\(Python: batch_get_triggers\)](#)

CreateTrigger 작업(Python: create_trigger)

새로운 트리거를 만듭니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

트리거의 이름입니다.

- WorkflowName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

트리거와 연결된 워크플로의 이름입니다.

- Type – 필수(Required): UTF-8 문자열입니다(유효 값: SCHEDULED | CONDITIONAL | ON_DEMAND | EVENT).

새로운 트리거의 유형입니다.

- Schedule – UTF-8 문자열입니다.

일정을 지정하는 데 사용되는 cron 표현식입니다([작업 및 크롤러의 시간 기반 일정](#) 참조). 예를 들어, 매일 오후 12시 15분(UTC)에 실행하려면 cron(15 12 * * ? *)을 지정합니다.

트리거 유형이 SCHEDULED(예정)되면 필드가 필요합니다.

- Predicate – [조건자](#) 객체입니다.

새로운 트리거가 시작할 시기를 지정하는 조건자입니다.

트리거 유형이 CONDITIONAL이면 이 필드는 필수입니다.

- Actions – 필수(Required): [작업](#) 객체의 배열입니다.

트리거가 발생하면 이 트리거가 시작되는 작업입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

새로운 트리거에 대한 설명.

- StartOnCreation – 부울입니다.

생성 시 SCHEDULED 및 CONDITIONAL 트리거를 시작하려면 true로 설정합니다. ON_DEMAND 트리거에는 True가 지원되지 않습니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이 트리거에서 사용할 태그입니다. 태그를 사용하여 트리거에 대한 액세스를 제한할 수 있습니다. AWS Glue의 태그에 대한 자세한 내용은 개발자 안내서의 [AWS Glue의 AWS 태그](#)를 참조하세요.

- EventBatchingCondition – [EventBatchingCondition](#) 객체입니다.

EventBridge 이벤트 트리거가 실행되기 전에 충족되어야 하는 배치 조건(수신된 이벤트의 지정된 수 또는 배치 기간이 만료됨).

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

트리거의 이름입니다.

Errors

- AlreadyExistsException
- EntityNotFoundException
- InvalidInputException
- IdempotentParameterMismatchException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

StartTrigger 작업(Python: start_trigger)

기존 트리거 시작. 다른 유형의 트리거가 시작하는 방법에 대한 자세한 내용은 [작업 트리거](#)를 참조하십시오.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

시작할 트리거의 이름입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

시작된 트리거의 이름입니다.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `ConcurrentRunsExceededException`

GetTrigger 작업(Python: `get_trigger`)

트리거의 정의를 검색합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

가져올 트리거의 이름입니다.

응답

- `Trigger` – [트리거](#) 객체입니다.

요청한 트리거 정의입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException

GetTrigger 작업(Python: get_triggers)

이 작업과 연결된 모든 트리거를 얻습니다.

요청

- NextToken – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

- DependentJobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

트리거를 가져올 작업 이름입니다. 이 작업을 시작할 수 있는 트리거가 반환되고, 이런 트리거가 없으면 모든 트리거가 반환됩니다.

- MaxResults - 1 이상 200 이하의 숫자(정수)입니다.

응답의 최대 크기입니다.

응답

- Triggers – [트리거](#) 객체의 배열입니다.

지정된 작업에 대한 트리거 목록.

- NextToken – UTF-8 문자열입니다.

모든 요청된 트리거가 반환하지 않은 경우의 지속 토큰입니다.

Errors

- EntityNotFoundException
- InvalidInputException

- `InternalServiceException`
- `OperationTimeoutException`

UpdateTrigger 작업(Python: `update_trigger`)

트리거 정의 업데이트

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트할 트리거의 이름입니다.

- `TriggerUpdate` – 필수(Required): [TriggerUpdate](#) 객체입니다.

트리거를 업데이트하여 얻은 새로운 값입니다.

응답

- `Trigger` – [트리거](#) 객체입니다.

결과 트리거 정의입니다.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

StopTrigger 작업(Python: `stop_trigger`)

지정된 트리거를 중지합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

중지할 트리거의 이름입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

중지된 트리거의 이름입니다.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

DeleteTrigger 작업(Python: `delete_trigger`)

지정된 트리거를 삭제합니다. 트리거를 못 찾으면 어떤 예외도 없습니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 트리거의 이름입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제된 트리거의 이름입니다.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentModificationException`

ListTriggers 작업(Python: `list_triggers`)

AWS 계정의 모든 트리거 리소스 또는 지정된 태그를 가진 리소스의 이름을 검색합니다. 이 작업을 통해 계정에서 사용 가능한 리소스와 그 이름을 확인할 수 있습니다.

이 작업을 수행하면 응답에서 필터로 사용할 수 있는 선택 사항인 `Tags` 필드가 검색되기 때문에 태그가 지정된 리소스를 하나의 그룹으로 검색할 수 있습니다. 태그 필터링을 사용하기로 선택하면 태그가 포함된 리소스만 검색됩니다.

요청

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적인 요청이라면 지속적인 토큰입니다.

- `DependentJobName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

트리거를 가져올 작업 이름 이 작업을 시작할 수 있는 트리거가 반환됩니다. 그런 트리거가 없으면 모든 트리거가 반환됩니다.

- `MaxResults` - 1 이상 200 이하의 숫자(정수)입니다.

반환할 목록의 최대 크기.

- `Tags` – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이렇게 태그가 지정된 리소스만 반환하도록 지정합니다.

응답

- `TriggerNames` – UTF-8 문자열의 배열입니다.

계정의 모든 트리거 또는 지정된 태그를 가진 트리거의 이름입니다.

- `NextToken` – UTF-8 문자열입니다.

반환된 목록이 사용가능한 마지막 지표를 포함하지 경우의 연속 토큰입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

BatchGetTriggers 작업(Python: `batch_get_triggers`)

주어진 트리거 이름 목록에 대한 리소스 메타데이터 목록을 반환합니다. `ListTriggers` 작업을 호출한 후에는 권한이 부여된 데이터에 액세스하기 위해 이 작업을 호출할 수 있습니다. 이 작업은 태그를 사용하는 권한 조건을 포함해 모든 IAM 권한을 지원합니다.

요청

- `TriggerNames` – 필수(Required): UTF-8 문자열의 배열입니다.

트리거 이름(`ListTriggers` 작업에서 반환된 이름일 수 있음)의 목록입니다.

응답

- `Triggers` – [트리거](#) 객체의 배열입니다.

트리거 정의 목록입니다.

- `TriggersNotFound` – UTF-8 문자열의 배열입니다.

찾을 수 없는 트리거의 이름 목록입니다.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

대화형 세션 API

대화형 세션 API는 대화형 세션을 사용하여 AWS Glue 데이터 통합을 위한 ETL (추출, 변환, 로드) 스크립트를 빌드하고 테스트하는 것과 관련된 AWS Glue API를 설명합니다.

데이터 타입

- [세션 구조](#)
- [SessionCommand 구조](#)
- [명령문 구조](#)
- [StatementOutput 구조체](#)
- [StatementOutputData 구조](#)
- [ConnectionsList 구조](#)

세션 구조

원격 Spark 런타임 환경이 실행되는 기간입니다.

필드

- `Id` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

세션의 ID입니다.

- `CreatedOn` – 타임스탬프입니다.

세션이 생성된 시간 및 날짜입니다.

- `Status` – UTF-8 문자열입니다(유효 값: PROVISIONING | READY | FAILED | TIMEOUT | STOPPING | STOPPED).

세션 상태입니다.

- ErrorMessage – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

세션 중 표시되는 오류 메시지입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

세션에 대한 설명입니다.

- Role – [Custom string pattern #26](#)과 일치하는 UTF-8 문자열입니다(20~2,048바이트).

세션과 연결된 IAM 역할의 이름 또는 Amazon 리소스 이름(ARN)입니다.

- Command – [SessionCommand](#) 객체입니다.

명령 개체 SessionCommand. 참조.

- DefaultArguments – 75개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

각 값은 [URI address multi-line string pattern](#)와 일치하는 UTF-8 문자열(4,096바이트 이하)입니다.

키-값 페어의 맵 배열입니다. 최대 75페어입니다.

- Connections – [ConnectionsList](#) 객체입니다.

세션에 사용되는 연결 수입니다.

- Progress - 숫자(double)입니다.

세션의 코드 실행 진행률입니다.

- MaxCapacity - 숫자(double)입니다.

작업 실행 시 할당할 수 있는 AWS Glue 데이터 처리 장치 (DPU) 수 DPU는 4 vCPU의 컴퓨팅 용량과 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다.

- SecurityConfiguration – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

세션과 함께 사용할 SecurityConfiguration 구조의 이름.

- GlueVersion – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue 버전에 따라 지원되는 아파치 스파크와 Python 버전이 결정됩니다. AWS Glue 는 2.0보다 GlueVersion 커야 합니다.

- `DataAccessId` – UTF-8 문자열입니다(1~36바이트 이하)

세션의 데이터 액세스 ID입니다.

- `PartitionId` – UTF-8 문자열입니다(1~36바이트 이하)

세션의 파티션 ID입니다.

- `NumberOfWorkers` - 숫자(정수)입니다.

세션에 사용할 정의된 `WorkerType`의 작업자 수입니다.

- `WorkerType` – UTF-8 문자열입니다(유효한 값: `Standard=""` | `G.1X=""` | `G.2X=""` | `G.025X=""` | `G.4X=""` | `G.8X=""` | `Z.2X=""`).

세션이 실행될 때 할당되는 미리 정의된 작업자 유형입니다. Spark 세션에 대해 `G.1X`, `G.2X`, `G.4X` 또는 `G.8X`의 값을 허용합니다. Ray 세션에 대해 `Z.2X`의 값을 허용합니다.

- `CompletedOn` – 타임스탬프입니다.

이 세션이 완료된 날짜 및 시간입니다.

- `ExecutionTime` - 숫자(double)입니다.

세션이 실행된 총 시간입니다.

- `DPUSeconds` - 숫자(double)입니다.

세션에서 소비한 DPU (공식: `ExecutionTime * MaxCapacity`)

- `IdleTimeout` - 숫자(정수)입니다.

세션 시간이 초과되기까지 유휴 상태의 시간(분)입니다.

- `ProfileName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

세션과 관련된 AWS Glue 사용 프로필의 이름.

SessionCommand 구조

작업을 실행하는 `SessionCommand`입니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
의 이름을 지정합니다 SessionCommand. 'glueetl' 또는 'gluestreaming'일 수 있습니다.
- PythonVersion – [Custom string pattern #21](#)과(와) 일치하는 UTF-8 문자열입니다.
Python 버전을 지정합니다. Python의 버전으로 Spark 유형의 작업에 대해 지원되는 버전을 확인할 수 있습니다.

명령문 구조

세션에서 발생할 특정 작업에 대한 명령문 또는 요청입니다.

필드

- Id - 숫자(정수)입니다.
문의 ID입니다.
- Code – UTF-8 문자열입니다.
문의 실행 코드입니다.
- State – UTF-8 문자열입니다(유효 값: WAITING | RUNNING | AVAILABLE | CANCELLING | CANCELLED | ERROR).
요청이 실행되는 동안의 상태입니다.
- Output – [StatementOutput](#) 객체입니다.
JSON의 출력입니다.
- Progress - 숫자(double)입니다.
코드 실행 진행률입니다.
- StartedOn - 숫자(정수)입니다.
작업 정의가 시작된 Unix 시간 및 날짜입니다.
- CompletedOn - 숫자(정수)입니다.
작업 정의가 완료된 Unix 시간 및 날짜입니다.

StatementOutput 구조체

JSON 형식의 코드 실행 출력입니다.

필드

- Data – [StatementOutputData](#) 객체입니다.
코드 실행 출력입니다.
- ExecutionCount - 숫자(정수)입니다.
출력의 실행 수입니다.
- Status – UTF-8 문자열입니다(유효 값: WAITING | RUNNING | AVAILABLE | CANCELLING | CANCELLED | ERROR).
코드 실행 출력의 상태입니다.
- ErrorName – UTF-8 문자열입니다.
출력의 오류 이름입니다.
- ErrorValue – UTF-8 문자열입니다.
출력의 오류 값입니다.
- Traceback – UTF-8 문자열의 배열입니다.
출력의 트레이스백입니다.

StatementOutputData 구조

JSON 형식의 코드 실행 출력입니다.

필드

- TextPlain – UTF-8 문자열입니다.
텍스트 형식의 코드 실행 출력입니다.

ConnectionsList 구조

작업이 사용한 연결을 지정합니다.

필드

- **Connections** – UTF-8 문자열의 배열입니다.
작업이 사용한 연결 목록입니다.

운영

- [CreateSession](#) 액션 (Python: `create_session`)
- [StopSession](#) 액션 (파이썬: `스탑_세션`)
- [DeleteSession](#) 액션 (Python: `삭제_세션`)
- [GetSession](#) 액션 (파이썬: `get_session`)
- [ListSessions](#) 액션 (Python: `리스트_세션`)
- [RunStatement](#) 액션 (Python: `run_statement`)
- [CancelStatement](#) 액션 (Python: `취소_명령문`)
- [GetStatement](#) 액션 (파이썬: `get_statement`)
- [ListStatements](#) 액션 (Python: `리스트_스테이트먼트`)

CreateSession 액션 (Python: `create_session`)

새 세션을 생성합니다.

요청

새 세션 생성을 요청합니다.

- **Id** – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
세션 요청의 ID입니다.
- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.
세션에 대한 설명입니다.
- **Role** – 필수: [Custom string pattern #26](#)과 일치하는 UTF-8 문자열입니다(20~2,048바이트).
IAM 역할 ARN
- **Command** – 필수: [SessionCommand](#) 객체입니다.

작업을 실행하는 SessionCommand입니다.

- Timeout – 1 이상의 숫자(정수)입니다.

세션 시간이 초과되기 전 경과되는 시간(분)입니다. Spark ETL 작업의 기본값은 이 작업 유형의 최대 세션 수명 시간인 48시간(2,880분)입니다. 다른 작업 유형에 대해서는 설명서를 참조하세요.

- IdleTimeout – 1 이상의 숫자(정수)입니다.

세션 시간이 초과되기까지 유휴 상태의 시간(초)입니다. Spark ETL 작업의 기본값은 시간 제한 값입니다. 다른 작업 유형에 대해서는 설명서를 참조하세요.

- DefaultArguments – 75개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

각 값은 [URI address multi-line string pattern](#)와 일치하는 UTF-8 문자열(4,096바이트 이하)입니다.

키-값 페어의 맵 배열입니다. 최대 75페어입니다.

- Connections – [ConnectionsList](#) 객체입니다.

세션에 사용할 연결 수입니다.

- MaxCapacity - 숫자(double)입니다.

작업 실행 시 할당할 수 있는 AWS Glue 데이터 처리 장치 (DPU) 수입니다. DPU는 4 vCPU의 컴퓨팅 용량과 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다.

- NumberOfWorkers - 숫자(정수)입니다.

세션에 사용할 정의된 WorkerType의 작업자 수입니다.

- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

작업이 실행될 때 할당되는 미리 정의된 작업자 유형입니다. Spark 작업에 대해 G.1X, G.2X, G.4X 또는 G.8X의 값을 허용합니다. Ray 노트북에 대해 Z.2X 값을 허용합니다.

- G.1X 작업자 유형에서 각 작업자는 84GB 디스크(약 34GB의 여유 공간)에서 1개의 DPU(vCPU 4개, 메모리 16GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수 있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.
- G.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 77GB의 여유 공간)에서 2개의 DPU(vCPU 8개, 메모리 32GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 대부분의 작업을 실행할 수

있는 확장 가능하고 비용 효율적인 방법을 제공하기 위해 데이터 변환, 조인, 쿼리와 같은 워크로드에서 이 작업자 유형을 사용하는 것이 좋습니다.

- G.4X 작업자 유형에서 각 작업자는 256GB 디스크(약 235GB의 여유 공간)에서 4개의 DPU(vCPU 16개, 메모리 64GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 미국 동부 (오하이오), 미국 동부 (버지니아 북부), 미국 서부 (오레곤), 아시아 태평양 (싱가포르), 아시아 태평양 (시드니), 아시아 태평양 (도쿄), 캐나다 (중부), 유럽 (프랑크푸르트), 유럽 (아일랜드) 및 유럽 (스톡홀름) 의 AWS Glue 버전 3.0 이상 Spark ETL 작업에만 사용할 수 있습니다. AWS
- G.8X 작업자 유형에서 각 작업자는 512GB 디스크(약 487GB의 여유 공간)에서 8개의 DPU(vCPU 32개, 메모리 128GB)에 매핑되고, 작업자당 1개의 실행기를 제공합니다. 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에서 이 작업자 유형을 사용하는 것이 좋습니다. 이 작업자 유형은 작업자 유형에 지원되는 AWS 지역과 동일한 지역에서 AWS Glue 버전 3.0 이상의 Spark ETL G.4X 작업에만 사용할 수 있습니다.
- Z.2X 작업자 유형에서 각 작업자는 128GB 디스크(약 120GB의 여유 공간)에서 2개의 M-DPU(vCPU 8개, 메모리 64GB)에 매핑되고, Autoscaler에 따라 최대 8개의 Ray 작업자를 제공합니다.
- SecurityConfiguration – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

세션과 함께 사용할 SecurityConfiguration 구조의 이름

- GlueVersion – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue 버전에 따라 지원되는 아파치 스파크와 Python 버전이 결정됩니다. AWS Glue 는 2.0보다 GlueVersion 커야 합니다.

- DataAccessId – UTF-8 문자열입니다(1~36바이트 이하)

세션의 데이터 액세스 ID입니다.

- PartitionId – UTF-8 문자열입니다(1~36바이트 이하)

세션의 파티션 ID입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

세션에 속하는 키 값 페어(태그)의 맵입니다.

- RequestOrigin – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

요청의 오리진입니다.

- ProfileName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

세션과 관련된 AWS Glue 사용자 프로필의 이름.

응답

- Session – [세션](#) 객체입니다.

응답으로 세션 객체를 반환합니다.

Errors

- AccessDeniedException
- IdempotentParameterMismatchException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException
- AlreadyExistsException
- ResourceNumberLimitExceededException

StopSession 액션 (파이썬: 스탱_세션)

세션을 중지합니다.

요청

- Id – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

중지할 세션의 ID입니다.

- RequestOrigin – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

요청의 오리진입니다.

응답

- Id – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
중지된 세션의 ID를 반환합니다.

Errors

- AccessDeniedException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException
- ConcurrentModificationException

DeleteSession 액션 (Python: 삭제_세션)

세션을 삭제합니다.

요청

- Id – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
삭제되는 세션의 ID입니다.
- RequestOrigin – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.
세션 삭제 요청의 오리진 이름입니다.

응답

- Id – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제된 세션의 ID를 반환합니다.

Errors

- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`
- `ConcurrentModificationException`

GetSession 액션 (파이썬: `get_session`)

세션을 검색합니다.

요청

- `Id` – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

세션의 ID입니다.

- `RequestOrigin` – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

요청의 오리진입니다.

응답

- `Session` – [세션](#) 객체입니다.

세션 객체가 응답으로 반환됩니다.

Errors

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

- `OperationTimeoutException`
- `InvalidInputException`

ListSessions 액션 (Python: 리스트_세션)

세션 목록을 검색합니다.

요청

- `NextToken` – 400,000바이트 이하 길이의 UTF-8 문자열입니다.

다음 결과 세트를 가져오기 위한 토큰이지만 결과가 더 없는 경우에는 null 값을 갖습니다.

- `MaxResults` – 1~1,000의 숫자(정수)입니다.

최대 결과 수입니다.

- `Tags` – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

세션에 속한 태그입니다.

- `RequestOrigin` – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

요청의 오리진입니다.

응답

- `Ids` – UTF-8 문자열의 배열입니다.

세션의 ID를 반환합니다.

- `Sessions` – [세션](#) 객체의 배열입니다.

세션 객체를 반환합니다.

- `NextToken` – 400,000바이트 이하 길이의 UTF-8 문자열입니다.

다음 결과 세트를 가져오기 위한 토큰이지만 결과가 더 없는 경우에는 null 값을 갖습니다.

Errors

- `AccessDeniedException`
- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`

RunStatement 액션 (Python: `run_statement`)

문을 실행합니다.

요청

- `SessionId` – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행할 문의 세션 ID입니다.

- `Code` – 필수: UTF-8 문자열입니다(68,000바이트 이하).

실행할 문 코드입니다.

- `RequestOrigin` – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

요청의 오리진입니다.

응답

- `Id` - 숫자(정수)입니다.

실행된 문의 ID를 반환합니다.

Errors

- `EntityNotFoundException`
- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`

- `InvalidInputException`
- `ValidationException`
- `ResourceNumberLimitExceededException`
- `IllegalSessionStateException`

CancelStatement 액션 (Python: 취소_명령문)

문을 취소합니다.

요청

- `SessionId` – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

취소할 문의 세션 ID입니다.

- `Id` – 필수(Required): 숫자(정수)입니다.

취소할 문의 ID입니다.

- `RequestOrigin` – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

문 취소 요청의 오리진입니다.

응답

- 무응답 파라미터.

Errors

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`

GetStatement 액션 (파이썬: get_statement)

문을 검색합니다.

요청

- SessionId – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

문의 세션 ID입니다.

- Id – 필수(Required): 숫자(정수)입니다.

문의 ID입니다.

- RequestOrigin – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

요청의 오리지인입니다.

응답

- Statement – [문](#) 객체입니다.

문을 반환합니다.

Errors

- AccessDeniedException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- IllegalSessionStateException

ListStatements 액션 (Python: 리스트_스테이트먼트)

세션에 대한 문을 나열합니다.

요청

- `SessionId` – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

문의 세션 ID입니다.

- `RequestOrigin` – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

문 나열 요청의 오리진입니다.

- `NextToken` – 400,000바이트 이하 길이의 UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- `Statements` – [문](#) 객체의 배열입니다.

문 목록을 반환합니다.

- `NextToken` – 400,000바이트 이하 길이의 UTF-8 문자열입니다.

모든 문이 아직 반환되지 않은 경우의 지속 토큰입니다.

Errors

- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`
- `IllegalSessionStateException`

개발 엔드포인트 API

개발 엔드포인트 API는 사용자 지정 `DevEndpoint`를 사용한 테스트와 관련된 AWS Glue API에 대해 설명합니다.

데이터 타입

- [DevEndpoint 구조](#)
- [DevEndpointCustomLibraries 구조](#)

DevEndpoint 구조

개발자가 원격으로 ETL(추출, 변환 및 로드) 스크립트를 디버그할 수 있는 개발 엔드포인트입니다.

필드

- `EndpointName` – UTF-8 문자열입니다.

`DevEndpoint`의 이름입니다.

- `RoleArn` – [AWS IAM ARN string pattern](#)과(와) 일치하는 UTF-8 문자열입니다.

이 `DevEndpoint`에 사용되는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

- `SecurityGroupIds` – UTF-8 문자열의 배열입니다.

이 `DevEndpoint`에 사용된 보안 그룹 식별자 목록입니다.

- `SubnetId` – UTF-8 문자열입니다.

이 `DevEndpoint`에 대한 서브넷 ID입니다.

- `YarnEndpointAddress` – UTF-8 문자열입니다.

이 `DevEndpoint`가 사용하는 YARN 엔드포인트 주소입니다.

- `PrivateAddress` – UTF-8 문자열입니다.

`DevEndpoint`가 하나의 VPC에 생성되었다면 VPC 내 `DevEndpoint`에 액세스할 수 있는 프라이빗 IP 주소입니다. `PrivateAddress` 필드는 VPC 내에 `DevEndpoint`를 생성할 때만 표시됩니다.

- `ZeppelinRemoteSparkInterpreterPort` - 숫자(정수)입니다.

원격 Apache Spark 인터프리터용 Apache Zeppelin 포트입니다.

- `PublicAddress` – UTF-8 문자열입니다.

이 `DevEndpoint`가 사용하는 퍼블릭 IP 주소입니다. `PublicAddress` 필드는 비Virtual Private Cloud(VPC) `DevEndpoint`를 생성할 때만 표시됩니다.

- `Status` – UTF-8 문자열입니다.

이 DevEndpoint의 현재 상태입니다.

- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

개발 엔드포인트로 할당되는 미리 정의된 작업자 유형입니다. Standard, G.1X 또는 G.2X 값을 허용합니다.

- Standard 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 50GB 디스크와, 작업자당 실행기 2개를 제공합니다.
- G.1X 작업자 유형의 경우, 각 작업자가 1DPU(4vCPU, 16GB 메모리, 64GB 디스크)에 매핑되고, 작업자당 실행기 1개를 제공합니다. 메모리 집약적인 작업의 경우 이 작업자 유형을 사용하는 것이 좋습니다.
- G.2X 작업자 유형의 경우, 각 작업자가 2DPU(8vCPU, 32GB 메모리, 128GB 디스크)에 매핑되고, 작업자당 실행기 1개를 제공합니다. 메모리 집약적인 작업의 경우 이 작업자 유형을 사용하는 것이 좋습니다.

알려진 문제: 개발 엔드포인트가 G.2X WorkerType 구성으로 만들어질 경우 개발 엔드포인트의 Spark 드라이버가 4 vCPU, 16GB 메모리 및 64GB 디스크에서 실행됩니다.

- GlueVersion – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Glue 버전에 따라AWS Glue에서 지원하는 Apache Spark와 Python의 버전이 정해집니다. Python 버전은 개발 엔드포인트에서 ETL 스크립트를 실행하기 위해 지원되는 버전을 나타냅니다.

이용 가능한 AWS Glue 버전과 그에 상응하는 Spark 및 Python 버전에 대한 자세한 내용은 개발자 안내서의 [Glue 버전](#)을 참조하세요.

Glue 버전 지정 없이 생성된 개발 엔드포인트는 Glue 0.9로 기본 지정됩니다.

CreateDevEndpoint 또는 UpdateDevEndpoint API의 Arguments 파라미터를 사용하여 개발 엔드포인트에 지원되는 Python 버전을 지정할 수 있습니다. 인수가 제공되지 않으면 버전은 기본적으로 Python 2입니다.

- NumberOfWorkers - 숫자(정수)입니다.

개발 엔드포인트로 할당되는 정의된 workerType의 작업자 수입니다.

정의할 수 있는 최대 작업자 수는 G.1X의 경우 299개, G.2X의 경우 149개입니다.

- NumberOfNodes - 숫자(정수)입니다.

이 DevEndpoint에 할당된 AWS Glue Glue 데이터 처리 장치(DPU) 수입입니다.

- AvailabilityZone – UTF-8 문자열입니다.

이 DevEndpoint가 위치한 AWS 가용 영역입니다.

- VpcId – UTF-8 문자열입니다.

이 DevEndpoint가 사용한 Virtual Private Cloud(VPC)의 ID입니다.

- ExtraPythonLibsS3Path – UTF-8 문자열입니다.

DevEndpoint에서 로드되어야 할 Amazon S3 버킷에 있는 하나 이상의 Python 라이브러리에 대한 경로입니다. 여러 값은 쉼표(,)로 구분된 완전한 경로여야 합니다.

Note

DevEndpoint에서는 순수 Python 라이브러리만 사용할 수 있습니다. [pandas](#) Python 데이터 분석 라이브러리 등 C 확장을 활용하는 라이브러리는 현재 지원되지 않습니다.

- ExtraJarsS3Path – UTF-8 문자열입니다.

DevEndpoint에서 로드되어야 할 S3 버킷에 있는 하나 이상의 Java .jar 파일에 대한 경로입니다.

Note

DevEndpoint에서는 순수 Java/Scala 라이브러리만 사용할 수 있습니다.

- FailureReason – UTF-8 문자열입니다.

이 DevEndpoint의 현재 실패 이유입니다.

- LastUpdateStatus – UTF-8 문자열입니다.

마지막 업데이트의 상태입니다.

- CreatedTimestamp – 타임스탬프입니다.

이 DevEndpoint가 생성된 시점.

- LastModifiedTimestamp – 타임스탬프입니다.

이 DevEndpoint가 마지막으로 수정된 시점.

- PublicKey – UTF-8 문자열입니다.

인증용으로 이 DevEndpoint에서 사용될 퍼블릭 키입니다. 사용할 권장 속성이 퍼블릭 키이므로 이전 버전과의 호환성을 위해 이 속성이 제공됩니다.

- `PublicKeys` – 5개 이하의 문자열로 구성된 UTF-8 문자열의 배열입니다.

인증용으로 DevEndpoints에서 사용될 퍼블릭 키 목록입니다. 퍼블릭 키를 사용하면 클라이언트마다 다른 프라이빗 키를 지정할 수 있으므로 이 속성의 사용이 단일 퍼블릭 키보다 우선됩니다.

Note

이전에 퍼블릭 키로 엔드포인트를 생성한 경우, 퍼블릭 키 목록을 설정할 수 있으려면 해당 키를 제거해야 합니다. `deletePublicKeys` 속성의 퍼블릭 키 콘텐츠와 `addPublicKeys` 속성의 새 키 목록을 사용하여 `UpdateDevEndpoint` API를 호출하십시오.

- `SecurityConfiguration` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 DevEndpoint에 사용할 `SecurityConfiguration` 구조의 이름입니다.

- `Arguments` – 100개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

DevEndpoint 구성에 사용되는 인수의 맵입니다.

유효한 인수는 다음과 같습니다.

- `"--enable-glue-datacatalog": ""`

`CreateDevEndpoint` 또는 `UpdateDevEndpoint` API의 `Arguments` 파라미터를 사용하여 개발 엔드포인트에 지원되는 Python 버전을 지정할 수 있습니다. 인수가 제공되지 않으면 버전은 기본적으로 Python 2입니다.

DevEndpointCustomLibraries 구조

개발 엔드포인트에 로드될 사용자 지정 라이브러리입니다.

필드

- `ExtraPythonLibsS3Path` – UTF-8 문자열입니다.

DevEndpoint에서 로드되어야 할 Amazon Simple Storage Service(Amazon S3) 버킷에 있는 하나 이상의 Python 라이브러리에 대한 경로입니다. 여러 값은 쉼표(,)로 구분된 완전한 경로여야 합니다.

Note

DevEndpoint에서는 순수 Python 라이브러리만 사용할 수 있습니다. [pandas](#) Python 데이터 분석 라이브러리 등 C 확장을 활용하는 라이브러리는 현재 지원되지 않습니다.

- ExtraJarsS3Path – UTF-8 문자열입니다.

DevEndpoint에서 로드되어야 할 S3 버킷에 있는 하나 이상의 Java .jar 파일에 대한 경로입니다.

Note

DevEndpoint에서는 순수 Java/Scala 라이브러리만 사용할 수 있습니다.

운영

- [CreateDevEndpoint 작업\(Python: create_dev_endpoint\)](#)
- [UpdateDevEndpoint 작업\(Python: update_dev_endpoint\)](#)
- [DeleteDevEndpoint 작업\(Python: delete_dev_endpoint\)](#)
- [GetDevEndpoint 작업\(Python: get_dev_endpoint\)](#)
- [GetDevEndpoints 작업\(Python: get_dev_endpoints\)](#)
- [BatchGetDevEndpoints 작업\(Python: batch_get_dev_endpoints\)](#)
- [ListDevEndpoints 작업\(Python: list_dev_endpoints\)](#)

CreateDevEndpoint 작업(Python: create_dev_endpoint)

새 개발 엔드포인트를 생성합니다.

요청

- EndpointName – 필수(Required): UTF-8 문자열입니다.

새 DevEndpoint에 지정된 이름입니다.

- RoleArn – 필수(Required): [AWS IAM ARN string pattern](#)과(와) 일치하는 UTF-8 문자열입니다.

DevEndpoint의 IAM 역할입니다.

- SecurityGroupIds – UTF-8 문자열의 배열입니다.

새로운 DevEndpoint가 사용할 보안 그룹의 보안 그룹 ID입니다.

- SubnetId – UTF-8 문자열입니다.

사용할 새 DevEndpoint의 서브넷 ID입니다.

- PublicKey – UTF-8 문자열입니다.

인증용으로 이 DevEndpoint에서 사용될 퍼블릭 키입니다. 사용할 권장 속성이 퍼블릭 키이므로 이전 버전과의 호환성을 위해 이 속성이 제공됩니다.

- PublicKeys – 5개 이하의 문자열로 구성된 UTF-8 문자열의 배열입니다.

인증용으로 개발 엔드포인트에서 사용될 퍼블릭 키 목록입니다. 퍼블릭 키를 사용하면 클라이언트마다 다른 프라이빗 키를 지정할 수 있으므로 이 속성의 사용이 단일 퍼블릭 키보다 우선됩니다.

Note

이전에 퍼블릭 키로 엔드포인트를 생성한 경우, 퍼블릭 키 목록을 설정할 수 있으려면 해당 키를 제거해야 합니다. deletePublicKeys 속성의 퍼블릭 키 콘텐츠와 addPublicKeys 속성의 새 키 목록을 사용하여 UpdateDevEndpoint API를 호출하십시오.

- NumberOfNodes - 숫자(정수)입니다.

이 DevEndpoint에 할당할 AWS Glue 데이터 처리 장치(DPU) 수입니다.

- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

개발 엔드포인트로 할당되는 미리 정의된 작업자 유형입니다. Standard, G.1X 또는 G.2X 값을 허용합니다.

- Standard 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 50GB 디스크와, 작업자당 실행기 2개를 제공합니다.
- G.1X 작업자 유형의 경우, 각 작업자가 1DPU(4vCPU, 16GB 메모리, 64GB 디스크)에 매핑되고, 작업자당 실행기 1개를 제공합니다. 메모리 집약적인 작업의 경우 이 작업자 유형을 사용하는 것이 좋습니다.

- G.2X 작업자 유형의 경우, 각 작업자가 2DPU(8vCPU, 32GB 메모리, 128GB 디스크)에 매핑되고, 작업자당 실행기 1개를 제공합니다. 메모리 집약적인 작업의 경우 이 작업자 유형을 사용하는 것이 좋습니다.

알려진 문제: 개발 엔드포인트가 G.2X WorkerType 구성으로 만들어질 경우 개발 엔드포인트의 Spark 드라이버가 4 vCPU, 16GB 메모리 및 64GB 디스크에서 실행됩니다.

- GlueVersion – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Glue 버전에 따라 AWS Glue에서 지원하는 Apache Spark와 Python의 버전이 정해집니다. Python 버전은 개발 엔드포인트에서 ETL 스크립트를 실행하기 위해 지원되는 버전을 나타냅니다.

이용 가능한 AWS Glue 버전과 그에 상응하는 Spark 및 Python 버전에 대한 자세한 내용은 개발자 안내서의 [Glue 버전](#)을 참조하세요.

Glue 버전 지정 없이 생성된 개발 엔드포인트는 Glue 0.9로 기본 지정됩니다.

CreateDevEndpoint 또는 UpdateDevEndpoint API의 Arguments 파라미터를 사용하여 개발 엔드포인트에 지원되는 Python 버전을 지정할 수 있습니다. 인수가 제공되지 않으면 버전은 기본적으로 Python 2입니다.

- NumberOfWorkers - 숫자(정수)입니다.

개발 엔드포인트로 할당되는 정의된 workerType의 작업자 수입니다.

정의할 수 있는 최대 작업자 수는 G.1X의 경우 299개, G.2X의 경우 149개입니다.

- ExtraPythonLibsS3Path – UTF-8 문자열입니다.

DevEndpoint에서 로드되어야 할 Amazon S3 버킷에 있는 하나 이상의 Python 라이브러리에 대한 경로입니다. 여러 값은 쉼표(,)로 구분된 완전한 경로여야 합니다.

Note

DevEndpoint에서는 순수 Python 라이브러리만 사용할 수 있습니다. [pandas](#) Python 데이터 분석 라이브러리 등 C 확장을 활용하는 라이브러리는 아직 지원되지 않습니다.

- ExtraJarsS3Path – UTF-8 문자열입니다.

DevEndpoint에서 로드되어야 할 S3 버킷에 있는 하나 이상의 Java .jar 파일에 대한 경로입니다.

- **SecurityConfiguration** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 DevEndpoint에 사용할 SecurityConfiguration 구조의 이름입니다.

- **Tags** – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이 DevEndpoint에서 사용할 태그입니다. 태그를 사용하여 DevEndpoint에 대한 액세스를 제한할 수 있습니다. AWS Glue의 태그에 대한 자세한 내용은 개발자 안내서의 [AWS Glue의 AWS 태그](#)를 참조하세요.

- **Arguments** – 100개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

DevEndpoint 구성에 사용되는 인수의 맵입니다.

응답

- **EndpointName** – UTF-8 문자열입니다.

새로운 DevEndpoint에 할당된 이름입니다.

- **Status** – UTF-8 문자열입니다.

새 DevEndpoint의 현재 상태입니다.

- **SecurityGroupIds** – UTF-8 문자열의 배열입니다.

새로운 DevEndpoint에 지정된 보안 그룹입니다.

- **SubnetId** – UTF-8 문자열입니다.

새로운 DevEndpoint에 지정된 서브넷 ID입니다.

- **RoleArn** – [AWS IAM ARN string pattern](#)과(와) 일치하는 UTF-8 문자열입니다.

새로운 DevEndpoint에 할당된 역할의 Amazon 리소스 이름(ARN)입니다.

- **YarnEndpointAddress** – UTF-8 문자열입니다.

이 DevEndpoint가 사용하는 YARN 엔드포인트 주소입니다.

- ZeppelinRemoteSparkInterpreterPort - 숫자(정수)입니다.

원격 Apache Spark 인터프리터용 Apache Zeppelin 포트입니다.

- NumberOfNodes - 숫자(정수)입니다.

이 DevEndpoint에 할당된 AWS Glue 데이터 처리 장치(DPU) 수입니다.

- WorkerType - UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

개발 엔드포인트로 할당되는 미리 정의된 작업자 유형입니다. Standard, G.1X 또는 G.2X 값일 수 있습니다.

- GlueVersion - [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

Glue 버전에 따라 AWS Glue에서 지원하는 Apache Spark와 Python의 버전이 정해집니다. Python 버전은 개발 엔드포인트에서 ETL 스크립트를 실행하기 위해 지원되는 버전을 나타냅니다.

이용 가능한 AWS Glue 버전과 그에 상응하는 Spark 및 Python 버전에 대한 자세한 내용은 개발자 안내서의 [Glue 버전](#)을 참조하세요.

- NumberOfWorkers - 숫자(정수)입니다.

개발 엔드포인트로 할당되는 정의된 workerType의 작업자 수입니다.

- AvailabilityZone - UTF-8 문자열입니다.

이 DevEndpoint가 위치한 AWS 가용 영역입니다.

- VpcId - UTF-8 문자열입니다.

이 DevEndpoint가 사용한 Virtual Private Cloud(VPC)의 ID입니다.

- ExtraPythonLibsS3Path - UTF-8 문자열입니다.

DevEndpoint에서 로드되는 S3 버킷에 있는 하나 이상의 Python 라이브러리에 대한 경로입니다.

- ExtraJarsS3Path - UTF-8 문자열입니다.

DevEndpoint에서 로드되는 S3 버킷에 있는 하나 이상의 Java .jar에 대한 경로입니다.

- FailureReason - UTF-8 문자열입니다.

이 DevEndpoint의 현재 실패 이유입니다.

- SecurityConfiguration – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 DevEndpoint에 사용 중인 SecurityConfiguration 구조의 이름입니다.

- CreatedTimestamp – 타임스탬프입니다.

이 DevEndpoint가 생성된 시점입니다.

- Arguments – 100개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

이 DevEndpoint를 구성하는 데 사용되는 인수의 맵입니다.

유효한 인수는 다음과 같습니다.

- "--enable-glue-datacatalog": ""

CreateDevEndpoint 또는 UpdateDevEndpoint API의 Arguments 파라미터를 사용하여 개발 엔드포인트에 지원되는 Python 버전을 지정할 수 있습니다. 인수가 제공되지 않으면 버전은 기본적으로 Python 2입니다.

Errors

- AccessDeniedException
- AlreadyExistsException
- IdempotentParameterMismatchException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException
- ResourceNumberLimitExceededException

UpdateDevEndpoint 작업(Python: update_dev_endpoint)

지정한 개발 엔드포인트를 업데이트합니다.

요청

- `EndpointName` – 필수(Required): UTF-8 문자열입니다.

업데이트할 `DevEndpoint`의 이름입니다.

- `PublicKey` – UTF-8 문자열입니다.

사용할 `DevEndpoint`의 퍼블릭 키입니다.

- `AddPublicKeys` – 5개 이하의 문자열로 구성된 UTF-8 문자열의 배열입니다.

사용할 `DevEndpoint`의 퍼블릭 키 목록입니다.

- `DeletePublicKeys` – 5개 이하의 문자열로 구성된 UTF-8 문자열의 배열입니다.

`DevEndpoint`에서 삭제할 퍼블릭 키 목록입니다.

- `CustomLibraries` – [DevEndpointCustomLibraries](#) 객체입니다.

이 `DevEndpoint`에 로드될 사용자 지정 Python 또는 Java 라이브러리입니다.

- `UpdateEtlLibraries` – 부울입니다.

개발 엔드포인트에 로드될 사용자 지정 라이브러리 목록을 업데이트해야 하면 `True`이고 그렇지 않으면 `False`입니다.

- `DeleteArguments` – UTF-8 문자열의 배열입니다.

`DevEndpoint` 구성에 사용되는 인수의 맵에서 삭제할 인수 키의 목록입니다.

- `AddArguments` – 100개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

`DevEndpoint` 구성에 사용되는 인수의 맵을 추가하기 위한 인수의 맵입니다.

유효한 인수는 다음과 같습니다.

- `"--enable-glue-datacatalog": ""`

CreateDevEndpoint 또는 UpdateDevEndpoint API의 Arguments 파라미터를 사용하여 개발 엔드포인트에 지원되는 Python 버전을 지정할 수 있습니다. 인수가 제공되지 않으면 버전은 기본적으로 Python 2입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- ValidationException

DeleteDevEndpoint 작업(Python: delete_dev_endpoint)

지정한 개발 엔드포인트를 삭제합니다.

요청

- EndpointName – 필수(Required): UTF-8 문자열입니다.

DevEndpoint의 이름입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

- `InvalidInputException`

GetDevEndpoint 작업(Python: `get_dev_endpoint`)

지정된 개발 엔드포인트에 대한 정보를 가져옵니다.

Note

Virtual Private Cloud(VPC)에서 개발 엔드포인트를 생성하면, AWS Glue은(는) 프라이빗 IP 주소만 반환하며 퍼블릭 IP 주소 필드는 입력되지 않습니다. 비 VPC 개발 엔드포인트를 생성할 때 AWS Glue은(는) 퍼블릭 IP 주소만 반환합니다.

요청

- `EndpointName` – 필수(Required): UTF-8 문자열입니다.

정보를 검색할 `DevEndpoint`의 이름입니다.

응답

- `DevEndpoint` – [DevEndpoint](#) 객체입니다.

`DevEndpoint` 정의입니다.

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetDevEndpoints 작업(Python: `get_dev_endpoints`)

이 AWS 계정에서 모든 개발 엔드포인트를 검색합니다.

Note

Virtual Private Cloud(VPC)에서 개발 엔드포인트를 생성하면, AWS Glue는 프라이빗 IP 주소만 반환하며 퍼블릭 IP 주소 필드는 입력되지 않습니다. 비 VPC 개발 엔드포인트를 생성할 때 AWS Glue은(는) 퍼블릭 IP 주소만 반환합니다.

요청

- `MaxResults` – 1~1,000의 숫자(정수)입니다.

반환할 정보의 최대 크기.

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- `DevEndpoints` – [DevEndpoint](#) 객체의 배열입니다.

`DevEndpoint` 정의 목록입니다.

- `NextToken` – UTF-8 문자열입니다.

모든 `DevEndpoint` 정의가 아직 반환되지 않은 경우의 지속 토큰입니다.

Errors

- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

BatchGetDevEndpoints 작업(Python: `batch_get_dev_endpoints`)

주어진 개발 엔드포인트 이름 목록에 대한 리소스 메타데이터 목록을 반환합니다.

`ListDevEndpoints` 작업을 호출한 후에는 권한이 부여된 데이터에 액세스하기 위해 이 작업을 호출할 수 있습니다. 이 작업은 태그를 사용하는 권한 조건을 포함해 모든 IAM 권한을 지원합니다.

요청

- `customerAccountId` – UTF-8 문자열입니다.

AWS 계정 ID입니다.

- `DevEndpointNames` – 필수(Required): 1~25개 문자열의 UTF-8 문자열의 배열입니다.

`DevEndpoint` 이름(`ListDevEndpoint` 작업에서 반환된 이름일 수 있음)의 목록입니다.

응답

- `DevEndpoints` – [DevEndpoint](#) 객체의 배열입니다.

`DevEndpoint` 정의 목록입니다.

- `DevEndpointsNotFound` – 1~25개 문자열로 구성된 UTF-8 문자열의 배열입니다.

찾을 수 없는 `DevEndpoints` 목록입니다.

Errors

- `AccessDeniedException`
- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

ListDevEndpoints 작업(Python: `list_dev_endpoints`)

이 AWS 계정의 모든 `DevEndpoint` 리소스 또는 지정된 태그를 가진 리소스의 이름을 검색합니다. 이 작업을 통해 계정에서 사용 가능한 리소스와 그 이름을 확인할 수 있습니다.

이 작업을 수행하면 응답에서 필터로 사용할 수 있는 선택 사항인 `Tags` 필드가 검색되기 때문에 태그가 지정된 리소스를 하나의 그룹으로 검색할 수 있습니다. 태그 필터링을 사용하기로 선택하면 태그가 포함된 리소스만 검색됩니다.

요청

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적인 요청이라면 지속적인 토큰입니다.

- `MaxResults` – 1~1,000의 숫자(정수)입니다.

반환할 목록의 최대 크기.

- `Tags` – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이렇게 태그가 지정된 리소스만 반환하도록 지정합니다.

응답

- `DevEndpointNames` – UTF-8 문자열의 배열입니다.

계정의 모든 `DevEndpoint` 또는 지정된 태그를 가진 `DevEndpoint`의 이름입니다.

- `NextToken` – UTF-8 문자열입니다.

반환된 목록이 사용가능한 마지막 지표를 포함하지 경우의 연속 토큰입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

Schema Registry

스키마 레지스트리 API는 에서 스키마 작업과 관련된 데이터 유형 및 API를 설명합니다. AWS Glue

데이터 타입

- [RegistryId 구조](#)
- [RegistryListItem 구조](#)

- [MetadataInfo 구조](#)
- [OtherMetadataValueListItem 구조](#)
- [SchemaListItem 구조](#)
- [SchemaVersionListItem 구조](#)
- [MetadataKeyValuePair 구조](#)
- [SchemaVersionErrorItem 구조](#)
- [ErrorDetails 구조](#)
- [SchemaVersionNumber 구조](#)
- [Schemald 구조](#)

RegistryId 구조

레지스트리 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

필드

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

레지스트리의 이름입니다. 조회에만 사용됩니다. RegistryArn 또는 RegistryName 중 하나가 제공되어야 합니다.

- RegistryArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

업데이트할 레지스트리의 ARN입니다. RegistryArn 또는 RegistryName 중 하나가 제공되어야 합니다.

RegistryListItem 구조

레지스트리에 대한 세부 정보를 포함하는 구조입니다.

필드

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

레지스트리의 이름입니다.

- RegistryArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

레지스트리의 Amazon 리소스 이름(ARN)입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

레지스트리에 대한 설명입니다.

- Status – UTF-8 문자열입니다(유효한 값: AVAILABLE | DELETING).

레지스트리의 상태입니다.

- CreatedTime – UTF-8 문자열입니다.

레지스트리가 생성된 데이터입니다.

- UpdatedTime – UTF-8 문자열입니다.

레지스트리가 업데이트된 날짜입니다.

MetadataInfo 구조

스키마 버전에 대한 메타데이터 정보를 포함하는 구조입니다.

필드

- MetadataValue – [Custom string pattern #33](#)과(와) 일치하는 1~256바이트 길이의 UTF-8 문자열입니다.

메타데이터 키의 해당하는 값입니다.

- CreatedTime – UTF-8 문자열입니다.

항목이 생성된 시간입니다.

- OtherMetadataValueList – [OtherMetadataValueListItem](#) 객체의 배열입니다.

동일한 메타데이터 키에 속하는 다른 메타데이터입니다.

OtherMetadataValueListItem 구조

동일한 메타데이터 키에 속하는 스키마 버전에 대한 다른 메타데이터를 포함하는 구조입니다.

필드

- `MetadataValue` – [Custom string pattern #33](#)과(와) 일치하는 1~256바이트 길이의 UTF-8 문자열입니다.

동일한 메타데이터 키에 속하는 스키마 버전에 대한 다른 메타데이터를 포함하는 구조입니다.

- `CreatedTime` – UTF-8 문자열입니다.

항목이 생성된 시간입니다.

SchemaListItem 구조

스키마에 대한 최소한의 세부 정보를 포함하는 객체입니다.

필드

- `RegistryName` – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

스키마가 있는 레지스트리의 이름입니다.

- `SchemaName` – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

스키마의 이름입니다.

- `SchemaArn` – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN)입니다.

- `Description` – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

스키마에 대한 설명입니다.

- `SchemaStatus` – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | DELETING).

스키마의 상태입니다.

- `CreatedTime` – UTF-8 문자열입니다.

스키마가 생성된 날짜 및 시간입니다.

- `UpdatedTime` – UTF-8 문자열입니다.

스키마가 업데이트된 날짜 및 시간입니다.

SchemaVersionListItem 구조

스키마 버전에 대한 세부 정보를 포함하는 객체입니다.

필드

- `SchemaArn` – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN).

- `SchemaVersionId` – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전의 고유한 식별자입니다.

- `VersionNumber` - 1~100,000의 숫자(정수)입니다.

스키마의 버전 번호입니다.

- `Status` – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | FAILURE | DELETING).

스키마 버전의 상태입니다.

- `CreatedTime` – UTF-8 문자열입니다.

스키마 버전이 생성된 날짜 및 시간입니다.

MetadataKeyValuePair 구조

메타데이터에 대한 키 값 페어를 포함하는 구조입니다.

필드

- `MetadataKey` – [Custom string pattern #33](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

메타데이터 키입니다.

- `MetadataValue` – [Custom string pattern #33](#)과(와) 일치하는 1~256바이트 길이의 UTF-8 문자열입니다.

메타데이터 키의 해당하는 값입니다.

SchemaVersionErrorItem 구조

스키마 버전에 대한 작업에 대한 오류 세부 정보가 포함된 객체입니다.

필드

- `VersionNumber` - 1~100,000의 숫자(정수)입니다.

스키마의 버전 번호입니다.

- `ErrorDetails` - [ErrorDetails](#) 객체입니다.

스키마 버전에 대한 오류 세부 정보입니다.

ErrorDetails 구조

오류 세부 정보가 포함된 객체입니다.

필드

- `ErrorCode` - UTF-8 문자열입니다.

오류에 대한 오류 코드입니다.

- `ErrorMessage` - UTF-8 문자열입니다.

오류에 대한 오류 메시지입니다.

SchemaVersionNumber 구조

스키마 버전 정보를 포함하는 구조입니다.

필드

- `LatestVersion` - 부울입니다.

스키마에 사용할 수 있는 최신 버전입니다.

- `VersionNumber` - 1~100,000의 숫자(정수)입니다.

스키마의 버전 번호입니다.

Schemald 구조

스키마 레지스트리에 AWS Glue 있는 스키마의 고유 ID.

필드

- SchemaArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN). SchemaArn 또는 SchemaName 중 하나가 제공되어야 합니다.

- SchemaName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

스키마의 이름입니다. SchemaArn 또는 SchemaName 중 하나가 제공되어야 합니다.

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

해당 스키마가 포함된 스키마 레지스트리의 이름입니다.

운영

- [CreateRegistry](#) 액션 (Python: [레지스트리 만들기](#))
- [CreateSchema](#) 액션 (Python: [생성_스키마](#))
- [GetSchema](#) 액션 (파이썬: [get_schema](#))
- [ListSchemaVersions](#) 액션 (Python: [목록_스키마_버전](#))
- [GetSchemaVersion](#) 액션 (Python: [get_schema_version](#))
- [GetSchemaVersionsDiff](#) 액션 (Python: [get_schema_versions_diff](#))
- [ListRegistries](#) 액션 (Python: [리스트_레지스트리](#))
- [ListSchemas](#) 액션 (Python: [리스트_스키마](#))
- [RegisterSchemaVersion](#) 액션 (Python: [레지스터_스키마_버전](#))
- [UpdateSchema](#) 액션 (Python: [업데이트_스키마](#))
- [CheckSchemaVersionValidity](#) 조치 (Python: [검사_스키마_버전_유효성_검사](#))

- [UpdateRegistry](#) 액션 (Python: 업데이트_레지스트리)
- [GetSchemaByDefinition](#) 액션 (Python: get_schema_by_definition)
- [GetRegistry](#) 액션 (파이썬: get_registry)
- [PutSchemaVersionMetadata](#) 액션 (Python: put_schema_version_메타데이터)
- [QuerySchemaVersionMetadata](#) 액션 (Python: 쿼리_스키마_버전_메타데이터)
- [RemoveSchemaVersionMetadata](#) 액션 (Python: 스키마_버전_메타데이터 제거)
- [DeleteRegistry](#) 액션 (Python: 레지스트리 삭제)
- [DeleteSchema](#) 액션 (Python: 삭제_스키마)
- [DeleteSchemaVersions](#) 액션 (Python: 스키마_버전 삭제)

CreateRegistry 액션 (Python: 레지스트리 만들기)

스키마 컬렉션을 보유하는 데 사용할 수 있는 새 레지스트리를 생성합니다.

요청

- RegistryName – 필수(Required): [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

레지스트리 이름은 최대 255자이며 문자, 숫자, 하이픈, 밑줄, 달러 기호 또는 해시 표시만 포함할 수 있습니다. 공백은 없습니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

레지스트리에 대한 설명입니다. 설명이 제공되지 않으면 이에 대한 기본값이 없습니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

AWS 키 값 쌍을 포함하고 콘솔, 명령줄 또는 API로 검색할 수 있는 태그

응답

- RegistryArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

새로 생성된 레지스트리의 Amazon 리소스 이름(ARN)입니다.

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

레지스트리의 이름입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

레지스트리에 대한 설명입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

레지스트리에 대한 태그입니다.

Errors

- InvalidInputException
- AccessDeniedException
- AlreadyExistsException
- ResourceNumberLimitExceededException
- ConcurrentModificationException
- InternalServiceException

CreateSchema 액션 (Python: 생성_스키마)

새 스키마 집합을 생성하고 스키마 정의를 등록합니다. 실제로 버전을 등록하지 않고 스키마 집합이 이미 존재하는 경우 오류를 반환합니다.

스키마 집합이 생성되면 버전 체크포인트가 첫 번째 버전으로 설정됩니다. 호환성 모드 "DISABLED"는 첫 번째 스키마 버전 이후에 추가되는 모든 추가 스키마 버전을 제한합니다. 다른 모든 호환성 모드의 경우 호환성 설정의 검증은 RegisterSchemaVersion API를 사용할 때 두 번째 버전부터만 적용됩니다.

이 API가 RegistryId 없이 호출되면 레지스트리 데이터베이스 테이블에 "default-registry" 항목이 생성됩니다(아직 없는 경우).

요청

- RegistryId – [RegistryId](#) 객체입니다.

이것은 레지스트리 자격 증명 필드를 포함하는 래퍼 세이프입니다. 제공되지 않으면 기본 레지스트리가 사용됩니다. 동일한 ARN 포맷은 `arn:aws:glue:us-east-2:<customer id>:registry/default-registry:random-5-letter-id`입니다.

- SchemaName – 필수(Required): [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

스키마 이름은 최대 255자이며 문자, 숫자, 하이픈, 밑줄, 달러 기호 또는 해시 표시만 포함할 수 있습니다. 공백은 없습니다.

- DataFormat – 필수: UTF-8 문자열입니다(유효한 값: AVRO | JSON | PROTOBUF).

스키마 정의의 데이터 형식입니다. 현재 AVRO, JSON 및 PROTOBUF가 지원됩니다.

- Compatibility – UTF-8 문자열입니다(유효한 값: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL).

스키마의 호환성 모드입니다. 가능한 값은 다음과 같습니다.

- NONE: 호환 모드가 적용되지 않습니다. 개발 시나리오에서 또는 스키마에 적용할 호환성 모드를 모르는 경우 이 선택 사항을 사용할 수 있습니다. 추가된 모든 새 버전은 호환성 검사를 거치지 않고 수락됩니다.
- DISABLED: 이 호환성 선택 항목은 특정 스키마에 대한 버전 관리를 방지합니다. 이 선택 사항을 사용하여 스키마의 향후 버전 관리를 방지할 수 있습니다.
- BACKWARD: 이 호환성 선택 항목은 데이터 수신자가 현재 및 하나의 이전 스키마 버전을 모두 읽을 수 있도록 하므로 권장됩니다. 즉, 예를 들어 새 스키마 버전은 데이터 필드를 삭제하거나 이러한 필드의 유형을 변경할 수 없으므로 이전 버전을 사용하는 리더가 읽을 수 없습니다.
- BACKWARD_ALL: 이 호환성 선택을 통해 데이터 수신자는 현재 및 모든 이전 스키마 버전을 모두 읽을 수 있습니다. 필드를 삭제하거나 선택적 필드를 추가하고 모든 이전 스키마 버전과의 호환성을 확인해야 할 때 이 선택 사항을 사용할 수 있습니다.
- FORWARD: 이 호환성 선택을 통해 데이터 수신기는 현재 및 하나의 다음 스키마 버전을 모두 읽을 수 있지만 반드시 이후 버전은 아닙니다. 필드를 추가하거나 선택적 필드를 삭제해야 할 때 이 선택 사항을 사용할 수 있지만 마지막 스키마 버전과의 호환성만 확인합니다.

- **FORWARD_ALL**: 이 호환성 선택을 통해 데이터 수신기는 새로 등록된 스키마의 생산자가 작성한 내용을 읽을 수 있습니다. 필드를 추가하거나 선택적 필드를 삭제하고 모든 이전 스키마 버전과의 호환성을 확인해야 할 때 이 선택 사항을 사용할 수 있습니다.
- **FULL**: 이 호환성 선택을 통해 데이터 수신자는 이전 또는 다음 버전의 스키마를 사용하여 생산자가 작성한 데이터를 읽을 수 있지만 반드시 이전 또는 이후 버전은 아닙니다. 선택적 필드를 추가하거나 제거해야 할 때 이 선택 사항을 사용할 수 있지만 마지막 스키마 버전과의 호환성만 확인합니다.
- **FULL_ALL**: 이 호환성 선택을 통해 데이터 수신자는 모든 이전 스키마 버전을 사용하여 생산자가 작성한 데이터를 읽을 수 있습니다. 선택적 필드를 추가 또는 제거하고 모든 이전 스키마 버전과의 호환성을 확인해야 할 때 이 선택 사항을 사용할 수 있습니다.
- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

스키마에 대한 설명입니다(선택 사항). 설명이 제공되지 않으면 이에 대한 자동 기본값이 없습니다.

- **Tags** – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

AWS 키 값 쌍을 포함하고 콘솔, 명령줄 또는 API로 검색할 수 있는 태그 지정된 경우 AWS tags-on-create 패턴을 따릅니다.

- **SchemaDefinition** – [Custom string pattern #32](#)과(와) 일치하는 1~170,000바이트 길이의 UTF-8 문자열입니다.

SchemaName에 대해 DataFormat 설정을 사용하는 스키마 정의입니다.

응답

- **RegistryName** – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

레지스트리의 이름입니다.

- **RegistryArn** – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

레지스트리의 Amazon 리소스 이름(ARN)입니다.

- **SchemaName** – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

스키마의 이름입니다.

- **SchemaArn** – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN).

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

스키마에 대한 설명입니다(생성 시 지정한 경우).

- **DataFormat** – UTF-8 문자열입니다(유효한 값: AVRO | JSON | PROTOBUF).

스키마 정의의 데이터 형식입니다. 현재 AVRO, JSON 및 PROTOBUF가 지원됩니다.

- **Compatibility** – UTF-8 문자열입니다(유효한 값: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL).

스키마 호환성 모드입니다.

- **SchemaCheckpoint** - 1~100,000의 숫자(정수)입니다.

체크포인트의 버전 번호(호환성 모드가 마지막으로 변경된 시간)입니다.

- **LatestSchemaVersion** - 1~100,000의 숫자(정수)입니다.

반환된 스키마 정의와 연결된 스키마의 최신 버전입니다.

- **NextSchemaVersion** - 1~100,000의 숫자(정수)입니다.

반환된 스키마 정의와 연결된 스키마의 다음 버전입니다.

- **SchemaStatus** – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | DELETING).

스키마의 상태입니다.

- **Tags** – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

스키마의 태그입니다.

- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

첫 번째 스키마 버전의 고유 식별자입니다.

- SchemaVersionStatus – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | FAILURE | DELETING).

생성된 첫 번째 스키마 버전의 상태입니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- AlreadyExistsException
- ResourceNumberLimitExceededException
- ConcurrentModificationException
- InternalServiceException

GetSchema 액션 (파이썬: get_schema)

지정된 스키마를 자세히 설명합니다.

요청

- SchemaId – 필수(Required): [SchemaId](#) 객체입니다.

스키마 자격 증명 필드를 포함하는 래퍼 구조입니다. 구조에는 다음이 포함됩니다.

- SchemaId\$SchemaArn: 스키마의 아마존 리소스 이름 (ARN). SchemaArn 또는 SchemaName 및 RegistryName이 제공되어야 합니다.
- SchemaId\$SchemaName: 스키마의 이름. SchemaArn 또는 SchemaName 및 RegistryName이 제공되어야 합니다.

응답

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

레지스트리의 이름입니다.

- RegistryArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

레지스트리의 Amazon 리소스 이름(ARN)입니다.

- SchemaName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

스키마의 이름입니다.

- SchemaArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN).

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

스키마에 대한 설명입니다(생성 시 지정한 경우).

- DataFormat – UTF-8 문자열입니다(유효한 값: AVRO | JSON | PROTOBUF).

스키마 정의의 데이터 형식입니다. 현재 AVRO, JSON 및 PROTOBUF가 지원됩니다.

- Compatibility – UTF-8 문자열입니다(유효한 값: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL).

스키마의 호환성 모드입니다.

- SchemaCheckpoint - 1~100,000의 숫자(정수)입니다.

체크포인트의 버전 번호(호환성 모드가 마지막으로 변경된 시간)입니다.

- LatestSchemaVersion - 1~100,000의 숫자(정수)입니다.

반환된 스키마 정의와 연결된 스키마의 최신 버전입니다.

- NextSchemaVersion - 1~100,000의 숫자(정수)입니다.

반환된 스키마 정의와 연결된 스키마의 다음 버전입니다.

- `SchemaStatus` – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | DELETING).
스키마의 상태입니다.
- `CreatedTime` – UTF-8 문자열입니다.
스키마가 생성된 날짜 및 시간입니다.
- `UpdatedTime` – UTF-8 문자열입니다.
스키마가 업데이트된 날짜 및 시간입니다.

Errors

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

ListSchemaVersions 액션 (Python: 목록_스키마_버전)

최소한의 정보로 생성한 스키마 버전 목록을 반환합니다. [삭제됨(Deleted)] 상태의 스키마 버전은 결과에 포함되지 않습니다. 사용 가능한 스키마 버전이 없으면 빈 결과가 반환됩니다.

요청

- `SchemaId` – 필수(Required): [Schemald](#) 객체입니다.
스키마 자격 증명 필드를 포함하는 래퍼 구조입니다. 구조에는 다음이 포함됩니다.
 - `Schemald$SchemaArn`: 스키마의 아마존 리소스 이름 (ARN). `SchemaArn` 또는 `SchemaName` 및 `RegistryName`이 제공되어야 합니다.
 - `Schemald$SchemaName`: 스키마의 이름. `SchemaArn` 또는 `SchemaName` 및 `RegistryName`이 제공되어야 합니다.
- `MaxResults` – 1~100의 숫자(정수)입니다.
페이지당 필요한 최대 결과 수입니다. 값이 제공되지 않으면 페이지당 기본값이 25로 설정됩니다.
- `NextToken` – UTF-8 문자열입니다.
이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- Schemas – [SchemaVersionListItem](#) 객체의 배열입니다.

각 스키마 버전의 세부 정보를 포함하는 SchemaVersionList 객체의 배열입니다.

- NextToken – UTF-8 문자열입니다.

목록의 현재 세그먼트가 마지막이 아니면 반환된 토큰 목록에 페이지를 매기는 지속적인 토큰은 반환됩니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

GetSchemaVersion 액션 (Python: get_schema_version)

스키마 버전을 생성하거나 등록할 때 할당된 고유 ID로 지정된 스키마를 가져옵니다. [삭제됨(Deleted)] 상태의 스키마 버전은 결과에 포함되지 않습니다.

요청

- SchemaId – [Schemald](#) 객체입니다.

스키마 자격 증명 필드를 포함하는 래퍼 구조입니다. 구조에는 다음이 포함됩니다.

- Schemald\$SchemaArn: 스키마의 아마존 리소스 이름 (ARN). SchemaArn 또는 SchemaName 및 RegistryName이 제공되어야 합니다.
- Schemald\$SchemaName: 스키마의 이름. SchemaArn 또는 SchemaName 및 RegistryName이 제공되어야 합니다.
- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전의 SchemaVersionId입니다. 이 필드는 스키마 ID로 가져오기 위해 필요합니다. 이것 또는 SchemaId 래퍼를 제공해야 합니다.

- SchemaVersionNumber – [SchemaVersionNumber](#) 객체입니다.

스키마의 버전 번호입니다.

응답

- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전의 SchemaVersionId입니다.

- SchemaDefinition – [Custom string pattern #32](#)과(와) 일치하는 1~170,000바이트 길이의 UTF-8 문자열입니다.

스키마 ID에 대한 스키마 정의입니다.

- DataFormat – UTF-8 문자열입니다(유효한 값: AVRO | JSON | PROTOBUF).

스키마 정의의 데이터 형식입니다. 현재 AVRO, JSON 및 PROTOBUF가 지원됩니다.

- SchemaArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN).

- VersionNumber - 1~100,000의 숫자(정수)입니다.

스키마의 버전 번호입니다.

- Status – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | FAILURE | DELETING).

스키마 버전의 상태입니다.

- CreatedTime – UTF-8 문자열입니다.

스키마 버전이 생성된 날짜 및 시간입니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

GetSchemaVersionsDiff 액션 (Python: get_schema_versions_diff)

Schema Registry에 저장된 두 스키마 버전 간에 지정된 차이 유형의 스키마 버전 차이를 가져옵니다.

이 API를 사용하면 동일한 스키마 아래의 두 스키마 정의 간에 두 스키마 버전을 비교할 수 있습니다.

요청

- SchemaId – 필수(Required): [SchemaId](#) 객체입니다.

스키마 자격 증명 필드를 포함하는 래퍼 구조입니다. 구조에는 다음이 포함됩니다.

- Schemald\$SchemaArn: 스키마의 아마존 리소스 이름 (ARN). SchemaArn 또는 SchemaName 중 하나가 제공되어야 합니다.
- Schemald\$SchemaName: 스키마의 이름. SchemaArn 또는 SchemaName 중 하나가 제공되어야 합니다.
- FirstSchemaVersionNumber – 필수(Required): [SchemaVersionNumber](#) 객체입니다.

비교할 두 스키마 버전 중 첫 번째 버전입니다.

- SecondSchemaVersionNumber – 필수(Required): [SchemaVersionNumber](#) 객체입니다.

비교할 두 스키마 버전 중 두 번째 버전입니다.

- SchemaDiffType – 필수: UTF-8 문자열입니다(유효한 값: SYNTAX_DIFF).

현재 지원되는 diff 유형인 SYNTAX_DIFF를 나타냅니다.

응답

- Diff – [Custom string pattern #32](#)과(와) 일치하는 1~340,000바이트 길이의 UTF-8 문자열입니다.

문자열 JsonPatch 형식의 스키마 간의 차이.

Errors

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- InternalServiceException

ListRegistries 액션 (Python: 리스트_레지스트리)

최소한의 레지스트리 정보로 생성한 레지스트리 목록을 반환합니다. Deleting 상태의 레지스트리는 결과에 포함되지 않습니다. 사용 가능한 레지스트리가 없으면 빈 결과가 반환됩니다.

요청

- MaxResults – 1~100의 숫자(정수)입니다.

페이지당 필요한 최대 결과 수입니다. 값이 제공되지 않으면 페이지당 기본값이 25로 설정됩니다.

- NextToken – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- Registries – [RegistryListItem](#) 객체의 배열입니다.

각 레지스트리에 대한 최소한의 세부 정보를 포함하는 RegistryDetailedListItem 객체의 배열입니다.

- NextToken – UTF-8 문자열입니다.

목록의 현재 세그먼트가 마지막이 아니면 반환된 토큰 목록에 페이지를 매기는 지속적인 토큰은 반환됩니다.

Errors

- InvalidInputException
- AccessDeniedException
- InternalServiceException

ListSchemas 액션 (Python: 리스트_스키마)

최소한의 세부 정보가 포함된 스키마 목록을 반환합니다. [삭제 중(Deleting)] 상태의 스키마는 결과에 포함되지 않습니다. 사용 가능한 스키마가 없으면 빈 결과가 반환됩니다.

RegistryId가 제공되지 않으면 레지스트리 전체의 모든 스키마가 API 응답의 일부가 됩니다.

요청

- RegistryId – [RegistryId](#) 객체입니다.

레지스트리 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

- MaxResults – 1~100의 숫자(정수)입니다.

페이지당 필요한 최대 결과 수입니다. 값이 제공되지 않으면 페이지당 기본값이 25로 설정됩니다.

- NextToken – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- Schemas – [SchemaListItem](#) 객체의 배열입니다.

각 스키마의 세부 정보를 포함하는 SchemaListItem 객체의 배열입니다.

- NextToken – UTF-8 문자열입니다.

목록의 현재 세그먼트가 마지막이 아니면 반환된 토큰 목록에 페이지를 매기는 지속적인 토큰은 반환됩니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

RegisterSchemaVersion 액션 (Python: 레지스터_스키마_버전)

기존 스키마에 새 버전을 추가합니다. 새 버전의 스키마가 스키마 집합의 호환성 요구 사항을 충족하지 않는 경우 오류를 반환합니다. 이 API는 새 스키마 집합을 생성하지 않으며 스키마 집합이 Schema Registry에 아직 없는 경우 404 오류를 반환합니다.

이것이 Schema Registry에 등록되는 첫 번째 스키마 정의인 경우 이 API는 스키마 버전을 저장하고 즉시 반환합니다. 그렇지 않으면 이 호출은 호환성 모드로 인해 다른 작업보다 더 오래 실행될 가능성이

있습니다. SchemaVersionId로 GetSchemaVersion API를 호출하여 호환 모드를 확인할 수 있습니다.

동일한 스키마 정의가 이미 Schema Registry에 버전으로 저장되어 있는 경우 기존 스키마의 스키마 ID가 호출자에게 반환됩니다.

요청

- SchemaId – 필수(Required): [Schemald](#) 객체입니다.

스키마 자격 증명 필드를 포함하는 래퍼 구조입니다. 구조에는 다음이 포함됩니다.

- Schemald\$SchemaArn: 스키마의 아마존 리소스 이름 (ARN). SchemaArn 또는 SchemaName 및 RegistryName이 제공되어야 합니다.
- Schemald\$SchemaName: 스키마의 이름. SchemaArn 또는 SchemaName 및 RegistryName이 제공되어야 합니다.
- SchemaDefinition – 필수(Required): [Custom string pattern #32](#)과(와) 일치하는 1~170,000바이트 길이의 UTF-8 문자열입니다.

SchemaName에 대해 DataFormat 설정을 사용하는 스키마 정의입니다.

응답

- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

이 스키마의 버전을 나타내는 고유 ID입니다.

- VersionNumber - 1~100,000의 숫자(정수)입니다.

이 스키마의 버전입니다(동기화 흐름 전용, 이것이 첫 번째 버전인 경우).

- Status – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | FAILURE | DELETING).

스키마 버전의 상태입니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException

- ResourceNumberLimitExceededException
- ConcurrentModificationException
- InternalServiceException

UpdateSchema 액션 (Python: 업데이트_스키마)

스키마 집합에 대한 설명, 호환성 설정 또는 버전 검사점을 업데이트합니다.

호환성 설정을 업데이트하기 위해 호출은 새 호환성 설정을 사용하여 전체 스키마 버전 집합에 대한 호환성을 확인하지 않습니다. Compatibility 값이 제공되면 VersionNumber(체크포인트)도 필요합니다. API는 일관성을 위해 체크포인트 버전 번호를 검증합니다.

VersionNumber(체크포인트) 값이 제공되면 Compatibility는 선택 사항이며 스키마에 대한 검사점을 설정/재설정하는 데 사용할 수 있습니다.

이 업데이트는 스키마가 AVAILABLE 상태인 경우에만 발생합니다.

요청

- SchemaId – 필수(Required): [Schemald](#) 객체입니다.

스키마 자격 증명 필드를 포함하는 래퍼 구조입니다. 구조에는 다음이 포함됩니다.

- Schemald\$SchemaArn: 스키마의 아마존 리소스 이름 (ARN). SchemaArn 또는 SchemaName 중 하나가 제공되어야 합니다.
- Schemald\$SchemaName: 스키마의 이름. SchemaArn 또는 SchemaName 중 하나가 제공되어야 합니다.
- SchemaVersionNumber – [SchemaVersionNumber](#) 객체입니다.

체크 포인팅에 필요한 버전 번호입니다. VersionNumber 또는 Compatibility 중 하나가 제공되어야 합니다.

- Compatibility – UTF-8 문자열입니다(유효한 값: NONE | DISABLED | BACKWARD | BACKWARD_ALL | FORWARD | FORWARD_ALL | FULL | FULL_ALL).

스키마에 대한 새 호환성 설정입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

스키마에 대한 새 설명입니다.

응답

- SchemaArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN).

- SchemaName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

스키마의 이름입니다.

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

해당 스키마가 포함된 레지스트리의 이름입니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ConcurrentModificationException
- InternalServiceException

CheckSchemaVersionValidity 조치 (Python: 검사_스키마_버전_유효성 검사)

제공된 스키마를 검증합니다. 이 호출은 부작용이 없으며 단순히 DataFormat을 포맷으로 사용하여 제공된 스키마를 사용하여 검증합니다. 스키마 집합 이름을 사용하지 않으므로 호환성 검사가 수행되지 않습니다.

요청

- DataFormat – 필수: UTF-8 문자열입니다(유효한 값: AVRO | JSON | PROTOBUF).

스키마 정의의 데이터 형식입니다. 현재 AVRO, JSON 및 PROTOBUF가 지원됩니다.

- SchemaDefinition – 필수(Required): [Custom string pattern #32](#)과(와) 일치하는 1~170,000바이트 길이의 UTF-8 문자열입니다.

검증해야 하는 스키마의 정의입니다.

응답

- Valid – 부울입니다.

스키마가 유효하면 true를 반환하고 그렇지 않으면 false를 반환합니다.

- Error – 1~5,000바이트 길이의 UTF-8 문자열입니다.

검증 실패 메시지입니다.

Errors

- InvalidInputException
- AccessDeniedException
- InternalServiceException

UpdateRegistry 액션 (Python: 업데이트_레지스트리)

스키마 컬렉션을 보유하는 데 사용되는 기존 레지스트리를 업데이트합니다. 업데이트된 속성은 레지스트리와 관련되며 레지스트리 내의 스키마를 수정하지 않습니다.

요청

- RegistryId – 필수(Required): [RegistryId](#) 객체입니다.

이는 레지스트리 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

- Description – 필수(Required): [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

레지스트리에 대한 설명입니다. 설명이 제공되지 않으면 이 필드는 업데이트되지 않습니다.

응답

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트된 레지스트리의 이름입니다.

- RegistryArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

업데이트된 레지스트리의 Amazon 리소스 이름(ARN)입니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- ConcurrentModificationException
- InternalServiceException

GetSchemaByDefinition 액션 (Python: `get_schema_by_definition`)

SchemaDefinition으로 스키마를 검색합니다. 스키마 정의는 Schema Registry로 전송되고 정규화되고 해시됩니다. 해시가 SchemaName 또는 ARN의 범위 내에서 일치하는 경우(또는 제공되지 않은 경우 기본 레지스트리) 해당 스키마의 메타데이터가 반환됩니다. 그렇지 않으면 404 또는 오류가 반환됩니다. NotFound Deleted 상태의 스키마 버전은 결과에 포함되지 않습니다.

요청

- SchemaId – 필수(Required): [Schemald](#) 객체입니다.

스키마 자격 증명 필드를 포함하는 래퍼 구조입니다. 구조에는 다음이 포함됩니다.

- Schemald\$SchemaArn: 스키마의 아마존 리소스 이름 (ARN). SchemaArn 또는 SchemaName 중 하나가 제공되어야 합니다.
- Schemald\$SchemaName: 스키마의 이름. SchemaArn 또는 SchemaName 중 하나가 제공되어야 합니다.
- SchemaDefinition – 필수(Required): [Custom string pattern #32](#)과(와) 일치하는 1~170,000바이트 길이의 UTF-8 문자열입니다.

스키마 세부 정보가 필요한 스키마의 정의입니다.

응답

- `SchemaVersionId` – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전의 스키마 ID입니다.

- `SchemaArn` – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN).

- `DataFormat` – UTF-8 문자열입니다(유효한 값: AVRO | JSON | PROTOBUF).

스키마 정의의 데이터 형식입니다. 현재 AVRO, JSON 및 PROTOBUF가 지원됩니다.

- `Status` – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | FAILURE | DELETING).

스키마 버전의 상태입니다.

- `CreatedTime` – UTF-8 문자열입니다.

스키마가 생성된 날짜 및 시간입니다.

Errors

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`
- `InternalServiceException`

GetRegistry 액션 (파이썬: `get_registry`)

지정된 레지스트리를 자세히 설명합니다.

요청

- `RegistryId` – 필수(Required): [RegistryId](#) 객체입니다.

이는 레지스트리 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

응답

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

레지스트리의 이름입니다.

- RegistryArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

레지스트리의 Amazon 리소스 이름(ARN)입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

레지스트리에 대한 설명입니다.

- Status – UTF-8 문자열입니다(유효한 값: AVAILABLE | DELETING).

레지스트리의 상태입니다.

- CreatedTime – UTF-8 문자열입니다.

레지스트리가 생성된 날짜 및 시간입니다.

- UpdatedTime – UTF-8 문자열입니다.

레지스트리가 업데이트된 날짜 및 시간입니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException
- InternalServiceException

PutSchemaVersionMetadata 액션 (Python: put_schema_version_메타데이터)

지정된 스키마 버전 ID에 대한 메타데이터 키 값 페어를 넣습니다. 스키마 버전당 최대 10개의 키 값 페어가 허용됩니다. 하나 이상의 호출을 통해 추가할 수 있습니다.

요청

- SchemaId – [SchemaId](#) 객체입니다.
스키마의 고유 ID입니다.
- SchemaVersionNumber – [SchemaVersionNumber](#) 객체입니다.
스키마의 버전 번호입니다.
- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.
스키마 버전의 고유한 버전 ID입니다.
- MetadataKeyValue – 필수(Required): [MetadataKeyValuePair](#) 객체입니다.
메타데이터 키의 해당하는 값입니다.

응답

- SchemaArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.
스키마의 Amazon 리소스 이름(ARN)입니다.
- SchemaName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
스키마의 이름입니다.
- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
레지스트리의 이름입니다.
- LatestVersion – 부울입니다.
스키마의 최신 버전입니다.
- VersionNumber - 1~100,000의 숫자(정수)입니다.
스키마의 버전 번호입니다.
- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전의 고유한 버전 ID입니다.

- MetadataKey – [Custom string pattern #33](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

메타데이터 키입니다.

- MetadataValue – [Custom string pattern #33](#)과(와) 일치하는 1~256바이트 길이의 UTF-8 문자열입니다.

메타데이터 키의 값입니다.

Errors

- InvalidInputException
- AccessDeniedException
- AlreadyExistsException
- EntityNotFoundException
- ResourceNumberLimitExceededException

QuerySchemaVersionMetadata 액션 (Python: 쿼리_스키마_버전_메타데이터)

스키마 버전 메타데이터 정보에 대한 쿼리입니다.

요청

- SchemaId – [SchemaId](#) 객체입니다.

스키마 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

- SchemaVersionNumber – [SchemaVersionNumber](#) 객체입니다.

스키마의 버전 번호입니다.

- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전의 고유한 버전 ID입니다.

- MetadataList – [MetadataKeyValuePair](#) 객체의 배열입니다.

메타데이터에 대한 키-값 페어를 검색합니다. 제공되지 않으면 모든 메타데이터 정보를 가져옵니다.

- `MaxResults` – 1~50의 숫자(정수)입니다.

페이지당 필요한 최대 결과 수입니다. 값이 제공되지 않으면 페이지당 기본값이 25로 설정됩니다.

- `NextToken` – UTF-8 문자열입니다.

이것이 지속적으로 호출되면 지속적인 토큰입니다.

응답

- `MetadataInfoMap` – 키-값 페어의 맵 배열입니다.

각 키는 [Custom string pattern #33](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

각 값은 [MetadataInfo](#) 객체입니다.

메타데이터 키 및 관련 값의 맵입니다.

- `SchemaVersionId` – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전의 고유한 버전 ID입니다.

- `NextToken` – UTF-8 문자열입니다.

목록의 현재 세그먼트가 마지막이 아니면 반환된 토큰 목록에 페이지를 매기는 지속적인 토큰은 반환됩니다.

Errors

- `InvalidInputException`
- `AccessDeniedException`
- `EntityNotFoundException`

RemoveSchemaVersionMetadata 액션 (Python: 스키마_버전_메타데이터 제거)

지정된 스키마 버전 ID에 대한 스키마 버전 메타데이터에서 키-값 페어를 제거합니다.

요청

- SchemaId – [SchemaId](#) 객체입니다.

스키마 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

- SchemaVersionNumber – [SchemaVersionNumber](#) 객체입니다.

스키마의 버전 번호입니다.

- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전의 고유한 버전 ID입니다.

- MetadataKeyValue – 필수(Required): [MetadataKeyValuePair](#) 객체입니다.

메타데이터 키의 값입니다.

응답

- SchemaArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

스키마의 Amazon 리소스 이름(ARN).

- SchemaName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

스키마의 이름입니다.

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

레지스트리의 이름입니다.

- LatestVersion – 부울입니다.

스키마의 최신 버전입니다.

- VersionNumber - 1~100,000의 숫자(정수)입니다.

스키마의 버전 번호입니다.

- SchemaVersionId – [Custom string pattern #17](#)과(와) 일치하는 36바이트 이상 길이의 UTF-8 문자열입니다.

스키마 버전에 대한 버전 ID입니다.

- MetadataKey – [Custom string pattern #33](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

메타데이터 키입니다.

- MetadataValue – [Custom string pattern #33](#)과(와) 일치하는 1~256바이트 길이의 UTF-8 문자열입니다.

메타데이터 키의 값입니다.

Errors

- InvalidInputException
- AccessDeniedException
- EntityNotFoundException

DeleteRegistry 액션 (Python: 레지스트리 삭제)

스키마와 모든 해당 버전을 포함한 전체 레지스트리를 삭제합니다. 삭제 작업의 상태를 얻으려면 비동기 호출 후 GetRegistry API를 호출할 수 있습니다. 레지스트리를 삭제하면 UpdateRegistry, CreateSchema, UpdateSchema 및 RegisterSchemaVersion API와 같은 레지스트리에 대한 모든 온라인 작업이 비활성화됩니다.

요청

- RegistryId – 필수(Required): [RegistryId](#) 객체입니다.

이는 레지스트리 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

응답

- RegistryName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제되는 레지스트리의 이름입니다.

- RegistryArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

삭제 중인 레지스트리의 Amazon 리소스 이름(ARN)입니다.

- Status – UTF-8 문자열입니다(유효한 값: AVAILABLE | DELETING).

레지스트리의 상태입니다. 작업이 성공하면 DeletIng 상태가 반환됩니다.

Errors

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- ConcurrentModificationException

DeleteSchema 액션 (Python: 삭제_스키마)

스키마 집합 및 모든 해당 버전을 포함하여 전체 스키마 집합을 삭제합니다. 삭제 작업의 상태를 얻으려면 비동기 호출 후 GetSchema API를 호출할 수 있습니다. 레지스트리를 삭제하면 GetSchemaByDefinition 및 RegisterSchemaVersion API와 같은 스키마에 대한 모든 온라인 작업이 비활성화됩니다.

요청

- SchemaId – 필수(Required): [Schemald](#) 객체입니다.

이는 스키마 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

응답

- SchemaArn – [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

삭제 중인 스키마의 Amazon 리소스 이름(ARN)입니다.

- SchemaName – [Custom string pattern #18](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제 중인 스키마의 이름입니다.

- Status – UTF-8 문자열입니다(유효한 값: AVAILABLE | PENDING | DELETING).

스키마의 상태입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `AccessDeniedException`
- `ConcurrentModificationException`

DeleteSchemaVersions 액션 (Python: 스키마_버전 삭제)

지정된 스키마에서 버전을 제거합니다. 버전 번호 또는 범위가 제공될 수 있습니다. 호환 모드가 BACKWARDS_FULL과 같이 필요한 버전의 삭제를 금지하는 경우 오류가 반환됩니다. 이 호출 후 GetSchemaVersions API를 호출하면 삭제된 버전의 상태가 나열됩니다.

버전 번호 범위에 체크포인트 버전이 포함된 경우 API는 409 충돌을 반환하고 삭제를 진행하지 않습니다. 이 API를 사용하기 전에 먼저 DeleteSchemaCheckpoint API를 사용하여 체크포인트를 제거해야 합니다.

DeleteSchemaVersions API를 사용하여 스키마 집합의 첫 번째 스키마 버전을 삭제할 수 없습니다. 첫 번째 스키마 버전은 DeleteSchema API로만 삭제할 수 있습니다. 이 작업은 스키마 버전 아래에 첨부된 SchemaVersionMetadata도 삭제합니다. 영구 삭제는 데이터베이스에 적용됩니다.

호환 모드가 BACKWARDS_FULL과 같이 필요한 버전의 삭제를 금지하는 경우 오류가 반환됩니다.

요청

- `SchemaId` – 필수(Required): [Schemald](#) 객체입니다.

이는 스키마 이름과 Amazon 리소스 이름(ARN)을 포함할 수 있는 래퍼 구조입니다.

- `Versions` – 필수(Required): [Custom string pattern #34](#)과(와) 일치하는 1~100,000바이트 길이의 UTF-8 문자열입니다.

다음 포맷의 버전 범위가 제공될 수 있습니다.

- 단일 버전 번호, 5
- 범위, 5~8 : 버전 5, 6, 7, 8을 삭제합니다.

응답

- SchemaVersionErrors – [SchemaVersionErrorItem](#) 객체의 배열입니다.
각각 오류 및 스키마 버전을 포함하는 SchemaVersionErrorItem 객체 목록입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- AccessDeniedException
- ConcurrentModificationException

워크플로

워크플로 API는 AWS Glue에서의 워크플로 생성, 업데이트 또는 확인과 관련된 API 및 데이터 유형에 대해 설명합니다. 워크플로 및 작업 실행의 경우 작업 실행 기록을 90일 동안 액세스할 수 있습니다.

데이터 타입

- [JobNodeDetails 구조](#)
- [CrawlerNodeDetails 구조](#)
- [TriggerNodeDetails 구조](#)
- [크롤 구조](#)
- [노드 구조](#)
- [엣지 구조](#)
- [워크플로우 구조](#)
- [WorkflowGraph 구조](#)
- [WorkflowRun 구조](#)
- [WorkflowRunStatistics 구조](#)
- [StartingEventBatchCondition 구조](#)
- [블루프린트 구조](#)
- [BlueprintDetails 구조](#)
- [LastActiveDefinition 구조](#)

- [BlueprintRun 구조](#)

JobNodeDetails 구조

워크플로에 있는 작업 노드의 세부 정보입니다.

필드

- JobRuns – [JobRun](#) 객체의 배열입니다.
작업 노드가 나타내는 작업 실행에 대한 정보입니다.

CrawlerNodeDetails 구조

워크플로에 있는 크롤러 노드의 세부 정보입니다.

필드

- Crawls – [Crawl](#) 객체의 배열입니다.
크롤 노드가 나타내는 크롤의 목록입니다.

TriggerNodeDetails 구조

워크플로에 있는 트리거 노드의 세부 정보입니다.

필드

- Trigger – [트리거](#) 객체입니다.
트리거 노드가 나타내는 트리거의 정보입니다.

크롤 구조

워크플로에 있는 크롤의 세부 정보입니다.

필드

- State – UTF-8 문자열입니다(유효 값: RUNNING | CANCELLING | CANCELLED | SUCCEEDED | FAILED | ERROR).

크롤러의 상태입니다.

- StartedOn – 타임스탬프입니다.

크롤이 시작된 날짜와 시간입니다.

- CompletedOn – 타임스탬프입니다.

크롤이 완료된 날짜와 시간입니다.

- ErrorMessage – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

크롤과 연결된 오류 메시지입니다.

- LogGroup – [Log group string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.

크롤과 연결된 로그 그룹입니다.

- LogStream – [Log-stream string pattern](#)과(와) 일치하는 1~512바이트 길이의 UTF-8 문자열입니다.

크롤과 연결된 로그 스트림입니다.

노드 구조

노드는 워크플로 그래프에서 AWS Glue 구성 요소(트리거, 크롤러 또는 작업)를 나타냅니다.

필드

- Type – UTF-8 문자열입니다(유효한 값: CRAWLER | JOB | TRIGGER).

노드가 나타내는 AWS Glue 구성 요소의 유형입니다.

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

노드가 나타내는 AWS Glue 구성 요소의 이름입니다.

- UniqueId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

워크플로 내의 노드에 할당된 고유 ID입니다.

- TriggerDetails – [TriggerNodeDetails](#) 객체입니다.

노드가 트리거를 나타낼 때 트리거의 세부 정보입니다.

- JobDetails – [JobNodeDetails](#) 객체입니다.

노드가 작업을 나타낼 때 작업의 세부 정보입니다.

- `CrawlerDetails` – [CrawlerNodeDetails](#) 객체입니다.

노드가 크롤러를 나타낼 때 크롤러의 세부 정보입니다.

엣지 구조

엣지는 엣지가 속한 워크플로의 일부인 두 개의 AWS Glue 구성 요소 간에 방향이 있는 연결을 나타냅니다.

필드

- `SourceId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

엣지가 시작되는 워크플로 내 노드의 고유 ID입니다.

- `DestinationId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

엣지가 종료되는 워크플로 내 노드의 고유 ID입니다.

워크플로우 구조

워크플로는 복잡한 ETL 태스크를 완료하기 위해 실행되는 여러 종속 AWS Glue 작업 및 크롤러의 모음입니다. 워크플로는 모든 작업 및 크롤러의 실행 및 모니터링을 관리합니다.

필드

- `Name` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

워크플로의 이름입니다.

- `Description` – UTF-8 문자열입니다.

워크플로에 대한 설명입니다.

- `DefaultRunProperties` – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

워크플로의 각 실행의 일부로 사용할 속성 모음입니다. 실행 속성은 워크플로의 각 작업에서 사용할 수 있습니다. 작업은 흐름의 다음 작업에 대한 속성을 수정할 수 있습니다.

- **CreatedOn** – 타임스탬프입니다.

워크플로가 생성된 날짜와 시간입니다.

- **LastModifiedOn** – 타임스탬프입니다.

워크플로가 마지막으로 수정된 날짜와 시간입니다.

- **LastRun** – [WorkflowRun](#) 객체입니다.

워크플로의 마지막 실행에 대한 정보입니다.

- **Graph** – [WorkflowGraph](#) 객체입니다.

워크플로에 속하는 모든 AWS Glue 구성 요소를 이들 간의 방향 있는 연결과 노드를 엣지로 나타내는 그래프입니다.

- **CreationStatus** – UTF-8 문자열입니다(유효 값: CREATING | CREATED | CREATION_FAILED).

워크플로의 생성 상태입니다.

- **MaxConcurrentRuns** - 숫자(정수)입니다.

이 파라미터를 사용하여 데이터에 대한 원치 않는 다중 업데이트를 방지하거나 비용을 제어하거나 경우에 따라 구성 요소 작업의 최대 동시 실행 수를 초과하는 것을 방지할 수 있습니다. 이 파라미터를 공백으로 두면 동시 워크플로 실행 수에 제한이 없습니다.

- **BlueprintDetails** – [BlueprintDetails](#) 객체입니다.

이 구조는 이 특정 워크플로가 생성된 블루프린트의 세부 정보를 나타냅니다.

WorkflowGraph 구조

워크플로 그래프는 워크플로에 있는 모든 AWS Glue 구성 요소와 그 사이의 모든 방향 있는 연결이 포함된 완전한 워크플로를 나타냅니다.

필드

- **Nodes** – [노드](#) 객체의 배열입니다.

노드로 나타내는 워크플로에 속한 AWS Glue 구성 요소의 목록입니다.

- Edges – [Edge](#) 객체의 배열입니다.

워크플로에 속한 노드 간의 모든 방향 있는 연결의 목록입니다.

WorkflowRun 구조

워크플로 실행은 모든 실행 시간 정보를 제공하는 워크플로의 실행입니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행된 워크플로의 이름입니다.

- WorkflowRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 워크플로 실행의 ID입니다.

- PreviousRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이전 워크플로 실행의 ID입니다.

- WorkflowRunProperties – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

실행 중에 설정된 워크플로 실행 속성입니다.

- StartedOn – 타임스탬프입니다.

워크플로 실행이 시작된 날짜와 시간입니다.

- CompletedOn – 타임스탬프입니다.

워크플로 실행이 완료된 날짜와 시간입니다.

- Status – UTF-8 문자열입니다(유효 값: RUNNING | COMPLETED | STOPPING | STOPPED | ERROR).

워크플로 실행의 상태입니다.

- ErrorMessage – UTF-8 문자열입니다.

이 오류 메시지는 워크플로 실행을 시작할 때 발생할 수 있는 모든 오류에 대해 설명합니다. 현재 유일한 오류 메시지는 "워크플로에 대한 동시 실행 초과(Concurrent runs exceeded for workflow): foo"입니다.

- Statistics – [WorkflowRunStatistics](#) 객체입니다.

실행의 통계입니다.

- Graph – [WorkflowGraph](#) 객체입니다.

워크플로에 속하는 모든 AWS Glue 구성 요소를 이들 간의 방향 있는 연결과 노드를 엣지로 나타내는 그래프입니다.

- StartingEventBatchCondition – [StartingEventBatchCondition](#) 객체입니다.

워크플로 실행을 시작한 배치 조건입니다.

WorkflowRunStatistics 구조

워크플로 실행 통계는 워크플로 실행에 대한 통계를 제공합니다.

필드

- TotalActions - 숫자(정수)입니다.

워크플로 실행에 있는 총 작업 수입니다.

- TimeoutActions - 숫자(정수)입니다.

시간 초과된 총 작업 수입니다.

- FailedActions - 숫자(정수)입니다.

실패한 총 작업 수입니다.

- StoppedActions - 숫자(정수)입니다.

중지된 총 작업 수입니다.

- SucceededActions - 숫자(정수)입니다.

성공한 총 작업 수입니다.

- RunningActions - 숫자(정수)입니다.

실행 상태의 총 작업 수입니다.

- **ErroredActions** - 숫자(정수)입니다.
워크플로 실행에서 ERROR 상태의 작업 실행 수를 나타냅니다.
- **WaitingActions** - 숫자(정수)입니다.
워크플로 실행에서 WAITING 상태의 작업 실행 수를 나타냅니다.

StartingEventBatchCondition 구조

워크플로 실행을 시작한 배치 조건입니다. **BatchSize** 멤버가 0이 아닌 경우 배치 크기의 이벤트 수가 도착했거나 **BatchWindow** 멤버가 0이 아닌 경우 배치 기간이 만료되었습니다.

필드

- **BatchSize** - 숫자(정수)입니다.
배치의 이벤트 수입니다.
- **BatchWindow** - 숫자(정수)입니다.
배치 기간의 기간(초)입니다.

블루프린트 구조

블루프린트의 세부 정보입니다.

필드

- **Name** – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.
블루프린트의 이름입니다.
- **Description** – 1~512바이트 길이의 UTF-8 문자열입니다.
블루프린트에 대한 설명입니다.
- **CreatedOn** – 타임스탬프입니다.
블루프린트가 등록된 날짜 및 시간입니다.
- **LastModifiedOn** – 타임스탬프입니다.
블루프린트가 마지막으로 수정된 날짜 및 시간입니다.

- **ParameterSpec** – 1~131,072바이트 길이의 UTF-8 문자열입니다.

블루프린트에 대한 파라미터 사양 목록을 나타내는 JSON 문자열입니다.

- **BlueprintLocation** – UTF-8 문자열입니다.

블루프린트가 게시되는 Amazon S3의 경로를 지정합니다.

- **BlueprintServiceLocation** – UTF-8 문자열입니다.

CreateBlueprint/UpdateBlueprint를 호출하여 AWS Glue에 블루프린트를 등록할 때 블루프린트가 복사되는 Amazon S3의 경로를 지정합니다.

- **Status** – UTF-8 문자열입니다(유효 값: CREATING | ACTIVE | UPDATING | FAILED).

블루프린트 등록의 상태입니다.

- [생성(Creating)] - 블루프린트 등록이 진행 중입니다.
- [활성(Active)] - 블루프린트가 성공적으로 등록되었습니다.
- [업데이트 중(Updating)] - 블루프린트 등록에 대한 업데이트가 진행 중입니다.
- [실패(Failed)] - 블루프린트 등록에 실패했습니다.

- **ErrorMessage** – UTF-8 문자열입니다.

오류 메시지입니다.

- **LastActiveDefinition** – [LastActiveDefinition](#) 객체입니다.

블루프린트의 여러 버전이 있고 최신 버전에 일부 오류가 있는 경우 이 속성은 서비스에서 사용할 수 있는 마지막으로 성공한 블루프린트 정의를 나타냅니다.

BlueprintDetails 구조

블루프린트의 세부 정보입니다.

필드

- **BlueprintName** – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

블루프린트의 이름입니다.

- **RunId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 블루프린트의 실행 ID입니다.

LastActiveDefinition 구조

블루프린트의 여러 버전이 있고 최신 버전에 일부 오류가 있는 경우 이 속성은 서비스에서 사용할 수 있는 마지막으로 성공한 블루프린트 정의를 나타냅니다.

필드

- **Description** – 1~512바이트 길이의 UTF-8 문자열입니다.

블루프린트에 대한 설명입니다.

- **LastModifiedOn** – 타임스탬프입니다.

블루프린트가 마지막으로 수정된 날짜 및 시간입니다.

- **ParameterSpec** – 1~131,072바이트 길이의 UTF-8 문자열입니다.

블루프린트의 파라미터를 지정하는 JSON 문자열입니다.

- **BlueprintLocation** – UTF-8 문자열입니다.

AWS Glue 개발자가 블루프린트를 게시하는 Amazon S3의 경로를 지정합니다.

- **BlueprintServiceLocation** – UTF-8 문자열입니다.

블루프린트를 생성하거나 업데이트할 때 블루프린트가 복사되는 Amazon S3의 경로를 지정합니다.

BlueprintRun 구조

블루프린트 실행의 세부 정보입니다.

필드

- **BlueprintName** – [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

블루프린트의 이름입니다.

- **RunId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 블루프린트 실행의 실행 ID입니다.

- **WorkflowName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

성공적인 블루프린트 실행의 결과로 생성된 워크플로의 이름입니다. 블루프린트 실행에 오류가 있으면 워크플로가 생성되지 않습니다.

- State – UTF-8 문자열입니다(유효 값: RUNNING | SUCCEEDED | FAILED | ROLLING_BACK).

블루프린트 실행 상태입니다. 가능한 값은 다음과 같습니다.

- [실행 중(Running)] - 블루프린트 실행이 진행 중입니다.
- [성공(Succeeded)] - 블루프린트 실행이 성공적으로 완료되었습니다.
- [실패(Failed)] - 블루프린트 실행이 실패하고 롤백이 완료되었습니다.
- [롤백 중(Rolling Back)] - 블루프린트 실행이 실패하여 롤백이 진행 중입니다.
- StartedOn – 타임스탬프입니다.

블루프린트 실행이 시작된 날짜 및 시간입니다.

- CompletedOn – 타임스탬프입니다.

블루프린트 실행이 완료된 날짜 및 시간입니다.

- ErrorMessage – UTF-8 문자열입니다.

블루프린트를 실행하는 동안 표시되는 모든 오류를 나타냅니다.

- RollbackErrorMessage – UTF-8 문자열입니다.

워크플로의 엔터티를 생성하는 동안 오류가 발생하면 해당 시점까지 생성된 엔터티를 롤백하고 삭제하려고 합니다. 이 속성은 생성된 엔터티를 삭제하는 동안 발생한 오류를 나타냅니다.

- Parameters – 1~131,072바이트 길이의 UTF-8 문자열입니다.

블루프린트 파라미터(문자열). Blueprint\$ParameterSpec에 정의된 파라미터 사양에서 필요한 각 키에 대한 값을 제공해야 합니다.

- RoleArn – [Custom string pattern #26](#)과(와) 일치하는 1~1,024바이트 길이의 UTF-8 문자열입니다.

역할 ARN입니다. 이 역할은 AWS Glue 서비스에서 수입하며 워크플로 및 워크플로의 기타 엔터티를 생성하는 데 사용됩니다.

연영

- [CreateWorkflow 작업\(Python: create_workflow\)](#)
- [UpdateWorkflow 작업\(Python: update_workflow\)](#)

- [DeleteWorkflow](#) 작업(Python: `delete_workflow`)
- [GetWorkflow](#) 작업(Python: `get_workflow`)
- [ListWorkflows](#) 작업(Python: `list_workflows`)
- [BatchGetWorkflows](#) 작업(Python: `batch_get_workflows`)
- [GetWorkflowRun](#) 작업(Python: `get_workflow_run`)
- [GetWorkflowRuns](#) 작업(Python: `get_workflow_runs`)
- [GetWorkflowRunProperties](#) 작업(Python: `get_workflow_run_properties`)
- [PutWorkflowRunProperties](#) 작업(Python: `put_workflow_run_properties`)
- [CreateBlueprint](#) 작업(Python: `create_blueprint`)
- [UpdateBlueprint](#) 작업(Python: `update_blueprint`)
- [DeleteBlueprint](#) 작업(Python: `delete_blueprint`)
- [ListBlueprints](#) 작업(Python: `list_blueprints`)
- [BatchGetBlueprints](#) 작업(Python: `batch_get_blueprints`)
- [StartBlueprintRun](#) 작업(Python: `start_blueprint_run`)
- [GetBlueprintRun](#) 작업(Python: `get_blueprint_run`)
- [GetBlueprintRuns](#) 작업(Python: `get_blueprint_runs`)
- [StartWorkflowRun](#) 작업(Python: `start_workflow_run`)
- [StopWorkflowRun](#) 작업(Python: `stop_workflow_run`)
- [ResumeWorkflowRun](#) 작업(Python: `resume_workflow_run`)

CreateWorkflow 작업(Python: `create_workflow`)

새 워크플로를 생성합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

워크플로에 할당할 이름입니다. 이름은 계정 내에서 고유해야 합니다.

- Description – UTF-8 문자열입니다.

워크플로에 대한 설명입니다.

- DefaultRunProperties – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

워크플로의 각 실행의 일부로 사용할 속성 모음입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이 워크플로에 사용할 태그입니다.

- MaxConcurrentRuns - 숫자(정수)입니다.

이 파라미터를 사용하여 데이터에 대한 원치 않는 다중 업데이트를 방지하거나 비용을 제어하거나 경우에 따라 구성 요소 작업의 최대 동시 실행 수를 초과하는 것을 방지할 수 있습니다. 이 파라미터를 공백으로 두면 동시 워크플로 실행 수에 제한이 없습니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

요청의 일부로 제공된 워크플로의 이름입니다.

Errors

- AlreadyExistsException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

UpdateWorkflow 작업(Python: update_workflow)

기존 워크플로를 업데이트합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트할 워크플로의 이름입니다.

- Description – UTF-8 문자열입니다.

워크플로에 대한 설명입니다.

- DefaultRunProperties – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

워크플로의 각 실행의 일부로 사용할 속성 모음입니다.

- MaxConcurrentRuns - 숫자(정수)입니다.

이 파라미터를 사용하여 데이터에 대한 원치 않는 다중 업데이트를 방지하거나 비용을 제어하거나 경우에 따라 구성 요소 작업의 최대 동시 실행 수를 초과하는 것을 방지할 수 있습니다. 이 파라미터를 공백으로 두면 동시 워크플로 실행 수에 제한이 없습니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

입력에서 지정된 워크플로의 이름입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

DeleteWorkflow 작업(Python: delete_workflow)

워크플로를 삭제합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 워크플로의 이름입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

입력에서 지정된 워크플로의 이름입니다.

Errors

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ConcurrentModificationException

GetWorkflow 작업(Python: get_workflow)

워크플로에 대한 리소스 메타데이터를 검색합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

검색할 워크플로의 이름입니다.

- IncludeGraph – 부울입니다.

워크플로 리소스 메타데이터를 반환할 때 그래프를 포함할지 여부를 지정합니다.

응답

- Workflow – [워크플로](#) 객체입니다.

워크플로에 대한 리소스 메타데이터입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

ListWorkflows 작업(Python: list_workflows)

계정에서 생성된 워크플로의 이름을 나열합니다.

요청

- NextToken – UTF-8 문자열입니다.

이것이 지속적인 요청이라면 지속적인 토큰입니다.

- MaxResults - 1 이상 25 이하의 숫자(정수)입니다.

반환할 목록의 최대 크기.

응답

- Workflows – 1~25개 문자열로 구성된 UTF-8 문자열의 배열입니다.

계정에 있는 워크플로의 이름 목록입니다.

- NextToken – UTF-8 문자열입니다.

일부 워크플로 이름이 반환되지 않은 경우의 지속 토큰입니다.

Errors

- InvalidInputException

- `InternalServiceException`
- `OperationTimeoutException`

BatchGetWorkflows 작업(Python: `batch_get_workflows`)

제공된 워크플로 이름 목록에 대한 리소스 메타데이터의 목록을 반환합니다. `ListWorkflows` 작업을 호출한 후에는 권한이 부여된 데이터에 액세스하기 위해 이 작업을 호출할 수 있습니다. 이 작업은 태그를 사용하는 권한 조건을 포함해 모든 IAM 권한을 지원합니다.

요청

- `Names` – 필수(Required): 1~25개 문자열의 UTF-8 문자열의 배열입니다.

`ListWorkflows` 작업에서 반환된 이름일 수 있는 워크플로 이름의 목록입니다.

- `IncludeGraph` – 부울입니다.

워크플로 리소스 메타데이터를 반환할 때 그래프를 포함할지 여부를 지정합니다.

응답

- `Workflows` – [워크플로](#) 객체의 배열이며 구조는 1~25개입니다.

워크플로 리소스 메타데이터의 목록입니다.

- `MissingWorkflows` – 1~25개 문자열로 구성된 UTF-8 문자열의 배열입니다.

찾을 수 없는 워크플로의 이름 목록입니다.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

GetWorkflowRun 작업(Python: `get_workflow_run`)

제공된 워크플로 실행에 대한 메타데이터를 검색합니다. 워크플로 및 작업 실행의 경우 작업 실행 기록을 90일 동안 액세스할 수 있습니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행되고 있는 워크플로의 이름입니다.

- RunId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

워크플로 실행의 ID입니다.

- IncludeGraph – 부울입니다.

워크플로 그래프를 응답에 포함할지 여부를 지정합니다.

응답

- Run – [WorkflowRun](#) 객체입니다.

요청된 워크플로 실행 메타데이터입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetWorkflowRuns 작업(Python: get_workflow_runs)

제공된 워크플로의 모든 실행에 대한 메타데이터를 검색합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행 메타데이터를 반환해야 하는 워크플로의 이름입니다.

- IncludeGraph – 부울입니다.

워크플로 그래프를 응답에 포함할지 여부를 지정합니다.

- NextToken – UTF-8 문자열입니다.

응답의 최대 크기입니다.

- MaxResults – 1~1,000의 숫자(정수)입니다.

응답에 포함할 최대 워크플로 실행 수입니다.

응답

- Runs – [WorkflowRun](#) 객체의 배열이며 구조는 1~1,000개입니다.

워크플로 실행 메타데이터 객체의 목록입니다.

- NextToken – UTF-8 문자열입니다.

일부 요청된 워크플로 실행이 반환되지 않은 경우 지속 토큰입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetWorkflowRunProperties 작업(Python: `get_workflow_run_properties`)

실행 중에 설정된 워크플로 실행 속성을 검색합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행된 워크플로의 이름입니다.

- RunId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행 속성을 반환해야 하는 워크플로 실행의 ID입니다.

응답

- `RunProperties` – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

지정된 실행 중에 설정된 워크플로 실행 속성입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`

PutWorkflowRunProperties 작업(Python: `put_workflow_run_properties`)

제공된 워크플로 실행에 대해 지정된 워크플로 실행 속성을 입력합니다. 지정된 실행에 대한 속성이 이미 있으면 이 속성이 값을 재정의하고 그렇지 않으면 속성을 기존 속성에 추가합니다.

요청

- `Name` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행된 워크플로의 이름입니다.

- `RunId` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행 속성을 업데이트해야 하는 워크플로 실행의 ID입니다.

- `RunProperties` – 필수(Required): 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

지정된 실행에 대해 입력할 속성입니다.

응답

- 무응답 파라미터.

Errors

- AlreadyExistsException
- EntityNotFoundException
- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- ConcurrentModificationException

CreateBlueprint 작업(Python: create_blueprint)

AWS Glue에 블루프린트를 등록합니다.

요청

- Name – 필수(Required): [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

블루프린트의 이름입니다.

- Description – 1~512바이트 길이의 UTF-8 문자열입니다.

블루프린트에 대한 설명입니다.

- BlueprintLocation – 필수(Required): [Custom string pattern #28](#)과(와) 일치하는 1~8,192바이트 길이의 UTF-8 문자열입니다.

블루프린트가 게시되는 Amazon S3의 경로를 지정합니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이 블루프린트에 적용할 태그입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
등록된 블루프린트의 이름을 반환합니다.

Errors

- AlreadyExistsException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

UpdateBlueprint 작업(Python: update_blueprint)

등록된 블루프린트를 업데이트합니다.

요청

- Name – 필수(Required): [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.
블루프린트의 이름입니다.
- Description – 1~512바이트 길이의 UTF-8 문자열입니다.
블루프린트에 대한 설명입니다.
- BlueprintLocation – 필수(Required): [Custom string pattern #28](#)과(와) 일치하는 1~8,192바이트 길이의 UTF-8 문자열입니다.
블루프린트가 게시되는 Amazon S3의 경로를 지정합니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
업데이트된 블루프린트의 이름을 반환합니다.

Errors

- EntityNotFoundException
- ConcurrentModificationException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- IllegalBlueprintStateException

DeleteBlueprint 작업(Python: delete_blueprint)

기존 블루프린트를 삭제합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
삭제할 블루프린트의 이름입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
삭제된 블루프린트의 이름을 반환합니다.

Errors

- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListBlueprints 작업(Python: list_blueprints)

계정의 모든 블루프린트 이름을 나열합니다.

요청

- NextToken – UTF-8 문자열입니다.

이것이 지속적인 요청이라면 지속적인 토큰입니다.

- MaxResults - 1 이상 25 이하의 숫자(정수)입니다.

반환할 목록의 최대 크기.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

AWS 리소스 태그로 목록을 필터링합니다.

응답

- Blueprints – UTF-8 문자열의 배열입니다.

계정에 있는 블루프린트의 이름 목록입니다.

- NextToken – UTF-8 문자열입니다.

모든 블루프린트 이름이 반환하지 않은 경우의 지속 토큰입니다.

Errors

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

BatchGetBlueprints 작업(Python: batch_get_blueprints)

블루프린트 목록에 대한 정보를 검색합니다.

요청

- **Names** – 필수(Required): 1~25개 문자열의 UTF-8 문자열의 배열입니다.
블루프린트 이름의 목록입니다.
- **IncludeBlueprint** – 부울입니다.
응답에 워크플로를 포함할지 여부를 지정합니다.
- **IncludeParameterSpec** – 부울입니다.
응답에 블루프린트에 대한 파라미터를 JSON 문자열로 포함할지 여부를 지정합니다.

응답

- **Blueprints** – [블루프린트](#) 객체의 배열입니다.
블루프린트 목록을 Blueprints 객체로 반환합니다.
- **MissingBlueprints** – UTF-8 문자열의 배열입니다.
찾을 수 없는 BlueprintNames 목록을 반환합니다.

Errors

- `InternalServiceException`
- `OperationTimeoutException`
- `InvalidInputException`

StartBlueprintRun 작업(Python: `start_blueprint_run`)

지정된 블루프린트의 새 실행을 시작합니다.

요청

- **BlueprintName** – 필수(Required): [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.
블루프린트의 이름입니다.
- **Parameters** – 1~131,072바이트 길이의 UTF-8 문자열입니다.

파라미터를 `BlueprintParameters` 객체로 지정합니다.

- `RoleArn` – 필수(Required): [Custom string pattern #26](#)과(와) 일치하는 1~1,024바이트 길이의 UTF-8 문자열입니다.

워크플로 생성에 사용되는 IAM 역할을 지정합니다.

응답

- `RunId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 블루프린트 실행의 실행 ID입니다.

Errors

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ResourceNumberLimitExceededException`
- `EntityNotFoundException`
- `IllegalBlueprintStateException`

GetBlueprintRun 작업(Python: `get_blueprint_run`)

블루프린트 실행의 세부 정보를 검색합니다.

요청

- `BlueprintName` – 필수(Required): [Custom string pattern #27](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

블루프린트의 이름입니다.

- `RunId` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

검색하려는 블루프린트 실행의 실행 ID입니다.

응답

- BlueprintRun – [BlueprintRun](#) 객체입니다.

BlueprintRun 객체를 반환합니다.

Errors

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

GetBlueprintRuns 작업(Python: get_blueprint_runs)

지정된 블루프린트에 대한 블루프린트 실행의 세부 정보를 검색합니다.

요청

- BlueprintName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

블루프린트의 이름입니다.

- NextToken – UTF-8 문자열입니다.

이것이 지속적인 요청이라면 지속적인 토큰입니다.

- MaxResults – 1~1,000의 숫자(정수)입니다.

반환할 목록의 최대 크기.

응답

- BlueprintRuns – [BlueprintRun](#) 객체의 배열입니다.

BlueprintRun 객체 목록을 반환합니다.

- NextToken – UTF-8 문자열입니다.

모든 블루프린트 실행이 반환하지 않은 경우의 지속 토큰입니다.

Errors

- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- InvalidInputException

StartWorkflowRun 작업(Python: start_workflow_run)

지정된 워크플로의 새 실행을 시작합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

시작할 워크플로의 이름입니다.

- RunProperties – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 UTF-8 문자열입니다.

새 워크플로 실행에 대한 워크플로 실행 속성입니다.

응답

- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

새 실행의 ID입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException

- ResourceNumberLimitExceededException
- ConcurrentRunsExceededException

StopWorkflowRun 작업(Python: stop_workflow_run)

지정된 워크플로 실행의 실행을 중지합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

중지할 워크플로의 이름입니다.

- RunId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

중지할 워크플로 실행의 ID입니다.

응답

- 무응답 파라미터.

Errors

- InvalidInputException
- EntityNotFoundException
- InternalServiceException
- OperationTimeoutException
- IllegalWorkflowStateException

ResumeWorkflowRun 작업(Python: resume_workflow_run)

이전에 부분적으로 완료된 워크플로 실행의 선택한 노드를 다시 시작하고 워크플로 실행을 재개합니다. 선택한 노드와 선택한 노드의 다운스트림에 있는 모든 노드가 실행됩니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

재개할 워크플로의 이름입니다.

- RunId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

재개할 워크플로 실행의 ID입니다.

- NodeIds – 필수(Required): UTF-8 문자열의 배열입니다.

다시 시작하려는 노드의 노드 ID 목록입니다. 다시 시작될 노드에는 원래 실행에서 실행 시도가 있어야 합니다.

응답

- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

재개된 워크플로 실행에 할당된 새 ID입니다. 워크플로 실행의 각 재개에는 새 실행 ID가 있습니다.

- NodeIds – UTF-8 문자열의 배열입니다.

실제로 다시 시작된 노드의 노드 ID 목록입니다.

Errors

- `InvalidInputException`
- `EntityNotFoundException`
- `InternalServiceException`
- `OperationTimeoutException`
- `ConcurrentRunsExceededException`
- `IllegalWorkflowStateException`

사용 프로필

사용 프로필 API는 에서 사용 프로필을 생성, 업데이트 또는 보는 것과 관련된 데이터 유형 및 API를 설명합니다 AWS Glue.

데이터 타입

- [ProfileConfiguration 구조](#)
- [ConfigurationObject 구조](#)
- [UsageProfileDefinition 구조](#)

ProfileConfiguration 구조

관리자가 AWS Glue 사용 프로필에서 구성하는 작업 및 세션 값을 지정합니다.

필드

- `SessionConfiguration` – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 [ConfigurationObject](#) 객체입니다.

세션 구성 매개변수의 키-값 맵. AWS Glue

- `JobConfiguration` – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 [ConfigurationObject](#) 객체입니다.

작업 구성 매개변수의 키-값 맵. AWS Glue

ConfigurationObject 구조

관리자가 AWS Glue 사용 프로필에 구성된 각 작업 또는 세션 매개 변수에 설정하는 값을 지정합니다.

필드

- **DefaultValue** – [Custom string pattern #31](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

파라미터의 기본값입니다.

- **AllowedValues** – UTF-8 문자열의 배열입니다.

매개변수에 허용되는 값 목록.

- **MinValue** – [Custom string pattern #31](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

매개변수에 허용되는 최소값.

- **MaxValue** – [Custom string pattern #31](#)과(와) 일치하는 1~128바이트 길이의 UTF-8 문자열입니다.

매개변수에 허용되는 최대값입니다.

UsageProfileDefinition 구조

AWS Glue 사용 프로필을 설명합니다.

필드

- **Name** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

사용 프로필의 이름.

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

사용 프로필에 대한 설명.

- **CreatedOn** – 타임스탬프입니다.

사용 프로필이 생성된 날짜 및 시간.

- **LastModifiedOn** – 타임스탬프입니다.

사용 프로필이 마지막으로 수정된 날짜 및 시간.

운영

- [CreateUsageProfile](#) 액션 (Python: [사용_프로필 생성하기](#))
- [GetUsageProfile](#) 액션 (Python: [get_usage_profile](#))
- [UpdateUsageProfile](#) 액션 (Python: [업데이트_사용_프로필](#))
- [DeleteUsageProfile](#) 액션 (Python: [사용_프로필 삭제](#))
- [ListUsageProfiles](#) 액션 (Python: [목록_사용_프로필](#))

CreateUsageProfile 액션 (Python: [사용_프로필 생성하기](#))

사용 프로필을 생성합니다 AWS Glue .

요청

- Name – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

사용 프로필의 이름.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

사용 프로필에 대한 설명.

- Configuration – 필수: [ProfileConfiguration](#) 객체입니다.

프로필의 작업 및 세션 값을 지정하는 ProfileConfiguration 개체입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

사용 프로필에 적용된 태그 목록.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

생성된 사용자 프로필의 이름.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `AlreadyExistsException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`
- `OperationNotSupportedException`

GetUsageProfile 액션 (Python: `get_usage_profile`)

지정된 사용 프로필에 대한 정보를 검색합니다. AWS Glue

요청

- `Name` – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
검색할 사용자 프로필의 이름.

응답

- `Name` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
사용 프로필의 이름.
- `Description` – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.
사용 프로필에 대한 설명.
- `Configuration` – [ProfileConfiguration](#) 객체입니다.
프로필의 작업 및 세션 값을 지정하는 `ProfileConfiguration` 개체입니다.
- `CreatedOn` – 타임스탬프입니다.
사용 프로필이 생성된 날짜 및 시간.
- `LastModifiedOn` – 타임스탬프입니다.
사용 프로필이 마지막으로 수정된 날짜 및 시간.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`
- `OperationTimeoutException`
- `OperationNotSupportedException`

UpdateUsageProfile 액션 (Python: 업데이트_사용_프로필)

사용 프로필 업데이트 AWS Glue .

요청

- **Name** – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

사용 프로필의 이름.

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

사용 프로필에 대한 설명.

- **Configuration** – 필수: [ProfileConfiguration](#) 객체입니다.

프로필의 작업 및 세션 값을 지정하는 ProfileConfiguration 개체입니다.

응답

- **Name** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트된 사용 프로필의 이름.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `EntityNotFoundException`

- `OperationTimeoutException`
- `OperationNotSupportedException`
- `ConcurrentModificationException`

DeleteUsageProfile 액션 (Python: `사용_프로필 삭제`)

지정된 사용 프로필을 삭제합니다. AWS Glue

요청

- `Name` – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
삭제할 사용자 프로필의 이름.

응답

- 무응답 파라미터.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `OperationNotSupportedException`

ListUsageProfiles 액션 (Python: `목록_사용_프로필`)

모든 사용 프로필을 나열하세요. AWS Glue

요청

- `NextToken` – 400,000바이트 이하 길이의 UTF-8 문자열입니다.
이것이 지속적으로 호출되면 지속적인 토큰을 포함합니다.
- `MaxResults` - 1 이상 200 이하의 숫자(정수)입니다.
단일 응답으로 반환할 수 있는 최대 사용 프로필 수입니다.

응답

- Profiles – [UsageProfileDefinition](#) 객체의 배열입니다.
사용 프로필 (UsageProfileDefinition) 개체 목록.
- NextToken – 400,000바이트 이하 길이의 UTF-8 문자열입니다.
현재 목록 부분이 유지가 되지 않으면 연속 토큰이 존재합니다.

Errors

- InternalServiceException
- OperationTimeoutException
- InvalidInputException
- OperationNotSupportedException

기계 학습 API

기계 학습 API는 기계 학습 데이터 형식을 설명하며 변환을 생성, 삭제 또는 업데이트하거나 기계 학습 작업 실행을 시작하기 위한 API를 포함합니다.

데이터 타입

- [TransformParameters 구조](#)
- [EvaluationMetrics 구조](#)
- [MLTransform 구조](#)
- [FindMatchesParameters 구조](#)
- [FindMatchesMetrics 구조](#)
- [ConfusionMatrix 구조](#)
- [GlueTable 구조](#)
- [TaskRun 구조](#)
- [TransformFilterCriteria 구조](#)
- [TransformSortCriteria 구조](#)
- [TaskRunFilterCriteria 구조](#)
- [TaskRunSortCriteria 구조](#)

- [TaskRunProperties 구조](#)
- [FindMatchesTaskRunProperties 구조](#)
- [ImportLabelsTaskRunProperties 구조](#)
- [ExportLabelsTaskRunProperties 구조](#)
- [LabelingSetGenerationTaskRunProperties 구조](#)
- [SchemaColumn 구조](#)
- [TransformEncryption 구조](#)
- [MLUserDataEncryption 구조](#)
- [ColumnImportance 구조](#)

TransformParameters 구조

기계 학습 변환과 연결된 알고리즘별 파라미터입니다.

필드

- `TransformType` – 필수: UTF-8 문자열입니다(유효한 값: `FIND_MATCHES`).

기계 학습 변환의 유형입니다.

기계 학습 변환 유형에 대한 자세한 내용은 [기계 학습 변환 생성](#)을 참조하십시오.

- `FindMatchesParameters` – [FindMatchesParameters](#) 객체입니다.

일치 항목 찾기 알고리즘에 대한 파라미터입니다.

EvaluationMetrics 구조

평가 지표는 기계 학습 변환의 예상 품질을 제공합니다.

필드

- `TransformType` – 필수: UTF-8 문자열입니다(유효한 값: `FIND_MATCHES`).

기계 학습 변환의 유형입니다.

- `FindMatchesMetrics` – [FindMatchesMetrics](#) 객체입니다.

일치 항목 찾기 알고리즘에 대한 평가 지표입니다.

MLTransform 구조

기계 학습 변환의 구조입니다.

필드

- **TransformId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환에 대해 생성된 고유 변환 ID입니다. ID는 고유한 것으로 보장되며 변경되지 않습니다.

- **Name** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 사용자 정의 이름입니다. 이름은 고유한 것으로 보장되지 않으며 언제든지 변경할 수 있습니다.

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

기계 학습 변환에 대한 사용자 정의된 긴 형식의 설명 텍스트입니다. 설명은 고유한 것으로 보장되지 않으며 언제든지 변경할 수 있습니다.

- **Status** – UTF-8 문자열입니다(유효한 값: NOT_READY | READY | DELETING).

기계 학습 변환의 현재 상태입니다.

- **CreatedOn** – 타임스탬프입니다.

타임스탬프입니다. 이 기계 학습 변환이 생성된 시간과 날짜입니다.

- **LastModifiedOn** – 타임스탬프입니다.

타임스탬프입니다. 이 기계 학습 변환이 수정된 마지막 시점입니다.

- **InputRecordTables** – [GlueTable](#) 객체의 배열이며 구조는 10개 이하입니다.

변환에 사용된 AWS Glue 테이블 정의의 목록입니다.

- **Parameters** – [TransformParameters](#) 객체입니다.

[TransformParameters](#) 객체입니다. 파라미터를 통해 기계 학습 변환이 학습하는 데이터 및 다양한 트레이드오프(예: 재현율 대비 귀중함 또는 비용 대비 정확도)에 대한 기본 설정을 지정하여 기계 학습 변환의 동작을 튜닝(사용자 지정)할 수 있습니다.

- **EvaluationMetrics** – [EvaluationMetrics](#) 객체입니다.

EvaluationMetrics 객체입니다. 평가 지표는 기계 학습 변환의 예상 품질을 제공합니다.

- LabelCount - 숫자(정수)입니다.

이 변환에 대해 AWS Glue에서 생성된 레이블 지정 파일에 대한 카운트 식별자입니다. 더 좋은 변환을 만들면 레이블 지정 파일을 반복적으로 다운로드하고 레이블 지정하고 업로드할 수 있습니다.

- Schema – [SchemaColumn](#) 객체의 배열이며 구조는 100개 이하입니다.

이 변환이 실행할 수 있는 열과 데이터 형식을 나타내는 키-값 페어의 맵입니다. 100열의 상한이 있습니다.

- Role – UTF-8 문자열입니다.

필수 권한이 있는 IAM 역할의 이름 또는 Amazon 리소스 이름(ARN)입니다. 필요한 권한에는 AWS Glue 리소스에 대한 AWS Glue 서비스 역할 권한과 변환에서 요구하는 Amazon S3 권한 모두가 포함됩니다.

- 이 역할에는 AWS Glue의 리소스에 대한 액세스 허용을 위한 AWS Glue 서비스 역할 권한이 필요합니다. [AWS Glue에 액세스하는 IAM 사용자에게 정책 연결](#)을 참조하세요.
- 이 역할에는 작업 실행에서 이 변환에 사용되는 Amazon Simple Storage Service(Amazon S3) 소스, 대상, 임시 디렉터리, 스크립트 및 모든 라이브러리에 대한 권한이 필요합니다.
- GlueVersion – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 값은 이러한 기계 학습 변환과 호환되는 AWS Glue의 버전을 결정합니다. Glue 1.0은 대부분의 고객에게 권장됩니다. 값이 설정되지 않은 경우 Glue 호환성은 Glue 0.9로 기본 설정됩니다. 자세한 내용은 개발자 안내서의 [AWS Glue 버전](#)을 참조하세요.

- MaxCapacity - 숫자(double)입니다.

이 변환의 태스크 실행에 할당된 AWS Glue 데이터 처리 장치(DPU)의 수입니다. 2~100DPU를 할당할 수 있으며, 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

MaxCapacity는 NumberOfWorkers 및 WorkerType과 함께 사용할 수 없는 옵션입니다.

- NumberOfWorkers나 WorkerType 중 하나가 설정되면 MaxCapacity를 설정할 수 없습니다.
- MaxCapacity가 설정되면 NumberOfWorkers와 WorkerType 모두 설정할 수 없습니다.
- WorkerType이 설정되면 NumberOfWorkers가 필요합니다(반대의 경우도 마찬가지).
- MaxCapacity와 NumberOfWorkers는 1 이상이어야 합니다.

WorkerType 필드를 Standard 이외의 다른 값으로 설정하면 MaxCapacity 필드가 자동으로 설정되고 읽기 전용이 됩니다.

- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

이 변환의 작업이 실행될 때 할당되는 미리 정의된 작업자의 유형입니다. Standard, G.1X 또는 G.2X 값을 허용합니다.

- Standard 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 50GB 디스크와, 작업자당 실행기 2개를 제공합니다.
- G.1X 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 64GB 디스크와, 작업자당 실행기 1개를 제공합니다.
- G.2X 작업자 유형의 경우, 각 작업자가 8vCPU, 32GB 메모리 및 128GB 디스크와, 작업자당 실행기 1개를 제공합니다.

MaxCapacity는 NumberOfWorkers 및 WorkerType과 함께 사용할 수 없는 옵션입니다.

- NumberOfWorkers나 WorkerType 중 하나가 설정되면 MaxCapacity를 설정할 수 없습니다.
- MaxCapacity가 설정되면 NumberOfWorkers와 WorkerType 모두 설정할 수 없습니다.
- WorkerType이 설정되면 NumberOfWorkers가 필요합니다(반대의 경우도 마찬가지).
- MaxCapacity와 NumberOfWorkers는 1 이상이어야 합니다.
- NumberOfWorkers - 숫자(정수)입니다.

변환의 작업이 실행될 때 할당되는 정의된 workerType의 작업자 수입니다.

WorkerType이 설정되면 NumberOfWorkers가 필요합니다(반대의 경우도 마찬가지).

- Timeout – 1 이상의 숫자(정수)입니다.

기계 학습 변환의 시간 초과(분)입니다.

- MaxRetries - 숫자(정수)입니다.

기계 학습 변환의 MLTaskRun이 실패한 후 최대 재시도 횟수입니다.

- TransformEncryption – [TransformEncryption](#) 객체입니다.

사용자 데이터 액세스에 적용되는 변환의 유희 시 암호화 설정입니다. 기계 학습 변환에서는 KMS를 사용하여 Amazon S3의 암호화된 사용자 데이터에 액세스할 수 있습니다.

FindMatchesParameters 구조

일치 항목 찾기 변환을 구성하기 위한 파라미터입니다.

필드

- `PrimaryKeyColumnName` – [Single-line string pattern](#)과(와) 일치하는 1~1,024바이트 길이의 UTF-8 문자열입니다.

소스 테이블에서 행을 고유하게 식별하는 열의 이름입니다. 일치하는 레코드를 식별하기 위해 사용됩니다.

- `PrecisionRecallTradeoff` – 1.0 이하의 숫자(실수)입니다.

정밀도와 재현율 간의 균형을 위해 변환을 튜닝할 때 선택하는 값입니다. 값 0.5는 기본 설정 없음, 값 1.0은 순전히 정밀도에 대한 바이어스, 값 0.0은 재현율에 대한 바이어스를 의미합니다. 이 값은 트레이드오프이기 때문에 1.0에 가까운 값을 선택하면 매우 낮은 재현율을 의미하고 0.0에 가까운 값을 선택하면 매우 낮은 정밀도를 나타냅니다.

정밀도 지표는 모델이 일치를 정확하게 예측하는 빈도를 나타냅니다.

재현율 지표는 실제 일치에 대해 모델이 일치를 예측하는 빈도를 나타냅니다.

- `AccuracyCostTradeoff` – 1.0 이하의 숫자(실수)입니다.

정확도와 비용 간의 균형을 위해 변환을 튜닝할 때 선택하는 값입니다. 값 0.5는 시스템이 정확도 및 비용 문제의 균형을 유지하고 있음을 의미합니다. 값 1.0은 순전히 정확도에 대한 바이어스를 의미하며 일반적으로 더 높은 비용, 때로는 상당히 더 높은 비용을 나타냅니다. 값 0.0은 순전히 비용에 대한 바이어스를 의미하며 비교적 정확하지 않은 FindMatches 변환, 때로는 용인할 수 없는 수준의 정확도를 나타냅니다.

정확도는 변환이 참 긍정과 참 부정을 얼마나 잘 찾는지 측정합니다. 정확도를 증가시키려면 더 많은 기계 리소스와 비용이 필요합니다. 하지만 이렇게 하면 재현율도 증가합니다.

비용은 변환을 실행하는 데 얼마나 많은 컴퓨팅 리소스(따라서 비용)가 사용되는지를 측정합니다.

- `EnforceProvidedLabels` – 부울입니다.

사용자가 제공한 레이블과 일치하는 강제 출력을 켜거나 끄는 값입니다. 값이 True이면 find matches 변환은 제공된 레이블과 일치하도록 출력을 강제합니다. 결과는 일반 융합 결과를 재정의합니다. 값이 False이면 find matches 변환은 제공된 모든 레이블이 존중될 것을 보장하지 않으며 결과는 교육된 모델에 따라 다릅니다.

이 값을 true로 설정하면 융합 실행 시간이 증가합니다.

FindMatchesMetrics 구조

일치 항목 찾기 알고리즘에 대한 평가 지표입니다. 기계 학습 변환의 품질은 변환을 가져와서 몇 가지 일치를 예측하고 동일한 데이터세트의 알려진 일치 항목과 결과를 비교하여 측정됩니다. 품질 지표는 데이터의 하위 세트를 기반으로 하므로 정밀하지 않습니다.

필드

- `AreaUnderPRCurve` – 1.0 이하의 숫자(실수)입니다.

정밀도/재현율 곡선(AUPRC) 아래 면적은 변환의 전체 품질을 측정하는 단일 숫자이며, 이 숫자는 재현율 대비 정밀도에 대해 수행한 선택과 무관합니다. 값이 높을수록 더 매력적인 재현율 대비 정밀도 트레이드오프가 있음을 나타냅니다.

자세한 내용은 Wikipedia의 [정밀도 및 재현율](#)을 참조하십시오.

- `Precision` – 1.0 이하의 숫자(실수)입니다.

정밀도 지표는 변환이 일치를 정확하게 예측하는 빈도를 나타냅니다. 특히 정밀도는 변환이 총 참 긍정에서 참 긍정을 얼마나 잘 찾는지를 측정합니다.

자세한 내용은 Wikipedia의 [정밀도 및 재현율](#)을 참조하십시오.

- `Recall` – 1.0 이하의 숫자(실수)입니다.

재현율 지표는 실제 일치에 대해 변환이 일치를 예측하는 빈도를 나타냅니다. 특히 재현율은 변환이 소스 데이터의 총 레코드에서 참 긍정을 얼마나 잘 찾는지를 측정합니다.

자세한 내용은 Wikipedia의 [정밀도 및 재현율](#)을 참조하십시오.

- `F1` – 1.0 이하의 숫자(실수)입니다.

최대 F1 지표는 0~1 범위에서 변환의 정확도를 나타냅니다. 여기서 1은 최상의 정확도입니다.

자세한 내용은 Wikipedia의 [F1 점수](#)를 참조하십시오.

- `ConfusionMatrix` – [ConfusionMatrix](#) 객체입니다.

혼동 행렬은 변환이 무엇을 정확하게 예측하고 있으며 어떤 유형의 오류가 발생하고 있는지를 보여줍니다.

자세한 내용은 Wikipedia의 [혼동 행렬](#)을 참조하십시오.

- ColumnImportances – [ColumnImportance](#) 객체의 배열이며 구조는 100개 이하입니다.

중요도 내림차순으로 정렬된 열 중요도 지표를 포함하는 ColumnImportance 구조 목록입니다.

ConfusionMatrix 구조

혼동 행렬은 변환이 무엇을 정확하게 예측하고 있으며 어떤 유형의 오류가 발생하고 있는지를 보여 줍니다.

자세한 내용은 Wikipedia의 [혼동 행렬](#)을 참조하십시오.

필드

- NumTruePositives - 숫자(정수)입니다.

변환에 대한 혼동 행렬에서 변환이 올바르게 찾은 데이터의 일치 항목 수입니다.

- NumFalsePositives - 숫자(정수)입니다.

변환에 대한 혼동 행렬에서 변환이 일치 항목으로 잘못 분류한 데이터의 불일치 항목 수입니다.

- NumTrueNegatives - 숫자(정수)입니다.

변환에 대한 혼동 행렬에서 변환이 올바르게 거부한 데이터의 일치 항목 수입니다.

- NumFalseNegatives - 숫자(정수)입니다.

변환에 대한 혼동 행렬에서 변환이 찾지 못한 데이터의 일치 항목 수입니다.

GlueTable 구조

입력 또는 출력 데이터에 사용되는 AWS Glue Data Catalog의 데이터베이스와 테이블입니다.

필드

- DatabaseName – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue Data Catalog의 데이터베이스 이름입니다.

- **TableName** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue Data Catalog의 테이블 이름입니다.

- **CatalogId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue Data Catalog의 고유 식별자입니다.

- **ConnectionName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue Data Catalog에 대한 연결 이름입니다.

- **AdditionalOptions** – 1~10개 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 [URI address multi-line string pattern](#)와 일치하는 설명 문자열(2,048바이트 이하)입니다.

테이블에 대한 추가 옵션입니다. 현재 지원되는 키는 두 가지입니다.

- **pushDownPredicate**: 데이터 세트 내 모든 파일을 나열하거나 읽지 않아도 파티션에 필터링합니다.
- **catalogPartitionPredicate**: AWS Glue Data Catalog에서 파티션 인덱스를 사용하여 서버 측 파티션 프루닝을 사용합니다.

TaskRun 구조

기계 학습 변환과 연결된 샘플링 파라미터입니다.

필드

- **TransformId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

변환의 고유 식별자입니다.

- **TaskRunId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 작업 실행의 고유 식별자입니다.

- **Status** – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

요청된 작업 실행의 현재 상태입니다.

- `LogGroupName` – UTF-8 문자열입니다.

이 작업 실행과 연결된 보안 로깅을 위한 로그 그룹의 이름입니다.

- `Properties` – [TaskRunProperties](#) 객체입니다.

이 작업 실행과 연결된 구성 속성을 지정합니다.

- `ErrorString` – UTF-8 문자열입니다.

이 작업 실행과 연결된 오류 문자열의 목록입니다.

- `StartedOn` – 타임스탬프입니다.

이 작업 실행이 시작된 날짜와 시간입니다.

- `LastModifiedOn` – 타임스탬프입니다.

요청된 작업 실행이 업데이트된 마지막 시점입니다.

- `CompletedOn` – 타임스탬프입니다.

요청된 작업 실행이 완료된 마지막 시점입니다.

- `ExecutionTime` - 숫자(정수)입니다.

이 작업 실행이 리소스를 사용한 시간(초).입니다.

TransformFilterCriteria 구조

기계 학습 변환을 필터링하는 데 사용되는 기준입니다.

필드

- `Name` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환을 필터링하는 데 사용되는 고유의 변환 이름입니다.

- `TransformType` – UTF-8 문자열입니다(유효한 값: `FIND_MATCHES`).

기계 학습 변환을 필터링하는 데 사용되는 기계 학습 변환의 유형입니다.

- `Status` – UTF-8 문자열입니다(유효한 값: `NOT_READY` | `READY` | `DELETING`).

마지막으로 알려진 변환 상태를 기준으로 기계 학습 변환 목록을 필터링합니다(변환을 사용할 수 있는지 여부를 나타냄). "NOT_READY", "READY" 또는 "DELETING" 중 하나입니다.

- `GlueVersion` – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 값은 이러한 기계 학습 변환과 호환되는 AWS Glue의 버전을 결정합니다. Glue 1.0은 대부분의 고객에게 권장됩니다. 값이 설정되지 않은 경우 Glue 호환성은 Glue 0.9로 기본 설정됩니다. 자세한 내용은 개발자 안내서의 [AWS Glue 버전](#)을 참조하세요.

- `CreatedBefore` – 타임스탬프입니다.

그 이전에 변환이 생성된 시간과 날짜입니다.

- `CreatedAfter` – 타임스탬프입니다.

그 이후에 변환이 생성된 시간과 날짜입니다.

- `LastModifiedBefore` – 타임스탬프입니다.

이 날짜 이전에 마지막으로 수정된 변환에서 필터링합니다.

- `LastModifiedAfter` – 타임스탬프입니다.

이 날짜 이후에 마지막으로 수정된 변환에서 필터링합니다.

- `Schema` – [SchemaColumn](#) 객체의 배열이며 구조는 100개 이하입니다.

특정 스키마를 사용하여 데이터세트에서 필터링합니다. `Map<Column, Type>` 객체는 이 변환이 허용하는 스키마를 나타내는 키-값 페어의 배열입니다. 여기서 `Column`은 열의 이름이고 `Type`은 정수 또는 문자열과 같은 데이터 형식입니다. 100열의 상한이 있습니다.

TransformSortCriteria 구조

기계 학습 변환과 연결된 정렬 기준입니다.

필드

- `Column` – 필수: UTF-8 문자열입니다(유효한 값: NAME | TRANSFORM_TYPE | STATUS | CREATED | LAST_MODIFIED).

기계 학습 변환과 연결된 정렬 기준에 사용할 열입니다.

- `SortDirection` – 필수: UTF-8 문자열입니다(유효한 값: DESCENDING | ASCENDING).

기계 학습 변환과 연결된 정렬 기준에 사용할 정렬 방향입니다.

TaskRunFilterCriteria 구조

기계 학습 변환에 대한 작업 실행을 필터링하는 데 사용되는 기준입니다.

필드

- **TaskRunType** – UTF-8 문자열입니다(유효한 값: EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES).

작업 실행의 유형입니다.

- **Status** – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

작업 실행의 현재 상태입니다.

- **StartedBefore** – 타임스탬프입니다.

이 날짜 이전에 시작된 작업 실행에서 필터링합니다.

- **StartedAfter** – 타임스탬프입니다.

이 날짜 이후에 시작된 작업 실행에서 필터링합니다.

TaskRunSortCriteria 구조

기계 학습 변환에 대한 작업 실행 목록을 정렬하는 데 사용되는 정렬 기준입니다.

필드

- **Column** – 필수: UTF-8 문자열입니다(유효한 값: TASK_RUN_TYPE | STATUS | STARTED).

기계 학습 변환에 대한 작업 실행 목록을 정렬하는 데 사용할 열입니다.

- **SortDirection** – 필수: UTF-8 문자열입니다(유효한 값: DESCENDING | ASCENDING).

기계 학습 변환에 대한 작업 실행 목록을 정렬하는 데 사용할 정렬 방향입니다.

TaskRunProperties 구조

작업 실행에 대한 구성 속성입니다.

필드

- **TaskType** – UTF-8 문자열입니다(유효한 값: EVALUATION | LABELING_SET_GENERATION | IMPORT_LABELS | EXPORT_LABELS | FIND_MATCHES).

작업 실행의 유형입니다.

- **ImportLabelsTaskRunProperties** – [ImportLabelsTaskRunProperties](#) 객체입니다.

가져오기 레이블 작업 실행에 대한 구성 속성입니다.

- **ExportLabelsTaskRunProperties** – [ExportLabelsTaskRunProperties](#) 객체입니다.

내보내기 레이블 작업 실행에 대한 구성 속성입니다.

- **LabelingSetGenerationTaskRunProperties** – [LabelingSetGenerationTaskRunProperties](#) 객체입니다.

레이블 지정 세트 생성 작업 실행에 대한 구성 속성입니다.

- **FindMatchesTaskRunProperties** – [FindMatchesTaskRunProperties](#) 객체입니다.

일치 항목 찾기 작업 실행에 대한 구성 속성입니다.

FindMatchesTaskRunProperties 구조

일치 항목 찾기 작업 실행에 대한 구성 속성을 지정합니다.

필드

- **JobId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

일치 항목 찾기 작업 실행의 작업 ID입니다.

- **JobName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

일치 항목 찾기 작업 실행에 대한 작업에 할당된 이름입니다

- **JobRunId** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

일치 항목 찾기 작업 실행의 작업 실행 ID입니다.

ImportLabelsTaskRunProperties 구조

가져오기 레이블 작업 실행에 대한 구성 속성을 지정합니다.

필드

- InputS3Path – UTF-8 문자열입니다.

레이블을 가져올 Amazon Simple Storage Service(Amazon S3) 경로입니다.

- Replace – 부울입니다.

기존 레이블을 덮어쓸지 여부를 표시합니다.

ExportLabelsTaskRunProperties 구조

내보내기 레이블 작업 실행에 대한 구성 속성을 지정합니다.

필드

- OutputS3Path – UTF-8 문자열입니다.

레이블을 내보낼 Amazon Simple Storage Service(Amazon S3) 경로입니다.

LabelingSetGenerationTaskRunProperties 구조

레이블 지정 세트 생성 작업 실행에 대한 구성 속성을 지정합니다.

필드

- OutputS3Path – UTF-8 문자열입니다.

레이블 지정 세트를 생성할 Amazon Simple Storage Service(Amazon S3) 경로입니다.

SchemaColumn 구조

이 변환이 실행할 수 있는 열과 데이터 형식을 나타내는 키-값 페어입니다. MLTransform의 Schema 파라미터에는 이러한 구조가 최대 100개까지 포함될 수 있습니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~1,024바이트 길이의 UTF-8 문자열입니다.

열의 이름입니다.

- DataType – [Single-line string pattern](#)과(와) 일치하는 131,072바이트 이하 길이의 UTF-8 문자열입니다.

열에 있는 데이터의 형식입니다.

TransformEncryption 구조

사용자 데이터 액세스에 적용되는 변환의 유효 시 암호화 설정입니다. 기계 학습 변환에서는 KMS를 사용하여 Amazon S3의 암호화된 사용자 데이터에 액세스할 수 있습니다.

또한 가져온 레이블 및 교육된 변환은 이제 고객이 제공한 KMS 키를 사용하여 암호화할 수 있습니다.

필드

- MLUserDataEncryption – [MLUserDataEncryption](#) 객체입니다.

MLUserDataEncryption 객체는 암호화 모드 및 고객 제공 KMS 키 ID를 포함합니다.

- TaskRunSecurityConfigurationName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

보안 구성의 이름입니다.

MLUserDataEncryption 구조

사용자 데이터 액세스에 적용되는 변환의 유효 시 암호화 설정입니다.

필드

- MLUserDataEncryptionMode – 필수: UTF-8 문자열입니다(유효한 값: DISABLED | SSE-KMS="SSEKMS").

사용자 데이터에 적용되는 암호화 모드입니다. 유효한 값은 다음과 같습니다.

- DISABLED: 암호화가 비활성화됨

- SSEKMS: Amazon S3에 저장된 사용자 데이터에 대해 AWS Key Management Service(SSE-KMS) 와 서버 측 암호화를 사용합니다.
- KmsKeyId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
고객이 제공한 KMS 키의 ID입니다.

ColumnImportance 구조

열에 대한 열 이름 및 열 중요도 점수를 포함하는 구조입니다.

열 중요도는 레코드에서 다른 열보다 더 중요한 열을 식별하여 열이 모델에 어떻게 기여하는지 이해하는 데 도움이 됩니다.

필드

- ColumnName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
열의 이름입니다.
- Importance – 1.0 이하의 숫자(실수)입니다.
열에 대한 열 중요도 점수(10진수)입니다.

운영

- [CreateMLTransform 작업\(Python: create_ml_transform\)](#)
- [UpdateMLTransform 작업\(Python: update_ml_transform\)](#)
- [DeleteMLTransform 작업\(Python: delete_ml_transform\)](#)
- [GetMLTransform 작업\(Python: get_ml_transform\)](#)
- [GetMLTransforms 작업\(Python: get_ml_transforms\)](#)
- [ListMLTransforms 작업\(Python: list_ml_transforms\)](#)
- [StartMLEvaluationTaskRun 작업\(Python: start_ml_evaluation_task_run\)](#)
- [StartMLLabelingSetGenerationTaskRun 작업\(Python: start_ml_labeling_set_generation_task_run\)](#)
- [GetMLTaskRun 작업\(Python: get_ml_task_run\)](#)
- [GetMLTaskRuns 작업\(Python: get_ml_task_runs\)](#)
- [CancelMLTaskRun 작업\(Python: cancel_ml_task_run\)](#)

- [StartExportLabelsTaskRun](#) 작업(Python: `start_export_labels_task_run`)
- [StartImportLabelsTaskRun](#) 작업(Python: `start_import_labels_task_run`)

CreateMLTransform 작업(Python: `create_ml_transform`)

AWS Glue 기계 학습 변환을 생성합니다. 이 작업은 변환과 변환을 교육하는 데 필요한 모든 파라미터를 생성합니다.

데이터 중복 제거에 기계 학습 변환(예: FindMatches 변환)을 사용하는 프로세스의 첫 번째 단계로 이 작업을 호출합니다. 알고리즘에 사용할 파라미터 외에도 선택 사항인 Description을 제공할 수 있습니다.

데이터에서 학습하고 고품질 기계 학습 변환을 생성하는 과정의 일부로 AWS Glue가 자동으로 실행하는 태스크에 대한 특정 파라미터도 지정해야 합니다. 이러한 파라미터에는 Role과 선택 사항으로 AllocatedCapacity, Timeout 및 MaxRetries가 포함됩니다. 자세한 내용은 [작업](#)을 참조하십시오.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

변환을 생성할 때 변환에 부여한 고유 이름입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

정의하고 있는 기계 학습 변환에 대한 설명입니다. 기본값은 빈 문자열입니다.

- InputRecordTables – 필수(Required): [GlueTable](#) 객체의 배열이며 구조는 10개 이하입니다.

변환에 사용된 AWS Glue 테이블 정의의 목록입니다.

- Parameters – 필수(Required): [TransformParameters](#) 객체입니다.

사용된 변환 유형에 특정한 알고리즘 파라미터입니다. 조건부로 변환 유형에 따라 다릅니다.

- Role – 필수(Required): UTF-8 문자열입니다.

필수 권한이 있는 IAM 역할의 이름 또는 Amazon 리소스 이름(ARN)입니다. 필요한 권한에는 AWS Glue 리소스에 대한 AWS Glue 서비스 역할 권한과 변환에서 요구하는 Amazon S3 권한 모두가 포함됩니다.

- 이 역할에는 AWS Glue의 리소스에 대한 액세스 허용을 위한 AWS Glue 서비스 역할 권한이 필요합니다. [AWS Glue에 액세스하는 IAM 사용자에게 정책 연결](#)을 참조하세요.
- 이 역할에는 작업 실행에서 이 변환에 사용되는 Amazon Simple Storage Service(Amazon S3) 소스, 대상, 임시 디렉터리, 스크립트 및 모든 라이브러리에 대한 권한이 필요합니다.
- GlueVersion - [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 값은 이러한 기계 학습 변환과 호환되는 AWS Glue의 버전을 결정합니다. Glue 1.0은 대부분의 고객에게 권장됩니다. 값이 설정되지 않은 경우 Glue 호환성은 Glue 0.9로 기본 설정됩니다. 자세한 내용은 개발자 안내서의 [AWS Glue 버전](#)을 참조하세요.

- MaxCapacity - 숫자(double)입니다.

이 변환의 태스크 실행에 할당된 AWS Glue 데이터 처리 장치(DPU)의 수입니다. 2~100DPU를 할당할 수 있으며, 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

MaxCapacity는 NumberOfWorkers 및 WorkerType과 함께 사용할 수 없는 옵션입니다.

- NumberOfWorkers나 WorkerType 중 하나가 설정되면 MaxCapacity를 설정할 수 없습니다.
- MaxCapacity가 설정되면 NumberOfWorkers와 WorkerType 모두 설정할 수 없습니다.
- WorkerType이 설정되면 NumberOfWorkers가 필요합니다(반대의 경우도 마찬가지).
- MaxCapacity와 NumberOfWorkers는 1 이상이어야 합니다.

WorkerType 필드를 Standard 이외의 다른 값으로 설정하면 MaxCapacity 필드가 자동으로 설정되고 읽기 전용이 됩니다.

WorkerType 필드를 Standard 이외의 다른 값으로 설정하면 MaxCapacity 필드가 자동으로 설정되고 읽기 전용이 됩니다.

- WorkerType - UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

이 작업이 실행될 때 할당되는 미리 정의된 작업자의 유형입니다. Standard, G.1X 또는 G.2X 값을 허용합니다.

- Standard 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 50GB 디스크와, 작업자당 실행기 2개를 제공합니다.
- G.1X 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 64GB 디스크와, 작업자당 실행기 1개를 제공합니다.

- G.2X 작업자 유형의 경우, 각 작업자가 8vCPU, 32GB 메모리 및 128GB 디스크와, 작업자당 실행기 1개를 제공합니다.

MaxCapacity는 NumberOfWorkers 및 WorkerType과 함께 사용할 수 없는 옵션입니다.

- NumberOfWorkers나 WorkerType 중 하나가 설정되면 MaxCapacity를 설정할 수 없습니다.
- MaxCapacity가 설정되면 NumberOfWorkers와 WorkerType 모두 설정할 수 없습니다.
- WorkerType이 설정되면 NumberOfWorkers가 필요합니다(반대의 경우도 마찬가지).
- MaxCapacity와 NumberOfWorkers는 1 이상이어야 합니다.
- NumberOfWorkers - 숫자(정수)입니다.

이 작업이 실행될 때 할당되는 정의된 workerType의 작업자 수입니다.

WorkerType이 설정되면 NumberOfWorkers가 필요합니다(반대의 경우도 마찬가지).

- Timeout - 1 이상의 숫자(정수)입니다.

이 변환에 대한 작업 실행의 제한 시간(분)입니다. 이 값은 이 변환에 대한 작업 실행이 종료되고 TIMEOUT 상태로 전환되기 전에 리소스를 사용할 수 있는 최대 시간입니다. 기본값은 2,880 분(48 시간)입니다.

- MaxRetries - 숫자(정수)입니다.

작업 실행이 실패한 후 이 변환에 대한 작업을 재시도할 최대 횟수입니다.

- Tags - 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이러한 기계 학습 변환에서 사용할 태그입니다. 태그를 사용하여 기계 학습 변환에 대한 액세스를 제한할 수 있습니다. AWS Glue의 태그에 대한 자세한 내용은 개발자 안내서의 [AWS Glue의 AWS 태그](#)를 참조하세요.

- TransformEncryption - [TransformEncryption](#) 객체입니다.

사용자 데이터 액세스에 적용되는 변환의 유휴 시 암호화 설정입니다. 기계 학습 변환에서는 KMS를 사용하여 Amazon S3의 암호화된 사용자 데이터에 액세스할 수 있습니다.

응답

- TransformId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

변환에 대해 생성되는 고유 식별자입니다.

Errors

- AlreadyExistsException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- AccessDeniedException
- ResourceNumberLimitExceededException
- IdempotentParameterMismatchException

UpdateMLTransform 작업(Python: update_ml_transform)

기존 기계 학습 변환을 업데이트합니다. 더 좋은 결과를 달성하도록 알고리즘 파라미터를 튜닝하려면 이 작업을 호출합니다.

이 작업을 호출한 후, StartMLEvaluationTaskRun 작업을 호출하여 새로운 파라미터가 목표를 얼마나 잘 달성했는지에 액세스할 수 있습니다(예: 기계 학습 변환의 품질 개선 또는 비용 효과 향상).

요청

- TransformId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

변환을 생성할 때 생성된 고유 식별자입니다.

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

생성할 때 변환에 부여한 고유 이름입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

변환에 대한 설명입니다. 기본값은 빈 문자열입니다.

- Parameters – [TransformParameters](#) 객체입니다.

사용된 변환 유형(알고리즘)에 특정한 구성 파라미터입니다. 조건부로 변환 유형에 따라 다릅니다.

- Role – UTF-8 문자열입니다.

필수 권한이 있는 IAM 역할의 이름 또는 Amazon 리소스 이름(ARN)입니다.

- GlueVersion – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 값은 이러한 기계 학습 변환과 호환되는 AWS Glue의 버전을 결정합니다. Glue 1.0은 대부분의 고객에게 권장됩니다. 값이 설정되지 않은 경우 Glue 호환성은 Glue 0.9로 기본 설정됩니다. 자세한 내용은 개발자 안내서의 [AWS Glue 버전](#)을 참조하세요.

- MaxCapacity - 숫자(double)입니다.

이 변환의 태스크 실행에 할당된 AWS Glue 데이터 처리 장치(DPU)의 수입니다. 2~100DPU를 할당할 수 있으며, 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

WorkerType 필드를 Standard 이외의 다른 값으로 설정하면 MaxCapacity 필드가 자동으로 설정되고 읽기 전용이 됩니다.

- WorkerType – UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

이 작업이 실행될 때 할당되는 미리 정의된 작업자의 유형입니다. Standard, G.1X 또는 G.2X 값을 허용합니다.

- Standard 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 50GB 디스크와, 작업자당 실행기 2개를 제공합니다.
- G.1X 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 64GB 디스크와, 작업자당 실행기 1개를 제공합니다.
- G.2X 작업자 유형의 경우, 각 작업자가 8vCPU, 32GB 메모리 및 128GB 디스크와, 작업자당 실행기 1개를 제공합니다.
- NumberOfWorkers - 숫자(정수)입니다.

이 작업이 실행될 때 할당되는 정의된 workerType의 작업자 수입니다.

- Timeout – 1 이상의 숫자(정수)입니다.

이 변환에 대한 작업 실행의 제한 시간(분)입니다. 이 값은 이 변환에 대한 작업 실행이 종료되고 TIMEOUT 상태로 전환되기 전에 리소스를 사용할 수 있는 최대 시간입니다. 기본값은 2,880 분(48 시간)입니다.

- MaxRetries - 숫자(정수)입니다.

작업 실행이 실패한 후 이 변환에 대한 작업을 재시도할 최대 횟수입니다.

응답

- TransformId - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

업데이트된 변환의 고유 식별자입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- AccessDeniedException

DeleteMLTransform 작업(Python: delete_ml_transform)

AWS Glue 기계 학습 변환을 삭제합니다. 기계 학습 변환은 기계 학습을 통해 사람이 제공한 사례에서 학습하여 수행할 변환에 대한 세부 정보를 학습하는 특수한 유형의 변환입니다. 그런 다음 이러한 변환은 AWS Glue에 의해 저장됩니다. 변환이 더 이상 필요하지 않으면 DeleteMLTransforms를 호출하여 변환을 삭제할 수 있습니다. 하지만 삭제된 변환을 여전히 참조하는 AWS Glue 작업은 더 이상 성공하지 못합니다.

요청

- TransformId - 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 변환의 고유 식별자입니다.

응답

- TransformId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제된 변환의 고유 식별자입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetMLTransform 작업(Python: get_ml_transform)

AWS Glue 기계 학습 변환 아티팩트와 모든 해당 메타데이터를 가져옵니다. 기계 학습 변환은 기계 학습을 통해 사람이 제공한 사례에서 학습하여 수행할 변환에 대한 세부 정보를 학습하는 특수한 유형의 변환입니다. 그런 다음 이러한 변환은 AWS Glue에 의해 저장됩니다. GetMLTransform을 호출하여 해당 메타데이터를 검색할 수 있습니다.

요청

- TransformId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

변환을 생성할 때 생성된 변환의 고유 식별자입니다.

응답

- TransformId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

변환을 생성할 때 생성된 변환의 고유 식별자입니다.

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

변환을 생성할 때 변환에 지정된 고유 이름입니다.

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

변환에 대한 설명입니다.

- **Status** – UTF-8 문자열입니다(유효한 값: NOT_READY | READY | DELETING).

변환의 마지막 알려진 상태입니다(변환을 사용할 수 있는지 여부를 나타냄). "NOT_READY", "READY" 또는 "DELETING" 중 하나입니다.

- **CreatedOn** – 타임스탬프입니다.

변환이 생성된 날짜와 시간입니다.

- **LastModifiedOn** – 타임스탬프입니다.

변환이 수정된 날짜와 시간입니다.

- **InputRecordTables** – [GlueTable](#) 객체의 배열이며 구조는 10개 이하입니다.

변환에 사용된 AWS Glue 테이블 정의의 목록입니다.

- **Parameters** – [TransformParameters](#) 객체입니다.

사용된 알고리즘에 특정한 구성 파라미터입니다.

- **EvaluationMetrics** – [EvaluationMetrics](#) 객체입니다.

최신 평가 지표입니다.

- **LabelCount** - 숫자(정수)입니다.

이 변환에 사용 가능한 레이블 수입니다.

- **Schema** – [SchemaColumn](#) 객체의 배열이며 구조는 100개 이하입니다.

이 변환이 허용하는 스키마를 나타내는 Map<Column, Type> 객체입니다. 100열의 상한이 있습니다.

- **Role** – UTF-8 문자열입니다.

필수 권한이 있는 IAM 역할의 이름 또는 Amazon 리소스 이름(ARN)입니다.

- **GlueVersion** – [Custom string pattern #20](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 값은 이러한 기계 학습 변환과 호환되는 AWS Glue의 버전을 결정합니다. Glue 1.0은 대부분의 고객에게 권장됩니다. 값이 설정되지 않은 경우 Glue 호환성은 Glue 0.9로 기본 설정됩니다. 자세한 내용은 개발자 안내서의 [AWS Glue 버전](#)을 참조하세요.

- MaxCapacity - 숫자(double)입니다.

이 변환의 태스크 실행에 할당된 AWS Glue 데이터 처리 장치(DPU)의 수입니다. 2~100DPU를 할당할 수 있으며, 기본값은 10입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다. 자세한 내용은 [AWS Glue 요금](#) 페이지를 참조하십시오.

WorkerType 필드를 Standard 이외의 다른 값으로 설정하면 MaxCapacity 필드가 자동으로 설정되고 읽기 전용이 됩니다.

- WorkerType - UTF-8 문자열입니다(유효한 값: Standard="" | G.1X="" | G.2X="" | G.025X="" | G.4X="" | G.8X="" | Z.2X="").

이 작업이 실행될 때 할당되는 미리 정의된 작업자의 유형입니다. Standard, G.1X 또는 G.2X 값을 허용합니다.

- Standard 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 50GB 디스크와, 작업자당 실행기 2개를 제공합니다.
- G.1X 작업자 유형의 경우, 각 작업자가 4vCPU, 16GB 메모리 및 64GB 디스크와, 작업자당 실행기 1개를 제공합니다.
- G.2X 작업자 유형의 경우, 각 작업자가 8vCPU, 32GB 메모리 및 128GB 디스크와, 작업자당 실행기 1개를 제공합니다.
- NumberOfWorkers - 숫자(정수)입니다.

이 작업이 실행될 때 할당되는 정의된 workerType의 작업자 수입니다.

- Timeout - 1 이상의 숫자(정수)입니다.

이 변환에 대한 작업 실행의 제한 시간(분)입니다. 이 값은 이 변환에 대한 작업 실행이 종료되고 TIMEOUT 상태로 전환되기 전에 리소스를 사용할 수 있는 최대 시간입니다. 기본값은 2,880 분(48 시간)입니다.

- MaxRetries - 숫자(정수)입니다.

작업 실행이 실패한 후 이 변환에 대한 작업을 재시도할 최대 횟수입니다.

- TransformEncryption - [TransformEncryption](#) 객체입니다.

사용자 데이터 액세스에 적용되는 변환의 유휴 시 암호화 설정입니다. 기계 학습 변환에서는 KMS를 사용하여 Amazon S3의 암호화된 사용자 데이터에 액세스할 수 있습니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetMLTransforms 작업(Python: `get_ml_transforms`)

기존 AWS Glue 기계 학습 변환의 정렬 가능하고 필터링 가능한 목록을 가져옵니다. 기계 학습 변환은 기계 학습을 통해 사람이 제공한 사례에서 학습하여 수행할 변환에 대한 세부 정보를 학습하는 특수한 유형의 변환입니다. 그런 다음 이러한 변환은 AWS Glue에 의해 저장되며, `GetMLTransforms`를 호출하여 해당 메타데이터를 검색할 수 있습니다.

요청

- `NextToken` – UTF-8 문자열입니다.
결과를 오프셋하기 위한 페이지 매김 토큰입니다.
- `MaxResults` – 1~1,000의 숫자(정수)입니다.
반환할 최대 결과 수입니다.
- `Filter` – [TransformFilterCriteria](#) 객체입니다.
변환 필터링 기준입니다.
- `Sort` – [TransformSortCriteria](#) 객체입니다.
정렬 기준입니다.

응답

- `Transforms` – 필수(Required): [MLTransform](#) 객체의 배열입니다.
기계 학습 변환의 목록입니다.

- NextToken – UTF-8 문자열입니다.

추가 결과를 사용할 수 있는 경우 페이지 매김 토큰입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListMLTransforms 작업(Python: list_ml_transforms)

이 AWS 계정에 있는 기존의 AWS Glue 기계 학습 변환이나 지정된 태그가 있는 리소스에 대한 정렬 및 필터링 가능한 목록을 검색합니다. 이 작업을 수행하면 응답의 필터로 사용할 수 있는 Tags 필드 옵션이 검색되기 때문에 태그가 지정된 리소스를 하나의 그룹으로 검색할 수 있습니다. 태그 필터링을 사용하기로 선택하면 태그가 포함된 리소스만 검색됩니다.

요청

- NextToken – UTF-8 문자열입니다.

이것이 지속적인 요청이라면 지속적인 토큰입니다.

- MaxResults – 1~1,000의 숫자(정수)입니다.

반환할 목록의 최대 크기.

- Filter – [TransformFilterCriteria](#) 객체입니다.

기계 학습 변환을 필터링하는 데 사용되는 TransformFilterCriteria입니다.

- Sort – [TransformSortCriteria](#) 객체입니다.

기계 학습 변환을 정렬하는 데 사용되는 TransformSortCriteria입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이렇게 태그가 지정된 리소스만 반환하도록 지정합니다.

응답

- `TransformIds` – 필수(Required): UTF-8 문자열의 배열입니다.

계정의 모든 기계 학습 변환이나 태그가 지정된 기계 학습의 식별자입니다.

- `NextToken` – UTF-8 문자열입니다.

반환된 목록이 사용가능한 마지막 지표를 포함하지 경우의 연속 토큰입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

StartMLEvaluationTaskRun 작업(Python: `start_ml_evaluation_task_run`)

변환 품질을 예상하기 위한 작업을 시작합니다.

레이블 집합을 `truth`의 사례로 제공하면 AWS Glue 기계 학습은 이러한 사례 중 일부를 사용하여 해당 사례에서 학습합니다. 나머지 레이블은 품질을 예상하기 위한 테스트로 사용됩니다.

실행의 고유 식별자를 반환합니다. `GetMLTaskRun`을 호출하여 `EvaluationTaskRun` 통계에 대한 자세한 정보를 가져올 수 있습니다.

요청

- `TransformId` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 고유 식별자입니다.

응답

- TaskRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다. 이 실행과 연결된 고유 식별자입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ConcurrentRunsExceededException
- MLTransformNotReadyException

StartMLLabelingSetGenerationTaskRun 작업(Python: start_ml_labeling_set_generation_task_run)

레이블 세트를 생성하고 레이블을 지정하여 변환의 품질을 개선하기 위해 기계 학습 변환에 대한 활성 학습 워크플로우를 시작합니다.

StartMLLabelingSetGenerationTaskRun이 완료되면 AWS Glue는 "레이블 지정 집합" 또는 사람이 답변할 질문 집합을 생성합니다.

FindMatches 변환의 경우 이러한 질문은 "일치하는 레코드로 완전히 구성된 그룹으로 이러한 행을 함께 그룹화하는 올바른 방법은 무엇입니까?"와 같은 형식입니다.

레이블 지정 프로세스가 완료된 후에는 StartImportLabelsTaskRun을 호출하여 레이블을 업로드할 수 있습니다. StartImportLabelsTaskRun이 완료된 후에는 기계 학습 변환의 모든 향후 실행이 새롭고 개선된 레이블을 사용하며 더 높은 품질의 변환을 수행합니다.

요청

- TransformId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 고유 식별자입니다.

- OutputS3Path – 필수(Required): UTF-8 문자열입니다.

레이블 지정 세트를 생성하는 Amazon Simple Storage Service(Amazon S3) 경로입니다.

응답

- TaskRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 작업 실행과 연결된 고유의 실행 식별자입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ConcurrentRunsExceededException

GetMLTaskRun 작업(Python: get_ml_task_run)

기계 학습 변환의 특정 작업 실행에 대한 세부 사항을 가져옵니다. 기계 학습 태스크 실행은 다양한 기계 학습 워크플로의 일부로 AWS Glue가 자동으로 실행하는 비동기 태스크입니다. TaskRunID 및 해당 상위 변환의 TransformID와 함께 GetMLTaskRun을 호출하여 모든 작업 실행의 통계를 확인할 수 있습니다.

요청

- TransformId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 고유 식별자입니다.

- TaskRunId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업 실행의 고유 식별자입니다.

응답

- TransformId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업 실행의 고유 식별자입니다.

- TaskRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 실행과 연결된 고유의 실행 식별자입니다.

- Status – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

이 작업 실행의 상태입니다.

- LogGroupName – UTF-8 문자열입니다.

작업 실행과 연결된 로그 그룹의 이름입니다.

- Properties – [TaskRunProperties](#) 객체입니다.

작업 실행과 연결된 속성의 목록입니다.

- ErrorString – UTF-8 문자열입니다.

작업 실행과 연결된 오류 문자열입니다.

- StartedOn – 타임스탬프입니다.

이 작업 실행이 시작된 날짜와 시간입니다.

- LastModifiedOn – 타임스탬프입니다.

이 작업 실행이 마지막으로 수정된 날짜와 시간입니다.

- CompletedOn – 타임스탬프입니다.

이 작업 실행이 완료된 날짜와 시간입니다.

- ExecutionTime - 숫자(정수)입니다.

이 작업 실행이 리소스를 사용한 시간(초).입니다.

Errors

- EntityNotFoundException

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetMLTaskRuns 작업(Python: `get_ml_task_runs`)

기계 학습 변환에 대한 실행 목록을 가져옵니다. 기계 학습 태스크 실행은 다양한 기계 학습 워크플로의 일부로 AWS Glue가 자동으로 실행하는 비동기 태스크입니다. 이 단원에 기록된 해당 상위 변환의 `TransformID` 및 기타 선택적 파라미터와 함께 `GetMLTaskRuns`를 호출하여 정렬 가능하고 필터링 가능한 기계 학습 작업 실행 목록을 가져올 수 있습니다.

이 작업은 기록 실행 목록을 반환하며 페이지를 매겨야 합니다.

요청

- `TransformId` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 고유 식별자입니다.

- `NextToken` – UTF-8 문자열입니다.

결과 페이지 매김에 대한 토큰입니다. 기본값은 비어 있음입니다.

- `MaxResults` – 1~1,000의 숫자(정수)입니다.

반환할 최대 결과 수입니다.

- `Filter` – [TaskRunFilterCriteria](#) 객체입니다.

작업 실행에 대한 `TaskRunFilterCriteria` 구조의 필터 기준입니다.

- `Sort` – [TaskRunSortCriteria](#) 객체입니다.

작업 실행에 대한 `TaskRunSortCriteria` 구조의 정렬 기준입니다.

응답

- `TaskRuns` – [TaskRun](#) 객체의 배열입니다.

변환과 연결된 작업 실행의 목록입니다.

- `NextToken` – UTF-8 문자열입니다.

추가 결과를 사용할 수 있는 경우 페이지 매김 토큰입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

CancelMLTaskRun 작업(Python: cancel_ml_task_run)

작업 실행을 취소(중지)합니다. 기계 학습 태스크 실행은 다양한 기계 학습 워크플로의 일부로 AWS Glue가 자동으로 실행하는 비동기 태스크입니다. 작업 실행의 상위 변환의 TransformID 및 작업 실행의 TaskRunId와 함께 CancelMLTaskRun을 호출하여 언제든지 기계 학습 작업 실행을 취소할 수 있습니다.

요청

- TransformId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 고유 식별자입니다.

- TaskRunId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업 실행의 고유 식별자입니다.

응답

- TransformId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 고유 식별자입니다.

- TaskRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업 실행의 고유 식별자입니다.

- Status – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

이 실행의 상태입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartExportLabelsTaskRun 작업(Python: start_export_labels_task_run)

특정 변환에 대해 레이블 지정된 모든 데이터를 내보내기 위한 비동기 작업을 시작합니다. 이 작업은 일반적인 활성 학습 워크플로우의 일부가 아닌 유일한 레이블 관련 API 호출입니다. 이전에 truth로 제출한 레이블을 제거하거나 변경하려는 경우와 같이 모든 기존 레이블을 동시에 작업하려는 경우에 일반적으로 StartExportLabelsTaskRun을 사용합니다. 이 API 작업은 레이블을 내보내려고 하는 TransformId 및 레이블을 내보낼 Amazon Simple Storage Service(Amazon S3) 경로를 허용합니다. 이 작업은 TaskRunId를 반환합니다. GetMLTaskRun API를 호출하여 작업 실행의 상태를 확인할 수 있습니다.

요청

- TransformId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 고유 식별자입니다.

- OutputS3Path – 필수(Required): UTF-8 문자열입니다.

레이블을 내보내는 Amazon S3 경로입니다.

응답

- TaskRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

작업 실행의 고유 식별자입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

StartImportLabelsTaskRun 작업(Python: start_import_labels_task_run)

기계 학습 변환을 교육하고 품질을 개선하는 데 사용할 추가 레이블(truth의 사례)을 제공할 수 있습니다. 일반적으로 이 API 작업은 StartMLLabelingSetGenerationTaskRun 호출로 시작하고 결국 기계 학습 변환의 품질을 개선하는 활성 학습 워크플로우의 일부로 사용됩니다.

StartMLLabelingSetGenerationTaskRun이 완료되면 AWS Glue 기계 학습은 사람이 답변할 일련의 질문을 생성합니다. (기계 학습 워크플로우에서는 이러한 질문에 답변하는 것을 흔히 '레이블 지정'이라고 합니다). FindMatches 변환의 경우 이러한 질문은 "일치하는 레코드로 완전히 구성된 그룹으로 이러한 행을 함께 그룹화하는 올바른 방법은 무엇입니까?"와 같은 형식입니다. 레이블 지정 프로세스가 완료된 후 사용자는 StartImportLabelsTaskRun을 호출하여 답변/레이블을 업로드할 수 있습니다. StartImportLabelsTaskRun이 완료된 후에는 기계 학습 변환의 모든 향후 실행이 새롭고 개선된 레이블을 사용하며 더 높은 품질의 변환을 수행합니다.

기본적으로 Replace를 true로 설정하지 않는 한 StartMLLabelingSetGenerationTaskRun은 업로드하는 모든 레이블에서 지속적으로 학습하고 업로드하는 모든 레이블을 결합합니다. Replace를 true로 설정하면 StartImportLabelsTaskRun은 이전에 업로드한 모든 레이블을 삭제하고 잊어버리며 업로드하는 정확한 세트에서만 학습합니다. 레이블 바꾸기는 잘못된 레이블을 이전에 업로드한 것을 인식하고 해당 레이블이 변환 품질에 부정적인 영향을 미치고 있다고 확신하는 경우에 유용할 수 있습니다.

GetMLTaskRun 작업을 호출하여 작업 실행의 상태를 확인할 수 있습니다.

요청

- TransformId – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

기계 학습 변환의 고유 식별자입니다.

- InputS3Path – 필수(Required): UTF-8 문자열입니다.

레이블을 가져올 Amazon Simple Storage Service(Amazon S3) 경로입니다.

- ReplaceAllLabels – 부울입니다.

기존 레이블을 덮어쓸지 여부를 표시합니다.

응답

- TaskRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
작업 실행의 고유 식별자입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- ResourceNumberLimitExceededException
- InternalServiceException

데이터 품질 API

데이터 품질 API는 데이터 품질 데이터 유형에 대해 설명하며 데이터 품질 규칙 세트, 실행 및 평가를 생성, 삭제 또는 업데이트하기 위한 API를 포함합니다.

데이터 타입

- [DataSource 구조](#)
- [DataQualityRulesetListDetails 구조](#)
- [DataQualityTargetTable 구조](#)
- [DataQualityRulesetEvaluationRunDescription 구조](#)
- [DataQualityRulesetEvaluationRunFilter 구조](#)
- [DataQualityEvaluationRunAdditionalRunOptions 구조](#)
- [DataQualityRuleRecommendationRunDescription 구조](#)
- [DataQualityRuleRecommendationRunFilter 구조](#)

- [DataQualityResult 구조](#)
- [DataQualityAnalyzerResult 구조](#)
- [DataQualityObservation 구조](#)
- [MetricBasedObservation 구조](#)
- [DataQualityMetricValues 구조](#)
- [DataQualityRuleResult 구조](#)
- [DataQualityResultDescription 구조](#)
- [DataQualityResultFilterCriteria 구조](#)
- [DataQualityRulesetFilterCriteria 구조](#)

DataSource 구조

데이터 품질 결과를 원하는 데이터 소스 (AWS Glue 테이블).

필드

- GlueTable – 필수: [GlueTable](#) 객체입니다.

AWS Glue 테이블.

DataQualityRulesetListDetails 구조

GetDataQualityRuleset에서 반환되는 데이터 품질 규칙 세트를 설명합니다.

필드

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 규칙 세트의 이름입니다.

- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

데이터 품질 규칙 세트에 대한 설명입니다.

- CreatedOn – 타임스탬프입니다.

데이터 품질 규칙 세트가 생성된 날짜와 시간입니다.

- LastModifiedOn – 타임스탬프입니다.

데이터 품질 규칙 세트가 마지막으로 수정된 날짜와 시간입니다.

- TargetTable – [DataQualityTargetTable](#) 객체입니다.

AWS Glue 테이블을 나타내는 객체.

- RecommendationRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

권장 실행에서 규칙 세트가 생성되면 이 실행 ID가 생성되어 두 규칙을 서로 연결합니다.

- RuleCount - 숫자(정수)입니다.

규칙 세트의 규칙 수입니다.

DataQualityTargetTable 구조

AWS Glue 테이블을 나타내는 객체.

필드

- TableName – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue 테이블 이름.

- DatabaseName – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue 테이블이 있는 데이터베이스의 이름.

- CatalogId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

AWS Glue 테이블이 있는 카탈로그 ID.

DataQualityRulesetEvaluationRunDescription 구조

데이터 품질 규칙 세트 평가 실행 결과를 설명합니다.

필드

- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 실행과 연결된 고유의 실행 식별자입니다.

- Status – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

이 실행의 상태입니다.

- StartedOn – 타임스탬프입니다.

실행이 시작된 날짜와 시간입니다.

- DataSource – [DataSource](#) 객체입니다.

실행과 관련된 데이터 소스 (AWS Glue 테이블)

DataQualityRulesetEvaluationRunFilter 구조

필터 기준입니다.

필드

- DataSource – 필수: [DataSource](#) 객체입니다.

실행과 관련된 데이터 소스 (AWS Glue 테이블) 를 기반으로 필터링합니다.

- StartedBefore – 타임스탬프입니다.

이 시간 이전에 시작된 실행을 기준으로 결과를 필터링합니다.

- StartedAfter – 타임스탬프입니다.

이 시간 이후에 시작된 실행을 기준으로 결과를 필터링합니다.

DataQualityEvaluationRunAdditionalRunOptions 구조

평가 실행에 대해 지정할 수 있는 추가 실행 옵션입니다.

필드

- CloudWatchMetricsEnabled – 부울입니다.

CloudWatch 메트릭 활성화 여부.

- ResultsS3Prefix – UTF-8 문자열입니다.

결과를 저장할 Amazon S3의 접두사입니다.

- CompositeRuleEvaluationMethod – UTF-8 문자열입니다(유효한 값: COLUMN | ROW).

규칙 세트의 복합 규칙에 대한 평가 방법을 ROW/COLUMN으로 설정합니다.

DataQualityRuleRecommendationRunDescription 구조

데이터 품질 규칙 권장 실행 결과를 설명합니다.

필드

- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 실행과 연결된 고유의 실행 식별자입니다.

- Status – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

이 실행의 상태입니다.

- StartedOn – 타임스탬프입니다.

이 실행이 시작된 날짜와 시간입니다.

- DataSource – [DataSource](#) 객체입니다.

추천 실행과 관련된 데이터 소스 (AWS Glue 테이블).

DataQualityRuleRecommendationRunFilter 구조

데이터 품질 권장 실행을 나열하기 위한 필터입니다.

필드

- DataSource – 필수: [DataSource](#) 객체입니다.

지정된 데이터 소스 (AWS Glue 테이블) 를 기반으로 필터링합니다.

- StartedBefore – 타임스탬프입니다.

제공된 시간 이전에 시작된 결과에 대한 시간을 기준으로 필터링합니다.

- StartedAfter – 타임스탬프입니다.

제공된 시간 이후에 시작된 결과에 대한 시간을 기준으로 필터링합니다.

DataQualityResult 구조

데이터 품질 결과를 설명합니다.

필드

- ResultId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과의 고유한 결과 ID입니다.

- Score – 1.0 이하의 숫자(실수)입니다.

집계된 데이터 품질 점수입니다. 총 규칙 수에 전달된 규칙의 비율을 나타냅니다.

- DataSource – [DataSource](#) 객체입니다.

데이터 품질 결과와 연결된 테이블입니다(있는 경우).

- RulesetName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과와 연결된 규칙 세트의 이름입니다.

- EvaluationContext – UTF-8 문자열입니다.

AWS Glue Studio의 작업 컨텍스트에서 캔버스의 각 노드에는 일반적으로 일종의 이름이 할당되며 데이터 품질 노드에는 이름이 지정됩니다. 여러 노드의 경우 evaluationContext에서 노드를 구분할 수 있습니다.

- StartedOn – 타임스탬프입니다.

이 데이터 품질 실행이 시작된 날짜와 시간입니다.

- CompletedOn – 타임스탬프입니다.

이 데이터 품질 실행이 완료된 날짜와 시간입니다.

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과와 연결된 작업 이름입니다(있는 경우).

- JobRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과와 연결된 작업 실행 ID입니다(있는 경우).

- `RulesetEvaluationRunId` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 데이터 품질 결과에 대한 규칙 세트 평가의 고유한 실행 ID입니다.
- `RuleResults` – [DataQualityRuleResult](#) 객체의 배열이며 구조는 2,000개 이하입니다.
각 규칙의 결과를 나타내는 `DataQualityRuleResult` 객체 목록입니다.
- `AnalyzerResults` – [DataQualityAnalyzerResult](#) 객체의 배열이며 구조는 2,000개 이하입니다.
각 분석기의 결과를 나타내는 `DataQualityAnalyzerResult` 객체의 목록입니다.
- `Observations` – [DataQualityObservation](#) 객체의 배열이며 구조는 50개 이하입니다.
규칙과 분석기를 평가한 후 생성된 관찰을 나타내는 `DataQualityObservation` 객체의 목록입니다.

DataQualityAnalyzerResult 구조

데이터 품질 분석기의 평가 결과를 설명합니다.

필드

- `Name` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
데이터 품질 분석기의 이름입니다.
- `Description` – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 UTF-8 문자열입니다.
데이터 품질 분석기에 대한 설명입니다.
- `EvaluationMessage` – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 UTF-8 문자열입니다.
평가 메시지입니다.
- `EvaluatedMetrics` – 키-값 페어의 맵 배열입니다.
각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.
각 값은 숫자(double)입니다.
분석기 평가와 관련된 지표의 맵입니다.

DataQualityObservation 구조

규칙과 분석기를 평가한 후 생성된 관찰을 설명합니다.

필드

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 UTF-8 문자열입니다.

데이터 품질 관찰에 대한 설명입니다.

- **MetricBasedObservation** – [MetricBasedObservation](#) 객체입니다.

평가된 데이터 품질 지표를 기반으로 하는 관찰을 나타내는 MetricBasedObservation 유형의 객체입니다.

MetricBasedObservation 구조

평가된 데이터 품질 지표를 기반으로 생성된 지표 기반 관찰을 설명합니다.

필드

- **MetricName** – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

관찰을 생성하는 데 사용된 데이터 품질 지표의 이름입니다.

- **MetricValues** – [DataQualityMetricValues](#) 객체입니다.

데이터 품질 지표 값의 분석을 나타내는 유형 DataQualityMetricValues의 객체입니다.

- **NewRules** – UTF-8 문자열의 배열입니다.

데이터 품질 지표 값을 기반으로 관찰의 일부로 생성된 새 데이터 품질 규칙의 목록입니다.

DataQualityMetricValues 구조

과거 데이터 분석에 따른 데이터 품질 지표 값을 설명합니다.

필드

- **ActualValue** - 숫자(double)입니다.

데이터 품질 지표의 실제 값입니다.

- `ExpectedValue` - 숫자(double)입니다.

과거 데이터 분석에 따른 데이터 품질 지표의 예상 값입니다.

- `LowerLimit` - 숫자(double)입니다.

과거 데이터 분석에 따른 데이터 품질 지표 값의 하한입니다.

- `UpperLimit` - 숫자(double)입니다.

과거 데이터 분석에 따른 데이터 품질 지표 값의 상한입니다.

DataQualityRuleResult 구조

데이터 품질 규칙 평가 결과를 설명합니다.

필드

- `Name` - [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 규칙의 이름입니다.

- `Description` - [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 UTF-8 문자열입니다.

데이터 품질 규칙에 대한 설명입니다.

- `EvaluationMessage` - [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 UTF-8 문자열입니다.

평가 메시지입니다.

- `Result` - UTF-8 문자열입니다(유효한 값: PASS | FAIL | ERROR).

규칙의 통과 또는 실패 상태입니다.

- `EvaluatedMetrics` - 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 숫자(double)입니다.

규칙 평가와 관련된 지표 맵입니다.

DataQualityResultDescription 구조

데이터 품질 결과를 설명합니다.

필드

- ResultId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 데이터 품질 결과의 고유한 결과 ID입니다.
- DataSource – [DataSource](#) 객체입니다.
데이터 품질 결과와 연결된 테이블 이름입니다.
- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
데이터 품질 결과와 연결된 작업 이름입니다.
- JobRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
데이터 품질 결과와 연결된 작업 실행 ID입니다.
- StartedOn – 타임스탬프입니다.
이 데이터 품질 결과에 대한 실행이 시작된 시간입니다.

DataQualityResultFilterCriteria 구조

데이터 품질 결과를 반환하는 데 사용되는 기준입니다.

필드

- DataSource – [DataSource](#) 객체입니다.
지정된 데이터 소스별로 결과를 필터링합니다. AWS Glue 테이블에 대한 모든 결과를 검색하는 경우를 예로 들 수 있습니다.
- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
지정된 작업 이름을 기준으로 결과를 필터링합니다.
- JobRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
지정된 작업 실행 ID를 기준으로 결과를 필터링합니다.
- StartedAfter – 타임스탬프입니다.

이 시간 이후에 시작된 실행을 기준으로 결과를 필터링합니다.

- `StartedBefore` – 타임스탬프입니다.

이 시간 이전에 시작된 실행을 기준으로 결과를 필터링합니다.

DataQualityRulesetFilterCriteria 구조

데이터 품질 규칙 세트를 필터링하는 데 사용되는 기준입니다.

필드

- `Name` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

규칙 세트 필터 기준의 이름입니다.

- `Description` – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

규칙 세트 필터 기준에 대한 설명입니다.

- `CreatedBefore` – 타임스탬프입니다.

이 날짜 이전에 생성된 규칙 세트에서 필터링합니다.

- `CreatedAfter` – 타임스탬프입니다.

이 날짜 이후에 생성된 규칙 세트에서 필터링합니다.

- `LastModifiedBefore` – 타임스탬프입니다.

이 날짜 이전에 마지막으로 수정된 규칙 세트에서 필터링합니다.

- `LastModifiedAfter` – 타임스탬프입니다.

이 날짜 이후에 마지막으로 수정된 규칙 세트에서 필터링합니다.

- `TargetTable` – [DataQualityTargetTable](#) 객체입니다.

대상 테이블의 이름 및 데이터베이스 이름입니다.

운영

- [StartDataQualityRulesetEvaluationRun](#) 액션 (Python: `시작_데이터_품질_규칙_세트_평가_실행`)

- [CancelDataQualityRulesetEvaluationRun](#) 액션 (Python: 취소_데이터_품질_규칙_세트_평가_실행)
- [GetDataQualityRulesetEvaluationRun](#) 액션 (Python: get_data_quality_ruleset_valuation_run)
- [ListDataQualityRulesetEvaluationRuns](#) 액션 (Python: 목록_데이터_품질_규칙_세트_평가_실행)
- [StartDataQualityRuleRecommendationRun](#) 액션 (Python: 시작_데이터_품질_규칙_권장_실행)
- [CancelDataQualityRuleRecommendationRun](#) 조치 (Python: 취소_데이터_품질_규칙_권장_실행)
- [GetDataQualityRuleRecommendationRun](#) 액션 (Python: get_data_품질_규칙_권장_실행)
- [ListDataQualityRuleRecommendationRuns](#) 액션 (Python: 목록_데이터_품질_규칙_권장_실행)
- [GetDataQualityResult](#) 액션 (Python: 데이터_품질_결과_가져오기)
- [BatchGetDataQualityResult](#) 액션 (Python: 배치_가져오기_데이터_품질_결과)
- [ListDataQualityResults](#) 액션 (Python: 목록_데이터_품질_결과)
- [CreateDataQualityRuleset](#) 액션 (Python: 생성_데이터_품질_규칙_세트)
- [DeleteDataQualityRuleset](#) 액션 (Python: 데이터_품질_규칙_세트_삭제)
- [GetDataQualityRuleset](#) 액션 (Python: get_data_quality_ruleset)
- [ListDataQualityRulesets](#) 액션 (Python: 목록_데이터_품질_규칙_세트)
- [UpdateDataQualityRuleset](#) 액션 (Python: 업데이트_데이터_품질_규칙_세트)

StartDataQualityRulesetEvaluationRun 액션 (Python: 시작_데이터_품질_규칙_세트_평가_실행)

규칙 세트 정의 (권장 또는 사용자 정의 정의) 가 있으면 이 작업을 호출하여 데이터 소스 (테이블) 를 기준으로 규칙 세트를 평가합니다.AWS Glue 평가 시 GetDataQualityResult API로 검색할 수 있는 결과가 계산됩니다.

요청

- DataSource – 필수: [DataSource](#) 객체입니다.
이 실행과 관련된 데이터 원본 (AWS Glue 테이블).
- Role – 필수: UTF-8 문자열입니다.
실행 결과를 암호화하기 위해 제공되는 IAM 역할입니다.
- NumberOfWorkers - 숫자(정수)입니다.
실행에 사용할 G.1X 작업자 수입니다. 기본값은 5입니다.

- Timeout – 1 이상의 숫자(정수)입니다.

실행 제한 시간(분)입니다. 실행에서 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 기본값은 2,880 분(48 시간)입니다.

- ClientToken – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

멥등성에 사용되며 동일한 리소스의 여러 인스턴스를 생성하거나 시작하지 않으려면 임의의 ID(예: UUID)로 설정하는 것이 좋습니다.

- AdditionalRunOptions – [DataQualityEvaluationRunAdditionalRunOptions](#) 객체입니다.

평가 실행에 대해 지정할 수 있는 추가 실행 옵션입니다.

- RulesetNames – 필수: 1~10개 문자열의 UTF-8 문자열 배열입니다.

규칙 세트 이름의 목록입니다.

- AdditionalDataSources – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 [DataSource](#) 객체입니다.

평가 실행에 대해 지정할 수 있는 추가 데이터 소스에 대한 참조 문자열의 맵입니다.

응답

- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 실행과 연결된 고유의 실행 식별자입니다.

Errors

- InvalidInputException
- EntityNotFoundException
- OperationTimeoutException
- InternalServiceException
- ConflictException

CancelDataQualityRulesetEvaluationRun 액션 (Python: 취소_데이터_품질_규칙 세트_평가_실행)

데이터 소스에 대해 규칙 세트를 평가 중인 실행을 취소합니다.

요청

- RunId – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다. 이 실행과 연결된 고유의 실행 식별자입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetDataQualityRulesetEvaluationRun 액션 (Python: get_data_quality_ruleset_valuation_run)

데이터 소스에 대해 규칙 세트를 평가하는 특정 실행을 검색합니다.

요청

- RunId – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다. 이 실행과 연결된 고유의 실행 식별자입니다.

응답

- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다. 이 실행과 연결된 고유의 실행 식별자입니다.

- DataSource – [DataSource](#) 객체입니다.

이 평가 실행과 관련된 데이터 소스 (테이블) AWS Glue

- Role – UTF-8 문자열입니다.

실행 결과를 암호화하기 위해 제공되는 IAM 역할입니다.

- NumberOfWorkers - 숫자(정수)입니다.

실행에 사용할 G.1X 작업자 수입니다. 기본값은 5입니다.

- Timeout – 1 이상의 숫자(정수)입니다.

실행 제한 시간(분)입니다. 실행에서 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 기본값은 2,880 분(48 시간)입니다.

- AdditionalRunOptions – [DataQualityEvaluationRunAdditionalRunOptions](#) 객체입니다.

평가 실행에 대해 지정할 수 있는 추가 실행 옵션입니다.

- Status – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

이 실행의 상태입니다.

- ErrorString – UTF-8 문자열입니다.

실행과 연결된 오류 문자열입니다.

- StartedOn – 타임스탬프입니다.

이 실행이 시작된 날짜와 시간입니다.

- LastModifiedOn – 타임스탬프입니다.

타임스탬프입니다. 이 데이터 품질 규칙 권장 실행이 수정된 마지막 시점입니다.

- CompletedOn – 타임스탬프입니다.

이 실행이 완료된 날짜와 시간입니다.

- ExecutionTime - 숫자(정수)입니다.

이 실행이 리소스를 사용한 시간(초)입니다.

- RulesetNames – UTF-8 문자열의 배열입니다(1~10개 문자열).

실행에 대한 규칙 세트 이름 목록입니다. 현재 이 파라미터는 하나의 규칙 집합 이름만 사용합니다.

- `ResultIds` – UTF-8 문자열의 배열입니다(1~10개 문자열).

실행의 데이터 품질 결과에 대한 결과 ID 목록입니다.

- `AdditionalDataSources` – 키-값 페어의 맵 배열입니다.

각 키는 [Single-line string pattern](#)과(와) 일치하는 1~255 바이트 길이의 UTF-8 문자열입니다.

각 값은 [DataSource](#) 객체입니다.

평가 실행에 대해 지정할 수 있는 추가 데이터 소스에 대한 참조 문자열의 맵입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityRulesetEvaluationRuns 액션 (Python: 목록_데이터_품질_규칙 세트_평가_실행)

규칙 세트가 데이터 소스에 대해 평가되는 필터 조건을 충족하는 모든 실행을 나열합니다.

요청

- `Filter` – [DataQualityRulesetEvaluationRunFilter](#) 객체입니다.

필터 기준입니다.

- `NextToken` – UTF-8 문자열입니다.

결과를 오프셋하기 위한 페이지 매김 토큰입니다.

- `MaxResults` – 1~1,000의 숫자(정수)입니다.

반환할 최대 결과 수입니다.

응답

- `Runs` – [DataQualityRulesetEvaluationRunDescription](#) 객체의 배열입니다.

데이터 품질 규칙 세트 실행을 나타내는 `DataQualityRulesetEvaluationRunDescription` 객체 목록입니다.

- `NextToken` – UTF-8 문자열입니다.

추가 결과를 사용할 수 있는 경우 페이지 매김 토큰입니다.

Errors

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

StartDataQualityRuleRecommendationRun 액션 (Python: 시작_데이터_품질_규칙_권장_실행)

어떤 규칙을 작성해야 할지 모를 경우 규칙을 생성하는 데 사용되는 권장 사항 실행을 시작합니다. AWS Glue Data Quality는 데이터를 분석하고 잠재적 규칙 세트에 대한 권장 사항을 제시합니다. 그런 다음 규칙 세트를 분류하고 생성된 규칙 세트를 원하는 대로 수정할 수 있습니다.

권장 사항 실행은 90일 후에 자동으로 삭제됩니다.

요청

- `DataSource` – 필수: [DataSource](#) 객체입니다.

이 실행과 관련된 데이터 소스 (AWS Glue 테이블).

- `Role` – 필수: UTF-8 문자열입니다.

실행 결과를 암호화하기 위해 제공되는 IAM 역할입니다.

- `NumberOfWorkers` - 숫자(정수)입니다.

실행에 사용할 G.1X 작업자 수입니다. 기본값은 5입니다.

- `Timeout` – 1 이상의 숫자(정수)입니다.

실행 제한 시간(분)입니다. 실행에서 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 기본값은 2,880 분(48 시간)입니다.

- CreatedRulesetName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

규칙 세트의 이름입니다.

- ClientToken – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

역등성에 사용되며 동일한 리소스의 여러 인스턴스를 생성하거나 시작하지 않으려면 임의의 ID(예: UUID)로 설정하는 것이 좋습니다.

응답

- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 실행과 연결된 고유의 실행 식별자입니다.

Errors

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- ConflictException

CancelDataQualityRuleRecommendationRun 조치 (Python: 취소_데이터_품질_규칙_권장_실행)

규칙을 생성하는 데 사용된 지정된 권장 실행을 취소합니다.

요청

- RunId – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 실행과 연결된 고유의 실행 식별자입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetDataQualityRuleRecommendationRun 액션 (Python: get_data_품질_규칙_권장_실행)

규칙을 생성하는 데 사용된 지정된 권장 실행을 가져옵니다.

요청

- RunId – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 실행과 연결된 고유의 실행 식별자입니다.

응답

- RunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

이 실행과 연결된 고유의 실행 식별자입니다.

- DataSource – [DataSource](#) 객체입니다.

이 실행과 관련된 데이터 소스 (테이블) AWS Glue

- Role – UTF-8 문자열입니다.

실행 결과를 암호화하기 위해 제공되는 IAM 역할입니다.

- NumberOfWorkers - 숫자(정수)입니다.

실행에 사용할 G.1X 작업자 수입니다. 기본값은 5입니다.

- Timeout – 1 이상의 숫자(정수)입니다.

실행 제한 시간(분)입니다. 실행에서 리소스를 소비하여 중지되기 전에 TIMEOUT 상태로 들어가는 최대 시간입니다. 기본값은 2,880 분(48 시간)입니다.

- `Status` – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT).

이 실행의 상태입니다.

- `ErrorMessage` – UTF-8 문자열입니다.

실행과 연결된 오류 문자열입니다.

- `StartedOn` – 타임스탬프입니다.

이 실행이 시작된 날짜와 시간입니다.

- `LastModifiedOn` – 타임스탬프입니다.

타임스탬프입니다. 이 데이터 품질 규칙 권장 실행이 수정된 마지막 시점입니다.

- `CompletedOn` – 타임스탬프입니다.

이 실행이 완료된 날짜와 시간입니다.

- `ExecutionTime` - 숫자(정수)입니다.

이 실행이 리소스를 사용한 시간(초)입니다.

- `RecommendedRuleset` – UTF-8 문자열입니다(1~65536바이트 이하)

시작 규칙 권장 실행이 완료되면 권장 규칙 세트(규칙 세트)가 생성됩니다. 이 멤버는 DQDL(데이터 품질 정의 언어) 형식 규칙을 포함합니다.

- `CreatedRulesetName` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

실행에서 생성된 규칙 세트의 이름입니다.

Errors

- `EntityNotFoundException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityRuleRecommendationRuns 액션 (Python: 목록_데이터_품질_규칙_권장_실행)

필터 기준에 맞는 권장 실행을 나열합니다.

요청

- `Filter` – [DataQualityRuleRecommendationRunFilter](#) 객체입니다.

필터 기준입니다.

- `NextToken` – UTF-8 문자열입니다.

결과를 오프셋하기 위한 페이지 매김 토큰입니다.

- `MaxResults` – 1~1,000의 숫자(정수)입니다.

반환할 최대 결과 수입니다.

응답

- `Runs` – [DataQualityRuleRecommendationRunDescription](#) 객체의 배열입니다.

`DataQualityRuleRecommendationRunDescription` 객체의 목록.

- `NextToken` – UTF-8 문자열입니다.

추가 결과를 사용할 수 있는 경우 페이지 매김 토큰입니다.

Errors

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

GetDataQualityResult 액션 (Python: 데이터_품질_결과 가져오기)

데이터 품질 규칙 평가 결과를 검색합니다.

요청

- ResultId – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과의 고유한 결과 ID입니다.

응답

- ResultId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과의 고유한 결과 ID입니다.

- Score – 1.0 이하의 숫자(실수)입니다.

집계된 데이터 품질 점수입니다. 총 규칙 수에 전달된 규칙의 비율을 나타냅니다.

- DataSource – [DataSource](#) 객체입니다.

데이터 품질 결과와 연결된 테이블입니다(있는 경우).

- RulesetName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과와 연결된 규칙 세트의 이름입니다.

- EvaluationContext – UTF-8 문자열입니다.

AWS Glue Studio의 작업 컨텍스트에서 캔버스의 각 노드에는 일반적으로 일종의 이름이 할당되며 데이터 품질 노드에는 이름이 지정됩니다. 여러 노드의 경우 evaluationContext에서 노드를 구분할 수 있습니다.

- StartedOn – 타임스탬프입니다.

이 데이터 품질 결과에 대한 실행이 시작된 날짜와 시간입니다.

- CompletedOn – 타임스탬프입니다.

이 데이터 품질 결과에 대한 실행이 완료된 날짜와 시간입니다.

- JobName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과와 연결된 작업 이름입니다(있는 경우).

- JobRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 결과와 연결된 작업 실행 ID입니다(있는 경우).

- RulesetEvaluationRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

규칙 세트 평가와 연결된 고유한 실행 ID입니다.

- RuleResults – [DataQualityRuleResult](#) 객체의 배열이며 구조는 2,000개 이하입니다.

각 규칙의 결과를 나타내는 DataQualityRuleResult 객체 목록입니다.

- AnalyzerResults – [DataQualityAnalyzerResult](#) 객체의 배열이며 구조는 2,000개 이하입니다.

각 분석기의 결과를 나타내는 DataQualityAnalyzerResult 객체의 목록입니다.

- Observations – [DataQualityObservation](#) 객체의 배열이며 구조는 50개 이하입니다.

규칙과 분석기를 평가한 후 생성된 관찰을 나타내는 DataQualityObservation 객체의 목록입니다.

Errors

- InvalidInputException
- OperationTimeoutException
- InternalServiceException
- EntityNotFoundException

BatchGetDataQualityResult 액션 (Python: 배치_가져오기_데이터_품질_결과)

지정된 결과 ID에 대한 데이터 품질 결과 목록을 검색합니다.

요청

- ResultIds – 필수: 1~100개 문자열의 UTF-8 문자열 배열입니다.

데이터 품질 결과에 대한 고유한 결과 ID 목록입니다.

응답

- **Results** – 필수: [DataQualityResult](#) 객체의 배열입니다.

데이터 품질 결과를 나타내는 DataQualityResult 객체 목록입니다.

- **ResultsNotFound** – UTF-8 문자열의 배열입니다(1~100개 문자열).

결과를 찾을 수 없는 결과 ID 목록입니다.

Errors

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

ListDataQualityResults 액션 (Python: 목록_데이터_품질_결과)

계정의 모든 데이터 품질 실행 결과를 반환합니다.

요청

- **Filter** – [DataQualityResultFilterCriteria](#) 객체입니다.

필터 기준입니다.

- **NextToken** – UTF-8 문자열입니다.

결과를 오프셋하기 위한 페이지 매김 토큰입니다.

- **MaxResults** – 1~1,000의 숫자(정수)입니다.

반환할 최대 결과 수입니다.

응답

- **Results** – 필수: [DataQualityResultDescription](#) 객체의 배열입니다.

DataQualityResultDescription 객체의 목록.

- **NextToken** – UTF-8 문자열입니다.

추가 결과를 사용할 수 있는 경우 페이지 매김 토큰입니다.

Errors

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

CreateDataQualityRuleset 액션 (Python: 생성_데이터_품질_규칙 세트)

지정된 테이블에 DQDL 규칙을 적용하여 데이터 품질 규칙 세트를 만듭니다. AWS Glue

DQDL(데이터 품질 정의 언어)을 사용하여 규칙 세트를 생성합니다. 자세한 내용은 개발자 안내서를 참조하십시오. AWS Glue

요청

- **Name** – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 규칙 세트의 고유한 이름입니다.

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

데이터 품질 규칙 세트에 대한 설명입니다.

- **Ruleset** – 필수: 2~65536바이트 길이의 UTF-8 문자열입니다.

DQDL(데이터 품질 정의 언어) 규칙 세트입니다. 자세한 내용은 AWS Glue 개발자 안내서를 참조하십시오.

- **Tags** – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

데이터 품질 규칙 세트에 적용된 태그 목록입니다.

- **TargetTable** – [DataQualityTargetTable](#) 객체입니다.

데이터 품질 규칙 세트와 연결된 대상 테이블입니다.

- RecommendationRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

권장 실행의 고유한 실행 ID입니다.

- ClientToken – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

역등성에 사용되며 동일한 리소스의 여러 인스턴스를 생성하거나 시작하지 않으려면 임의의 ID(예: UUID)로 설정하는 것이 좋습니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 규칙 세트의 고유한 이름입니다.

Errors

- InvalidInputException
- AlreadyExistsException
- OperationTimeoutException
- InternalServiceException
- ResourceNumberLimitExceededException

DeleteDataQualityRuleset 액션 (Python: 데이터_품질_규칙 세트 삭제)

데이터 품질 규칙 세트를 삭제합니다.

요청

- Name – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 규칙 세트의 이름입니다.

응답

- 무응답 파라미터.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

GetDataQualityRuleset 액션 (Python: get_data_quality_ruleset)

식별자 또는 이름을 기준으로 기존 규칙 세트를 반환합니다.

요청

- Name – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
규칙 세트의 이름입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
규칙 세트의 이름입니다.
- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.
규칙 세트에 대한 설명입니다.
- Ruleset – UTF-8 문자열입니다(1~65536바이트 이하)
DQDL(데이터 품질 정의 언어) 규칙 세트입니다. 자세한 내용은 개발자 안내서를 참조하십시오.
AWS Glue
- TargetTable – [DataQualityTargetTable](#) 객체입니다.
대상 테이블의 이름 및 데이터베이스 이름입니다.
- CreatedOn – 타임스탬프입니다.
타임스탬프입니다. 이 데이터 품질 규칙 세트가 생성된 날짜와 시간입니다.
- LastModifiedOn – 타임스탬프입니다.

타임스탬프입니다. 이 데이터 품질 규칙 세트가 수정된 마지막 시점입니다.

- RecommendationRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

권장 실행에서 규칙 세트가 생성되면 이 실행 ID가 생성되어 두 규칙을 서로 연결합니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

ListDataQualityRulesets 액션 (Python: 목록_데이터_품질_규칙 세트)

지정된 테이블 목록에 대해 페이지별로 구분된 규칙 세트 목록을 반환합니다. AWS Glue

요청

- NextToken – UTF-8 문자열입니다.

결과를 오프셋하기 위한 페이지 매김 토큰입니다.

- MaxResults – 1~1,000의 숫자(정수)입니다.

반환할 최대 결과 수입니다.

- Filter – [DataQualityRulesetFilterCriteria](#) 객체입니다.

필터 기준입니다.

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

키-값 페어 태그의 목록입니다.

응답

- Rulesets – [DataQualityRulesetListDetails](#) 객체의 배열입니다.
지정된 테이블 목록에 대해 페이지가 매겨진 규칙 세트 목록입니다. AWS Glue
- NextToken – UTF-8 문자열입니다.
추가 결과를 사용할 수 있는 경우 페이지 매김 토큰입니다.

Errors

- EntityNotFoundException
- InvalidInputException
- OperationTimeoutException
- InternalServiceException

UpdateDataQualityRuleset 액션 (Python: 업데이트_데이터_품질_규칙 세트)

지정된 데이터 품질 규칙 세트를 업데이트합니다.

요청

- Name – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
데이터 품질 규칙 세트의 이름입니다.
- Description – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.
규칙 세트에 대한 설명입니다.
- Ruleset – UTF-8 문자열입니다(1~65536바이트 이하)
DQDL(데이터 품질 정의 언어) 규칙 세트입니다. 자세한 내용은 개발자 안내서를 참조하십시오.
AWS Glue

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

데이터 품질 규칙 세트의 이름입니다.

- **Description** – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.

규칙 세트에 대한 설명입니다.

- **Ruleset** – UTF-8 문자열입니다(1~65536바이트 이하)

DQDL(데이터 품질 정의 언어) 규칙 세트입니다. 자세한 내용은 AWS Glue 개발자 안내서를 참조하십시오.

Errors

- `EntityNotFoundException`
- `AlreadyExistsException`
- `IdempotentParameterMismatchException`
- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`
- `ResourceNumberLimitExceededException`

민감한 데이터 감지 API

민감한 데이터 탐지 API는 정형 데이터의 열과 행에서 민감한 데이터를 탐지하는 데 사용되는 API를 설명합니다.

데이터 타입

- [CustomEntityType 구조](#)

CustomEntityType 구조

정형 데이터의 열과 행에서 민감한 데이터를 감지하기 위한 사용자 정의 패턴을 나타내는 객체입니다.

필드

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

나중에 검색하거나 삭제할 수 있는 사용자 정의 패턴의 이름입니다. 이 이름은 AWS 계정별로 고유해야 합니다.

- RegexString – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

사용자 정의 패턴에서 민감한 데이터를 감지하는 데 사용되는 정규식 문자열입니다.

- ContextWords – UTF-8 문자열의 배열입니다(1~20개 문자열).

컨텍스트 단어 목록입니다. 이러한 컨텍스트 단어가 정규식 주변에서 발견되지 않으면 데이터는 민감한 데이터로 감지되지 않습니다.

컨텍스트 단어가 전달되지 않으면 정규식만 검사됩니다.

운영

- [CreateCustomEntityType](#) 작업(Python: `create_custom_entity_type`)
- [DeleteCustomEntityType](#) 작업(Python: `delete_custom_entity_type`)
- [GetCustomEntityType](#) 작업(Python: `get_custom_entity_type`)
- [BatchGetCustomEntityTypes](#) 작업(Python: `batch_get_custom_entity_types`)
- [ListCustomEntityTypes](#) 작업(Python: `list_custom_entity_types`)

CreateCustomEntityType 작업(Python: `create_custom_entity_type`)

정형 데이터의 열과 행에서 민감한 데이터를 감지하는 데 사용되는 사용자 정의 패턴을 생성합니다.

생성하는 각 사용자 정의 패턴은 정규식과 컨텍스트 단어의 선택적 목록을 지정합니다. 컨텍스트 단어가 전달되지 않으면 정규식만 검사됩니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

나중에 검색하거나 삭제할 수 있는 사용자 정의 패턴의 이름입니다. 이 이름은 AWS 계정별로 고유해야 합니다.

- `RegexString` – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

사용자 정의 패턴에서 민감한 데이터를 감지하는 데 사용되는 정규식 문자열입니다.

- `ContextWords` – UTF-8 문자열의 배열입니다(1~20개 문자열).

컨텍스트 단어 목록입니다. 이러한 컨텍스트 단어가 정규식 주변에서 발견되지 않으면 데이터는 민감한 데이터로 감지되지 않습니다.

컨텍스트 단어가 전달되지 않으면 정규식만 검사됩니다.

- `Tags` – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

사용자 지정 엔터티 유형에 적용되는 태그 목록입니다.

응답

- `Name` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

생성한 사용자 정의 패턴의 이름입니다.

Errors

- `AccessDeniedException`
- `AlreadyExistsException`
- `IdempotentParameterMismatchException`
- `InternalServiceException`
- `InvalidInputException`
- `OperationTimeoutException`
- `ResourceNumberLimitExceededException`

DeleteCustomEntityType 작업(Python: delete_custom_entity_type)

이름을 지정하여 사용자 정의 패턴을 삭제합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제할 사용자 정의 패턴의 이름입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

삭제한 사용자 정의 패턴의 이름입니다.

Errors

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException

GetCustomEntityType 작업(Python: get_custom_entity_type)

이름을 지정하여 사용자 패턴의 세부 정보를 검색합니다.

요청

- Name – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

검색할 사용자 정의 패턴의 이름입니다.

응답

- Name – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

검색한 사용자 정의 패턴의 이름입니다.

- RegexpString – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

사용자 정의 패턴에서 민감한 데이터를 감지하는 데 사용되는 정규식 문자열입니다.

- ContextWords – UTF-8 문자열의 배열입니다(1~20개 문자열).

사용자 정의 패턴을 생성할 때 지정된 경우 컨텍스트 단어 목록입니다. 이러한 컨텍스트 단어가 정규식 주변에서 발견되지 않으면 데이터는 민감한 데이터로 감지되지 않습니다.

Errors

- EntityNotFoundException
- AccessDeniedException
- InternalServiceException
- InvalidInputException
- OperationTimeoutException

BatchGetCustomEntityTypes 작업(Python: batch_get_custom_entity_types)

이름 목록으로 지정된 사용자 정의 패턴에 대한 세부 정보를 검색합니다.

요청

- Names – 필수(Required): 1~50개 문자열의 UTF-8 문자열의 배열입니다.

검색할 사용자 정의 패턴의 이름 목록입니다.

응답

- CustomEntityTypes – [CustomEntityType](#) 객체의 배열입니다.

생성된 사용자 정의 패턴을 나타내는 CustomEntityType 객체의 목록입니다.

- CustomEntityTypesNotFound – UTF-8 문자열의 배열입니다(1~50개 문자열).
찾을 수 없는 사용자 정의 패턴의 이름 목록입니다.

Errors

- InvalidInputException
- InternalServiceException
- OperationTimeoutException

ListCustomEntityTypes 작업(Python: list_custom_entity_types)

생성된 모든 사용자 정의 패턴을 나열합니다.

요청

- NextToken – UTF-8 문자열입니다.
결과를 오프셋하기 위한 페이지 매김 토큰입니다.
- MaxResults – 1~1,000의 숫자(정수)입니다.
반환할 최대 결과 수입니다.
- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.
각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.
각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.
키-값 페어 태그의 목록입니다.

응답

- CustomEntityTypes – [CustomEntityType](#) 객체의 배열입니다.
사용자 정의 패턴을 나타내는 CustomEntityType 객체의 목록입니다.
- NextToken – UTF-8 문자열입니다.
추가 결과를 사용할 수 있는 경우 페이지 매김 토큰입니다.

Errors

- `InvalidInputException`
- `OperationTimeoutException`
- `InternalServiceException`

AWS Glue에서 API 태그 지정

데이터 타입

- [태그 구조](#)

태그 구조

Tag 객체는 AWS 리소스에 할당할 수 있는 레이블을 나타냅니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다.

AWS Glue의 태그 및 리소스 액세스 제어에 대한 자세한 내용은 개발자 가이드에서 [AWS Glue의 AWS 태그](#) 및 [AWS Glue 리소스 ARN 지정](#)을 참조하세요.

필드

- `key` – 1~128바이트 길이의 UTF-8 문자열입니다.

태그 키 객체에서 태그를 생성할 때 이 키는 필수입니다. 이 키는 대/소문자를 구분하며 접두사 `aws`를 포함해서는 안 됩니다.

- `value` – 256바이트 이하 길이의 UTF-8 문자열입니다.

태그 값 이 값은 객체에서 태그를 생성할 때 선택 사항입니다. 이 값은 대/소문자를 구분하며 접두사 `aws`를 포함해서는 안 됩니다.

운영

- [TagResource 작업\(Python: `tag_resource`\)](#)
- [UntagResource 작업\(Python: `untag_resource`\)](#)
- [GetTags 작업\(Python: `get_tags`\)](#)

TagResource 작업(Python: tag_resource)

태그를 리소스에 추가합니다. 태그는 AWS 리소스에 할당할 수 있는 레이블입니다. AWS Glue에서는 특정 리소스만 태그 지정할 수 있습니다. 태그 지정이 가능한 리소스에 대한 자세한 내용은 [AWS Glue의 AWS 태그](#)를 참조하세요.

태그 관련 API를 직접 호출하기 위한 태그 지정 권한 외에도 연결에서 태그 지정 API를 직접 호출할 수 있는 `glue:GetConnection` 권한과 데이터베이스에서 태그 지정 API를 직접 호출할 수 있는 `glue:GetDatabase` 권한도 필요합니다.

요청

- ResourceArn – 필수(Required): [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

태그를 추가할 AWS Glue 리소스의 ARN입니다. AWS Glue 리소스 ARN에 대한 자세한 내용은 [AWS Glue ARN 문자열 패턴](#)을 참조하세요.

- TagsToAdd – 필수(Required): 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.

각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.

각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.

이 리소스에 추가할 태그입니다.

응답

- 무응답 파라미터.

Errors

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- EntityNotFoundException

UntagResource 작업(Python: untag_resource)

리소스에서 태그를 제거합니다.

요청

- ResourceArn – 필수(Required): [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

태그를 제거할 리소스의 Amazon 리소스 이름(ARN)입니다.

- TagsToRemove – 필수(Required): 50개 이하의 문자열로 구성된 UTF-8 문자열의 배열입니다.

이 리소스에서 제거할 태그입니다.

응답

- 무응답 파라미터.

Errors

- InvalidInputException
- InternalServiceException
- OperationTimeoutException
- EntityNotFoundException

GetTags 작업(Python: get_tags)

리소스와 연결된 태그의 목록을 검색합니다.

요청

- ResourceArn – 필수(Required): [Custom string pattern #22](#)과(와) 일치하는 1~10,240바이트 길이의 UTF-8 문자열입니다.

태그를 검색할 리소스의 Amazon 리소스 이름(ARN)입니다.

응답

- Tags – 50개 이하의 페어로 구성된 키-값 페어의 맵 배열입니다.
 각 키는 길이가 1~128바이트인 UTF-8 문자열입니다.
 각 값은 256 바이트 이하 길이의 UTF-8 문자열입니다.
 요청된 태그입니다.

Errors

- `InvalidInputException`
- `InternalServiceException`
- `OperationTimeoutException`
- `EntityNotFoundException`

공통 데이터 형식

일반적인 데이터 유형은 AWS Glue의 기타 일반적인 데이터 유형에 대해 설명합니다.

태그 구조

Tag객체는 AWS 리소스에 할당할 수 있는 레이블을 나타냅니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다.

태그와 리소스에 대한 액세스 제어에 대한 자세한 내용은 개발자 [가이드의AWS 태그 AWS Glue](#) 입력 및 [AWS Glue 리소스 ARN 지정](#)을 참조하십시오. AWS Glue

필드

- `key` – 1~128바이트 길이의 UTF-8 문자열입니다.
 태그 키 객체에서 태그를 생성할 때 이 키는 필수입니다. 이 키는 대/소문자를 구분하며 접두사 `aws`를 포함해서는 안 됩니다.
- `value` – 256바이트 이하 길이의 UTF-8 문자열입니다.
 태그 값 이 값은 객체에서 태그를 생성할 때 선택 사항입니다. 이 값은 대/소문자를 구분하며 접두사 `aws`를 포함해서는 안 됩니다.

DecimalNumber 구조

십진수 형식의 숫자 값을 포함합니다.

필드

- `UnscaledValue` – 필수(Required): Blob입니다.
범위가 정해지지 않은 숫자 값.
- `Scale` – 필수(Required): 숫자(정수)입니다.
범위가 정해지지 않은 값 어디에 십진수가 오는지 결정하는 범위입니다.

ErrorDetail 구조

오류의 세부 정보를 포함합니다.

필드

- `ErrorCode` – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
이 오류와 연결된 코드입니다.
- `ErrorMessage` – [URI address multi-line string pattern](#)과(와) 일치하는 2,048바이트 이하 길이의 설명 문자열입니다.
메시지에서 오류를 설명합니다.

PropertyPredicate 구조

속성 조건자를 정의합니다.

필드

- `Key` – 1,024바이트 이하 길이의 값 문자열입니다.
속성 키입니다.
- `Value` – 1,024바이트 이하 길이의 값 문자열입니다.
속성 값입니다.

- **Comparator** – UTF-8 문자열입니다(유효 값: EQUALS | GREATER_THAN | LESS_THAN | GREATER_THAN_EQUALS | LESS_THAN_EQUALS).

비교자는 이 속성을 다른 속성과 비교합니다.

ResourceUri 구조

함수 리소스의 URI입니다.

필드

- **ResourceType** – UTF-8 문자열입니다(유효한 값: JAR | FILE | ARCHIVE).

리소스의 유형.

- **Uri** – [URI address multi-line string pattern](#)과(와) 일치하는 1~1,024바이트 길이의 URI(Uniform Resource Identifier)입니다.

리소스에 액세스하는 URI입니다.

ColumnStatistics 구조

테이블 또는 파티션에 대해 생성된 열 수준 통계를 나타냅니다.

필드

- **ColumnName** – 필수: [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

통계가 속한 열의 이름입니다.

- **ColumnType** – 필수(Required): [Single-line string pattern](#)과(와) 일치하는 길이 20,000바이트 이하의 유형 이름입니다.

열의 데이터 형식.

- **AnalyzedTime** – 필수(Required): 타임스탬프입니다.

열 통계가 생성된 시간의 타임스탬프입니다.

- **StatisticsData** – 필수(Required): [ColumnStatisticsData](#) 객체입니다.

통계 데이터 값을 포함하는 ColumnStatisticData 객체입니다.

ColumnStatisticsError 구조

실패한 ColumnStatistics 객체와 실패 이유를 캡슐화합니다.

필드

- ColumnStatistics – [ColumnStatistics](#) 객체입니다.
열의 ColumnStatistics입니다.
- Error – [ErrorDetail](#) 객체입니다.
작업 실패 이유가 포함된 오류 메시지입니다.

ColumnError 구조

실패한 열 이름과 실패 이유를 캡슐화합니다.

필드

- ColumnName – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.
실패한 열의 이름입니다.
- Error – [ErrorDetail](#) 객체입니다.
작업 실패 이유가 포함된 오류 메시지입니다.

ColumnStatisticsData 구조

열 통계 데이터의 개별 유형을 포함합니다. 하나의 데이터 객체만 설정하고 Type 속성으로 표시해야 합니다.

필드

- Type – 필수: UTF-8 문자열입니다(유효한 값: BOOLEAN | DATE | DECIMAL | DOUBLE | LONG | STRING | BINARY).
열 통계 데이터의 유형입니다.
- BooleanColumnStatisticsData – [BooleanColumnStatisticsData](#) 객체입니다.
부울 열 통계 데이터입니다.

- `DateColumnStatisticsData` – [DateColumnStatisticsData](#) 객체입니다.

날짜 열 통계 데이터입니다.

- `DecimalColumnStatisticsData` – [DecimalColumnStatisticsData](#) 객체입니다.

10진수 열 통계 데이터. `UnscaledValues` 내부에는 스케일링되지 않은 십진수 값을 보완하는 빅엔디안인 2를 저장하는 Base64로 인코딩된 이진 객체가 있습니다.

- `DoubleColumnStatisticsData` – [DoubleColumnStatisticsData](#) 객체입니다.

실수(Double) 열 통계 데이터입니다.

- `LongColumnStatisticsData` – [LongColumnStatisticsData](#) 객체입니다.

정수(Long) 열 통계 데이터입니다.

- `StringColumnStatisticsData` – [StringColumnStatisticsData](#) 객체입니다.

문자열 열 통계 데이터입니다.

- `BinaryColumnStatisticsData` – [BinaryColumnStatisticsData](#) 객체입니다.

이진수 열 통계 데이터입니다.

BooleanColumnStatisticsData 구조

부울 데이터 열에 대해 지원되는 열 통계를 정의합니다.

필드

- `NumberOfTrues` – 필수: None 이하의 숫자(정수)입니다.

열의 true 값 수입니다.

- `NumberOfFalses` – 필수(Required): None 이하의 숫자(정수)입니다.

열의 false 값 수입니다.

- `NumberOfNulls` – 필수(Required): None 이하의 숫자(정수)입니다.

열의 null 값 수입니다.

DateColumnStatisticsData 구조

타임스탬프 데이터 열에 대해 지원되는 열 통계를 정의합니다.

필드

- `MinimumValue` – 타임스탬프입니다.
열에서 최저 값입니다.
- `MaximumValue` – 타임스탬프입니다.
열에서 최고 값입니다.
- `NumberOfNulls` – 필수(Required): None 이하의 숫자(정수)입니다.
열의 null 값 수입니다.
- `NumberOfDistinctValues` – 필수(Required): None 이하의 숫자(정수)입니다.
열의 고유 값 수입니다.

DecimalColumnStatisticsData 구조

고정 소수점 데이터 열에 대해 지원되는 열 통계를 정의합니다.

필드

- `MinimumValue` – [DecimalNumber](#) 객체입니다.
열에서 최저 값입니다.
- `MaximumValue` – [DecimalNumber](#) 객체입니다.
열에서 최고 값입니다.
- `NumberOfNulls` – 필수(Required): None 이하의 숫자(정수)입니다.
열의 null 값 수입니다.
- `NumberOfDistinctValues` – 필수(Required): None 이하의 숫자(정수)입니다.
열의 고유 값 수입니다.

DoubleColumnStatisticsData 구조

부동 소수점 데이터 열에 대해 지원되는 열 통계를 정의합니다.

필드

- `MinimumValue` - 숫자(double)입니다.
열에서 최저 값입니다.
- `MaximumValue` - 숫자(double)입니다.
열에서 최고 값입니다.
- `NumberOfNulls` - 필수(Required): None 이하의 숫자(정수)입니다.
열의 null 값 수입니다.
- `NumberOfDistinctValues` - 필수(Required): None 이하의 숫자(정수)입니다.
열의 고유 값 수입니다.

LongColumnStatisticsData 구조

정수 데이터 열에 대해 지원되는 열 통계를 정의합니다.

필드

- `MinimumValue` - 숫자(정수)입니다.
열에서 최저 값입니다.
- `MaximumValue` - 숫자(정수)입니다.
열에서 최고 값입니다.
- `NumberOfNulls` - 필수(Required): None 이하의 숫자(정수)입니다.
열의 null 값 수입니다.
- `NumberOfDistinctValues` - 필수(Required): None 이하의 숫자(정수)입니다.
열의 고유 값 수입니다.

StringColumnStatisticsData 구조

문자 시퀀스 데이터 값에 대해 지원되는 열 통계를 정의합니다.

필드

- `MaxLength` – 필수: None 이하의 숫자(정수)입니다.
열에서 가장 긴 문자열의 크기입니다.
- `AverageLength` – 필수(Required): None 이하의 숫자(double)입니다.
열의 평균 문자열 길이입니다.
- `NumberOfNulls` – 필수(Required): None 이하의 숫자(정수)입니다.
열의 null 값 수입니다.
- `NumberOfDistinctValues` – 필수(Required): None 이하의 숫자(정수)입니다.
열의 고유 값 수입니다.

BinaryColumnStatisticsData 구조

비트 시퀀스 데이터 값에 대해 지원되는 열 통계를 정의합니다.

필드

- `MaxLength` – 필수: None 이하의 숫자(정수)입니다.
열에서 가장 긴 비트 시퀀스의 크기입니다.
- `AverageLength` – 필수(Required): None 이하의 숫자(double)입니다.
열의 평균 비트 시퀀스 길이입니다.
- `NumberOfNulls` – 필수(Required): None 이하의 숫자(정수)입니다.
열의 null 값 수입니다.

문자열 패턴

API는 다음 정규식을 사용하여 다양한 문자열 파라미터 및 멤버의 유효한 값이 무엇인지 정의합니다.

- 한 줄 문자열 패턴 - "[\u0020-\uD7FF\uE000-\uFFFF\uD800\uDC00-\uDBFF\uDFFF\t]*"
- URI 주소 여러 줄 문자열 패턴 - "[\u0020-\uD7FF\uE000-\uFFFF\uD800\uDC00-\uDBFF\uDFFF\r\n\t]*"
- Logstash Grok 문자열 패턴 - "[\u0020-\uD7FF\uE000-\uFFFF\uD800\uDC00-\uDBFF\uDFFF\r\t]*"
- 식별자 문자열 패턴 - "[A-Za-z_][A-Za-z0-9_]*"
- AWS IAM ARN 문자열 패턴 - "arn:aws:iam::\d{12}:role/.*"
 - 버전 문자열 패턴 - "^[a-zA-Z0-9-_-]+\$"
 - 로그 그룹 문자열 패턴 - "[\.\-_\/#A-Za-z0-9]+"
 - 로그 스트림 문자열 패턴 - "[^:]*"
 - 사용자 정의 문자열 패턴 #10 - "[^\r\n]"
 - 사용자 정의 문자열 패턴 #11 - "^arn:aws(-(cn|us-gov|iso(-[bef]))?)?:secretsmanager:.*\$"
 - 사용자 정의 문자열 패턴 #12 - "^(https?):\/\/[-a-zA-Z0-9+@#/%?=#~_!|:,.;]*[-a-zA-Z0-9+@#/%=#~_]"
 - 사용자 정의 문자열 패턴 #13 - "\S+"
 - 사용자 정의 문자열 패턴 #14 - "^(https?):\//\//[^\s/\$.?\#].[\s]*\$"
 - 사용자 정의 문자열 패턴 #15 - "^subnet-[a-z0-9]+\$"
 - 사용자 정의 문자열 패턴 #16 - "[\p{L}\p{N}\p{P}]*"
 - 사용자 정의 문자열 패턴 #17 - "[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"
 - 사용자 정의 문자열 패턴 #18 - "[a-zA-Z0-9-_\$#.]+"
 - 사용자 정의 문자열 패턴 #19 - "^w+\.\w+\.\w+\$"
 - 사용자 정의 문자열 패턴 #20 - "^w+\.\w+\$"
 - 사용자 정의 문자열 패턴 #21 - "^([2-3] | 3[.]9)\$"
 - 사용자 정의 문자열 패턴 #22 - "arn:(aws|aws-us-gov|aws-cn):glue:.*"
 - 사용자 지정 문자열 패턴 #23 - ' (^arn:aws:iam::\w{12}:root)'
 - 사용자 정의 문자열 패턴 #24 - "^arn:aws(-(cn|us-gov|iso(-[bef]))?)?:iam::[0-9]{12}:role/.+"
 - 사용자 정의 문자열 패턴 #25 - "arn:aws:kms:.*"
 - 사용자 정의 문자열 패턴 #26 - "arn:aws[^:]*:iam::[0-9]*:role/.+"

- 사용자 정의 문자열 패턴 #27 - "[\\.\\-_A-Za-z0-9]+"
- 사용자 정의 문자열 패턴 #28 - "^s3://([^/]+)/([^/]+/*)([^/]+)\$"
- 사용자 정의 문자열 패턴 #29 - ".*"
- 사용자 정의 문자열 패턴 #30 - "^((Sun|Mon|Tue|Wed|Thu|Fri|Sat):([01]?[0-9]|2[0-3]))\$"
- 사용자 정의 문자열 패턴 #31 - "[a-zA-Z0-9_.-]+"
- 사용자 정의 문자열 패턴 #32 - ".*\S.*"
- 사용자 정의 문자열 패턴 #33 - "[a-zA-Z0-9-=_./@]+"
- 사용자 지정 문자열 패턴 #34 - "[1-9][0-9]*|[1-9][0-9]*-[1-9][0-9]*"
- 사용자 지정 문자열 패턴 #35 - "[\s\S]*"
- 사용자 지정 문자열 패턴 #36 - "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[\^\\S\\r\\n"'= ;])*"
 - 사용자 지정 문자열 패턴 #37 - "[*A-Za-z0-9_-]*"
 - 사용자 지정 문자열 패턴 #38 - "([\u0020-\u007E\\r\\s\\n])*"
 - 사용자 지정 문자열 패턴 #39 - "[A-Za-z0-9_-]*"
 - 사용자 지정 문자열 패턴 #40 - "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[\^\\S\\r\\n''])*"
 - 사용자 지정 문자열 패턴 #41 - "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[\^\\S\\r\\n])*"
 - 사용자 지정 문자열 패턴 #42 - "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\\s])*"
 - 사용자 지정 문자열 패턴 #43 — "([\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF]|[\^\\r\\n])*"

예외

이 섹션에서는 문제의 원인을 찾고 해결하는 데 사용할 수 있는 AWS Glue 예외사항에 대해 설명합니다. 기계 학습과 관련된 예외에 관한 HTTP 오류 코드 및 문자열에 대한 자세한 내용은 [the section called “AWS Glue 기계 학습 예외 사항”](#) 단원을 참조하십시오.

AccessDeniedException 구조

리소스 액세스가 거부됩니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

AlreadyExistsException 구조

생성되는 또는 추가되는 리소스가 이미 있습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

ConcurrentModificationException 구조

두 절차로 동시에 리소스를 수정하고자 합니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

ConcurrentRunsExceededException 구조

다양한 작업이 동시에 실행됩니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

CrawlerNotRunningException 구조

지정한 크롤러가 실행되고 있지 않습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

CrawlerRunningException 구조

크롤러가 이미 수행되고 있기 때문에 작업을 수행할 수 없습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

CrawlerStoppingException 구조

지정한 크롤러가 중지되고 있습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

EntityNotFoundException 구조

지정한 개체가 존재하지 않습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

- FromFederationSource – 부울입니다.

예외가 페더레이션된 소스와 관련이 있는지 여부를 나타냅니다.

FederationSourceException 구조

페더레이션 소스에 실패했습니다.

필드

- FederationSourceErrorCode – UTF-8 문자열입니다(유효한 값: AccessDeniedException | EntityNotFoundException | InvalidCredentialsException | InvalidInputException | InvalidResponseException | OperationTimeoutException | OperationNotSupportedException | InternalServiceException | PartialFailureException | ThrottlingException).

문제의 오류 코드입니다.

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지입니다.

FederationSourceRetryableException 구조

페더레이션 소스에 실패했지만 작업이 재시도될 수 있습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

GlueEncryptionException 구조

암호화 연산에 실패했습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지입니다.

IdempotentParameterMismatchException 구조

두 가지 다른 기록과 연결된 동일한 고유 식별자입니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

IllegalWorkflowStateException 구조

워크플로가 요청된 작업을 수행할 수 있는 잘못된 상태입니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

InternalServiceException 구조

내부 서비스 오류가 발생했습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

InvalidExecutionEngineException 구조

알려지지 않거나 무효한 엔진이 지정되었습니다.

필드

- message – UTF-8 문자열입니다.

문제를 설명하는 메시지

InvalidInputException 구조

제공한 입력이 유효하지 않았습니다.

필드

- Message – UTF-8 문자열입니다.
문제를 설명하는 메시지
- FromFederationSource – 부울입니다.
예외가 페더레이션된 소스와 관련이 있는지 여부를 나타냅니다.

InvalidStateException 구조

데이터 상태가 잘못되었음을 나타내는 오류입니다.

필드

- Message – UTF-8 문자열입니다.
문제를 설명하는 메시지

InvalidTaskStatusTransitionException 구조

하나의 작업이 다음 실패한 작업으로의 적절한 전환.

필드

- message – UTF-8 문자열입니다.
문제를 설명하는 메시지

JobDefinitionErrorException 구조

작업 정의가 유효하지 않습니다.

필드

- message – UTF-8 문자열입니다.

문제를 설명하는 메시지

JobRunInTerminalStateException 구조

작업 실행의 터미널 상태는 실패를 암시합니다.

필드

- message – UTF-8 문자열입니다.

문제를 설명하는 메시지

JobRunInvalidStateTransitionException 구조

작업 실행으로 원본 상태에서 대상 상태로의 무효한 전환이 발생했습니다.

필드

- jobRunId – [Single-line string pattern](#)과(와) 일치하는 1~255바이트 길이의 UTF-8 문자열입니다.

문제의 작업 실행 ID입니다.

- message – UTF-8 문자열입니다.

문제를 설명하는 메시지

- sourceState – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED).

소스 데이터

- targetState – UTF-8 문자열입니다(유효한 값: STARTING | RUNNING | STOPPING | STOPPED | SUCCEEDED | FAILED | TIMEOUT | ERROR | WAITING | EXPIRED).

대상 상태

JobRunNotInTerminalStateException 구조

터미널 상태에서 작업이 실행되지 않습니다.

필드

- message – UTF-8 문자열입니다.

문제를 설명하는 메시지

LateRunnerException 구조

작업 실행이 늦습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

NoScheduleException 구조

적용 가능한 일정이 없습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

OperationTimeoutException 구조

작업 시간이 초과되었습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

ResourceNotReadyException 구조

리소스가 트랜잭션에 대해 준비되지 않았습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

ResourceNumberLimitExceededException 구조

리소스의 숫자 제한을 초과했습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

SchedulerNotRunningException 구조

지정한 스케줄러가 실행되고 있지 않습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

SchedulerRunningException 구조

지정한 스케줄러가 이미 실행되고 있습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

SchedulerTransitioningException 구조

지정한 스케줄러가 변환되고 있습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

UnrecognizedRunnerException 구조

작업 실행이 인식되지 않습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

ValidationException 구조

값이 유효하지 않을 수 있습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

VersionMismatchException 구조

버전 충돌이 있었습니다.

필드

- Message – UTF-8 문자열입니다.

문제를 설명하는 메시지

AWS Glue AWS SDK를 사용한 API 코드 예제

다음 코드 예제는 AWS 소프트웨어 개발 키트 (SDK) AWS Glue 와 함께 사용하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 교차 서비스 예시에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 [여기](#)를 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

시작하기

안녕하세요. AWS Glue

다음 코드 예제에서는 AWS Glue의 사용을 시작하는 방법을 보여 줍니다.

.NET

AWS SDK for .NET

Note

더 많은 정보가 있어요 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
namespace GlueActions;

public class HelloGlue
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Glue.
        using var host = Host.CreateDefaultBuilder(args)
```

```
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
                    LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft",
                    LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonGlue>()
                    .AddTransient<GlueWrapper>()
            )
            .Build();

logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
    .CreateLogger<HelloGlue>();
var glueClient = host.Services.GetRequiredService<IAmazonGlue>();

var request = new ListJobsRequest();

var jobNames = new List<string>();

do
{
    var response = await glueClient.ListJobsAsync(request);
    jobNames.AddRange(response.JobNames);
    request.NextToken = response.NextToken;
}
while (request.NextToken is not null);

Console.Clear();
Console.WriteLine("Hello, Glue. Let's list your existing Glue Jobs:");
if (jobNames.Count == 0)
{
    Console.WriteLine("You don't have any AWS Glue jobs.");
}
else
{
    jobNames.ForEach(Console.WriteLine);
}
}
```

- API 세부 정보는 AWS SDK for .NET API [ListJobs](#) 참조를 참조하십시오.

C++

SDK for C++

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

C MakeLists.txt CMake 파일의 코드입니다.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS glue)

# Set this project's name.
project("hello_glue")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
  libraries for the AWS SDK.
  string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
    "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
  list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
  # Copy relevant AWS SDK for C++ libraries into the current binary directory
  for running and debugging.
```

```

    # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
    may need to uncomment this

                                # and set the proper subdirectory to the
    executables' location.

    AWSSDK_COPY_DYN_LIBS(SERVICE_COMPONENTS ""
    ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
    hello_glue.cpp)

target_link_libraries(${PROJECT_NAME}
    ${AWSSDK_LINK_LIBRARIES})

```

hello_glue.cpp 소스 파일의 코드입니다.

```

#include <aws/core/Aws.h>
#include <aws/glue/GlueClient.h>
#include <aws/glue/model/ListJobsRequest.h>
#include <iostream>

/*
 * A "Hello Glue" starter application which initializes an AWS Glue client and
 * lists the
 * AWS Glue job definitions.
 *
 * main function
 *
 * Usage: 'hello_glue'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
    // options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).

```



```

// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient glueClient(clientConfig);

std::vector<Aws::String> jobs;

Aws::String nextToken; // Used for pagination.
do {
    Aws::Glue::Model::ListJobsRequest listJobsRequest;
    if (!nextToken.empty()) {
        listJobsRequest.SetNextToken(nextToken);
    }

    Aws::Glue::Model::ListJobsOutcome listRunsOutcome =
glueClient.ListJobs(
        listJobsRequest);

    if (listRunsOutcome.IsSuccess()) {
        const std::vector<Aws::String> &jobNames =
listRunsOutcome.GetResult().GetJobNames();
        jobs.insert(jobs.end(), jobNames.begin(), jobNames.end());

        nextToken = listRunsOutcome.GetResult().GetNextToken();
    } else {
        std::cerr << "Error listing jobs. "
            << listRunsOutcome.GetError().GetMessage()
            << std::endl;

        result = 1;
        break;
    }
} while (!nextToken.empty());

std::cout << "Your account has " << jobs.size() << " jobs."
    << std::endl;
for (size_t i = 0; i < jobs.size(); ++i) {
    std::cout << "    " << i + 1 << ". " << jobs[i] << std::endl;
}
}
Aws::ShutdownAPI(options); // Should only be called once.
return result;
}

```

- API에 대한 자세한 내용은 API 레퍼런스를 참조하십시오 [ListJobs](#).AWS SDK for C++

Java

SDK for Java 2.x

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
package com.example.glue;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.ListJobsRequest;
import software.amazon.awssdk.services.glue.model.ListJobsResponse;
import java.util.List;

public class HelloGlue {
    public static void main(String[] args) {
        GlueClient glueClient = GlueClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listJobs(glueClient);
    }

    public static void listJobs(GlueClient glueClient) {
        ListJobsRequest request = ListJobsRequest.builder()
            .maxResults(10)
            .build();
        ListJobsResponse response = glueClient.listJobs(request);
        List<String> jobList = response.jobNames();
        jobList.forEach(job -> {
            System.out.println("Job Name: " + job);
        });
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API [ListJobs](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";

const client = new GlueClient({});

export const main = async () => {
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- API 세부 정보는 AWS SDK for JavaScript API [ListJobs](#)참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
let mut list_jobs = glue.list_jobs().into_paginator().send();
while let Some(list_jobs_output) = list_jobs.next().await {
  match list_jobs_output {
```

```
Ok(list_jobs) => {
    let names = list_jobs.job_names();
    info!(?names, "Found these jobs")
}
Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
}
}
```

- API에 대한 자세한 내용은 Rust용AWS SDK API 레퍼런스를 참조하십시오 [ListJobs](#).

코드 예시

- [AWS SDK AWS Glue 사용을 위한 조치](#)
 - [AWS SDK 또는 CreateCrawler CLI와 함께 사용](#)
 - [AWS SDK 또는 CreateJob CLI와 함께 사용](#)
 - [AWS SDK 또는 DeleteCrawler CLI와 함께 사용](#)
 - [AWS SDK 또는 DeleteDatabase CLI와 함께 사용](#)
 - [AWS SDK 또는 DeleteJob CLI와 함께 사용](#)
 - [AWS SDK 또는 DeleteTable CLI와 함께 사용](#)
 - [AWS SDK 또는 GetCrawler CLI와 함께 사용](#)
 - [AWS SDK 또는 GetDatabase CLI와 함께 사용](#)
 - [AWS SDK 또는 GetDatabases CLI와 함께 사용](#)
 - [AWS SDK 또는 GetJob CLI와 함께 사용](#)
 - [AWS SDK 또는 GetJobRun CLI와 함께 사용](#)
 - [AWS SDK 또는 GetJobRuns CLI와 함께 사용](#)
 - [AWS SDK 또는 GetTables CLI와 함께 사용](#)
 - [AWS SDK 또는 ListJobs CLI와 함께 사용](#)
 - [AWS SDK 또는 StartCrawler CLI와 함께 사용](#)
 - [AWS SDK 또는 StartJobRun CLI와 함께 사용](#)
- [AWS SDK AWS Glue 사용 시나리오](#)
 - [SDK를 사용하여 AWS Glue 크롤러 및 작업 실행 시작하기 AWS](#)

AWS SDK AWS Glue 사용을 위한 조치

다음 코드 예제는 AWS SDK를 사용하여 개별 AWS Glue 작업을 수행하는 방법을 보여줍니다. 이 발췌문은 AWS Glue API를 호출하며 컨텍스트에서 실행해야 하는 대규모 프로그램에서 발췌한 코드입니다. 각 예제에는 코드 설정 및 실행 지침을 찾을 수 있는 링크가 포함되어 있습니다. [GitHub](#)

다음 예제에는 가장 일반적으로 사용되는 작업만 포함되어 있습니다. 전체 목록은 [AWS Glue API 참조](#)를 참조하세요.

예

- [AWS SDK 또는 CreateCrawler CLI와 함께 사용](#)
- [AWS SDK 또는 CreateJob CLI와 함께 사용](#)
- [AWS SDK 또는 DeleteCrawler CLI와 함께 사용](#)
- [AWS SDK 또는 DeleteDatabase CLI와 함께 사용](#)
- [AWS SDK 또는 DeleteJob CLI와 함께 사용](#)
- [AWS SDK 또는 DeleteTable CLI와 함께 사용](#)
- [AWS SDK 또는 GetCrawler CLI와 함께 사용](#)
- [AWS SDK 또는 GetDatabase CLI와 함께 사용](#)
- [AWS SDK 또는 GetDatabases CLI와 함께 사용](#)
- [AWS SDK 또는 GetJob CLI와 함께 사용](#)
- [AWS SDK 또는 GetJobRun CLI와 함께 사용](#)
- [AWS SDK 또는 GetJobRuns CLI와 함께 사용](#)
- [AWS SDK 또는 GetTables CLI와 함께 사용](#)
- [AWS SDK 또는 ListJobs CLI와 함께 사용](#)
- [AWS SDK 또는 StartCrawler CLI와 함께 사용](#)
- [AWS SDK 또는 StartJobRun CLI와 함께 사용](#)

AWS SDK 또는 **CreateCrawler** CLI와 함께 사용

다음 코드 예제는 CreateCrawler의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Create an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name for the crawler.</param>
/// <param name="crawlerDescription">A description of the crawler.</param>
/// <param name="role">The AWS Identity and Access Management (IAM) role to
/// be assumed by the crawler.</param>
/// <param name="schedule">The schedule on which the crawler will be
executed.</param>
/// <param name="s3Path">The path to the Amazon Simple Storage Service
(Amazon S3)
/// bucket where the Python script has been stored.</param>
/// <param name="dbName">The name to use for the database that will be
/// created by the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateCrawlerAsync(
    string crawlerName,
    string crawlerDescription,
    string role,
    string schedule,
    string s3Path,
    string dbName)
{
    var s3Target = new S3Target
    {
        Path = s3Path,
    };

    var targetList = new List<S3Target>
    {
```

```

        s3Target,
    };

    var targets = new CrawlerTargets
    {
        S3Targets = targetList,
    };

    var crawlerRequest = new CreateCrawlerRequest
    {
        DatabaseName = dbName,
        Name = crawlerName,
        Description = crawlerDescription,
        Targets = targets,
        Role = role,
        Schedule = schedule,
    };

    var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

```

- API 세부 정보는 AWS SDK for .NET API [CreateCrawler](#)참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region in which the bucket was created
    (overrides config file).
    // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

```

```

Aws::Glue::Model::S3Target s3Target;
s3Target.SetPath("s3://crawler-public-us-east-1/flight/2016/csv");
Aws::Glue::Model::CrawlerTargets crawlerTargets;
crawlerTargets.AddS3Targets(s3Target);

Aws::Glue::Model::CreateCrawlerRequest request;
request.SetTargets(crawlerTargets);
request.SetName(CRAWLER_NAME);
request.SetDatabaseName(CRAWLER_DATABASE_NAME);
request.SetTablePrefix(CRAWLER_DATABASE_PREFIX);
request.SetRole(roleArn);

Aws::Glue::Model::CreateCrawlerOutcome outcome =
client.CreateCrawler(request);

if (outcome.IsSuccess()) {
    std::cout << "Successfully created the crawler." << std::endl;
}
else {
    std::cerr << "Error creating a crawler. " <<
outcome.GetError().GetMessage()
    << std::endl;
    deleteAssets("", CRAWLER_DATABASE_NAME, "", bucketName,
clientConfig);
    return false;
}

```

- API 세부 정보는 AWS SDK for C++ API [CreateCrawler](#) 참조를 참조하십시오.

Java

SDK for Java 2.x

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
import software.amazon.awssdk.regions.Region;
```



```
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.CreateCrawlerRequest;
import software.amazon.awssdk.services.glue.model.CrawlerTargets;
import software.amazon.awssdk.services.glue.model.GlueException;
import software.amazon.awssdk.services.glue.model.S3Target;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class CreateCrawler {
    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <IAM> <s3Path> <cron> <dbName> <crawlerName>

            Where:
                IAM - The ARN of the IAM role that has AWS Glue and S3
permissions.\s
                s3Path - The Amazon Simple Storage Service (Amazon S3) target
that contains data (for example, CSV data).
                cron - A cron expression used to specify the schedule (i.e.,
cron(15 12 * * ? *).
                dbName - The database name.\s
                crawlerName - The name of the crawler.\s
            """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String iam = args[0];
        String s3Path = args[1];
        String cron = args[2];
        String dbName = args[3];
```

```
String crawlerName = args[4];
Region region = Region.US_EAST_1;
GlueClient glueClient = GlueClient.builder()
    .region(region)
    .build();

createGlueCrawler(glueClient, iam, s3Path, cron, dbName, crawlerName);
glueClient.close();
}

public static void createGlueCrawler(GlueClient glueClient,
    String iam,
    String s3Path,
    String cron,
    String dbName,
    String crawlerName) {

    try {
        S3Target s3Target = S3Target.builder()
            .path(s3Path)
            .build();

        // Add the S3Target to a list.
        List<S3Target> targetList = new ArrayList<>();
        targetList.add(s3Target);

        CrawlerTargets targets = CrawlerTargets.builder()
            .s3Targets(targetList)
            .build();

        CreateCrawlerRequest crawlerRequest = CreateCrawlerRequest.builder()
            .databaseName(dbName)
            .name(crawlerName)
            .description("Created by the AWS Glue Java API")
            .targets(targets)
            .role(iam)
            .schedule(cron)
            .build();

        glueClient.createCrawler(crawlerRequest);
        System.out.println(crawlerName + " was successfully created");

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
    }
}
```

```

        System.exit(1);
    }
}
}

```

- API 세부 정보는 AWS SDK for Java 2.x API [CreateCrawler](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

```

- API 세부 정보는 AWS SDK for JavaScript API [CreateCrawler](#)참조를 참조하십시오.

Kotlin

SDK for Kotlin

 Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
suspend fun createGlueCrawler(  
    iam: String?,  
    s3Path: String?,  
    cron: String?,  
    dbName: String?,  
    crawlerName: String,  
) {  
    val s3Target =  
        S3Target {  
            path = s3Path  
        }  
  
    // Add the S3Target to a list.  
    val targetList = mutableListOf<S3Target>()  
    targetList.add(s3Target)  
  
    val targetOb =  
        CrawlerTargets {  
            s3Targets = targetList  
        }  
  
    val request =  
        CreateCrawlerRequest {  
            databaseName = dbName  
            name = crawlerName  
            description = "Created by the AWS Glue Kotlin API"  
            targets = targetOb  
            role = iam  
            schedule = cron  
        }  
  
    GlueClient { region = "us-west-2" }.use { glueClient ->
```

```

        glueClient.createCrawler(request)
        println("$crawlerName was successfully created")
    }
}

```

- API 세부 정보는 Kotlin API용 AWS SDK 레퍼런스를 참조하세요 [CreateCrawler](#).

PHP

SDK for PHP

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

$crawlerName = "example-crawler-test-" . $uniqid;

$role = $iamService->getRole("AWSGlueServiceRole-DocExample");

$path = 's3://crawler-public-us-east-1/flight/2016/csv';
$glueService->createCrawler($crawlerName, $role['Role']['Arn'],
$databaseName, $path);

public function createCrawler($crawlerName, $role, $databaseName, $path):
Result
{
    return $this->customWaiter(function () use ($crawlerName, $role,
$databaseName, $path) {
        return $this->glueClient->createCrawler([
            'Name' => $crawlerName,
            'Role' => $role,
            'DatabaseName' => $databaseName,
            'Targets' => [
                'S3Targets' =>
                    [[
                        'Path' => $path,
                    ]],
            ],
        ]);
    });
}

```

```
    });
}
```

- API 세부 정보는 AWS SDK for PHP API [CreateCrawler](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def create_crawler(self, name, role_arn, db_name, db_prefix, s3_target):
        """
        Creates a crawler that can crawl the specified target and populate a
        database in your AWS Glue Data Catalog with metadata that describes the
        data
        in the target.

        :param name: The name of the crawler.
        :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and
        Access
                           Management (IAM) role that grants permission to let AWS
        Glue
                           access the resources it needs.
        :param db_name: The name to give the database that is created by the
        crawler.
```

```

        :param db_prefix: The prefix to give any database tables that are created
by
                        the crawler.
        :param s3_target: The URL to an S3 bucket that contains data that is
                        the target of the crawler.
        """
        try:
            self.glue_client.create_crawler(
                Name=name,
                Role=role_arn,
                DatabaseName=db_name,
                TablePrefix=db_prefix,
                Targets={"S3Targets": [{"Path": s3_target}]},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create crawler. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [CreateCrawler](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
a simplified interface for common operations.

```

```
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Creates a new crawler with the specified configuration.
  #
  # @param name [String] The name of the crawler.
  # @param role_arn [String] The ARN of the IAM role to be used by the crawler.
  # @param db_name [String] The name of the database where the crawler stores its
  # metadata.
  # @param db_prefix [String] The prefix to be added to the names of tables that
  # the crawler creates.
  # @param s3_target [String] The S3 path that the crawler will crawl.
  # @return [void]
  def create_crawler(name, role_arn, db_name, db_prefix, s3_target)
    @glue_client.create_crawler(
      name: name,
      role: role_arn,
      database_name: db_name,
      targets: {
        s3_targets: [
          {
            path: s3_target
          }
        ]
      }
    )
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not create crawler: \n#{e.message}")
    raise
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API [CreateCrawler](#) 참조를 참조하십시오.

Rust

SDK for Rust

 Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
let create_crawler = glue
    .create_crawler()
    .name(self.crawler())
    .database_name(self.database())
    .role(self.iam_role.expose_secret())
    .targets(
        CrawlerTargets::builder()
            .s3_targets(S3Target::builder().path(CRAWLER_TARGET).build())
            .build(),
    )
    .send()
    .await;

match create_crawler {
    Err(err) => {
        let glue_err: aws_sdk_glue::Error = err.into();
        match glue_err {
            aws_sdk_glue::Error::AlreadyExistsException(_) => {
                info!("Using existing crawler");
                Ok(())
            }
            _ => Err(GlueMvpError::GlueSdk(glue_err)),
        }
    }
    Ok(_) => Ok(()),
}??;
```

- API에 대한 자세한 내용은 Rust용AWS SDK API 레퍼런스를 참조하십시오 [CreateCrawler](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **CreateJob** CLI와 함께 사용

다음 코드 예제는 CreateJob의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

/// <summary>
/// Create an AWS Glue job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <param name="roleName">The name of the IAM role to be assumed by
/// the job.</param>
/// <param name="description">A description of the job.</param>
/// <param name="scriptUrl">The URL to the script.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateJobAsync(string dbName, string tableName,
string bucketUrl, string jobName, string roleName, string description, string
scriptUrl)
{
    var command = new JobCommand
    {
        PythonVersion = "3",
        Name = "glueetl",
        ScriptLocation = scriptUrl,
    };

```

```

var arguments = new Dictionary<string, string>
{
    { "--input_database", dbName },
    { "--input_table", tableName },
    { "--output_bucket_url", bucketUrl }
};

var request = new CreateJobRequest
{
    Command = command,
    DefaultArguments = arguments,
    Description = description,
    GlueVersion = "3.0",
    Name = jobName,
    NumberOfWorkers = 10,
    Role = roleName,
    WorkerType = "G.1X"
};

var response = await _amazonGlue.CreateJobAsync(request);
return response.HttpStatusCode == HttpStatusCode.OK;
}

```

- API 세부 정보는 AWS SDK for .NET API [CreateJob](#) 참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

```

```

Aws::Glue::GlueClient client(clientConfig);

    Aws::Glue::Model::CreateJobRequest request;
    request.SetName(JOB_NAME);
    request.SetRole(roleArn);
    request.SetGlueVersion(GLUE_VERSION);

    Aws::Glue::Model::JobCommand command;
    command.SetName(JOB_COMMAND_NAME);
    command.SetPythonVersion(JOB_PYTHON_VERSION);
    command.SetScriptLocation(
        Aws::String("s3://") + bucketName + "/" + PYTHON_SCRIPT);
    request.SetCommand(command);

    Aws::Glue::Model::CreateJobOutcome outcome = client.CreateJob(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully created the job." << std::endl;
    }
    else {
        std::cerr << "Error creating the job. " <<
outcome.GetError().GetMessage()
        << std::endl;
        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
            clientConfig);
        return false;
    }

```

- API 세부 정보는 AWS SDK for C++ API [CreateJob](#) 참조를 참조하십시오.

CLI

AWS CLI

데이터를 변환하는 작업을 생성하려면

다음 create-job 예제는 S3에 저장된 스크립트를 실행하는 스트리밍 작업을 생성합니다.

```

aws glue create-job \
    --name my-testing-job \
    --role AWSGlueServiceRoleDefault \

```

```

--command '{ \
  "Name": "gluestreaming", \
  "ScriptLocation": "s3://DOC-EXAMPLE-BUCKET/folder/" \
}' \
--region us-east-1 \
--output json \
--default-arguments '{ \
  "--job-language":"scala", \
  "--class":"GlueApp" \
}' \
--profile my-profile \
--endpoint https://glue.us-east-1.amazonaws.com

```

test_script.scala의 콘텐츠:

```

import com.amazonaws.services.glue.ChoiceOption
import com.amazonaws.services.glue.GlueContext
import com.amazonaws.services.glue.MappingSpec
import com.amazonaws.services.glue.ResolveSpec
import com.amazonaws.services.glue.errors.CallSite
import com.amazonaws.services.glue.util.GlueArgParser
import com.amazonaws.services.glue.util.Job
import com.amazonaws.services.glue.util.JsonOptions
import org.apache.spark.SparkContext
import scala.collection.JavaConverters._

object GlueApp {
  def main(sysArgs: Array[String]) {
    val spark: SparkContext = new SparkContext()
    val glueContext: GlueContext = new GlueContext(spark)
    // @params: [JOB_NAME]
    val args = GlueArgParser.getResolvedOptions(sysArgs,
Seq("JOB_NAME").toArray)
    Job.init(args("JOB_NAME"), glueContext, args.asJava)
    // @type: DataSource
    // @args: [database = "tempdb", table_name = "s3-source",
transformation_ctx = "datasource0"]
    // @return: datasource0
    // @inputs: []
    val datasource0 = glueContext.getCatalogSource(database = "tempdb",
tableName = "s3-source", redshiftTmpDir = "", transformationContext =
"datasource0").getDynamicFrame()
    // @type: ApplyMapping

```

```

    // @args: [mapping = [("sensorid", "int", "sensorid", "int"),
("currenttemperature", "int", "currenttemperature", "int"), ("status", "string",
"status", "string")], transformation_ctx = "applymapping1"]
    // @return: applymapping1
    // @inputs: [frame = datasource0]
    val applymapping1 = datasource0.applyMapping(mappings = Seq(("sensorid",
"int", "sensorid", "int"), ("currenttemperature", "int", "currenttemperature",
"int"), ("status", "string", "status", "string")), caseSensitive = false,
transformationContext = "applymapping1")
    // @type: SelectFields
    // @args: [paths = ["sensorid", "currenttemperature", "status"],
transformation_ctx = "selectfields2"]
    // @return: selectfields2
    // @inputs: [frame = applymapping1]
    val selectfields2 = applymapping1.selectFields(paths = Seq("sensorid",
"currenttemperature", "status"), transformationContext = "selectfields2")
    // @type: ResolveChoice
    // @args: [choice = "MATCH_CATALOG", database = "tempdb", table_name =
"my-s3-sink", transformation_ctx = "resolvechoice3"]
    // @return: resolvechoice3
    // @inputs: [frame = selectfields2]
    val resolvechoice3 = selectfields2.resolveChoice(choiceOption =
Some(ChoiceOption("MATCH_CATALOG")), database = Some("tempdb"), tableName =
Some("my-s3-sink"), transformationContext = "resolvechoice3")
    // @type: DataSink
    // @args: [database = "tempdb", table_name = "my-s3-sink",
transformation_ctx = "datasink4"]
    // @return: datasink4
    // @inputs: [frame = resolvechoice3]
    val datasink4 = glueContext.getCatalogSink(database = "tempdb",
tableName = "my-s3-sink", redshiftTmpDir = "", transformationContext =
"datasink4").writeDynamicFrame(resolvechoice3)
    Job.commit()
  }
}

```

출력:

```

{
  "Name": "my-testing-job"
}

```

자세한 내용은 Glue 개발자 가이드의 AWS [Glue에서 작업 작성](#)을 AWS 참조하십시오.

- API에 대한 자세한 내용은 AWS CLI 명령 참조를 참조하십시오 [CreateJob](#).

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [CreateJob](#) 참조를 참조하십시오.

PHP

SDK for PHP

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
$role = $iamService->getRole("AWSGlueServiceRole-DocExample");

$jobName = 'test-job-' . $uniqid;

$scriptLocation = "s3://$bucketName/run_job.py";
$job = $glueService->createJob($jobName, $role['Role']['Arn'],
$scriptLocation);

public function createJob($jobName, $role, $scriptLocation, $pythonVersion =
'3', $glueVersion = '3.0'): Result
{
    return $this->glueClient->createJob([
        'Name' => $jobName,
        'Role' => $role,
        'Command' => [
            'Name' => 'glueetl',
            'ScriptLocation' => $scriptLocation,
            'PythonVersion' => $pythonVersion,
        ],
        'GlueVersion' => $glueVersion,
    ]);
}
```

- API 세부 정보는 AWS SDK for PHP API [CreateJob](#)참조를 참조하십시오.

PowerShell

다음을 위한 도구 PowerShell

예 1: 이 예제는 AWS Glue에서 새 작업을 생성합니다. 명령 이름 값은 항상 **glueet1**. AWS Glue는 Python 또는 Scala로 작성된 작업 스크립트 실행을 지원합니다. 이 예제에서 작업 스크립트 (MyTestGlueJob.py) 는 Python으로 작성되었습니다. Python 매개 변수는 **\$DefArgs** 변수에 지정된 다음 해시 테이블을 허용하는 **DefaultArguments** 매개 변수의 PowerShell 명령에 전달됩니다. **\$JobParams** 변수의 매개 변수는 API에서 가져온 것으로, CreateJob AWS Glue API 참조의 작업 (<https://docs.aws.amazon.com/glue/latest/dg/aws-glue-api-jobs-job.html>) 주제에 설명되어 있습니다.

```
$Command = New-Object Amazon.Glue.Model.JobCommand
$Command.Name = 'glueet1'
$Command.ScriptLocation = 's3://aws-glue-scripts-000000000000-us-west-2/admin/MyTestGlueJob.py'
$Command

$Source = "source_test_table"
$Target = "target_test_table"
$Connections = $Source, $Target

$DefArgs = @{
    '--TempDir' = 's3://aws-glue-temporary-000000000000-us-west-2/admin'
    '--job-bookmark-option' = 'job-bookmark-disable'
    '--job-language' = 'python'
}
$DefArgs

$ExecutionProp = New-Object Amazon.Glue.Model.ExecutionProperty
$ExecutionProp.MaxConcurrentRuns = 1
$ExecutionProp

$JobParams = @{
    "AllocatedCapacity" = "5"
    "Command" = $Command
    "Connections_Connection" = $Connections
    "DefaultArguments" = $DefArgs
    "Description" = "This is a test"
    "ExecutionProperty" = $ExecutionProp
    "MaxRetries" = "1"
    "Name" = "My0regonTestGlueJob"
```

```

"Role"           = "Amazon-GlueServiceRoleForSSM"
"Timeout"       = "20"
}

```

```
New-GlueJob @JobParams
```

- API에 대한 자세한 내용은 AWS Tools for PowerShell Cmdlet 참조를 참조하십시오 [CreateJob](#).

Python

SDK for Python(Boto3)

Note

자세한 내용은 다음과 같습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def create_job(self, name, description, role_arn, script_location):
        """
        Creates a job definition for an extract, transform, and load (ETL) job
        that can
        be run by AWS Glue.

        :param name: The name of the job definition.
        :param description: The description of the job definition.
        :param role_arn: The ARN of an IAM role that grants AWS Glue the
        permissions
                           it requires to run the job.

```

```
        :param script_location: The Amazon S3 URL of a Python ETL script that is
run as                                part of the job. The script defines how the data
is                                    transformed.

"""
try:
    self.glue_client.create_job(
        Name=name,
        Description=description,
        Role=role_arn,
        Command={
            "Name": "glueetl",
            "ScriptLocation": script_location,
            "PythonVersion": "3",
        },
        GlueVersion="3.0",
    )
except ClientError as err:
    logger.error(
        "Couldn't create job %s. Here's why: %s: %s",
        name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [CreateJob](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.


```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Creates a new job with the specified configuration.
  #
  # @param name [String] The name of the job.
  # @param description [String] The description of the job.
  # @param role_arn [String] The ARN of the IAM role to be used by the job.
  # @param script_location [String] The location of the ETL script for the job.
  # @return [void]
  def create_job(name, description, role_arn, script_location)
    @glue_client.create_job(
      name: name,
      description: description,
      role: role_arn,
      command: {
        name: "glueetl",
        script_location: script_location,
        python_version: "3"
      },
      glue_version: "3.0"
    )
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not create job #{name}: \n#{e.message}")
    raise
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API [CreateJob](#) 참조를 참조하십시오.

Rust

SDK for Rust

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
let create_job = glue
    .create_job()
    .name(self.job())
    .role(self.iam_role.expose_secret())
    .command(
        JobCommand::builder()
            .name("glueetl")
            .python_version("3")
            .script_location(format!("s3://{}/job.py", self.bucket()))
            .build(),
    )
    .glue_version("3.0")
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

let job_name = create_job.name().ok_or_else(|| {
    GlueMvpError::Unknown("Did not get job name after creating
job".into())
})?;
```

- API에 대한 자세한 내용은 Rust용AWS SDK API 레퍼런스를 참조하십시오 [CreateJob](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **DeleteCrawler** CLI와 함께 사용

다음 코드 예제는 DeleteCrawler의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Delete an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteCrawlerAsync(string crawlerName)
{
    var response = await _amazonGlue.DeleteCrawlerAsync(new
DeleteCrawlerRequest { Name = crawlerName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- API 세부 정보는 AWS SDK for .NET API [DeleteCrawler](#)참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region in which the bucket was created
    (overrides config file).
    // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

    Aws::Glue::Model::DeleteCrawlerRequest request;
    request.SetName(crawler);

    Aws::Glue::Model::DeleteCrawlerOutcome outcome =
    client.DeleteCrawler(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted the crawler." << std::endl;
    }
    else {
        std::cerr << "Error deleting the crawler. "
            << outcome.GetError().GetMessage() << std::endl;
        result = false;
    }
}

```

- API 세부 정보는 AWS SDK for C++ API [DeleteCrawler](#) 참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

const deleteCrawler = (crawlerName) => {
    const client = new GlueClient({});

    const command = new DeleteCrawlerCommand({
        Name: crawlerName,
    });
}

```

```
return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [DeleteCrawler](#)참조를 참조하십시오.

PHP

SDK for PHP

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
echo "Delete the crawler.\n";
$glueClient->deleteCrawler([
    'Name' => $crawlerName,
]);

public function deleteCrawler($crawlerName)
{
    return $this->glueClient->deleteCrawler([
        'Name' => $crawlerName,
    ]);
}
```

- API 세부 정보는 AWS SDK for PHP API [DeleteCrawler](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def delete_crawler(self, name):
        """
        Deletes a crawler.

        :param name: The name of the crawler to delete.
        """
        try:
            self.glue_client.delete_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't delete crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [DeleteCrawler](#).

Ruby

SDK for Ruby

Note

자세한 내용은 다음과 같습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a crawler with the specified name.
  #
  # @param name [String] The name of the crawler to delete.
  # @return [void]
  def delete_crawler(name)
    @glue_client.delete_crawler(name: name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete crawler #{name}: \n#{e.message}")
    raise
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API [DeleteCrawler](#)참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
glue.delete_crawler()
    .name(self.crawler())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- API에 대한 자세한 내용은 Rust용AWS SDK API 레퍼런스를 참조하십시오 [DeleteCrawler](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **DeleteDatabase** CLI와 함께 사용

다음 코드 예제는 DeleteDatabase의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

/// <summary>
/// Delete the AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteDatabaseAsync(string dbName)
{
    var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

```

- API 세부 정보는 AWS SDK for .NET API [DeleteDatabase](#)참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::DeleteDatabaseRequest request;
request.SetName(database);

Aws::Glue::Model::DeleteDatabaseOutcome outcome = client.DeleteDatabase(
    request);

if (outcome.IsSuccess()) {
    std::cout << "Successfully deleted the database." << std::endl;
}

```

```
    }  
    else {  
        std::cerr << "Error deleting database. " <<  
outcome.GetError().GetMessage()  
        << std::endl;  
        result = false;  
    }  
}
```

- API 세부 정보는 AWS SDK for C++ API [DeleteDatabase](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
const deleteDatabase = (databaseName) => {  
    const client = new GlueClient({});  
  
    const command = new DeleteDatabaseCommand({  
        Name: databaseName,  
    });  
  
    return client.send(command);  
};
```

- API 세부 정보는 AWS SDK for JavaScript API [DeleteDatabase](#)참조를 참조하십시오.

PHP

SDK for PHP

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.


```
echo "Delete the databases.\n";
$glueClient->deleteDatabase([
    'Name' => $databaseName,
]);

public function deleteDatabase($databaseName)
{
    return $this->glueClient->deleteDatabase([
        'Name' => $databaseName,
    ]);
}
```

- API 세부 정보는 AWS SDK for PHP API [DeleteDatabase](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
```

```
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

def delete_database(self, name):
    """
    Deletes a metadata database from your Data Catalog.

    :param name: The name of the database to delete.
    """
    try:
        self.glue_client.delete_database(Name=name)
    except ClientError as err:
        logger.error(
            "Couldn't delete database %s. Here's why: %s: %s",
            name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [DeleteDatabase](#).

Ruby

SDK for Ruby

Note

자세한 내용은 다음과 같습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
a simplified interface for common operations.
```

```
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Removes a specified database from a Data Catalog.
  #
  # @param database_name [String] The name of the database to delete.
  # @return [void]
  def delete_database(database_name)
    @glue_client.delete_database(name: database_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete database: \n#{e.message}")
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API [DeleteDatabase](#)참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
glue.delete_database()
    .name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- API에 대한 자세한 내용은 Rust용AWS SDK API 레퍼런스를 참조하십시오 [DeleteDatabase](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **DeleteJob** CLI와 함께 사용

다음 코드 예제는 DeleteJob의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
/// <summary>
/// Delete an AWS Glue job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteJobAsync(string jobName)
{
    var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
    { JobName = jobName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- API 세부 정보는 AWS SDK for .NET API [DeleteJob](#)참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::DeleteJobRequest request;
request.SetJobName(job);

Aws::Glue::Model::DeleteJobOutcome outcome = client.DeleteJob(request);

if (outcome.IsSuccess()) {
    std::cout << "Successfully deleted the job." << std::endl;
}
else {
    std::cerr << "Error deleting the job. " <<
outcome.GetError().GetMessage()
        << std::endl;
    result = false;
}
```

- API 세부 정보는 AWS SDK for C++ API [DeleteJob](#)참조를 참조하십시오.

CLI

AWS CLI

작업을 삭제하려면

다음 delete-job 예제에서는 더 이상 필요하지 않은 작업을 삭제합니다.

```
aws glue delete-job \  
  --job-name my-testing-job
```

출력:

```
{  
  "JobName": "my-testing-job"  
}
```

자세한 내용은 Glue 개발자 안내서의 AWS [Glue 콘솔에서의 작업](#) 사용을 참조하십시오. AWS

- API에 대한 자세한 내용은 AWS CLI 명령 참조를 참조하십시오 [DeleteJob](#).

JavaScript

JavaScript (v3) 용 SDK

Note


더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const deleteJob = (jobName) => {  
  const client = new GlueClient({});  
  
  const command = new DeleteJobCommand({  
    JobName: jobName,  
  });  
  
  return client.send(command);  
};
```

- API 세부 정보는 AWS SDK for JavaScript API [DeleteJob](#) 참조를 참조하십시오.

PHP

SDK for PHP

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.


```
echo "Delete the job.\n";
$glueClient->deleteJob([
    'JobName' => $job['Name'],
]);

public function deleteJob($jobName)
{
    return $this->glueClient->deleteJob([
        'JobName' => $jobName,
    ]);
}
```

- API 세부 정보는 AWS SDK for PHP API [DeleteJob](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
```

```

        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def delete_job(self, job_name):
        """
        Deletes a job definition. This also deletes data about all runs that are
        associated with this job definition.

        :param job_name: The name of the job definition to delete.
        """
        try:
            self.glue_client.delete_job(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete job %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [DeleteJob](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
a simplified interface for common operations.

```

```
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a job with the specified name.
  #
  # @param job_name [String] The name of the job to delete.
  # @return [void]
  def delete_job(job_name)
    @glue_client.delete_job(job_name: job_name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not delete job: \n#{e.message}")
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API [DeleteJob](#) 참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
glue.delete_job()
  .job_name(self.job())
  .send()
  .await
  .map_err(GlueMvpError::from_glue_sdk)?;
```

- API에 대한 자세한 내용은 Rust용 AWS SDK API 레퍼런스를 참조하십시오 [DeleteJob](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **DeleteTable** CLI와 함께 사용

다음 코드 예제는 DeleteTable의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Delete a table from an AWS Glue database.
/// </summary>
/// <param name="tableName">The table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTableAsync(string dbName, string tableName)
{
    var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
    { Name = tableName, DatabaseName = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- API 세부 정보는 AWS SDK for .NET API [DeleteTable](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [DeleteTable](#)참조를 참조하십시오.

PHP

SDK for PHP

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
echo "Delete the tables.\n";
foreach ($tables['TableList'] as $table) {
    $glueService->deleteTable($table['Name'], $databaseName);
}

public function deleteTable($tableName, $databaseName)
```



```

{
    return $this->glueClient->deleteTable([
        'DatabaseName' => $databaseName,
        'Name' => $tableName,
    ]);
}

```

- API 세부 정보는 AWS SDK for PHP API [DeleteTable](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def delete_table(self, db_name, table_name):
        """
        Deletes a table from a metadata database.

        :param db_name: The name of the database that contains the table.
        :param table_name: The name of the table to delete.
        """
        try:
            self.glue_client.delete_table(DatabaseName=db_name, Name=table_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",

```

```

        table_name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise

```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [DeleteTable](#).

Ruby

SDK for Ruby

Note

자세한 내용은 다음과 같습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
# a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Deletes a table with the specified name.
  #
  # @param database_name [String] The name of the catalog database in which the
  # table resides.
  # @param table_name [String] The name of the table to be deleted.
  # @return [void]
  def delete_table(database_name, table_name)

```

```
@glue_client.delete_table(database_name: database_name, name: table_name)
rescue Aws::Glue::Errors::ServiceError => e
  @logger.error("Glue could not delete job: \n#{e.message}")
end
```

- API 세부 정보는 AWS SDK for Ruby API [DeleteTable](#) 참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
for t in &self.tables {
  glue.delete_table()
    .name(t.name())
    .database_name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
}
```

- API에 대한 자세한 내용은 Rust용 AWS SDK API 레퍼런스를 참조하십시오 [DeleteTable](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetCrawler** CLI와 함께 사용

다음 코드 예제는 GetCrawler의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
/// <summary>
/// Get information about an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Crawler object describing the crawler.</returns>
public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new GetCrawlerRequest
    {
        Name = crawlerName,
    };


    var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
    if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
    {
        var databaseName = response.Crawler.DatabaseName;
        Console.WriteLine($"{crawlerName} has the database {databaseName}");
        return response.Crawler;
    }

    Console.WriteLine($"No information regarding {crawlerName} could be
found.");
    return null;
}
```

- API 세부 정보는 AWS SDK for .NET API [GetCrawler](#)참조를 참조하십시오.

C++

SDK for C++

 Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::GetCrawlerRequest request;
request.SetName(CRAWLER_NAME);

Aws::Glue::Model::GetCrawlerOutcome outcome = client.GetCrawler(request);

if (outcome.IsSuccess()) {
    Aws::Glue::Model::CrawlerState crawlerState =
outcome.GetResult().GetCrawler().GetState();
    std::cout << "Retrieved crawler with state " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
        crawlerState)
    << "." << std::endl;
}
else {
    std::cerr << "Error retrieving a crawler. "
    << outcome.GetError().GetMessage() << std::endl;
    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
        clientConfig);
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API [GetCrawler](#)참조를 참조하십시오.

Java

SDK for Java 2.x

 Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.GetCrawlerRequest;
import software.amazon.awssdk.services.glue.model.GetCrawlerResponse;
import software.amazon.awssdk.services.glue.model.GlueException;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 */
public class GetCrawler {
    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <crawlerName>

            Where:
                crawlerName - The name of the crawler.\s
            """;

        if (args.length != 1) {
```

```
        System.out.println(usage);
        System.exit(1);
    }

    String crawlerName = args[0];
    Region region = Region.US_EAST_1;
    GlueClient glueClient = GlueClient.builder()
        .region(region)
        .build();

    getSpecificCrawler(glueClient, crawlerName);
    glueClient.close();
}

public static void getSpecificCrawler(GlueClient glueClient, String
crawlerName) {
    try {
        GetCrawlerRequest crawlerRequest = GetCrawlerRequest.builder()
            .name(crawlerName)
            .build();

        GetCrawlerResponse response = glueClient.getCrawler(crawlerRequest);
        Instant createDate = response.crawler().creationTime();

        // Convert the Instant to readable date
        DateTimeFormatter formatter =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
            .withLocale(Locale.US)
            .withZone(ZoneId.systemDefault());

        formatter.format(createDate);
        System.out.println("The create date of the Crawler is " +
createDate);

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- API 세부 정보는 AWS SDK for Java 2.x API [GetCrawler](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [GetCrawler](#)참조를 참조하십시오.

Kotlin

SDK for Kotlin

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun getSpecificCrawler(crawlerName: String?) {
  val request =
    GetCrawlerRequest {
      name = crawlerName
    }
  GlueClient { region = "us-east-1" }.use { glueClient ->
    val response = glueClient.getCrawler(request)
  }
}
```



```

        val role = response.crawler?.role
        println("The role associated with this crawler is $role")
    }
}

```

- API 세부 정보는 Kotlin API용 AWS SDK 레퍼런스를 참조하세요 [GetCrawler](#).

PHP

SDK for PHP

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

echo "Waiting for crawler";
do {
    $crawler = $glueService->getCrawler($crawlerName);
    echo ".";
    sleep(10);
} while ($crawler['Crawler']['State'] != "READY");
echo "\n";


public function getCrawler($crawlerName)
{
    return $this->customWaiter(function () use ($crawlerName) {
        return $this->glueClient->getCrawler([
            'Name' => $crawlerName,
        ]);
    });
}

```

- API 세부 정보는 AWS SDK for PHP API [GetCrawler](#) 참조를 참조하십시오.

Python

SDK for Python(Boto3)

 Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def get_crawler(self, name):
        """
        Gets information about a crawler.

        :param name: The name of the crawler to look up.
        :return: Data about the crawler.
        """
        crawler = None
        try:
            response = self.glue_client.get_crawler(Name=name)
            crawler = response["Crawler"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "EntityNotFoundException":
                logger.info("Crawler %s doesn't exist.", name)
            else:
                logger.error(
                    "Couldn't get crawler %s. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

```
return crawler
```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [GetCrawler](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
# a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific crawler.
  #
  # @param name [String] The name of the crawler to retrieve information about.
  # @return [Aws::Glue::Types::Crawler, nil] The crawler object if found, or nil
  # if not found.
  def get_crawler(name)
    @glue_client.get_crawler(name: name)
  rescue Aws::Glue::Errors::EntityNotFoundException
    @logger.info("Crawler #{name} doesn't exist.")
    false
  rescue Aws::Glue::Errors::GlueException => e
```

```
@logger.error("Glue could not get crawler #{name}: \n#{e.message}")
raise
end
```

- API 세부 정보는 AWS SDK for Ruby API [GetCrawler](#) 참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
let tmp_crawler = glue
    .get_crawler()
    .name(self.crawler())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- API에 대한 자세한 내용은 Rust용 AWS SDK API 레퍼런스를 참조하십시오 [GetCrawler](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetDatabase** CLI와 함께 사용

다음 코드 예제는 GetDatabase의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
/// <summary>
/// Get information about an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Database object containing information about the database.</
returns>
public async Task<Database> GetDatabaseAsync(string dbName)
{
    var databasesRequest = new GetDatabaseRequest
    {
        Name = dbName,
    };

    var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
    return response.Database;
}
```

- API 세부 정보는 AWS SDK for .NET API [GetDatabase](#)참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region in which the bucket was created
    (overrides config file).
    // clientConfig.region = "us-east-1";

    Aws::Glue::GlueClient client(clientConfig);

    Aws::Glue::Model::GetDatabaseRequest request;
    request.SetName(CRAWLER_DATABASE_NAME);

    Aws::Glue::Model::GetDatabaseOutcome outcome =
    client.GetDatabase(request);

    if (outcome.IsSuccess()) {
        const Aws::Glue::Model::Database &database =
    outcome.GetResult().GetDatabase();

        std::cout << "Successfully retrieve the database\n" <<
            database.Jsonize().View().WriteReadable() << ". " <<
    std::endl;
    }
    else {
        std::cerr << "Error getting the database. "
            << outcome.GetError().GetMessage() << std::endl;
        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
            clientConfig);
        return false;
    }
}

```

- API 세부 정보는 AWS SDK for C++ API [GetDatabase](#) 참조를 참조하십시오.

Java

SDK for Java 2.x

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.GetDatabaseRequest;
import software.amazon.awssdk.services.glue.model.GetDatabaseResponse;
import software.amazon.awssdk.services.glue.model.GlueException;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class GetDatabase {
    public static void main(String[] args) {
        final String usage = ""

                Usage:
                <databaseName>

                Where:
                databaseName - The name of the database.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String databaseName = args[0];
        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
            .region(region)
            .build();

        getSpecificDatabase(glueClient, databaseName);
    }
}
```

```
        glueClient.close();
    }

    public static void getSpecificDatabase(GlueClient glueClient, String
databaseName) {
        try {
            GetDatabaseRequest databasesRequest = GetDatabaseRequest.builder()
                .name(databaseName)
                .build();

            GetDatabaseResponse response =
glueClient.getDatabase(databasesRequest);
            Instant createDate = response.database().createTime();

            // Convert the Instant to readable date.
            DateTimeFormatter formatter =
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                .withLocale(Locale.US)
                .withZone(ZoneId.systemDefault());

            formatter.format(createDate);
            System.out.println("The create date of the database is " +
createDate);

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API [GetDatabase](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.


```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [GetDatabase](#) 참조를 참조하십시오.

Kotlin

SDK for Kotlin

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun getSpecificDatabase(databaseName: String?) {
  val request =
    GetDatabaseRequest {
      name = databaseName
    }

  GlueClient { region = "us-east-1" }.use { glueClient ->
    val response = glueClient.getDatabase(request)
    val dbDesc = response.database?.description
    println("The database description is $dbDesc")
  }
}
```

- API 세부 정보는 Kotlin API용 AWS SDK 레퍼런스를 참조하세요 [GetDatabase](#).

PHP

SDK for PHP

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
$databaseName = "doc-example-database-$uniqid";

$database = $glueService->getDatabase($databaseName);
echo "Found a database named " . $database['Database']['Name'] . "\n";

public function getDatabase(string $databaseName): Result
{
    return $this->customWaiter(function () use ($databaseName) {
        return $this->glueClient->getDatabase([
            'Name' => $databaseName,
        ]);
    });
}
```

- API 세부 정보는 AWS SDK for PHP API [GetDatabase](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""
```

```
def __init__(self, glue_client):
    """
    :param glue_client: A Boto3 Glue client.
    """
    self.glue_client = glue_client

def get_database(self, name):
    """
    Gets information about a database in your Data Catalog.

    :param name: The name of the database to look up.
    :return: Information about the database.
    """
    try:
        response = self.glue_client.get_database(Name=name)
    except ClientError as err:
        logger.error(
            "Couldn't get database %s. Here's why: %s: %s",
            name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return response["Database"]
```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [GetDatabase](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
  a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
  for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
  calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific database.
  #
  # @param name [String] The name of the database to retrieve information about.
  # @return [Aws::Glue::Types::Database, nil] The database object if found, or
  nil if not found.
  def get_database(name)
    response = @glue_client.get_database(name: name)
    response.database
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get database #{name}: \n#{e.message}")
    raise
  end
end

```

- API 세부 정보는 AWS SDK for Ruby API [GetDatabase](#)참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

let database = glue
    .get_database()

```

```

        .name(self.database())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?
        .to_owned();
    let database = database
        .database()
        .ok_or_else(|| GlueMvpError::Unknown("Could not find
database".into()))?;

```

- API에 대한 자세한 내용은 Rust용 AWS SDK API 레퍼런스를 참조하십시오 [GetDatabase](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetDatabases** CLI와 함께 사용

다음 코드 예제는 GetDatabases의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

CLI

AWS CLI

AWS Glue 데이터 카탈로그의 일부 또는 모든 데이터베이스의 정의를 나열하려면

다음 `get-databases` 예제는 데이터 카탈로그의 데이터베이스에 대한 정보를 반환합니다.

```
aws glue get-databases
```

출력:

```
{
  "DatabaseList": [
    {
      "Name": "default",

```

```
"Description": "Default Hive database",
"LocationUri": "file:/spark-warehouse",
"CreateTime": 1602084052.0,
"CreateTableDefaultPermissions": [
  {
    "Principal": {
      "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
    },
    "Permissions": [
      "ALL"
    ]
  }
],
"CatalogId": "111122223333"
},
{
  "Name": "flights-db",
  "CreateTime": 1587072847.0,
  "CreateTableDefaultPermissions": [
    {
      "Principal": {
        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
      },
      "Permissions": [
        "ALL"
      ]
    }
  ],
  "CatalogId": "111122223333"
},
{
  "Name": "legislators",
  "CreateTime": 1601415625.0,
  "CreateTableDefaultPermissions": [
    {
      "Principal": {
        "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
      },
      "Permissions": [
        "ALL"
      ]
    }
  ],
  "CatalogId": "111122223333"
```

```

    },
    {
      "Name": "tempdb",
      "CreateTime": 1601498566.0,
      "CreateTableDefaultPermissions": [
        {
          "Principal": {
            "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
          },
          "Permissions": [
            "ALL"
          ]
        }
      ],
      "CatalogId": "111122223333"
    }
  ]
}

```

자세한 내용은 AWS Glue 개발자 가이드의 [데이터 카탈로그에서 데이터베이스 정의](#)를 참조하세요.

- API에 대한 자세한 내용은 AWS CLI 명령 참조를 참조하십시오 [GetDatabases](#).

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

const getDatabases = () => {
  const client = new GlueClient({});

  const command = new GetDatabasesCommand({});

  return client.send(command);
};

```

- API 세부 정보는 AWS SDK for JavaScript API [GetDatabases](#) 참조를 참조하십시오.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 [을 참조하십시오](#) [AWS SDK와 함께 이 서비스 사용](#). 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetJob** CLI와 함께 사용

다음 코드 예제는 GetJob의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

CLI

AWS CLI

작업 정보를 검색하려면

다음 `get-job` 예제는 작업 정보를 검색합니다.

```
aws glue get-job \  
  --job-name my-testing-job
```

출력:

```
{  
  "Job": {  
    "Name": "my-testing-job",  
    "Role": "Glue_DefaultRole",  
    "CreatedOn": 1602805698.167,  
    "LastModifiedOn": 1602805698.167,  
    "ExecutionProperty": {  
      "MaxConcurrentRuns": 1  
    },  
    "Command": {  
      "Name": "gluestreaming",
```



```

        "ScriptLocation": "s3://janetst-bucket-01/Scripts/test_script.scala",
        "PythonVersion": "2"
    },
    "DefaultArguments": {
        "--class": "GlueApp",
        "--job-language": "scala"
    },
    "MaxRetries": 0,
    "AllocatedCapacity": 10,
    "MaxCapacity": 10.0,
    "GlueVersion": "1.0"
}
}

```

자세한 내용은 AWS Glue 개발자 안내서의 [작업](#)을 참조하세요.

- API 세부 정보는 AWS CLI 명령 [GetJob](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

const getJob = (jobName) => {
    const client = new GlueClient({});

    const command = new GetJobCommand({
        JobName: jobName,
    });

    return client.send(command);
};

```

- API 세부 정보는 AWS SDK for JavaScript API [GetJob](#)참조를 참조하십시오.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 [여기](#)를 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#). 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetJobRun** CLI와 함께 사용

다음 코드 예제는 GetJobRun의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Get information about a specific AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <param name="jobRunId">The Id of the job run.</param>
/// <returns>A JobRun object with information about the job run.</returns>
public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
{
    var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
    { JobName = jobName, RunId = jobRunId });
    return response.JobRun;
}
```

- API 세부 정보는 AWS SDK for .NET API [GetJobRun](#)참조를 참조하십시오.

C++

SDK for C++

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
// (overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::GetJobRunRequest jobRunRequest;
jobRunRequest.SetJobName(jobName);
jobRunRequest.SetRunId(jobRunID);

Aws::Glue::Model::GetJobRunOutcome jobRunOutcome = client.GetJobRun(
    jobRunRequest);

if (jobRunOutcome.IsSuccess()) {
    std::cout << "Displaying the job run JSON description." << std::endl;
    std::cout
        <<
jobRunOutcome.GetResult().GetJobRun().Jsonize().View().WriteReadable()
        << std::endl;
}
else {
    std::cerr << "Error get a job run. "
        << jobRunOutcome.GetError().GetMessage()
        << std::endl;
}
```

- API 세부 정보는 AWS SDK for C++ API [GetJobRun](#)참조를 참조하십시오.

CLI

AWS CLI

작업 실행 정보를 가져오려면

다음 `get-job-run` 예제는 작업 실행 정보를 검색합니다.

```
aws glue get-job-run \  
  --job-name "Combine legislators data" \  
  --run-id jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e
```

출력:

```
{  
  "JobRun": {  
    "Id":  
    "jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e",  
    "Attempt": 0,  
    "JobName": "Combine legislators data",  
    "StartedOn": 1602873931.255,  
    "LastModifiedOn": 1602874075.985,  
    "CompletedOn": 1602874075.985,  
    "JobRunState": "SUCCEEDED",  
    "Arguments": {  
      "--enable-continuous-cloudwatch-log": "true",  
      "--enable-metrics": "",  
      "--enable-spark-ui": "true",  
      "--job-bookmark-option": "job-bookmark-enable",  
      "--spark-event-logs-path": "s3://aws-glue-assets-111122223333-us-east-1/sparkHistoryLogs/"  
    },  
    "PredecessorRuns": [],  
    "AllocatedCapacity": 10,  
    "ExecutionTime": 117,  
    "Timeout": 2880,  
    "MaxCapacity": 10.0,  
    "WorkerType": "G.1X",  
    "NumberOfWorkers": 10,  
    "LogGroupName": "/aws-glue/jobs",  
    "GlueVersion": "2.0"  
  }  
}
```

자세한 내용은 AWS Glue 개발자 안내서의 [작업 실행](#)을 참조하세요.

- API 세부 정보는 AWS CLI 명령 [GetJobRun](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [GetJobRun](#)참조를 참조하십시오.

PHP

SDK for PHP

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
$jobName = 'test-job-' . $uniqid;

$outputBucketUrl = "s3://$bucketName";
```

```

    $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl)['JobRunId'];

    echo "waiting for job";
    do {
        $jobRun = $glueService->getJobRun($jobName, $runId);
        echo ".";
        sleep(10);
    } while (!array_intersect([$jobRun['JobRun']['JobRunState']],
['SUCCEEDED', 'STOPPED', 'FAILED', 'TIMEOUT']));
    echo "\n";

    public function getJobRun($jobName, $runId, $predecessorsIncluded = false):
Result
    {
        return $this->glueClient->getJobRun([
            'JobName' => $jobName,
            'RunId' => $runId,
            'PredecessorsIncluded' => $predecessorsIncluded,
        ]);
    }

```

- API 세부 정보는 AWS SDK for PHP API [GetJobRun](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """

```

```
self.glue_client = glue_client

def get_job_run(self, name, run_id):
    """
    Gets information about a single job run.

    :param name: The name of the job definition for the run.
    :param run_id: The ID of the run.
    :return: Information about the run.
    """
    try:
        response = self.glue_client.get_job_run(JobName=name, RunId=run_id)
    except ClientError as err:
        logger.error(
            "Couldn't get job run %s/%s. Here's why: %s: %s",
            name,
            run_id,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return response["JobRun"]
```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [GetJobRun](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
  a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
  for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
  calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves data for a specific job run.
  #
  # @param job_name [String] The name of the job run to retrieve data for.
  # @return [Glue::Types::GetJobRunResponse]
  def get_job_run(job_name, run_id)
    @glue_client.get_job_run(job_name: job_name, run_id: run_id)
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API [GetJobRun](#)참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
let get_job_run = || async {
  Ok:::<JobRun, GlueMvpError>(
    glue.get_job_run()
      .job_name(self.job())
      .run_id(job_run_id.to_string())
      .send()
      .await
  )
}
```



```

        .map_err(GlueMvpError::from_glue_sdk)?
        .job_run()
        .ok_or_else(|| GlueMvpError::Unknown("Failed to get
job_run".into()))?
        .to_owned(),
    )
};

let mut job_run = get_job_run().await?;
let mut state =
job_run.job_run_state().unwrap_or(&unknown_state).to_owned();

while matches!(
    state,
    JobRunState::Starting | JobRunState::Stopping | JobRunState::Running
) {
    info!(?state, "Waiting for job to finish");
    tokio::time::sleep(self.wait_delay).await;

    job_run = get_job_run().await?;
    state = job_run.job_run_state().unwrap_or(&unknown_state).to_owned();
}

```

- API에 대한 자세한 내용은 Rust용AWS SDK API 레퍼런스를 참조하십시오 [GetJobRun](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetJobRuns** CLI와 함께 사용

다음 코드 예제는 GetJobRuns의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
/// <summary>
/// Get information about all AWS Glue runs of a specific job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A list of JobRun objects.</returns>
public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
{
    var jobRuns = new List<JobRun>();

    var request = new GetJobRunsRequest
    {
        JobName = jobName,
    };

    // No need to loop to get all the log groups--the SDK does it for us
    behind the scenes
    var paginatorForJobRuns =
        _amazonGlue.Paginators.GetJobRuns(request);

    await foreach (var response in paginatorForJobRuns.Responses)
    {
        response.JobRuns.ForEach(jobRun =>
        {
            jobRuns.Add(jobRun);
        });
    }

    return jobRuns;
}
```

- API 세부 정보는 AWS SDK for .NET API [GetJobRuns](#) 참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::GetJobRunsRequest getJobRunsRequest;
getJobRunsRequest.SetJobName(jobName);

Aws::String nextToken; // Used for pagination.
std::vector<Aws::Glue::Model::JobRun> allJobRuns;
do {
    if (!nextToken.empty()) {
        getJobRunsRequest.SetNextToken(nextToken);
    }
    Aws::Glue::Model::GetJobRunsOutcome jobRunsOutcome =
client.GetJobRuns(
    getJobRunsRequest);

    if (jobRunsOutcome.IsSuccess()) {
        const std::vector<Aws::Glue::Model::JobRun> &jobRuns =
jobRunsOutcome.GetResult().GetJobRuns();
        allJobRuns.insert(allJobRuns.end(), jobRuns.begin(),
jobRuns.end());

        nextToken = jobRunsOutcome.GetResult().GetNextToken();
    }
    else {
        std::cerr << "Error getting job runs. "
```

```

        << jobRunsOutcome.GetError().GetMessage()
        << std::endl;
    }
    break;
}
} while (!nextToken.empty());

```

- API 세부 정보는 AWS SDK for C++ API [GetJobRuns](#) 참조를 참조하십시오.

CLI

AWS CLI

작업에 대한 모든 작업 실행 정보를 가져오려면

다음 `get-job-runs` 예제는 작업에 대한 작업 실행 정보를 검색합니다.

```
aws glue get-job-runs \
  --job-name "my-testing-job"
```

출력:

```
{
  "JobRuns": [
    {
      "Id":
      "jr_012e176506505074d94d761755e5c62538ee1aad6f17d39f527e9140cf0c9a5e",
      "Attempt": 0,
      "JobName": "my-testing-job",
      "StartedOn": 1602873931.255,
      "LastModifiedOn": 1602874075.985,
      "CompletedOn": 1602874075.985,
      "JobRunState": "SUCCEEDED",
      "Arguments": {
        "--enable-continuous-cloudwatch-log": "true",
        "--enable-metrics": "",
        "--enable-spark-ui": "true",
        "--job-bookmark-option": "job-bookmark-enable",
        "--spark-event-logs-path": "s3://aws-glue-assets-111122223333-us-
east-1/sparkHistoryLogs/"
      },
      "PredecessorRuns": [],
      "AllocatedCapacity": 10,
    }
  ]
}
```

```

        "ExecutionTime": 117,
        "Timeout": 2880,
        "MaxCapacity": 10.0,
        "WorkerType": "G.1X",
        "NumberOfWorkers": 10,
        "LogGroupName": "/aws-glue/jobs",
        "GlueVersion": "2.0"
    },
    {
        "Id":
"jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_2",
        "Attempt": 2,
        "PreviousRunId":
"jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_1",
        "JobName": "my-testing-job",
        "StartedOn": 1602811168.496,
        "LastModifiedOn": 1602811282.39,
        "CompletedOn": 1602811282.39,
        "JobRunState": "FAILED",
        "ErrorMessage": "An error occurred while calling
o122.pyWriteDynamicFrame.
                Access Denied (Service: Amazon S3; Status Code: 403; Error Code:
AccessDenied;
                Request ID: 021AAB703DB20A2D;
                S3 Extended Request ID: teZk24Y09TkXzBvMPG502L5VJBhe9DJuWA9/
TXtuG0qfByajkfl/Tlqt5JBGdEGpigAqzdMDM/U=)",
        "PredecessorRuns": [],
        "AllocatedCapacity": 10,
        "ExecutionTime": 110,
        "Timeout": 2880,
        "MaxCapacity": 10.0,
        "WorkerType": "G.1X",
        "NumberOfWorkers": 10,
        "LogGroupName": "/aws-glue/jobs",
        "GlueVersion": "2.0"
    },
    {
        "Id":
"jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f_attempt_1",
        "Attempt": 1,
        "PreviousRunId":
"jr_03cc19ddab11c4e244d3f735567de74ff93b0b3ef468a713ffe73e53d1aec08f",
        "JobName": "my-testing-job",
        "StartedOn": 1602811020.518,

```

```

        "LastModifiedOn": 1602811138.364,
        "CompletedOn": 1602811138.364,
        "JobRunState": "FAILED",
        "ErrorMessage": "An error occurred while calling
o122.pyWriteDynamicFrame.
                Access Denied (Service: Amazon S3; Status Code: 403; Error Code:
AccessDenied;
                Request ID: 2671D37856AE7ABB;
                S3 Extended Request ID: RLJCJw20brV
+PpC6Gp0RahyF2fp9f1B5SSb2bTGPnUSPVizLXR11PN3QZ1db+v1o9qRVktNYbW8=)",
        "PredecessorRuns": [],
        "AllocatedCapacity": 10,
        "ExecutionTime": 113,
        "Timeout": 2880,
        "MaxCapacity": 10.0,
        "WorkerType": "G.1X",
        "NumberOfWorkers": 10,
        "LogGroupName": "/aws-glue/jobs",
        "GlueVersion": "2.0"
    }
]
}

```

자세한 내용은 AWS Glue 개발자 안내서의 [작업 실행](#)을 참조하세요.

- API 세부 정보는 AWS CLI 명령 [GetJobRuns](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });
};

```

```
return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [GetJobRuns](#)참조를 참조하십시오.

PHP

SDK for PHP

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
$jobName = 'test-job-' . $uniqid;


$jobRuns = $glueService->getJobRuns($jobName);

public function getJobRuns($jobName, $maxResults = 0, $nextToken = ''):
Result
{
    $arguments = ['JobName' => $jobName];
    if ($maxResults) {
        $arguments['MaxResults'] = $maxResults;
    }
    if ($nextToken) {
        $arguments['NextToken'] = $nextToken;
    }
    return $this->glueClient->getJobRuns($arguments);
}
```

- API 세부 정보는 AWS SDK for PHP API [GetJobRuns](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

 Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def get_job_runs(self, job_name):
        """
        Gets information about runs that have been performed for a specific job
        definition.

        :param job_name: The name of the job definition to look up.
        :return: The list of job runs.
        """
        try:
            response = self.glue_client.get_job_runs(JobName=job_name)
        except ClientError as err:
            logger.error(
                "Couldn't get job runs for %s. Here's why: %s: %s",
                job_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["JobRuns"]
```


- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [GetJobRuns](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves a list of job runs for the specified job.
  #
  # @param job_name [String] The name of the job to retrieve job runs for.
  # @return [Array<Aws::Glue::Types::JobRun>]
  def get_job_runs(job_name)
    response = @glue_client.get_job_runs(job_name: job_name)
    response.job_runs
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get job runs: \n#{e.message}")
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API [GetJobRuns](#)참조를 참조하십시오.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 [을 참조하십시오](#) [AWS SDK와 함께 이 서비스 사용](#). 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetTables** CLI와 함께 사용

다음 코드 예제는 GetTables의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Get a list of tables for an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A list of Table objects.</returns>
public async Task<List<Table>> GetTablesAsync(string dbName)
{
    var request = new GetTablesRequest { DatabaseName = dbName };
    var tables = new List<Table>();

    // Get a paginator for listing the tables.
    var tablePaginator = _amazonGlue.Paginators.GetTables(request);

    await foreach (var response in tablePaginator.Responses)
    {
        tables.AddRange(response.TableList);
    }

    return tables;
}
```

```
}

```

- API 세부 정보는 AWS SDK for .NET API [GetTables](#) 참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::GetTablesRequest request;
request.SetDatabaseName(CRAWLER_DATABASE_NAME);
std::vector<Aws::Glue::Model::Table> all_tables;
Aws::String nextToken; // Used for pagination.
do {
    Aws::Glue::Model::GetTablesOutcome outcome =
client.GetTables(request);

    if (outcome.IsSuccess()) {
        const std::vector<Aws::Glue::Model::Table> &tables =
outcome.GetResult().GetTableList();
        all_tables.insert(all_tables.end(), tables.begin(),
tables.end());
        nextToken = outcome.GetResult().GetNextToken();
    }
    else {
        std::cerr << "Error getting the tables. "
        << outcome.GetError().GetMessage()
        << std::endl;
    }
}
```

```

        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                    clientConfig);
        return false;
    }
} while (!nextToken.empty());

std::cout << "The database contains " << all_tables.size()
          << (all_tables.size() == 1 ?
            " table." : "tables.") << std::endl;
std::cout << "Here is a list of the tables in the database.";
for (size_t index = 0; index < all_tables.size(); ++index) {
    std::cout << "    " << index + 1 << ": " <<
all_tables[index].GetName()
          << std::endl;
}

if (!all_tables.empty()) {
    int tableIndex = askQuestionForIntRange(
        "Enter an index to display the database detail ",
        1, static_cast<int>(all_tables.size()));
    std::cout << all_tables[tableIndex -
1].Jsonize().View().WriteReadable()
          << std::endl;

    tableName = all_tables[tableIndex - 1].GetName();
}

```

- API 세부 정보는 AWS SDK for C++ API [GetTables](#)참조를 참조하십시오.

CLI

AWS CLI

지정된 데이터베이스의 일부 또는 모든 테이블의 정의를 나열하려면

다음 `get-tables` 예제는 지정된 데이터베이스의 테이블에 대한 정보를 반환합니다.

```
aws glue get-tables --database-name 'tempdb'
```

출력:

```
{
```

```
"TableList": [  
  {  
    "Name": "my-s3-sink",  
    "DatabaseName": "tempdb",  
    "CreateTime": 1602730539.0,  
    "UpdateTime": 1602730539.0,  
    "Retention": 0,  
    "StorageDescriptor": {  
      "Columns": [  
        {  
          "Name": "sensorid",  
          "Type": "int"  
        },  
        {  
          "Name": "currenttemperature",  
          "Type": "int"  
        },  
        {  
          "Name": "status",  
          "Type": "string"  
        }  
      ],  
      "Location": "s3://janetst-bucket-01/test-s3-output/",  
      "Compressed": false,  
      "NumberOfBuckets": 0,  
      "SerdeInfo": {  
        "SerializationLibrary": "org.openx.data.jsonserde.JsonSerDe"  
      },  
      "SortColumns": [],  
      "StoredAsSubDirectories": false  
    },  
    "Parameters": {  
      "classification": "json"  
    },  
    "CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",  
    "IsRegisteredWithLakeFormation": false,  
    "CatalogId": "007436865787"  
  },  
  {  
    "Name": "s3-source",  
    "DatabaseName": "tempdb",  
    "CreateTime": 1602730658.0,  
    "UpdateTime": 1602730658.0,  
    "Retention": 0,  
  }  
]
```

```
    "StorageDescriptor": {
      "Columns": [
        {
          "Name": "sensorid",
          "Type": "int"
        },
        {
          "Name": "currenttemperature",
          "Type": "int"
        },
        {
          "Name": "status",
          "Type": "string"
        }
      ],
      "Location": "s3://janetst-bucket-01/",
      "Compressed": false,
      "NumberOfBuckets": 0,
      "SortColumns": [],
      "StoredAsSubDirectories": false
    },
    "Parameters": {
      "classification": "json"
    },
    "CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",
    "IsRegisteredWithLakeFormation": false,
    "CatalogId": "007436865787"
  },
  {
    "Name": "test-kinesis-input",
    "DatabaseName": "tempdb",
    "CreateTime": 1601507001.0,
    "UpdateTime": 1601507001.0,
    "Retention": 0,
    "StorageDescriptor": {
      "Columns": [
        {
          "Name": "sensorid",
          "Type": "int"
        },
        {
          "Name": "currenttemperature",
          "Type": "int"
        }
      ],
```

```

        {
            "Name": "status",
            "Type": "string"
        }
    ],
    "Location": "my-testing-stream",
    "Compressed": false,
    "NumberOfBuckets": 0,
    "SerdeInfo": {
        "SerializationLibrary": "org.openx.data.jsonserde.JsonSerDe"
    },
    "SortColumns": [],
    "Parameters": {
        "kinesisUrl": "https://kinesis.us-east-1.amazonaws.com",
        "streamName": "my-testing-stream",
        "typeOfData": "kinesis"
    },
    "StoredAsSubDirectories": false
},
"Parameters": {
    "classification": "json"
},
"CreatedBy": "arn:aws:iam::007436865787:user/JRSTERN",
"IsRegisteredWithLakeFormation": false,
"CatalogId": "007436865787"
}
]
}

```

자세한 내용은 Glue 개발자 가이드의 AWS [Glue 데이터 카탈로그에서 테이블 정의를](#) 참조하십시오.

- API에 대한 자세한 내용은 AWS CLI 명령 참조를 참조하십시오 [GetTables](#).

Java

SDK for Java 2.x

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.GetTableRequest;
import software.amazon.awssdk.services.glue.model.GetTableResponse;
import software.amazon.awssdk.services.glue.model.GlueException;
import java.time.Instant;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class GetTable {
    public static void main(String[] args) {
        final String usage = ""

                Usage:
                <dbName> <tableName>

                Where:
                dbName - The database name.\s
                tableName - The name of the table.\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String dbName = args[0];
        String tableName = args[1];
        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
            .region(region)
            .build();
    }
}
```



```
        getGlueTable(glueClient, dbName, tableName);
        glueClient.close();
    }

    public static void getGlueTable(GlueClient glueClient, String dbName, String
tableName) {
        try {
            GetTableRequest tableRequest = GetTableRequest.builder()
                .databaseName(dbName)
                .name(tableName)
                .build();

            GetTableResponse tableResponse = glueClient.getTable(tableRequest);
            Instant createDate = tableResponse.table().createTime();

            // Convert the Instant to readable date.
            DateTimeFormatter formatter =
                DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)
                    .withLocale(Locale.US)
                    .withZone(ZoneId.systemDefault());

            formatter.format(createDate);
            System.out.println("The create date of the table is " + createDate);

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API [GetTables](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [GetTables](#) 참조를 참조하십시오.

PHP

SDK for PHP

Note

자세한 내용은 다음과 같습니다. GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
$databaseName = "doc-example-database-$uniqid";

$tables = $glueService->getTables($databaseName);

public function getTables($databaseName): Result
{
    return $this->glueClient->getTables([
```

```

        'DatabaseName' => $databaseName,
    ]);
}

```

- API 세부 정보는 AWS SDK for PHP API [GetTables](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def get_tables(self, db_name):
        """
        Gets a list of tables in a Data Catalog database.

        :param db_name: The name of the database to query.
        :return: The list of tables in the database.
        """
        try:
            response = self.glue_client.get_tables(DatabaseName=db_name)
        except ClientError as err:
            logger.error(
                "Couldn't get tables %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )

```

```

    )
    raise
  else:
    return response["TableList"]

```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [GetTables](#).

Ruby

SDK for Ruby

Note

자세한 내용은 다음과 같습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
# a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves a list of tables in the specified database.
  #
  # @param db_name [String] The name of the database to retrieve tables from.
  # @return [Array<Aws::Glue::Types::Table>]
  def get_tables(db_name)
    response = @glue_client.get_tables(database_name: db_name)
    response.table_list
  rescue Aws::Glue::Errors::GlueException => e

```

```
@logger.error("Glue could not get tables #{db_name}: \n#{e.message}")
raise
end
```

- API 세부 정보는 AWS SDK for Ruby API [GetTables](#)참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
let tables = glue
  .get_tables()
  .database_name(self.database())
  .send()
  .await
  .map_err(GlueMvpError::from_glue_sdk)?;

let tables = tables.table_list();
```

- API에 대한 자세한 내용은 Rust용AWS SDK API 레퍼런스를 참조하십시오 [GetTables](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **ListJobs** CLI와 함께 사용

다음 코드 예제는 ListJobs의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// List AWS Glue jobs using a paginator.
/// </summary>
/// <returns>A list of AWS Glue job names.</returns>
public async Task<List<string>> ListJobsAsync()
{
    var jobNames = new List<string>();

    var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new
ListJobsRequest { MaxResults = 10 });
    await foreach (var response in listJobsPaginator.Responses)
    {
        jobNames.AddRange(response.JobNames);
    }

    return jobNames;
}
```

- API 세부 정보는 AWS SDK for .NET API [ListJobs](#)참조를 참조하십시오.

C++

SDK for C++

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::ListJobsRequest listJobsRequest;

Aws::String nextToken;
std::vector<Aws::String> allJobNames;

do {
    if (!nextToken.empty()) {
        listJobsRequest.SetNextToken(nextToken);
    }
    Aws::Glue::Model::ListJobsOutcome listRunsOutcome = client.ListJobs(
        listJobsRequest);

    if (listRunsOutcome.IsSuccess()) {
        const std::vector<Aws::String> &jobNames =
listRunsOutcome.GetResult().GetJobNames();
        allJobNames.insert(allJobNames.end(), jobNames.begin(),
jobNames.end());
        nextToken = listRunsOutcome.GetResult().GetNextToken();
    }
    else {
        std::cerr << "Error listing jobs. "
            << listRunsOutcome.GetError().GetMessage()
            << std::endl;
    }
} while (!nextToken.empty());
```

- API 세부 정보는 AWS SDK for C++ API [ListJobs](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const listJobs = () => {
  const client = new GlueClient({});

  const command = new ListJobsCommand({});

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [ListJobs](#)참조를 참조하십시오.

PHP

SDK for PHP

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
$jobs = $glueService->listJobs();
echo "Current jobs:\n";
foreach ($jobs['JobNames'] as $jobsName) {
    echo "{$jobsName}\n";
}

public function listJobs($maxResults = null, $nextToken = null, $tags = []):
Result
{
```



```

    $arguments = [];
    if ($maxResults) {
        $arguments['MaxResults'] = $maxResults;
    }
    if ($nextToken) {
        $arguments['NextToken'] = $nextToken;
    }
    if (!empty($tags)) {
        $arguments['Tags'] = $tags;
    }
    return $this->glueClient->listJobs($arguments);
}

```

- API 세부 정보는 AWS SDK for PHP API [ListJobs](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def list_jobs(self):
        """
        Lists the names of job definitions in your account.

        :return: The list of job definition names.
        """

```

```
try:
    response = self.glue_client.list_jobs()
except ClientError as err:
    logger.error(
        "Couldn't list jobs. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return response["JobNames"]
```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [ListJobs](#).

Ruby

SDK for Ruby

Note

자세한 내용은 다음과 같습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
# a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves a list of jobs in AWS Glue.
```

```
#
# @return [Aws::Glue::Types::ListJobsResponse]
def list_jobs
  @glue_client.list_jobs
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not list jobs: \n#{e.message}")
  raise
end
```

- API 세부 정보는 AWS SDK for Ruby API [ListJobs](#) 참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
let mut list_jobs = glue.list_jobs().into_paginator().send();
while let Some(list_jobs_output) = list_jobs.next().await {
  match list_jobs_output {
    Ok(list_jobs) => {
      let names = list_jobs.job_names();
      info!(?names, "Found these jobs")
    }
    Err(err) => return Err(GlueMvpError::from_glue_sdk(err)),
  }
}
```

- API에 대한 자세한 내용은 Rust용 AWS SDK API 레퍼런스를 참조하십시오 [ListJobs](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 [여기](#)를 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **StartCrawler** CLI와 함께 사용

다음 코드 예제는 StartCrawler의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Start an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StartCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new StartCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- API 세부 정보는 AWS SDK for .NET API [StartCrawler](#)참조를 참조하십시오.

C++

SDK for C++

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::StartCrawlerRequest request;
request.SetName(CRAWLER_NAME);

Aws::Glue::Model::StartCrawlerOutcome outcome =
client.StartCrawler(request);

if (outcome.IsSuccess() || (Aws::Glue::GlueErrors::CRAWLER_RUNNING ==
                           outcome.GetError().GetErrorType())) {
    if (!outcome.IsSuccess()) {
        std::cout << "Crawler was already started." << std::endl;
    }
    else {
        std::cout << "Successfully started crawler." << std::endl;
    }

    std::cout << "This may take a while to run." << std::endl;

    Aws::Glue::Model::CrawlerState crawlerState =
    Aws::Glue::Model::CrawlerState::NOT_SET;
    int iterations = 0;
    while (Aws::Glue::Model::CrawlerState::READY != crawlerState) {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++iterations;
        if ((iterations % 10) == 0) { // Log status every 10 seconds.
```

```
        std::cout << "Crawler status " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
        crawlerState)
        << ". After " << iterations
        << " seconds elapsed."
        << std::endl;
    }
    Aws::Glue::Model::GetCrawlerRequest getCrawlerRequest;
    getCrawlerRequest.SetName(CRAWLER_NAME);

    Aws::Glue::Model::GetCrawlerOutcome getCrawlerOutcome =
client.GetCrawler(
        getCrawlerRequest);

    if (getCrawlerOutcome.IsSuccess()) {
        crawlerState =
getCrawlerOutcome.GetResult().GetCrawler().GetState();
    }
    else {
        std::cerr << "Error getting crawler.  "
        << getCrawlerOutcome.GetError().GetMessage() <<
std::endl;
        break;
    }
}

if (Aws::Glue::Model::CrawlerState::READY == crawlerState) {
    std::cout << "Crawler finished running after " << iterations
        << " seconds."
        << std::endl;
}
}
else {
    std::cerr << "Error starting a crawler.  "
        << outcome.GetError().GetMessage()
        << std::endl;

    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
        clientConfig);
    return false;
}
```

- API 세부 정보는 AWS SDK for C++ API [StartCrawler](#) 참조를 참조하십시오.

CLI

AWS CLI

크롤러를 시작하려면

다음 `start-crawler` 예제에서는 크롤러를 시작합니다.

```
aws glue start-crawler --name my-crawler
```

출력:

```
None
```

자세한 내용은 AWS Glue 개발자 안내서의 [크롤러 정의](#) 섹션을 참조하세요.

- API 세부 정보는 AWS CLI 명령 [StartCrawler](#) 참조를 참조하십시오.

Java

SDK for Java 2.x

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.glue.GlueClient;
import software.amazon.awssdk.services.glue.model.GlueException;
import software.amazon.awssdk.services.glue.model.StartCrawlerRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 */
```

```
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
*/
public class StartCrawler {
    public static void main(String[] args) {
        final String usage = ""

            Usage:
                <crawlerName>

            Where:
                crawlerName - The name of the crawler.\s
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String crawlerName = args[0];
        Region region = Region.US_EAST_1;
        GlueClient glueClient = GlueClient.builder()
            .region(region)
            .build();

        startSpecificCrawler(glueClient, crawlerName);
        glueClient.close();
    }

    public static void startSpecificCrawler(GlueClient glueClient, String
crawlerName) {
        try {
            StartCrawlerRequest crawlerRequest = StartCrawlerRequest.builder()
                .name(crawlerName)
                .build();

            glueClient.startCrawler(crawlerRequest);

        } catch (GlueException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```


- API 세부 정보는 AWS SDK for Java 2.x API [StartCrawler](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [StartCrawler](#)참조를 참조하십시오.

Kotlin

SDK for Kotlin

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun startSpecificCrawler(crawlerName: String?) {
```

```

val request =
    StartCrawlerRequest {
        name = crawlerName
    }

GlueClient { region = "us-west-2" }.use { glueClient ->
    glueClient.startCrawler(request)
    println("$crawlerName was successfully started.")
}
}

```

- API 세부 정보는 Kotlin API용 AWS SDK 레퍼런스를 참조하세요 [StartCrawler](#).

PHP

SDK for PHP

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

$crawlerName = "example-crawler-test-" . $uniqid;

$databaseName = "doc-example-database-$uniqid";

$glueService->startCrawler($crawlerName);


public function startCrawler($crawlerName): Result
{
    return $this->glueClient->startCrawler([
        'Name' => $crawlerName,
    ]);
}

```

- API 세부 정보는 AWS SDK for PHP API [StartCrawler](#) 참조를 참조하십시오.

Python

SDK for Python(Boto3)

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def start_crawler(self, name):
        """
        Starts a crawler. The crawler crawls its configured target and creates
        metadata that describes the data it finds in the target data source.

        :param name: The name of the crawler to start.
        """
        try:
            self.glue_client.start_crawler(Name=name)
        except ClientError as err:
            logger.error(
                "Couldn't start crawler %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [StartCrawler](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Starts a crawler with the specified name.
  #
  # @param name [String] The name of the crawler to start.
  # @return [void]
  def start_crawler(name)
    @glue_client.start_crawler(name: name)
  rescue Aws::Glue::Errors::ServiceError => e
    @logger.error("Glue could not start crawler #{name}: \n#{e.message}")
    raise
  end
end
```

- API 세부 정보는 AWS SDK for Ruby API [StartCrawler](#) 참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

let start_crawler =
glue.start_crawler().name(self.crawler()).send().await;

match start_crawler {
    Ok(_) => Ok(()),
    Err(err) => {
        let glue_err: aws_sdk_glue::Error = err.into();
        match glue_err {
            aws_sdk_glue::Error::CrawlerRunningException(_) => Ok(()),
            _ => Err(GlueMvpError::GlueSdk(glue_err)),
        }
    }
}
};

```

- API에 대한 자세한 내용은 Rust용AWS SDK API 레퍼런스를 참조하십시오 [StartCrawler](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **StartJobRun** CLI와 함께 사용

다음 코드 예제는 StartJobRun의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- [크롤러 및 작업 시작하기](#)

.NET

AWS SDK for .NET

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
/// <summary>
/// Start an AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A string representing the job run Id.</returns>
public async Task<string> StartJobRunAsync(
    string jobName,
    string inputDatabase,
    string inputTable,
    string bucketName)
{
    var request = new StartJobRunRequest
    {
        JobName = jobName,
        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
```

- API 세부 정보는 AWS SDK for .NET API [StartJobRun](#) 참조를 참조하십시오.

C++

SDK for C++

 Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region in which the bucket was created
// (overrides config file).
// clientConfig.region = "us-east-1";

Aws::Glue::GlueClient client(clientConfig);

Aws::Glue::Model::StartJobRunRequest request;
request.SetJobName(JOB_NAME);

Aws::Map<Aws::String, Aws::String> arguments;
arguments["--input_database"] = CRAWLER_DATABASE_NAME;
arguments["--input_table"] = tableName;
arguments["--output_bucket_url"] = Aws::String("s3://") + bucketName +
"/";
request.SetArguments(arguments);

Aws::Glue::Model::StartJobRunOutcome outcome =
client.StartJobRun(request);

if (outcome.IsSuccess()) {
    std::cout << "Successfully started the job." << std::endl;

    Aws::String jobRunId = outcome.GetResult().GetJobRunId();

    int iterator = 0;
    bool done = false;
    while (!done) {
        ++iterator;
        std::this_thread::sleep_for(std::chrono::seconds(1));
        Aws::Glue::Model::GetJobRunRequest jobRunRequest;
        jobRunRequest.SetJobName(JOB_NAME);
```

```

        jobRunRequest.SetRunId(jobRunId);

        Aws::Glue::Model::GetJobRunOutcome jobRunOutcome =
client.GetJobRun(
        jobRunRequest);

        if (jobRunOutcome.IsSuccess()) {
            const Aws::Glue::Model::JobRun &jobRun =
jobRunOutcome.GetResult().GetJobRun();
            Aws::Glue::Model::JobRunState jobRunState =
jobRun.GetJobRunState();

            if ((jobRunState == Aws::Glue::Model::JobRunState::STOPPED)
||
                (jobRunState == Aws::Glue::Model::JobRunState::FAILED) ||
                (jobRunState == Aws::Glue::Model::JobRunState::TIMEOUT))
{
                std::cerr << "Error running job. "
                    << jobRun.GetErrorMessage()
                    << std::endl;
                deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME,
JOB_NAME,
                    bucketName,
                    clientConfig);
                return false;
            }
            else if (jobRunState ==
                Aws::Glue::Model::JobRunState::SUCCEEDED) {
                std::cout << "Job run succeeded after " << iterator <<
                    " seconds elapsed." << std::endl;
                done = true;
            }
            else if ((iterator % 10) == 0) { // Log status every 10
seconds.
                std::cout << "Job run status " <<

                Aws::Glue::Model::JobRunStateMapper::GetNameForJobRunState(
                    jobRunState) <<
                    ". " << iterator <<
                    " seconds elapsed." << std::endl;
            }
        }
        else {
            std::cerr << "Error retrieving job run state. "

```



```

        << jobRunOutcome.GetError().GetMessage()
        << std::endl;
        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
                    bucketName, clientConfig);
        return false;
    }
}
else {
    std::cerr << "Error starting a job. " <<
outcome.GetError().GetMessage()
        << std::endl;
    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
bucketName,
                clientConfig);
    return false;
}

```

- API 세부 정보는 AWS SDK for C++ API [StartJobRun](#)참조를 참조하십시오.

CLI

AWS CLI

작업을 실행하기 시작하려면

다음 `start-job-run` 예제에서는 작업을 시작합니다.

```
aws glue start-job-run \
  --job-name my-job
```

출력:

```
{
  "JobRunId":
  "jr_22208b1f44eb5376a60569d4b21dd20fcb8621e1a366b4e7b2494af764b82ded"
}
```

자세한 내용은 AWS Glue 개발자 안내서의 [작업 작성](#)을 참조하세요.

- API 세부 정보는 AWS CLI 명령 [StartJobRun](#)참조를 참조하십시오.

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};
```

- API 세부 정보는 AWS SDK for JavaScript API [StartJobRun](#)참조를 참조하십시오.

PHP

SDK for PHP

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
$jobName = 'test-job-' . $uniqid;
```

```

    $databaseName = "doc-example-database-$uniqid";

    $tables = $glueService->getTables($databaseName);

    $outputBucketUrl = "s3://$bucketName";
    $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
    $outputBucketUrl)['JobRunId'];

    public function startJobRun($jobName, $databaseName, $tables,
    $outputBucketUrl): Result
    {
        return $this->glueClient->startJobRun([
            'JobName' => $jobName,
            'Arguments' => [
                'input_database' => $databaseName,
                'input_table' => $tables['TableList'][0]['Name'],
                'output_bucket_url' => $outputBucketUrl,
                '--input_database' => $databaseName,
                '--input_table' => $tables['TableList'][0]['Name'],
                '--output_bucket_url' => $outputBucketUrl,
            ],
        ]);
    }

```

- API 세부 정보는 AWS SDK for PHP API [StartJobRun](#)참조를 참조하십시오.

Python

SDK for Python(Boto3)

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """

```

```

        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def start_job_run(self, name, input_database, input_table,
output_bucket_name):
        """
        Starts a job run. A job run extracts data from the source, transforms it,
        and loads it to the output bucket.

        :param name: The name of the job definition.
        :param input_database: The name of the metadata database that contains
tables
                                that describe the source data. This is typically
created
                                by a crawler.
        :param input_table: The name of the table in the metadata database that
                                describes the source data.
        :param output_bucket_name: The S3 bucket where the output is written.
        :return: The ID of the job run.
        """
        try:
            # The custom Arguments that are passed to this function are used by
the
            # Python ETL script to determine the location of input and output
data.

            response = self.glue_client.start_job_run(
                JobName=name,
                Arguments={
                    "--input_database": input_database,
                    "--input_table": input_table,
                    "--output_bucket_url": f"s3://{output_bucket_name}/",
                },
            )
        except ClientError as err:
            logger.error(
                "Couldn't start job run %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:

```

```
return response["JobRunId"]
```

- API에 대한 자세한 내용은 파이썬용 AWS SDK (Boto3) API 레퍼런스를 참조하십시오 [StartJobRun](#).

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
# a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Starts a job run for the specified job.
  #
  # @param name [String] The name of the job to start the run for.
  # @param input_database [String] The name of the input database for the job.
  # @param input_table [String] The name of the input table for the job.
  # @param output_bucket_name [String] The name of the output S3 bucket for the
  # job.
  # @return [String] The ID of the started job run.
  def start_job_run(name, input_database, input_table, output_bucket_name)
    response = @glue_client.start_job_run(
      job_name: name,
```

```

arguments: {
  '--input_database': input_database,
  '--input_table': input_table,
  '--output_bucket_url': "s3://#{output_bucket_name}/"
}
)
response.job_run_id
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not start job run #{name}: \n#{e.message}")
  raise
end

```

- API 세부 정보는 AWS SDK for Ruby API [StartJobRun](#) 참조를 참조하십시오.

Rust

SDK for Rust

Note

자세한 내용은 다음과 같습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```

let job_run_output = glue
  .start_job_run()
  .job_name(self.job())
  .arguments("--input_database", self.database())
  .arguments(
    "--input_table",
    self.tables
      .first()
      .ok_or_else(|| GlueMvpError::Unknown("Missing crawler
table".into()))?
      .name(),
  )
  .arguments("--output_bucket_url", self.bucket())
  .send()
  .await
  .map_err(GlueMvpError::from_glue_sdk)?;

```

```
let job = job_run_output
    .job_run_id()
    .ok_or_else(|| GlueMvpError::Unknown("Missing run id from just
started job".into()))?
    .to_string();
```

- API에 대한 자세한 내용은 Rust용 AWS SDK API 레퍼런스를 참조하십시오 [StartJobRun](#).

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#) 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK AWS Glue 사용 시나리오

다음 코드 예제는 AWS SDK를 사용하여 일반적인 시나리오를 구현하는 방법을 보여줍니다. AWS Glue 이 시나리오는 내에서 여러 함수를 호출하여 특정 작업을 수행하는 방법을 보여줍니다. AWS Glue 각 시나리오에는 코드 설정 및 실행 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다.

GitHub

예

- [SDK를 사용하여 AWS Glue 크롤러 및 작업 실행 시작하기 AWS](#)

SDK를 사용하여 AWS Glue 크롤러 및 작업 실행 시작하기 AWS

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 퍼블릭 Amazon S3 버킷을 크롤링하고 CSV 형식의 메타데이터 데이터베이스를 생성하는 크롤러를 생성합니다.
- 내 데이터베이스 및 테이블에 대한 정보를 나열하십시오. AWS Glue Data Catalog
- 작업을 생성하여 S3 버킷에서 CSV 데이터를 추출하고, 데이터를 변환하며, JSON 형식의 출력을 다른 S3 버킷으로 로드합니다.
- 작업 실행에 대한 정보를 나열하고 변환된 데이터를 확인하며 리소스를 정리합니다.

자세한 내용은 [자습서: AWS Glue Studio 시작하기](#)를 참조하십시오.

.NET

AWS SDK for .NET

Note

자세한 내용은 에서 확인할 수 GitHub 있습니다. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

시나리오에서 사용되는 AWS Glue 함수를 래핑하는 클래스를 만드세요.

```
using System.Net;

namespace GlueActions;

public class GlueWrapper
{
    private readonly IAmazonGlue _amazonGlue;

    /// <summary>
    /// Constructor for the AWS Glue actions wrapper.
    /// </summary>
    /// <param name="amazonGlue"></param>
    public GlueWrapper(IAmazonGlue amazonGlue)
    {
        _amazonGlue = amazonGlue;
    }

    /// <summary>
    /// Create an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name for the crawler.</param>
    /// <param name="crawlerDescription">A description of the crawler.</param>
    /// <param name="role">The AWS Identity and Access Management (IAM) role to
    /// be assumed by the crawler.</param>
    /// <param name="schedule">The schedule on which the crawler will be
    executed.</param>
    /// <param name="s3Path">The path to the Amazon Simple Storage Service
    (Amazon S3)
    /// bucket where the Python script has been stored.</param>
    /// <param name="dbName">The name to use for the database that will be
```



```
/// created by the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateCrawlerAsync(
    string crawlerName,
    string crawlerDescription,
    string role,
    string schedule,
    string s3Path,
    string dbName)
{
    var s3Target = new S3Target
    {
        Path = s3Path,
    };

    var targetList = new List<S3Target>
    {
        s3Target,
    };

    var targets = new CrawlerTargets
    {
        S3Targets = targetList,
    };

    var crawlerRequest = new CreateCrawlerRequest
    {
        DatabaseName = dbName,
        Name = crawlerName,
        Description = crawlerDescription,
        Targets = targets,
        Role = role,
        Schedule = schedule,
    };

    var response = await _amazonGlue.CreateCrawlerAsync(crawlerRequest);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Create an AWS Glue job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
```

```
/// <param name="roleName">The name of the IAM role to be assumed by
/// the job.</param>
/// <param name="description">A description of the job.</param>
/// <param name="scriptUrl">The URL to the script.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateJobAsync(string dbName, string tableName,
string bucketUrl, string jobName, string roleName, string description, string
scriptUrl)
{
    var command = new JobCommand
    {
        PythonVersion = "3",
        Name = "glueetl",
        ScriptLocation = scriptUrl,
    };

    var arguments = new Dictionary<string, string>
    {
        { "--input_database", dbName },
        { "--input_table", tableName },
        { "--output_bucket_url", bucketUrl }
    };

    var request = new CreateJobRequest
    {
        Command = command,
        DefaultArguments = arguments,
        Description = description,
        GlueVersion = "3.0",
        Name = jobName,
        NumberOfWorkers = 10,
        Role = roleName,
        WorkerType = "G.1X"
    };

    var response = await _amazonGlue.CreateJobAsync(request);
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
```

```
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteCrawlerAsync(string crawlerName)
{
    var response = await _amazonGlue.DeleteCrawlerAsync(new
DeleteCrawlerRequest { Name = crawlerName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete the AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteDatabaseAsync(string dbName)
{
    var response = await _amazonGlue.DeleteDatabaseAsync(new
DeleteDatabaseRequest { Name = dbName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete an AWS Glue job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteJobAsync(string jobName)
{
    var response = await _amazonGlue.DeleteJobAsync(new DeleteJobRequest
{ JobName = jobName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete a table from an AWS Glue database.
/// </summary>
/// <param name="tableName">The table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteTableAsync(string dbName, string tableName)
{
    var response = await _amazonGlue.DeleteTableAsync(new DeleteTableRequest
{ Name = tableName, DatabaseName = dbName });
}
```

```
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get information about an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A Crawler object describing the crawler.</returns>
    public async Task<Crawler?> GetCrawlerAsync(string crawlerName)
    {
        var crawlerRequest = new GetCrawlerRequest
        {
            Name = crawlerName,
        };

        var response = await _amazonGlue.GetCrawlerAsync(crawlerRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            var databaseName = response.Crawler.DatabaseName;
            Console.WriteLine($"{crawlerName} has the database {databaseName}");
            return response.Crawler;
        }

        Console.WriteLine($"No information regarding {crawlerName} could be
found.");
        return null;
    }

    /// <summary>
    /// Get information about the state of an AWS Glue crawler.
    /// </summary>
    /// <param name="crawlerName">The name of the crawler.</param>
    /// <returns>A value describing the state of the crawler.</returns>
    public async Task<CrawlerState> GetCrawlerStateAsync(string crawlerName)
    {
        var response = await _amazonGlue.GetCrawlerAsync(
            new GetCrawlerRequest { Name = crawlerName });
        return response.Crawler.State;
    }

    /// <summary>
```

```
/// Get information about an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A Database object containing information about the database.</
returns>
public async Task<Database> GetDatabaseAsync(string dbName)
{
    var databasesRequest = new GetDatabaseRequest
    {
        Name = dbName,
    };

    var response = await _amazonGlue.GetDatabaseAsync(databasesRequest);
    return response.Database;
}

/// <summary>
/// Get information about a specific AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <param name="jobRunId">The Id of the job run.</param>
/// <returns>A JobRun object with information about the job run.</returns>
public async Task<JobRun> GetJobRunAsync(string jobName, string jobRunId)
{
    var response = await _amazonGlue.GetJobRunAsync(new GetJobRunRequest
{ JobName = jobName, RunId = jobRunId });
    return response.JobRun;
}

/// <summary>
/// Get information about all AWS Glue runs of a specific job.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A list of JobRun objects.</returns>
public async Task<List<JobRun>> GetJobRunsAsync(string jobName)
{
    var jobRuns = new List<JobRun>();

    var request = new GetJobRunsRequest
    {
        JobName = jobName,
    };
};
```

```
// No need to loop to get all the log groups--the SDK does it for us
behind the scenes
var paginatorForJobRuns =
    _amazonGlue.Paginators.GetJobRuns(request);

await foreach (var response in paginatorForJobRuns.Responses)
{
    response.JobRuns.ForEach(jobRun =>
    {
        jobRuns.Add(jobRun);
    });
}

return jobRuns;
}

/// <summary>
/// Get a list of tables for an AWS Glue database.
/// </summary>
/// <param name="dbName">The name of the database.</param>
/// <returns>A list of Table objects.</returns>
public async Task<List<Table>> GetTablesAsync(string dbName)
{
    var request = new GetTablesRequest { DatabaseName = dbName };
    var tables = new List<Table>();

    // Get a paginator for listing the tables.
    var tablePaginator = _amazonGlue.Paginators.GetTables(request);

    await foreach (var response in tablePaginator.Responses)
    {
        tables.AddRange(response.TableList);
    }

    return tables;
}

/// <summary>
/// List AWS Glue jobs using a paginator.
/// </summary>
/// <returns>A list of AWS Glue job names.</returns>
```

```
public async Task<List<string>> ListJobsAsync()
{
    var jobNames = new List<string>();

    var listJobsPaginator = _amazonGlue.Paginators.ListJobs(new
ListJobsRequest { MaxResults = 10 });
    await foreach (var response in listJobsPaginator.Responses)
    {
        jobNames.AddRange(response.JobNames);
    }

    return jobNames;
}

/// <summary>
/// Start an AWS Glue crawler.
/// </summary>
/// <param name="crawlerName">The name of the crawler.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> StartCrawlerAsync(string crawlerName)
{
    var crawlerRequest = new StartCrawlerRequest
    {
        Name = crawlerName,
    };

    var response = await _amazonGlue.StartCrawlerAsync(crawlerRequest);

    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}

/// <summary>
/// Start an AWS Glue job run.
/// </summary>
/// <param name="jobName">The name of the job.</param>
/// <returns>A string representing the job run Id.</returns>
public async Task<string> StartJobRunAsync(
    string jobName,
    string inputDatabase,
    string inputTable,
    string bucketName)
{
```

```

    var request = new StartJobRunRequest
    {
        JobName = jobName,
        Arguments = new Dictionary<string, string>
        {
            {"--input_database", inputDatabase},
            {"--input_table", inputTable},
            {"--output_bucket_url", $"s3://{bucketName}/"}
        }
    };

    var response = await _amazonGlue.StartJobRunAsync(request);
    return response.JobRunId;
}
}

```

시나리오를 실행하는 클래스를 생성합니다.

```

global using Amazon.Glue;
global using GlueActions;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;

using Amazon.Glue.Model;
using Amazon.S3;
using Amazon.S3.Model;

namespace GlueBasics;

public class GlueBasics
{
    private static ILogger logger = null!;
    private static IConfiguration _configuration = null!;
}

```



```
static async Task Main(string[] args)
{
    // Set up dependency injection for AWS Glue.
    using var host = Host.CreateDefaultBuilder(args)
        .ConfigureLogging(logging =>
            logging.AddFilter("System", LogLevel.Debug)
                .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
        .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonGlue>()
                .AddTransient<GlueWrapper>()
                .AddTransient<UiWrapper>()
            )
        .Build();

    logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
        .CreateLogger<GlueBasics>();

    _configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("settings.json") // Load settings from .json file.
        .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
        .Build();

    // These values are stored in settings.json
    // Once you have run the CDK script to deploy the resources,
    // edit the file to set "BucketName", "RoleName", and "ScriptURL"
    // to the appropriate values. Also set "CrawlerName" to the name
    // you want to give the crawler when it is created.
    string bucketName = _configuration["BucketName"]!;
    string bucketUrl = _configuration["BucketUrl"]!;
    string crawlerName = _configuration["CrawlerName"]!;
    string roleName = _configuration["RoleName"]!;
    string sourceData = _configuration["SourceData"]!;
    string dbName = _configuration["DbName"]!;
    string cron = _configuration["Cron"]!;
    string scriptUrl = _configuration["ScriptURL"]!;
    string jobName = _configuration["JobName"]!;

    var wrapper = host.Services.GetRequiredService<GlueWrapper>();
    var uiWrapper = host.Services.GetRequiredService<UiWrapper>();
}
```

```
uiWrapper.DisplayOverview();
uiWrapper.PressEnter();

// Create the crawler and wait for it to be ready.
uiWrapper.DisplayTitle("Create AWS Glue crawler");
Console.WriteLine("Let's begin by creating the AWS Glue crawler.");

var crawlerDescription = "Crawler created for the AWS Glue Basics
scenario.";
var crawlerCreated = await wrapper.CreateCrawlerAsync(crawlerName,
crawlerDescription, roleName, cron, sourceData, dbName);
if (crawlerCreated)
{
    Console.WriteLine($"The crawler: {crawlerName} has been created. Now
let's wait until it's ready.");
    CrawlerState crawlerState;
    do
    {
        crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
    }
    while (crawlerState != "READY");
    Console.WriteLine($"The crawler {crawlerName} is now ready for
use.");
}
else
{
    Console.WriteLine($"Couldn't create crawler {crawlerName}.");
    return; // Exit the application.
}

uiWrapper.DisplayTitle("Start AWS Glue crawler");
Console.WriteLine("Now let's wait until the crawler has successfully
started.");
var crawlerStarted = await wrapper.StartCrawlerAsync(crawlerName);
if (crawlerStarted)
{
    CrawlerState crawlerState;
    do
    {
        crawlerState = await wrapper.GetCrawlerStateAsync(crawlerName);
    }
    while (crawlerState != "READY");
}
```

```
        Console.WriteLine($"The crawler {crawlerName} is now ready for
use.");
    }
    else
    {
        Console.WriteLine($"Couldn't start the crawler {crawlerName}.");
        return; // Exit the application.
    }

    uiWrapper.PressEnter();

    Console.WriteLine($"
Let's take a look at the database: {dbName}");
    var database = await wrapper.GetDatabaseAsync(dbName);

    if (database != null)
    {
        uiWrapper.DisplayTitle($"{database.Name} Details");
        Console.WriteLine($"{database.Name} created on
{database.CreateTime}");
        Console.WriteLine(database.Description);
    }

    uiWrapper.PressEnter();

    var tables = await wrapper.GetTablesAsync(dbName);
    if (tables.Count > 0)
    {
        tables.ForEach(table =>
        {
            Console.WriteLine($"{table.Name}\tCreated:
[table.CreateTime]\tUpdated: {table.UpdateTime}");
        });
    }

    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Create AWS Glue job");
    Console.WriteLine("Creating a new AWS Glue job.");
    var description = "An AWS Glue job created using the AWS SDK for .NET";
    await wrapper.CreateJobAsync(dbName, tables[0].Name, bucketUrl, jobName,
roleName, description, scriptUrl);

    uiWrapper.PressEnter();
```

```
    uiWrapper.DisplayTitle("Starting AWS Glue job");
    Console.WriteLine("Starting the new AWS Glue job...");
    var jobRunId = await wrapper.StartJobRunAsync(jobName, dbName,
tables[0].Name, bucketName);
    var jobRunComplete = false;
    var jobRun = new JobRun();
    do
    {
        jobRun = await wrapper.GetJobRunAsync(jobName, jobRunId);
        if (jobRun.JobRunState == "SUCCEEDED" || jobRun.JobRunState ==
"STOPPED" ||
            jobRun.JobRunState == "FAILED" || jobRun.JobRunState ==
"TIMEOUT")
        {
            jobRunComplete = true;
        }
    } while (!jobRunComplete);

    uiWrapper.DisplayTitle($"Data in {bucketName}");

    // Get the list of data stored in the S3 bucket.
    var s3Client = new AmazonS3Client();

    var response = await s3Client.ListObjectsAsync(new ListObjectsRequest
{ BucketName = bucketName });
    response.S3Objects.ForEach(s3Object =>
    {
        Console.WriteLine(s3Object.Key);
    });

    uiWrapper.DisplayTitle("AWS Glue jobs");
    var jobNames = await wrapper.ListJobsAsync();
    jobNames.ForEach(jobName =>
    {
        Console.WriteLine(jobName);
    });

    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Get AWS Glue job run information");
    Console.WriteLine("Getting information about the AWS Glue job.");
    var jobRuns = await wrapper.GetJobRunsAsync(jobName);

    jobRuns.ForEach(jobRun =>
```

```
    {
        Console.WriteLine($"{jobRun.JobName}\t{jobRun.JobRunState}\t{jobRun.CompletedOn}");
    });

    uiWrapper.PressEnter();

    uiWrapper.DisplayTitle("Deleting resources");
    Console.WriteLine("Deleting the AWS Glue job used by the example.");
    await wrapper.DeleteJobAsync(jobName);

    Console.WriteLine("Deleting the tables from the database.");
    tables.ForEach(async table =>
    {
        await wrapper.DeleteTableAsync(dbName, table.Name);
    });

    Console.WriteLine("Deleting the database.");
    await wrapper.DeleteDatabaseAsync(dbName);

    Console.WriteLine("Deleting the AWS Glue crawler.");
    await wrapper.DeleteCrawlerAsync(crawlerName);

    Console.WriteLine("The AWS Glue scenario has completed.");
    uiWrapper.PressEnter();
}
}

namespace GlueBasics;

public class UiWrapper
{
    public readonly string SepBar = new string('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the scenario.
    /// </summary>
    public void DisplayOverview()
    {
        Console.Clear();
        DisplayTitle("Amazon Glue: get started with crawlers and jobs");

        Console.WriteLine("This example application does the following:");
    }
}
```

```

        Console.WriteLine("\t 1. Create a crawler, pass it the IAM role and the
URL to the public S3 bucket that contains the source data");
        Console.WriteLine("\t 2. Start the crawler.");
        Console.WriteLine("\t 3. Get the database created by the crawler and the
tables in the database.");
        Console.WriteLine("\t 4. Create a job.");
        Console.WriteLine("\t 5. Start a job run.");
        Console.WriteLine("\t 6. Wait for the job run to complete.");
        Console.WriteLine("\t 7. Show the data stored in the bucket.");
        Console.WriteLine("\t 8. List jobs for the account.");
        Console.WriteLine("\t 9. Get job run details for the job that was run.");
        Console.WriteLine("\t10. Delete the demo job.");
        Console.WriteLine("\t11. Delete the database and tables created for the
demo.");
        Console.WriteLine("\t12. Delete the crawler.");
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.WriteLine("\nPlease press <Enter> to continue. ");
        _ = Console.ReadLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to center on the screen.</param>
    /// <returns>The string padded to make it center on the screen.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)

```

```
{  
    Console.WriteLine(SepBar);  
    Console.WriteLine(CenterString(strTitle));  
    Console.WriteLine(SepBar);  
}  
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하십시오.

- [CreateCrawler](#)
- [CreateJob](#)
- [DeleteCrawler](#)
- [DeleteDatabase](#)
- [DeleteJob](#)
- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)
- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

C++

SDK for C++

 Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```

//! Scenario which demonstrates using AWS Glue to add a crawler and run a job.
/!*
  \sa runGettingStartedWithGlueScenario()
  \param bucketName: An S3 bucket created in the setup.
  \param roleName: An AWS Identity and Access Management (IAM) role created in the
  setup.
  \param clientConfig: AWS client configuration.
  \return bool: Successful completion.
*/

bool AwsDoc::Glue::runGettingStartedWithGlueScenario(const Aws::String
&bucketName,
                                                    const Aws::String &roleName,
                                                    const
Aws::Client::ClientConfiguration &clientConfig) {
  Aws::Glue::GlueClient client(clientConfig);

  Aws::String roleArn;
  if (!getRoleArn(roleName, roleArn, clientConfig)) {
    std::cerr << "Error getting role ARN for role." << std::endl;
    return false;
  }

  // 1. Upload the job script to the S3 bucket.
  {
    std::cout << "Uploading the job script '"
              << AwsDoc::Glue::PYTHON_SCRIPT
              << "'." << std::endl;

    if (!AwsDoc::Glue::uploadFile(bucketName,
                                  AwsDoc::Glue::PYTHON_SCRIPT_PATH,
                                  AwsDoc::Glue::PYTHON_SCRIPT,

```



```
        clientConfig)) {
            std::cerr << "Error uploading the job file." << std::endl;
            return false;
        }
    }

    // 2. Create a crawler.
    {
        Aws::Glue::Model::S3Target s3Target;
        s3Target.SetPath("s3://crawler-public-us-east-1/flight/2016/csv");
        Aws::Glue::Model::CrawlerTargets crawlerTargets;
        crawlerTargets.AddS3Targets(s3Target);

        Aws::Glue::Model::CreateCrawlerRequest request;
        request.SetTargets(crawlerTargets);
        request.SetName(CRAWLER_NAME);
        request.SetDatabaseName(CRAWLER_DATABASE_NAME);
        request.SetTablePrefix(CRAWLER_DATABASE_PREFIX);
        request.SetRole(roleArn);

        Aws::Glue::Model::CreateCrawlerOutcome outcome =
client.CreateCrawler(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully created the crawler." << std::endl;
        }
        else {
            std::cerr << "Error creating a crawler. " <<
outcome.GetError().GetMessage()
                << std::endl;
            deleteAssets("", CRAWLER_DATABASE_NAME, "", bucketName,
clientConfig);
            return false;
        }
    }

    // 3. Get a crawler.
    {
        Aws::Glue::Model::GetCrawlerRequest request;
        request.SetName(CRAWLER_NAME);

        Aws::Glue::Model::GetCrawlerOutcome outcome = client.GetCrawler(request);

        if (outcome.IsSuccess()) {
```

```

        Aws::Glue::Model::CrawlerState crawlerState =
outcome.GetResult().GetCrawler().GetState();
        std::cout << "Retrieved crawler with state " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
        crawlerState)
        << "." << std::endl;
    }
    else {
        std::cerr << "Error retrieving a crawler. "
        << outcome.GetError().GetMessage() << std::endl;
        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
            clientConfig);
        return false;
    }
}

// 4. Start a crawler.
{
    Aws::Glue::Model::StartCrawlerRequest request;
    request.SetName(CRAWLER_NAME);

    Aws::Glue::Model::StartCrawlerOutcome outcome =
client.StartCrawler(request);

    if (outcome.IsSuccess() || (Aws::Glue::GlueErrors::CRAWLER_RUNNING ==
        outcome.GetError().GetErrorType())) {
        if (!outcome.IsSuccess()) {
            std::cout << "Crawler was already started." << std::endl;
        }
        else {
            std::cout << "Successfully started crawler." << std::endl;
        }

        std::cout << "This may take a while to run." << std::endl;

        Aws::Glue::Model::CrawlerState crawlerState =
Aws::Glue::Model::CrawlerState::NOT_SET;
        int iterations = 0;
        while (Aws::Glue::Model::CrawlerState::READY != crawlerState) {
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++iterations;
            if ((iterations % 10) == 0) { // Log status every 10 seconds.

```

```

        std::cout << "Crawler status " <<

Aws::Glue::Model::CrawlerStateMapper::GetNameForCrawlerState(
        crawlerState)
        << ". After " << iterations
        << " seconds elapsed."
        << std::endl;
    }
    Aws::Glue::Model::GetCrawlerRequest getCrawlerRequest;
    getCrawlerRequest.SetName(CRAWLER_NAME);

    Aws::Glue::Model::GetCrawlerOutcome getCrawlerOutcome =
client.GetCrawler(
        getCrawlerRequest);

    if (getCrawlerOutcome.IsSuccess()) {
        crawlerState =
getCrawlerOutcome.GetResult().GetCrawler().GetState();
    }
    else {
        std::cerr << "Error getting crawler.  "
        << getCrawlerOutcome.GetError().GetMessage() <<
std::endl;

        break;
    }
}

if (Aws::Glue::Model::CrawlerState::READY == crawlerState) {
    std::cout << "Crawler finished running after " << iterations
        << " seconds."
        << std::endl;
}
}
else {
    std::cerr << "Error starting a crawler.  "
        << outcome.GetError().GetMessage()
        << std::endl;

    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
        clientConfig);
    return false;
}
}
}

```

```
// 5. Get a database.
{
    Aws::Glue::Model::GetDatabaseRequest request;
    request.SetName(CRAWLER_DATABASE_NAME);

    Aws::Glue::Model::GetDatabaseOutcome outcome =
client.GetDatabase(request);

    if (outcome.IsSuccess()) {
        const Aws::Glue::Model::Database &database =
outcome.GetResult().GetDatabase();

        std::cout << "Successfully retrieve the database\n" <<
            database.Jsonize().View().WriteReadable() << ". " <<
std::endl;
    }
    else {
        std::cerr << "Error getting the database. "
            << outcome.GetError().GetMessage() << std::endl;
        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
            clientConfig);
        return false;
    }
}

// 6. Get tables.
Aws::String tableName;
{
    Aws::Glue::Model::GetTablesRequest request;
    request.SetDatabaseName(CRAWLER_DATABASE_NAME);
    std::vector<Aws::Glue::Model::Table> all_tables;
    Aws::String nextToken; // Used for pagination.
    do {
        Aws::Glue::Model::GetTablesOutcome outcome =
client.GetTables(request);

        if (outcome.IsSuccess()) {
            const std::vector<Aws::Glue::Model::Table> &tables =
outcome.GetResult().GetTableList();
            all_tables.insert(all_tables.end(), tables.begin(),
tables.end());
            nextToken = outcome.GetResult().GetNextToken();
        }
        else {
```

```

        std::cerr << "Error getting the tables. "
                << outcome.GetError().GetMessage()
                << std::endl;
        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
                    clientConfig);
        return false;
    }
} while (!nextToken.empty());

std::cout << "The database contains " << all_tables.size()
          << (all_tables.size() == 1 ?
            " table." : "tables.") << std::endl;
std::cout << "Here is a list of the tables in the database.";
for (size_t index = 0; index < all_tables.size(); ++index) {
    std::cout << "    " << index + 1 << ": " <<
all_tables[index].GetName()
          << std::endl;
}

if (!all_tables.empty()) {
    int tableIndex = askQuestionForIntRange(
        "Enter an index to display the database detail ",
        1, static_cast<int>(all_tables.size()));
    std::cout << all_tables[tableIndex -
1].Jsonize().View().WriteReadable()
          << std::endl;

    tableName = all_tables[tableIndex - 1].GetName();
}
}

// 7. Create a job.
{
    Aws::Glue::Model::CreateJobRequest request;
    request.SetName(JOB_NAME);
    request.SetRole(roleArn);
    request.SetGlueVersion(GLUE_VERSION);

    Aws::Glue::Model::JobCommand command;
    command.SetName(JOB_COMMAND_NAME);
    command.SetPythonVersion(JOB_PYTHON_VERSION);
    command.SetScriptLocation(
        Aws::String("s3://") + bucketName + "/" + PYTHON_SCRIPT);
    request.SetCommand(command);
}

```

```
Aws::Glue::Model::CreateJobOutcome outcome = client.CreateJob(request);

if (outcome.IsSuccess()) {
    std::cout << "Successfully created the job." << std::endl;
}
else {
    std::cerr << "Error creating the job. " <<
outcome.GetError().GetMessage()
    << std::endl;
    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, "", bucketName,
    clientConfig);
    return false;
}
}

// 8. Start a job run.
{
    Aws::Glue::Model::StartJobRunRequest request;
    request.SetJobName(JOB_NAME);

    Aws::Map<Aws::String, Aws::String> arguments;
    arguments["--input_database"] = CRAWLER_DATABASE_NAME;
    arguments["--input_table"] = tableName;
    arguments["--output_bucket_url"] = Aws::String("s3://") + bucketName +
"/";
    request.SetArguments(arguments);

    Aws::Glue::Model::StartJobRunOutcome outcome =
client.StartJobRun(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully started the job." << std::endl;

        Aws::String jobRunId = outcome.GetResult().GetJobRunId();

        int iterator = 0;
        bool done = false;
        while (!done) {
            ++iterator;
            std::this_thread::sleep_for(std::chrono::seconds(1));
            Aws::Glue::Model::GetJobRunRequest jobRunRequest;
            jobRunRequest.SetJobName(JOB_NAME);
            jobRunRequest.SetRunId(jobRunId);
```

```

        Aws::Glue::Model::GetJobRunOutcome jobRunOutcome =
client.GetJobRun(
            jobRunRequest);

        if (jobRunOutcome.IsSuccess()) {
            const Aws::Glue::Model::JobRun &jobRun =
jobRunOutcome.GetResult().GetJobRun();
            Aws::Glue::Model::JobRunState jobRunState =
jobRun.GetJobRunState();

            if ((jobRunState == Aws::Glue::Model::JobRunState::STOPPED)
||
                (jobRunState == Aws::Glue::Model::JobRunState::FAILED) ||
                (jobRunState == Aws::Glue::Model::JobRunState::TIMEOUT))
{
                std::cerr << "Error running job. "
                    << jobRun.GetErrorMessage()
                    << std::endl;
                deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME,
JOB_NAME,
                            bucketName,
                            clientConfig);
                return false;
            }
            else if (jobRunState ==
                Aws::Glue::Model::JobRunState::SUCCEEDED) {
                std::cout << "Job run succeeded after " << iterator <<
                    " seconds elapsed." << std::endl;
                done = true;
            }
            else if ((iterator % 10) == 0) { // Log status every 10
seconds.
                std::cout << "Job run status " <<

                Aws::Glue::Model::JobRunStateMapper::GetNameForJobRunState(
                    jobRunState) <<
                    ". " << iterator <<
                    " seconds elapsed." << std::endl;
            }
        }
        else {
            std::cerr << "Error retrieving job run state. "
                << jobRunOutcome.GetError().GetMessage()

```

```
        << std::endl;
        deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
                    bucketName, clientConfig);
        return false;
    }
}
}
else {
    std::cerr << "Error starting a job. " <<
outcome.GetError().GetMessage()
    << std::endl;
    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
bucketName,
                clientConfig);
    return false;
}
}

// 9. List the output data stored in the S3 bucket.
{
    Aws::S3::S3Client s3Client;
    Aws::S3::Model::ListObjectsV2Request request;
    request.SetBucket(bucketName);
    request.SetPrefix(OUTPUT_FILE_PREFIX);

    Aws::String continuationToken; // Used for pagination.
    std::vector<Aws::S3::Model::Object> allObjects;
    do {
        if (!continuationToken.empty()) {
            request.SetContinuationToken(continuationToken);
        }
        Aws::S3::Model::ListObjectsV2Outcome outcome =
s3Client.ListObjectsV2(
                request);

        if (outcome.IsSuccess()) {
            const std::vector<Aws::S3::Model::Object> &objects =
                outcome.GetResult().GetContents();
            allObjects.insert(allObjects.end(), objects.begin(),
objects.end());
            continuationToken =
outcome.GetResult().GetNextContinuationToken();
        }
        else {
```



```

        std::cerr << "Error listing objects. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        break;
    }
} while (!continuationToken.empty());

std::cout << "Data from your job is in " << allObjects.size() <<
          " files in the S3 bucket, " << bucketName << "." << std::endl;

for (size_t i = 0; i < allObjects.size(); ++i) {
    std::cout << "    " << i + 1 << ". " << allObjects[i].GetKey()
              << std::endl;
}

int objectIndex = askQuestionForIntRange(
    std::string(
        "Enter the number of a block to download it and see the
first ") +
    std::to_string(LINES_OF_RUN_FILE_TO_DISPLAY) +
    " lines of JSON output in the block: ", 1,
    static_cast<int>(allObjects.size()));

Aws::String objectKey = allObjects[objectIndex - 1].GetKey();

std::stringstream stringStream;
if (getObjectFromBucket(bucketName, objectKey, stringStream,
    clientConfig)) {
    for (int i = 0; i < LINES_OF_RUN_FILE_TO_DISPLAY && stringStream; +
+i) {
        std::string line;
        std::getline(stringStream, line);
        std::cout << "    " << line << std::endl;
    }
}
else {
    deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
bucketName,
                clientConfig);
    return false;
}
}

// 10. List all the jobs.

```

```

Aws::String jobName;
{
    Aws::Glue::Model::ListJobsRequest listJobsRequest;

    Aws::String nextToken;
    std::vector<Aws::String> allJobNames;

    do {
        if (!nextToken.empty()) {
            listJobsRequest.SetNextToken(nextToken);
        }
        Aws::Glue::Model::ListJobsOutcome listRunsOutcome = client.ListJobs(
            listJobsRequest);

        if (listRunsOutcome.IsSuccess()) {
            const std::vector<Aws::String> &jobNames =
listRunsOutcome.GetResult().GetJobNames();
            allJobNames.insert(allJobNames.end(), jobNames.begin(),
jobNames.end());
            nextToken = listRunsOutcome.GetResult().GetNextToken();
        }
        else {
            std::cerr << "Error listing jobs. "
                << listRunsOutcome.GetError().GetMessage()
                << std::endl;
        }
    } while (!nextToken.empty());
    std::cout << "Your account has " << allJobNames.size() << " jobs."
        << std::endl;
    for (size_t i = 0; i < allJobNames.size(); ++i) {
        std::cout << "    " << i + 1 << ". " << allJobNames[i] << std::endl;
    }
    int jobIndex = askQuestionForIntRange(
        Aws::String("Enter a number between 1 and ") +
        std::to_string(allJobNames.size()) +
        " to see the list of runs for a job: ",
        1, static_cast<int>(allJobNames.size()));

    jobName = allJobNames[jobIndex - 1];
}

// 11. Get the job runs for a job.
Aws::String jobRunID;
if (!jobName.empty()) {

```

```

Aws::Glue::Model::GetJobRunsRequest getJobRunsRequest;
getJobRunsRequest.SetJobName(jobName);

Aws::String nextToken; // Used for pagination.
std::vector<Aws::Glue::Model::JobRun> allJobRuns;
do {
    if (!nextToken.empty()) {
        getJobRunsRequest.SetNextToken(nextToken);
    }
    Aws::Glue::Model::GetJobRunsOutcome jobRunsOutcome =
client.GetJobRuns(
    getJobRunsRequest);

    if (jobRunsOutcome.IsSuccess()) {
        const std::vector<Aws::Glue::Model::JobRun> &jobRuns =
jobRunsOutcome.GetResult().GetJobRuns();
        allJobRuns.insert(allJobRuns.end(), jobRuns.begin(),
jobRuns.end());

        nextToken = jobRunsOutcome.GetResult().GetNextToken();
    }
    else {
        std::cerr << "Error getting job runs. "
        << jobRunsOutcome.GetError().GetMessage()
        << std::endl;
        break;
    }
} while (!nextToken.empty());

std::cout << "There are " << allJobRuns.size() << " runs in the job '"
    <<
    jobName << "'." << std::endl;

for (size_t i = 0; i < allJobRuns.size(); ++i) {
    std::cout << "    " << i + 1 << ". " << allJobRuns[i].GetJobName()
        << std::endl;
}

int runIndex = askQuestionForIntRange(
    Aws::String("Enter a number between 1 and ") +
    std::to_string(allJobRuns.size()) +
    " to see details for a run: ",
    1, static_cast<int>(allJobRuns.size()));
jobRunID = allJobRuns[runIndex - 1].GetId();

```

```

    }

    // 12. Get a single job run.
    if (!jobRunID.empty()) {
        Aws::Glue::Model::GetJobRunRequest jobRunRequest;
        jobRunRequest.SetJobName(jobName);
        jobRunRequest.SetRunId(jobRunID);

        Aws::Glue::Model::GetJobRunOutcome jobRunOutcome = client.GetJobRun(
            jobRunRequest);

        if (jobRunOutcome.IsSuccess()) {
            std::cout << "Displaying the job run JSON description." << std::endl;
            std::cout
                <<
jobRunOutcome.GetResult().GetJobRun().Jsonize().View().WriteReadable()
                << std::endl;
        }
        else {
            std::cerr << "Error get a job run. "
                << jobRunOutcome.GetError().GetMessage()
                << std::endl;
        }
    }

    return deleteAssets(CRAWLER_NAME, CRAWLER_DATABASE_NAME, JOB_NAME,
        bucketName,
            clientConfig);
}

//! Cleanup routine to delete created assets.
/*!
    \\sa deleteAssets()
    \\param crawler: Name of an AWS Glue crawler.
    \\param database: The name of an AWS Glue database.
    \\param job: The name of an AWS Glue job.
    \\param bucketName: The name of an S3 bucket.
    \\param clientConfig: AWS client configuration.
    \\return bool: Successful completion.
*/
bool AwsDoc::Glue::deleteAssets(const Aws::String &crawler, const Aws::String
    &database,
        const Aws::String &job, const Aws::String
    &bucketName,

```

```
const Aws::Client::ClientConfiguration
&clientConfig) {
    const Aws::Glue::GlueClient client(clientConfig);
    bool result = true;

    // 13. Delete a job.
    if (!job.empty()) {
        Aws::Glue::Model::DeleteJobRequest request;
        request.SetJobName(job);

        Aws::Glue::Model::DeleteJobOutcome outcome = client.DeleteJob(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the job." << std::endl;
        }
        else {
            std::cerr << "Error deleting the job. " <<
outcome.GetError().GetMessage()
                << std::endl;
            result = false;
        }
    }

    // 14. Delete a database.
    if (!database.empty()) {
        Aws::Glue::Model::DeleteDatabaseRequest request;
        request.SetName(database);

        Aws::Glue::Model::DeleteDatabaseOutcome outcome = client.DeleteDatabase(
            request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the database." << std::endl;
        }
        else {
            std::cerr << "Error deleting database. " <<
outcome.GetError().GetMessage()
                << std::endl;
            result = false;
        }
    }

    // 15. Delete a crawler.
```

```

    if (!crawler.empty()) {
        Aws::Glue::Model::DeleteCrawlerRequest request;
        request.SetName(crawler);

        Aws::Glue::Model::DeleteCrawlerOutcome outcome =
client.DeleteCrawler(request);

        if (outcome.IsSuccess()) {
            std::cout << "Successfully deleted the crawler." << std::endl;
        }
        else {
            std::cerr << "Error deleting the crawler. "
                << outcome.GetError().GetMessage() << std::endl;
            result = false;
        }
    }

    // 16. Delete the job script and run data from the S3 bucket.
    result &= AwsDoc::Glue::deleteAllObjectsInS3Bucket(bucketName,
clientConfig);

    return result;
}

//! Routine which uploads a file to an S3 bucket.
/*!
    \sa uploadFile()
    \param bucketName: An S3 bucket created in the setup.
    \param filePath: The path of the file to upload.
    \param fileName The name for the uploaded file.
    \param clientConfig: AWS client configuration.
    \return bool: Successful completion.
*/
bool
AwsDoc::Glue::uploadFile(const Aws::String &bucketName,
                        const Aws::String &filePath,
                        const Aws::String &fileName,
                        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::S3::S3Client s3_client(clientConfig);

    Aws::S3::Model::PutObjectRequest request;
    request.SetBucket(bucketName);
    request.SetKey(fileName);

    std::shared_ptr<Aws::IOStream> inputData =

```

```

        Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
                                     filePath.c_str(),
                                     std::ios_base::in |
std::ios_base::binary);

    if (!*inputData) {
        std::cerr << "Error unable to read file " << filePath << std::endl;
        return false;
    }

    request.SetBody(inputData);

    Aws::S3::Model::PutObjectOutcome outcome =
        s3_client.PutObject(request);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error: PutObject: " <<
            outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Added object '" << filePath << "' to bucket '"
            << bucketName << "'." << std::endl;
    }

    return outcome.IsSuccess();
}

//! Routine which deletes all objects in an S3 bucket.
/*!
  \sa deleteAllObjectsInS3Bucket()
  \param bucketName: The S3 bucket name.
  \param clientConfig: AWS client configuration.
  \return bool: Successful completion.
  */
bool AwsDoc::Glue::deleteAllObjectsInS3Bucket(const Aws::String &bucketName,
                                              const
    Aws::Client::ClientConfiguration &clientConfig) {
    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::ListObjectsV2Request listObjectsRequest;
    listObjectsRequest.SetBucket(bucketName);

    Aws::String continuationToken; // Used for pagination.
    bool result = true;
    do {

```

```
    if (!continuationToken.empty()) {
        listObjectsRequest.SetContinuationToken(continuationToken);
    }

    Aws::S3::Model::ListObjectsV2Outcome listObjectsOutcome =
client.ListObjectsV2(
    listObjectsRequest);

    if (listObjectsOutcome.IsSuccess()) {
        const std::vector<Aws::S3::Model::Object> &objects =
listObjectsOutcome.GetResult().GetContents();
        if (!objects.empty()) {
            Aws::S3::Model::DeleteObjectsRequest deleteObjectsRequest;
            deleteObjectsRequest.SetBucket(bucketName);

            std::vector<Aws::S3::Model::ObjectIdentifier> objectIdentifiers;
            for (const Aws::S3::Model::Object &object: objects) {
                objectIdentifiers.push_back(
                    Aws::S3::Model::ObjectIdentifier().WithKey(
                        object.GetKey()));
            }
            Aws::S3::Model::Delete objectsDelete;
            objectsDelete.SetObjects(objectIdentifiers);
            objectsDelete.SetQuiet(true);
            deleteObjectsRequest.SetDelete(objectsDelete);

            Aws::S3::Model::DeleteObjectsOutcome deleteObjectsOutcome =
                client.DeleteObjects(deleteObjectsRequest);

            if (!deleteObjectsOutcome.IsSuccess()) {
                std::cerr << "Error deleting objects. " <<
                    deleteObjectsOutcome.GetError().GetMessage() <<
std::endl;

                result = false;
                break;
            }
            else {
                std::cout << "Successfully deleted the objects." <<
std::endl;

            }
        }
    }
    else {
        std::cout << "No objects to delete in '" << bucketName << "'."

```



```

        << std::endl;
    }

    continuationToken =
listObjectsOutcome.GetResult().GetNextContinuationToken();
    }
    else {
        std::cerr << "Error listing objects. "
            << listObjectsOutcome.GetError().GetMessage() << std::endl;
        result = false;
        break;
    }
} while (!continuationToken.empty());

return result;
}

//! Routine which retrieves an object from an S3 bucket.
/*!
  \sa getObjectFromBucket()
  \param bucketName: The S3 bucket name.
  \param objectKey: The object's name.
  \param objectStream: A stream to receive the retrieved data.
  \param clientConfig: AWS client configuration.
  \return bool: Successful completion.
  */
bool AwsDoc::Glue::getObjectFromBucket(const Aws::String &bucketName,
                                       const Aws::String &objectKey,
                                       std::ostream &objectStream,
                                       const Aws::Client::ClientConfiguration
&clientConfig) {
    Aws::S3::S3Client client(clientConfig);
    Aws::S3::Model::GetObjectRequest request;
    request.SetBucket(bucketName);
    request.SetKey(objectKey);

    Aws::S3::Model::GetObjectOutcome outcome = client.GetObject(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully retrieved '" << objectKey << "'." <<
std::endl;
        auto &body = outcome.GetResult().GetBody();
        objectStream << body.rdbuf();
    }
}

```

```
    }
    else {
        std::cerr << "Error retrieving object. " <<
outcome.GetError().GetMessage()
        << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API 세부 정보는 AWS SDK for C++ API 참조의 다음 주제를 참조하십시오.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobRun](#)
 - [GetJobRuns](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobRun](#)

Java

SDK for Java 2.x

 Note

더 많은 것이 있어요 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 *
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
 * started.html
 *
 * To set up the resources, see this documentation topic:
 *
 * https://docs.aws.amazon.com/glue/latest/ug/tutorial-add-crawler.html
 *
 * This example performs the following tasks:
 *
 * 1. Create a database.
 * 2. Create a crawler.
 * 3. Get a crawler.
 * 4. Start a crawler.
 * 5. Get a database.
 * 6. Get tables.
 * 7. Create a job.
 * 8. Start a job run.
 * 9. List all jobs.
 * 10. Get job runs.
 * 11. Delete a job.
 * 12. Delete a database.
 * 13. Delete a crawler.
 */

public class GlueScenario {
```

```

public static final String DASHES = new String(new char[80]).replace("\0",
"-");

public static void main(String[] args) throws InterruptedException {
    final String usage = ""

        Usage:
            <iam> <s3Path> <cron> <dbName> <crawlerName> <jobName>\s

        Where:
            iam - The ARN of the IAM role that has AWS Glue and S3
permissions.\s
            s3Path - The Amazon Simple Storage Service (Amazon S3) target
that contains data (for example, CSV data).
            cron - A cron expression used to specify the schedule (i.e.,
cron(15 12 * * ? *).
            dbName - The database name.\s
            crawlerName - The name of the crawler.\s
            jobName - The name you assign to this job definition.
            scriptLocation - The Amazon S3 path to a script that runs a
job.

            locationUri - The location of the database
            bucketNameSc - The Amazon S3 bucket name used when creating a
job

        """;

    if (args.length != 9) {
        System.out.println(usage);
        System.exit(1);
    }

    String iam = args[0];
    String s3Path = args[1];
    String cron = args[2];
    String dbName = args[3];
    String crawlerName = args[4];
    String jobName = args[5];
    String scriptLocation = args[6];
    String locationUri = args[7];
    String bucketNameSc = args[8];

    Region region = Region.US_EAST_1;
    GlueClient glueClient = GlueClient.builder()
        .region(region)

```

```
        .build());
System.out.println(DASHES);
System.out.println("Welcome to the AWS Glue scenario.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("1. Create a database.");
createDatabase(glueClient, dbName, locationUri);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("2. Create a crawler.");
createGlueCrawler(glueClient, iam, s3Path, cron, dbName, crawlerName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("3. Get a crawler.");
getSpecificCrawler(glueClient, crawlerName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("4. Start a crawler.");
startSpecificCrawler(glueClient, crawlerName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("5. Get a database.");
getSpecificDatabase(glueClient, dbName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("**** Wait 5 min for the tables to become available");
TimeUnit.MINUTES.sleep(5);
System.out.println("6. Get tables.");
String myTableName = getGlueTables(glueClient, dbName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("7. Create a job.");
createJob(glueClient, jobName, iam, scriptLocation);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("8. Start a Job run.");
```

```
startJob(glueClient, jobName, dbName, myTableName, bucketNameSc);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("9. List all jobs.");
getAllJobs(glueClient);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("10. Get job runs.");
getJobRuns(glueClient, jobName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("11. Delete a job.");
deleteJob(glueClient, jobName);
System.out.println("*** Wait 5 MIN for the " + crawlerName + " to stop");
TimeUnit.MINUTES.sleep(5);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("12. Delete a database.");
deleteDatabase(glueClient, dbName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("Delete a crawler.");
deleteSpecificCrawler(glueClient, crawlerName);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("Successfully completed the AWS Glue Scenario");
System.out.println(DASHES);
}

public static void createDatabase(GlueClient glueClient, String dbName,
String locationUri) {
    try {
        DatabaseInput input = DatabaseInput.builder()
            .description("Built with the AWS SDK for Java V2")
            .name(dbName)
            .locationUri(locationUri)
            .build();
```

```
        CreateDatabaseRequest request = CreateDatabaseRequest.builder()
            .databaseInput(input)
            .build();

        glueClient.createDatabase(request);
        System.out.println(dbName + " was successfully created");

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void createGlueCrawler(GlueClient glueClient,
    String iam,
    String s3Path,
    String cron,
    String dbName,
    String crawlerName) {

    try {
        S3Target s3Target = S3Target.builder()
            .path(s3Path)
            .build();

        List<S3Target> targetList = new ArrayList<>();
        targetList.add(s3Target);
        CrawlerTargets targets = CrawlerTargets.builder()
            .s3Targets(targetList)
            .build();

        CreateCrawlerRequest crawlerRequest = CreateCrawlerRequest.builder()
            .databaseName(dbName)
            .name(crawlerName)
            .description("Created by the AWS Glue Java API")
            .targets(targets)
            .role(iam)
            .schedule(cron)
            .build();

        glueClient.createCrawler(crawlerRequest);
        System.out.println(crawlerName + " was successfully created");

    } catch (GlueException e) {
```

```
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void getSpecificCrawler(GlueClient glueClient, String
crawlerName) {
    try {
        GetCrawlerRequest crawlerRequest = GetCrawlerRequest.builder()
            .name(crawlerName)
            .build();

        boolean ready = false;
        while (!ready) {
            GetCrawlerResponse response =
glueClient.getCrawler(crawlerRequest);
            String status = response.crawler().stateAsString();
            if (status.compareTo("READY") == 0) {
                ready = true;
            }
            Thread.sleep(3000);
        }

        System.out.println("The crawler is now ready");

    } catch (GlueException | InterruptedException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}

public static void startSpecificCrawler(GlueClient glueClient, String
crawlerName) {
    try {
        StartCrawlerRequest crawlerRequest = StartCrawlerRequest.builder()
            .name(crawlerName)
            .build();

        glueClient.startCrawler(crawlerRequest);
        System.out.println(crawlerName + " was successfully started!");

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```



```
    }  
  }  
  
  public static void getSpecificDatabase(GlueClient glueClient, String  
databaseName) {  
    try {  
      GetDatabaseRequest databasesRequest = GetDatabaseRequest.builder()  
        .name(databaseName)  
        .build();  
  
      GetDatabaseResponse response =  
glueClient.getDatabase(databasesRequest);  
      Instant createDate = response.database().createTime();  
  
      // Convert the Instant to readable date.  
      DateTimeFormatter formatter =  
DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)  
        .withLocale(Locale.US)  
        .withZone(ZoneId.systemDefault());  
  
      formatter.format(createDate);  
      System.out.println("The create date of the database is " +  
createDate);  
  
    } catch (GlueException e) {  
      System.err.println(e.awsErrorDetails().errorMessage());  
      System.exit(1);  
    }  
  }  
  
  public static String getGlueTables(GlueClient glueClient, String dbName) {  
    String myTableName = "";  
    try {  
      GetTablesRequest tableRequest = GetTablesRequest.builder()  
        .databaseName(dbName)  
        .build();  
  
      GetTablesResponse response = glueClient.getTables(tableRequest);  
      List<Table> tables = response.tableList();  
      if (tables.isEmpty()) {  
        System.out.println("No tables were returned");  
      } else {  
        for (Table table : tables) {  
          myTableName = table.name();  
        }  
      }  
    }  
  }  
}
```

```
        System.out.println("Table name is: " + myTableName);
    }
}

} catch (GlueException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
return myTableName;
}

public static void startJob(GlueClient glueClient, String jobName, String
inputDatabase, String inputTable,
    String outBucket) {
    try {
        Map<String, String> myMap = new HashMap<>();
        myMap.put("--input_database", inputDatabase);
        myMap.put("--input_table", inputTable);
        myMap.put("--output_bucket_url", outBucket);

        StartJobRunRequest runRequest = StartJobRunRequest.builder()
            .workerType(WorkerType.G_1_X)
            .numberOfWorkers(10)
            .arguments(myMap)
            .jobName(jobName)
            .build();

        StartJobRunResponse response = glueClient.startJobRun(runRequest);
        System.out.println("The request Id of the job is " +
response.responseMetadata().requestId());

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void createJob(GlueClient glueClient, String jobName, String
iam, String scriptLocation) {
    try {
        JobCommand command = JobCommand.builder()
            .pythonVersion("3")
            .name("glueetl")
            .scriptLocation(scriptLocation)
```

```
        .build();

        CreateJobRequest jobRequest = CreateJobRequest.builder()
            .description("A Job created by using the AWS SDK for Java
V2")
            .glueVersion("2.0")
            .workerType(WorkerType.G_1_X)
            .numberOfWorkers(10)
            .name(jobName)
            .role(iam)
            .command(command)
            .build();

        glueClient.createJob(jobRequest);
        System.out.println(jobName + " was successfully created.");

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void getAllJobs(GlueClient glueClient) {
    try {
        GetJobsRequest jobsRequest = GetJobsRequest.builder()
            .maxResults(10)
            .build();

        GetJobsResponse jobsResponse = glueClient.getJobs(jobsRequest);
        List<Job> jobs = jobsResponse.jobs();
        for (Job job : jobs) {
            System.out.println("Job name is : " + job.name());
            System.out.println("The job worker type is : " +
job.workerType().name());
        }

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void getJobRuns(GlueClient glueClient, String jobName) {
    try {
```

```
GetJobRunsRequest runsRequest = GetJobRunsRequest.builder()
    .jobName(jobName)
    .maxResults(20)
    .build();

boolean jobDone = false;
while (!jobDone) {
    GetJobRunsResponse response = glueClient.getJobRuns(runsRequest);
    List<JobRun> jobRuns = response.jobRuns();
    for (JobRun jobRun : jobRuns) {
        String jobState = jobRun.jobRunState().name();
        if (jobState.compareTo("SUCCEEDED") == 0) {
            System.out.println(jobName + " has succeeded");
            jobDone = true;

        } else if (jobState.compareTo("STOPPED") == 0) {
            System.out.println("Job run has stopped");
            jobDone = true;

        } else if (jobState.compareTo("FAILED") == 0) {
            System.out.println("Job run has failed");
            jobDone = true;

        } else if (jobState.compareTo("TIMEOUT") == 0) {
            System.out.println("Job run has timed out");
            jobDone = true;

        } else {
            System.out.println("*** Job run state is " +
jobRun.jobRunState().name());
            System.out.println("Job run Id is " + jobRun.id());
            System.out.println("The Glue version is " +
jobRun.glueVersion());
        }
        TimeUnit.SECONDS.sleep(5);
    }
}

} catch (GlueException | InterruptedException e) {
    System.err.println(e.getMessage());
    System.exit(1);
}
}
```

```
public static void deleteJob(GlueClient glueClient, String jobName) {
    try {
        DeleteJobRequest jobRequest = DeleteJobRequest.builder()
            .jobName(jobName)
            .build();

        glueClient.deleteJob(jobRequest);
        System.out.println(jobName + " was successfully deleted");

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void deleteDatabase(GlueClient glueClient, String databaseName)
{
    try {
        DeleteDatabaseRequest request = DeleteDatabaseRequest.builder()
            .name(databaseName)
            .build();

        glueClient.deleteDatabase(request);
        System.out.println(databaseName + " was successfully deleted");

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void deleteSpecificCrawler(GlueClient glueClient, String
crawlerName) {
    try {
        DeleteCrawlerRequest deleteCrawlerRequest =
DeleteCrawlerRequest.builder()
            .name(crawlerName)
            .build();

        glueClient.deleteCrawler(deleteCrawlerRequest);
        System.out.println(crawlerName + " was deleted");

    } catch (GlueException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
    }
}
```

```
        System.exit(1);
    }
}
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하십시오.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobRun](#)
 - [GetJobRuns](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobRun](#)

JavaScript

JavaScript (v3) 용 SDK

Note

더 많은 내용이 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

공용 Amazon Simple Storage Service (S3) 버킷을 크롤링하고 검색한 CSV 형식의 데이터를 설명하는 메타데이터 데이터베이스를 생성하는 크롤러를 만들고 실행합니다.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
    return true;
  }
};
```

```
    } catch {
      return false;
    }
  };

/**
 * @param {{ createCrawler: import('../../actions/create-crawler.js').createCrawler}} actions
 */
const makeCreateCrawlerStep = (actions) => async (context) => {
  if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
    log("Crawler already exists. Skipping creation.");
  } else {
    await actions.createCrawler(
      process.env.CRAWLER_NAME,
      process.env.ROLE_NAME,
      process.env.DATABASE_NAME,
      process.env.TABLE_PREFIX,
      process.env.S3_TARGET_PATH,
    );

    log("Crawler created successfully.", { type: "success" });
  }

  return { ...context };
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
  const waitTimeInSeconds = 30;
  const { Crawler } = await getCrawler(crawlerName);

  if (!Crawler) {
    throw new Error(`Crawler with name ${crawlerName} not found.`);
  }

  if (Crawler.State === "READY") {
    return;
  }
}
```



```

log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
await wait(waitTimeInSeconds);
return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
({ startCrawler, getCrawler }) =>
async (context) => {
  log("Starting crawler.");
  await startCrawler(process.env.CRAWLER_NAME);
  log("Crawler started.", { type: "success" });

  log("Waiting for crawler to finish running. This can take a while.");
  await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
  log("Crawler ready.", { type: "success" });

  return { ...context };
};

```

내 데이터베이스 및 테이블에 대한 정보를 나열하세요 AWS Glue Data Catalog.

```

const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};

const makeGetDatabaseStep =

```

```

({ getDatabase }) =>
async (context) => {
  const {
    Database: { Name },
  } = await getDatabase(process.env.DATABASE_NAME);
  log(`Database: ${Name}`);
  return { ...context };
};

/**
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-glue').GetTablesCommandOutput>}} config
 */
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => ` • ${table.Name}\n`));
    return { ...context };
  };

```

소스 Amazon S3 버킷에서 CSV 데이터를 추출하고, 필드를 제거하고 이름을 변경하여 변환하고, JSON 형식의 출력을 다른 Amazon S3 버킷으로 로드하는 작업을 만들고 실행합니다.

```

const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};

```

```

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput> } getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }
}

```

```
switch (JobRun.JobRunState) {
  case "FAILED":
  case "TIMEOUT":
  case "STOPPED":
    throw new Error(
      `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
    );
  case "RUNNING":
    break;
  case "SUCCEEDED":
    return;
  default:
    throw new Error(`Unknown job run state: ${JobRun.JobRunState}`);
}

log(
  `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
);
await wait(waitTimeInSeconds);
return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }}
 * context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
      view the output.`);
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {
```

```

log("Starting job.");
const { JobRunId } = await startJobRun(
  process.env.JOB_NAME,
  process.env.DATABASE_NAME,
  process.env.TABLE_NAME,
  process.env.BUCKET_NAME,
);
log("Job started.", { type: "success" });

log("Waiting for job to finish running. This can take a while.");
await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
log("Job run succeeded.", { type: "success" });

await promptToOpen(context);

return { ...context };
};

```

작업 실행에 대한 정보를 나열하고 변환된 데이터 중 일부를 볼 수 있습니다.

```

const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */

```

```
/**
 * @typedef {() => Promise<import('@aws-sdk/client-
 glue').GetJobRunCommandOutput>} getJobRun
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-
 glue').GetJobRunsCommandOutput>} getJobRuns
 */

/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
 *
 * @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
 */
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (** @type { Context } */ context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };
};
```

데모 중에 생성된 모든 리소스를 삭제합니다.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};

const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};

const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

```
};

/**
 *
 * @param {import('.././../actions/delete-job.js').deleteJob} deleteJobFn
 * @param {string[]} jobNames
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
   * @type {{ selectedJobNames: string[] }}
   */
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error)),
    );
    log("Jobs deleted.", { type: "success" });
  }
};

/**
 * @param {{
 *   listJobs: import('.././../actions/list-jobs.js').listJobs,
 *   deleteJob: import('.././../actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }
  }
};
```



```

    return { ...context };
  };

/**
 * @param {import('.././../actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

/**
 * @param {{
 *   getTables: import('.././../actions/get-tables.js').getTables,
 *   deleteTable: import('.././../actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}}} context
   */
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null }),
    );

    if (TableList && TableList.length > 0) {
      /**
       * @type {{ tableNames: string[] }}
       */
      const { tableNames } = await context.prompter.prompt({
        name: "tableNames",
        type: "checkbox",
        message: "Let's clean up tables. Select tables to delete.",
        choices: TableList.map((t) => t.Name),
      });

      if (tableNames.length === 0) {
        log("No tables selected.");
      }
    }
  }

```

```

    } else {
      log("Deleting tables.");
      await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
      log("Tables deleted.", { type: "success" });
    }
  }

  return { ...context };
};

/**
 * @param {import('.././../actions/delete-database.js').deleteDatabase}
deleteDatabase
 * @param {string[]} databaseNames
 */
const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('.././../actions/get-databases.js').getDatabases
 *   deleteDatabase: import('.././../actions/delete-database.js').deleteDatabase
 * }} config
 */
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any>}} } context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbName: string[] }} */
      const { dbName } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbName.length === 0) {

```

```
        log("No databases selected.");
    } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
    }
}

return { ...context };
};

const cleanUpCrawlerStep = async (context) => {
    log(`Deleting crawler.`);

    try {
        await deleteCrawler(process.env.CRAWLER_NAME);
        log("Crawler deleted.", { type: "success" });
    } catch (err) {
        if (err.name === "EntityNotFoundException") {
            log(`Crawler is already deleted.`);
        } else {
            throw err;
        }
    }

    return { ...context };
};
```

- API 세부 정보는 AWS SDK for JavaScript API 참조의 다음 주제를 참조하십시오.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)

- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

Kotlin

SDK for Kotlin

Note

더 많은 정보가 있습니다 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <iam> <s3Path> <cron> <dbName> <crawlerName> <jobName>
            <scriptLocation> <locationUri>

        Where:
            iam - The Amazon Resource Name (ARN) of the AWS Identity and Access
            Management (IAM) role that has AWS Glue and Amazon Simple Storage Service
            (Amazon S3) permissions.
            s3Path - The Amazon Simple Storage Service (Amazon S3) target that
            contains data (for example, CSV data).
            cron - A cron expression used to specify the schedule (for example,
            cron(15 12 * * ? *).
            dbName - The database name.
            crawlerName - The name of the crawler.
            jobName - The name you assign to this job definition.
            scriptLocation - Specifies the Amazon S3 path to a script that runs a
            job.
            locationUri - Specifies the location of the database
        """
}
```

```
if (args.size != 8) {
    println(usage)
    exitProcess(1)
}

val iam = args[0]
val s3Path = args[1]
val cron = args[2]
val dbName = args[3]
val crawlerName = args[4]
val jobName = args[5]
val scriptLocation = args[6]
val locationUri = args[7]

println("About to start the AWS Glue Scenario")
createDatabase(dbName, locationUri)
createCrawler(iam, s3Path, cron, dbName, crawlerName)
getCrawler(crawlerName)
startCrawler(crawlerName)
getDatabase(dbName)
getGlueTables(dbName)
createJob(jobName, iam, scriptLocation)
startJob(jobName)
getJobs()
getJobRuns(jobName)
deleteJob(jobName)
println("**** Wait for 5 MIN so the $crawlerName is ready to be deleted")
TimeUnit.MINUTES.sleep(5)
deleteMyDatabase(dbName)
deleteCrawler(crawlerName)
}

suspend fun createDatabase(
    dbName: String?,
    locationUriVal: String?,
) {
    val input =
        DatabaseInput {
            description = "Built with the AWS SDK for Kotlin"
            name = dbName
            locationUri = locationUriVal
        }
}
```

```
val request =
    CreateDatabaseRequest {
        databaseInput = input
    }

GlueClient { region = "us-east-1" }.use { glueClient ->
    glueClient.createDatabase(request)
    println("The database was successfully created")
}
}

suspend fun createCrawler(
    iam: String?,
    s3Path: String?,
    cron: String?,
    dbName: String?,
    crawlerName: String,
) {
    val s3Target =
        S3Target {
            path = s3Path
        }

    val targetList = ArrayList<S3Target>()
    targetList.add(s3Target)

    val targetOb =
        CrawlerTargets {
            s3Targets = targetList
        }

    val crawlerRequest =
        CreateCrawlerRequest {
            databaseName = dbName
            name = crawlerName
            description = "Created by the AWS Glue Java API"
            targets = targetOb
            role = iam
            schedule = cron
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.createCrawler(crawlerRequest)
        println("$crawlerName was successfully created")
    }
}
```

```
    }
}

suspend fun getCrawler(crawlerName: String?) {
    val request =
        GetCrawlerRequest {
            name = crawlerName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getCrawler(request)
        val role = response.crawler?.role
        println("The role associated with this crawler is $role")
    }
}

suspend fun startCrawler(crawlerName: String) {
    val crawlerRequest =
        StartCrawlerRequest {
            name = crawlerName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.startCrawler(crawlerRequest)
        println("$crawlerName was successfully started.")
    }
}

suspend fun getDatabase(databaseName: String?) {
    val request =
        GetDatabaseRequest {
            name = databaseName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getDatabase(request)
        val dbDesc = response.database?.description
        println("The database description is $dbDesc")
    }
}

suspend fun getGlueTables(dbName: String?) {
    val tableRequest =
        GetTablesRequest {
```

```
        databaseName = dbName
    }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getTables(tableRequest)
        response.tableList?.forEach { tableName ->
            println("Table name is ${tableName.name}")
        }
    }
}

suspend fun startJob(jobNameVal: String?) {
    val runRequest =
        StartJobRunRequest {
            workerType = WorkerType.G1X
            numberOfWorkers = 10
            jobName = jobNameVal
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.startJobRun(runRequest)
        println("The job run Id is ${response.jobRunId}")
    }
}

suspend fun createJob(
    jobName: String,
    iam: String?,
    scriptLocationVal: String?,
) {
    val commandOb =
        JobCommand {
            pythonVersion = "3"
            name = "MyJob1"
            scriptLocation = scriptLocationVal
        }

    val jobRequest =
        CreateJobRequest {
            description = "A Job created by using the AWS SDK for Java V2"
            glueVersion = "2.0"
            workerType = WorkerType.G1X
            numberOfWorkers = 10
            name = jobName
        }
}
```



```
        role = iam
        command = commandOb
    }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.createJob(jobRequest)
        println("$jobName was successfully created.")
    }
}

suspend fun getJobs() {
    val request =
        GetJobsRequest {
            maxResults = 10
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getJobs(request)
        response.jobs?.forEach { job ->
            println("Job name is ${job.name}")
        }
    }
}

suspend fun getJobRuns(jobNameVal: String?) {
    val request =
        GetJobRunsRequest {
            jobName = jobNameVal
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        val response = glueClient.getJobRuns(request)
        response.jobRuns?.forEach { job ->
            println("Job name is ${job.jobName}")
        }
    }
}

suspend fun deleteJob(jobNameVal: String) {
    val jobRequest =
        DeleteJobRequest {
            jobName = jobNameVal
        }
}
```

```
GlueClient { region = "us-east-1" }.use { glueClient ->
    glueClient.deleteJob(jobRequest)
    println("$jobNameVal was successfully deleted")
}
}

suspend fun deleteMyDatabase(databaseName: String) {
    val request =
        DeleteDatabaseRequest {
            name = databaseName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.deleteDatabase(request)
        println("$databaseName was successfully deleted")
    }
}

suspend fun deleteCrawler(crawlerName: String) {
    val request =
        DeleteCrawlerRequest {
            name = crawlerName
        }

    GlueClient { region = "us-east-1" }.use { glueClient ->
        glueClient.deleteCrawler(request)
        println("$crawlerName was deleted")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API reference의 다음 주제를 참조하세요.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)

- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

PHP

SDK for PHP

Note

더 많은 것이 있어요 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

```
namespace Glue;

use Aws\Glue\GlueClient;
use Aws\S3\S3Client;
use AwsUtilities\AWSServiceClass;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithGlue
{
    public function run()
    {
        echo("\n");
        echo("-----\n");
        print("Welcome to the AWS Glue getting started demo using PHP!\n");
        echo("-----\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
```

```
];
$uniqid = uniqid();

$glueClient = new GlueClient($clientArgs);
$glueService = new GlueService($glueClient);
$IAMService = new IAMService();
$crawlerName = "example-crawler-test-" . $uniqid;

AWSServiceClass::$waitTime = 5;
AWSServiceClass::$maxWaitAttempts = 20;

$role = $IAMService->getRole("AWSGlueServiceRole-DocExample");

$databaseName = "doc-example-database-$uniqid";
$path = 's3://crawler-public-us-east-1/flight/2016/csv';
$glueService->createCrawler($crawlerName, $role['Role']['Arn'],
$databaseName, $path);
$glueService->startCrawler($crawlerName);

echo "Waiting for crawler";
do {
    $crawler = $glueService->getCrawler($crawlerName);
    echo ".";
    sleep(10);
} while ($crawler['Crawler']['State'] != "READY");
echo "\n";

$database = $glueService->getDatabase($databaseName);
echo "Found a database named " . $database['Database']['Name'] . "\n";

//Upload job script
$s3client = new S3Client($clientArgs);
$bucketName = "test-glue-bucket-" . $uniqid;
$s3client->createBucket([
    'Bucket' => $bucketName,
    'CreateBucketConfiguration' => ['LocationConstraint' => 'us-west-2'],
]);

$s3client->putObject([
    'Bucket' => $bucketName,
    'Key' => 'run_job.py',
    'SourceFile' => __DIR__ . '/flight_etl_job_script.py'
]);
$s3client->putObject([
```

```

        'Bucket' => $bucketName,
        'Key' => 'setup_scenario_getting_started.yaml',
        'SourceFile' => __DIR__ . '/setup_scenario_getting_started.yaml'
    ]);

    $tables = $glueService->getTables($databaseName);

    $jobName = 'test-job-' . $uniqid;
    $scriptLocation = "s3://$bucketName/run_job.py";
    $job = $glueService->createJob($jobName, $role['Role']['Arn'],
    $scriptLocation);

    $outputBucketUrl = "s3://$bucketName";
    $runId = $glueService->startJobRun($jobName, $databaseName, $tables,
    $outputBucketUrl)['JobRunId'];

    echo "waiting for job";
    do {
        $jobRun = $glueService->getJobRun($jobName, $runId);
        echo ".";
        sleep(10);
    } while (!array_intersect([$jobRun['JobRun']['JobRunState']],
    ['SUCCEEDED', 'STOPPED', 'FAILED', 'TIMEOUT']));
    echo "\n";

    $jobRuns = $glueService->getJobRuns($jobName);

    $objects = $s3client->listObjects([
        'Bucket' => $bucketName,
    ])['Contents'];

    foreach ($objects as $object) {
        echo $object['Key'] . "\n";
    }

    echo "Downloading " . $objects[1]['Key'] . "\n";
    /** @var Stream $downloadObject */
    $downloadObject = $s3client->getObject([
        'Bucket' => $bucketName,
        'Key' => $objects[1]['Key'],
    ]['Body']->getContents());
    echo "Here is the first 1000 characters in the object.";
    echo substr($downloadObject, 0, 1000);

```

```
$jobs = $glueService->listJobs();
echo "Current jobs:\n";
foreach ($jobs['JobNames'] as $jobsName) {
    echo "{$jobsName}\n";
}

echo "Delete the job.\n";
$glueClient->deleteJob([
    'JobName' => $job['Name'],
]);

echo "Delete the tables.\n";
foreach ($tables['TableList'] as $table) {
    $glueService->deleteTable($table['Name'], $databaseName);
}

echo "Delete the databases.\n";
$glueClient->deleteDatabase([
    'Name' => $databaseName,
]);

echo "Delete the crawler.\n";
$glueClient->deleteCrawler([
    'Name' => $crawlerName,
]);

$deleteObjects = $s3client->listObjectsV2([
    'Bucket' => $bucketName,
]);
echo "Delete all objects in the bucket.\n";
$deleteObjects = $s3client->deleteObjects([
    'Bucket' => $bucketName,
    'Delete' => [
        'Objects' => $deleteObjects['Contents'],
    ]
]);
echo "Delete the bucket.\n";
$s3client->deleteBucket(['Bucket' => $bucketName]);

echo "This job was brought to you by the number $uniqid\n";
}
}

namespace Glue;
```

```
use Aws\Glue\GlueClient;
use Aws\Result;

use function PHPUnit\Framework\isEmpty;

class GlueService extends \AwsUtilities\AWSServiceClass
{
    protected GlueClient $glueClient;

    public function __construct($glueClient)
    {
        $this->glueClient = $glueClient;
    }

    public function getCrawler($crawlerName)
    {
        return $this->customWaiter(function () use ($crawlerName) {
            return $this->glueClient->getCrawler([
                'Name' => $crawlerName,
            ]);
        });
    }

    public function createCrawler($crawlerName, $role, $databaseName, $path):
    Result
    {
        return $this->customWaiter(function () use ($crawlerName, $role,
        $databaseName, $path) {
            return $this->glueClient->createCrawler([
                'Name' => $crawlerName,
                'Role' => $role,
                'DatabaseName' => $databaseName,
                'Targets' => [
                    'S3Targets' =>
                        [[
                            'Path' => $path,
                        ]]
                ],
            ]);
        });
    }

    public function startCrawler($crawlerName): Result
```

```
{
    return $this->glueClient->startCrawler([
        'Name' => $crawlerName,
    ]);
}

public function getDatabase(string $databaseName): Result
{
    return $this->customWaiter(function () use ($databaseName) {
        return $this->glueClient->getDatabase([
            'Name' => $databaseName,
        ]);
    });
}

public function getTables($databaseName): Result
{
    return $this->glueClient->getTables([
        'DatabaseName' => $databaseName,
    ]);
}

public function createJob($jobName, $role, $scriptLocation, $pythonVersion =
'3', $glueVersion = '3.0'): Result
{
    return $this->glueClient->createJob([
        'Name' => $jobName,
        'Role' => $role,
        'Command' => [
            'Name' => 'glueetl',
            'ScriptLocation' => $scriptLocation,
            'PythonVersion' => $pythonVersion,
        ],
        'GlueVersion' => $glueVersion,
    ]);
}

public function startJobRun($jobName, $databaseName, $tables,
$outputBucketUrl): Result
{
    return $this->glueClient->startJobRun([
        'JobName' => $jobName,
        'Arguments' => [
            'input_database' => $databaseName,
```



```

        'input_table' => $tables['TableList'][0]['Name'],
        'output_bucket_url' => $outputBucketUrl,
        '--input_database' => $databaseName,
        '--input_table' => $tables['TableList'][0]['Name'],
        '--output_bucket_url' => $outputBucketUrl,
    ],
    ]);
}

public function listJobs($maxResults = null, $nextToken = null, $tags = []):
Result
{
    $arguments = [];
    if ($maxResults) {
        $arguments['MaxResults'] = $maxResults;
    }
    if ($nextToken) {
        $arguments['NextToken'] = $nextToken;
    }
    if (!empty($tags)) {
        $arguments['Tags'] = $tags;
    }
    return $this->glueClient->listJobs($arguments);
}

public function getJobRuns($jobName, $maxResults = 0, $nextToken = ''):
Result
{
    $arguments = ['JobName' => $jobName];
    if ($maxResults) {
        $arguments['MaxResults'] = $maxResults;
    }
    if ($nextToken) {
        $arguments['NextToken'] = $nextToken;
    }
    return $this->glueClient->getJobRuns($arguments);
}

public function getJobRun($jobName, $runId, $predecessorsIncluded = false):
Result
{
    return $this->glueClient->getJobRun([
        'JobName' => $jobName,
        'RunId' => $runId,

```

```
        'PredecessorsIncluded' => $predecessorsIncluded,
    ]);
}

public function deleteJob($jobName)
{
    return $this->glueClient->deleteJob([
        'JobName' => $jobName,
    ]);
}

public function deleteTable($tableName, $databaseName)
{
    return $this->glueClient->deleteTable([
        'DatabaseName' => $databaseName,
        'Name' => $tableName,
    ]);
}

public function deleteDatabase($databaseName)
{
    return $this->glueClient->deleteDatabase([
        'Name' => $databaseName,
    ]);
}

public function deleteCrawler($crawlerName)
{
    return $this->glueClient->deleteCrawler([
        'Name' => $crawlerName,
    ]);
}
}
```

- API 세부 정보는 AWS SDK for PHP API 참조의 다음 주제를 참조하십시오.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)

- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)
- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

Python

SDK for Python(Boto3)

Note

더 많은 것이 있어요 GitHub. [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

시나리오에서 사용되는 AWS Glue 함수를 래핑하는 클래스를 만드세요.

```
class GlueWrapper:
    """Encapsulates AWS Glue actions."""

    def __init__(self, glue_client):
        """
        :param glue_client: A Boto3 Glue client.
        """
        self.glue_client = glue_client

    def get_crawler(self, name):
        """
        Gets information about a crawler.
```

```

:param name: The name of the crawler to look up.
:return: Data about the crawler.
"""
crawler = None
try:
    response = self.glue_client.get_crawler(Name=name)
    crawler = response["Crawler"]
except ClientError as err:
    if err.response["Error"]["Code"] == "EntityNotFoundException":
        logger.info("Crawler %s doesn't exist.", name)
    else:
        logger.error(
            "Couldn't get crawler %s. Here's why: %s: %s",
            name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
return crawler

def create_crawler(self, name, role_arn, db_name, db_prefix, s3_target):
    """
    Creates a crawler that can crawl the specified target and populate a
    database in your AWS Glue Data Catalog with metadata that describes the
    data
    in the target.

    :param name: The name of the crawler.
    :param role_arn: The Amazon Resource Name (ARN) of an AWS Identity and
    Access
    Management (IAM) role that grants permission to let AWS
    Glue
    access the resources it needs.
    :param db_name: The name to give the database that is created by the
    crawler.
    :param db_prefix: The prefix to give any database tables that are created
    by
    the crawler.
    :param s3_target: The URL to an S3 bucket that contains data that is
    the target of the crawler.
    """
    try:

```

```
        self.glue_client.create_crawler(
            Name=name,
            Role=role_arn,
            DatabaseName=db_name,
            TablePrefix=db_prefix,
            Targets={"S3Targets": [{"Path": s3_target}]},
        )
    except ClientError as err:
        logger.error(
            "Couldn't create crawler. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

def start_crawler(self, name):
    """
    Starts a crawler. The crawler crawls its configured target and creates
    metadata that describes the data it finds in the target data source.

    :param name: The name of the crawler to start.
    """
    try:
        self.glue_client.start_crawler(Name=name)
    except ClientError as err:
        logger.error(
            "Couldn't start crawler %s. Here's why: %s: %s",
            name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

def get_database(self, name):
    """
    Gets information about a database in your Data Catalog.

    :param name: The name of the database to look up.
    :return: Information about the database.
    """
    try:
        response = self.glue_client.get_database(Name=name)
```

```
except ClientError as err:
    logger.error(
        "Couldn't get database %s. Here's why: %s: %s",
        name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return response["Database"]

def get_tables(self, db_name):
    """
    Gets a list of tables in a Data Catalog database.

    :param db_name: The name of the database to query.
    :return: The list of tables in the database.
    """
    try:
        response = self.glue_client.get_tables(DatabaseName=db_name)
    except ClientError as err:
        logger.error(
            "Couldn't get tables %s. Here's why: %s: %s",
            db_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return response["TableList"]

def create_job(self, name, description, role_arn, script_location):
    """
    Creates a job definition for an extract, transform, and load (ETL) job
    that can
    be run by AWS Glue.

    :param name: The name of the job definition.
    :param description: The description of the job definition.
    :param role_arn: The ARN of an IAM role that grants AWS Glue the
    permissions
                    it requires to run the job.
```

```

        :param script_location: The Amazon S3 URL of a Python ETL script that is
run as
                                part of the job. The script defines how the data
is
                                transformed.

    """
    try:
        self.glue_client.create_job(
            Name=name,
            Description=description,
            Role=role_arn,
            Command={
                "Name": "glueetl",
                "ScriptLocation": script_location,
                "PythonVersion": "3",
            },
            GlueVersion="3.0",
        )
    except ClientError as err:
        logger.error(
            "Couldn't create job %s. Here's why: %s: %s",
            name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

    def start_job_run(self, name, input_database, input_table,
output_bucket_name):
    """
        Starts a job run. A job run extracts data from the source, transforms it,
and loads it to the output bucket.

        :param name: The name of the job definition.
        :param input_database: The name of the metadata database that contains
tables
                                that describe the source data. This is typically
created
                                by a crawler.
        :param input_table: The name of the table in the metadata database that
describes the source data.
        :param output_bucket_name: The S3 bucket where the output is written.
        :return: The ID of the job run.

```

```
"""
try:
    # The custom Arguments that are passed to this function are used by
the
    # Python ETL script to determine the location of input and output
data.
    response = self.glue_client.start_job_run(
        JobName=name,
        Arguments={
            "--input_database": input_database,
            "--input_table": input_table,
            "--output_bucket_url": f"s3://{output_bucket_name}/",
        },
    )
except ClientError as err:
    logger.error(
        "Couldn't start job run %s. Here's why: %s: %s",
        name,
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
else:
    return response["JobRunId"]

def list_jobs(self):
    """
    Lists the names of job definitions in your account.

    :return: The list of job definition names.
    """
    try:
        response = self.glue_client.list_jobs()
    except ClientError as err:
        logger.error(
            "Couldn't list jobs. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return response["JobNames"]
```



```
def get_job_runs(self, job_name):
    """
    Gets information about runs that have been performed for a specific job
    definition.

    :param job_name: The name of the job definition to look up.
    :return: The list of job runs.
    """
    try:
        response = self.glue_client.get_job_runs(JobName=job_name)
    except ClientError as err:
        logger.error(
            "Couldn't get job runs for %s. Here's why: %s: %s",
            job_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return response["JobRuns"]

def get_job_run(self, name, run_id):
    """
    Gets information about a single job run.

    :param name: The name of the job definition for the run.
    :param run_id: The ID of the run.
    :return: Information about the run.
    """
    try:
        response = self.glue_client.get_job_run(JobName=name, RunId=run_id)
    except ClientError as err:
        logger.error(
            "Couldn't get job run %s/%s. Here's why: %s: %s",
            name,
            run_id,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
    else:
        return response["JobRun"]
```

```
def delete_job(self, job_name):
    """
    Deletes a job definition. This also deletes data about all runs that are
    associated with this job definition.

    :param job_name: The name of the job definition to delete.
    """
    try:
        self.glue_client.delete_job(JobName=job_name)
    except ClientError as err:
        logger.error(
            "Couldn't delete job %s. Here's why: %s: %s",
            job_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

def delete_table(self, db_name, table_name):
    """
    Deletes a table from a metadata database.

    :param db_name: The name of the database that contains the table.
    :param table_name: The name of the table to delete.
    """
    try:
        self.glue_client.delete_table(DatabaseName=db_name, Name=table_name)
    except ClientError as err:
        logger.error(
            "Couldn't delete table %s. Here's why: %s: %s",
            table_name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

def delete_database(self, name):
    """
    Deletes a metadata database from your Data Catalog.
```

```
        :param name: The name of the database to delete.
        """
    try:
        self.glue_client.delete_database(Name=name)
    except ClientError as err:
        logger.error(
            "Couldn't delete database %s. Here's why: %s: %s",
            name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise

def delete_crawler(self, name):
    """
    Deletes a crawler.

    :param name: The name of the crawler to delete.
    """
    try:
        self.glue_client.delete_crawler(Name=name)
    except ClientError as err:
        logger.error(
            "Couldn't delete crawler %s. Here's why: %s: %s",
            name,
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

시나리오를 실행하는 클래스를 생성합니다.

```
class GlueCrawlerJobScenario:
    """
    Encapsulates a scenario that shows how to create an AWS Glue crawler and job
    and use
    them to transform data from CSV to JSON format.
    """
```

```

def __init__(self, glue_client, glue_service_role, glue_bucket):
    """
    :param glue_client: A Boto3 AWS Glue client.
    :param glue_service_role: An AWS Identity and Access Management (IAM)
role
                                that AWS Glue can assume to gain access to the
                                resources it requires.
    :param glue_bucket: An S3 bucket that can hold a job script and output
data
                                from AWS Glue job runs.
    """
    self.glue_client = glue_client
    self.glue_service_role = glue_service_role
    self.glue_bucket = glue_bucket

    @staticmethod
    def wait(seconds, tick=12):
        """
        Waits for a specified number of seconds, while also displaying an
animated
        spinner.

        :param seconds: The number of seconds to wait.
        :param tick: The number of frames per second used to animate the spinner.
        """
        progress = "|/-\\"
        waited = 0
        while waited < seconds:
            for frame in range(tick):
                sys.stdout.write(f"\r{progress[frame % len(progress)]}")
                sys.stdout.flush()
                time.sleep(1 / tick)
            waited += 1

    def upload_job_script(self, job_script):
        """
        Uploads a Python ETL script to an S3 bucket. The script is used by the
AWS Glue
        job to transform data.

        :param job_script: The relative path to the job script.
        """
        try:

```

```
        self.glue_bucket.upload_file(Filename=job_script, Key=job_script)
        print(f"Uploaded job script '{job_script}' to the example bucket.")
    except S3UploadFailedError as err:
        logger.error("Couldn't upload job script. Here's why: %s", err)
        raise

    def run(self, crawler_name, db_name, db_prefix, data_source, job_script,
job_name):
        """
        Runs the scenario. This is an interactive experience that runs at a
command
prompt and asks you for input throughout.

:param crawler_name: The name of the crawler used in the scenario. If the
crawler does not exist, it is created.
:param db_name: The name to give the metadata database created by the
crawler.
:param db_prefix: The prefix to give tables added to the database by the
crawler.
:param data_source: The location of the data source that is targeted by
the
crawler and extracted during job runs.
:param job_script: The job script that is used to transform data during
job
runs.
:param job_name: The name to give the job definition that is created
during the
scenario.
        """
        wrapper = GlueWrapper(self.glue_client)
        print(f"Checking for crawler {crawler_name}.")
        crawler = wrapper.get_crawler(crawler_name)
        if crawler is None:
            print(f"Creating crawler {crawler_name}.")
            wrapper.create_crawler(
                crawler_name,
                self.glue_service_role.arn,
                db_name,
                db_prefix,
                data_source,
            )
            print(f"Created crawler {crawler_name}.")
            crawler = wrapper.get_crawler(crawler_name)
        pprint(crawler)
```

```
print("-" * 88)

print(
    f"When you run the crawler, it crawls data stored in {data_source}
and "
    f"creates a metadata database in the AWS Glue Data Catalog that
describes "
    f"the data in the data source."
)
print("In this example, the source data is in CSV format.")
ready = False
while not ready:
    ready = Question.ask_question(
        "Ready to start the crawler? (y/n) ", Question.is_yesno
    )
wrapper.start_crawler(crawler_name)
print("Let's wait for the crawler to run. This typically takes a few
minutes.")
crawler_state = None
while crawler_state != "READY":
    self.wait(10)
    crawler = wrapper.get_crawler(crawler_name)
    crawler_state = crawler["State"]
    print(f"Crawler is {crawler['State']}")
print("-" * 88)

database = wrapper.get_database(db_name)
print(f"The crawler created database {db_name}:")
pprint(database)
print(f"The database contains these tables:")
tables = wrapper.get_tables(db_name)
for index, table in enumerate(tables):
    print(f"\t{index + 1}. {table['Name']}")
table_index = Question.ask_question(
    f"Enter the number of a table to see more detail: ",
    Question.is_int,
    Question.in_range(1, len(tables)),
)
pprint(tables[table_index - 1])
print("-" * 88)

print(f"Creating job definition {job_name}.")
wrapper.create_job(
    job_name,
```

```

        "Getting started example job.",
        self.glue_service_role.arn,
        f"s3://{self.glue_bucket.name}/{job_script}",
    )
    print("Created job definition.")
    print(
        f"When you run the job, it extracts data from {data_source},
transforms it "
        f"by using the {job_script} script, and loads the output into "
        f"S3 bucket {self.glue_bucket.name}."
    )
    print(
        "In this example, the data is transformed from CSV to JSON, and only
a few "
        "fields are included in the output."
    )
    job_run_status = None
    if Question.ask_question(f"Ready to run? (y/n) ", Question.is_yesno):
        job_run_id = wrapper.start_job_run(
            job_name, db_name, tables[0]["Name"], self.glue_bucket.name
        )
        print(f"Job {job_name} started. Let's wait for it to run.")
        while job_run_status not in ["SUCCEEDED", "STOPPED", "FAILED",
"TIMEOUT"]:
            self.wait(10)
            job_run = wrapper.get_job_run(job_name, job_run_id)
            job_run_status = job_run["JobRunState"]
            print(f"Job {job_name}/{job_run_id} is {job_run_status}.")
        print("-" * 88)

        if job_run_status == "SUCCEEDED":
            print(
                f"Data from your job run is stored in your S3 bucket
'{self.glue_bucket.name}':"
            )
            try:
                keys = [
                    obj.key for obj in
self.glue_bucket.objects.filter(Prefix="run-")
                ]
                for index, key in enumerate(keys):
                    print(f"\t{index + 1}: {key}")
                lines = 4
                key_index = Question.ask_question(

```

```

        f"Enter the number of a block to download it and see the
first {lines} "
        f"lines of JSON output in the block: ",
        Question.is_int,
        Question.in_range(1, len(keys)),
    )
    job_data = io.BytesIO()
    self.glue_bucket.download_fileobj(keys[key_index - 1], job_data)
    job_data.seek(0)
    for _ in range(lines):
        print(job_data.readline().decode("utf-8"))
except ClientError as err:
    logger.error(
        "Couldn't get job run data. Here's why: %s: %s",
        err.response["Error"]["Code"],
        err.response["Error"]["Message"],
    )
    raise
print("-" * 88)

job_names = wrapper.list_jobs()
if job_names:
    print(f"Your account has {len(job_names)} jobs defined:")
    for index, job_name in enumerate(job_names):
        print(f"\t{index + 1}. {job_name}")
    job_index = Question.ask_question(
        f"Enter a number between 1 and {len(job_names)} to see the list
of runs for "
        f"a job: ",
        Question.is_int,
        Question.in_range(1, len(job_names)),
    )
    job_runs = wrapper.get_job_runs(job_names[job_index - 1])
    if job_runs:
        print(f"Found {len(job_runs)} runs for job {job_names[job_index -
1]}:")

        for index, job_run in enumerate(job_runs):
            print(
                f"\t{index + 1}. {job_run['JobRunState']} on "
                f"{job_run['CompletedOn']:%Y-%m-%d %H:%M:%S}"
            )
        run_index = Question.ask_question(
            f"Enter a number between 1 and {len(job_runs)} to see details
for a run: ",

```



```

        Question.is_int,
        Question.in_range(1, len(job_runs)),
    )
    pprint(job_runs[run_index - 1])
else:
    print(f"No runs found for job {job_names[job_index - 1]}")
else:
    print("Your account doesn't have any jobs defined.")
print("-" * 88)

print(
    f"Let's clean up. During this example we created job definition
    '{job_name}'."
)
if Question.ask_question(
    "Do you want to delete the definition and all runs? (y/n) ",
    Question.is_yesno,
):
    wrapper.delete_job(job_name)
    print(f"Job definition '{job_name}' deleted.")
tables = wrapper.get_tables(db_name)
print(f"We also created database '{db_name}' that contains these
tables:")
for table in tables:
    print(f"\t{table['Name']}")
if Question.ask_question(
    "Do you want to delete the tables and the database? (y/n) ",
    Question.is_yesno,
):
    for table in tables:
        wrapper.delete_table(db_name, table["Name"])
        print(f"Deleted table {table['Name']}")
    wrapper.delete_database(db_name)
    print(f"Deleted database {db_name}.")
print(f"We also created crawler '{crawler_name}'.")
if Question.ask_question(
    "Do you want to delete the crawler? (y/n) ", Question.is_yesno
):
    wrapper.delete_crawler(crawler_name)
    print(f"Deleted crawler {crawler_name}.")
print("-" * 88)

def parse_args(args):

```

```
"""
Parse command line arguments.

:param args: The command line arguments.
:return: The parsed arguments.
"""
parser = argparse.ArgumentParser(
    description="Runs the AWS Glue getting started with crawlers and jobs
scenario. "
    "Before you run this scenario, set up scaffold resources by running "
    "'python scaffold.py deploy'."
)
parser.add_argument(
    "role_name",
    help="The name of an IAM role that AWS Glue can assume. This role must
grant access "
    "to Amazon S3 and to the permissions granted by the AWSGlueServiceRole "
    "managed policy.",
)
parser.add_argument(
    "bucket_name",
    help="The name of an S3 bucket that AWS Glue can access to get the job
script and "
    "put job results.",
)
parser.add_argument(
    "--job_script",
    default="flight_etl_job_script.py",
    help="The name of the job script file that is used in the scenario.",
)
return parser.parse_args(args)

def main():
    args = parse_args(sys.argv[1:])
    try:
        print("-" * 88)
        print(
            "Welcome to the AWS Glue getting started with crawlers and jobs
scenario."
        )
        print("-" * 88)
        scenario = GlueCrawlerJobScenario(
            boto3.client("glue"),
```

```

        boto3.resource("iam").Role(args.role_name),
        boto3.resource("s3").Bucket(args.bucket_name),
    )
    scenario.upload_job_script(args.job_script)
    scenario.run(
        "doc-example-crawler",
        "doc-example-database",
        "doc-example-",
        "s3://crawler-public-us-east-1/flight/2016/csv",
        args.job_script,
        "doc-example-job",
    )
    print("-" * 88)
    print(
        "To destroy scaffold resources, including the IAM role and S3 bucket
"
        "used in this scenario, run 'python scaffold.py destroy'."
    )
    print("\nThanks for watching!")
    print("-" * 88)
except Exception:
    logging.exception("Something went wrong with the example.")

```

작업 실행 중에 데이터를 추출, 변환 및 AWS Glue 로드하는 데 사용되는 ETL 스크립트를 만드십시오.

```

import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

"""
These custom arguments must be passed as Arguments to the StartJobRun request.
    --input_database    The name of a metadata database that is contained in
your
                        AWS Glue Data Catalog and that contains tables that
describe
                        the data to be processed.

```

```

--input_table      The name of a table in the database that describes the
data to
                    be processed.
--output_bucket_url An S3 bucket that receives the transformed output data.
"""
args = getResolvedOptions(
    sys.argv, ["JOB_NAME", "input_database", "input_table", "output_bucket_url"]
)
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 Flight Data.
S3FlightData_node1 = glueContext.create_dynamic_frame.from_catalog(
    database=args["input_database"],
    table_name=args["input_table"],
    transformation_ctx="S3FlightData_node1",
)

# This mapping performs two main functions:
# 1. It simplifies the output by removing most of the fields from the data.
# 2. It renames some fields. For example, `fl_date` is renamed to `flight_date`.
ApplyMapping_node2 = ApplyMapping.apply(
    frame=S3FlightData_node1,
    mappings=[
        ("year", "long", "year", "long"),
        ("month", "long", "month", "tinyint"),
        ("day_of_month", "long", "day", "tinyint"),
        ("fl_date", "string", "flight_date", "string"),
        ("carrier", "string", "carrier", "string"),
        ("fl_num", "long", "flight_num", "long"),
        ("origin_city_name", "string", "origin_city_name", "string"),
        ("origin_state_abr", "string", "origin_state_abr", "string"),
        ("dest_city_name", "string", "dest_city_name", "string"),
        ("dest_state_abr", "string", "dest_state_abr", "string"),
        ("dep_time", "long", "departure_time", "long"),
        ("wheels_off", "long", "wheels_off", "long"),
        ("wheels_on", "long", "wheels_on", "long"),
        ("arr_time", "long", "arrival_time", "long"),
        ("mon", "string", "mon", "string"),
    ],
    transformation_ctx="ApplyMapping_node2",

```

```
)  
  
# Script generated for node Revised Flight Data.  
RevisedFlightData_node3 = glueContext.write_dynamic_frame.from_options(  
    frame=ApplyMapping_node2,  
    connection_type="s3",  
    format="json",  
    connection_options={"path": args["output_bucket_url"], "partitionKeys": []},  
    transformation_ctx="RevisedFlightData_node3",  
)  
  
job.commit()
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하십시오.
 - [CreateCrawler](#)
 - [CreateJob](#)
 - [DeleteCrawler](#)
 - [DeleteDatabase](#)
 - [DeleteJob](#)
 - [DeleteTable](#)
 - [GetCrawler](#)
 - [GetDatabase](#)
 - [GetDatabases](#)
 - [GetJob](#)
 - [GetJobRun](#)
 - [GetJobRuns](#)
 - [GetTables](#)
 - [ListJobs](#)
 - [StartCrawler](#)
 - [StartJobRun](#)

Ruby

SDK for Ruby

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

시나리오에서 사용되는 AWS Glue 함수를 래핑하는 클래스를 만드세요.

```
# The `GlueWrapper` class serves as a wrapper around the AWS Glue API, providing
# a simplified interface for common operations.
# It encapsulates the functionality of the AWS SDK for Glue and provides methods
# for interacting with Glue crawlers, databases, tables, jobs, and S3 resources.
# The class initializes with a Glue client and a logger, allowing it to make API
# calls and log any errors or informational messages.
class GlueWrapper
  def initialize(glue_client, logger)
    @glue_client = glue_client
    @logger = logger
  end

  # Retrieves information about a specific crawler.
  #
  # @param name [String] The name of the crawler to retrieve information about.
  # @return [Aws::Glue::Types::Crawler, nil] The crawler object if found, or nil
  # if not found.
  def get_crawler(name)
    @glue_client.get_crawler(name: name)
  rescue Aws::Glue::Errors::EntityNotFoundException
    @logger.info("Crawler #{name} doesn't exist.")
    false
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get crawler #{name}: \n#{e.message}")
    raise
  end

  # Creates a new crawler with the specified configuration.
  #
  # @param name [String] The name of the crawler.
```

```
# @param role_arn [String] The ARN of the IAM role to be used by the crawler.
# @param db_name [String] The name of the database where the crawler stores its
metadata.
# @param db_prefix [String] The prefix to be added to the names of tables that
the crawler creates.
# @param s3_target [String] The S3 path that the crawler will crawl.
# @return [void]
def create_crawler(name, role_arn, db_name, db_prefix, s3_target)
  @glue_client.create_crawler(
    name: name,
    role: role_arn,
    database_name: db_name,
    targets: {
      s3_targets: [
        {
          path: s3_target
        }
      ]
    }
  )
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not create crawler: \n#{e.message}")
  raise
end

# Starts a crawler with the specified name.
#
# @param name [String] The name of the crawler to start.
# @return [void]
def start_crawler(name)
  @glue_client.start_crawler(name: name)
rescue Aws::Glue::Errors::ServiceError => e
  @logger.error("Glue could not start crawler #{name}: \n#{e.message}")
  raise
end

# Deletes a crawler with the specified name.
#
# @param name [String] The name of the crawler to delete.
# @return [void]
def delete_crawler(name)
  @glue_client.delete_crawler(name: name)
rescue Aws::Glue::Errors::ServiceError => e
  @logger.error("Glue could not delete crawler #{name}: \n#{e.message}")
end
```

```
    raise
  end

  # Retrieves information about a specific database.
  #
  # @param name [String] The name of the database to retrieve information about.
  # @return [Aws::Glue::Types::Database, nil] The database object if found, or
  nil if not found.
  def get_database(name)
    response = @glue_client.get_database(name: name)
    response.database
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get database #{name}: \n#{e.message}")
    raise
  end

  # Retrieves a list of tables in the specified database.
  #
  # @param db_name [String] The name of the database to retrieve tables from.
  # @return [Array<Aws::Glue::Types::Table>]
  def get_tables(db_name)
    response = @glue_client.get_tables(database_name: db_name)
    response.table_list
  rescue Aws::Glue::Errors::GlueException => e
    @logger.error("Glue could not get tables #{db_name}: \n#{e.message}")
    raise
  end

  # Creates a new job with the specified configuration.
  #
  # @param name [String] The name of the job.
  # @param description [String] The description of the job.
  # @param role_arn [String] The ARN of the IAM role to be used by the job.
  # @param script_location [String] The location of the ETL script for the job.
  # @return [void]
  def create_job(name, description, role_arn, script_location)
    @glue_client.create_job(
      name: name,
      description: description,
      role: role_arn,
      command: {
        name: "glueetl",
        script_location: script_location,
        python_version: "3"
      }
    )
  end
end
```



```

    },
    glue_version: "3.0"
  )
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not create job #{name}: \n#{e.message}")
  raise
end

# Starts a job run for the specified job.
#
# @param name [String] The name of the job to start the run for.
# @param input_database [String] The name of the input database for the job.
# @param input_table [String] The name of the input table for the job.
# @param output_bucket_name [String] The name of the output S3 bucket for the
job.
# @return [String] The ID of the started job run.
def start_job_run(name, input_database, input_table, output_bucket_name)
  response = @glue_client.start_job_run(
    job_name: name,
    arguments: {
      '--input_database': input_database,
      '--input_table': input_table,
      '--output_bucket_url': "s3://#{output_bucket_name}/"
    }
  )
  response.job_run_id
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not start job run #{name}: \n#{e.message}")
  raise
end

# Retrieves a list of jobs in AWS Glue.
#
# @return [Aws::Glue::Types::ListJobsResponse]
def list_jobs
  @glue_client.list_jobs
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not list jobs: \n#{e.message}")
  raise
end

# Retrieves a list of job runs for the specified job.
#
# @param job_name [String] The name of the job to retrieve job runs for.

```

```
# @return [Array<Aws::Glue::Types::JobRun>]
def get_job_runs(job_name)
  response = @glue_client.get_job_runs(job_name: job_name)
  response.job_runs
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not get job runs: \n#{e.message}")
end

# Retrieves data for a specific job run.
#
# @param job_name [String] The name of the job run to retrieve data for.
# @return [Glue::Types::GetJobRunResponse]
def get_job_run(job_name, run_id)
  @glue_client.get_job_run(job_name: job_name, run_id: run_id)
rescue Aws::Glue::Errors::GlueException => e
  @logger.error("Glue could not get job runs: \n#{e.message}")
end

# Deletes a job with the specified name.
#
# @param job_name [String] The name of the job to delete.
# @return [void]
def delete_job(job_name)
  @glue_client.delete_job(job_name: job_name)
rescue Aws::Glue::Errors::ServiceError => e
  @logger.error("Glue could not delete job: \n#{e.message}")
end

# Deletes a table with the specified name.
#
# @param database_name [String] The name of the catalog database in which the
table resides.
# @param table_name [String] The name of the table to be deleted.
# @return [void]
def delete_table(database_name, table_name)
  @glue_client.delete_table(database_name: database_name, name: table_name)
rescue Aws::Glue::Errors::ServiceError => e
  @logger.error("Glue could not delete job: \n#{e.message}")
end

# Removes a specified database from a Data Catalog.
#
# @param database_name [String] The name of the database to delete.
# @return [void]
```

```

def delete_database(database_name)
  @glue_client.delete_database(name: database_name)
rescue Aws::Glue::Errors::ServiceError => e
  @logger.error("Glue could not delete database: \n#{e.message}")
end

# Uploads a job script file to an S3 bucket.
#
# @param file_path [String] The local path of the job script file.
# @param bucket_resource [Aws::S3::Bucket] The S3 bucket resource to upload the
file to.
# @return [void]
def upload_job_script(file_path, bucket_resource)
  File.open(file_path) do |file|
    bucket_resource.client.put_object({
      body: file,
      bucket: bucket_resource.name,
      key: file_path
    })
  end
rescue Aws::S3::Errors::S3UploadFailedError => e
  @logger.error("S3 could not upload job script: \n#{e.message}")
  raise
end

end

```

시나리오를 실행하는 클래스를 생성합니다.

```

class GlueCrawlerJobScenario
  def initialize(glue_client, glue_service_role, glue_bucket, logger)
    @glue_client = glue_client
    @glue_service_role = glue_service_role
    @glue_bucket = glue_bucket
    @logger = logger
  end

  def run(crawler_name, db_name, db_prefix, data_source, job_script, job_name)
    wrapper = GlueWrapper.new(@glue_client, @logger)

    new_step(1, "Create a crawler")
    puts "Checking for crawler #{crawler_name}."
  end
end

```

```
crawler = wrapper.get_crawler(crawler_name)
if crawler == false
  puts "Creating crawler #{crawler_name}."
  wrapper.create_crawler(crawler_name, @glue_service_role.arn, db_name,
db_prefix, data_source)
  puts "Successfully created #{crawler_name}:"
  crawler = wrapper.get_crawler(crawler_name)
  puts JSON.pretty_generate(crawler).yellow
end
print "\nDone!\n".green

new_step(2, "Run a crawler to output a database.")
puts "Location of input data analyzed by crawler: #{data_source}"
puts "Outputs: a Data Catalog database in CSV format containing metadata on
input."
wrapper.start_crawler(crawler_name)
puts "Starting crawler... (this typically takes a few minutes)"
crawler_state = nil
while crawler_state != "READY"
  custom_wait(15)
  crawler = wrapper.get_crawler(crawler_name)
  crawler_state = crawler[0]["state"]
  print "Status check: #{crawler_state}.".yellow
end
print "\nDone!\n".green

new_step(3, "Query the database.")
database = wrapper.get_database(db_name)
puts "The crawler created database #{db_name}:"
print "#{database}.yellow
puts "\nThe database contains these tables:"
tables = wrapper.get_tables(db_name)
tables.each_with_index do |table, index|
  print "\t#{index + 1}. #{table['name']}".yellow
end
print "\nDone!\n".green

new_step(4, "Create a job definition that runs an ETL script.")
puts "Uploading Python ETL script to S3..."
wrapper.upload_job_script(job_script, @glue_bucket)
puts "Creating job definition #{job_name}:\n"
response = wrapper.create_job(job_name, "Getting started example job.",
@glue_service_role.arn, "s3://#{@glue_bucket.name}/#{job_script}")
puts JSON.pretty_generate(response).yellow
```

```
print "\nDone!\n".green

new_step(5, "Start a new job")
job_run_status = nil
job_run_id = wrapper.start_job_run(
  job_name,
  db_name,
  tables[0]["name"],
  @glue_bucket.name
)
puts "Job #{job_name} started. Let's wait for it to run."
until ["SUCCEEDED", "STOPPED", "FAILED", "TIMEOUT"].include?(job_run_status)
  custom_wait(10)
  job_run = wrapper.get_job_runs(job_name)
  job_run_status = job_run[0]["job_run_state"]
  print "Status check: #{job_name}/#{job_run_id} - #{job_run_status}.".yellow
end
print "\nDone!\n".green

new_step(6, "View results from a successful job run.")
if job_run_status == "SUCCEEDED"
  puts "Data from your job run is stored in your S3 bucket
'#{@glue_bucket.name}'. Files include:"
  begin

    # Print the key name of each object in the bucket.
    @glue_bucket.objects.each do |object_summary|
      if object_summary.key.include?("run-")
        print "#{object_summary.key}".yellow
      end
    end

    # Print the first 256 bytes of a run file
    desired_sample_objects = 1
    @glue_bucket.objects.each do |object_summary|
      if object_summary.key.include?("run-")
        if desired_sample_objects > 0
          sample_object = @glue_bucket.object(object_summary.key)
          sample = sample_object.get(range: "bytes=0-255").body.read
          puts "\nSample run file contents:"
          print "#{sample}".yellow
          desired_sample_objects -= 1
        end
      end
    end
  end
end
```

```

    end
  rescue Aws::S3::Errors::ServiceError => e
    logger.error(
      "Couldn't get job run data. Here's why: %s: %s",
      e.response.error.code, e.response.error.message
    )
    raise
  end
end
end
print "\nDone!\n".green

new_step(7, "Delete job definition and crawler.")
wrapper.delete_job(job_name)
puts "Job deleted: #{job_name}."
wrapper.delete_crawler(crawler_name)
puts "Crawler deleted: #{crawler_name}."
wrapper.delete_table(db_name, tables[0]["name"])
puts "Table deleted: #{tables[0]["name"]} in #{db_name}."
wrapper.delete_database(db_name)
puts "Database deleted: #{db_name}."
print "\nDone!\n".green
end
end

def main

  banner("../helpers/banner.txt")
  puts
  "#####"
  puts "#
                                     #".yellow
  puts "#                               EXAMPLE CODE DEMO:
                                     #".yellow
  puts "#                               AWS Glue
                                     #".yellow
  puts "#
                                     #".yellow
  puts
  "#####"
  puts ""
  puts "You have launched a demo of AWS Glue using the AWS for Ruby v3 SDK. Over
the next 60 seconds, it will"
  puts "do the following:"
  puts "  1. Create a crawler."

```

```
puts "    2. Run a crawler to output a database."
puts "    3. Query the database."
puts "    4. Create a job definition that runs an ETL script."
puts "    5. Start a new job."
puts "    6. View results from a successful job run."
puts "    7. Delete job definition and crawler."
puts ""

confirm_begin
billing
security
puts "\e[H\e[2J"

# Set input file names
job_script_filepath = "job_script.py"
resource_names = YAML.load_file("resource_names.yaml")

# Instantiate existing IAM role.
iam = Aws::IAM::Resource.new(region: "us-east-1")
iam_role_name = resource_names["glue_service_role"]
iam_role = iam.role(iam_role_name)

# Instantiate existing S3 bucket.
s3 = Aws::S3::Resource.new(region: "us-east-1")
s3_bucket_name = resource_names["glue_bucket"]
s3_bucket = s3.bucket(s3_bucket_name)

scenario = GlueCrawlerJobScenario.new(
  Aws::Glue::Client.new(region: "us-east-1"),
  iam_role,
  s3_bucket,
  @logger
)

random_int = rand(10 ** 4)
scenario.run(
  "doc-example-crawler-#{random_int}",
  "doc-example-database-#{random_int}",
  "doc-example-#{random_int}-",
  "s3://crawler-public-us-east-1/flight/2016/csv",
  job_script_filepath,
  "doc-example-job-#{random_int}"
)
```

```
puts "-" * 88
puts "You have reached the end of this tour of AWS Glue."
puts "To destroy CDK-created resources, run:\n      cdk destroy"
puts "-" * 88

end
```

작업 실행 중에 데이터를 추출, 변환 및 AWS Glue 로드하는 데 사용되는 ETL 스크립트를 만드십시오.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

"""
These custom arguments must be passed as Arguments to the StartJobRun request.
  --input_database    The name of a metadata database that is contained in
your
                        AWS Glue Data Catalog and that contains tables that
describe
                        the data to be processed.
  --input_table       The name of a table in the database that describes the
data to
                        be processed.
  --output_bucket_url An S3 bucket that receives the transformed output data.
"""
args = getResolvedOptions(
    sys.argv, ["JOB_NAME", "input_database", "input_table", "output_bucket_url"]
)
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

# Script generated for node S3 Flight Data.
S3FlightData_node1 = glueContext.create_dynamic_frame.from_catalog(
    database=args["input_database"],
    table_name=args["input_table"],
```



```

    transformation_ctx="S3FlightData_node1",
  )

# This mapping performs two main functions:
# 1. It simplifies the output by removing most of the fields from the data.
# 2. It renames some fields. For example, `fl_date` is renamed to `flight_date`.
ApplyMapping_node2 = ApplyMapping.apply(
  frame=S3FlightData_node1,
  mappings=[
    ("year", "long", "year", "long"),
    ("month", "long", "month", "tinyint"),
    ("day_of_month", "long", "day", "tinyint"),
    ("fl_date", "string", "flight_date", "string"),
    ("carrier", "string", "carrier", "string"),
    ("fl_num", "long", "flight_num", "long"),
    ("origin_city_name", "string", "origin_city_name", "string"),
    ("origin_state_abr", "string", "origin_state_abr", "string"),
    ("dest_city_name", "string", "dest_city_name", "string"),
    ("dest_state_abr", "string", "dest_state_abr", "string"),
    ("dep_time", "long", "departure_time", "long"),
    ("wheels_off", "long", "wheels_off", "long"),
    ("wheels_on", "long", "wheels_on", "long"),
    ("arr_time", "long", "arrival_time", "long"),
    ("mon", "string", "mon", "string"),
  ],
  transformation_ctx="ApplyMapping_node2",
)

# Script generated for node Revised Flight Data.
RevisedFlightData_node3 = glueContext.write_dynamic_frame.from_options(
  frame=ApplyMapping_node2,
  connection_type="s3",
  format="json",
  connection_options={"path": args["output_bucket_url"], "partitionKeys": []},
  transformation_ctx="RevisedFlightData_node3",
)

job.commit()

```

- API 세부 정보는 AWS SDK for Ruby API 참조의 다음 주제를 참조하십시오.
 - [CreateCrawler](#)
 - [CreateJob](#)

- [DeleteCrawler](#)
- [DeleteDatabase](#)
- [DeleteJob](#)
- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)
- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

Rust

SDK for Rust

Note

자세한 내용은 여기에서 확인할 수 있습니다. GitHub [AWS 코드 예제 리포지토리](#)에서 전체 예제를 찾고 설정 및 실행하는 방법을 배우보세요.

공용 Amazon Simple Storage Service (S3) 버킷을 크롤링하고 검색한 CSV 형식의 데이터를 설명하는 메타데이터 데이터베이스를 생성하는 크롤러를 만들고 실행합니다.

```
let create_crawler = glue
    .create_crawler()
    .name(self.crawler())
    .database_name(self.database())
    .role(self.iam_role.expose_secret())
    .targets(
        CrawlerTargets::builder()
            .s3_targets(S3Target::builder().path(CRAWLER_TARGET).build())
```

```

        .build(),
    )
    .send()
    .await;

match create_crawler {
    Err(err) => {
        let glue_err: aws_sdk_glue::Error = err.into();
        match glue_err {
            aws_sdk_glue::Error::AlreadyExistsException(_) => {
                info!("Using existing crawler");
                Ok(())
            }
            _ => Err(GlueMvpError::GlueSdk(glue_err)),
        }
    }
    Ok(_) => Ok(()),
}??;

let start_crawler =
glue.start_crawler().name(self.crawler()).send().await;

match start_crawler {
    Ok(_) => Ok(()),
    Err(err) => {
        let glue_err: aws_sdk_glue::Error = err.into();
        match glue_err {
            aws_sdk_glue::Error::CrawlerRunningException(_) => Ok(()),
            _ => Err(GlueMvpError::GlueSdk(glue_err)),
        }
    }
}??;

```

내 데이터베이스 및 테이블에 대한 정보를 나열하세요 AWS Glue Data Catalog.

```

let database = glue
    .get_database()
    .name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?
    .to_owned();

```

```

let database = database
    .database()
    .ok_or_else(|| GlueMvpError::Unknown("Could not find
database".into()))?;

let tables = glue
    .get_tables()
    .database_name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

let tables = tables.table_list();

```

소스 Amazon S3 버킷에서 CSV 데이터를 추출하고, 필드를 제거하고 이름을 변경하여 변환하고, JSON 형식의 출력을 다른 Amazon S3 버킷으로 로드하는 작업을 만들고 실행합니다.

```

let create_job = glue
    .create_job()
    .name(self.job())
    .role(self.iam_role.expose_secret())
    .command(
        JobCommand::builder()
            .name("glueetl")
            .python_version("3")
            .script_location(format!("s3://{}/job.py", self.bucket()))
            .build(),
    )
    .glue_version("3.0")
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

let job_name = create_job.name().ok_or_else(|| {
    GlueMvpError::Unknown("Did not get job name after creating
job".into())
})?;

let job_run_output = glue
    .start_job_run()
    .job_name(self.job())
    .arguments("--input_database", self.database())

```

```

        .arguments(
            "--input_table",
            self.tables
                .first()
                .ok_or_else(|| GlueMvpError::Unknown("Missing crawler
table".into()))?
                .name(),
        )
        .arguments("--output_bucket_url", self.bucket())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;

let job = job_run_output
    .job_run_id()
    .ok_or_else(|| GlueMvpError::Unknown("Missing run id from just
started job".into()))?
    .to_string();

```

데모 중에 생성된 모든 리소스를 삭제합니다.

```

glue.delete_job()
    .job_name(self.job())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

for t in &self.tables {
    glue.delete_table()
        .name(t.name())
        .database_name(self.database())
        .send()
        .await
        .map_err(GlueMvpError::from_glue_sdk)?;
}

glue.delete_database()
    .name(self.database())
    .send()
    .await
    .map_err(GlueMvpError::from_glue_sdk)?;

```

```
glue.delete_crawler()  
    .name(self.crawler())  
    .send()  
    .await  
    .map_err(GlueMvpError::from_glue_sdk)?;
```

- API 세부 정보는 AWS SDK for Rust API 참조의 다음 주제를 참조하십시오.

- [CreateCrawler](#)
- [CreateJob](#)
- [DeleteCrawler](#)
- [DeleteDatabase](#)
- [DeleteJob](#)
- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)
- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 [여기](#)를 참조하십시오. [AWS SDK와 함께 이 서비스 사용](#). 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS Glue의 보안

AWS는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다.

보안은 AWS와 사용자의 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS클라우드에서 AWS서비스를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사원은 정기적으로 [AWS 규제 준수 프로그램](#)의 일환으로 보안 효과를 테스트하고 검증합니다. AWS Glue에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 [규정 준수 프로그램 제공 범위 내의 AWS 서비스](#)를 참조하십시오.
- 클라우드 내 보안 - 귀하의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터의 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 AWS Glue 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 AWS Glue을(를) 구성하는 방법을 보여줍니다. 또한 AWS Glue 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스 사용 방법을 알아봅니다.

주제

- [AWS Glue의 데이터 보호](#)
- [AWS Glue의 ID 및 액세스 관리](#)
- [AWS Glue의 로깅 및 모니터링](#)
- [에 대한 규정 준수 검증 AWS Glue](#)
- [의 탄력성 AWS Glue](#)
- [AWS Glue에서 인프라 보안](#)

AWS Glue의 데이터 보호

AWS Glue는 데이터를 보호하기 위해 설계된 여러 기능을 제공합니다.

주제

- [저장 중 암호화](#)
- [전송 중 암호화](#)
- [FIPS 규정 준수](#)
- [키 관리](#)
- [기타 AWS 서비스에서 AWS Glue 종속성](#)
- [개발 엔드포인트](#)

저장 중 암호화

AWS Glue는 [AWS Glue Studio](#)를 사용하여 시각적 ETL 작업 구축 및 [개발 엔드포인트](#)를 사용하여 [스크립트 개발](#)에 대한 미사용 데이터 암호화를 지원합니다. [AWS Key Management Service\(AWS KMS\)](#) 키를 사용하여 암호화된 미사용 데이터를 기록하도록 ETL(추출, 변환 및 로드) 작업 및 개발 엔드포인트를 구성할 수 있습니다. AWS KMS로 관리하는 키를 사용하여 [AWS Glue Data Catalog](#)에 저장된 메타데이터도 암호화할 수 있습니다. 또한 AWS KMS 키를 사용하여 [크롤러](#) 및 ETL 작업에서 생성된 로그와 작업 북마크를 암호화할 수 있습니다.

작업, 크롤러 및 개발 엔드포인트를 통해 Amazon Simple Storage Service(Amazon S3)와 Amazon CloudWatch Logs에 기록된 데이터 외에 AWS Glue Data Catalog의 메타데이터 객체도 암호화할 수 있습니다. AWS Glue에 작업, 크롤러 및 개발 엔드포인트를 만들 때 보안 구성을 연결하여 암호화 설정을 제공할 수 있습니다. 보안 구성에는 AWS KMS(SSE-KMS)에 저장된 고객 마스터 키(CMK)나 Amazon S3 관리형 서버 측 암호화 키(SSE-S3)가 포함됩니다. AWS Glue 콘솔에서 보안 구성을 만들 수 있습니다.

계정에서 전체 데이터 카탈로그에 대한 암호화를 설정할 수도 있습니다. 이렇게 하려면 AWS KMS에 저장된 CMK를 지정합니다.

Important

AWS Glue에서는 대칭 고객 관리형 키만 지원합니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 [고객 관리형 키\(CMK\)](#)를 참조하세요.

암호화를 설정한 상태에서 데이터 카탈로그 객체를 추가하거나, 크롤러를 실행하거나, 작업을 실행하거나, 개발 엔드포인트를 시작할 때 SSE-S3 또는 SSE-KMS 키를 사용하여 유휴 데이터를 씁니다. 그리고 신뢰할 수 있는 전송 계층 보안(TLS) 프로토콜을 통해서만 Java 데이터베이스 연결(JDBC) 데이터 스토어에 액세스하도록 AWS Glue를 구성할 수 있습니다.

AWS Glue에서 다음 위치의 암호화 설정을 제어합니다.

- 데이터 카탈로그의 설정입니다.
- 사용자가 생성하는 보안 구성입니다.
- 사용자의 AWS Glue ETL(추출, 변환 및 로드) 작업에 파라미터로 전달되는 서버 측 암호화 설정 (SSE-S3 또는 SSE-KMS)입니다.

암호화를 설정하는 방법에 대한 자세한 내용은 [AWS Glue에서 암호화 설정](#) 단원을 참조하십시오.

주제

- [데이터 카탈로그 암호화](#)
- [연결 암호 암호화](#)
- [AWS Glue에서 작성한 데이터 암호화](#)

데이터 카탈로그 암호화

AWS Glue Data Catalog 암호화는 민감한 데이터에 대한 보안을 강화합니다. AWS Glue는 AWS Key Management Service(AWS KMS)와 통합되어 데이터 카탈로그에 저장된 메타데이터를 암호화합니다. AWS Glue 콘솔 또는 AWS CLI를 사용하여 데이터 카탈로그의 리소스에 대한 암호화 설정을 활성화하거나 비활성화할 수 있습니다.

데이터 카탈로그의 암호화를 활성화하면 새로 만드는 모든 객체가 암호화됩니다. 암호화를 비활성화하면 새로 만든 객체는 암호화되지 않지만 기존의 암호화된 객체는 암호화된 상태로 유지됩니다.

AWS 관리형 암호화 키 또는 고객 관리 암호화 키를 사용하여 전체 데이터 카탈로그를 암호화할 수 있습니다. 키 유형 및 상태에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 [AWS Key Management Service 개념](#)을 참조하세요.

AWS 관리형 키

AWS 관리형 키는 AWS KMS와 통합된 AWS 서비스가 고객의 계정에서 고객 대신 생성하고 관리하고 사용하는 KMS 키입니다. 계정에서 AWS 관리형 키를 확인하고 키 정책을 확인하며 AWS CloudTrail 로그에서의 사용을 감사할 수 있습니다. 하지만 이러한 키를 관리하거나 이들의 권한을 변경할 수는 없습니다.

저장된 암호화는 메타데이터를 암호화하는 데 사용되는 AWS Glue에 대한 AWS 관리형 키를 관리하기 위해 AWS KMS와 자동으로 통합됩니다. 메타데이터 암호화를 활성화할 때 AWS 관리형 키가 없는 경우 AWS KMS가 자동으로 새 키를 생성합니다.

자세한 내용은 [AWS 관리형 키](#)를 참조하세요.

고객 관리형 키

고객 관리형 키는 사용자가 생성, 소유 및 관리하는 AWS 계정의 KMS 키입니다. 이러한 KMS 키를 완전히 제어할 수 있습니다. 다음을 할 수 있습니다.

- 키 정책, IAM 정책, 권한 부여 설정 및 관리
- 활성화 및 비활성화
- 암호화 자료 교체
- 태그 추가
- 이를 참조하는 별칭 생성
- 이를 삭제하도록 예약

고객 관리형 키의 권한 관리에 대한 자세한 내용은 [고객 관리형 키](#)를 참조하세요.

Important

AWS Glue에서는 대칭 고객 관리형 키만 지원합니다. KMS 키 목록에는 대칭 키만 표시됩니다. 그러나 Choose a KMS key ARN(KMS 키 ARN 선택)을 선택하면 콘솔에서 모든 키 유형에 대해 ARN을 입력할 수 있습니다. 대칭 키에 대한 ARN만 입력해야 합니다.

대칭 고객 관리형 키를 생성하려면 AWS Key Management Service 개발자 안내서의 [대칭 고객 관리형 키 생성](#) 단계를 따르세요.

저장된 데이터 카탈로그 암호화를 활성화하면 다음 리소스 유형이 KMS 키를 사용하여 암호화됩니다.

- 데이터베이스 수
- 표
- 파티션
- 테이블 버전
- 열 통계값
- 사용자 정의 함수
- 데이터 카탈로그 보기

AWS Glue 암호화 컨텍스트

암호화 컨텍스트는 데이터에 대한 추가 컨텍스트 정보를 포함하는 키-값 페어의 선택적 집합입니다. AWS KMS는 암호화 컨텍스트를 **인증된 암호화**를 지원하기 위한 **추가 인증 데이터**로 사용합니다. 데이터 암호화 요청에 암호화 컨텍스트를 포함하는 경우, AWS KMS는(는) 암호화된 데이터에 암호화 컨텍스트를 바인딩합니다. 데이터의 암호를 해독하려면 요청에 동일한 암호화 컨텍스트를 포함합니다. AWS Glue는 키가 glue_catalog_id이고 값이 catalogId인 모든 AWS KMS 암호화 연산에서 동일한 암호화 컨텍스트를 사용합니다.

```
"encryptionContext": {
  "glue_catalog_id": "111122223333"
}
```

AWS 관리형 키 또는 대칭 고객 관리형 키를 사용하여 데이터 카탈로그를 암호화하는 경우 감사 기록 및 로그의 암호화 컨텍스트를 사용하여 키가 어떻게 사용되고 있는지 파악할 수도 있습니다. 암호화 컨텍스트는 AWS CloudTrail 또는 Amazon CloudWatch 로그에서 생성되는 로그에도 나타납니다.

암호화 활성화

AWS Glue 콘솔의 데이터 카탈로그 설정에서 또는 AWS CLI를 사용하여 AWS Glue Data Catalog 객체에 대한 암호화를 활성화할 수 있습니다.

Console

콘솔을 사용하여 암호화 활성화

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창의 데이터 카탈로그를 선택합니다.
3. 데이터 카탈로그 설정 페이지에서 메타데이터 암호화 확인란을 선택하고 AWS KMS 키를 선택합니다.

암호화를 사용 설정할 때 고객 관리 키를 지정하지 않으면 암호화 설정에서 AWS 관리형 KMS 키를 사용합니다.

4. (선택 사항) 고객 관리형 키를 사용하여 데이터 카탈로그를 암호화하는 경우, 데이터 카탈로그는 리소스를 암호화 및 해독하기 위해 IAM 역할을 등록할 수 있는 옵션을 제공합니다. AWS Glue가 사용자를 대신하여 수행할 수 있는 IAM 역할 권한을 부여해야 합니다. 여기에는 데이터 암호화 및 암호 해독을 위한 AWS KMS 권한이 포함됩니다.

데이터 카탈로그에서 새 리소스를 생성하면 AWS Glue가 데이터 암호화를 위해 제공된 IAM 역할을 말합니다. 마찬가지로 소비자가 리소스에 액세스할 때 AWS Glue가 데이터 암호 해독을 위해 IAM 역할을 말합니다. 필요한 권한이 있는 IAM 역할을 등록하면 호출 주체는 더 이상 키에 액세스하고 데이터를 해독하는 데 권한이 필요하지 않습니다.

⚠ Important

고객 관리형 키를 사용하여 데이터 카탈로그 리소스를 암호화하는 경우에만 KMS 작업을 IAM 역할에 위임할 수 있습니다. 현재 KMS 역할 위임 기능은 데이터 카탈로그 리소스를 암호화하는 데 AWS 관리형 키를 사용하는 것을 지원하지 않습니다.

⚠ Warning

IAM 역할이 KMS 작업을 위임하도록 설정하면 이전에 AWS 관리형 키를 사용하여 암호화되었던 데이터 카탈로그 리소스에 더 이상 액세스할 수 없습니다.

- a. AWS Glue가 사용자를 대신하여 데이터를 암호화하고 암호를 해독할 수 있는 IAM 역할을 활성화하려면 KMS 작업을 IAM 역할에 위임 옵션을 선택합니다.
- b. 다음으로 IAM 역할을 선택합니다.

IAM 역할을 생성하려면 [AWS Glue의 IAM 역할 생성](#)을 참조하십시오.

AWS Glue가 데이터 카탈로그에 액세스하기 위해 가정하는 IAM 역할에는 데이터 카탈로그의 메타데이터를 암호화 및 암호 해독을 위한 권한이 있어야 합니다. IAM 역할을 생성하여 다음 인라인 정책을 연결할 수 있습니다.

- 다음 정책을 추가하여 데이터 카탈로그의 암호화 및 암호 해독을 위한 AWS KMS 권한을 포함하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
```

```

        "kms:Encrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "arn:aws:kms:<region>:<account-id>:key/<key-id>"
}
]
}

```

- 다음으로 AWS Glue 서비스에 대한 역할에 다음 신뢰 정책을 추가하여 IAM 역할을 위임합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "glue.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}

```

- 그런 다음 IAM 역할에 iam:PassRole 권한을 추가합니다.

```

    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "iam:PassRole"
          ],
          "Resource": [
            "arn:aws:iam::<account-id>:role/<encryption-role-name>"
          ]
        }
      ]
    }
}

```

암호화를 사용 설정할 때 AWS Glue가 맡을 IAM 역할을 지정하지 않은 경우 데이터 카탈로그에 액세스하는 주체는 다음 API 작업을 수행할 수 있는 권한이 있어야 합니다.

- kms:Decrypt
- kms:Encrypt
- kms:GenerateDataKey

AWS CLI

SDK 또는 AWS CLI을 사용하여 암호화 활성화

- PutDataCatalogEncryptionSettings API 작업을 사용합니다. 키를 지정하지 않으면 AWS Glue는 고객 계정에 대해 AWS 관리형 암호화 키를 사용하여 데이터 카탈로그를 암호화합니다.

```
aws glue put-data-catalog-encryption-settings \
  --data-catalog-encryption-settings '{
    "EncryptionAtRest": {
      "CatalogEncryptionMode": "SSE-KMS-WITH-SERVICE-ROLE",
      "SseAwsKmsKeyId": "arn:aws:kms:<region>:<account-id>:key/<key-id>",
      "CatalogEncryptionServiceRole": "arn:aws:iam::<account-
id>:role/<encryption-role-name>"
    }
  }'
```

암호화를 활성화하면 데이터 카탈로그 객체에서 생성하는 모든 객체가 암호화됩니다. 이 설정을 삭제하면 데이터 카탈로그에서 생성하는 객체가 더 이상 암호화되지 않습니다. 필요한 KMS 권한을 사용하여 데이터 카탈로그의 기존 암호화된 객체에 계속 액세스할 수 있습니다.

Important

AWS KMS 키는 데이터 카탈로그에 함께 암호화된 객체의 AWS KMS 키 스토어에서 계속 사용할 수 있어야 합니다. 키를 제거하면 객체의 암호를 해독할 수 없습니다. 일부 시나리오

에서는 데이터 카탈로그 메타데이터에 대한 액세스를 방지하기 위해 이것이 필요할 수 있습니다.

AWS Glue에 대한 KMS 키 모니터링

데이터 카탈로그 리소스와 함께 KMS 키를 사용하는 경우 AWS CloudTrail 또는 Amazon CloudWatch 로그를 사용하여 AWS Glue가 AWS KMS로 보내는 요청을 추적할 수 있습니다. AWS CloudTrail은 AWS Glue가 KMS 키로 암호화된 데이터에 액세스하기 위해 호출하는 KMS 작업을 모니터링하고 기록합니다.

다음 예제는 Decrypt 및 GenerateDataKey 작업에 대한 AWS CloudTrail 이벤트입니다.

Decrypt

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAXPHTESTANDEXAMPLE:Sampleuser01",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAXPHTESTANDEXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2024-01-10T14:33:56Z",
        "mfaAuthenticated": "false"
      }
    },
    "invokedBy": "glue.amazonaws.com"
  },
  "eventTime": "2024-01-10T15:18:11Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
```

```

"awsRegion": "eu-west-2",
"sourceIPAddress": "glue.amazonaws.com",
"userAgent": "glue.amazonaws.com",
"requestParameters": {
  "encryptionContext": {
    "glue_catalog_id": "111122223333"
  },
  "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
},
"responseElements": null,
"requestID": "43b019aa-34b8-4798-9b98-ee968b2d63df",
"eventID": "d7614763-d3fe-4f84-a1e1-3ca4d2a5bbd5",
"readOnly": true,
"resources": [
  {
    "accountId": "111122223333",
    "type": "AWS::KMS::Key",
    "ARN": "arn:aws:kms:<region>:111122223333:key/<key-id>"
  }
],
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "111122223333",
"eventCategory": "Management",
"sessionCredentialFromConsole": "true"
}

```

GenerateDataKey

```

{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId":
"AROAXPHTESTANDEXAMPLE:V_00_GLUE_KMS_GENERATE_DATA_KEY_111122223333",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/
V_00_GLUE_KMS_GENERATE_DATA_KEY_111122223333",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",

```



```

        "principalId": "AROAXPHTESTANDEXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Admin"
    },
    "webIdFederationData": {},
    "attributes": {
        "creationDate": "2024-01-05T21:15:47Z",
        "mfaAuthenticated": "false"
    }
},
"invokedBy": "glue.amazonaws.com"
},
"eventTime": "2024-01-05T21:15:47Z",
"eventSource": "kms.amazonaws.com",
"eventName": "GenerateDataKey",
"awsRegion": "eu-west-2",
"sourceIPAddress": "glue.amazonaws.com",
"userAgent": "glue.amazonaws.com",
"requestParameters": {
    "keyId": "arn:aws:kms:eu-west-2:AKIAIOSFODNN7EXAMPLE:key/
AKIAIOSFODNN7EXAMPLE",
    "encryptionContext": {
        "glue_catalog_id": "111122223333"
    },
    "keySpec": "AES_256"
},
"responseElements": null,
"requestID": "64d1783a-4b62-44ba-b0ab-388b50188070",
"eventID": "1c73689b-2ef2-443b-aed7-8c126585ca5e",
"readOnly": true,
"resources": [
    {
        "accountId": "111122223333",
        "type": "AWS::KMS::Key",
        "ARN": "arn:aws:kms:eu-west-2:111122223333:key/AKIAIOSFODNN7EXAMPLE"
    }
],
"eventType": "AwsApiCall",
"managementEvent": true,
"recipientAccountId": "111122223333",
"eventCategory": "Management"
}

```

연결 암호 암호화

GetConnection 및 GetConnections API 작업을 사용하여 AWS Glue Data Catalog에서 연결 암호를 검색할 수 있습니다. 이러한 암호는 데이터 카탈로그 연결에 저장되고 AWS Glue가 Java Database Connectivity(JDBC) 데이터 스토어에 연결할 때 사용됩니다. 연결이 생성되거나 업데이트되면 데이터 카탈로그 설정의 옵션이 암호가 암호화되었는지 여부를 확인하고, 암호화된 경우 지정된 AWS Key Management Service(AWS KMS) 키를 확인합니다.

AWS Glue 콘솔의 [데이터 카탈로그 설정(Data catalog settings)] 페이지에서 이 옵션을 설정할 수 있습니다.

연결 암호를 암호화하려면

1. AWS Management Console에 로그인하여 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 설정을 선택합니다.
3. 데이터 카탈로그 설정 페이지에서 Encrypt connection passwords(연결 암호 암호화)를 선택하고 AWS KMS 키를 선택합니다.

Important

AWS Glue에서는 대칭 고객 마스터 키(CMK)만 지원합니다. AWS KMS 키(key) 목록에는 대칭 키만 표시됩니다. 그러나 Choose a AWS KMS key ARN(KMS 키 ARN 선택)을 선택하면 콘솔에서 모든 키 유형의 ARN을 입력할 수 있습니다. 대칭 키에 대한 ARN만 입력해야 합니다.

자세한 내용은 [데이터 카탈로그 설정](#) 단원을 참조하십시오.

AWS Glue에서 작성한 데이터 암호화

보안 구성은 AWS Glue에서 사용할 수 있는 보안 속성의 집합입니다. 보안 구성을 사용하여 미사용 데이터를 암호화할 수 있습니다. 다음 시나리오는 보안 구성을 사용할 수 있는 몇 가지 방법을 보여줍니다.

- 암호화된 Amazon CloudWatch Logs를 기록할 AWS Glue 크롤러에 보안 구성을 연결합니다. 크롤러에 보안 구성을 첨부하는 방법에 대한 자세한 내용은 [the section called “3단계: 보안 설정 구성”](#) 섹션을 참조하세요.
- 암호화된 Amazon Simple Storage Service(Amazon S3) 대상과 암호화된 CloudWatch Logs를 기록할 추출, 변환 및 로드(ETL) 작업에 보안 구성을 연결합니다.
- 암호화된 Amazon S3 데이터로 작업 북마크를 기록할 ETL 작업에 보안 구성을 연결합니다.
- 암호화된 Amazon S3 대상을 기록할 개발 엔드포인트에 보안 구성을 연결합니다.

Important

현재 보안 구성은 ETL 작업 파라미터로 전달되는 서버 측 암호화(SSE-S3)를 재정의합니다. 따라서 보안 구성과 SSE-S3 파라미터가 모두 작업과 연결된 경우 SSE-S3 파라미터는 무시됩니다.

보안 구성에 대한 자세한 내용은 [AWS Glue 콘솔의 보안 구성 관련 작업](#) 단원을 참조하세요.

주제

- [보안 구성을 사용하도록 AWS Glue 설정](#)
- [VPC 작업 및 크롤러용 AWS KMS로 가는 경로 생성](#)
- [AWS Glue 콘솔의 보안 구성 관련 작업](#)

보안 구성을 사용하도록 AWS Glue 설정

다음 단계에 따라 보안 구성을 사용하도록 AWS Glue 환경을 설정합니다.

1. AWS Key Management Service(AWS KMS) 키를 생성하거나 업데이트하여 CloudWatch Logs를 암호화할 AWS Glue 크롤러 및 작업에 전달되는 IAM 역할에 AWS KMS 권한을 부여합니다. 자세한 내용은 Amazon CloudWatch Logs User Guide의 [Encrypt Log Data in CloudWatch Logs Using AWS KMS](#)를 참조하세요.

다음 예에서 `"role1"`, `"role2"`, `"role3"`은 크롤러와 작업에 전달되는 IAM 역할입니다.

```
{
  "Effect": "Allow",
  "Principal": { "Service": "logs.region.amazonaws.com",
  "AWS": [
    "role1",
    "role2",
    "role3"
  ] },
  "Action": [
    "kms:Encrypt*",
    "kms:Decrypt*",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:Describe*"
  ],
  "Resource": "*"
}
```

키를 사용하여 CloudWatch Logs를 암호화하는 경우 `"Service": "logs.region.amazonaws.com"`과 같이 Service 문이 필요합니다.

2. 사용 전에 AWS KMS 키가 ENABLED 상태인지 확인합니다.

Note

Iceberg를 데이터 레이크 프레임워크로 사용하는 경우 Iceberg 테이블에는 서버 측 암호화를 활성화하는 자체 메커니즘이 있습니다. AWS Glue의 보안 구성과 함께 이러한 구성을 활성화해야 합니다. Iceberg 테이블에서 서버 측 암호화를 활성화하려면 [Iceberg 설명서](#)의 지침을 검토하세요.

VPC 작업 및 크롤러용 AWS KMS로 가는 경로 생성

인터넷을 통해 연결하지 않고 Virtual Private Cloud(VPC)의 프라이빗 엔드포인트를 통해 AWS KMS에 직접 연결할 수 있습니다. VPC 엔드포인트를 사용하는 경우 VPC와 AWS KMS 사이의 통신은 모두 AWS 네트워크에서 수행됩니다.

VPC 안에 AWS KMS VPC 엔드포인트를 생성할 수 있습니다. 이 단계를 거치지 않으면 작업의 kms timeout 또는 크롤러의 internal service exception에서 작업이나 크롤러가 실패할 수 있습니다. 자세한 지침은 AWS Key Management Service Developer Guide의 [Connecting to AWS KMS Through a VPC Endpoint](#)를 참조하세요.

이러한 설명에 따라 [VPC 콘솔](#)에서 다음을 수행해야 합니다.

- 프라이빗 DNS 이름 활성화를 선택합니다.
- Java Database Connectivity(JDBC)에 액세스하는 작업 또는 크롤러에 사용할 보안 그룹(자체 참조 규칙 포함)을 선택합니다. AWS Glue 연결에 대한 자세한 정보는 [데이터에 연결](#) 섹션을 참조하세요.

JDBC 데이터 스토어에 액세스하는 크롤러 또는 작업에 보안 구성을 추가할 때 AWS Glue에 AWS KMS 엔드포인트로 가는 경로가 있어야 합니다. 네트워크 주소 변환(NAT) 게이트웨이 또는 AWS KMS VPC 엔드포인트에 경로를 제공할 수 있습니다. NAT 게이트웨이를 생성하려면 Amazon VPC 사용 설명서의 [NAT 게이트웨이](#)를 참조하세요.

AWS Glue 콘솔의 보안 구성 관련 작업

Warning

AWS Glue 보안 구성은 현재 Ray 작업에서 지원되지 않습니다.

AWS Glue의 보안 구성에는 암호화된 데이터를 쓸 때 필요한 속성이 포함되어 있습니다. AWS Glue 콘솔에서 보안 구성을 생성하여 크롤러, 작업 및 개발 엔드포인트에서 사용되는 암호화 속성을 제공합니다.

생성한 보안 구성 목록을 모두 보려면 <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 열고 탐색 창에서 [보안 구성(Security configurations)]을 선택합니다.

보안 구성 목록은 각 구성에 대한 다음 속성을 표시합니다.

명칭

구성 생성 시 제공한 고유 이름입니다. 이름은 문자(A~Z), 숫자(0~9), 하이픈(-) 또는 밑줄(_) 포함할 수 있으며 최대 255자로 지정할 수 있습니다.

Amazon S3 암호화 활성화

설정된 경우 SSE-KMS 또는 SSE-S3와 같은 Amazon Simple Storage Service(S3) 암호화 모드가 데이터 카탈로그의 메타데이터 스토어에 대해 활성화합니다.

Amazon CloudWatch Logs 암호화 활성화

설정된 경우 로그를 Amazon CloudWatch에 작성할 때 SSE-KMS와 같은 Amazon S3 암호화 모드가 활성화됩니다.

고급 설정: 작업 북마크 암호화 활성화

설정된 경우 작업이 북마크될 때 SSE-KMS와 같은 Amazon S3 암호화 모드가 활성화됩니다.

콘솔의 보안 구성 섹션에서 구성을 추가하거나 삭제할 수 있습니다. 목록에서 구성 이름을 선택하여 구성에 대한 더 자세한 정보를 알아봅니다. 세부 정보는 구성을 생성할 시 정의한 정보를 포함합니다.

보안 구성 추가

AWS Glue 콘솔을 사용하여 보안 구성을 추가하려면 보안 구성 페이지에서 Add security configuration(보안 구성 추가)을 선택합니다.

Add security configuration

Choose encryption and permission options for your accounts data catalog.

Security configuration properties

Name

Enter a unique security configuration name

Name may contain letters (A-Z), numbers (0-9), hyphens (-), or underscores (_), and can be up to 255 characters long.

Encryption settings

Enable and choose options for at-rest encryption.

- Enable S3 encryption**
Enable at-rest encryption for metadata stored in the data catalog.
- Enable CloudWatch logs encryption**
Enable at-rest encryption when writing logs to Amazon CloudWatch.

▼ Advanced settings

- Enable job bookmark encryption**
Enable at-rest encryption of job bookmark.

Cancel

Save

보안 구성 속성

고유한 보안 구성 이름을 입력합니다. 이름은 문자(A~Z), 숫자(0~9), 하이픈(-) 또는 밑줄(_) 포함할 수 있으며 최대 255자로 지정할 수 있습니다.

암호화 설정

Amazon S3의 데이터 카탈로그에 저장된 메타데이터와 Amazon CloudWatch의 로그에 저장된 메타데이터에 대해 저장 중 암호화를 활성화할 수 있습니다. AWS Glue 콘솔에서 AWS Key Management Service(AWS KMS) 키를 통해 데이터 및 메타데이터의 암호화를 설정하려면 콘솔 사용자에게 정책을 추가합니다. 다음 예와 같이 이 정책은 허용된 리소스를 Amazon S3 데이터 스토어 암호화에 사용되는 주요 Amazon 리소스 이름(ARN)으로 지정해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt",
      "kms:Encrypt"],
    "Resource": "arn:aws:kms:region:account-id:key/key-id"}
}
```

⚠ Important

보안 구성이 크롤러나 작업에 연결되면 전달되는 IAM 역할에 AWS KMS 권한이 있어야 합니다. 자세한 내용은 [AWS Glue에서 작성한 데이터 암호화](#) 단원을 참조하십시오.

구성을 정의할 때 필요한 다음 속성에 대한 값을 제공할 수 있습니다.

S3 암호화 활성화

Amazon S3 데이터를 작성할 때 Amazon S3 관리형 키(SSE-S3)를 사용하는 서버 측 암호화 또는 AWS KMS 관리형 키(SSE-KMS)를 사용하는 서버 측 암호화를 사용합니다. 이 필드는 선택 사항입니다. Amazon S3 액세스를 허용하려면 AWS KMS 키를 선택하거나 [키 ARN 입력(Enter a key ARN)]을 선택하고 키의 ARN을 제공합니다. `arn:aws:kms:region:account-id:key/key-id` 형식에 ARN 이름을 입력합니다. 또한 `arn:aws:kms:region:account-id:alias/alias-name` 같은 키 별칭으로 ARN을 제공할 수도 있습니다.

작업에 대해 Spark UI를 활성화하면 Amazon S3에 업로드된 Spark UI 로그 파일이 동일한 암호화로 적용됩니다.

⚠ Important

AWS Glue에서는 대칭 고객 마스터 키(CMK)만 지원합니다. AWS KMS 키(key) 목록에는 대칭 키만 표시됩니다. 그러나 Choose a AWS KMS key ARN(KMS 키 ARN 선택)을 선택하면 콘솔에서 모든 키 유형의 ARN을 입력할 수 있습니다. 대칭 키에 대한 ARN만 입력해야 합니다.

CloudWatch Logs 암호화 활성화

서버 측(SSE-KMS) 암호화는 CloudWatch Logs 암호화에 사용됩니다. 이 필드는 선택 사항입니다. 이를 설정하려면 AWS KMS 키를 선택하거나 [키 ARN 입력(Enter a key ARN)]을 선택하고 키에 대한 ARN 이름을 제공합니다. `arn:aws:kms:region:account-id:key/key-id` 형식에 ARN 이름을 입력합니다. 또한 `arn:aws:kms:region:account-id:alias/alias-name` 같은 키 별칭으로 ARN을 제공할 수도 있습니다.

고급 설정: 작업 북마크 암호화

클라이언트 측(CSE-KMS) 암호화는 작업 북마크를 암호화하는 데 사용됩니다. 이 필드는 선택 사항입니다. 북마크 데이터는 저장을 위해 Amazon S3에 전송되기 전에 암호화됩니다. 이를 설정하려면 AWS KMS 키를 선택하거나 [키 ARN 입력(Enter a key ARN)]을 선택하고 키에 대한 ARN 이름을 제공합니다. `arn:aws:kms:region:account-id:key/key-id` 형식에 ARN 이름을 입력합니다. 또한 `arn:aws:kms:region:account-id:alias/alias-name` 같은 키 별칭으로 ARN을 제공할 수도 있습니다.

자세한 내용은 Amazon Simple Storage Service 사용 설명서의 다음 주제를 참조하세요.

- SSE-S3에 대한 자세한 내용은 [Amazon S3가 관리하는 암호화 키\(SSE-S3\)를 사용하는 서버 측 암호화로 데이터 보호](#)를 참조하십시오.
- SSE-KMS에 대한 자세한 내용은 [AWS KMS keys를 사용하는 서버 측 암호화로 데이터 보호](#)를 참조하세요.
- CSE-KMS에 대한 자세한 내용은 [AWS KMS에 저장된 KMS 키 사용](#)을 참조하세요.

전송 중 암호화

AWS는 이동 중인 데이터에 대해 전송 계층 보안(TLS) 암호화를 제공합니다. AWS Glue의 [보안 구성](#)을 사용하여 크롤러, ETL 작업 및 개발 엔드포인트에 대한 암호화 설정을 구성할 수 있습니다. 데이터 카탈로그에 대한 설정을 통해 AWS Glue Data Catalog 암호화를 설정할 수 있습니다.

2018년 9월 4일부터 AWS Glue ETL 및 AWS Glue Data Catalog에 대한 AWS KMS(기존 보유 키 사용 및 서버 측 암호화)가 지원됩니다.

FIPS 규정 준수

명령줄 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-2 인증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용하세요. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 [Federal Information Processing Standard\(FIPS\) 140-2](#)를 참조하십시오.

키 관리

AWS Identity and Access Management(IAM)를 AWS Glue와 함께 사용하여 사용자, AWS 리소스, 그룹, 역할 및 액세스, 거부 등에 대한 세분화된 정책을 정의할 수 있습니다.

조직의 필요에 따라 리소스 기반 및 자격 증명 기반 정책을 모두 사용하여 메타데이터에 대한 액세스를 정의할 수 있습니다. 리소스 기반 정책은 교차 계정 액세스와 같은 정책을 설정할 수 있도록 리소스에 대한 액세스가 허용되거나 거부되는 원칙을 나열합니다. 자격 증명 기반 정책은 특히 IAM 내의 사용자, 그룹 및 역할에 연결됩니다.

단계별 예제는 [AWS 빅 데이터 블로그의 리소스 레벨 IAM 권한 및 리소스 기반 정책으로 AWS Glue Data Catalog에 대한 액세스 제한](#)을 참조하세요.

정책의 세분화된 액세스 부분은 Resource 절 내에서 정의됩니다. 이 부분은 작업을 수행할 수 있는 AWS Glue Data Catalog 객체 및 해당 작업에서 반환되는 결과 객체를 모두 정의합니다.

개발 엔드포인트는 AWS Glue 스크립트를 개발하고 테스트할 수 있는 환경입니다. 개발 엔드 포인트의 SSH 키를 추가하거나 삭제, 회전할 수 있습니다.

2018년 9월 4일부터 AWS KMS ETL 및 AWS Glue에 대한 AWS Glue Data Catalog(기존 보유 키 사용 및 서버 측 암호화)가 지원됩니다.

기타 AWS 서비스에서 AWS Glue 종속성

사용자가 AWS Glue 콘솔로 작업하려면 해당 사용자는 AWS 계정에 대한 AWS Glue 리소스로 작업하도록 허용하는 최소 권한 집합이 있어야 합니다. 이 AWS Glue 권한 이외에 콘솔은 다음 서비스로부터 권한을 필요로 합니다.

- 로그를 표시할 수 있는 Amazon CloudWatch Logs 권한.
- 역할을 나열하고 전달할 수 있는 AWS Identity and Access Management(IAM) 권한.
- 스택으로 작업할 수 있는 AWS CloudFormation 권한.
- Virtual Private Cloud(VPC), 서브넷, 보안 그룹, 인스턴스 및 기타 객체를 나열할 수 있는(작업, 크롤러를 생성하고 개발 엔드포인트를 생성할 때 VPC와 같은 Amazon EC2 항목을 설정할 수 있는) Amazon Elastic Compute Cloud(Amazon EC2) 권한.

- 버킷과 객체를 나열하고 스크립트를 검색하고 저장할 수 있는 Amazon Simple Storage Service(Amazon S3) 권한.
- 클러스터 작업을 위한 Amazon Redshift 권한.
- 인스턴스를 나열할 수 있는 Amazon Relational Database Service(Amazon RDS) 권한.

개발 엔드포인트

개발 엔드포인트는 AWS Glue 스크립트를 개발하고 테스트할 수 있는 환경입니다. AWS Glue를 사용하여 개발 엔드포인트를 생성, 편집 및 삭제할 수 있습니다. 생성된 모든 개발 엔드포인트를 나열할 수 있습니다. 개발 엔드포인트의 SSH 키를 추가하거나 삭제, 회전할 수 있습니다. 개발 엔드포인트를 사용하는 노트북을 만들 수도 있습니다.

구성 값을 제공하여 개발 환경을 제공합니다. 이러한 값은 네트워크를 설정하는 방법을 AWS Glue에 알립니다. 따라서 안전하게 개발 엔드포인트에 액세스할 수 있으며 엔드포인트가 데이터 스토어에 액세스할 수 있습니다. 그런 다음, 개발 엔드포인트에 연결하는 노트북을 생성할 수 있습니다. 노트북을 사용하여 ETL 스크립트를 작성하고 테스트할 수 있습니다.

AWS Glue ETL 작업을 실행하는 데 사용하는 IAM 역할과 비슷한 권한을 통해 AWS Identity and Access Management(IAM) 역할을 사용합니다. Virtual Private Cloud(VPC), 서브넷 및 보안 그룹을 사용하여 데이터 리소스에 안전하게 연결할 수 있는 개발 엔드포인트를 생성합니다. SSH를 사용하여 개발 엔드포인트에 연결할 SSH 키 페어를 생성합니다.

JDBC를 사용하여 데이터 집합에 액세스하는 데 사용할 수 있는 VPC 내에서 Amazon S3 데이터에 대한 개발 엔드포인트를 생성할 수 있습니다.

로컬 시스템에 Jupyter Notebook 클라이언트를 설치하고 노트북을 사용하여 개발 엔드포인트에서 ETL 스크립트를 디버깅하고 테스트할 수 있습니다. 아니면 Sagemaker 노트북을 사용하여 AWS에 대한 JupyterLab에서 ETL 스크립트를 작성할 수 있습니다. [개발 엔드포인트와 SageMaker 노트북 함께 사용하기](#)를 참조하세요.

AWS Glue는 `aws-glue-dev-endpoint`를 통해 접두사가 지정되는 이름으로 Amazon EC2 인스턴스를 태깅합니다.

개발 엔드포인트에서 노트북 서버를 설정하여 AWS Glue 확장 프로그램을 통해 PySpark를 실행할 수 있습니다.

AWS Glue의 ID 및 액세스 관리

AWS Identity and Access Management (IAM)은 관리자가 리소스에 대한 액세스를 안전하게 제어할 수 있도록 AWS 서비스 있도록 AWS 도와줍니다. IAM 관리자는 AWS Glue 리소스를 사용할 수 있는 인증 (로그인) 및 권한 부여 (권한 보유)를 받을 수 있는 사용자를 제어합니다. IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

Note

AWS Glue 방법 또는 권한 부여를 사용하여 AWS Glue Data Catalog의 데이터에 대한 액세스 AWS Lake Formation 권한을 부여할 수 있습니다. AWS Identity and Access Management (IAM) 정책을 사용하여 메서드로 세분화된 액세스 제어를 설정할 수 있습니다. AWS Glue Lake Formation은 관계형 데이터베이스 시스템의 GRANT/REVOKE 명령과 유사한 더 간단한 GRANT/REVOKE 권한 모델을 사용합니다.

이 섹션에는 AWS Glue 메서드 사용 방법에 대한 정보를 포함합니다. Lake Formation 권한 부여 사용에 대한 자세한 내용은 AWS Lake Formation Developer Guide의 [Granting Lake Formation permissions](#)를 참조하세요.

주제

- [고객](#)
- [ID를 통한 인증](#)
- [정책을 사용한 액세스 관리](#)
- [AWS Glue가 IAM과 함께 작동하는 방식](#)
- [AWS Glue에 대한 IAM 권한 구성](#)
- [AWS Glue 액세스 제어 정책 예제](#)
- [AWS Glue에 대한 관리형 정책](#)
- [AWS Glue 리소스 ARN 지정](#)
- [교차 계정 액세스 권한 부여](#)
- [AWS Glue ID 및 액세스 문제 해결](#)

고객

사용 방법 AWS Identity and Access Management (IAM)은 AWS Glue에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 - AWS Glue 서비스를 사용하여 작업을 수행하는 경우 관리자가 필요한 자격 증명과 권한을 제공합니다. 더 많은 AWS Glue 기능을 사용하여 작업을 수행함에 따라 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. AWS Glue의 기능에 액세스할 수 없는 경우 [AWS Glue ID 및 액세스 문제 해결](#) 섹션을 참조하세요.

서비스 관리자 — 회사에서 AWS Glue 리소스를 담당하는 경우 Glue에 AWS 대한 전체 액세스 권한이 있을 것입니다. 서비스 사용자가 액세스해야 하는 AWS Glue 기능과 리소스를 결정하는 것은 여러분의 몫입니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사에서 IAM과 AWS Glue를 사용하는 방법에 대한 자세한 내용은 [AWS Glue가 IAM과 함께 작동하는 방식](#)을 참조하십시오.

IAM 관리자 — IAM 관리자인 경우 AWS Glue에 대한 액세스를 관리하는 정책을 작성하는 방법에 대해 자세히 알아보는 것이 좋습니다. IAM에서 사용할 수 있는 AWS Glue ID 기반 정책의 예를 보려면 [AWS Glue 자격 증명 기반 정책 예제](#)를 참조하십시오.

ID를 통한 인증

인증은 ID 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페더레이션 ID의 예입니다. 페더레이션 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 [내 로그인 방법을](#) 참조하십시오. AWS 계정

AWS 프로그래밍 방식으로 액세스하는 경우 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트 (SDK)와 명령줄 인터페이스 (CLI)를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS [API 요청 서명](#)을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA)을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 [다중 인증](#) 및 IAM 사용 설명서의 [AWS에서 다중 인증\(MFA\) 사용](#)을 참조하십시오.

AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 태스크를 수행하는 데 사용하세요. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 [루트 사용자 보안 인증이 필요한 작업](#)을 참조하십시오.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 비롯한 수동 AWS 서비스 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 액세스하도록 하는 것입니다.

페더레이션 ID는 기업 사용자 디렉토리, 웹 ID 공급자, Identity Center 디렉터리의 사용자 또는 ID 소스를 통해 제공된 자격 증명을 사용하여 액세스하는 AWS 서비스 모든 사용자를 말합니다. AWS Directory Service 페더레이션 ID에 AWS 계정 액세스하면 이들이 역할을 맡고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center(을)를 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 자체 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화하여 모든 사용자 및 애플리케이션에서 사용할 수 있습니다. AWS 계정 IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 [IAM Identity Center란 무엇입니까?](#)를 참조하십시오.

IAM 사용자 및 그룹

[IAM 사용자는 단일 사용자](#) 또는 애플리케이션에 대한 특정 권한을 AWS 계정 가진 사용자 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 보안 인증이 있는 IAM 사용자를 생성하는 대신 임시 보안 인증을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 보안 인증이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체](#)를 참조하십시오.

[IAM 그룹](#)은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수입할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지

고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자를 만들어야 하는 경우\(역할이 아님\)](#)를 참조하십시오.

IAM 역할

[IAM 역할](#)은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console [전환하여](#) 에서 일시적으로 IAM 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 역할 사용](#)을 참조하십시오.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 [서드 파티 ID 공급자의 역할 생성](#) 단원을 참조하십시오. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 [권한 세트](#)를 참조하십시오.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 태스크에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스 경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 크로스 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 [IAM의 크로스 계정 리소스 액세스](#)를 참조하세요.
- 서비스 간 액세스 — 일부는 다른 AWS 서비스기능을 AWS 서비스 사용합니다. 예를 들어 서비스에서 직접 호출을 수행하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 태스크를 수행할 수 있습니다.
- 순방향 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 보안 AWS 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 AWS 서비스서비스에 AWS 서비스 요청하기 위한 요청과 결합하여 사용합니다. FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 [전달 액세스 세션](#)을 참조하세요.

- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 [IAM 역할](#)입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 서비스에 대한 권한을 위임할 역할 생성](#)을 참조하십시오.
- 서비스 연결 역할 — 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 — IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여](#)를 참조하십시오.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 [IAM 사용 설명서](#)의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하십시오.

정책을 사용한 액세스 관리

정책을 생성하고 이를 AWS ID 또는 리소스에 AWS 연결하여 액세스를 제어할 수 있습니다. 정책은 ID 또는 리소스와 연결될 때 AWS 해당 권한을 정의하는 객체입니다. AWS 주도자 (사용자, 루트 사용자 또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 JSON 문서로 AWS 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 내용은 IAM 사용 설명서의 [JSON 정책 개요](#)를 참조하십시오.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, iam:GetRole 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI, 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

보안 인증 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성](#)을 참조하십시오.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 내 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책과 인라인 정책의 선택](#)을 참조하십시오.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우, 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. IAM의 AWS 관리형 정책은 리소스 기반 정책에 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ACL을 지원하는 서비스의 예로는 아마존 S3와 아마존 VPC가 있습니다. AWS WAF ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 가이드의 [ACL\(액세스 제어 목록\) 개요](#)를 참조하십시오.

기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 타입에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 - 권한 경계는 자격 증명 기반 정책에 따라 IAM 엔티티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 보안 인증 기반 정책과 그 권한 경계의 교집합입니다. Principal 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 엔티티에 대한 권한 경계](#)를 참조하십시오.
- 서비스 제어 정책 (SCP) - SCP는 조직 또는 조직 단위 (OU) 에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations AWS Organizations 사업체가 소유한 여러 AWS 계정 개를 그룹화하고 중앙에서 관리하는 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책 (SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 항목을 포함하여 구성원 계정의 엔티티에 대한 권한을 제한합니다. AWS 계정 루트 사용자조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 [SCP 작동 방식](#)을 참조하십시오.
- 세션 정책 - 세션 정책은 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 보안 인증 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 내용은 IAM 사용 설명서의 [세션 정책](#)을 참조하십시오.

여러 정책 타입

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련된 경우 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 참조하십시오.

AWS Glue가 IAM과 함께 작동하는 방식

IAM을 사용하여 Glue에 대한 액세스를 관리하기 전에 AWS Glue와 함께 사용할 수 있는 IAM 기능에 대해 알아보세요. AWS

Glue와 함께 AWS 사용할 수 있는 IAM 기능

IAM 특성	AWS Glue 지원
ID 기반 정책	예
리소스 기반 정책	부분

IAM 특성	AWS Glue 지원
정책 작업	예
정책 리소스	예
정책 조건 키(서비스별)	예
ACLs	아니요
ABAC(정책 내 태그)	부분
임시 보안 인증	예
보안 주체 권한	아니요
서비스 역할	예
서비스 연결 역할	아니요

AWS Glue 및 기타 AWS 서비스가 대부분의 IAM 기능과 어떻게 작동하는지 자세히 알아보려면 IAM 사용 설명서에서 [IAM과 함께 작동하는AWS 서비스를](#) 참조하십시오.

Glue에 대한 ID 기반 정책 AWS

보안 인증 기반 정책 지원	예
----------------	---

자격 증명 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성](#)을 참조하십시오.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. 보안 인증 기반 정책에서는 보안 주체가 연결된 사용자 또는 역할에 적용되므로 보안 주체를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하십시오.

AWS Glue는 모든 AWS Glue 작업에 대해 자격 증명 기반 정책(IAM 정책)을 지원합니다. 정책을 연결하여 AWS Glue Data Catalog의 테이블과 같은 AWS Glue 리소스를 생성, 액세스 또는 수정할 수 있는 권한을 부여할 수 있습니다.

Glue의 ID 기반 정책 예제 AWS

AWS Glue ID 기반 정책의 예를 보려면 [을 참조하십시오. AWS Glue 자격 증명 기반 정책 예제](#)

Glue 내 리소스 기반 정책 AWS

리소스 기반 정책 지원	부분
--------------	----

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우, 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 등이 포함될 수 있습니다. AWS 서비스

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 보안 주체로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하십시오. 보안 주체와 리소스가 다른 AWS 계정경우 신뢰할 수 있는 계정의 IAM 관리자는 보안 주체 개체 (사용자 또는 역할) 에게 리소스에 액세스할 수 있는 권한도 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 보안 주체에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 [IAM의 교차 계정 리소스 액세스](#)를 참조하십시오.

Note

AWS Glue 리소스 정책은 데이터 카탈로그 리소스에 대한 권한을 관리하는 데에만 사용할 수 있습니다. 작업, 트리거, 개발 엔드포인트, 크롤러 또는 분류자 등과 같은 다른 AWS Glue 리소스에는 리소스 정책을 연결할 수 없습니다.

카탈로그당 하나의 리소스 정책만 허용되며, 크기는 10KB로 제한됩니다.

AWS Glue에서는 앞서 언급한 모든 종류의 Data Catalog 리소스를 위한 가상 컨테이너인 카탈로그에 리소스 정책이 연결됩니다. 각 AWS 계정은 카탈로그 ID가 계정 ID와 동일한 AWS 지역의 단일 카탈로그를 AWS 소유합니다. 카탈로그는 삭제하거나 수정할 수 없습니다.

리소스 정책은 호출자 보안 주체가 정책 문서의 "Principal" 블록에 포함되어 있는 카탈로그에 대한 모든 API 호출에 대해 평가됩니다.

AWS Glue 리소스 기반 정책의 예를 보려면 [여기](#)를 참조하십시오. [AWS Glue용 리소스 기반 정책 예제](#)

AWS Glue에 대한 정책 조치

정책 작업 지원 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 Action요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 태스크를 설명합니다. 정책 작업은 일반적으로 관련 AWS API 작업과 이름이 같습니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하십시오.

AWS Glue 작업 목록을 보려면 서비스 권한 부여 참조의 AWS [Glue에서 정의한 작업을](#) 참조하십시오.

AWS Glue의 정책 액션은 액션 앞에 다음 접두사를 사용합니다.

```
glue
```

단일 문서에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
  "glue:action1",
  "glue:action2"
]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, Get라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

```
"Action": "glue:Get*"
```

정책 예제를 보려면 [AWS Glue 액세스 제어 정책 예제](#)를 참조하십시오.

AWS Glue를 위한 정책 리소스

정책 리소스 지원	예
-----------	---

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 개체를 지정합니다. 문장에는 Resource 또는 NotResource 요소가 반드시 추가되어야 합니다. 모범 사례에 따라 [Amazon 리소스 이름\(ARN\)](#)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이 태스크를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

ARN을 사용하여 AWS Glue 리소스에 대한 액세스를 제어하는 방법에 대한 자세한 내용은 [을 참조하십시오](#) [AWS Glue 리소스 ARN 지정](#).

AWS Glue 리소스 유형 및 ARN 목록을 보려면 서비스 권한 부여 참조의 AWS [Glue에서 정의한 리소스](#)를 참조하십시오. 각 리소스의 ARN을 지정하는 데 사용할 수 있는 작업을 알아보려면 [AWS Glue에서 정의한 작업을](#) 참조하세요.

AWS Glue의 정책 조건 키

서비스별 정책 조건 키 지원	예
-----------------	---

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 옵션입니다. 같거나 작음과 같은 [조건 연산자](#)를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 AND 태스크를 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우는 논리적 OR 연산을 사용하여 조건을 AWS 평가합니다. 명문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예컨대, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 정책 요소: 변수 및 태그](#)를 참조하십시오.

AWS 글로벌 조건 키 및 서비스별 조건 키를 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 [AWS 설명서의 글로벌 조건 컨텍스트 키](#)를 참조하십시오.

AWS Glue 조건 키 목록을 보려면 서비스 인증 참조의 AWS [Glue용 조건 키](#)를 참조하십시오. 조건 키를 사용할 수 있는 작업 및 리소스를 알아보려면 [AWS Glue에서 정의한 작업을](#) 참조하세요.

정책 예제를 보려면 [조건 키 또는 컨텍스트 키를 사용하여 설정 제어](#)를 참조하십시오.

ACLs in AWS Glue

ACL 지원

아니요

ACL(액세스 통제 목록)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ABAC (접착제 포함) AWS

ABAC(정책 내 태그) 지원

부분

ABAC(속성 기반 액세스 통제)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. AWS에서는 이러한 속성을 태그라고 합니다. IAM 개체 (사용자 또는 역할) 및 여러 AWS 리소스에 태그를 첨부

할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 보안 주체의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 [조건 요소](#)에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 정보는 IAM 사용 설명서의 [ABAC란 무엇입니까?](#)를 참조하십시오. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 [속성 기반 액세스 제어\(ABAC\) 사용](#)을 참조하십시오.

Important

조건 컨텍스트 키는 크롤러, 작업, 트리거 및 개발 엔드포인트에서 AWS Glue API 작업에만 적용됩니다. 영향을 받는 API 작업에 대한 자세한 내용은 [AWS Glue의 조건 키](#)를 참조하십시오. AWS Glue 데이터 카탈로그 API 작업은 현재 `aws:referrer` 및 `aws:UserAgent` 글로벌 조건 컨텍스트 키를 지원하지 않습니다.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예제는 [태그를 사용한 액세스 권한 부여](#) 섹션에서 확인할 수 있습니다.

AWS Glue에서 임시 자격 증명 사용

임시 보안 인증 지원

예

임시 자격 증명을 사용하여 로그인하면 일부 자격 증명에 AWS 서비스 작동하지 않습니다. 임시 자격 증명을 사용하는 방법을 AWS 서비스 비롯한 추가 정보는 [IAM 사용 설명서의 IAM과 AWS 서비스 연동되는](#) 내용을 참조하십시오.

사용자 이름과 암호를 제외한 다른 방법을 AWS Management Console 사용하여 로그인하면 임시 자격 증명을 사용하는 것입니다. 예를 들어 회사의 SSO (Single Sign-On) 링크를 AWS 사용하여 액세스하는 경우 이 프로세스에서 자동으로 임시 자격 증명을 생성합니다. 또한 콘솔에 사용자로 로그인한 다

음 역할을 전환할 때 임시 보안 인증을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 [역할로 전환\(콘솔\)](#)을 참조하십시오.

또는 API를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다 AWS CLI . AWS 그런 다음 해당 임시 자격 증명을 사용하여 액세스할 수 AWS있습니다. AWS 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성할 것을 권장합니다. 자세한 정보는 [IAM의 임시 보안 자격 증명](#) 섹션을 참조하십시오.

Glue에 대한 AWS 크로스 서비스 사용자 권한

전달 액세스 세션(FAS) 지원

아니요

IAM 사용자 또는 역할을 사용하여 작업을 수행하는 AWS경우 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 서비스에 AWS 서비스 요청하라는 요청과 결합하여 사용합니다. AWS 서비스 FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 [전달 액세스 세션](#)을 참조하세요.

AWS Glue의 서비스 역할

서비스 역할 지원

예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 [IAM 역할](#)입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 서비스에 대한 권한을 위임할 역할 생성](#)을 참조하십시오.

Warning

서비스 역할의 권한을 변경하면 AWS Glue 기능이 작동하지 않을 수 있습니다. AWS Glue에서 안내를 제공하는 경우에만 서비스 역할을 수정하세요.

AWS Glue의 서비스 역할을 생성하는 방법에 대한 자세한 지침은 [1단계: AWS Glue 서비스를 위한 IAM 정책 생성](#) 및 [2단계: AWS Glue에 대한 IAM 역할 생성](#)을 참조하십시오.

Glue의 서비스 연결 역할 AWS

서비스 연결 역할 지원

아니요

서비스 연결 역할은 에 연결된 서비스 역할 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 [IAM으로 작업하는AWS 서비스](#)를 참조하십시오. 서비스 연결 역할 열에서 Yes(이)가 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 Yes(네) 링크를 선택합니다.

AWS Glue에 대한 IAM 권한 구성

AWS Identity and Access Management(IAM)를 사용하여 AWS Glue에서 리소스에 액세스하는 데 사용하는 정책과 역할을 정의합니다. 다음 단계는 AWS Glue에 대한 권한 설정을 위한 다양한 옵션을 소개합니다. 비즈니스의 요구 사항에 따라 리소스 액세스를 추가하거나 줄여야 할 경우도 있습니다.

Note

대신 AWS Glue에 대한 기본 IAM 권한으로 시작하려면 [AWS Glue에 대한 IAM 권한 설정](#) 섹션을 참조하세요.

1. [AWS Glue 서비스를 위한 IAM 정책 생성](#): AWS Glue 리소스에 액세스하도록 허용하는 서비스 정책을 생성합니다.
2. [AWS Glue에 대한 IAM 역할 생성](#): IAM 역할을 생성하고 AWS Glue에서 사용하는 Amazon Simple Storage Service(S3) 리소스에 대한 AWS Glue 서비스 정책 및 정책을 연결합니다.
3. [AWS Glue에 액세스하는 사용자 또는 그룹에 정책 연결](#): AWS Glue 콘솔에 로그인하는 모든 사용자 또는 그룹에 정책을 연결합니다.
4. [노트북용 IAM 정책 생성](#): 개발 엔드포인트에서 노트북 서버 생성에 사용할 노트북 서버 정책을 생성합니다.
5. [노트북용 IAM 역할 생성](#): IAM 역할을 생성하고 노트북 서버 정책을 연결합니다.
6. [Amazon SageMaker 노트북에 대한 IAM 정책 생성](#): 개발 엔드포인트에서 Amazon SageMaker 노트북을 생성할 때 사용할 IAM 정책을 생성합니다.

7. [Amazon SageMaker 노트북용 IAM 역할 생성](#): IAM 역할을 생성하고 개발 엔드포인트에서 Amazon SageMaker 노트북을 생성할 때 권한을 부여하는 정책을 연결합니다.

1단계: AWS Glue 서비스를 위한 IAM 정책 생성

Amazon S3 객체에 액세스하는 것처럼 다른 AWS 리소스의 데이터에 액세스하는 작업의 경우, 사용자를 대신하여 리소스에 액세스할 수 있는 권한이 AWS Glue에 있어야 합니다. AWS Identity and Access Management(IAM)을 사용하여 그러한 권한을 제공합니다.

Note

AWS 관리형 정책 **AWSGlueServiceRole**을 사용하면 이 단계를 생략해도 좋습니다.

이 단계에서는 **AWSGlueServiceRole**과 비슷한 정책을 만듭니다. IAM 콘솔에서 **AWSGlueServiceRole**의 최신 버전을 확인할 수 있습니다.

AWS Glue를 위한 IAM 정책을 생성하려면

이 정책은 몇 가지 Amazon S3 작업에 대한 권한을 부여하여 이 정책에 따른 역할을 AWS Glue에서 말을 때 필요한 계정 내 리소스를 관리할 수 있도록 합니다. 이 정책에 지정된 몇 가지 리소스는 Amazon S3 버킷, Amazon S3 ETL 스크립트, CloudWatch Logs 및 Amazon EC2 리소스에 대해 AWS Glue에서 사용하는 기본 이름을 나타냅니다. 간소화를 위해 AWS Glue는 기본적으로 `aws-glue-*`가 접두사인 계정에서 몇 가지 Amazon S3 객체를 버킷에 작성합니다.

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 정책을 선택합니다.
3. 정책 생성(Create Policy)을 선택합니다.
4. [Create Policy(정책 생성)] 화면에서 탭으로 이동하여 JSON을 편집합니다. 다음 JSON 설명으로 정책 문서를 만든 다음 [Review policy(정책 보기)]를 선택합니다.

Note

Amazon S3 리소스에 필요한 모든 액세스를 추가합니다. 필요한 리소스만의 액세스 정책에 대한 리소스 섹션의 범위를 정하고자 할 수 있습니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:*",
        "s3:GetBucketLocation",
        "s3:ListBucket",
        "s3:ListAllMyBuckets",
        "s3:GetBucketAcl",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeRouteTables",
        "ec2:CreateNetworkInterface",
        "ec2>DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcAttribute",
        "iam:ListRolePolicies",
        "iam:GetRole",
        "iam:GetRolePolicy",
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:PutBucketPublicAccessBlock"
      ],
      "Resource": [
        "arn:aws:s3:::aws-glue-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",

```

```

        "s3:PutObject",
        "s3:DeleteObject"
    ],
    "Resource": [
        "arn:aws:s3:::aws-glue-*/**",
        "arn:aws:s3:::*/*aws-glue-*/**"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject"
    ],
    "Resource": [
        "arn:aws:s3:::crawler-public*",
        "arn:aws:s3:::aws-glue-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:AssociateKmsKey"
    ],
    "Resource": [
        "arn:aws:logs:*:*:log-group:/aws-glue/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateTags",
        "ec2:DeleteTags"
    ],
    "Condition": {
        "ForAllValues:StringEquals": {
            "aws:TagKeys": [
                "aws-glue-service-resource"
            ]
        }
    },
    "Resource": [

```

```

        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:security-group/*",
        "arn:aws:ec2:*:*:instance/*"
    ]
}

```

다음 테이블은 이 정책이 보장하는 권한을 설명합니다.

작업	리소스	설명
"glue:*"	"*"	모든 AWS Glue API 작업을 실행할 수 있는 권한을 부여합니다.
"s3:GetBucketLocation", "s3:ListBucket", "s3:ListAllMyBuckets", "s3:GetBucketAcl",	"*"	크롤러, 작업, 개발 엔드포인트 및 노트북 서버의 Amazon S3 버킷 목록을 허용합니다.
"ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:CreateNetworkInterface", "ec2>DeleteNetworkInterface", "ec2:DescribeNetworkInterfaces", "ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcAttribute",	"*"	작업, 크롤러 및 개발 엔드포인트를 실행할 때 Virtual Private Cloud(VPC)와 같은 Amazon EC2 네트워크 항목을 설정할 수 있도록 허용합니다.
"iam:ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy"	"*"	크롤러, 작업, 개발 엔드포인트 및 노트북 서버의 IAM 역할 목록을 허용합니다.

작업	리소스	설명
"cloudwatch:PutMetricData"	"*"	작업에 대한 CloudWatch 지표 작성을 허용합니다.
"s3:CreateBucket", "s3:PutBucketPublicAccessBlock"	"arn:aws:s3:::aws-glue-*"	작업과 노트북 서버로부터 계정의 Amazon S3 버킷 생성을 허용합니다. 명명 규칙: aws-glue-라는 Amazon S3 폴더를 사용합니다. AWS Glue를 사용하여 퍼블릭 액세스를 차단하는 버킷을 생성합니다.
"s3:GetObject", "s3:PutObject", "s3:DeleteObject"	"arn:aws:s3:::aws-glue-*/*", "arn:aws:s3:::*/*aws-glue-*/*"	ETL 스크립트 및 노트북 서버 위치와 같은 객체를 저장할 때 Amazon S3 객체를 계정에서 얻고 넣고 삭제하는 것을 허용합니다. 명명 규칙: 이름이 aws-glue-로 시작하는 Amazon S3 버킷 또는 폴더에 권한을 부여합니다.
"s3:GetObject"	"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"	크롤러 및 작업의 예제 및 튜토리얼이 사용한 Amazon S3 객체 얻기를 허용합니다. 명명 규칙: Amazon S3 버킷 이름은 crawler-public 및 aws-glue-로 시작합니다.

작업	리소스	설명
"logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"	"arn:aws:logs:*:*: log-group:/aws-glu e/*"	CloudWatch Logs에 로그를 쓸 수 있습니다. 이름 전환: AWS Glue는 aws-glue로 시작하는 로그 투 로그 그룹 이름을 작성합 니다.
"ec2:CreateTags", "ec2:DeleteTags"	"arn:aws:ec2:*:*:n etwork-interface/ *", "arn:aws: ec2:*:*:security-g roup/*", "arn:aws: ec2:*:*:instance/* "	개발 엔드포인트를 생성하는 Amazon EC2 리소스 태깅을 허용합니다. 이름 전환: AWS Glue는 Amazon EC2 네트워크 인 터페이스, 보안 그룹 및 인스 턴스를 aws-glue-service-r esource에 태그 지정합니다.

5. 정책 검토 화면에서 정책 이름을 입력합니다(예: GlueServiceRolePolicy). 조건부 설명을 입력하고 정책에 만족하면 정책 생성을 선택합니다.

2단계: AWS Glue에 대한 IAM 역할 생성

AWS Glue가 사용자 대신 다른 서비스를 호출할 때 이용할 수 있도록 IAM 역할 권한을 부여해야 합니다. 여기에는 AWS Glue에서 사용하는 소스, 대상, 스크립트 및 임시 디렉터리에 대한 Amazon S3 액세스 권한이 포함됩니다. 크롤러, 작업 및 개발 엔드포인트는 권한이 필요합니다.

AWS Identity and Access Management(IAM)을 사용하여 그러한 권한을 제공합니다. IAM 역할에 정책을 추가하여 AWS Glue에 전달합니다.

AWS Glue용 IAM 역할 생성

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 생성을 선택합니다.

4. 역할 유형의 경우, AWS 서비스(Service)를 선택하고 Glue를 찾아 선택한 후, 다음: 권한(Next: Permissions)을 선택합니다.
5. 권한 정책 연결(Attach permissions policy) 페이지에서 필요한 권한이 포함된 정책을 선택합니다. 예를 들어 일반 AWS Glue 권한에 대한 AWS 관리형 정책 AWSGlueServiceRole과 Amazon S3 리소스에 대한 액세스에 대한 AWS 관리형 정책 AmazonS3FullAccess가 있습니다. 그런 다음 다음: 검토(Next: Review)를 선택합니다.

Note

이 역할의 정책 중 하나가 Amazon S3 소스 및 대상에 대한 권한을 보장한다는 것을 확인합니다. 특정 Amazon S3 리소스에 액세스를 위한 자체 정책을 제공하고자 할 수 있습니다. 데이터 원본은 s3:ListBucket 및 s3:GetObject 권한을 요구합니다. 데이터 대상은 s3:ListBucket, s3:PutObject 및 s3>DeleteObject 권한을 요구합니다. 리소스에 대한 Amazon S3 정책을 생성하는 방법은 [정책에서 리소스 지정을 참조하세요](#). Amazon S3 정책 예제는 [IAM 정책 작성하기: Amazon S3 버킷으로의 액세스를 보장하는 방법](#)을 참조하세요.

SSE-KMS로 암호화된 Amazon S3 소스 및 대상에 액세스하고자 할 경우, AWS Glue 크롤러, 작업 및 개발 엔드포인트가 데이터를 복호화할 수 있게 허용하는 정책을 연결합니다. 자세한 내용은 [AWS KMS 관리형 키\(SSE-KMS\)를 사용하는 서버 측 암호화로 데이터 보호](#)를 참조하세요.

다음은 예입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
      ]
    }
  ]
}
```

6. [역할 이름(Role name)]에 역할 이름을 입력합니다(예: AWSGlueServiceRoleDefault). 이름의 접두사가 문자열 AWSGlueServiceRole인 역할을 생성하고, 콘솔 사용자가 서비스로

해당 역할을 전달할 수 있도록 합니다. AWS Glue가 제공한 정책에서는 IAM 서비스 역할이 `AWSGlueServiceRole`로 시작될 것으로 예상합니다. 그렇지 않으면 정책을 추가하여 IAM 역할에 대한 `iam:PassRole` 권한을 사용자에게 부여하고 이름 규정과 일치시켜야 합니다. 역할 생성을 선택합니다.

Note

역할이 있는 노트북을 생성하면 해당 역할이 대화형 세션으로 전달되므로 두 위치에서 동일한 역할을 사용할 수 있습니다. 따라서 `iam:PassRole` 권한은 역할의 정책의 일부여야 합니다.

다음 예제를 사용하여 역할에 대한 새 정책을 생성합니다. 계정 번호를 사용자의 고유한 역할 이름으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::090000000210:role/<role_name>"
    }
  ]
}
```

3단계: AWS Glue에 액세스하는 사용자 또는 그룹에 정책 연결

관리자는 AWS Glue 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 모든 사용자, 그룹 또는 역할에 권한을 할당해야 합니다. 정책을 통해 AWS Identity and Access Management(IAM)를 사용하여 권한을 제공합니다. 이 단계에서는 사용자 또는 그룹에 권한을 할당하는 방법을 설명합니다.

이 단계가 완료되면 사용자 또는 그룹은 다음과 같은 정책이 연결됩니다.

- AWS 관리용 정책 `AWSGlueConsoleFullAccess` 또는 사용자 정책 `GlueConsoleAccessPolicy`
- **`AWSGlueConsoleSageMakerNotebookFullAccess`**
- **`CloudWatchLogsReadOnlyAccess`**
- **`AWSCloudFormationReadOnlyAccess`**

• AmazonAthenaFullAccess

인라인 정책을 연결하여 사용자 또는 그룹에 추가하는 방법

AWS 관리형 정책 또는 인라인 정책을 사용자 또는 그룹에 연결하여 AWS Glue 콘솔에 액세스할 수 있습니다. 이 정책에 지정된 몇 가지 리소스는 Amazon S3 버킷, Amazon S3 ETL 스크립트, CloudWatch Logs, AWS CloudFormation 및 Amazon EC2 리소스에 대해 AWS Glue에서 사용하는 기본 이름을 나타냅니다. 간소화를 위해 AWS Glue는 기본적으로 `aws-glue-*`가 접두사인 계정에서 몇 가지 Amazon S3 객체를 버킷에 작성합니다.

Note

AWS 관리형 정책 **AWSGlueConsoleFullAccess**를 사용하면 이 단계를 생략해도 좋습니다.

Important

AWS Glue는 사용자를 대신하여 작업을 수행하는 데 사용되는 역할에 대한 권한이 필요합니다. 이 과정을 완료하려면 **iam:PassRole** 권한을 AWS Glue 사용자 또는 그룹에 부여합니다. 이 정책은 AWS Glue 서비스 역할의 경우 **AWSGlueServiceRole**로 시작되는 역할에 권한을 부여하고, 노트북 서버를 생성할 때 필요한 역할 중에는 **AWSGlueServiceNotebookRole**로 시작하는 역할에 권한을 부여합니다. 이름 전환에 따라 **iam:PassRole** 권한에 대한 자체 정책을 생성할 수도 있습니다.

보안 모범 사례에 따라 Amazon S3 버킷 및 Amazon CloudWatch 로그 그룹에 대한 액세스를 추가로 제한하는 정책을 강화하여 액세스를 제한하는 것이 좋습니다. Amazon S3 정책 예제는 [IAM 정책 작성하기: Amazon S3 버킷으로의 액세스를 보장하는 방법](#)을 참조하세요.

이 단계에서는 **AWSGlueConsoleFullAccess**와 비슷한 정책을 만듭니다. IAM 콘솔에서 **AWSGlueConsoleFullAccess**의 최신 버전을 확인할 수 있습니다.

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 사용자 또는 사용자 그룹을 선택합니다.
3. 목록에서 정책을 삽입할 사용자 또는 그룹 이름을 선택합니다.
4. 권한 탭을 선택하고 필요하다면 Permissions policies(권한 정책) 섹션을 확장합니다.
5. [Add Inline Policy(인라인 정책 추가)] 링크를 선택합니다.

6. [Create Policy(정책 생성)] 화면에서 탭으로 이동하여 JSON을 편집합니다. 다음 JSON 설명으로 정책 문서를 만든 다음 [Review policy(정책 보기)]를 선택합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:*",
        "redshift:DescribeClusters",
        "redshift:DescribeClusterSubnetGroups",
        "iam:ListRoles",
        "iam:ListUsers",
        "iam:ListGroups",
        "iam:ListRolePolicies",
        "iam:GetRole",
        "iam:GetRolePolicy",
        "iam:ListAttachedRolePolicies",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeRouteTables",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeKeyPairs",
        "ec2:DescribeInstances",
        "rds:DescribeDBInstances",
        "rds:DescribeDBClusters",
        "rds:DescribeDBSubnetGroups",
        "s3:ListAllMyBuckets",
        "s3:ListBucket",
        "s3:GetBucketAcl",
        "s3:GetBucketLocation",
        "cloudformation:DescribeStacks",
        "cloudformation:GetTemplateSummary",
        "dynamodb:ListTables",
        "kms:ListAliases",
        "kms:DescribeKey",
        "cloudwatch:GetMetricData",
        "cloudwatch:ListDashboards"
      ],
      "Resource": [
```

```

        "*"
    ],
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::*/*aws-glue-*/*",
            "arn:aws:s3:::aws-glue-*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "tag:GetResources"
        ],
        "Resource": [
            "*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:CreateBucket",
            "s3:PutBucketPublicAccessBlock"
        ],
        "Resource": [
            "arn:aws:s3:::aws-glue-*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs:GetLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:*:*/aws-glue/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [

```

```

        "cloudformation:CreateStack",
        "cloudformation>DeleteStack"
    ],
    "Resource": "arn:aws:cloudformation:*:*:stack/aws-glue*/**"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:RunInstances"
    ],
    "Resource": [
        "arn:aws:ec2:*:*:instance/**",
        "arn:aws:ec2:*:*:key-pair/**",
        "arn:aws:ec2:*:*:image/**",
        "arn:aws:ec2:*:*:security-group/**",
        "arn:aws:ec2:*:*:network-interface/**",
        "arn:aws:ec2:*:*:subnet/**",
        "arn:aws:ec2:*:*:volume/**"
    ]
},
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam:*:*:role/AWSGlueServiceRole*",
    "Condition": {
        "StringLike": {
            "iam:PassedToService": [
                "glue.amazonaws.com"
            ]
        }
    }
},
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam:*:*:role/AWSGlueServiceNotebookRole*",
    "Condition": {
        "StringLike": {
            "iam:PassedToService": [
                "ec2.amazonaws.com"
            ]
        }
    }
}

```

```
    ]
  }
},
{
  "Action": [
    "iam:PassRole"
  ],
  "Effect": "Allow",
  "Resource": [
    "arn:aws:iam::*:role/service-role/AWSGlueServiceRole*"
  ],
  "Condition": {
    "StringLike": {
      "iam:PassedToService": [
        "glue.amazonaws.com"
      ]
    }
  }
}
]
```

다음 테이블은 이 정책이 보장하는 권한을 설명합니다.

작업	리소스	설명
"glue:*"	"*"	<p>모든 AWS Glue API 작업을 실행할 수 있는 권한을 부여합니다.</p> <p>"glue:*" 작업 없이 이전에 정책을 생성했다면 정책에 다음과 같은 개별 권한을 추가해야 합니다.</p> <ul style="list-style-type: none"> "glue:ListCrawlers" "glue:BatchGetCrawlers" "glue:ListTriggers" "glue:BatchGetTriggers" "glue:ListDevEndpoints" "glue:BatchGetDevEndpoints" "glue:ListJobs" "glue:BatchGetJobs"
"redshift:DescribeClusters", "redshift:DescribeClusterSubnetGroups"	"*"	Amazon Redshift로 연결을 생성하도록 허용합니다.
"iam:ListRoles", "iam:ListRolePolicies", "iam:GetRole", "iam:GetRolePolicy", "iam:ListAttachedRolePolicies"	"*"	크롤러, 작업, 개발 엔드포인트 및 노트북 서버와 작업할 경우 IAM 역할 목록을 허용합니다.


작업	리소스	설명
"ec2:DescribeSecurityGroups", "ec2:DescribeSubnets", "ec2:DescribeVpcs", "ec2:DescribeVpcEndpoints", "ec2:DescribeRouteTables", "ec2:DescribeVpcAttributes", "ec2:DescribeKeyPairs", "ec2:DescribeInstances"	"*"	작업, 크롤러 및 개발 엔드포인트를 실행할 때 VPC와 같은 Amazon EC2 네트워크 항목의 설치를 허용합니다.
"rds:DescribeDBInstances"	"*"	Amazon RDS로 연결을 생성하도록 허용합니다.
"s3:ListAllMyBuckets", "s3:ListBucket", "s3:GetBucketAcl", "s3:GetBucketLocation"	"*"	크롤러, 작업, 개발 엔드포인트 및 노트북 서버와 작업할 경우, Amazon S3 버킷 목록을 허용합니다.
"dynamodb:ListTables"	"*"	DynamoDB 테이블 나열을 허용합니다.
"kms:ListAliases", "kms:DescribeKey"	"*"	KMS 키 작업을 허용합니다.
"cloudwatch:GetMetricData", "cloudwatch:ListDashboards"	"*"	CloudWatch 지표 작업을 허용합니다.

작업	리소스	설명
"s3:GetObject", "s3:PutObject"	"arn:aws:s3::: aws-glue-*/*", "arn:aws:s3::: */*aws-glue-*/*", "arn:aws:s3::: aws-glue-*"	<p>ETL 스크립트 및 노트북 서버 위치와 같은 객체를 저장할 때 Amazon S3 객체를 계정에서 얻고 넣는 것을 허용합니다.</p> <p>명명 규칙: 이름이 aws-glue-로 시작하는 Amazon S3 버킷 또는 폴더에 권한을 부여합니다.</p>
"tag:GetResources"	"*"	AWS 태그 검색을 허용합니다.
"s3:CreateBucket", "s3:PutBucketPublicAccessBlock"	"arn:aws:s3::: aws-glue-*"	<p>ETL 스크립트 및 노트북 서버 위치와 같은 객체를 저장할 때 Amazon S3 버킷을 계정에 생성하는 것을 허용합니다.</p> <p>명명 규칙: 이름이 aws-glue-로 시작하는 Amazon S3 버킷에 권한을 부여합니다.</p> <p>AWS Glue를 사용하여 퍼블릭 액세스를 차단하는 버킷을 생성합니다.</p>

작업	리소스	설명
"logs:GetLogEvents"	"arn:aws:logs:*:*: /aws-glue/*"	CloudWatch Logs의 검색 허용 이름 전환: AWS Glue는 aws-glue로 시작하는 로그 투 로그 그룹 이름을 작성합니다.
"cloudformation:CreateStack", "cloudformation>DeleteStack"	"arn:aws:cloudformation:*:*:stack/ aws-glue*/*"	노트북 서버로 작업할 때 AWS CloudFormation 스택 관리를 허용합니다. 이름 전환: AWS Glue는 aws-glue로 시작하는 스택 이름을 생성합니다.
"ec2:RunInstances"	"arn:aws:ec2:*:*:instance/*", "arn:aws:ec2:*:*:key-pair/*", "arn:aws:ec2:*:*:image/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:subnet/*", "arn:aws:ec2:*:*:volume/*"	개발 엔드포인트 및 노트북 서버의 실행을 허용합니다.
"iam:PassRole"	"arn:aws:iam:*:*:role/ AWSGlueServiceRole*"	AWS Glue가 AWSGlueServiceRole 로 시작하는 역할에 대해 PassRole 권한을 수입하도록 허용합니다.

작업	리소스	설명
"iam:PassRole"	"arn:aws:iam::*:role/ AWSGlueServiceNotebookRole*"	Amazon EC2가 AWSGlueServiceNotebookRole 로 시작하는 역할에 대해 PassRole 권한을 수임하도록 허용합니다.
"iam:PassRole"	"arn:aws:iam::*:role/service-role/ AWSGlueServiceRole*"	AWS Glue가 service-role/AWSGlueServiceRole 로 시작하는 역할에 대해 PassRole 권한을 수임하도록 허용합니다.


7. 정책 검토 화면에서 정책의 이름을 입력합니다(예: GlueConsoleAccessPolicy). 정책에 만족하면 정책 생성을 선택합니다. 화면 상단에 있는 빨간색 상자에 표시되는 오류가 있지 않은지 확인합니다. 표시되는 오류가 있다면 수정합니다.

 Note

[Use autoformatting]을 선택하면 정책을 열거나 [Validate Policy]를 선택할 때마다 정책의 형식이 다시 지정됩니다.

AWSGlueConsoleFullAccess 관리 정책을 연결하는 방법

AWSGlueConsoleFullAccess 정책을 추가하여 AWS Glue 콘솔 사용자가 필요한 권한을 제공합니다.

 Note

AWS Glue 콘솔 액세스용 자체 정책을 생성했다면 이 단계를 건너뛸 수 있습니다.

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.

2. 탐색 창에서 정책을 선택합니다.
3. 정책 목록에서 `AWSGlueConsoleFullAccess` 옆의 확인란을 선택합니다. [Filter] 메뉴와 검색 상자를 사용하여 정책 목록을 필터링할 수 있습니다.
4. 정책 조치를 선택한 후 연결을 선택합니다.
5. 정책을 연결하려는 사용자를 선택합니다. 필터 메뉴와 검색 상자를 사용하면 보안 주체 개체 목록을 필터링할 수 있습니다. 정책을 추가할 사용자를 선택한 다음 [Attach policy(정책 추가)]를 선택합니다.

`AWSGlueConsoleSageMakerNotebookFullAccess` 관리형 정책을 연결하려면

`AWSGlueConsoleSageMakerNotebookFullAccess` 정책을 사용자에게 연결해 AWS Glue 콘솔에서 생성한 SageMaker 노트북을 관리할 수 있습니다. 필요한 다른 AWS Glue 콘솔 권한 이외에도 이 정책은 SageMaker 노트북 관리에 필요한 리소스에 대한 액세스 권한을 부여합니다.

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 목록에서 `AWSGlueConsoleSageMakerNotebookFullAccess` 옆의 확인란을 선택합니다. [Filter] 메뉴와 검색 상자를 사용하여 정책 목록을 필터링할 수 있습니다.
4. 정책 조치를 선택한 후 연결을 선택합니다.
5. 정책을 연결하려는 사용자를 선택합니다. 필터 메뉴와 검색 상자를 사용하면 보안 주체 개체 목록을 필터링할 수 있습니다. 정책을 추가할 사용자를 선택한 다음 [Attach policy(정책 추가)]를 선택합니다.

`CloudWatchLogsReadOnlyAccess` 관리 정책을 연결하는 방법

`CloudWatchLogsReadOnlyAccess` 정책을 사용자에게 연결하고 CloudWatch Logs 콘솔에서 AWS Glue가 생성한 로그를 볼 수 있습니다.

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 목록에서 [CloudWatchLogsReadOnlyAccess] 이름 옆의 확인란을 선택합니다. [Filter] 메뉴와 검색 상자를 사용하여 정책 목록을 필터링할 수 있습니다.
4. 정책 조치를 선택한 후 연결을 선택합니다.

5. 정책을 연결하려는 사용자를 선택합니다. 필터 메뉴와 검색 상자를 사용하면 보안 주체 개체 목록을 필터링할 수 있습니다. 정책을 추가할 사용자를 선택한 다음 [Attach policy(정책 추가)]를 선택합니다.

AWSCloudFormationReadOnlyAccess 관리 정책을 연결하는 방법

AWSCloudFormationReadOnlyAccess 정책을 사용자에게 연결하고 AWS Glue가 사용한 AWS CloudFormation 스택을 AWS CloudFormation 콘솔에서 볼 수 있습니다.

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 목록에서 AWSCloudFormationReadOnlyAccess 이름 옆의 확인란을 선택합니다. [Filter] 메뉴와 검색 상자를 사용하여 정책 목록을 필터링할 수 있습니다.
4. 정책 조치를 선택한 후 연결을 선택합니다.
5. 정책을 연결하려는 사용자를 선택합니다. 필터 메뉴와 검색 상자를 사용하면 보안 주체 개체 목록을 필터링할 수 있습니다. 정책을 추가할 사용자를 선택한 다음 [Attach policy(정책 추가)]를 선택합니다.

AmazonAthenaFullAccess 관리 정책을 추가하는 방법

AmazonAthenaFullAccess 정책을 사용자에게 연결하고 Amazon S3 데이터를 Athena 콘솔에서 볼 수 있습니다.

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택합니다.
3. 정책 목록에서 [AmazonAthenaFullAccess(Amazon Athena 완전한 액세스)] 옆의 확인란을 선택합니다. [Filter] 메뉴와 검색 상자를 사용하여 정책 목록을 필터링할 수 있습니다.
4. 정책 조치를 선택한 후 연결을 선택합니다.
5. 정책을 연결하려는 사용자를 선택합니다. 필터 메뉴와 검색 상자를 사용하면 보안 주체 개체 목록을 필터링할 수 있습니다. 정책을 추가할 사용자를 선택한 다음 [Attach policy(정책 추가)]를 선택합니다.

4단계: 노트북 서버용 IAM 정책 생성

개발 엔드포인트로 노트북을 사용하고자 한다면 노트북 서버를 생성할 때 권한을 지정해야 합니다. AWS Identity and Access Management(IAM)을 사용하여 그러한 권한을 제공합니다.

이 정책은 몇 가지 Amazon S3 작업에 대한 권한을 부여하여 이 정책에 따른 역할을 AWS Glue에서 맡을 때 필요한 계정 내 리소스를 관리할 수 있도록 합니다. 이 정책에 지정된 몇 가지 리소스는 Amazon S3 버킷, Amazon S3 ETL 스크립트, Amazon EC2 리소스에 대해 AWS Glue에서 사용하는 기본 이름을 나타냅니다. 간소화를 위해 AWS Glue는 기본적으로 접두사가 `aws-glue-*`인 계정에서 몇 가지 Amazon S3 객체를 버킷에 작성합니다.

Note

AWS 관리형 정책 **AWSGlueServiceNotebookRole**를 사용하면 이 단계를 생략해도 좋습니다.

이 단계에서는 **AWSGlueServiceNotebookRole**과 비슷한 정책을 만듭니다. IAM 콘솔에서 **AWSGlueServiceNotebookRole**의 최신 버전을 확인할 수 있습니다.

노트북에 대한 IAM 정책을 생성하려면

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 정책을 선택합니다.
3. 정책 생성(Create Policy)을 선택합니다.
4. [Create Policy(정책 생성)] 화면에서 탭으로 이동하여 JSON을 편집합니다. 다음 JSON 설명으로 정책 문서를 만든 다음 [Review policy(정책 보기)]를 선택합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateDatabase",
        "glue:CreatePartition",
        "glue:CreateTable",
        "glue>DeleteDatabase",
        "glue>DeletePartition",

```

```
"glue:DeleteTable",
"glue:GetDatabase",
"glue:GetDatabases",
"glue:GetPartition",
"glue:GetPartitions",
"glue:GetTable",
"glue:GetTableVersions",
"glue:GetTables",
"glue:UpdateDatabase",
"glue:UpdatePartition",
"glue:UpdateTable",
"glue:GetJobBookmark",
"glue:ResetJobBookmark",
"glue:CreateConnection",
"glue:CreateJob",
"glue>DeleteConnection",
"glue>DeleteJob",
"glue:GetConnection",
"glue:GetConnections",
"glue:GetDevEndpoint",
"glue:GetDevEndpoints",
"glue:GetJob",
"glue:GetJobs",
"glue:UpdateJob",
"glue:BatchDeleteConnection",
"glue:UpdateConnection",
"glue:GetUserDefinedFunction",
"glue:UpdateUserDefinedFunction",
"glue:GetUserDefinedFunctions",
"glue>DeleteUserDefinedFunction",
"glue:CreateUserDefinedFunction",
"glue:BatchGetPartition",
"glue:BatchDeletePartition",
"glue:BatchCreatePartition",
"glue:BatchDeleteTable",
"glue:UpdateDevEndpoint",
"s3:GetBucketLocation",
"s3:ListBucket",
"s3:ListAllMyBuckets",
"s3:GetBucketAcl"
],
"Resource": [
  "*"
]
```



```
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::crawler-public*",
        "arn:aws:s3:::aws-glue*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::aws-glue*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags",
        "ec2:DeleteTags"
      ],
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": [
            "aws-glue-service-resource"
          ]
        }
      },
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:security-group/*",
        "arn:aws:ec2:*:*:instance/*"
      ]
    }
  ]
}
```

다음 테이블은 이 정책이 보장하는 권한을 설명합니다.

작업	리소스	설명
"glue:*"	"*"	모든 AWS Glue API 작업을 실행할 수 있는 권한을 부여합니다.
"s3:GetBucketLocation", "s3:ListBucket", "s3:ListAllMyBuckets", "s3:GetBucketAcl"	"*"	노트북 서버의 Amazon S3 버킷 목록을 허용합니다.
"s3:GetObject"	"arn:aws:s3:::crawler-public*", "arn:aws:s3:::aws-glue-*"	노트북의 예제 및 튜토리얼이 사용한 Amazon S3 객체 얻기를 허용합니다. 명명 규칙: Amazon S3 버킷 이름은 crawler-public 및 aws-glue-로 시작합니다.
"s3:PutObject", "s3>DeleteObject"	"arn:aws:s3:::aws-glue*"	노트북의 계정으로 Amazon S3 객체를 넣고 삭제하는 것을 허용합니다. 명명 규칙: aws-glue라는 Amazon S3 폴더를 사용합니다.

작업	리소스	설명
"ec2:CreateTags", "ec2>DeleteTags"	"arn:aws:ec2:*:*:network-interface/*", "arn:aws:ec2:*:*:security-group/*", "arn:aws:ec2:*:*:instance/*"	노트북 서버를 생성하는 Amazon EC2 리소스 태깅을 허용합니다. 이름 전환: AWS Glue는 Amazon EC2 인스턴스를 aws-glue-service-resource에 태그 지정합니다.

5. 정책 검토 화면에서 정책 이름을 입력합니다(예: GlueServiceNotebookPolicyDefault). 조건부 설명을 입력하고 정책에 만족하면 정책 생성을 선택합니다.

5단계: 노트북 서버용 IAM 역할 생성

개발 엔드포인트로 노트북을 사용하고자 한다면 IAM 역할 권한을 부여해야 합니다. IAM 역할을 통해 AWS Identity and Access Management IAM을 사용하여 권한을 제공합니다.

Note

IAM 콘솔을 사용하여 IAM 역할을 생성하면 인스턴스 프로파일이 자동으로 생성되고 해당 역할과 동일한 이름이 지정됩니다.

노트북용 IAM 역할을 생성하려면

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 생성을 선택합니다.
4. 역할 유형의 경우, AWS 서비스를 선택하고 EC2를 찾아 선택한 다음 EC2 사용 사례를 선택하고 다음: 권한을 선택합니다.
5. 권한 정책 연결(Attach permissions policy) 페이지에서 필요한 권한이 포함된 정책을 선택합니다. 예를 들어 일반 AWS Glue 권한에 대한 AWSGlueServiceNotebookRole과 Amazon S3 리소스에

대한 액세스에 대한 AWS 관리형 정책 AmazonS3FullAccess가 있습니다. 그런 다음 다음: 검토 (Next: Review)를 선택합니다.

Note

이 역할의 정책 중 하나가 Amazon S3 소스 및 대상에 대한 권한을 보장한다는 것을 확인합니다. 정책이 노트북 서버를 생성할 때 노트북을 저장하는 위치로 완전한 액세스가 허용되는지 확인합니다. 특정 Amazon S3 리소스에 액세스를 위한 자체 정책을 제공하고자 할 수 있습니다. 리소스에 대한 Amazon S3 정책을 생성하는 방법은 [정책에서 리소스 지정을 참조](#)하세요.

SSE-KMS로 암호화된 Amazon S3 소스 및 대상에 액세스하고자 할 경우, 노트북이 데이터를 복호화할 수 있게 허용하는 정책을 연결합니다. 자세한 내용은 [AWS KMS 관리형 키 \(SSE-KMS\)를 사용하는 서버 측 암호화로 데이터 보호](#)를 참조하세요.

다음은 예입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
      ]
    }
  ]
}
```

6. 역할 이름에 역할의 이름을 입력합니다. 이름의 접두사가 문자열 AWSGlueServiceNotebookRole인 역할을 생성하고, 콘솔 사용자가 노트북 서버로 해당 역할을 전달할 수 있도록 합니다. AWS Glue가 제공한 정책에서는 IAM 서비스 역할이 AWSGlueServiceNotebookRole로 시작될 것으로 예상합니다. 그렇지 않으면 사용자에게 정책을 추가하여 IAM 역할의 iam:PassRole 권한이 이름 규정과 일치하도록 해야 합니다. 예를 들면 AWSGlueServiceNotebookRoleDefault를 입력합니다. 그런 다음 역할 생성을 선택합니다.

6단계: SageMaker 노트북용 IAM 정책 생성

개발 엔드포인트에서 SageMaker 노트북을 사용할 계획이라면 노트북을 생성할 때 권한을 지정해야 합니다. AWS Identity and Access Management(IAM)을 사용하여 그러한 권한을 제공합니다.

SageMaker 노트북용 IAM 정책을 생성하려면

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 정책을 선택합니다.
3. 정책 생성(Create Policy)을 선택합니다.
4. 정책 생성 페이지에서 탭으로 이동하여 JSON을 편집합니다. 다음 JSON 문을 사용해 정책 문서를 생성합니다. 환경에 대한 *bucket-name*, *region-code* 및 *account-id*를 편집합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::bucket-name"
      ]
    },
    {
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::bucket-name*"
      ]
    },
    {
      "Action": [
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:CreateLogGroup"
      ]
    }
  ]
}
```

```

    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:logs:region-code:account-id:log-group:/aws/sagemaker/*",
        "arn:aws:logs:region-code:account-id:log-group:/aws/sagemaker/
*:log-stream:aws-glue-*"
    ]
  },
  {
    "Action": [
      "glue:UpdateDevEndpoint",
      "glue:GetDevEndpoint",
      "glue:GetDevEndpoints"
    ],
    "Effect": "Allow",
    "Resource": [
      "arn:aws:glue:region-code:account-id:devEndpoint/*"
    ]
  },
  {
    "Action": [
      "sagemaker:ListTags"
    ],
    "Effect": "Allow",
    "Resource": [
      "arn:aws:sagemaker:region-code:account-id:notebook-instance/*"
    ]
  }
]
}

```

그런 다음 정책 검토를 선택합니다.

다음 테이블은 이 정책이 보장하는 권한을 설명합니다.

작업	리소스	설명
"s3:ListBucket*"	"arn:aws:s3::: <i>bucket-name</i> "	Amazon S3 버킷을 나열할 수 있는 권한을 부여합니다.

작업	리소스	설명
"s3:GetObject"	"arn:aws:s3::: <i>bucket-name</i> *"	SageMaker 노트북에서 사용하는 Amazon S3 객체를 가져오도록 권한을 부여합니다.
"logs:CreateLogStream", "logs:DescribeLogStreams", "logs:PutLogEvents", "logs:CreateLogGroup"	"arn:aws:logs: <i>region-code</i> : <i>account-id</i> :log-group:/aws/sagemaker/*", "arn:aws:logs: <i>region-code</i> : <i>account-id</i> :log-group:/aws/sagemaker/*:log-stream:aws-glue-*"	노트북에서 Amazon CloudWatch Logs에 로그를 쓸 수 있는 권한을 부여합니다. 이름 지정 규칙: 이름이 aws-glue로 시작하는 로그 그룹에 씁니다.
"glue:UpdateDevEndpoint", "glue:GetDevEndpoint", "glue:GetDevEndpoints"	"arn:aws:glue: <i>region-code</i> : <i>account-id</i> :devEndpoint/*"	SageMaker 노트북에서 개발 엔드포인트를 사용할 수 있는 권한을 부여합니다.
"sagemaker:ListTags"	"arn:aws:sagemaker : <i>region-code</i> : <i>account-id</i> :notebook-instance/*"	SageMaker 리소스에 대한 태그를 반환할 수 있는 권한을 부여합니다. aws-glue-dev-endpoint 태그는 SageMaker 노트북을 개발 엔드포인트에 연결하기 위해 필요합니다.

- [정책 검토(Review Policy)] 화면에서 [정책 이름(Policy Name)]을 입력합니다(예: AWSGlueSageMakerNotebook). 조건부 설명을 입력하고 정책에 만족하면 정책 생성을 선택합니다.

7단계: SageMaker 노트북용 IAM 역할 생성

개발 엔드포인트로 SageMaker 노트북을 사용하고자 한다면 IAM 역할 권한을 부여해야 합니다. IAM 역할을 통해 AWS Identity and Access Management(IAM)를 사용하여 권한을 제공합니다.

SageMaker 노트북용 IAM 역할을 생성하려면

1. AWS Management Console에 로그인하고 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 생성을 선택합니다.
4. 역할 유형의 경우, AWS 서비스를 선택하고 SageMaker를 찾아 선택한 다음 SageMaker - 실행 사용 사례를 선택합니다. 그런 다음 다음: 권한을 선택합니다.
5. Attach permissions policy(권한 정책 연결) 페이지에서 필요한 권한을 포함한 정책을 선택합니다 (예: AmazonSageMakerFullAccess). 다음: 검토를 선택합니다.

SSE-KMS로 암호화된 Amazon S3 소스 및 대상에 액세스하고자 할 경우, 다음 예에 표시된 것처럼 노트북이 데이터를 복호화하도록 허용하는 정책을 연결합니다. 자세한 내용은 [AWS KMS 관리형 키\(SSE-KMS\)를 사용하는 서버 측 암호화로 데이터 보호](#)를 참조하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:*:account-id-without-hyphens:key/key-id"
      ]
    }
  ]
}
```

6. 역할 이름에 역할의 이름을 입력합니다. 콘솔 사용자로부터 SageMaker으로 역할이 전달되도록 하려면 문자열 `AWSGlueServiceSageMakerNotebookRole`로 접두사가 지정된 이름을 사용합니다. AWS Glue에서 제공한 정책은 IAM 역할이 `AWSGlueServiceSageMakerNotebookRole`로 시작할 것이라고 기대합니다. 그렇지 않으면 사용자에게 정책을 추가하여 IAM 역할의 `iam:PassRole` 권한이 이름 규정과 일치하도록 해야 합니다.

예를 들어 `AWSGlueServiceSageMakerNotebookRole-Default`를 입력한 다음 [역할 생성 (Create role)]을 선택합니다.

7. 역할을 생성한 후 AWS Glue에서 SageMaker 노트북을 생성하는 데 필요한 추가 권한을 허용하는 정책을 연결합니다.

방금 생성한 역할인 `AWSGlueServiceSageMakerNotebookRole-Default`를 열고 [정책 연결 (Attach policies)]을 선택합니다. 역할에 생성한 `AWSGlueSageMakerNotebook` 정책을 연결합니다.

AWS Glue 액세스 제어 정책 예제

이 섹션에는 자격 증명 기반(IAM) 액세스 제어 정책과 AWS Glue 리소스 정책 모두의 예제가 포함되어 있습니다.

목차

- [AWS Glue 자격 증명 기반 정책 예제](#)
 - [정책 모범 사례](#)
 - [리소스 수준 권한은 특정 AWS Glue 객체에만 적용됨](#)
 - [AWS Glue Console에서 사용](#)
 - [사용자가 자신의 고유한 권한을 볼 수 있도록 허용](#)
 - [테이블에 대한 읽기 전용 권한 부여](#)
 - [GetTables 권한으로 테이블 필터링](#)
 - [테이블 및 모든 파티션에 전체 액세스 권한 부여](#)
 - [이름 접두사 및 명시적 거부로 액세스 제어](#)
 - [태그를 사용한 액세스 권한 부여](#)
 - [태그를 사용한 액세스 거부](#)
 - [목록 및 배치 API 작업과 함께 태그 사용](#)
 - [조건 키 또는 컨텍스트 키를 사용하여 설정 제어](#)
 - [조건 키를 사용하여 설정을 제어하는 정책 제어](#)
 - [컨텍스트 키를 사용하여 설정을 제어하는 정책 제어](#)
 - [ID가 데이터 미리 보기 세션을 생성하지 못하게 하기](#)
- [AWS Glue용 리소스 기반 정책 예제](#)
 - [AWS Glue에서 리소스 기반 정책 사용 시 고려 사항](#)
 - [리소스 정책을 사용하여 동일한 계정에서 액세스 제어](#)

AWS Glue 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할에는 AWS Glue 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 AWS API를 사용해 태스크를 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용자 설명서의 [IAM 정책 생성](#)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 포함하여 AWS Glue에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 승인 참조의 [AWS Glue에 사용되는 작업, 리소스 및 조건 키](#)를 참조하세요.

Note

이 섹션에 제공된 예제는 모두 us-west-2 리전을 사용합니다. 이 리전을 사용하려는 AWS 리전으로 바꿀 수 있습니다.

주제

- [정책 모범 사례](#)
- [리소스 수준 권한은 특정 AWS Glue 객체에만 적용됨](#)
- [AWS Glue Console에서 사용](#)
- [사용자가 자신의 고유한 권한을 볼 수 있도록 허용](#)
- [테이블에 대한 읽기 전용 권한 부여](#)
- [GetTables 권한으로 테이블 필터링](#)
- [테이블 및 모든 파티션에 전체 액세스 권한 부여](#)
- [이름 접두사 및 명시적 거부로 액세스 제어](#)
- [태그를 사용한 액세스 권한 부여](#)
- [태그를 사용한 액세스 거부](#)
- [목록 및 배치 API 작업과 함께 태그 사용](#)
- [조건 키 또는 컨텍스트 키를 사용하여 설정 제어](#)
- [ID가 데이터 미리 보기 세션을 생성하지 못하게 하기](#)

정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 AWS Glue 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용자 설명서의 [AWS 관리형 정책](#) 또는 [AWS 직무에 대한 관리형 정책](#)을 참조하세요.
- 최소 권한 적용 - IAM 정책을 사용하여 권한을 설정하는 경우 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용자 설명서에 있는 [IAM의 정책 및 권한](#)을 참조하세요.
- Use conditions in IAM policies to further restrict access(IAM 정책의 조건을 사용하여 액세스 추가 제한) - 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어 SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS CloudFormation와 같이, 특정 AWS 서비스를 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 정보는 IAM 사용자 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 검증하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 IAM 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 검증합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 권장 사항을 제공하여 안전하고 기능적인 정책을 생성하도록 돕습니다. 자세한 정보는 IAM 사용자 설명서의 [IAM Access Analyzer 정책 검증](#)을 참조하세요.
- 다중 인증(MFA) 필요 - AWS 계정계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 설정합니다. API 작업을 직접 호출할 때 MFA가 필요하다면 정책에 MFA 조건을 추가합니다. 자세한 정보는 IAM 사용자 설명서의 [MFA 보호 API 액세스 구성](#)을 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용자 설명서의 [IAM의 보안 모범 사례](#)를 참조하세요.

리소스 수준 권한은 특정 AWS Glue 객체에만 적용됨

AWS Glue에서 특정 객체에 대한 세분화된 제어만 정의할 수 있습니다. 따라서 Resource 문에 대한 Amazon 리소스 이름(ARN)을 허용하는 API 작업이 ARN을 허용하지 않는 API 작업과 혼합되지 않도록 클라이언트의 IAM 정책을 작성해야 합니다.

예를 들어, 다음 IAM 정책은 GetClassifier 및 GetJobRun에 대한 API 작업을 허용합니다. AWS Glue는 식별자와 작업 실행에 ARN을 허용하지 않기 때문에 이 정책은 Resource를 *로 정의합니다. ARN은 GetDatabase 및 GetTable 같은 특정 API 작업에 대해 허용이 되기 때문에 정책의 후반부에서 ARN을 지정할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetClassifier*",
        "glue:GetJobRun*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "glue:Get*"
      ],
      "Resource": [
        "arn:aws:glue:us-east-1:123456789012:catalog",
        "arn:aws:glue:us-east-1:123456789012:database/default",
        "arn:aws:glue:us-east-1:123456789012:table/default/e*1*",
        "arn:aws:glue:us-east-1:123456789012:connection/connection2"
      ]
    }
  ]
}
```

ARN을 허용하는 AWS Glue 객체 목록은 [리소스 ARN](#) 단원을 참조하십시오.

AWS Glue Console에서 사용

AWS Glue Console에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은 AWS 계정에서 AWS Glue 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다. 최소 필수 권한보다 더

제한적인 자격 증명 기반 정책을 만들면 콘솔이 해당 정책에 연결된 엔터티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 AWS Glue Console을 여전히 사용할 수 있도록 하려면 AWS Glue *ConsoleAccess* 또는 *ReadOnly* AWS 관리형 정책을 엔터티에 추가합니다. 자세한 내용은 IAM 사용 설명서의 [사용자에게 권한 추가](#)를 참조하십시오.

사용자가 AWS Glue 콘솔로 작업하려면 해당 사용자는 AWS 계정에 대한 AWS Glue 리소스로 작업하도록 허용하는 최소 권한 집합이 있어야 합니다. 이 AWS Glue 권한 이외에 콘솔은 다음 서비스로부터 권한을 필요로 합니다.

- 로그를 표시할 수 있는 Amazon CloudWatch Logs 권한.
- 역할을 나열하고 전달할 수 있는 AWS Identity and Access Management(IAM) 권한.
- 스택으로 작업할 수 있는 AWS CloudFormation 권한.
- VPC, 서브넷, 보안 그룹, 인스턴스 및 기타 객체를 나열할 수 있는 Amazon Elastic Compute Cloud(Amazon EC2) 권한.
- 버킷과 객체를 나열하고 스크립트를 검색하고 저장할 수 있는 Amazon Simple Storage Service(Amazon S3) 권한.
- 클러스터 작업을 위한 Amazon Redshift 권한.
- 인스턴스를 나열할 수 있는 Amazon Relational Database Service(Amazon RDS) 권한.

사용자가 AWS Glue 콘솔을 보고 작업하는 데 필요한 권한에 대한 자세한 내용은 [3단계: AWS Glue에 액세스하는 사용자 또는 그룹에 정책 연결](#) 단원을 참조하세요.

최소 필수 권한보다 더 제한적인 IAM 정책을 만들면 콘솔은 해당 IAM 정책에 연결된 사용자에게 의도대로 작동하지 않습니다. 이 사용자가 AWS Glue Console을 사용할 수 있도록 하려면 *AWSGlueConsoleFullAccess* 관리형 정책을 사용자에게 연결합니다([AWS에 대한 관리형\(사전 정의된\) 정책 AWS Glue](#) 참조).

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예시는 IAM 사용자가 자신의 사용자 자격 증명에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI나 AWS API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

테이블에 대한 읽기 전용 권한 부여

다음 정책은 데이터베이스 db1에 있는 테이블 books에 대한 읽기 전용 권한을 부여합니다. Amazon 리소스 이름(ARN)에 대한 자세한 내용은 [Data Catalog ARN](#) 단원을 참조하십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTablesActionOnBooks",
```

```

    "Effect": "Allow",
    "Action": [
      "glue:GetTables",
      "glue:GetTable"
    ],
    "Resource": [
      "arn:aws:glue:us-west-2:123456789012:catalog",
      "arn:aws:glue:us-west-2:123456789012:database/db1",
      "arn:aws:glue:us-west-2:123456789012:table/db1/books"
    ]
  }
]
}

```

이 정책은 데이터베이스 db1에 있는 테이블 books에 읽기 전용 권한을 부여합니다. 테이블에 Get 권한을 부여하려면 카탈로그 및 데이터베이스 리소스에 대한 권한도 필요합니다.

다음 정책은 데이터베이스 db1에서 테이블 tb1을 생성하기 위해 필요한 최소 권한을 부여합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:table/db1/tb11",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:catalog"
      ]
    }
  ]
}

```

GetTables 권한으로 테이블 필터링

데이터베이스 db1에 customers, stores 및 store_sales와 같은 3개의 테이블이 있다고 가정합니다. 다음 정책은 customers는 제외하고 stores 및 store_sales에 대한 GetTables 권한을 부여합니다. 이 정책으로 GetTables을 호출하면 결과에는 인증된 테이블 두 개만 포함됩니다. customers 테이블은 반환되지 않습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTablesExample",
      "Effect": "Allow",
      "Action": [
        "glue:GetTables"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/store_sales",
        "arn:aws:glue:us-west-2:123456789012:table/db1/stores"
      ]
    }
  ]
}
```

store로 시작하는 모든 테이블 이름과 일치하도록 store*를 사용하여 이전 정책을 간소화할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTablesExample2",
      "Effect": "Allow",
      "Action": [
        "glue:GetTables"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/store*"
      ]
    }
  ]
}
```

마찬가지로, db1의 모든 테이블과 일치하도록 /db1/*을 사용하면 다음 정책은 db1의 모든 테이블에 대한 GetTables 액세스 권한을 부여합니다.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTablesReturnAll",
      "Effect": "Allow",
      "Action": [
        "glue:GetTables"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/*"
      ]
    }
  ]
}
```

테이블 ARN을 제공하지 않으면 GetTables 호출은 성공하지만 빈 목록이 반환됩니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTablesEmptyResults",
      "Effect": "Allow",
      "Action": [
        "glue:GetTables"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1"
      ]
    }
  ]
}
```

데이터베이스 ARN이 정책에 누락된 경우에는 GetTables 호출이 실패하고 AccessDeniedException이 발생합니다.

```
{
  "Version": "2012-10-17",
```

```

    "Statement": [
      {
        "Sid": "GetTablesAccessDeny",
        "Effect": "Allow",
        "Action": [
          "glue:GetTables"
        ],
        "Resource": [
          "arn:aws:glue:us-west-2:123456789012:catalog",
          "arn:aws:glue:us-west-2:123456789012:table/db1/*"
        ]
      }
    ]
  }
}

```

테이블 및 모든 파티션에 전체 액세스 권한 부여

다음 정책은 데이터베이스 db1에 있는 books 테이블에 대한 모든 권한을 부여합니다. 여기에는 테이블 자체, 테이블의 보관된 버전 및 테이블의 모든 파티션에 대한 읽기 및 쓰기 권한이 포함됩니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FullAccessOnTable",
      "Effect": "Allow",
      "Action": [
        "glue:CreateTable",
        "glue:GetTable",
        "glue:GetTables",
        "glue:UpdateTable",
        "glue>DeleteTable",
        "glue:BatchDeleteTable",
        "glue:GetTableVersion",
        "glue:GetTableVersions",
        "glue>DeleteTableVersion",
        "glue:BatchDeleteTableVersion",
        "glue:CreatePartition",
        "glue:BatchCreatePartition",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:BatchGetPartition",
        "glue:UpdatePartition",

```

```

        "glue:DeletePartition",
        "glue:BatchDeletePartition"
    ],
    "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/books"
    ]
}
]
}

```

실제로 이전 정책은 다음과 같이 간소화할 수 있습니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "FullAccessOnTable",
      "Effect": "Allow",
      "Action": [
        "glue:*Table*",
        "glue:*Partition*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/db1",
        "arn:aws:glue:us-west-2:123456789012:table/db1/books"
      ]
    }
  ]
}

```

세분화된 액세스 제어의 최소 세부 수준은 테이블 수준입니다. 즉, 사용자에게 테이블의 파티션 중 일부 또는 테이블 열 중 일부에 대한 권한을 부여할 수는 없습니다. 사용자는 테이블 전체에 대한 액세스 권한을 갖거나 해당 테이블에 대한 권한을 갖지 못하거나 둘 중 하나입니다.

이름 접두사 및 명시적 거부로 액세스 제어

이 예제에서는 AWS Glue Data Catalog의 데이터베이스 및 테이블이 이름 접두사를 사용하여 구성되었다고 가정합니다. 개발 스테이지의 데이터 베이스에 이름 접두사 dev-가 있고, 프로덕션 스테이지의 데이터베이스에는 이름 접두사 prod-가 있습니다. 다음 정책을 사용하여 이름 접두사가 dev-인

모든 데이터베이스, 테이블, UDF 등에 대한 모든 액세스를 개발자에게 부여할 수 있습니다. 하지만 이 름 접두사가 prod-인 모든 항목에는 읽기 전용 액세스 권한을 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DevAndProdFullAccess",
      "Effect": "Allow",
      "Action": [
        "glue:*Database*",
        "glue:*Table*",
        "glue:*Partition*",
        "glue:*UserDefinedFunction*",
        "glue:*Connection*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:catalog",
        "arn:aws:glue:us-west-2:123456789012:database/dev-*",
        "arn:aws:glue:us-west-2:123456789012:database/prod-*",
        "arn:aws:glue:us-west-2:123456789012:table/dev-*/**",
        "arn:aws:glue:us-west-2:123456789012:table/*/dev-*",
        "arn:aws:glue:us-west-2:123456789012:table/prod-*/**",
        "arn:aws:glue:us-west-2:123456789012:table/*/prod-*",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/dev-*/**",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/dev-*",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/prod-*/**",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/prod-*",
        "arn:aws:glue:us-west-2:123456789012:connection/dev-*",
        "arn:aws:glue:us-west-2:123456789012:connection/prod-*"
      ]
    },
    {
      "Sid": "ProdWriteDeny",
      "Effect": "Deny",
      "Action": [
        "glue:*Create*",
        "glue:*Update*",
        "glue:*Delete*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:123456789012:database/prod-*",
        "arn:aws:glue:us-west-2:123456789012:table/prod-*/**",

```

```

        "arn:aws:glue:us-west-2:123456789012:table/*/prod-*",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/prod-*/*",
        "arn:aws:glue:us-west-2:123456789012:userDefinedFunction/*/prod-*",
        "arn:aws:glue:us-west-2:123456789012:connection/prod-*"
    ]
}
]
}

```

이전 정책의 두 번째 문에서는 명시적 deny를 사용합니다. 명시적 deny를 사용해 보안 주체에 부여된 모든 allow 권한을 덮어쓸 수 있습니다. 그러면 중요한 리소스에 대한 액세스를 잠그고 다른 정책이 이러한 리소스에 대해 실수로 액세스 권한을 부여하지 않도록 방지할 수 있습니다.

이전 예제에서는 첫 번째 문이 prod- 리소스에 대한 모든 액세스 권한을 부여하더라도 두 번째 문이 리소스에 대한 쓰기 액세스 권한을 명시적으로 취소해 prod- 리소스에 대해 일기 전용 액세스만 남아 있게 됩니다.

태그를 사용한 액세스 권한 부여

예를 들어 트리거 t2에 대한 액세스를 계정에 있는 Tom이라는 특정 사용자로 제한하려 한다고 가정합니다. Sam을 포함해 다른 모든 사용자들은 t1을 트리거하기 위한 액세스 권한을 갖습니다. 트리거 t1 및 t2는 다음과 같은 속성을 갖추고 있습니다.

```

aws glue get-triggers
{
  "Triggers": [
    {
      "State": "CREATED",
      "Type": "SCHEDULED",
      "Name": "t1",
      "Actions": [
        {
          "JobName": "j1"
        }
      ],
      "Schedule": "cron(0 0/1 * * ? *)"
    },
    {
      "State": "CREATED",
      "Type": "SCHEDULED",
      "Name": "t2",
      "Actions": [

```

```

        {
            "JobName": "j1"
        }
    ],
    "Schedule": "cron(0 0/1 * * ? *)"
}
]
}

```

t2를 트리거하기 위한 태그 값 Tom(`aws:ResourceTag/Name": "Tom"`)에 AWS Glue 관리자가 연결되어 있습니다. 또한 AWS Glue 관리자는 태그를 기준으로 조건 문과 함께 IAM 정책을 Tom에게 제공했습니다. 결과적으로 Tom은 태그 값 Tom으로 리소스에서 수행되는 AWS Glue 작업만 사용할 수 있습니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "glue:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Name": "Tom"
        }
      }
    }
  ]
}

```

Tom이 트리거 t1에 대한 액세스를 시도하면 액세스 거부 메시지가 수신됩니다. 한편, Tom은 트리거 t2를 성공적으로 검색할 수 있습니다.

```
aws glue get-trigger --name t1
```

```
An error occurred (AccessDeniedException) when calling the GetTrigger operation:
User: Tom is not authorized to perform: glue:GetTrigger on resource: arn:aws:glue:us-east-1:123456789012:trigger/t1
```

```
aws glue get-trigger --name t2
```

```
{
  "Trigger": {
```

```

    "State": "CREATED",
    "Type": "SCHEDULED",
    "Name": "t2",
    "Actions": [
      {
        "JobName": "j1"
      }
    ],
    "Schedule": "cron(0 0/1 * * ? *)"
  }
}

```

이 작업은 태그에 대한 필터링을 지원하지 않기 때문에 Tom은 여러 개의 GetTriggers API 작업을 사용하여 트리거를 나열할 수 없습니다.

Tom에게 GetTriggers에 대한 액세스 권한을 부여하기 위해 AWS Glue 관리자는 권한을 두 개의 섹션으로 분할하는 정책을 생성합니다. 첫 번째 섹션에서는 Tom에게 GetTriggers API 작업에서 모든 트리거에 액세스할 수 있도록 허용합니다. 두 번째 섹션에서는 Tom에게 Tom으로 태그 값이 지정된 API 작업에 액세스할 수 있도록 허용합니다. 이 정책에 따라 Tom에게는 트리거 t2에 대한 GetTriggers 및 GetTrigger 액세스가 모두 허용됩니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "glue:GetTriggers",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "glue:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Name": "Tom"
        }
      }
    }
  ]
}

```

태그를 사용한 액세스 거부

또 다른 리소스 정책 접근 방식은 리소스에 대한 액세스를 명시적으로 거부하는 것입니다.

Important

명시적 거부 정책은 GetTriggers와 같은 여러 개의 API 작업에 적용되지 않습니다.

다음 예제 정책에서는 모든 AWS Glue 작업이 허용됩니다. 그러나 두 번째 Effect 명령문은 Team 키와 Special 값으로 태그가 지정된 작업에 대한 액세스를 명시적으로 거부합니다.

관리자가 자격 증명에 다음 정책을 연결하면 해당 자격 증명은 Team 키와 Special 값이 태그로 지정된 작업을 제외한 모든 작업에 액세스할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "glue:*",
      "Resource": "arn:aws:glue:us-east-1:123456789012:job/*"
    },
    {
      "Effect": "Deny",
      "Action": "glue:*",
      "Resource": "arn:aws:glue:us-east-1:123456789012:job/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Team": "Special"
        }
      }
    }
  ]
}
```

목록 및 배치 API 작업과 함께 태그 사용

리소스 정책을 작성하는 세 번째 접근 방식은 태그 값에 대해 리소스를 나열하도록 List API 작업을 사용해 리소스에 대한 액세스를 허용하는 것입니다. 그런 다음, 해당되는 Batch API 작업을 사용하여 특정 리소스의 세부 정보에 대한 액세스를 허용합니다. 이러한 접근 방식에서는 관리자가 여러 개의

GetCrawlers, GetDevEndpoints, GetJobs 또는 GetTriggers API 작업에 대한 액세스를 허용할 필요가 없습니다. 대신에 다음 API 작업을 통해 리소스를 나열할 수 있도록 허용할 수 있습니다.

- ListCrawlers
- ListDevEndpoints
- ListJobs
- ListTriggers

그리고 다음 API 작업을 통해 개별 리소스에 대한 세부 정보를 확인할 수 있도록 허용할 수 있습니다.

- BatchGetCrawlers
- BatchGetDevEndpoints
- BatchGetJobs
- BatchGetTriggers

관리자는 이러한 접근 방식을 사용하기 위해 다음을 수행할 수 있습니다.

1. 크롤러, 개발 엔드포인트, 작업 및 트리거에 태그를 추가합니다.
2. GetCrawlers, GetDevEndpoints, GetJobs, GetTriggers 같은 Get API 작업에 대한 사용자 액세스를 거부합니다.
3. 사용자가 액세스 권한을 가진 태그 지정 리소스를 확인할 수 있도록 ListCrawlers, ListDevEndpoints, ListJobs, ListTriggers 같은 List API 작업에 대한 사용자 액세스를 허용합니다.
4. TagResource 및 UntagResource 같은 AWS Glue 태그 지정 API에 대한 사용자 액세스를 거부합니다.
5. BatchGetCrawlers, BatchGetDevEndpoints, BatchGetJobs, BatchGetTriggers 같은 BatchGet API 작업을 통해 리소스 세부 정보에 대한 사용자 액세스를 허용합니다.

예를 들어, ListCrawlers 작업을 호출할 때 사용자 이름과 일치하도록 태그 값을 제공합니다. 그러면 결과는 제공된 태그 값과 일치하는 크롤러의 목록이 됩니다. 주어진 태그를 사용해 각 크롤러에 대한 세부 정보를 얻을 수 있도록 BatchGetCrawlers에 이름 목록을 제공합니다.

예를 들어 Tom이 Tom으로 태그가 지정된 트리거의 세부 정보만 검색할 수 있도록 하려면 관리자가 Tom에서 트리거에 태그를 추가하고, 모든 사용자에게 대해 GetTriggers API 작업 액세스를 거부하고, 모든 사용자에게 대해 ListTriggers 및 BatchGetTriggers 액세스를 허용할 수 있습니다.

다음은 AWS Glue 관리자가 Tom에게 부여하는 리소스 정책입니다. 이 정책의 첫 번째 섹션에서는 GetTriggers에 대한 AWS Glue API 작업이 거부됩니다. 정책의 두 번째 섹션에서는 모든 리소스에 대해 ListTriggers가 허용됩니다. 그러나 세 번째 섹션에서는 Tom이라고 태그가 지정된 리소스에게 BatchGetTriggers 액세스가 허용됩니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "glue:GetTriggers",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "glue:ListTriggers"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "glue:BatchGetTriggers"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Name": "Tom"
        }
      }
    }
  ]
}
```

Tom은 이전 예제와 똑같은 트리거를 사용하여 트리거 t2에 액세스할 수 있지만, 트리거 t1에는 액세스할 수 없습니다. 다음 예제는 Tom이 BatchGetTriggers를 통해 t1 및 t2에 액세스를 시도할 때의 결과를 보여줍니다.

```
aws glue batch-get-triggers --trigger-names t2
{
  "Triggers": {
    "State": "CREATED",
    "Type": "SCHEDULED",
    "Name": "t2",
    "Actions": [
      {
        "JobName": "j2"
      }
    ],
    "Schedule": "cron(0 0/1 * * ? *)"
  }
}
```

```
aws glue batch-get-triggers --trigger-names t1
```

An error occurred (AccessDeniedException) when calling the BatchGetTriggers operation:
No access to any requested resource.

다음 예제는 Tom이 동일한 BatchGetTriggers 호출에서 트리거 t2와 트리거 t3(존재하지 않음) 모두에 액세스를 시도할 때의 결과를 보여줍니다. Tom은 트리거 t2에 대한 액세스 권한을 가지고 있고 이 트리거가 존재하기 때문에 t2만 반환됩니다. Tom에게 트리거 t3에 대한 액세스가 허용되지만 트리거 t3가 존재하지 않기 때문에 "TriggersNotFound": [] 목록에서 응답으로 t3가 반환됩니다.

```
aws glue batch-get-triggers --trigger-names t2 t3
{
  "Triggers": {
    "State": "CREATED",
    "Type": "SCHEDULED",
    "Name": "t2",
    "Actions": [
      {
        "JobName": "j2"
      }
    ],
    "TriggersNotFound": ["t3"],
    "Schedule": "cron(0 0/1 * * ? *)"
  }
}
```

조건 키 또는 컨텍스트 키를 사용하여 설정 제어

작업 생성 및 업데이트 권한을 부여할 때 조건 키 또는 컨텍스트 키를 사용할 수 있습니다. 다음 섹션에서 키에 대해 설명합니다.

- [조건 키를 사용하여 설정을 제어하는 정책 제어](#)
- [컨텍스트 키를 사용하여 설정을 제어하는 정책 제어](#)

조건 키를 사용하여 설정을 제어하는 정책 제어

AWS Glue는 `glue:VpcIds`, `glue:SubnetIds` 및 `glue:SecurityGroupIds`의 세 가지 IAM 조건 키를 제공합니다. 작업 생성 및 업데이트 권한을 부여할 때 조건 키를 IAM 정책에 사용할 수 있습니다. 이 설정을 사용하여 원하는 VPC 환경 외부에서 실행하려는 경우 작업 또는 세션을 생성하거나 업데이트하지 않도록 할 수 있습니다. VPC 설정 정보는 `CreateJob` 요청에서 직접 입력하지 않고 AWS Glue 연결을 가리키는 작업 '연결' 필드에서 유추됩니다.

사용 예

원하는 `VpcId` 'vpc-id1234', `SubnetIds`, `SecurityGroupIds`를 사용하여 '트래픽 모니터링 연결'이라는 AWS Glue 네트워크 유형 연결을 생성합니다.

IAM 정책에서 `CreateJob` 및 `UpdateJob` 작업에 대한 조건 키 조건을 지정합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "glue:CreateJob",
    "glue:UpdateJob"
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "ForAnyValue:StringLike": {
      "glue:VpcIds": [
        "vpc-id1234"
      ]
    }
  }
}
```

연결 정보를 지정하지 않고 AWS Glue 작업을 생성할 수 없도록 하는 유사한 IAM 정책을 생성할 수 있습니다.

VPC의 세션 제한

지정된 VPC 내에서 생성된 세션을 실행하도록 하려면 `glue:vpc-id`가 `vpc-<123>`이 아니라는 조건에서 `glue:CreateSession` 작업에 Deny 적용을 추가하여 역할 권한을 제한합니다. 예:

```
"Effect": "Deny",
"Action": [
  "glue:CreateSession"
],
"Condition": {
  "StringNotEquals" : {"glue:VpcIds" : ["vpc-123"]}
}
```

또한 `glue:vpc-id`가 null이라는 조건에서 `glue:CreateSession` 작업에 Deny 적용을 추가하여 VPC 내에서 생성된 세션을 실행하도록 할 수도 있습니다. 예:

```
{
  "Effect": "Deny",
  "Action": [
    "glue:CreateSession"
  ],
  "Condition": {
    "Null": {"glue:VpcIds": true}
  }
},
{
  "Effect": "Allow",
  "Action": [
    "glue:CreateSession"
  ],
  "Resource": ["*"]
}
```

컨텍스트 키를 사용하여 설정을 제어하는 정책 제어

AWS Glue는 작업 및 개발자 엔드포인트에서 AWS Glue를 통해 사용할 수 있는 각 역할 세션에 컨텍스트 키(`glue:CredentialIssuingService= glue.amazonaws.com`)를 제공합

니다. 이렇게 하면 AWS Glue 스크립트에서 수행하는 작업에 대한 보안 제어를 구현할 수 있습니다. AWS Glue는 고객을 대신하여 AWS Glue가 다른 AWS 서비스를 호출(작업/개발 엔드포인트가 아닌 AWS Glue 서비스를 통해 직접 호출)하는 각 역할 세션에 다른 컨텍스트 키(`glue:RoleAssumedBy=glue.amazonaws.com`)를 제공합니다.

사용 예

IAM 정책에서 조건부 권한을 지정하고 AWS Glue 작업에서 사용할 역할에 이 권한을 연결합니다. 이렇게 하면 역할 세션이 AWS Glue 작업 런타임 환경에 사용되는지 여부에 따라 특정 작업이 허용/거부됩니다.

```
{
  "Effect": "Allow",
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::confidential-bucket/*",
  "Condition": {
    "StringEquals": {
      "glue:CredentialIssuingService": "glue.amazonaws.com"
    }
  }
}
```

ID가 데이터 미리 보기 세션을 생성하지 못하게 하기

이 섹션에는 ID가 데이터 미리 보기 세션을 생성하지 못하게 하는 데 사용되는 IAM 정책 예제가 포함되어 있습니다. 실행 중 데이터 미리 보기 세션에서 사용되는 역할과 별개인 ID에 이 정책을 연결합니다.

```
{
  "Sid": "DatapreviewDeny",
  "Effect": "Deny",
  "Action": [
    "glue:CreateSession"
  ],
  "Resource": [
    "arn:aws:glue:*:*:session/glue-studio-datapreview*"
  ]
}
```

AWS Glue용 리소스 기반 정책 예제

이 섹션에는 크로스 계정 액세스 권한을 부여하는 정책을 비롯해 리소스 기반 정책 예제가 포함되어 있습니다.

다음 예제에서는 AWS Command Line Interface(AWS CLI)를 사용하여 AWS Glue 서비스 API 작업과 상호 작용합니다. AWS Glue 콘솔에서 또는 AWS SDK 중 하나를 사용하여 동일한 작업을 수행할 수 있습니다.

Important

AWS Glue 리소스 정책을 변경하면 계정 내 기존 AWS Glue 사용자에게 대한 권한을 실수로 취소해 예기치 않은 중단을 일으킬 수 있습니다. 다음 예제는 개발 또는 테스트 계정에서만 시도해 보고, 변경하기 전에 이러한 예제가 기존 워크플로우에 문제를 일으키지 않는지 확인하십시오.

주제

- [AWS Glue에서 리소스 기반 정책 사용 시 고려 사항](#)
- [리소스 정책을 사용하여 동일한 계정에서 액세스 제어](#)

AWS Glue에서 리소스 기반 정책 사용 시 고려 사항

Note

IAM 정책과 AWS Glue 리소스 정책을 둘 다 전파하려면 몇 초 가량 걸립니다. 새 정책을 연결하더라도 새 정책이 시스템 전체에서 전파되기까지 이전 정책이 계속해서 적용됩니다.

JSON 형식으로 작성된 정책 문서를 사용해 리소스 정책을 생성하거나 수정할 수 있습니다. 정책 구문은 자격 증명 기반 IAM 정책에 대해 동일하지만([IAM JSON 정책 참조](#) 확인) 다음과 같은 예외가 있습니다.

- 각 정책 설명에는 "Principal" 또는 "NotPrincipal" 블록이 필요합니다.
- "Principal" 또는 "NotPrincipal"은 유효한 기존 보안 주체를 식별해야 합니다. 와일드카드 패턴(예: `arn:aws:iam::account-id:user/*`)은 허용되지 않습니다.
- 정책의 "Resource" 블록에는 다음 정규식 구문과 일치하는 모든 리소스 ARN이 필요합니다. 여기서 첫 번째 %s는 *region*이고 두 번째 %s는 *account-id*입니다.

```
*arn:aws:glue:%s:%s:(\*[a-zA-Z\*]+\/*?.*)
```

예를 들어, `arn:aws:glue:us-west-2:account-id:*`와 `arn:aws:glue:us-west-2:account-id:database/default` 둘 다 허용되지만 `*`는 허용되지 않습니다.

- 자격 증명 기반 정책과 달리 AWS Glue 리소스 정책에는 정책이 연결된 카탈로그에 속한 리소스의 Amazon 리소스 이름(ARN)만 포함되어야 합니다. 이러한 ARN은 항상 `arn:aws:glue:`로 시작합니다.
- 정책은 자격 증명이 추가 정책 생성 또는 수정을 수행할 수 없도록 잠기게 만들 수는 없습니다.
- 리소스 정책 JSON 문서는 크기가 10KB를 초과할 수 없습니다.

리소스 정책을 사용하여 동일한 계정에서 액세스 제어

이 예제에서 계정 A의 관리 사용자는 계정 A의 IAM 사용자인 Alice에게 카탈로그에 대한 모든 액세스 권한을 부여하는 리소스 정책을 생성합니다. Alice에게는 연결된 IAM 정책이 없습니다.

이렇게 하기 위해 관리 사용자가 다음 AWS CLI 명령을 실행합니다.

```
# Run as admin of Account A
$ aws glue put-resource-policy --profile administrator-name --region us-west-2 --
policy-in-json '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": {
        "AWS": [
          "arn:aws:iam::account-A-id:user/Alice"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:account-A-id:*"
      ]
    }
  ]
}'
```


JSON 정책 문서를 AWS CLI 명령의 일부로 입력하는 대신 정책 문서를 파일로 저장한 다음 AWS CLI 명령에서 파일 경로(file://로 접두사가 지정됨)를 참조할 수 있습니다. 다음은 이렇게 할 수 있는 방법을 보여주는 예제입니다.

```
$ echo '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Principal": {
        "AWS": [
          "arn:aws:iam::account-A-id:user/Alice"
        ]
      },
      "Effect": "Allow",
      "Action": [
        "glue:*"
      ],
      "Resource": [
        "arn:aws:glue:us-west-2:account-A-id:*"
      ]
    }
  ]
}' > /temp/policy.json

$ aws glue put-resource-policy --profile admin1 \
  --region us-west-2 --policy-in-json file:///temp/policy.json
```

리소스 정책이 전파된 후 Alice는 다음과 같이 계정 A에 있는 모든 AWS Glue 리소스에 액세스할 수 있습니다.

```
# Run as user Alice
$ aws glue create-database --profile alice --region us-west-2 --database-input '{
  "Name": "new_database",
  "Description": "A new database created by Alice",
  "LocationUri": "s3://my-bucket"
}'

$ aws glue get-table --profile alice --region us-west-2 --database-name "default" --
table-name "tbl1"}
```

Alice의 get-table 호출에 대한 응답으로 AWS Glue 서비스는 다음을 반환합니다.

```
{
  "Table": {
    "Name": "tbl1",
    "PartitionKeys": [],
    "StorageDescriptor": {
      .....
    },
    .....
  }
}
```

AWSAWS Glue에 대한 관리형 정책

AWS 관리형 정책은 에서 생성하고 관리하는 독립형 정책입니다. AWS AWS 관리형 정책은 많은 일반 사용 사례에 대한 권한을 제공하도록 설계되었으므로 사용자, 그룹 및 역할에 권한을 할당하기 시작할 수 있습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수도 있다는 점에 유의하세요. 사용 사례에 고유한 [고객 관리형 정책](#)을 정의하여 권한을 줄이는 것이 좋습니다.

관리형 정책에 정의된 권한은 변경할 수 없습니다. AWS AWS 관리형 정책에 정의된 권한을 업데이트 하는 경우 AWS 해당 업데이트는 정책이 연결된 모든 주체 ID (사용자, 그룹, 역할) 에 영향을 미칩니다. AWS 새 API 작업이 시작되거나 기존 서비스에 새 AWS 서비스 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 가장 높습니다.

자세한 내용은 IAM 사용자 설명서의 [AWS 관리형 정책](#)을 참조하세요.

AWS 에 대한 관리형 (사전 정의된) 정책 AWS Glue

AWS 에서 생성하고 관리하는 독립형 IAM 정책을 제공하여 많은 일반적인 사용 사례를 해결합니다. AWS 이러한 AWS 관리형 정책은 일반적인 사용 사례에 필요한 권한을 부여하므로 필요한 권한을 조사하지 않아도 됩니다. 자세한 내용은 IAM 사용자 설명서의 [AWS 관리형 정책](#)을 참조하세요.

계정의 ID에 연결할 수 있는 다음과 같은 AWS 관리형 정책은 사용 사례 시나리오에만 적용되며 사용 사례 시나리오별로 그룹화되어 있습니다. AWS Glue

- [AWSGlueConsoleFullAccess](#)— 정책이 연결된 자격 증명이 를 사용하는 경우 AWS Glue 리소스에 대한 전체 액세스 권한을 부여합니다. AWS Management Console이 정책에 지정된 리소스의 이름 변환을 따르면 사용자는 콘솔 전체 용량을 소유합니다. 이 정책은 보통 AWS Glue 콘솔의 사용자에게 해당됩니다.

- [AWSGlueServiceRole](#)— 다양한 AWS Glue 프로세스를 사용자 대신 실행하는 데 필요한 리소스에 대한 액세스 권한을 부여합니다. 이러한 리소스에는 Amazon S3, IAM, CloudWatch 로그 및 Amazon EC2가 포함됩니다. 이 정책에 지정된 리소스의 이름 변환을 따르고자 한다면 AWS Glue 절차는 필요한 권한을 소유합니다. 이 정책은 크롤러, 작업 및 개발 엔드포인트를 정의할 때 지정된 역할에 일반적으로 추가됩니다.
- [AwsGlueSessionUserRestrictedServiceRole](#)— 세션을 제외한 모든 AWS Glue 리소스에 대한 전체 액세스 권한을 제공합니다. 사용자와 연결된 대화형 세션만 사용자가 생성하고 사용할 수 있도록 허용합니다. 이 정책에는 다른 AWS 서비스의 AWS Glue 리소스를 관리하는 AWS Glue 데 필요한 기타 권한이 포함됩니다. 또한 이 정책에서는 다른 AWS 서비스의 AWS Glue 리소스에 태그를 추가할 수 있습니다.

Note

전체 보안 이점을 얻으려면 `AWSGlueServiceRole`, `AWSGlueConsoleFullAccess` 또는 `AWSGlueConsoleSageMakerNotebookFullAccess` 정책이 할당된 사용자에게 이 정책을 부여하지 않습니다.

- [AwsGlueSessionUserRestrictedPolicy](#)— 담당자의 AWS 사용자 ID와 일치하는 태그 키 “소유자”와 값이 제공된 경우에만 `CreateSession` API 작업을 사용하여 AWS Glue 대화형 세션을 생성할 수 있는 액세스 권한을 제공합니다. 이 자격 증명 정책은 `CreateSession` API 작업을 호출하는 IAM 사용자에게 연결됩니다. 또한 이 정책은 담당자가 자신의 사용자 ID와 일치하는 “소유자” 태그 및 값으로 생성된 AWS Glue 대화형 세션 리소스와 상호 작용할 수 있도록 허용합니다. AWS 이 정책은 세션이 생성된 후 AWS Glue 세션 리소스에서 'owner' 태그를 변경하거나 제거할 수 있는 권한을 거부합니다.

Note

전체 보안 이점을 얻으려면 `AWSGlueServiceRole`, `AWSGlueConsoleFullAccess` 또는 `AWSGlueConsoleSageMakerNotebookFullAccess` 정책이 할당된 사용자에게 이 정책을 부여하지 않습니다.

- [AwsGlueSessionUserRestrictedNotebookServiceRole](#)— 특정 AWS Glue 대화형 세션 리소스와 상호 작용할 수 있도록 AWS Glue Studio 노트북 세션에 대한 충분한 액세스 권한을 제공합니다. 이러한 리소스는 노트북을 생성하는 주체 (IAM 사용자 또는 역할) 의 AWS 사용자 ID와 일치하는 “소유자” 태그 값으로 생성되는 리소스입니다. 이러한 태그에 대한 자세한 내용은 IAM 사용 설명서의 [보안 주체 키 값](#) 차트를 참조하세요.

이 서비스-역할 정책은 노트북 내에서 매직 명령문으로 지정되었거나 CreateSession API 작업에 역할로 전달된 역할에 연결됩니다. 또한 이 정책은 태그 키 “소유자”와 값이 보안 주체의 AWS 사용자 ID와 일치하는 경우에만 보안 주체가 AWS Glue Studio 노트북 인터페이스에서 AWS Glue 대화형 세션을 생성할 수 있도록 허용합니다. 이 정책은 세션이 생성된 후 AWS Glue 세션 리소스에서 'owner' 태그를 변경하거나 제거할 수 있는 권한을 거부합니다. 이 정책에는 Amazon S3 버킷 쓰기 및 읽기, CloudWatch 로그 작성, 에서 사용하는 Amazon EC2 리소스의 태그 생성 및 삭제에 대한 권한도 포함됩니다. AWS Glue

Note

전체 보안 이점을 얻으려면 AWSGlueServiceRole, AWSGlueConsoleFullAccess 또는 AWSGlueConsoleSageMakerNotebookFullAccess 정책이 할당된 역할에 이 정책을 부여하지 않습니다.

- [AwsGlueSessionUserRestrictedNotebookPolicy](#)— 노트북을 생성한 주체 (IAM 사용자 또는 역할) 의 AWS 사용자 ID와 일치하는 태그 키 “소유자”와 값이 있는 경우에만 AWS Glue Studio 노트북 인터페이스에서 AWS Glue 대화형 세션을 생성할 수 있는 액세스 권한을 제공합니다. 이러한 태그에 대한 자세한 내용은 IAM 사용 설명서의 [보안 주체 키 값](#) 차트를 참조하세요.

이 정책은 AWS Glue Studio 노트북 인터페이스에서 세션을 생성하는 보안 주체(IAM 사용자 또는 역할)에 연결됩니다. 또한 이 정책은 AWS Glue Studio 노트북에 충분히 액세스하여 특정 AWS Glue 대화형 세션 리소스와 상호 작용할 수 있도록 허용합니다. 이러한 리소스는 보안 주체의 AWS 사용자 ID와 일치하는 “owner” 태그 값으로 생성되는 리소스입니다. 이 정책은 세션이 생성된 후 AWS Glue 세션 리소스에서 'owner' 태그를 변경하거나 제거할 수 있는 권한을 거부합니다.

- [AWSGlueServiceNotebookRole](#)— AWS Glue Studio 노트북에서 시작된 AWS Glue 세션에 대한 액세스 권한을 부여합니다. 이 정책은 모든 세션의 세션 정보를 나열하고 가져올 수 있도록 허용하지만 사용자가 자신의 AWS 사용자 ID로 태그가 지정된 세션을 만들고 사용할 수만 허용합니다. 이 정책은 해당 ID로 태그가 지정된 AWS Glue 세션 리소스에서 “소유자” 태그를 변경하거나 제거할 수 있는 권한을 거부합니다. AWS

에서 노트북 인터페이스를 사용하여 작업을 생성하는 AWS 사용자에게 이 정책을 할당하십시오.
AWS Glue Studio

- [AWSGlueConsoleSageMakerNotebookFullAccess](#)— 정책이 연결된 ID가 를 사용하는 경우 AWS Glue 및 SageMaker 리소스에 대한 전체 액세스 권한을 AWS Management Console부여합니다. 이 정책에 지정된 리소스의 이름 변환을 따르면 사용자는 콘솔 전체 용량을 소유합니다. 이 정책은 일반적으로 SageMaker 노트북을 관리하는 AWS Glue 콘솔 사용자에게 적용됩니다.

- [AWSGlueSchemaRegistryFullAccess](#)— 정책이 연결된 ID가 AWS Glue OR를 사용하는 경우 스키마 레지스트리 리소스에 대한 전체 액세스 권한을 부여합니다 AWS Management Console . AWS CLI 이 정책에 지정된 리소스의 이름 변환을 따르면 사용자는 콘솔 전체 용량을 소유합니다. 이 정책은 일반적으로 AWS Glue 콘솔 사용자 또는 AWS Glue 스키마 레지스트리를 관리하는 AWS CLI 사용자에게 연결됩니다.
- [AWSGlueSchemaRegistryReadOnlyAccess](#)— 정책이 연결된 ID가 OR를 사용하는 경우 AWS Glue 스키마 레지스트리 리소스에 대한 읽기 전용 액세스 권한을 부여합니다 AWS Management Console . AWS CLI 이 정책에 지정된 리소스의 이름 변환을 따르면 사용자는 콘솔 전체 용량을 소유합니다. 이 정책은 일반적으로 AWS Glue 콘솔 사용자 또는 AWS Glue 스키마 레지스트리를 사용하는 AWS CLI 사용자에게 연결됩니다.

Note

IAM 콘솔에 로그인하고 이 콘솔에서 특정 정책을 검색하여 이러한 권한 정책을 검토할 수 있습니다.

AWS Glue 작업 및 리소스에 대한 권한을 허용하는 고유의 사용자 정의 IAM 정책을 생성할 수도 있습니다. 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 사용자 지정 정책을 연결할 수 있습니다.

AWSAWS 관리형 정책에 대한 Glue 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 AWS Glue의 AWS 관리형 정책 업데이트에 대한 세부 정보를 확인하세요. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 AWS Glue Document 기록 페이지에서 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
AwsGlueSessionUserRestrictedPolicy — 기존 정책에 대한 최소한의 업데이트.	glue:StartCompletion 및 glue:GetCompletion 을 정책에 추가합니다. AWS Glue에서 Amazon Q 데이터를 통합하는 데 필요합니다.	2024년 4월 30일

변경 사항	설명	날짜
AwsGlueSessionUserRestrictedNotebookServiceRole — 기존 정책의 사소한 업데이트.	glue:StartCompletion 및 glue:GetCompletion 을 정책에 추가합니다. AWS Glue에서 Amazon Q 데이터를 통합하는 데 필요합니다.	2024년 4월 30일
AwsGlueSessionUserRestrictedServiceRole — 기존 정책의 사소한 업데이트.	glue:StartCompletion 및 glue:GetCompletion 을 정책에 추가합니다. AWS Glue에서 Amazon Q 데이터를 통합하는 데 필요합니다.	2024년 4월 30일
AWSGlueServiceNotebookRole — 기존 정책의 사소한 업데이트.	glue:StartCompletion 및 glue:GetCompletion 을 정책에 추가합니다. AWS Glue에서 Amazon Q 데이터를 통합하는 데 필요합니다.	2024년 1월 30일
AwsGlueSessionUserRestrictedNotebookPolicy — 기존 정책의 사소한 업데이트.	glue:StartCompletion 및 glue:GetCompletion 을 정책에 추가합니다. AWS Glue에서 Amazon Q 데이터를 통합하는 데 필요합니다.	2023년 11월 29일
AWSGlueServiceNotebookRole — 기존 정책에 대한 사소한 업데이트.	codewhisperer:GenerateRecommendations 를 정책에 추가합니다. AWS Glue가 CodeWhisperer 추천을 생성하는 새 기능에 필요합니다.	2023년 10월 9일
AWSGlueServiceRole — 기존 정책에 대한 사소한 업데이트.	AWS Glue 로깅을 더 잘 반영하도록 CloudWatch 권한 범위를 좁히십시오.	2023년 8월 4일

변경 사항	설명	날짜
<p>AWSGlueConsoleFullAccess — 기존 정책에 대한 사소한 업데이트.</p>	<p>databrew 레시피 나열 및 설명 권한을 정책에 추가합니다. AWS Glue가 레시피에 액세스할 수 있는 새 기능에 대한 전체 관리 액세스 권한을 제공하는 데 필요합니다.</p>	<p>2023년 5월 9일</p>
<p>AWSGlueConsoleFullAccess — 기존 정책에 대한 사소한 업데이트.</p>	<p>cloudformation:ListStacks 를 정책에 추가합니다. AWS CloudFormation 인증 요구 사항이 변경된 후에도 기존 기능을 보존합니다.</p>	<p>2023년 3월 28일</p>
<p>대화형 세션 기능에 대해 추가된 새로운 관리형 정책:</p> <ul style="list-style-type: none"> • AwsGlueSessionUserRestrictedServiceRole • AwsGlueSessionUserRestrictedPolicy • AwsGlueSessionUserRestrictedNotebookServiceRole • AwsGlueSessionUserRestrictedNotebookPolicy 	<p>이러한 정책은 AWS Glue Studio에서 대화형 세션과 노트북에 대한 추가 보안을 제공하도록 설계되었습니다. 소유자만 액세스할 수 있도록 정책에서 CreateSession API 작업에 대한 액세스를 제한합니다.</p>	<p>2021년 11월 30일</p>

변경 사항	설명	날짜
AWSGlueConsoleSageMakerNotebookFullAccess — 기존 정책 업데이트.	AWS Glue가 스크립트 및 임시 파일을 저장하는 데 사용하는 Amazon S3 버킷에 대한 읽기/쓰기 권한을 부여하는 작업에 대한 중복 리소스 ARN(arn:aws:s3:::aws-glue-*/*)을 제거했습니다. "StringEquals" 를 "ForAnyValue:StringLike" 로 변경하여 구문 문제를 수정하고, 순서가 어긋난 각 위치에서 "Effect": "Allow" 줄을 "Action": 줄 앞으로 이동했습니다.	2021년 7월 15일
AWSGlueConsoleFullAccess — 기존 정책 업데이트.	arn:aws:s3:::aws-glue-*/* 가 스크립트 및 임시 파일을 저장하는 데 사용하는 Amazon S3 버킷에 대한 읽기/쓰기 권한을 부여하는 작업에 대한 중복 리소스 ARN(AWS Glue)을 제거했습니다.	2021년 7월 15일
AWS Glue에서 변경 사항 추적 시작	AWS Glue AWS 관리형 정책의 변경 사항 추적을 시작했습니다.	2021년 6월 10일

AWS Glue 리소스 ARN 지정

AWS Glue에서 AWS Identity and Access Management(IAM) 정책을 사용하여 리소스에 대한 액세스를 제어할 수 있습니다. 정책에서 Amazon 리소스 이름(ARN)을 사용하여 정책이 적용되는 리소스를 식별합니다. AWS Glue의 일부 리소스는 ARN을 지원하지 않습니다.

주제

- [Data Catalog ARN](#)
- [AWS Glue에서 카탈로그 이외 객체의 ARN](#)
- [AWS Glue 카탈로그 이외 단수 API 작업에 대한 액세스 제어](#)
- [여러 항목을 검색하는 AWS Glue 카탈로그 이외 API 작업에 대한 액세스 제어](#)
- [AWS Glue 카탈로그 이외 BatchGet API 작업에 대한 액세스 제어](#)

Data Catalog ARN

Data Catalog 리소스에는 catalog가 루트인 계층 구조가 있습니다.

```
arn:aws:glue:region:account-id:catalog
```

각 AWS 계정에는 12자리 계정 ID가 카탈로그 ID인 Data Catalog가 AWS 리전에 하나씩 있습니다. 다음 표와 같이 리소스에는 고유한 ARN이 연결되어 있습니다.

리소스 유형	ARN 형식
카탈로그	arn:aws:glue: <i>region</i> : <i>account-id</i> :catalog 예: arn:aws:glue:us-east-1:123456789012:catalog
데이터베이스	arn:aws:glue: <i>region</i> : <i>account-id</i> :database/ <i>database name</i> 예: arn:aws:glue:us-east-1:123456789012:database/db1
표	arn:aws:glue: <i>region</i> : <i>account-id</i> :table/ <i>database name</i> / <i>table name</i> 예: arn:aws:glue:us-east-1:123456789012:table/db1/tb11
사용자 정의 함수	arn:aws:glue: <i>region</i> : <i>account-id</i> :userDefinedFunction/ <i>database name</i> / <i>user-defined function name</i> 예: arn:aws:glue:us-east-1:123456789012:userDefinedFunction/db1/func1

리소스 유형	ARN 형식
연결	arn:aws:glue: <i>region:account-id</i> :connection/ <i>connection name</i> 예: arn:aws:glue:us-east-1:123456789012:connection/connection1
대화형 세션	arn:aws:glue: <i>region:account-id</i> :session/ <i>interactive session id</i> 예: arn:aws:glue:us-east-1:123456789012:session/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111

세분화된 액세스 제어를 사용하려면 IAM 정책 및 리소스 정책에서 이러한 ARN을 사용하여 특정 리소스에 대한 액세스 권한을 부여 및 거부할 수 있습니다. 와일드카드가 정책에서 허용됩니다. 예를 들어, 다음 ARN은 데이터베이스 default의 모든 테이블과 일치합니다.

```
arn:aws:glue:us-east-1:123456789012:table/default/*
```

Important

Data Catalog 리소스에 대해 수행된 모든 작업에는 리소스와 해당 리소스의 모든 상위 항목에 대한 권한이 필요합니다. 예를 들어, 테이블에 대한 파티션을 생성하려면 해당 테이블, 데이터베이스 및 이 테이블이 있는 카탈로그에 대한 권한이 필요합니다. 다음 예에서는 Data Catalog에서 데이터베이스 PrivateDatabase의 테이블 PrivateTable에 대한 파티션을 생성하는데 필요한 권한을 보여줍니다.

```
{
  "Sid": "GrantCreatePartitions",
  "Effect": "Allow",
  "Action": [
    "glue:BatchCreatePartitions"
  ],
  "Resource": [
    "arn:aws:glue:us-east-1:123456789012:table/PrivateDatabase/PrivateTable",
    "arn:aws:glue:us-east-1:123456789012:database/PrivateDatabase",
    "arn:aws:glue:us-east-1:123456789012:catalog"
  ]
}
```

}

리소스와 해당 리소스의 모든 상위 항목에 대한 권한 이외에도 모든 삭제 작업에는 해당 리소스의 모든 하위 항목에 대한 권한이 필요합니다. 예를 들어, 데이터베이스를 삭제하려면 데이터베이스 및 데이터베이스가 위치하고 있는 카탈로그 외에도 데이터베이스에 있는 모든 테이블과 사용자 정의 함수에 대한 권한이 필요합니다. 다음 예에서는 Data Catalog에서 PrivateDatabase 데이터베이스를 삭제하는 데 필요한 권한을 보여줍니다.

```
{
  "Sid": "GrantDeleteDatabase",
  "Effect": "Allow",
  "Action": [
    "glue:DeleteDatabase"
  ],
  "Resource": [
    "arn:aws:glue:us-east-1:123456789012:table/PrivateDatabase/*",
    "arn:aws:glue:us-east-1:123456789012:userDefinedFunction/PrivateDatabase/*",
    "arn:aws:glue:us-east-1:123456789012:database/PrivateDatabase",
    "arn:aws:glue:us-east-1:123456789012:catalog"
  ]
}
```

즉, Data Catalog 리소스에 대한 작업은 다음 권한 규칙을 따릅니다.

- 카탈로그에 대한 작업에는 카탈로그에 대한 권한만 필요합니다.
- 데이터베이스에 대한 작업에는 데이터베이스와 카탈로그에 대한 권한이 필요합니다.
- 데이터베이스에 대한 삭제 작업에는 데이터베이스 및 카탈로그와 해당 데이터베이스 내 모든 테이블과 사용자 정의 함수에 대한 권한이 필요합니다.
- 테이블, 파티션 또는 테이블 버전에 대한 작업에는 테이블, 데이터베이스 및 카탈로그에 대한 권한이 필요합니다.
- 사용자 정의 함수에 대한 작업에는 사용자 정의 함수, 데이터베이스 및 카탈로그에 대한 권한이 필요합니다.
- 연결에 대한 작업에는 연결과 카탈로그에 대한 권한이 필요합니다.

AWS Glue에서 카탈로그 이외 객체의 ARN

일부 AWS Glue 리소스에서는 ARN을 사용하여 액세스를 제어할 수 있도록 리소스 수준 권한을 허용합니다. IAM 정책에서 이러한 ARN을 사용하여 세분화된 액세스 제어를 활성화할 수 있습니다. 다음 표에는 리소스 ARN을 포함할 수 있는 리소스가 나와 있습니다.

리소스 유형	ARN 형식
크롤러	arn:aws:glue: <i>region:account-id</i> :crawler/ <i>crawler-name</i> 예: arn:aws:glue:us-east-1:123456789012:crawler/mycrawler
작업	arn:aws:glue: <i>region:account-id</i> :job/ <i>job-name</i> 예: arn:aws:glue:us-east-1:123456789012:job/testjob
트리거	arn:aws:glue: <i>region:account-id</i> :trigger/ <i>trigger-name</i> 예: arn:aws:glue:us-east-1:123456789012:trigger/sampletrigger
개발 엔드포인트	arn:aws:glue: <i>region:account-id</i> :devEndpoint/ <i>development-endpoint-name</i> 예: arn:aws:glue:us-east-1:123456789012:devEndpoint/temporarydevendpoint
기계 학습 변환	arn:aws:glue: <i>region:account-id</i> :mlTransform/ <i>transform-id</i> 예: arn:aws:glue:us-east-1:123456789012:mlTransform/tfm-1234567890

AWS Glue 카탈로그 이외 단수 API 작업에 대한 액세스 제어

AWS Glue 카탈로그 이외 단수 API 작업은 단일 항목(개발 엔드포인트)에 대해 수행됩니다. GetDevEndpoint, CreateUpdateDevEndpoint 및 UpdateDevEndpoint를 예로 들 수 있습니다. 이러한 작업의 경우 정책은 "action" 블록에 API 이름을, "resource" 블록에 리소스 ARN을 배치해야 합니다.

사용자가 GetDevEndpoint 작업을 호출하도록 허용하려 한다고 가정해 보겠습니다. 다음 정책은 myDevEndpoint-1이라는 엔드포인트에 필요한 최소 권한을 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MinimumPermissions",
      "Effect": "Allow",
      "Action": "glue:GetDevEndpoint",
      "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/
myDevEndpoint-1"
    }
  ]
}
```

다음 정책은 와일드카드(*)를 사용하여 myDevEndpoint-와 일치하는 리소스에 대한 UpdateDevEndpoint 액세스를 허용합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PermissionWithWildcard",
      "Effect": "Allow",
      "Action": "glue:UpdateDevEndpoint",
      "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/myDevEndpoint-
*"
    }
  ]
}
```

다음 예에서처럼 정책 2개를 결합할 수 있습니다. 이름이 A로 시작하는 개발 엔드포인트에 대한 `EntityNotFoundException`을 볼 수 있습니다. 하지만 다른 개발 엔드포인트에 액세스하려고 시도하면 액세스 거부 오류가 반환됩니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CombinedPermissions",
      "Effect": "Allow",
      "Action": [
        "glue:UpdateDevEndpoint",
        "glue:GetDevEndpoint"
      ],
      "Resource": "arn:aws:glue:us-east-1:123456789012:devEndpoint/A*"
    }
  ]
}
```

여러 항목을 검색하는 AWS Glue 카탈로그 이외 API 작업에 대한 액세스 제어

일부 AWS Glue API 작업은 여러 항목(예: 여러 개발 엔드포인트)을 검색합니다(예: `GetDevEndpoints`). 이 작업의 경우 와일드카드(*) 리소스만 지정할 수 있으며 특정 ARN을 지정할 수 없습니다.

예를 들어 정책에 `GetDevEndpoints`를 포함하려면 리소스의 범위를 와일드카드(*)로 지정해야 합니다. 이 예에서는 단수 작업(`GetDevEndpoint`, `CreateDevEndpoint` 및 `DeleteDevEndpoint`)의 범위 또한 모든(*) 리소스로 지정합니다.

```
{
  "Sid": "PluralAPIIncluded",
  "Effect": "Allow",
  "Action": [
    "glue:GetDevEndpoints",
    "glue:GetDevEndpoint",
    "glue:CreateDevEndpoint",
    "glue:UpdateDevEndpoint"
  ],
  "Resource": [
    "*"
  ]
}
```

```
}

```

AWS Glue 카탈로그 이외 BatchGet API 작업에 대한 액세스 제어

일부 AWS Glue API 작업은 여러 항목(예: 여러 개발 엔드포인트)을 검색합니다(예: BatchGetDevEndpoints). 이 작업에서 액세스할 수 있는 리소스의 범위를 제한하도록 ARN을 지정할 수 있습니다.

예를 들어 특정 개발 엔드포인트에 대한 액세스를 허용하려면 정책에 리소스 ARN과 함께 BatchGetDevEndpoints를 포함시킵니다.

```
{
  "Sid": "BatchGetAPIIncluded",
  "Effect": "Allow",
  "Action": [
    "glue:BatchGetDevEndpoints"
  ],
  "Resource": [
    "arn:aws:glue:us-east-1:123456789012:devEndpoint/de1"
  ]
}
```

이 정책을 통해 de1이라는 개발 엔드포인트에 성공적으로 액세스할 수 있습니다. 그러나 de2라는 개발 엔드포인트에 액세스를 시도하면 오류가 반환됩니다.

An error occurred (AccessDeniedException) when calling the BatchGetDevEndpoints operation: No access to any requested resource.

Important

List 및 BatchGet API 작업 사용과 같이 IAM 정책을 설정하기 위한 대체 접근 방식은 [AWS Glue 자격 증명 기반 정책 예제](#) 단원을 참조하십시오.

교차 계정 액세스 권한 부여

계정 간에 데이터 카탈로그 리소스에 대한 액세스 권한을 부여하면 추출, 변환, 로드(ETL) 작업에서 다른 계정의 데이터를 쿼리하고 결합할 수 있습니다.

주제

- [AWS Glue에서 교차 계정 액세스 권한을 부여하는 방법](#)
- [데이터 카탈로그 리소스 정책 추가 또는 업데이트](#)
- [교차 계정 API 호출 수행](#)
- [교차 계정 ETL 호출 수행](#)
- [교차 계정 CloudTrail 로깅](#)
- [교차 계정 리소스 소유권 및 결제](#)
- [교차 계정 액세스 제한 사항](#)

AWS Glue에서 교차 계정 액세스 권한을 부여하는 방법

AWS Glue 메서드를 사용하거나 AWS Lake Formation 교차 계정 권한 부여를 사용하여 외부 AWS 계정에 데이터에 대한 액세스 권한을 부여할 수 있습니다. AWS Glue 메서드는 AWS Identity and Access Management(IAM) 정책을 사용하여 세분화된 액세스 제어를 수행합니다. Lake Formation은 관계형 데이터베이스 시스템의 GRANT/REVOKE 명령과 유사한 더 간단한 GRANT/REVOKE 권한 모델을 사용합니다.

이 섹션에서는 AWS Glue 메서드 사용에 대해 설명합니다. Lake Formation 교차 계정 권한 부여 사용에 대한 자세한 내용은 AWS Lake Formation Developer Guide의 [Granting Lake Formation Permissions](#)를 참조하세요.

리소스에 대한 교차 계정 액세스 권한을 부여하기 위한 두 가지 AWS Glue 방법이 있습니다.

- 데이터 카탈로그 리소스 정책 사용
- IAM 역할 사용

리소스 정책을 사용하여 교차 계정 액세스 권한 부여

다음은 데이터 카탈로그 리소스 정책을 사용하여 교차 계정 액세스 권한을 부여하는 일반적인 단계입니다.

1. 계정 A의 관리자(또는 기타 권한이 있는 자격 증명)가 계정 A의 데이터 카탈로그에 리소스 정책을 연결합니다. 이 정책은 계정 B에게 계정 A의 카탈로그에 있는 리소스에 대한 작업을 수행할 수 있는 특정 교차 계정 권한을 부여합니다.
2. 계정 B의 관리자는 계정 A에게서 받은 권한을 위임하는 계정 B의 IAM 자격 증명에 IAM 정책을 연결합니다.

계정 B의 자격 증명이 이제 계정 A의 지정된 리소스에 대한 액세스 권한을 갖습니다.

자격 증명은 리소스에 액세스하기 위해 리소스 소유자(계정 A)와 상위 계정(계정 B) 모두의 권한이 필요합니다.

IAM 역할을 사용하여 교차 계정 액세스 권한 부여

다음은 IAM 역할을 사용하여 교차 계정 액세스 권한을 부여하는 일반적인 단계입니다.

1. 리소스를 소유한 계정(계정 A)의 관리자(또는 권한이 있는 다른 자격 증명)가 IAM 역할을 생성합니다.
2. 계정 A의 관리자는 문제의 리소스에 대한 교차 계정 액세스 권한을 부여하는 역할에 정책을 연결합니다.
3. 계정 A의 관리자는 다른 계정(계정 B)의 IAM 자격 증명을 해당 역할을 맡을 수 있는 보안 주체로 식별하는 역할에 신뢰 정책을 연결합니다.

AWS 서비스에 역할을 맡을 수 있는 권한을 부여하고 싶다면, 신뢰 정책의 보안 주체는 AWS 서비스 보안 주체가 됩니다.

4. 이제, 계정 B의 관리자가 계정 B에 있는 하나 이상의 IAM 자격 증명에 권한을 위임해 해당 자격 증명에 역할을 맡을 수 있습니다. 그러면 계정 B의 해당 자격 증명에 계정 A의 리소스에 대한 액세스 권한이 부여됩니다.

IAM을 사용하여 권한을 위임하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [액세스 관리](#) 단원을 참조하십시오. 사용자, 그룹, 역할 및 권한에 대한 자세한 내용은 IAM User Guide의 [Identities \(users, groups, and roles\)](#)를 참조하세요.

두 가지 방법을 비교하려면 IAM 사용 설명서의 [IAM 역할과 리소스 기반 정책의 차이점](#)을 참조하세요. AWS Glue에서는 두 옵션을 모두 지원하지만, 리소스 정책은 데이터 카탈로그 리소스에 대한 액세스 권한만 부여할 수 있다는 제한 사항이 있습니다.

예를 들어, 계정 B의 Dev 역할에게 계정 A의 데이터베이스 db1에 대한 액세스 권한을 부여하려면 다음 리소스 정책을 계정 A의 카탈로그에 연결합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

    "Effect": "Allow",
    "Action": [
      "glue:GetDatabase"
    ],
    "Principal": {"AWS": [
      "arn:aws:iam::account-B-id:role/Dev"
    ]},
    "Resource": [
      "arn:aws:glue:us-east-1:account-A-id:catalog",
      "arn:aws:glue:us-east-1:account-A-id:database/db1"
    ]
  }
]
}

```

또한 계정 B는 계정 A의 db1에 액세스하기 전에 다음 IAM 정책을 Dev 역할에 연결해야 합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase"
      ],
      "Resource": [
        "arn:aws:glue:us-east-1:account-A-id:catalog",
        "arn:aws:glue:us-east-1:account-A-id:database/db1"
      ]
    }
  ]
}

```

데이터 카탈로그 리소스 정책 추가 또는 업데이트

콘솔, API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 AWS Glue 데이터 카탈로그 리소스 정책을 추가하거나 업데이트할 수 있습니다.

Important

AWS Lake Formation이 있는 계정에서 이미 교차 계정 권한을 부여한 경우 데이터 카탈로그 리소스 정책을 추가하거나 업데이트하려면 추가 단계가 필요합니다. 자세한 내용은 AWS Lake

Formation 개발자 가이드의 [AWS Glue 및 Lake Formation을 모두 사용하여 계정 권한 관리](#)를 참조하세요.

Lake Formation 크로스 계정 권한 부여가 있는지 확인하려면 `glue:GetResourcePolicies` API 작업 또는 AWS CLI를 사용합니다. `glue:GetResourcePolicies`이(가) 이미 존재하는 데이터 카탈로그 정책 이외의 정책을 반환하는 경우 Lake Formation 권한 부여가 존재합니다. 자세한 내용은 AWS Lake Formation 개발자 가이드의 [GetResourcePolicies API 작업을 사용하여 크로스 계정 권한 부여 보기](#)를 참조하세요.

데이터 카탈로그 리소스 정책을 추가하거나 업데이트하려면(콘솔)

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.

`glue:PutResourcePolicy` 권한이 있는 AWS Identity and Access Management(IAM) 관리 사용자로 로그인합니다.

2. 탐색 창에서 설정을 선택합니다.
3. [데이터 카탈로그 설정(Data catalog settings)] 페이지의 [권한(Permissions)]에서 리소스 정책을 텍스트 영역에 붙여넣습니다. 그런 다음 저장을 선택합니다.

정책의 권한이 Lake Formation을 사용하여 부여된 권한에 추가된다는 알림이 콘솔에 표시되면 [계속(Proceed)]을 선택합니다.

데이터 카탈로그 리소스 정책을 추가하거나 업데이트하려면(AWS CLI)

- `aws glue put-resource-policy` 명령을 제출합니다. Lake Formation 권한 부여가 이미 있는 경우 `--enable-hybrid` 옵션을 'TRUE' 값과 함께 포함해야 합니다.

이 명령의 사용 예는 [AWS Glue용 리소스 기반 정책 예제](#) 섹션을 참조하세요.

교차 계정 API 호출 수행

모든 AWS Glue Data Catalog 작업에는 `CatalogId` 필드가 있습니다. 교차 계정 액세스를 활성화하기 위한 필수 권한이 부여된 경우 호출자는 계정을 교차하여 데이터 카탈로그 API 호출을 수행할 수 있습니다. 호출자는 해당 대상 계정의 리소스에 액세스하기 위해 `CatalogId`의 대상 AWS 계정 ID를 전달하여 이렇게 합니다.

`CatalogId` 값을 입력하지 않으면 AWS Glue에서는 기본적으로 호출자의 계정 ID를 사용하고 교차 계정 호출이 수행되지 않습니다.

교차 계정 ETL 호출 수행

일부 AWS Glue PySpark 및 Scala API에는 카탈로그 ID 필드가 있습니다. 교차 계정 액세스가 가능하도록 모든 필수 권한이 부여된 경우 ETL 작업은 대상 계정의 데이터 카탈로그 리소스에 액세스할 수 있도록 카탈로그 ID 필드에 대상 AWS 계정 ID를 전달하여 계정 간에 API 작업에 대해 PySpark 및 Scala를 호출할 수 있습니다.

카탈로그 ID 값을 입력하지 않으면 AWS Glue에서는 기본적으로 호출자의 계정 ID를 사용하고 교차 계정 호출이 수행되지 않습니다.

catalog_id를 지원하는 PySpark API는 [GlueContext 수업](#) 단원을 참조하십시오. catalogId를 지원하는 Scala API는 [AWS Glue스칼라 API GlueContext](#) 단원을 참조하십시오.

다음 예는 ETL 작업을 실행하기 위해 피부여자에게 필요한 권한을 보여줍니다. 이 예에서 *grantee-account-id*는 작업을 실행하는 클라이언트의 catalog-id이고, *grantor-account-id*는 리소스의 소유자입니다. 이 예에서는 부여자의 계정에 있는 모든 카탈로그 리소스에 대한 권한을 부여합니다. 부여되는 리소스의 범위를 제한하기 위해 카탈로그, 데이터베이스, 테이블 및 연결에 해당하는 특정 ARN을 제공할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetConnection",
        "glue:GetDatabase",
        "glue:GetTable",
        "glue:GetPartition"
      ],
      "Principal": {"AWS": ["arn:aws:iam::grantee-account-id:root"]},
      "Resource": [
        "arn:aws:glue:us-east-1:grantor-account-id:"
      ]
    }
  ]
}
```

Note

부여자의 계정에 있는 테이블이 부여자의 계정에 있는 Amazon S3 위치를 가리키는 경우 피부여자의 계정에서 ETL 작업을 실행하는 데 사용하는 IAM 역할은 부여자의 계정에서 객체를 나열하고 가져올 수 있는 권한이 있어야 합니다.

계정 A의 클라이언트가 ETL 작업을 생성하고 실행할 수 있는 권한을 이미 가지고 있는 경우 교차 계정 액세스를 위해 ETL 작업을 설정하는 기본 단계는 다음과 같습니다.

1. 교차 계정 데이터 액세스를 허용합니다(Amazon S3 교차 계정 액세스가 이미 설정된 경우에는 이 단계를 건너뛴).
 - a. 계정 A에서 교차 계정 액세스를 허용하도록 계정 B에서 Amazon S3 버킷 정책을 업데이트합니다.
 - b. 계정 B에서 버킷에 대한 액세스를 허용하도록 계정 A에서 IAM 정책을 업데이트합니다.
2. 교차 계정 데이터 카탈로그 액세스를 허용합니다.
 - a. 계정 A에서 액세스를 허용하도록 계정 B에서 데이터 카탈로그에 연결된 리소스 정책을 생성하거나 업데이트합니다.
 - b. 계정 B에서 데이터 카탈로그에 대한 액세스를 허용하도록 계정 A에서 IAM 정책을 업데이트합니다.

교차 계정 CloudTrail 로깅

AWS Glue 추출, 변환, 로드 작업이 AWS Lake Formation 교차 계정 권한 부여를 통해 공유되는 데이터 카탈로그 테이블의 기본 데이터에 액세스하면 추가 AWS CloudTrail 로깅 동작이 있습니다.

이 설명을 위해 테이블을 공유한 AWS 계정은 소유자 계정이고 테이블을 공유한 계정은 수신자 계정입니다. 수신자 계정의 ETL 작업이 소유자 계정의 테이블에 있는 데이터에 액세스하면 수신자 계정의 로그에 추가된 데이터 액세스 CloudTrail 이벤트가 소유자 계정의 CloudTrail 로그에 복사됩니다. 이는 소유자 계정이 다양한 수신자 계정의 데이터 액세스를 추적할 수 있도록 하기 위한 것입니다. 기본적으로 CloudTrail 이벤트에는 사람이 읽을 수 있는 보안 주체 식별자(기본 ARN)가 포함되어 있지 않습니다. 수신자 계정의 관리자는 로그에 보안 주체 ARN을 포함하도록 선택할 수 있습니다.

자세한 내용은 AWS Lake Formation 개발자 가이드의 [크로스 계정 CloudTrail 로깅](#)을 참조하세요.

참고

- [the section called “로깅 및 모니터링”](#)

교차 계정 리소스 소유권 및 결제

AWS 계정 하나(계정 A)의 사용자가 다른 계정(계정 B)에서 새 리소스(예: 데이터베이스)를 생성하는 경우 리소스의 소유자는 해당 리소스가 생성된 계정인 계정 B입니다. 계정 B의 관리자는 읽기 및 쓰기와 세 번째 계정에게 액세스 권한 부여 등을 비롯해 새 리소스에 액세스하기 위한 전체 권한을 자동으로 가져옵니다. 계정 A의 사용자는 계정 B가 부여한 적절한 권한이 있는 경우에만 방금 생성한 리소스에 액세스할 수 있습니다.

스토리지 비용과 새 리소스와 직접 연관된 기타 비용이 리소스 소유자인 계정 B에게 청구됩니다. 리소스를 생성한 사용자의 요청 비용은 요청자의 계정인 계정 A로 청구됩니다.

AWS Glue 결제 및 요금에 대한 자세한 내용은 [AWS 요금 책정 방식](#)을 참조하세요.

교차 계정 액세스 제한 사항

AWS Glue 교차 계정 액세스에는 다음과 같은 제한 사항이 있습니다.

- 이전의 AWS Glue 지원 이전에 Amazon Athena 또는 Amazon Redshift Spectrum을 사용하여 데이터베이스와 테이블을 생성했으며 리소스 소유자 계정이 Amazon Athena 데이터 카탈로그를 AWS Glue로 마이그레이션하지 않은 경우에는 AWS Glue에 대한 교차 계정 액세스가 허용되지 않습니다. [GetCatalogImportStatus\(get_catalog_import_status\)](#)를 사용하여 현재 마이그레이션 상태를 찾을 수 있습니다. Athena 카탈로그를 AWS Glue로 마이그레이션하는 방법에 대한 자세한 내용은 Amazon Athena 사용 설명서의 [AWS Glue Data Catalog로 단계별 업그레이드](#)를 참조하세요.
- 교차 계정 액세스는 데이터베이스, 테이블, 사용자 정의 함수 및 연결을 포함한 데이터 카탈로그 리소스에 대해서만 지원됩니다.
- Athena에서 데이터 카탈로그에 대한 교차 계정 액세스를 사용하려면 카탈로그를 Athena DataCatalog 리소스로 등록해야 합니다. 지침은 Amazon Athena 사용 설명서의 [다른 계정에서 AWS Glue Data Catalog 등록](#)을 참조하세요.

AWS Glue ID 및 액세스 문제 해결

다음 정보를 사용하면 AWS Glue 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 해결하는 데 도움이 됩니다.

주제

- [AWS Glue에서 작업을 수행할 권한이 없습니다.](#)
- [저는 IAM을 수행할 권한이 없습니다. PassRole](#)
- [외부 사용자가 내 AWS Glue 리소스에 액세스할 AWS 계정 수 있도록 허용하고 싶습니다.](#)

AWS Glue에서 작업을 수행할 권한이 없습니다.

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음 예제 오류는 mateojacksonIAM 사용자가 콘솔을 사용하여 가상 *my-example-widget* 리소스에 대한 세부 정보를 보려고 하지만 가상 *glue:GetWidget* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
glue:GetWidget on resource: my-example-widget
```

이 경우 *glue:GetWidget* 작업을 사용하여 *my-example-widget* 리소스에 액세스할 수 있도록 mateojackson 사용자 정책을 업데이트해야 합니다.

도움이 필요하면 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

저는 IAM을 수행할 권한이 없습니다. PassRole

작업을 수행할 권한이 없다는 오류가 발생하는 경우 AWS Glue에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다. *iam:PassRole*

일부 AWS 서비스 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 AWS Glue에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

이 경우, Mary가 *iam:PassRole* 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요하면 관리자에게 문의하세요. AWS 관리자는 로그인 자격 증명을 제공한 사람입니다.

외부 사용자가 내 AWS Glue 리소스에 액세스할 AWS 계정 수 있도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- AWS Glue가 이러한 기능을 지원하는지 알아보려면 [AWS Glue가 IAM과 함께 작동하는 방식](#).
- 소유한 리소스에 대한 액세스 권한을 AWS 계정 부여하는 방법을 알아보려면 IAM 사용 [설명서에서 소유하고 AWS 계정 있는 다른 IAM 사용자에게 액세스 권한 제공](#)을 참조하십시오.
- [제3자에게 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 AWS 계정 소유에 대한 액세스 제공](#)을 참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [외부에서 인증된 사용자에게 액세스 권한 제공\(자격 증명 페더레이션\)](#)을 참조하십시오.
- 교차 계정 액세스에 대한 역할 사용과 리소스 기반 정책의 차이점을 알아보려면 [IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스](#)를 참조하십시오.

AWS Glue의 로깅 및 모니터링

ETL(추출, 변환 및 로드) 작업 실행을 자동화할 수 있습니다. AWS Glue는 모니터링할 수 있는 크롤러 및 작업에 대한 지표를 제공합니다. 필요한 메타데이터로 AWS Glue Data Catalog을 설치한 후, AWS Glue는 환경 상태에 대한 지표를 제공합니다. Cron을 기반으로 시간 기반 일정으로 작업 및 크롤러의 호출을 자동화할 수 있습니다. 이벤트 기반 트리거를 시작할 경우 작업을 시작할 수 있습니다.

AWS Glue는 AWS Glue에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. 추적을 생성하는 경우 CloudTrail 이벤트를 Amazon Simple Storage Service(Amazon S3) 버킷, Amazon CloudWatch Logs 및 Amazon CloudWatch Events로 지속적으로 전송할 수 있습니다. 모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에게 대한 정보가 들어 있습니다.

Amazon CloudWatch Events를 사용하여 AWS 서비스를 자동화하고 애플리케이션 가용성 문제나 리소스 변경 같은 시스템 이벤트에 자동으로 응답합니다. AWS 서비스의 이벤트는 거의 실시간으로

CloudWatch Events로 전송됩니다. 어떤 이벤트에 관심이 있으며 이벤트가 규칙과 일치할 때 어떤 자동화된 작업을 수행할지를 표시하는 간단한 규칙을 작성할 수 있습니다.

i 다음 사항도 참조하십시오.

- [CloudWatch Events로 AWS Glue 자동화](#)
- [교차 계정 CloudTrail 로깅](#)

클라우드에서 중요한 보안 요소 중 하나는 로깅입니다. 클라우드 인프라를 디버깅하고 보호하는 데 필요한 정보를 캡처하면서 보안 암호 및 기밀 자료를 캡처하지 않는 방식으로 로깅을 구성해야 합니다. 어떤 정보와 데이터가 로깅되는지 숙지해야 합니다.

에 대한 규정 준수 검증 AWS Glue

특정 규정 준수 프로그램의 범위 내에 AWS 서비스 있는지 알아보려면 AWS 서비스 규정 준수 [프로그램의 AWS 서비스 범위별, 규정](#) 참조하여 관심 있는 규정 준수 프로그램을 선택하십시오. 일반 정보는 [AWS 규정 준수 프로그램 AWS 보증 프로그램 규정 AWS](#) 참조하십시오.

를 사용하여 AWS Artifact 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 <https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html> 참조하십시오 AWS Artifact.

사용 시 규정 준수 AWS 서비스 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- [보안 및 규정 준수 킷 스타트 가이드](#) - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 AWS 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- [Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계 — 이 백서에서는 기업이 HIPAA 적격 애플리케이션을 만드는 AWS 데 사용할 수 있는 방법을 설명합니다.](#)

i Note

모든 AWS 서비스 사람이 HIPAA 자격을 갖춘 것은 아닙니다. 자세한 내용은 [HIPAA 적격 서비스 참조](#)를 참조하십시오.

- [AWS 규정 준수 리소스 AWS](#) — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.

- [AWS 고객 규정 준수 가이드](#) — 규정 준수의 관점에서 공동 책임 모델을 이해하십시오. 이 가이드에서는 보안을 유지하기 위한 모범 사례를 AWS 서비스 요약하고 여러 프레임워크 (미국 표준 기술 연구소 (NIST), 결제 카드 산업 보안 표준 위원회 (PCI), 국제 표준화기구 (ISO) 등) 에서 보안 제어에 대한 지침을 매핑합니다.
- AWS Config 개발자 안내서의 [규칙을 사용하여 리소스 평가](#) — 이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- [AWS Security Hub](#) — 이를 AWS 서비스 통해 내부 AWS 보안 상태를 포괄적으로 파악할 수 있습니다. Security Hub는 보안 제어를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 [Security Hub 제어 참조](#)를 참조하십시오.
- [Amazon GuardDuty](#) — 환경에 의심스럽고 악의적인 활동이 있는지 AWS 계정모니터링하여 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 해결하는 데 도움이 될 수 있습니다.
- [AWS Audit Manager](#) — 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 단순화할 수 있습니다.

의 탄력성 AWS Glue

AWS 글로벌 인프라는 AWS 지역 및 가용 영역을 중심으로 구축됩니다. AWS 지역은 물리적으로 분리되고 격리된 여러 가용 영역을 제공하며, 이러한 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워크로 연결됩니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS [지역 및 가용 영역에 대한 자세한 내용은 글로벌 인프라를 참조하십시오AWS](#).

AWS Glue 작업 복원력에 대한 자세한 내용은 [오류: 의 VPC 간 장애 조치 동작을](#) 참조하십시오. AWS Glue

AWS Glue에서 인프라 보안

관리형 서비스인 AWS Glue은 [Amazon Web Services: 보안 프로세스 개요](#) 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다.

AWS에서 게시한 API 직접 호출을 사용하여 네트워크를 통해 AWS Glue에 액세스합니다. 클라이언트가 전송 계층 보안(TLS) 1.0 이상을 지원해야 합니다. TLS 1.2 이상을 권장합니다. 클라이언트

는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 PFS(전달 완전 보안, Perfect Forward Secrecy)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 비밀 액세스 키를 사용하여 서명해야 합니다. 또는 [AWS Security Token Service\(AWS STS\)](#)를 사용하여 임시 보안 인증 정보를 생성하여 요청에 서명할 수 있습니다.

주제

- [AWS Glue 및 인터페이스 VPC 엔드포인트\(AWS PrivateLink\)](#)
- [공유 Amazon VPC](#)

AWS Glue 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트를 생성하여 VPC와 AWS Glue 간에 프라이빗 연결을 설정할 수 있습니다. 인터페이스 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 비공개로 AWS Glue API에 액세스할 수 있도록 지원하는 [AWS PrivateLink](#) 기술로 구동됩니다. VPC의 인스턴스는 AWS Glue API와 통신하는 데 퍼블릭 IP 주소를 필요로 하지 않습니다. VPC와 AWS Glue 간의 트래픽은 당사의 네트워크를 벗어나지 않습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 [탄력적 네트워크 인터페이스](#)로 표현됩니다.

자세한 내용은 Amazon VPC 사용 설명서에서 [인터페이스 VPC 종단점\(AWS PrivateLink\)](#)을 참조하십시오.

AWS Glue VPC 엔드포인트 고려 사항

AWS Glue에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 VPC 사용 설명서에서 [인터페이스 엔드포인트 속성 및 제한 사항](#)을 검토해야 합니다.

AWS Glue은 VPC에서 모든 API 작업에 대한 직접 호출 수행을 지원합니다.

AWS Glue에 대한 인터페이스 VPC 엔드포인트 생성

VPC 콘솔이나 AWS Command Line Interface(AWS CLI)을 사용하여 AWS Glue 서비스에 대한 VPC 엔드포인트를 생성할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [인터페이스 엔드포인트 생성](#)을 참조하세요.

다음 서비스 이름을 사용하여 AWS Glue용 VPC 종단점을 생성합니다.

- `com.amazonaws.region.glue`

엔드포인트에 프라이빗 DNS를 사용하도록 설정하는 경우, 리전에 대한 기본 DNS 이름(예: `glue.us-east-1.amazonaws.com`)을 사용하여 AWS Glue에 API 요청을 할 수 있습니다.

자세한 내용은 VPC 사용 설명서의 [인터페이스 엔드포인트를 통해 서비스 액세스](#)를 참조하세요.

AWS Glue에 대한 VPC 엔드포인트 정책 생성

AWS Glue에 대한 액세스를 제어하는 VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 정보는 Amazon VPC 사용 설명서의 [VPC 엔드포인트를 통해 서비스에 대한 액세스 제어](#)를 참조하세요.

예제: AWS Glue에서 작업 생성 및 업데이트를 허용하도록 하는 VPC 엔드포인트 정책

다음은 AWS Glue에 대한 엔드포인트 정책의 예입니다. 이 정책은 엔드포인트에 연결될 때 모든 리소스의 모든 보안 주체에 대한 액세스 권한을 나열된 AWS Glue 작업에 부여합니다.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "glue:CreateJob",
        "glue:UpdateJob",
        "iam:PassRole"
      ],
      "Resource": "*"
    }
  ]
}
```

예제: 읽기 전용 Data Catalog 액세스를 허용하는 VPC 엔드포인트 정책

다음은 AWS Glue에 대한 엔드포인트 정책의 예입니다. 이 정책은 엔드포인트에 연결될 때 모든 리소스의 모든 보안 주체에 대한 액세스 권한을 나열된 AWS Glue 작업에 부여합니다.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:GetDatabases",
        "glue:GetTable",
        "glue:GetTables",
        "glue:GetTableVersion",
        "glue:GetTableVersions",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:BatchGetPartition",
        "glue:SearchTables"
      ],
      "Resource": "*"
    }
  ]
}
```

공유 Amazon VPC

AWS Glue는 Amazon Virtual Private Cloud에서 공유 Virtual Private Cloud(VPC)를 지원합니다. Amazon VPC 공유를 통해 여러 AWS 계정이 Amazon EC2 인스턴스 및 Amazon Relational Database Service(Amazon RDS) 데이터베이스와 같은 애플리케이션 리소스를 공유 중앙 관리형 Amazon VPC에 생성할 수 있습니다. 이 모델에서 VPC(소유자)를 소유하는 계정은 AWS Organizations의 동일한 조직에 속한 다른 계정(참여자)과 한 개 또는 여러 개의 서브넷을 공유합니다. 서브넷을 공유한 후 참여자는 공유된 서브넷의 해당 애플리케이션 리소스를 보고, 생성하고, 수정하고, 삭제할 수 있습니다.

AWS Glue에서 공유된 서브넷과의 연결을 생성하려면 계정 내에 보안 그룹을 생성하고 해당 그룹을 공유된 서브넷에 연결해야 합니다.

자세한 내용은 다음 주제를 참조하십시오.

- Amazon VPC 사용 설명서의 [공유 VPC 작업](#)
- AWS Organizations User Guide의 [What Is AWS Organizations?](#)

AWS Glue 문제 해결

주제

- [AWS Glue 문제 해결 정보 모으기](#)
- [AWS Glue Spark의 오류 문제 해결](#)
- [로그에서 AWS Glue for Ray 오류 해결](#)
- [AWS Glue 기계 학습 예외 사항](#)
- [AWS Glue 할당량](#)

AWS Glue 문제 해결 정보 모으기

AWS Glue의 오류나 예상 밖의 행동이 나타나고 AWS Support에 연결할 필요가 있다면 먼저 실패한 작업과 관련된 이름, ID 및 로그에 대한 정보를 모아야 합니다. 이 정보가 있으면 AWS Support가 해당 문제의 해결을 지원할 수 있습니다.

계정 ID와 함께 각 실패 유형에 따라 다음 정보를 모으십시오.

크롤러가 실패하면 다음 정보를 모으십시오.

- 크롤러 이름

크롤러 작업의 로그는 `/aws-glue/crawlers` 하에 CloudWatch Logs에 위치합니다.

테스트 연결이 되지 않으면 다음 정보를 모으십시오.

- 연결 이름
- 연결 ID
- `jdbc:protocol://host:port/database-name` 형식의 JDBC 연결 문자열.

테스트 연결의 로그는 `/aws-glue/testconnection` 하에 CloudWatch Logs에 위치합니다.

작업이 실패하면 다음 정보를 모으십시오.

- 작업 이름
- `jr_xxxxx` 형식의 작업 실행 ID.

작업 실행의 로그는 `/aws-glue/jobs` 하에 CloudWatch Logs에 위치합니다.

AWS Glue Spark의 오류 문제 해결

에서 AWS Glue 오류가 발생하는 경우 다음 해결 방법을 사용하여 문제의 원인을 찾아 해결하십시오.

Note

AWS Glue GitHub 리포지토리에는 [AWS Glue FAQ](#)에 추가 문제 해결 지침이 포함되어 있습니다.

주제

- [오류: 사용 가능하지 않는 리소스](#)
- [오류: VPC에서 서브넷 ID용 S3 엔드포인트 또는 NAT 게이트웨이를 찾을 수 없습니다.](#)
- [오류: 보안 그룹의 인바운드 규칙이 필요합니다.](#)
- [오류: 보안 그룹의 아웃바운드 규칙이 필요합니다.](#)
- [오류: 전달된 역할에 AWS Glue 서비스에 대한 역할 수임 권한을 부여해야 하므로 작업 실행이 실패했습니다.](#)
- [오류: DescribeVpcEndpoints 작업이 승인되지 않았습니다. VPC ID vpc-id를 검증할 수 없습니다.](#)
- [오류: DescribeRouteTables 작업이 승인되지 않았습니다. 서브넷 ID를 확인할 수 없습니다. VPC ID의 서브넷 ID: vpc-id](#)
- [오류: ec2 호출 실패: DescribeSubnets](#)
- [오류: ec2 호출 실패: DescribeSecurityGroups](#)
- [오류: AZ용 서브넷을 찾을 수 없습니다.](#)
- [오류: JDBC 대상으로 작성할 경우의 작업 실행 예외](#)
- [오류: Amazon S3: 객체의 스토리지 클래스에 대해 작업이 유효하지 않습니다.](#)
- [오류: Amazon S3 시간 제한](#)
- [오류: Amazon S3 액세스가 거부되었습니다.](#)
- [오류: Amazon S3 액세스 키 ID가 없습니다.](#)
- [오류: s3a:// URI를 사용해 Amazon S3에 액세스할 때 작업 실행이 실패합니다.](#)
- [오류: Amazon S3 서비스 토큰이 만료되었습니다.](#)
- [오류: 네트워크 인터페이스용 프라이빗 DNS를 찾을 수 없습니다.](#)
- [오류: 개발 엔드포인트 프로비저닝이 실패했습니다.](#)

- [오류: 노트북 서버 생성이 실패했습니다.](#)
- [오류: 로컬 노트북 시작이 실패했습니다.](#)
- [오류: 크롤러 실행이 실패했습니다.](#)
- [오류: 파티션이 업데이트되지 않았습니다.](#)
- [오류: 버전 불일치로 인해 작업 북마크 업데이트 실패](#)
- [오류: 작업 북마크가 사용 설정된 경우 작업이 데이터를 재처리합니다.](#)
- [오류: 의 VPC 간 페일오버 동작 AWS Glue](#)
- [크롤러가 Lake Formation 보안 인증을 사용할 때 발생하는 크롤러 오류 문제 해결](#)

오류: 사용 가능하지 않는 리소스

리소스를 사용할 수 없다는 메시지가 AWS Glue 반환되는 경우 오류 메시지나 로그를 확인하여 문제에 대해 자세히 알아볼 수 있습니다. 다음 작업은 문제 해결을 위한 일반적인 방법을 설명합니다.

- 사용할 연결과 개발 엔드포인트의 경우, 탄력적 네트워크 인터페이스를 벗어나지 않은 클러스터를 확인합니다.

오류: VPC에서 서브넷 ID용 S3 엔드포인트 또는 NAT 게이트웨이를 찾을 수 없습니다.

메시지에서 서브넷 ID 및 VPC ID를 선택하여 문제를 진단하십시오.

- AWS Glue에 필요한 Amazon S3 VPC 엔드포인트 설정이 있는지 확인합니다. 또한, NAT 게이트웨이가 구성 중 일부라면 확인하십시오. 자세한 정보는 [Amazon S3용 Amazon VPC 엔드포인트](#)를 참조하세요.

오류: 보안 그룹의 인바운드 규칙이 필요합니다.

적어도 하나 이상의 보안 그룹은 모든 내보내기 포트를 열어야 합니다. 트래픽을 제한하기 위해서 인바운드 규칙의 소스 보안 그룹은 동일한 보안 그룹으로 제한될 수 있습니다.

- 사용하는 연결의 경우, 자기 참조적 인바운드 규칙을 위한 보안 그룹을 확인하십시오. 자세한 정보는 [데이터 스토어에 대한 네트워크 액세스 설정](#)을 참조하세요.
- 개발 엔드포인트를 사용 중이면 자기 참조적 인바운드 규칙을 위한 보안 그룹을 확인하십시오. 자세한 정보는 [데이터 스토어에 대한 네트워크 액세스 설정](#)을 참조하세요.

오류: 보안 그룹의 아웃바운드 규칙이 필요합니다.

적어도 하나 이상의 보안 그룹은 모든 들어오기 포트를 열어야 합니다. 트래픽을 제한하기 위해서 아웃바운드 규칙의 소스 보안 그룹은 동일한 보안 그룹으로 제한될 수 있습니다.

- 사용하는 연결의 경우, 자기 참조적 아웃바운드 규칙을 위한 보안 그룹을 확인하십시오. 자세한 정보는 [데이터 스토어에 대한 네트워크 액세스 설정](#)을 참조하세요.
- 개발 엔드포인트를 사용 중이면 자기 참조적 아웃바운드 규칙을 위한 보안 그룹을 확인하십시오. 자세한 정보는 [데이터 스토어에 대한 네트워크 액세스 설정](#)을 참조하세요.

오류: 전달된 역할에 AWS Glue 서비스에 대한 역할 수임 권한을 부여해야 하므로 작업 실행이 실패했습니다.

작업을 정의하는 사용자는 AWS Glue에 대한 iam:PassRole 권한이 있어야 합니다.

- 사용자가 AWS Glue 작업을 생성할 때 사용자 역할에 다음을 포함하는 정책이 포함되어 있는지 확인하십시오. AWS Glue. iam:PassRole 자세한 정보는 [3단계: AWS Glue에 액세스하는 사용자 또는 그룹에 정책 연결](#)을 참조하세요.

오류: DescribeVpcEndpoints 작업이 승인되지 않았습니다. VPC ID vpc-id를 검증할 수 없습니다.

- 전달된 정책에서 권한을 확인하십시오. AWS Glue ec2:DescribeVpcEndpoints

오류: DescribeRouteTables 작업이 승인되지 않았습니다. 서브넷 ID를 확인할 수 없습니다. VPC ID의 서브넷 ID: vpc-id

- 전달된 AWS Glue 정책에서 권한을 확인하십시오. ec2:DescribeRouteTables

오류: ec2 호출 실패: DescribeSubnets

- 전달된 정책에서 ec2:DescribeSubnets 권한을 확인하십시오. AWS Glue

오류: ec2 호출 실패: DescribeSecurityGroups

- 전달된 정책에서 `ec2:DescribeSecurityGroups` 권한을 확인하십시오. AWS Glue

오류: AZ용 서브넷을 찾을 수 없습니다.

- 가용 영역을 사용할 수 없을 수도 AWS Glue 있습니다. 메시지에 있는 지정된 영역 중 새로운 서브넷을 다른 가용 영역에서 만들고 사용합니다.

오류: JDBC 대상으로 작성할 경우의 작업 실행 예외

JDBC 대상으로 작성하는 작업을 실행하고 있다면 작업은 다음과 같은 상황에서 오류가 발생합니다.

- 작업이 Microsoft SQL Server 테이블에 쓴다면 테이블은 Boolean 유형으로 정의된 열을 가지고 있고 SQL 서버 데이터베이스에서 미리 정의되어야 합니다. 데이터 대상의 테이블 생성 옵션과 함께 SQL Server 대상을 사용하여 AWS Glue 콘솔에서 작업을 정의할 때는 데이터 유형의 대상 열에 원본 열을 매핑하지 마십시오 Boolean. 작업이 실행되면 오류가 발생할 수 있습니다.

다음을 수행하여 오류를 피합니다.

- [Boolean(부울)]을 사용하여 기존 테이블을 선택합니다.
- ApplyMapping 변환을 편집하고 소스의 [Boolean(부울)] 열을 대상의 숫자 또는 문자열로 매핑합니다.
- ApplyMapping 변환을 편집하여 소스의 [Boolean(부울)] 열을 제거합니다.
- 작업이 Oracle 테이블에 쓴다면 Oracle 객체 이름 길이를 조정해야 합니다. Oracle의 몇 가지 버전에서는 최대 분류자 길이는 30바이트 또는 128바이트로 제한됩니다. 이 제한은 Oracle 대상 데이터 스토어의 테이블 이름과 열 이름에 영향을 줍니다.

다음을 수행하여 오류를 피합니다.

- 버전의 제한 범위 안에서 Oracle 대상 테이블 이름을 짓습니다.
- 기본 열 이름은 데이터의 필드 이름에서 생성됩니다. 열 이름이 한도보다 긴 경우, ApplyMapping 또는 RenameField 변환을 사용하여 열 이름이 한도를 넘지 않도록 변경합니다.

오류: Amazon S3: 객체의 스토리지 클래스에 대해 작업이 유효하지 않습니다.

이 오류가 AWS Glue 반환되면 AWS Glue 작업에서 Amazon S3 스토리지 클래스 계층 전체에 파티션이 있는 테이블에서 데이터를 읽고 있을 수 있습니다.

- 스토리지 클래스 제외를 사용하면 이러한 스토리지 클래스 계층 전체에 파티션이 있는 테이블에서 작업이 제대로 작동하는지 확인할 수 있습니다. AWS Glue 제외하지 않으면 이러한 계층에서 데이터를 읽는 작업은 `AmazonS3Exception: The operation is not valid for the object's storage class` 오류와 함께 실패합니다.

자세한 정보는 [Amazon S3 스토리지 클래스 제외](#)을 참조하세요.

오류: Amazon S3 시간 제한

연결 시간 초과 오류가 AWS Glue 반환되는 경우 다른 AWS 지역의 Amazon S3 버킷에 액세스하려고 하기 때문일 수 있습니다.

- Amazon S3 VPC 엔드포인트는 트래픽을 지역 내 버킷으로만 라우팅할 수 있습니다. AWS 버킷을 다른 지역으로 연결할 필요가 있다면 차선책으로 NAT 게이트웨이를 사용합니다. 자세한 내용은 [NAT 게이트웨이](#) 단원을 참조하세요.

오류: Amazon S3 액세스가 거부되었습니다.

Amazon S3 버킷 또는 객체에 대한 액세스 거부 오류가 AWS Glue 반환되는 경우, 제공된 IAM 역할에 데이터 스토어에 대한 권한을 가진 정책이 없기 때문일 수 있습니다.

- ETL 작업은 소스 또는 대상으로 사용되는 Amazon S3 데이터 스토어로 액세스할 수 있어야 합니다. 크롤러는 크롤하는 Amazon S3 데이터 스토어로 액세스할 수 있어야 합니다. 자세한 정보는 [2단계: AWS Glue에 대한 IAM 역할 생성](#)을 참조하세요.

오류: Amazon S3 액세스 키 ID가 없습니다.

작업을 실행할 때 액세스 키 ID가 존재하지 않음 오류가 AWS Glue 반환되는 경우 다음 이유 중 하나 때문일 수 있습니다.

- ETL 작업은 IAM 역할을 사용하여 데이터 스토어로 액세스합니다. 작업 시작 전에 작업용 IAM 역할이 삭제된 것은 아닌지 확인합니다.
- IAM 역할은 데이터 스토어에 대한 액세스 권한을 가지고 있습니다. `s3:ListBucket`이 포함된 모든 연결된 Amazon S3 정책이 올바른지 확인합니다.

오류: `s3a://` URI를 사용해 Amazon S3에 액세스할 때 작업 실행이 실패합니다.

작업이 핸들러 클래스로 XML 문서 파싱에 실패 같은 오류를 반환한다면 `s3a://` URI를 사용하여 수백 개의 파일을 기록하는 데 실패했기 때문입니다. 대신에 `s3://` URI를 사용하여 데이터 스토어에 액세스합니다. 다음 예외 흔적은 다음과 같은 오류를 찾아봐야 합니다.

```

1. com.amazonaws.SdkClientException: Failed to parse XML document with handler class
   com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser$ListBucketHandler
2. at
   com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseXmlInputStream(XmlResponsesSaxParser.java:100)
3. at
   com.amazonaws.services.s3.model.transform.XmlResponsesSaxParser.parseListBucketObjectsResponse(XmlResponsesSaxParser.java:110)
4. at com.amazonaws.services.s3.model.transform.Unmarshallers
   $ListObjectsUnmarshaller.unmarshall(Unmarshallers.java:70)
5. at com.amazonaws.services.s3.model.transform.Unmarshallers
   $ListObjectsUnmarshaller.unmarshall(Unmarshallers.java:59)
6. at
   com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponseHandler.java:62)
7. at
   com.amazonaws.services.s3.internal.S3XmlResponseHandler.handle(S3XmlResponseHandler.java:31)
8. at
   com.amazonaws.http.response.AwsResponseHandlerAdapter.handle(AwsResponseHandlerAdapter.java:70)
9. at com.amazonaws.http.AmazonHttpClient
   $RequestExecutor.handleResponse(AmazonHttpClient.java:1554)
10. at com.amazonaws.http.AmazonHttpClient
   $RequestExecutor.executeOneRequest(AmazonHttpClient.java:1272)
11. at com.amazonaws.http.AmazonHttpClient
   $RequestExecutor.executeHelper(AmazonHttpClient.java:1056)
12. at com.amazonaws.http.AmazonHttpClient
   $RequestExecutor.doExecute(AmazonHttpClient.java:743)
13. at com.amazonaws.http.AmazonHttpClient
   $RequestExecutor.executeWithTimer(AmazonHttpClient.java:717)
14. at com.amazonaws.http.AmazonHttpClient
   $RequestExecutor.execute(AmazonHttpClient.java:699)

```

```

15. at com.amazonaws.http.AmazonHttpClient$RequestExecutor.access
$500(AmazonHttpClient.java:667)
16. at com.amazonaws.http.AmazonHttpClient
$RequestExecutionBuilderImpl.execute(AmazonHttpClient.java:649)
17. at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:513)
18. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4325)
19. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4272)
20. at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4266)
21. at com.amazonaws.services.s3.AmazonS3Client.listObjects(AmazonS3Client.java:834)
22. at org.apache.hadoop.fs.s3a.S3AFileSystem.getFileStatus(S3AFileSystem.java:971)
23. at
org.apache.hadoop.fs.s3a.S3AFileSystem.deleteUnnecessaryFakeDirectories(S3AFileSystem.java:115)
24. at org.apache.hadoop.fs.s3a.S3AFileSystem.finishedWrite(S3AFileSystem.java:1144)
25. at org.apache.hadoop.fs.s3a.S3AOutputStream.close(S3AOutputStream.java:142)
26. at org.apache.hadoop.fs.FSDataOutputStream
$PositionCache.close(FSDataOutputStream.java:74)
27. at org.apache.hadoop.fs.FSDataOutputStream.close(FSDataOutputStream.java:108)
28. at org.apache.parquet.hadoop.ParquetFileWriter.end(ParquetFileWriter.java:467)
29. at
org.apache.parquet.hadoop.InternalParquetRecordWriter.close(InternalParquetRecordWriter.java:1)
30. at
org.apache.parquet.hadoop.ParquetRecordWriter.close(ParquetRecordWriter.java:112)
31. at
org.apache.spark.sql.execution.datasources.parquet.ParquetOutputWriter.close(ParquetOutputWriter.java:1)
32. at org.apache.spark.sql.execution.datasources.FileFormatWriter
$SingleDirectoryWriteTask.releaseResources(FileFormatWriter.scala:252)
33. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun
$org$apache$spark$sql$execution$databases$FileFormatWriter$$executeTask
$3.apply(FileFormatWriter.scala:191)
34. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun
$org$apache$spark$sql$execution$databases$FileFormatWriter$$executeTask
$3.apply(FileFormatWriter.scala:188)
35. at org.apache.spark.util.Utils
$.tryWithSafeFinallyAndFailureCallbacks(Utils.scala:1341)
36. at org.apache.spark.sql.execution.datasources.FileFormatWriter$.org$apache$spark
$sql$execution$databases$FileFormatWriter$$executeTask(FileFormatWriter.scala:193)
37. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$
$anonfun$3.apply(FileFormatWriter.scala:129)
38. at org.apache.spark.sql.execution.datasources.FileFormatWriter$$anonfun$write$1$
$anonfun$3.apply(FileFormatWriter.scala:128)
39. at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
40. at org.apache.spark.scheduler.Task.run(Task.scala:99)
41. at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:282)
42. at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)

```

```
43. at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
44. at java.lang.Thread.run(Thread.java:748)
```

오류: Amazon S3 서비스 토큰이 만료되었습니다.

Amazon Redshift 간에 데이터를 이동할 때에는 1시간 후에 만료되는 임시 Amazon S3 자격 증명에 사용됩니다. 따라서 실행 시간이 긴 작업은 실패할 수 있습니다. Amazon Redshift 간에 데이터를 이동하기 위해 오래 실행되는 작업을 설정하는 방법은 [aws-glue-programming-etl-connect-redshift-home](#) 섹션을 참조하세요.

오류: 네트워크 인터페이스용 프라이빗 DNS를 찾을 수 없습니다.

작업 또는 개발 엔드포인트가 할당하는 작업을 실패하면 네트워크 설정 문제일 것입니다.

- Amazon이 제공한 DNS를 사용 중이면 `enableDnsHostnames` 값은 `true`로 설정해야 합니다. 자세한 내용은 [DNS](#)를 참조하십시오.

오류: 개발 엔드포인트 프로비저닝이 실패했습니다.

개발 엔드포인트를 성공적으로 AWS Glue 프로비저닝하지 못하는 경우 네트워크 설정 문제 때문일 수 있습니다.

- 개발 엔드포인트를 정의하면 VPC, 서브넷 및 보안 그룹이 특정 요구 사항에 부합한다는 것을 확인하도록 검증할 수 있습니다.
- 조건부 SSH 퍼블릭 키를 제공했다면 유효한 SSH 퍼블릭 키인지 확인합니다.
- VPC가 유효한 [DHCP option set(DHCP 옵션 세트)]를 사용하도록 VPC 콘솔을 확인합니다. 자세한 내용은 [DHCP 옵션 세트](#)를 참조하십시오.
- 여전히 클러스터가 프로비저닝 상태로 유지된다면 AWS Support에 문의합니다.

오류: 노트북 서버 생성이 실패했습니다.

개발 엔드포인트용 노트북 서버를 만들지 AWS Glue 못하면 다음 문제 중 하나가 원인일 수 있습니다.

- AWS Glue 노트북 서버를 설정할 때 Amazon EC2에 IAM 역할을 전달합니다. IAM 역할은 Amazon EC2의 신뢰 관계를 맺고 있어야 합니다.
- IAM 역할은 동일한 이름의 인스턴스 프로파일을 가지고 있어야 합니다. IAM 콘솔에서 Amazon EC2를 위한 역할을 생성할 때 동일한 이름의 인스턴스 프로파일이 자동으로 생성됩니다. 유효하지 않은

인스턴스 프로파일 이름 `iamInstanceProfile.name`에 따라 로그 오류를 확인합니다. 자세한 내용은 [인스턴스 프로파일 사용](#) 섹션을 참조하세요.

- 역할이 노트북 서버를 생성하기 위해 통과한 정책에서 `aws-glue*` 버킷에 액세스할 수 있는 권한이 있는지 확인합니다.

오류: 로컬 노트북 시작이 실패했습니다.

로컬 노트북이 시작을 못하고 디렉토리 혹은 폴더를 찾을 수 없다는 오류를 보고 하면 다음과 같은 문제 중 하나일 수 있습니다.

- Microsoft Windows에서 실행하고 있다면 `JAVA_HOME` 환경 변수가 올바른 자바 디렉토리를 가리키는지 확인하십시오. 이 변수를 업데이트하지 않고서 Java를 업데이트할 수 있습니다. 그리고 존재하지 않는 폴더를 가리키면 Jupyter 노트북은 시작하지 못합니다.

오류: 크롤러 실행이 실패했습니다.

데이터를 카탈로그화하는 크롤러를 성공적으로 AWS Glue 실행하지 못하면 다음 이유 중 하나가 원인일 수 있습니다. 먼저 오류가 AWS Glue 콘솔 크롤러 목록에 있는지 확인합니다. 크롤러 이름 옆에 중요 표시 아이콘이 있는지 확인하고 관련 메시지가 아이콘 옆에 뜨는지 확인합니다.

- Logs under에서 크롤러 실행에 대한 로그를 확인하세요. `CloudWatch /aws-glue/crawlers`

오류: 파티션이 업데이트되지 않았습니다.

ETL 작업을 실행할 때 데이터 카탈로그의 파티션이 업데이트되지 않은 경우 로그에 있는 `DataSink` 클래스의 다음 로그 명령문이 유용할 수 있습니다. `CloudWatch`

- `"Attempting to fast-forward updates to the Catalog - nameSpace:"` - 이 작업에서 수정을 시도한 데이터베이스, 테이블 및 `catalogId`를 표시합니다. 이 문이 여기에 없는 경우 `enableUpdateCatalog`가 `true`로 설정되어 있으며 `getSink()` 파라미터로 또는 `additional_options`로 제대로 전달되는지 확인하십시오.
- `"Schema change policy behavior:"` - 전달한 스키마 `updateBehavior` 값을 표시합니다.
- `"Schemas qualify (schema compare):"` - `true` 또는 `false`입니다.
- `"Schemas qualify (case-insensitive compare):"` - `true` 또는 `false`입니다.

- 둘 다 updateBehavior false이고 로 UPDATE_IN_DATABASE 설정되지 않은 경우 DynamicFrame 스키마가 동일하거나 데이터 카탈로그 테이블 스키마에 표시된 열의 하위 집합을 포함해야 합니다.

파티션 업데이트에 대한 자세한 내용은 [AWS Glue ETL 작업을 사용하여 데이터 카탈로그에서 스키마 업데이트 및 새 파티션 추가](#) 섹션을 참조하세요.

오류: 버전 불일치로 인해 작업 북마크 업데이트 실패

Amazon S3의 다양한 데이터 세트에 동일한 변환/로직을 적용하기 위해 AWS Glue 작업을 파라미터화하려고 할 수 있습니다. 제공된 위치에서 처리된 파일을 추적하고자 합니다. 동일한 소스 버킷에서 동일한 작업을 실행하고 동일한/다른 대상에 동시에 쓰는 경우(동시성 >1) 다음 오류와 함께 작업이 실패합니다.

```
py4j.protocol.Py4JJavaError: An error occurred while
callingz:com.amazonaws.services.glue.util.Job.commit.:com.amazonaws.services.gluejobexecutor.m
Continuation update failed due to version mismatch. Expected version 2 but found
version 3
```

솔루션: 동시성을 1로 설정하거나 작업을 동시에 실행하지 마십시오.

현재 AWS Glue 북마크는 동시 작업을 실행을 지원하지 않으며 커밋은 실패합니다.

오류: 작업 북마크가 사용 설정된 경우 작업이 데이터를 재처리합니다.

AWS Glue 작업 북마크를 활성화했지만 ETL 작업이 이전 실행에서 이미 처리된 데이터를 재처리하는 경우가 있을 수 있습니다. 이 오류의 일반적인 원인을 확인해 보십시오.

최대 동시성

작업의 최대 동시 실행 횟수를 기본값인 1보다 크게 설정하면 작업 북마크를 방해할 수 있습니다. 이는 작업 북마크가 객체의 마지막 수정 시간을 확인하여 재처리가 필요한 객체를 확인할 때 발생할 수 있습니다. 자세한 내용은 [에서 Spark 작업에 대한 작업 속성 구성 AWS Glue](#)에서 최대 동시성 관련 내용을 참조하십시오.

작업 객체 누락

작업 실행 스크립트는 다음 커밋으로 끝나야 합니다.

```
job.commit()
```


이 객체를 포함하면 작업 실행의 타임스탬프와 경로가 AWS Glue 기록됩니다. 동일한 경로를 사용하여 작업을 다시 실행하는 경우는 새 파일만 AWS Glue 처리합니다. 이 객체를 포함하지 않고 작업 북마크를 활성화한 경우, 이 작업은 새 파일과 함께 이미 처리된 파일도 재처리하고 작업의 대상 데이터 스토어에 중복 항목을 생성합니다.

변환 컨텍스트 파라미터 누락

변환 컨텍스트가 GlueContext 클래스의 선택적 파라미터이기는 하지만, 이 파라미터를 넣지 않으면 작업 북마크가 제대로 작동하지 않습니다. 이 오류를 해결하려면 다음과 같이 [만들](#) 때 변환 컨텍스트 매개 변수를 추가하십시오. DynamicFrame

```
sample_dynF=create_dynamic_frame_from_catalog(database,
table_name,transformation_ctx="sample_dynF")
```

입력 소스

입력 소스에 대해 관계형 데이터베이스(JDBC 연결)를 사용하는 경우 테이블의 기본 키가 순차적인 경우에만 작업 북마크가 작동합니다. 작업 북마크는 업데이트된 행이 아니라 새 행에 대해 작동합니다. 이는 작업 북마크가 이미 있는 기본 키를 검색하기 때문입니다. 입력 소스가 Amazon Simple Storage Service(Amazon S3)인 경우에는 적용되지 않습니다.

최근 수정 시간

Amazon S3 입력 소스의 경우 작업 북마크는 파일 이름이 아니라 객체의 마지막으로 수정된 시간을 검사해 어떤 객체를 재처리해야 하는지 확인합니다. 마지막 작업 실행 이후로 입력 소스 데이터가 수정된 경우, 작업을 다시 실행하면 그 파일이 재처리됩니다.

오류: 의 VPC 간 페일오버 동작 AWS Glue

AWS Glue 4.0 및 이전 버전의 작업에 대한 페일오버에는 다음 프로세스가 사용됩니다.

요약: AWS Glue 연결은 작업 실행이 제출될 때 선택됩니다. 작업 실행 시 일부 문제(IP 주소 부족, 소스 연결, 라우팅 문제)가 발생하면 작업 실행이 실패합니다. 재시도가 구성된 AWS Glue 경우 동일한 연결로 재시도합니다.

1. 각 실행 시도에 대해 작업 구성에 나열된 순서대로 연결 상태를 확인합니다. 이때 사용할 수 있는 연결을 찾을 때까지 해당 연결 상태를 확인합니다. AWS Glue 가용 영역 (AZ) 에 장애가 발생하는 경우 해당 AZ의 연결은 검사에 실패하고 건너뛰게 됩니다.
2. AWS Glue 다음을 사용하여 연결을 검증합니다.
 - 유효한 Amazon VPC ID 및 서브넷을 확인합니다.

- NAT 게이트웨이 또는 Amazon VPC 엔드포인트가 존재하는지 확인합니다.
- 서브넷에 할당된 IP 주소가 1개 이상인지 확인합니다.
- AZ가 정상 상태인지 확인합니다.

AWS Glue 작업 실행 제출 시에는 연결을 확인할 수 없습니다.

3. Amazon VPC를 사용하는 작업의 경우, 모든 드라이버와 실행기는 작업 실행 제출 시 선택한 연결을 사용하여 동일한 AZ에서 생성됩니다.
4. 재시도가 구성된 AWS Glue 경우 동일한 연결로 재시도합니다. 이 연결과 관련된 문제가 오래 지속된다고 보장할 수 없기 때문입니다. AZ에 장애가 발생하면 해당 AZ의 기존 작업 실행(작업 실행 단계에 따라 다름)이 실패할 수 있습니다. 재시도를 통해 AZ 장애를 감지하고 새 실행을 위해 다른 AZ를 선택해야 합니다.

크롤러가 Lake Formation 보안 인증을 사용할 때 발생하는 크롤러 오류 문제 해결

아래 정보를 사용하여 Lake Formation 보안 인증을 사용하여 크롤러를 구성하는 동안 다양한 문제를 진단하고 수정합니다.

오류: The S3 location: s3://examplepath is not registered

Lake Formation 보안 인증을 사용하여 크롤러를 실행하려면 먼저 Lake Formation 권한을 설정해야 합니다. 이 오류를 해결하려면 Lake Formation에 대상 Amazon S3 위치를 등록하세요. 자세한 내용을 알아보려면 [Registering an Amazon S3 location](#)(Amazon S3 위치 등록)을 참조하세요.

오류: User/Role is not authorized to perform: lakeformation:GetDataAccess on resource

IAM 콘솔 또는 AWS CLI(를) 사용하여 크롤러 역할에 대해 `lakeformation:GetDataAccess` 권한을 추가하세요. 이 권한을 통해 Lake Formation은 데이터에 액세스하기 위한 임시 보안 인증 요청을 승인합니다. 아래 정책을 참조하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lakeformation:GetDataAccess"
      ]
    }
  ]
}
```

```

    "Resource": "*"
  }
}

```

오류: Insufficient Lake Formation permission(s) on (Database name: exampleDatabase, Table Name: exampleTable)

Lake Formation 콘솔(<https://console.aws.amazon.com/lakeformation/>)에서 데이터베이스에 대한 크롤러 역할 액세스 권한(Create, Describe, Alter)을 부여합니다. 이는 출력 데이터베이스로 지정됩니다. 테이블에 대한 권한도 부여할 수 있습니다. 자세한 내용을 알아보려면 [명명된 리소스 메서드를 사용하여 데이터베이스 권한 부여](#)를 참조하세요.

오류: Insufficient Lake Formation permission(s) on s3://examplepath

1. 교차 계정 크롤링

- Amazon S3 버킷이 등록된 계정(계정 B)을 사용하여 Lake Formation 콘솔(<https://console.aws.amazon.com/lakeformation/>)에 로그인합니다. 크롤러가 실행될 계정에 데이터 위치 권한을 부여합니다. 이렇게 하면 크롤러가 대상 Amazon S3 위치에서 데이터를 읽을 수 있습니다.
- 크롤러가 생성된 계정(계정 A)에서 크롤러 실행에 사용되는 IAM 역할에 대상 Amazon S3 위치에 대한 데이터 위치 권한을 부여하여 크롤러가 Lake Formation의 대상에서 데이터를 읽을 수 있도록 합니다. 자세한 내용을 알아보려면 [데이터 위치 권한 부여\(외부 계정\)](#)를 참조하세요.

- 계정 내(크롤러 및 등록된 Amazon S3 위치가 동일한 계정에 있음) 크롤링 - Amazon S3 위치에서 크롤러 실행에 사용되는 IAM 역할에 데이터 위치 권한을 부여하여 크롤러가 Lake Formation의 대상에서 데이터를 읽을 수 있도록 합니다. 자세한 내용을 알아보려면 [Granting data location permissions \(same account\)](#)(데이터 위치 권한 부여(동일한 계정))를 참조하세요.

Lake Formation 보안 인증을 사용한 크롤러 구성에 대해 자주 묻는 질문

1. AWS 콘솔을 사용하여 Lake Formation 보안 인증을 사용하여 실행되도록 크롤러를 구성하려면 어떻게 해야 하나요?

AWS Glue 콘솔(<https://console.aws.amazon.com/glue/>)에서 크롤러를 구성하는 동안 Use Lake Formation credentials for crawling Amazon S3 data source(Amazon S3 데이터 소스 크롤링에 Lake Formation 자격 증명 사용) 옵션을 선택합니다. 크로스 계정 크롤링의 경우 대상 Amazon S3 위치가 Lake Formation에 등록된 AWS 계정 ID를 지정합니다. 계정 내 크롤링의 경우 accountId 필드는 선택 사항입니다.

2. AWS CLI을(를) 사용하여 Lake Formation 보안 인증을 사용하여 실행되도록 크롤러를 구성하려면 어떻게 해야 합니까?

CreateCrawler API 직접 호출 중에 LakeFormationConfiguration 추가:

```
"LakeFormationConfiguration": {
  "UseLakeFormationCredentials": true,
  "AccountId": "111111111111" (AWS account ID where the target Amazon S3 location
  is registered with Lake Formation)
}
```

3. Lake Formation 보안 인증을 사용하는 크롤러에 대해 지원되는 대상은 무엇입니까?

Lake Formation 보안 인증을 사용하는 크롤러는 Amazon S3(계정 내 및 크로스 계정 크롤링), 계정 내 데이터 카탈로그 대상(기본 위치가 Amazon S3인 경우), Apache Iceberg 대상에 대해서만 지원됩니다.

4. Lake Formation 보안 인증을 사용하여 단일 크롤러의 일부로 여러 Amazon S3 버킷을 크롤링할 수 있습니까?

아니요, Lake Formation 보안 인증 벤딩을 사용하는 크롤링 대상의 경우 기본 Amazon S3 위치가 동일한 버킷에 속해야 합니다. 예를 들어, 고객은 동일한 버킷(버킷 1) 아래에 있는 경우 여러 대상 위치 (s3://bucket1/folder1, s3://bucket1/folder2)을(를) 사용할 수 있습니다. 다른 버킷 지정(s3://bucket1/folder1, s3://bucket2/folder2)은 지원되지 않습니다.

로그에서 AWS Glue for Ray 오류 해결

AWS Glue를 사용하면 작업 실행 중에 Ray 프로세스에서 생성된 로그에 액세스할 수 있습니다. Ray 작업에서 오류나 예상치 못한 동작이 발생하는 경우 먼저 로그에서 정보를 수집하여 오류의 원인을 파악합니다. 대화형 세션에 대해서도 유사한 로그를 제공합니다. 세션 로그는 /aws-glue/ray/sessions 접두사와 함께 제공됩니다.

작업이 실행되면 로그 라인이 CloudWatch에 실시간으로 전송됩니다. 실행을 완료한 후 인쇄 명령문이 CloudWatch 로그에 추가됩니다. 로그는 작업 실행 후 2주 동안 보관됩니다.

Ray 작업 로그 검사

작업이 실패하면 작업 이름과 작업 실행 ID를 수집합니다. 이러한 항목은 AWS Glue 콘솔에서 찾을 수 있습니다. 작업 페이지로 이동한 다음 Runs(실행) 탭으로 이동합니다. Ray 작업 로그는 다음과 같은 전용 CloudWatch 로그 그룹에 저장됩니다.

- `/aws-glue/ray/jobs/script-log/` - 기본 Ray 스크립트에서 내보낸 로그를 저장합니다.
- `/aws-glue/ray/jobs/ray-monitor-log/` - Ray autoscaler 프로세스에서 내보낸 로그를 저장합니다. 이러한 로그는 헤드 노드에 대해 생성되며 다른 워커 노드에 대해서는 생성되지 않습니다.
- `/aws-glue/ray/jobs/ray-gcs-logs/` - 글로벌 제어 스토어(GCS) 프로세스에서 생성된 로그를 저장합니다. 이러한 로그는 헤드 노드에 대해 생성되며 다른 워커 노드에 대해서는 생성되지 않습니다.
- `/aws-glue/ray/jobs/ray-process-logs/` - 헤드 노드에서 실행되는 다른 Ray 프로세스(주로 대시보드 에이전트)에서 생성된 로그를 저장합니다. 이러한 로그는 헤드 노드에 대해 생성되며 다른 워커 노드에 대해서는 생성되지 않습니다.
- `/aws-glue/ray/jobs/ray-raylet-logs/` - 각 raylet 프로세스에서 생성된 로그를 저장합니다. 이러한 로그는 헤드 노드를 포함하여 각 워커 노드에 대해 단일 스트림으로 수집됩니다.
- `/aws-glue/ray/jobs/ray-worker-out-logs/` - 클러스터의 각 작업자에 대한 stdout 로그를 저장합니다. 이러한 로그는 헤드 노드를 포함한 각 워커 노드에서 생성됩니다.
- `/aws-glue/ray/jobs/ray-worker-err-logs/` - 클러스터의 각 작업자에 대한 stderr 로그를 저장합니다. 이러한 로그는 헤드 노드를 포함한 각 워커 노드에서 생성됩니다.
- `/aws-glue/ray/jobs/ray-runtime-env-log/` - Ray 설정 프로세스에 대한 로그를 저장합니다. 이러한 로그는 헤드 노드를 포함한 각 워커 노드에서 생성됩니다.

Ray 작업 오류 해결

Ray 로그 그룹의 구성을 이해하고 오류 해결에 도움이 되는 로그 그룹을 찾으려면 Ray 아키텍처에 대한 배경 정보를 알고 있으면 도움이 됩니다.

AWS Glue ETL에서 작업자는 인스턴스에 해당합니다. AWS Glue 작업에 대한 작업자를 구성할 때 해당 작업 전용 인스턴스의 유형과 수량을 설정합니다. Ray에서는 작업자라는 용어를 다양한 방식으로 사용합니다.

Ray에서는 헤드 노드와 워커 노드를 사용하여 Ray 클러스터 내에서 인스턴스의 책임을 구분합니다. Ray 워커 노드는 분산 계산 결과를 얻기 위해 계산을 수행하는 여러 액터 프로세스를 호스팅할 수 있습니다. 함수의 복제본을 실행하는 액터를 복제본이라고 합니다. 복제본 액터를 작업자 프로세스라고도 합니다. 복제본은 클러스터를 조정하기 위한 추가 프로세스를 실행하기 때문에 헤드라고 하는 헤드 노드에서도 실행될 수 있습니다.

계산에 기여하는 각 액터는 자체 로그 스트림을 생성합니다. 이를 통해 다음과 같은 몇 가지 인사이트를 얻을 수 있습니다.

- 로그를 내보내는 프로세스 수가 작업에 할당된 작업자 수보다 많을 수 있습니다. 각 인스턴스의 각 코어에는 주로 액터가 있습니다.
- Ray 헤드 노드는 클러스터 관리 및 시작 로그를 내보냅니다. 반면 Ray 워커 노드는 수행된 작업에 대한 로그만 내보냅니다.

Ray 아키텍처에 대한 자세한 내용은 Ray 설명서의 [아키텍처 백서](#)를 참조하세요.

문제 영역: Amazon S3 액세스

작업 실행 실패 메시지를 확인합니다. 충분한 정보가 제공되지 않는 경우 `/aws-glue/ray/jobs/script-log/` 섹션을 확인하세요.

문제 영역: PIP 종속성 관리

`/aws-glue/ray/jobs/ray-runtime-env-log/`을 검토합니다.

문제 영역: 메인 프로세스의 중간 값 검사

기본 스크립트에서 `stderr` 또는 `stdout`에 쓰고 `/aws-glue/ray/jobs/script-log/`에서 로그를 검색합니다.

문제 영역: 하위 프로세스의 중간 값 검사

`remote` 함수에서 `stderr` 또는 `stdout`에 씁니다. 그런 다음 `/aws-glue/ray/jobs/ray-worker-out-logs/` 또는 `/aws-glue/ray/jobs/ray-worker-err-logs/`에서 로그를 검색합니다. 함수가 복제본에서 실행되었을 수 있으므로 원하는 출력을 찾기 위해 여러 로그를 검사해야 할 수 있습니다.

문제가 있는 영역: 오류 메시지에서 IP 주소 해석

특정 오류 상황에서는 작업에서 IP 주소가 포함된 오류 메시지를 생성할 수 있습니다. 이러한 IP 주소는 임시 주소이며 클러스터에서 노드를 식별하고 노드 사이에서 통신하는 데 사용됩니다. 노드의 로그는 IP 주소를 기반으로 고유한 접미사를 사용하는 로그 스트림에 게시됩니다.

CloudWatch에서는 이 접미사를 식별하여 로그를 필터링해 이 IP 주소에 특정한 로그를 검사할 수 있습니다. 예를 들어 주어진 `FAILED_IP` 및 `JOB_RUN_ID`에 대해 다음을 사용해 접미사를 식별할 수 있습니다.

```
filter @logStream like /JOB_RUN_ID/
| filter @message like /IP-/
| parse @message "IP-[*]" as ip
```

```

| filter ip like /FAILED_IP/
| fields replace(ip, ":", "_") as uIP
| stats count_distinct by uIP as logStreamSuffix
| display logStreamSuffix

```

AWS Glue 기계 학습 예외 사항

이 주제에서는 기계 학습과 관련된 AWS Glue 예외에 대한 HTTP 오류 코드 및 문자열에 대해 설명합니다. 작업을 수행할 때 발생할 수 있는 각 기계 학습 작업에 대해 오류 코드 및 오류 문자열이 제공됩니다. 또한 오류를 일으킨 작업을 다시 시도할 수 있는지 여부를 확인할 수 있습니다.

CancelMLTaskRunActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”
 - “[transformName] 변환을 위한 [accountId] 계정에서 [taskRunId]에 대한 ML 작업 실행을 찾을 수 없습니다.”

재시도 가능: 아니요.

CreateMLTaskRunActivity

이 활동에는 다음과 같은 예외가 있습니다.

- InvalidInputException (400)
 - “예기치 않은 입력으로 인해 내부 서비스 오류가 발생했습니다.”
 - “변환에서 AWS Glue 테이블 입력 소스를 지정해야 합니다.”
 - “입력 소스 열 [columnName]에 카탈로그에 정의된 잘못된 데이터 형식이 있습니다.”
 - “정확히 하나의 입력 레코드 테이블을 제공해야 합니다.”
 - “데이터베이스 이름을 지정해야 합니다.”
 - “테이블 이름을 지정해야 합니다.”
 - “스키마가 변환에 정의되지 않았습니다.”
 - “스키마는 주어진 기본 키 [primaryKey]를 포함해야 합니다.”

- “데이터 카탈로그 스키마를 가져오는 중 문제가 발생했습니다. [message].”
- “최대 용량과 작업자 수/유형을 동시에 설정할 수 없습니다.”
- “WorkerType과 NumberOfWorkers 둘 다 설정해야 합니다.”
- “MaxCapacity는 >=[maxCapacity]이어야 합니다.”
- “NumberOfWorkers는 >=[maxCapacity]이어야 합니다.”
- “최대 재시도는 음수가 아니어야 합니다.”
- “일치 항목 찾기 파라미터가 설정되지 않았습니다.”
- “일치 항목 찾기 파라미터에 기본 키를 지정해야 합니다.”

재시도 가능: 아니요.

- AlreadyExistsException (400)
 - “이름이 [transformName]인 변형이 이미 있습니다.”

재시도 가능: 아니요.

- IdempotentParameterMismatchException (400)
 - “[transformName] 변환에 대한 멱등성 생성 요청에 일치하지 않는 파라미터가 있습니다.”

재시도 가능: 아니요.

- InternalServiceException (500)
 - “종속성 오류가 발생했습니다.”

재시도 가능: 예.

- ResourceNumberLimitExceededException (400)
 - “ML 변환 수([count])가 [limit] 변환의 제한을 초과했습니다.”

재시도 가능: 예, 이 새 변환을 위한 공간을 만들기 위해 변환을 삭제한 후 재시도합니다.

DeleteMLTransformActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

GetMLTaskRunActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”
 - “[transformName] 변환을 위한 [accountId] 계정에서 [taskId]에 대한 ML 작업 실행을 찾을 수 없습니다.”

재시도 가능: 아니요.

GetMLTaskRunsActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”
 - “[transformName] 변환을 위한 [accountId] 계정에서 [taskId]에 대한 ML 작업 실행을 찾을 수 없습니다.”

재시도 가능: 아니요.

GetMLTransformActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

GetMLTransformsActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

- `InvalidInputException` (400)
 - “계정 ID는 비워 둘 수 없습니다.”
 - “[column] 열에는 정렬이 지원되지 않습니다.”
 - “[column]은(는) 비워 둘 수 없습니다.”
 - “예기치 않은 입력으로 인해 내부 서비스 오류가 발생했습니다.”

재시도 가능: 아니요.

GetSaveLocationForTransformArtifactActivity

이 활동에는 다음과 같은 예외가 있습니다.

- `EntityNotFoundException` (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

- `InvalidInputException` (400)
 - “지원되지 않는 아티팩트 유형 [artifactType]입니다.”
 - “예기치 않은 입력으로 인해 내부 서비스 오류가 발생했습니다.”

재시도 가능: 아니요.

GetTaskRunArtifactActivity

이 활동에는 다음과 같은 예외가 있습니다.

- `EntityNotFoundException` (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”
 - “[transformName] 변환을 위한 [accountId] 계정에서 [taskRunId]에 대한 ML 작업 실행을 찾을 수 없습니다.”

재시도 가능: 아니요.

- `InvalidInputException` (400)
 - “파일 이름 '[fileName]'이(가) 게시에 유효하지 않습니다.”

- “[taskType] 작업 유형에 대한 아티팩트를 검색할 수 없습니다.”
- “[artifactType]에 대한 아티팩트를 검색할 수 없습니다.”
- “예기치 않은 입력으로 인해 내부 서비스 오류가 발생했습니다.”

재시도 가능: 아니요.

PublishMLTransformModelActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”
 - “계정 ID - [accountId] - 및 변환 ID - [transformId]에서 버전 - [version] 버전이 있는 기존 모델을 찾을 수 없습니다.”

재시도 가능: 아니요.

- InvalidInputException (400)
 - “파일 이름 '[fileName]'이(가) 게시에 유효하지 않습니다.”
 - “부호 없는 문자열 [string] 앞에 잘못된 마이너스 부호가 있습니다.”
 - “[string]의 끝에 잘못된 숫자가 있습니다.”
 - “문자열 값 [string]은(는) 부호 없는 long 범위를 초과합니다.”
 - “예기치 않은 입력으로 인해 내부 서비스 오류가 발생했습니다.”

재시도 가능: 아니요.

PullLatestMLTransformModelActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

- InvalidInputException (400)
 - “예기치 않은 입력으로 인해 내부 서비스 오류가 발생했습니다.”

재시도 가능: 아니요.

- **ConcurrentModificationException (400)**
 - “파라미터가 일치하지 않는 레이스 입력으로 인해 훈련할 모델 버전을 만들 수 없습니다.”
 - “변환 ID [transformId]에 대한 ML 변환 모델이 오래되었거나 다른 프로세스에 의해 업데이트되고 있습니다. 다시 시도하십시오.”

재시도 가능: 예.

PutJobMetadataForMLTransformActivity

이 활동에는 다음과 같은 예외가 있습니다.

- **EntityNotFoundException (400)**
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”
 - “[transformName] 변환을 위한 [accountId] 계정에서 [taskRunId]에 대한 ML 작업 실행을 찾을 수 없습니다.”

재시도 가능: 아니요.

- **InvalidInputException (400)**
 - “예기치 않은 입력으로 인해 내부 서비스 오류가 발생했습니다.”
 - “알 수 없는 작업 메타데이터 유형 [jobType]입니다.”
 - “업데이트할 작업 실행 ID를 제공해야 합니다.”

재시도 가능: 아니요.

StartExportLabelsTaskRunActivity

이 활동에는 다음과 같은 예외가 있습니다.

- **EntityNotFoundException (400)**
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”
 - “계정 ID [accountId]에 변환 ID [transformId]에 대한 labelset가 없습니다.”

재시도 가능: 아니요.

- **InvalidInputException (400)**

- “[message].”
- “제공된 S3 경로가 변환과 동일한 리전에 있지 않습니다. 예상한 리전은 [region]이지만 받은 리전은 [region]입니다.”

재시도 가능: 아니요.

StartImportLabelsTaskRunActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

- InvalidInputException (400)
 - “[message].”
 - “잘못된 레이블 파일 경로입니다.”
 - “[labelPath]에서 레이블 파일에 액세스할 수 없습니다. [message].”
 - “변환에 제공된 IAM 역할을 사용할 수 없습니다. 역할: [role]입니다.”
 - “크기가 0인 잘못된 레이블 파일입니다.”
 - “제공된 S3 경로가 변환과 동일한 리전에 있지 않습니다. 예상한 리전은 [region]이지만 받은 리전은 [region]입니다.”

재시도 가능: 아니요.

- ResourceNumberLimitExceededException (400)
 - “레이블 파일이 [limit]MB 제한을 초과했습니다.”

재시도 가능: 아니요. 레이블 파일을 여러 개의 작은 파일로 나누는 것이 좋습니다.

StartMLEvaluationTaskRunActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

- `InvalidInputException` (400)
 - “정확히 하나의 입력 레코드 테이블을 제공해야 합니다.”
 - “데이터베이스 이름을 지정해야 합니다.”
 - “테이블 이름을 지정해야 합니다.”
 - “일치 항목 찾기 파라미터가 설정되지 않았습니다.”
 - “일치 항목 찾기 파라미터에 기본 키를 지정해야 합니다.”

재시도 가능: 아니요.

- `MLTransformNotReadyException` (400)
 - “이 작업은 준비 상태에 있는 변환에만 적용할 수 있습니다.”

재시도 가능: 아니요.

- `InternalServiceException` (500)
 - “종속성 오류가 발생했습니다.”

재시도 가능: 예.

- `ConcurrentRunsExceededException` (400)
 - “ML 작업 실행 수 [count]이(가) 작업 실행 [limit]의 변환 제한을 초과했습니다.”
 - “ML 작업 실행 수 [count]이(가) 작업 실행 [limit]의 제한을 초과했습니다.”

재시도 가능: 예, 작업 실행이 완료될 때까지 기다린 후 재시도합니다.

StartMLLabelingSetGenerationTaskRunActivity

이 활동에는 다음과 같은 예외가 있습니다.

- `EntityNotFoundException` (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

- `InvalidInputException` (400)
 - “정확히 하나의 입력 레코드 테이블을 제공해야 합니다.”
 - “데이터베이스 이름을 지정해야 합니다.”

- “테이블 이름을 지정해야 합니다.”
- “일치 항목 찾기 파라미터가 설정되지 않았습니다.”
- “일치 항목 찾기 파라미터에 기본 키를 지정해야 합니다.”

재시도 가능: 아니요.

- InternalServiceException (500)
 - “종속성 오류가 발생했습니다.”

재시도 가능: 예.

- ConcurrentRunsExceededException (400)
 - “ML 작업 실행 수 [count]이(가) 작업 실행 [limit]의 변환 제한을 초과했습니다.”

재시도 가능: 예, 작업 실행이 완료된 후 재시도합니다.

UpdateMLTransformActivity

이 활동에는 다음과 같은 예외가 있습니다.

- EntityNotFoundException (400)
 - “[transformName] 핸들이 있는 [accountId] 계정에서 MLTransform을 찾을 수 없습니다.”

재시도 가능: 아니요.

- InvalidInputException (400)
 - “이름이 [transformName]인 다른 변환이 이미 있습니다.”
 - “[message].”
 - “변환 이름은 비워 둘 수 없습니다.”
 - “최대 용량과 작업자 수/유형을 동시에 설정할 수 없습니다.”
 - “WorkerType과 NumberOfWorkers 둘 다 설정해야 합니다.”
 - “MaxCapacity는 >=[minMaxCapacity]이어야 합니다.”
 - “NumberOfWorkers는 >=[minNumWorkers]이어야 합니다.”
 - “최대 재시도는 음수가 아니어야 합니다.”
 - “예기치 않은 입력으로 인해 내부 서비스 오류가 발생했습니다.”
 - “일치 항목 찾기 파라미터가 설정되지 않았습니다.”
 - “일치 항목 찾기 파라미터에 기본 키를 지정해야 합니다.”

재시도 가능: 아니요.

- AlreadyExistsException (400)
 - “이름이 [transformName]인 변형이 이미 있습니다.”

재시도 가능: 아니요.

- IdempotentParameterMismatchException (400)
 - “[transformName] 변환에 대한 멱등성 생성 요청에 일치하지 않는 파라미터가 있습니다.”

재시도 가능: 아니요.

AWS Glue 할당량

AWS Support에 연락하여 AWS 일반 참조에 나열된 서비스 할당량에 대해 [할당량 증가를 요청](#)할 수 있습니다. 다르게 표시되지 않는 한 리전별로 각 할당량이 적용됩니다. 자세한 내용은 [AWS Glue 엔드포인트 및 할당량](#)을 참조하십시오.

AWS Glue 퍼포먼스 개선

성능 조정을 위한 기준선 전략

AWS Glue 성능을 향상시키기 위해 특정 성능 관련 AWS Glue 매개 변수를 업데이트하는 것을 고려할 수 있습니다. 파라미터를 조정할 준비를 할 때는 다음과 같은 모범 사례를 고려하세요.

- 문제를 식별하기 전에 성능 목표를 결정합니다.
- 조정 파라미터를 변경하기 전에 지표를 사용하여 문제를 식별합니다.

작업을 조정할 때 가장 일관된 결과를 얻으려면 조정 작업에 대한 기준선 전략을 수립하세요.

일반적으로 성능 조정은 다음 워크플로우에서 수행됩니다.

- 성능 목표를 결정합니다.
- 지표를 측정합니다.
- 병목 현상을 식별합니다.
- 병목 현상이 미치는 영향을 줄입니다.
- 의도한 목표를 달성할 때까지 2~4단계를 반복합니다.

작업 유형에 맞는 튜닝 전략

Spark 작업 - 규범적 지침의 [Apache Spark 작업 성능 조정 AWS Glue 모범 사례의](#) 지침을 따르십시오. AWS

기타 작업 — 다른 런타임 환경에서 사용할 수 있는 전략을 적용하여 Ray 및 AWS Glue Python 셀 작업에 맞게 조정할 AWS Glue 수 있습니다.

Apache Spark용 AWS Glue 작업의 성능 개선

Spark용 AWS Glue의 성능을 개선하려면 특정 성능 관련 AWS Glue 및 Spark 파라미터를 업데이트하는 것을 고려할 수 있습니다.

지표를 통해 병목 현상을 식별하고 그 영향을 줄이기 위한 구체적인 전략에 대한 자세한 내용은 AWS 권장 가이드의 [Apache Spark용 AWS Glue 작업 성능 조정 모범 사례](#)를 참조하세요. 이 가이드에서는 Spark 아키텍처, 탄력적 분산 데이터 세트 등 모든 런타임 환경에서 Apache Spark에 적용할 수 있는

주요 주제를 소개합니다. 설명서에서는 이러한 주제를 사용하여 셔플 최적화 및 작업 병렬화와 같은 특정 성능 튜닝 전략을 구현하도록 안내합니다.

Spark UI를 표시하도록 AWS Glue를 구성하여 병목 현상을 식별할 수 있습니다. 자세한 내용은 [the section called “Spark UI로 모니터링”](#) 단원을 참조하십시오.

또한 AWS Glue는 작업이 연결되는 특정 유형의 데이터 스토어에 적용할 수 있는 성능 특성을 제공합니다. 데이터 스토어의 성능 파라미터에 대한 참조 정보는 [the section called “연결 파라미터”](#)에서 찾을 수 있습니다.

AWS Glue ETL에서 푸시다운을 사용한 읽기 최적화

푸시다운은 데이터 검색에 대한 로직을 데이터 소스에 더 가깝게 푸시하는 최적화 기법입니다. 소스는 Amazon S3와 같은 파일 시스템 또는 데이터베이스일 수 있습니다. 소스에서 직접 특정 작업을 실행하는 경우 네트워크를 통해 AWS Glue에서 관리하는 Spark 엔진으로 모든 데이터를 가져오지 않아도 되므로 시간과 처리 용량을 절감할 수 있습니다.

이를 다른 방법으로 설명하자면 푸시다운이 데이터 스캔을 줄이는 것입니다. 이 기법이 적절한 시기를 식별하는 프로세스에 대한 자세한 내용은 AWS 권장 가이드의 Apache Spark용 AWS Glue 작업 성능 조정 모범 사례 가이드에 있는 [데이터 스캔량 감소](#)를 참조하세요.

Amazon S3에 저장된 파일에서 조건자 푸시다운

Amazon S3에서 접두사를 기준으로 구성된 파일을 사용하는 경우 푸시다운 조건자를 정의하여 대상 Amazon S3 경로를 필터링할 수 있습니다. 전체 데이터 세트를 읽고 DynamicFrame 내에서 필터를 적용하는 대신 AWS Glue 데이터 카탈로그에 저장된 파티션 메타데이터에 필터를 직접 적용할 수 있습니다. 이 방법을 사용하면 필요한 데이터만 선택적으로 나열하고 읽을 수 있습니다. 파티션별 버킷 쓰기를 포함하여 이 프로세스에 대한 자세한 내용은 [the section called “파티션 관리”](#) 섹션을 참조하세요.

Amazon S3에서 `push_down_predicate` 파라미터를 사용하여 조건부 푸시다운을 수행합니다. Amazon S3의 버킷을 연도, 월, 일을 기준으로 파티셔닝했다고 가정합니다. 2022년 6월의 고객 데이터를 검색하려는 경우 관련 Amazon S3 경로만 읽도록 AWS Glue에 지시할 수 있습니다. 이 경우 `push_down_predicate`는 `year='2022' and month='06'`입니다. 따라서 다음과 같이 읽기 작업을 수행할 수 있습니다.

Python

```
customer_records = glueContext.create_dynamic_frame.from_catalog(
    database = "customer_db",
    table_name = "customer_tbl",
```

```
push_down_predicate = "year='2022' and month='06'"
)
```

Scala

```
val customer_records = glueContext.getCatalogSource(
  database="customer_db",
  tableName="customer_tbl",
  pushDownPredicate="year='2022' and month='06'"
).getDynamicFrame()
```

이전 시나리오에서 `push_down_predicate`는 AWS Glue 데이터 카탈로그의 모든 파티션 목록을 검색하고 기본 Amazon S3 파일을 읽기 전에 파티션을 필터링합니다. 대부분의 경우 이 방법이 유용하지만 수백만 개의 파티션이 있는 데이터 세트를 작업할 때는 파티션을 나열하는 데 시간이 많이 걸릴 수 있습니다. 이 문제를 해결하기 위해 서버 측 파티션 정리를 사용하여 성능을 개선할 수 있습니다. AWS Glue 데이터 카탈로그에서 데이터에 대한 파티션 인덱스를 구축하면 됩니다. 파티션 인덱스에 대한 자세한 내용은 [the section called “파티션 인덱스 작업”](#) 섹션을 참조하세요. 그런 다음 `catalogPartitionPredicate` 옵션을 사용하여 인덱스를 참조할 수 있습니다. `catalogPartitionPredicate`를 사용하여 파티션을 검색하는 예제는 [the section called “카탈로그 파티션 조건자”](#) 섹션을 참조하세요.

JDBC 소스를 사용할 때 푸시다운

GlueContext에서 사용되는 AWS Glue JDBC 리더는 소스에서 직접 실행할 수 있는 사용자 지정 SQL 쿼리를 제공하여 지원되는 데이터베이스에서의 푸시다운을 지원합니다. `sampleQuery` 파라미터를 설정하여 이 작업을 수행할 수 있습니다. 샘플 쿼리는 선택할 열을 지정할 수 있을 뿐만 아니라 Spark 엔진으로 전송되는 데이터를 제한하는 푸시다운 조건자를 제공할 수 있습니다.

기본적으로 샘플 쿼리는 단일 노드에서 작동하므로 대량의 데이터를 처리할 때 작업에 실패할 수 있습니다. 이 기능을 사용하여 대규모로 데이터를 쿼리하려면 `enablePartitioningForSampleQuery`를 `true`로 설정하여 쿼리 파티셔닝을 구성해야 합니다. 그러면 선택한 키의 여러 노드에 쿼리가 배포됩니다. 쿼리 파티셔닝에는 몇 가지 다른 필수 구성 파라미터도 필요합니다. 쿼리 파티셔닝에 대한 자세한 내용은 [the section called “JDBC를 병렬로 읽기”](#) 섹션을 참조하세요.

`enablePartitioningForSampleQuery`를 설정하면 AWS Glue는 데이터베이스를 쿼리할 때 푸시다운 조건자를 파티셔닝 조건자와 결합합니다. AWS Glue에서 파티션 조건을 추가하려면 `sampleQuery`는 AND로 끝나야 합니다. (푸시다운 조건자를 제공하지 않는 경우 `sampleQuery`는

WHERE로 끝나야 합니다.) id가 1,000보다 큰 행만 검색하도록 조건자를 푸시하는 아래 예제를 참조하세요. 이 sampleQuery에서는 id가 지정된 값보다 큰 행의 이름 및 위치 열만 반환합니다.

Python

```
sample_query = "select name, location from customer_tbl WHERE id>=1000 AND"
customer_records = glueContext.create_dynamic_frame.from_catalog(
    database="customer_db",
    table_name="customer_tbl",
    sample_query = "select name, location from customer_tbl WHERE id>=1000 AND",

    additional_options = {
        "hashpartitions": 36 ,
        "hashfield":"id",
        "enablePartitioningForSampleQuery":True,
        "sampleQuery":sample_query
    }
)
```

Scala

```
val additionalOptions = Map(
    "hashpartitions" -> "36",
    "hashfield" -> "id",
    "enablePartitioningForSampleQuery" -> "true",
    "sampleQuery" -> "select name, location from customer_tbl WHERE id >= 1000
AND"
)

val customer_records = glueContext.getCatalogSource(
    database="customer_db",
    tableName="customer_tbl").getDynamicFrame()
```

Note

customer_tbl이 데이터 카탈로그와 기초 데이터 스토어의 이름이 다른 경우 쿼리가 기초 데이터 스토어로 전달되므로 sample_query에 기초 테이블 이름을 제공해야 합니다.

AWS Glue 데이터 카탈로그와 통합하지 않고도 JDBC 테이블을 쿼리할 수도 있습니다. 사용자 이름과 암호를 메서드의 파라미터로 제공하는 대신 `useConnectionProperties` 및 `connectionName`을 제공하여 기존 연결의 보안 인증을 재사용할 수 있습니다. 이 예제에서는 `my_postgre_connection`이라는 연결에서 보안 인증을 검색합니다.

Python

```
connection_options_dict = {
    "useConnectionProperties": True,
    "connectionName": "my_postgre_connection",
    "dbtable": "customer_tbl",
    "sampleQuery": "select name, location from customer_tbl WHERE id >= 1000 AND",
    "enablePartitioningForSampleQuery": True,
    "hashfield": "id",
    "hashpartitions": 36
}

customer_records = glueContext.create_dynamic_frame.from_options(
    connection_type="postgresql",
    connection_options=connection_options_dict
)
```

Scala

```
val connectionOptionsJson = """
{
  "useConnectionProperties": true,
  "connectionName": "my_postgre_connection",
  "dbtable": "customer_tbl",
  "sampleQuery": "select name, location from customer_tbl WHERE id >= 1000 AND",
  "enablePartitioningForSampleQuery" : true,
  "hashfield" : "id",
  "hashpartitions" : 36
}
"""

val connectionOptions = new JsonOptions(connectionOptionsJson)

val dyf = glueContext.getSource("postgresql",
connectionOptions).getDynamicFrame()
```

AWS Glue에서 푸시다운에 대한 참고 및 제한 사항

푸시다운은 개념상 비스트리밍 소스(AWS)에서 읽을 때 적용됩니다. Glue는 다양한 소스를 지원하며, 푸시다운 기능은 소스와 커넥터에 따라 다릅니다.

- Snowflake에 연결할 때는 query 옵션을 사용할 수 있습니다. AWS Glue 4.0 이상 버전의 Redshift 커넥터에도 비슷한 기능이 있습니다. query를 사용하여 Snowflake에서 읽는 방법에 대한 자세한 내용은 [the section called “Snowflake에서 읽기”](#) 섹션을 참조하세요.
- DynamoDB ETL 리더는 필터 또는 푸시다운 조건자를 지원하지 않습니다. MongoDB 및 DocumentDB도 이러한 종류의 기능을 지원하지 않습니다.
- Amazon S3에서 오픈 테이블 형식으로 저장된 데이터를 읽을 때는 Amazon S3에 있는 파일을 파티셔닝하는 방법이 더 이상 적합하지 않습니다. 오픈 테이블 형식을 사용하여 파티션에서 읽고 쓰려면 해당 형식에 대한 설명서를 참조하세요.
- DynamicFrame 메서드는 Amazon S3 프로젝션 푸시다운을 수행하지 않습니다. 조건자 필터를 통한 파일에서 모든 열을 읽습니다.
- AWS Glue에서 custom.jdbc 커넥터를 사용할 때 푸시다운 기능은 소스와 커넥터에 따라 달라집니다. 해당 커넥터 설명서를 검토하여 AWS Glue에서 푸시다운을 지원하는지 여부와 지원 방법을 확인하세요.

AWS Glue에 Auto Scaling 사용

AWS Glue 버전 3.0 이상에서 AWS Glue ETL 및 스트리밍 작업에 Auto Scaling을 사용할 수 있습니다.

Auto Scaling을 사용하면 다음과 같은 이점을 얻을 수 있습니다.

- AWS Glue는 각 스테이지의 병렬 처리나 작업 실행의 마이크로 배치에 따라 클러스터의 작업자를 자동으로 추가하고 제거합니다.
- AWS Glue ETL 작업에 대해 할당할 작업자 수를 실험하고 결정할 필요가 없어집니다.
- 최대 작업자 수를 선택하면 AWS Glue에서 워크로드에 적합한 크기의 리소스를 선택합니다.
- AWS Glue Studio의 작업 실행 세부 정보 페이지에서 CloudWatch 지표를 살펴보면 클러스터 크기가 작업 실행 중에 어떻게 변경되는지 확인할 수 있습니다.

AWS Glue ETL 및 스트리밍 작업에 Auto Scaling을 사용하면 AWS Glue 작업의 컴퓨팅 리소스를 온디맨드로 확장 및 축소할 수 있습니다. 온디맨드 확장은 처음에 작업 실행 시작 시 필요한 컴퓨팅 리소스만 할당하고 작업 중 수요에 따라 필요한 리소스를 프로비저닝하는 데 도움이 됩니다.

Auto Scaling은 작업 중 AWS Glue 작업 리소스의 동적 축소도 지원합니다. 작업 실행 중 Spark 애플리케이션에서 더 많은 실행기를 요청하면 클러스터에 더 많은 작업자가 추가됩니다. 실행기가 활성 계산 태스크 없이 유휴 상태인 경우 실행자와 해당 작업자가 제거됩니다.

Auto Scaling이 Spark 애플리케이션의 비용 및 사용률에 도움이 되는 일반적인 시나리오에는 Amazon S3에 있는 다수의 파일을 나열하거나 실행기가 비활성 상태인 동안 부하를 수행하는 Spark 드라이버, 오버프로비저닝으로 인해 몇 개의 실행기만으로 실행되는 Spark 스테이지, Spark 스테이지 전반에 걸친 데이터 기울이기 또는 균일하지 않은 계산 수요 등이 포함됩니다.

요구 사항

Auto Scaling은 AWS Glue 버전 3.0 이상에서만 사용 가능합니다. Auto Scaling을 사용하려면 [마이그레이션 가이드](#)에 따라 기존 작업을 AWS Glue 버전 3.0 이상으로 마이그레이션하거나 AWS Glue 버전 3.0 이상에서 새 작업을 생성합니다.

Auto Scaling은 G.1X, G.2X, G.4X, G.8X 또는 G.025X(스트리밍 작업만 해당) 작업자 유형을 포함하는 AWS Glue 작업에서 사용 가능합니다. 표준 DPU는 지원되지 않습니다.

AWS Glue Studio에서 Auto Scaling 사용

AWS Glue Studio의 Job details(작업 세부 정보) 탭에서 유형을 Spark 또는 Spark Streaming으로, Glue version(Glue 버전)을 **Glue 3.0** 또는 **Glue 4.0**으로 선택합니다. 그러면 작업자 유형(Worker type) 아래에 확인란이 표시됩니다.

- 작업자 수 자동 크기 조정(Automatically scale the number of workers) 옵션을 선택합니다.
- 최대 작업자 수(Maximum number of workers)를 설정하여 작업 실행에 판매할 수 있는 최대 작업자 수를 정의합니다.

Visual
Script
Job details
Runs
Data quality
Schedules

Version Control

Type
The type of ETL job. This is set automatically based on the types of data sources you have selected.

Spark

Glue version [Info](#)

Glue 4.0 - Supports spark 3.3, Scala 2, Python 3 ▼

Language

Python 3 ▼

Worker type
Set the type of predefined worker that is allowed when a job runs.

G 1X
(4vCPU and 16GB RAM) ▼

Automatically scale the number of workers
AWS Glue will optimize costs and resource usage by dynamically scaling the number of workers up and down throughout the job run. Requires Glue 3.0 or later.

Maximum number of workers
The number of workers you want AWS Glue to allocate to this job.

10

AWS CLI 또는 SDK를 사용하여 Auto Scaling 사용 설정

AWS CLI에서 Auto Scaling을 사용 설정하여 작업을 실행하려면, 다음 구성으로 `start-job-run`을 실행합니다.

```
{
  "JobName": "<your job name>",
  "Arguments": {
    "--enable-auto-scaling": "true"
  },
  "WorkerType": "G.2X", // G.1X and G.2X are allowed for Auto Scaling Jobs
}
```



```
"NumberOfWorkers": 20, // represents Maximum number of workers
...other job run configurations...
}
```

ETL 작업 실행이 완료되면 `get-job-run`을 호출하여 DPU 초 단위로 작업 실행의 실제 리소스 사용량을 확인할 수도 있습니다. 참고: 새 필드 `DPUSeconds`는 Auto Scaling이 사용 설정된 AWS Glue 3.0 이상의 배치 작업에만 표시됩니다. 이 필드는 스트리밍 작업에는 지원되지 않습니다.

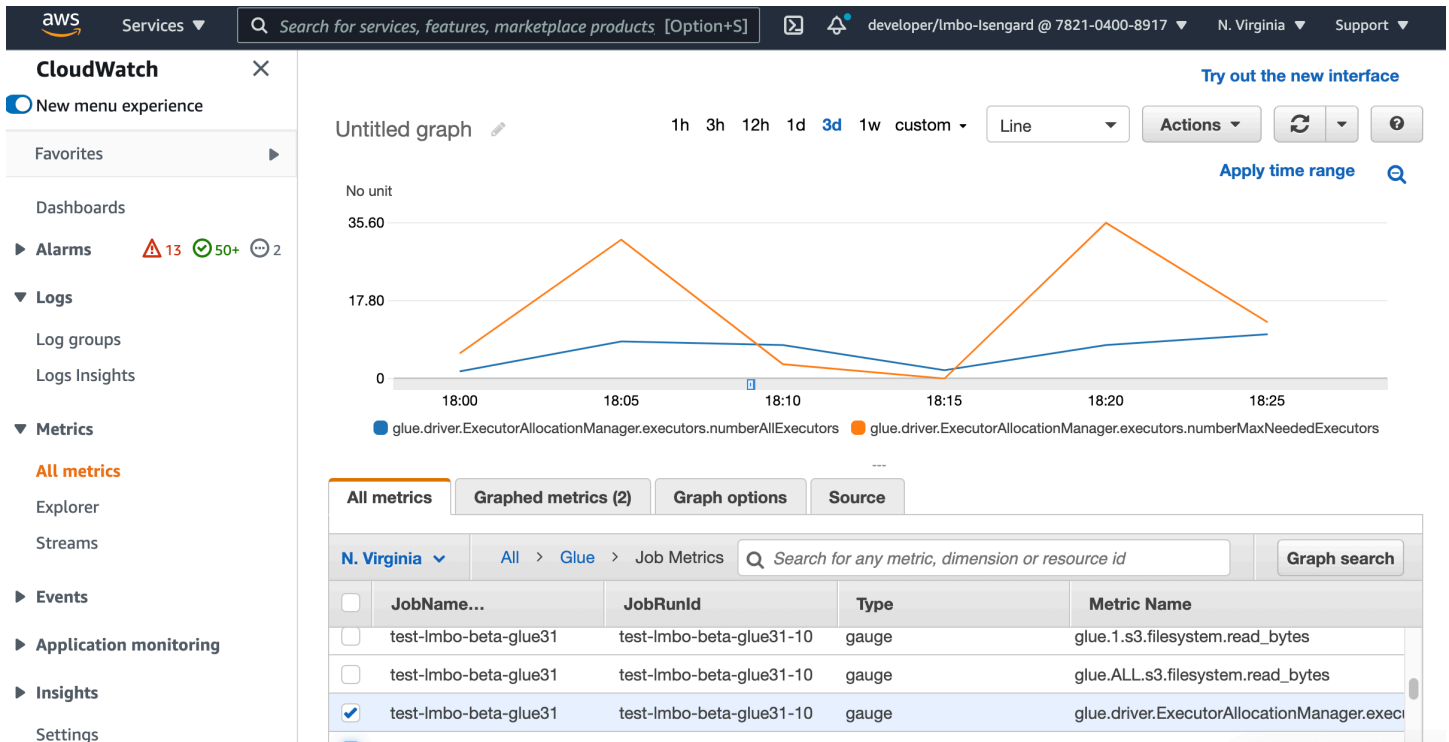
```
$ aws glue get-job-run --job-name your-job-name --run-id jr_xx --endpoint https://
glue.us-east-1.amazonaws.com --region us-east-1
{
  "JobRun": {
    ...
    "GlueVersion": "3.0",
    "DPUSeconds": 386.0
  }
}
```

동일한 구성의 [AWS Glue SDK](#)를 사용하여 Auto Scaling으로 작업 실행을 구성할 수도 있습니다.

Amazon CloudWatch 지표로 Auto Scaling 모니터링

CloudWatch 실행기 지표는 Auto Scaling을 사용하는 경우 AWS Glue 3.0 이상 작업에서 사용 가능합니다. 지표를 사용하여 Auto Scaling으로 사용 가능한 Spark 애플리케이션의 실행기 수요와 최적 사용량을 모니터링할 수 있습니다. 자세한 내용은 [Amazon CloudWatch 지표를 사용하여 AWS Glue 모니터링](#) 섹션을 참조하세요.

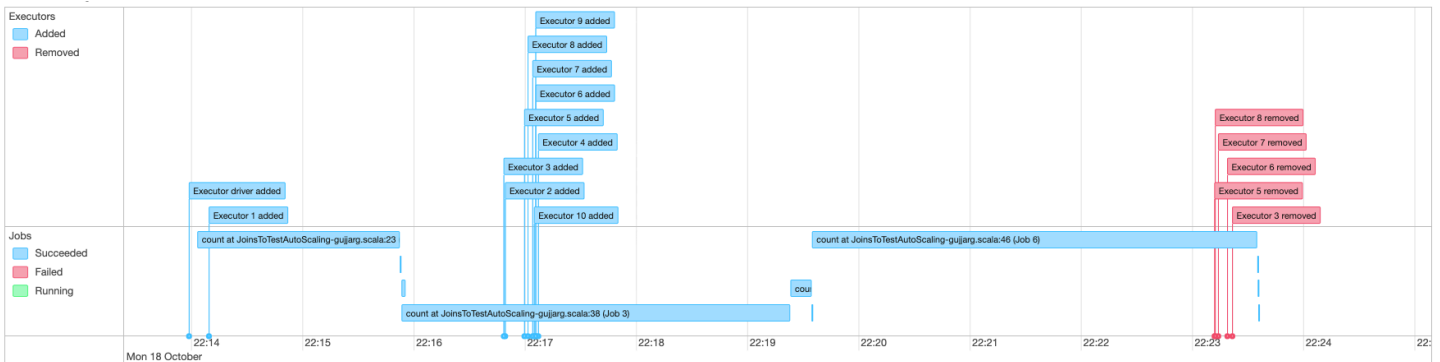
- `glue.driver.ExecutorAllocationManager.executors.numberAllExecutors`
- `glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors`



이러한 지표에 대한 자세한 내용은 [DPU 용량 계획 모니터링](#) 섹션을 참조하세요.

Spark UI로 Auto Scaling 모니터링

Auto Scaling이 사용 가능한 경우 AWS Glue 작업의 수요에 따른 동적 확장 및 축소를 사용하여 추가 및 제거되는 실행기를 Glue Spark UI로 모니터링할 수도 있습니다. 자세한 내용은 [AWS Glue 작업을 위한 Apache Spark 웹 UI 사용 설정](#) 단원을 참조하십시오.



Auto Scaling 작업 실행 DPU 사용량 모니터링

[AWS Glue Studio 작업 실행 보기](#)를 사용하여 Auto Scaling 작업의 DPU 사용량을 확인할 수 있습니다.

1. AWS Glue Studio 탐색 창에서 모니터링을 선택합니다. 모니터링 페이지가 표시됩니다.

- 아래로 스크롤하여 작업 실행 차트로 이동합니다.
- 원하는 작업 실행으로 이동하고 DPU 시간 열로 스크롤하여 특정 작업 실행의 사용량을 확인합니다.

제한 사항

AWS Glue 스트리밍 Auto Scaling은 현재 `ForEachBatch` 외부에서 생성된 정적 `DataFrame`과 스트리밍 `DataFrame`의 조인을 지원하지 않습니다. `ForEachBatch` 내부에서 생성된 정적 `DataFrame`은 예상대로 작동합니다.

제한된 실행을 통한 워크로드 분할

Spark 애플리케이션의 오류는 일반적으로 비효율적인 Spark 스크립트, 대규모 트랜스포메이션의 분산된 인메모리 실행 및 데이터 집합 이상으로 인해 발생합니다. 드라이버 또는 실행기의 메모리 부족 문제를 일으킬 수 있는 많은 이유가 있습니다(예: 데이터 왜곡, 너무 많은 객체 나열 또는 큰 데이터 셔플). 이러한 문제는 Spark로 엄청난 양의 백로그 데이터를 처리할 때 자주 발생합니다.

AWS Glue를 사용하면 OOM 문제를 해결하고 워크로드 분할을 통해 ETL 처리를 더 쉽게 할 수 있습니다. 워크로드 분할이 활성화되면 각 ETL 작업 실행은 처리되지 않은 데이터만 선택하며, 이 작업 실행으로 처리할 파일 수 또는 데이터 집합 크기에 대한 상한이 있습니다. 향후 작업 실행은 나머지 데이터를 처리합니다. 예를 들어 1,000개의 파일을 처리해야 하는 경우 파일 수를 500으로 설정하고 2개의 작업 실행으로 분리할 수 있습니다.

워크로드 분할은 Amazon S3 데이터 원본에 대해서만 지원됩니다.

워크로드 분할 사용

스크립트에서 수동으로 옵션을 설정하거나 카탈로그 테이블 속성을 추가하여 제한된 실행을 사용할 수 있습니다.

스크립트에서 제한된 실행으로 워크로드 분할을 사용하려면

- 데이터를 다시 처리하지 않으려면 새 작업 또는 기존 작업에서 작업 북마크를 사용합니다. 자세한 내용은 [작업 북마크를 사용하여 처리된 데이터 추적](#)을 참조하세요.
- 스크립트를 수정하고 AWS Glue `getSource` API의 추가 옵션에서 경계 제한을 설정합니다. `state` 요소를 저장하기 위해 작업 북마크에 대한 트랜스포메이션 컨텍스트도 설정해야 합니다. 예:

```
Python
```

```
glueContext.create_dynamic_frame.from_catalog(
  database = "database",
  table_name = "table_name",
  redshift_tmp_dir = "",
  transformation_ctx = "datasource0",
  additional_options = {
    "boundedFiles" : "500", # need to be string
    # "boundedSize" : "1000000000" unit is byte
  }
)
```

Scala

```
val datasource0 = glueContext.getCatalogSource(
  database = "database", tableName = "table_name", redshiftTmpDir = "",
  transformationContext = "datasource0",
  additionalOptions = JsonOptions(
    Map("boundedFiles" -> "500") // need to be string
    //"boundedSize" -> "1000000000" unit is byte
  )
).getDynamicFrame()
```

```
val connectionOptions = JsonOptions(
  Map("paths" -> List(baseLocation), "boundedFiles" -> "30")
)
val source = glueContext.getSource("s3", connectionOptions, "datasource0", "")
```

Data Catalog 테이블에서 제한된 실행으로 워크로드 분할을 사용하려면

1. Data Catalog에서 테이블 구조의 parameters 필드에 키-값 페어를 설정합니다. 자세한 내용은 [테이블 세부 정보 보기 및 편집](#)을 참조하세요.
2. 데이터 집합 크기 또는 처리되는 파일 수의 상한을 설정합니다.
 - boundedSize를 데이터 집합의 대상 크기(바이트)로 설정합니다. 테이블에서 대상 크기에 도달하면 작업 실행이 중지됩니다.
 - boundedFiles를 대상 파일 수로 설정합니다. 대상 파일 수를 처리한 후 작업 실행이 중지됩니다.

Note

하나의 경계만 지원되므로 `boundedSize` 또는 `boundedFiles` 중 하나만 설정해야 합니다.

작업을 자동으로 실행하도록 AWS Glue 트리거 설정

제한된 실행을 사용하면 작업을 자동으로 실행하고 순차적 실행에서 데이터를 증분적으로 로드하도록 AWS Glue 트리거를 설정할 수 있습니다. AWS Glue 콘솔로 이동하여 트리거를 생성하고 일정 시간을 설정하고 작업에 연결합니다. 그런 다음 자동으로 다음 작업 실행을 트리거하고 새 데이터 배치를 처리합니다.

또한 AWS Glue 워크플로로 여러 작업을 오케스트레이션하여 서로 다른 파티션의 데이터를 병렬로 처리할 수 있습니다. 자세한 내용은 [AWS Glue 트리거](#) 및 [AWS Glue 워크플로](#)를 참조하세요.

사용 사례 및 옵션에 대한 자세한 내용은 [AWS Glue에서 워크로드 분할로 Spark 애플리케이션 최적화](#)를 참조하세요.

AWS Glue의 알려진 문제

다음은 AWS Glue에서 알려진 문제입니다.

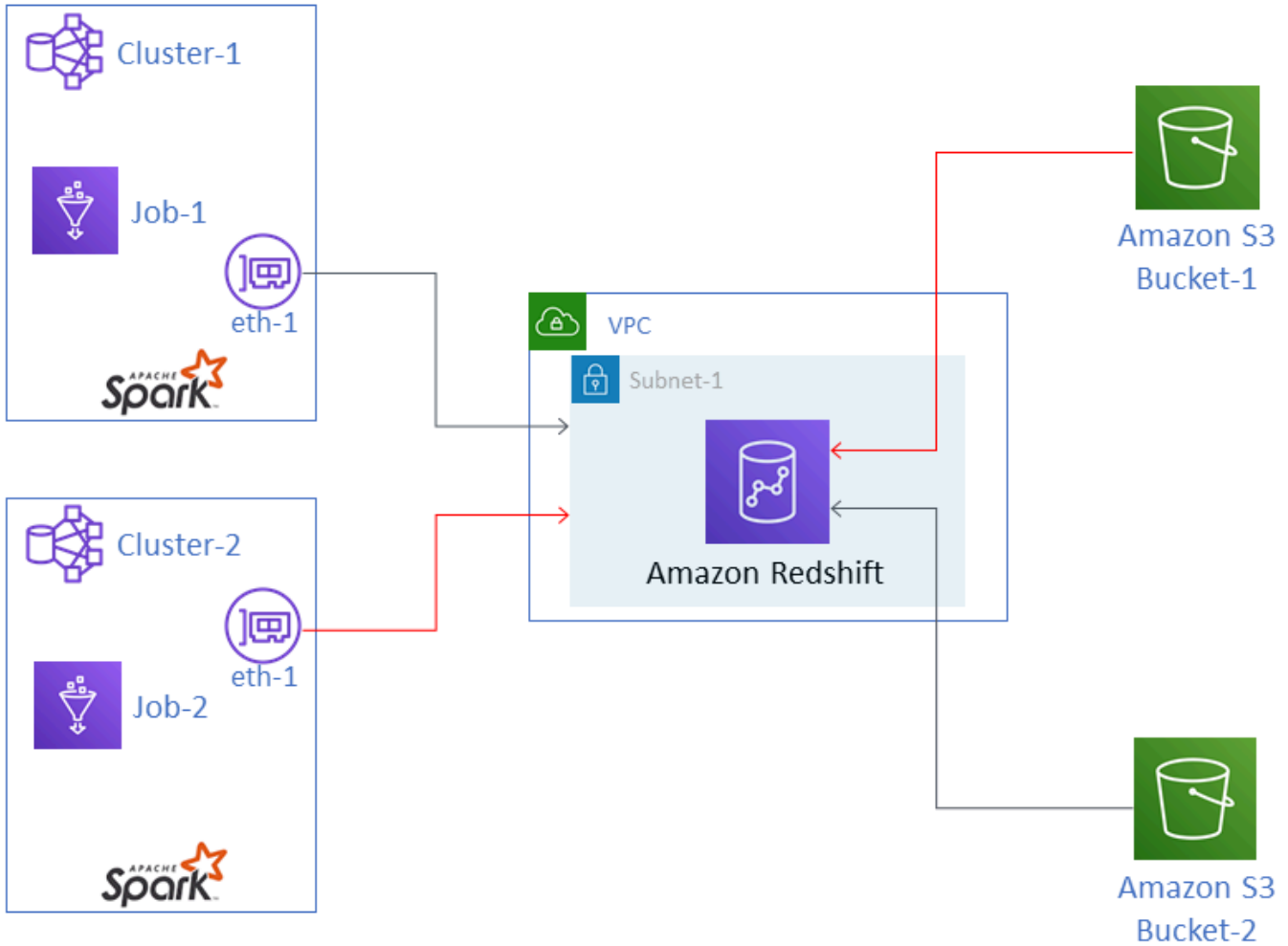
주제

- [작업 간 데이터 액세스 방지](#)

작업 간 데이터 액세스 방지

각각 별도의 AWS Glue Spark 클러스터에서 실행되는 AWS Glue Spark 작업 2개가 단일 AWS 계정에 있다고 가정하겠습니다. 두 작업은 AWS Glue 연결을 사용해 동일한 가상 프라이빗 클라우드(VPC)의 리소스에 액세스합니다. 이때 한쪽 클러스터에서 실행되는 작업은 다른 클러스터에서 실행되는 작업의 데이터에 액세스할 수 있어야 합니다.

다음 다이어그램은 이러한 상황을 예로 든 그림입니다.



위 다이어그램을 보면 AWS Glue Job-1은 Cluster-1에서, 그리고 Job-2는 Cluster-2에서 실행 중입니다. 두 작업 모두 VPC의 Subnet-1에 속하면서 동일한 Amazon Redshift 인스턴스와 통신하고 있습니다. 이때 Subnet-1은 퍼블릭 서브넷 또는 프라이빗 서브넷이 될 수 있습니다.

Job-1은 Amazon Simple Storage Service(Amazon S3) Bucket-1의 데이터를 변환하여 Amazon Redshift에 작성하고 있습니다. Job-2는 Bucket-2의 데이터를 사용해 동일하게 실행하고 있습니다. Job-1은 AWS Identity and Access Management(IAM) 역할인 Role-1(그림에 보이지 않음)을 사용해 Bucket-1에 대한 액세스 권한을 얻습니다. Job-2는 Role-2(그림에 보이지 않음)를 사용해 Bucket-2에 대한 액세스 권한을 얻습니다.

두 작업은 서로 클러스터와 통신할 수 있는 네트워크 경로가 있기 때문에 각 데이터에 액세스할 수 있습니다. 예를 들어 Job-2는 Bucket-1의 데이터에 액세스할 수 있습니다. 위 다이어그램에서는 빨간색으로 표시된 선이 네트워크 경로입니다.

이러한 상황을 방지하려면 Job-1과 Job-2에 서로 다른 보안 구성을 연결하는 것이 좋습니다. 보안 구성을 연결하면 AWS Glue에서 생성되는 인증서를 통해 작업 간 데이터 액세스가 차단됩니다. 보안 구성은 더미 구성이 될 수 있습니다. 이 말은 Amazon S3 데이터, Amazon CloudWatch 데이터 또는 작업 북마크를 암호화하지 않고 보안 구성을 생성할 수 있다는 것을 의미합니다. 세 가지 암호화 옵션 모두 비활성화가 가능합니다.

보안 구성에 대한 자세한 내용은 [the section called “AWS Glue에서 작성한 데이터 암호화” 단원을 참조하십시오.](#)

보안 구성을 작업에 연결하려면

1. <https://console.aws.amazon.com/glue/>에서 AWS Glue 콘솔을 엽니다.
2. 해당 작업의 Configure the job properties(작업 속성 구성) 페이지에서 Security configuration, script libraries, and job parameters(보안 구성, 스크립트 라이브러리 및 작업 파라미터) 섹션을 확장합니다.
3. 목록에서 보안 구성을 선택합니다.

AWS Glue에 대한 문서 기록

변경 사항	설명	날짜
AWS Glue 사용 프로필 지원	관리자는 개발자, 테스터, 제품 팀 등 계정 내 다양한 사용자 계층의 AWS Glue 사용 프로필을 만들 수 있습니다. 이러한 유연성 덕분에 관리자는 각 사용자 클래스에 대해 서로 다른 사용량 및 비용 관리를 적용할 수 있습니다. 자세한 내용은 AWS Glue 사용 프로필 설정을 참조 하십시오.	2024년 6월 18일
Spark용 세일즈포스 커넥터 지원 AWS Glue	Salesforce의 새 AWS Glue 커넥터에 대한 정보가 추가되었습니다. 이 기능을 사용하면 Spark에서 4.0 AWS Glue 이상 버전에서 Salesforce에서 AWS Glue 읽고 쓸 수 있습니다. 자세한 내용은 Salesforce에 연결 을 참조하십시오.	2024년 5월 22일
Amazon Q 데이터 통합 AWS Glue (GA)	Amazon Q 데이터 AWS Glue 통합은 데이터 엔지니어와 ETL 개발자가 자연어를 사용하여 데이터 통합 작업을 구축할 수 있게 AWS Glue 해주는 새로운 생성 AI 기능입니다. 엔지니어와 개발자는 Q에 작업을 작성하고, 문제를 해결하고, 데이터 통합에 대한 AWS Glue 질문에 답하도록 요청할 수 있습니다. 자세한 내용은 AWS Glue의 Amazon	2024년 4월 30일

[Q 데이터 통합](#)을 참조하세요. 이 기능에는 `AwsGlueSessionUserRestrictedPolicy`, `AwsGlueSessionUserRestrictedNotebookServiceRole`, `AwsGlueSessionUserRestrictedServiceRole` AWS 관리형 정책에 대한 업데이트가 포함됩니다. 자세한 내용은 [AWS 관리형 정책AWS Glue 업데이트](#)를 참조하십시오.

[Amazon Q 데이터 통합 AWS Glue \(미리 보기\)](#)

Amazon Q 데이터 AWS Glue 통합은 데이터 엔지니어와 ETL 개발자가 자연어를 사용하여 데이터 통합 작업을 구축할 수 있게 AWS Glue 해주는 새로운 생성 AI 기능입니다. 엔지니어와 개발자는 Q에 작업을 작성하고, 문제를 해결하고, 데이터 통합에 대한 AWS Glue 질문에 답하도록 요청할 수 있습니다. 자세한 내용은 [AWS Glue의 Amazon Q 데이터 통합](#)을 참조하세요. 이 기능에는 `AwsGlueSessionUserRestrictedNotebookPolicy` AWS 관리형 정책에 대한 업데이트가 포함됩니다. 자세한 내용은 [AWS 관리형 정책AWS Glue 업데이트](#)를 참조하십시오.

2024년 1월 30일

[AWS Glue 스트리밍에 대한 설명서 업데이트](#)

AWS Glue 스트리밍을 위한 신규 및 재구성된 콘텐츠가 포함된 새 챕터가 추가되었습니다. 이 콘텐츠는 스트리밍의 작동 방식 AWS Glue, 실시간 데이터 처리의 특징, 스트리밍 작업을 모니터링하는 방법을 설명합니다. 자세한 내용은 [AWS Glue 스트리밍](#)을 참조하세요.

2023년 12월 27일

[세분화된 민감한 데이터 탐지 사용 지원](#)

민감한 데이터 탐지 변환은 사용자가 정의하거나 AWS Glue에서 사전 정의한 엔터티를 탐지, 마스크 또는 제거하는 기능을 제공합니다. 또한 세분화된 액션을 통해 엔터티별로 특정 액션을 적용할 수 있습니다. 자세한 내용은 [세분화된 민감한 데이터 탐지 사용](#)을 참조하세요.

2023년 11월 26일

[AWS Glue 옵저버빌리티 메트릭을 사용한 작업 모니터링 지원](#)

AWS Glue 관찰성 메트릭을 사용하면 Apache Spark의 AWS Glue 내부에서 일어나는 일에 대한 통찰력을 얻어 문제의 분류 및 분석을 개선할 수 있습니다. 자세한 내용은 [AWS Glue 관찰성 지표를 사용한 모니터링](#)을 참조하세요.

2023년 11월 26일

[데이터 품질의 이상 탐지 지원
AWS Glue](#)

AWS Glue Data Quality 이상 탐지는 시간 경과에 따른 데이터 통계에 기계 학습(ML) 알고리즘을 적용하여 규칙으로 탐지하기 어려운 비정상적인 패턴과 숨겨진 데이터 품질 문제를 탐지합니다. 자세한 내용은 [AWS Glue Data Quality의 이상 탐지](#)를 참조하세요.

2023년 11월 26일

[기본 Spark UI 로깅 동작으로
업데이트](#)

Spark UI 로그를 생성하는 Spark 작업은 이제 콘솔에서 Spark UI를 지원하기 위해 다른 파일 이름 패턴으로 작성됩니다. AWS Glue 이렇게 해도 로그 동작은 변경되지 않습니다. CloudWatch 작업 구성을 업데이트하여 기존 동작으로 되돌릴 수 있습니다. 자세한 내용은 [Apache Spark 웹 UI를 사용하여 작업 모니터링](#)을 참조하세요.

2023년 11월 17일

[Spark의 AWS Glue 새 데이터 소스 지원](#)

Amazon OpenSearch Service, Azure SQL, NoSQL용 Azure Cosmos, SAP HANA 테라데이터 밴티지 및 버티카와의 연결이 이제 내에서 기본적으로 지원됩니다. AWS Glue 또한 이제 MongoDB와 함께 이러한 데이터 소스에 대한 연결을 AWS Glue Studio 비주얼 편집기에서 사용할 수 있습니다. 자세한 내용은 Spark 지원에 [대한 자세한 내용은 Spark의 AWS Glue ETL 연결 유형 및 옵션](#) 및 Studio 비주얼 편집기에서의 사용에 대한 자세한 내용은 [AWS Glue 연결 추가를](#) 참조하십시오. AWS Glue AWS Glue

2023년 11월 17일

[열 통계 생성 지원](#)

추가 데이터 파이프라인을 설정하지 않고도 Parquet, ORC, JSON, ION, CSV 및 XML과 같은 데이터 형식의 AWS Glue Data Catalog 테이블에 대한 열 수준 통계를 계산할 수 있습니다. 자세한 내용은 [열 통계 작업을](#) 참조하세요.

2023년 11월 16일

<u>Iceberg 테이블의 데이터 압축 지원</u>	Amazon Athena, Amazon EMR과 같은 AWS 분석 서비스와 AWS Glue ETL 작업의 읽기 성능을 향상시키기 위해 데이터 카탈로그는 데이터 카탈로그의 Iceberg 테이블에 대한 관리형 압축 (작은 Amazon S3 객체를 큰 객체로 압축하는 프로세스) 을 제공합니다. 자세한 내용은 <u>Iceberg 테이블 최적화</u> 를 참조하세요.	2023년 11월 13일
<u>작업 실행 대기 동작 업데이트</u>	표준 Spark 및 Python 셸 작업 실행이 이제 특정 상황에서 WAITING으로 즉시 전환되지 않고 FAILED로 전환됩니다. 자세한 내용은 <u>AWS Glue 작업 실행 상태</u> 를 참조하세요.	2023년 11월 8일
<u>AWS Glue Studio 사용자 설명서가 AWS Glue 개발자 설명서로 통합됨</u>	AWS Glue Studio 사용 설명서가 개발자 안내서로 이동하여 AWS Glue 콘솔 및 AWS Glue Studio 프로그래밍 액세스에 대한 AWS Glue Studio의 단일 통합 사용 설명서가 생성되었습니다.	2023년 10월 25일
<u>관리형 정책 업데이트 AWSGlueServiceNotebookRole AWS</u>	AWSGlueServiceNotebookRole AWS 관리형 정책의 사소한 업데이트에 대한 정보가 추가되었습니다. 자세한 내용은 <u>AWS 관리형 정책 AWS Glue 업데이트</u> 를 참조하십시오.	2023년 10월 9일

[AWS Glue Studio에서 다섯 가지 새로운 기본 제공 변환 지원](#)

AWS Glue Studio에서는 5가지 새로운 기본 제공 변환(레코드 일치, null 행 제거, JSON 열 구문 분석, JSON 경로 추출, 정규식 추출기)을 지원합니다. 자세한 내용은 [AWS Glue 관리형 데이터 변환 노드 편집을](#) 참조하십시오.

2023년 8월 11일

[AWSGlueServiceRole AWS 관리형 정책 업데이트](#)

AWSGlueServiceRole AWS 관리형 정책의 사소한 업데이트에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS 관리형 정책 AWS Glue 업데이트를](#) 참조하십시오.

2023년 8월 4일

[Apache Hudi 테이블 크롤링에 대한 지원](#)

Amazon S3 버킷에서 Hudi 테이블을 AWS Glue 크롤링하는 데 사용하고 Hudi 테이블에 등록하는 방법에 대한 정보가 추가되었습니다. AWS Glue Data Catalog 자세한 내용은 [크롤링할 수 있는 데이터 스토어는 무엇인가요?](#) 및 [Crawler properties](#)를 참조하세요.

2023년 7월 21일

[관리형 정책 업데이트 AWSGlueConsoleFullAccess AWS](#)

AWSGlueConsoleFullAccess AWS 관리형 정책의 사소한 업데이트에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS 관리형 정책 AWS Glue 업데이트를](#) 참조하십시오.

2023년 7월 14일

[Apache Iceberg 테이블 크롤링에 대한 지원](#)

Amazon S3 버킷에서 Iceberg 테이블을 AWS Glue 크롤링하는 데 사용하고 Iceberg 테이블을 에 등록하는 방법에 대한 정보가 추가되었습니다. AWS Glue Data Catalog 자세한 내용은 [크롤링할 수 있는 데이터 스토어는 무엇인가요?](#) 및 [Crawler properties](#)를 참조하세요.

2023년 7월 7일

[AWS Glue With Ray 지원](#)

AWS Glue 작업을 뒷받침할 수 있는 새로운 엔진인 AWS Glue with Ray에 대한 정보가 추가되었습니다. Spark AWS Glue 콘텐츠로 기존 콘텐츠를 재구성하여 명확하게 정리했습니다.

2023년 5월 30일

[AWS Glue 데이터 품질 지원 \(GA\)](#)

AWS Glue 이제 데이터 품질을 일반적으로 사용할 수 있습니다. AWS Glue 데이터 품질은 데이터 품질을 평가하고 모니터링하는 데 도움이 됩니다. 데이터 카탈로그에서 AWS Glue 데이터 품질을 사용하는 방법에 대한 자세한 내용은 [AWS Glue 데이터 품질을 참조하십시오](#). 의 AWS Glue 데이터 품질에 대해 AWS Glue Studio 알아보려면 [사용하여 데이터 품질 평가를 참조하십시오](#) AWS Glue Studio.

2023년 5월 24일

[Apache Spark 작업에 대한 대 규모 작업자 유형 지원](#)

이제 Apache Spark 작업에 대한 G.4X 및 G.8X 작업자 유형을 사용할 수 있습니다. 이러한 작업자 유형은 워크로드에 가장 까다로운 변환, 집계, 조인 및 쿼리가 포함된 작업에 적합합니다. 자세한 내용은 [AWS Glue의 작업 추가](#)를 참조하십시오.

2023년 5월 8일

[테이블 크롤링 시 파티션 인덱스 생성에 대한 지원](#)

크롤러가 감지하는 테이블의 파티션 인덱스 생성을 크롤러에서 지원하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [파티션 인덱스 크롤러 구성 옵션 설정](#)을 참조하세요.

2023년 4월 24일

[리소스 사용량 지표에 대한 지 원](#)

CloudWatchAmazon에서 서비스의 리소스 사용량을 확인하고 경보를 구성하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue 리소스 모니터링](#)을 참조하세요.

2023년 4월 7일

[AWSGlueConsoleFullAccess AWS 관리형 정책 업데이트](#)

AWSGlueConsoleFullAccess AWS 관리형 정책의 사소한 업데이트에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS 관리형 정책 AWS Glue 업데이트](#)를 참조하십시오.

2023년 3월 28일

[AWS Glue AWS SDK와 함께 사용하기 위한 지침이 예제와 함께 추가되었습니다.](#)

AWS Glue 개발자 안내서에는 AWS Glue AWS SDK와 함께 사용하는 데 도움이 되는 정보를 제공하는 두 개의 새로운 섹션이 있습니다. 자세한 내용은 [AWS Glue SDK와 함께 사용 및 AWS SDK AWS Glue 사용에 AWS 대한 코드 예제를 참조](#) 하십시오.

2023년 2월 23일

[다음을 사용하여 IAM 설명서를 업데이트하십시오. AWS Glue](#)

IAM 사용에 대한 정보가 재구성되고 추가되었습니다. AWS Glue 자세한 내용은 [AWS Glue의 자격 증명 및 액세스 관리](#)를 참조하세요.

2023년 2월 15일

[AWS Glue 버전 4.0에서 ETL 작업 스트리밍 실행 지원](#)

Glue 버전 4.0에서 ETL 작업 스트리밍을 실행하는 데 대한 지원과 Kafka 클러스터 또는 Apache Kafka 클러스터용 Amazon 관리형 스트리밍 및 Amazon Kinesis Data Streams에 연결하기 위한 새로운 옵션에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue에서 ETL 작업 스트리밍 추가 및 AWS Glue의 ETL에 대한 연결 유형 및 옵션](#)을 참조하세요.

2023년 2월 8일

[MongoDB Atlas 데이터 소스 크롤링 지원](#)

MongoDB Atlas 데이터 AWS Glue 원본을 크롤링하는 데 사용하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [어떤 데이터 스토어를 크롤링할 수 있나요?](#), [MongoDB 및 MongoDB Atlas 연결 속성](#), [MongoDB 또는 MongoDB Atlas 연결 사용](#)을 참조하세요.

2023년 2월 6일

[기본 Delta Lake 커넥터를 사용하여 Delta Lake 테이블 크롤링 지원](#)

기본 Delta Lake 커넥터를 사용하여 Delta Lake 테이블을 AWS Glue 크롤링하는 데 사용하는 방법에 대한 정보가 추가되었습니다. 이 기능을 사용하면 AWS 쿼리 엔진을 사용하여 Delta 트랜잭션 로그를 직접 쿼리하고 시간 이동 및 ACID 보장과 같은 기능을 사용할 수 있으며, Amazon S3 트랜잭션 파일의 Delta Lake 메타데이터를 데이터 카탈로그에 동기화하여 Lake Formation에서 쿼리에 대한 열 권한을 활성화할 수 있습니다. 자세한 내용은 [Delta Lake 데이터 스토어에 대한 구성 옵션을 지정하는 방법](#), [Delta Lake 테이블 쿼리하기](#)를 참조하세요.

2022년 12월 15일

[AWS Glue 데이터 품질 지원 \(미리 보기\)](#)

이제 AWS Glue 데이터 품질 (미리 보기)에 대한 지원이 제공됩니다. AWS Glue 데이터 품질을 사용하면 AWS Glue 3.0을 사용할 때 데이터 품질을 평가하고 모니터링할 수 있습니다. 데이터 카탈로그에서 AWS Glue 데이터 품질을 사용하는 방법에 대한 자세한 내용은 [AWS Glue 데이터 품질 \(미리 보기\)](#)을 참조하십시오. 의 AWS Glue 데이터 품질에 대해 AWS Glue Studio알아보려면 [사용하여 데이터 품질 평가를](#) 참조하십시오 AWS Glue Studio.

2022년 11월 30일

[새로운 기능과 향상된 성능을 갖춘 새로운 Amazon Redshift Spark 커넥터 지원](#)

이제 데이터 수집 및 변환 파이프라인의 일부로 Amazon Redshift에서 데이터를 읽고 쓰는 Apache Spark 애플리케이션을 빌드하기 위해 AWS Glue ETL 작업에 사용할 수 있는 새로운 JDBC 드라이버가 포함된 Amazon Redshift Spark 커넥터를 지원할 수 있습니다. 자세한 내용은 [Amazon Redshift 간 데이터 이동](#)을 참조하세요.

2022년 11월 29일

[AWS Glue 버전 4.0을 지원합니다.](#)

AWS Glue 버전 4.0에 대한 지원 정보가 추가되었습니다. 기능으로는 Apache Hudi, Delta Lake 및 Apache Iceberg에서 개방형 데이터 레이크 프레임워크를 기본적으로 지원하고, Amazon S3를 사용하여 셔플링 및 탄력적인 스토리지 용량을 지원하기 위해 Amazon S3 기반 클라우드 셔플 스토리지 플러그인(Apache Spark 플러그인)을 기본적으로 지원합니다. 자세한 내용은 [AWS Glue 릴리스 정보](#)와 [AWS Glue 버전 4.0으로 AWS Glue 작업 마이그레이션](#)을 참조하세요.

2022년 11월 28일

[AWS Glue Studio에서는 이제 사용자 지정 시각적 변환을 제 공합니다.](#)

고객은 사용자 지정 시각적 변환을 통해 팀 간에 비즈니스별 ETL 로직을 정의, 재사용, 공유 할 수 있습니다. 자세한 내용은 [사용자 지정 시각적 변환](#)을 참조하세요.

2022년 11월 28일

[AWS Glue 크롤러를 사용하여 JDBC 데이터 스토어용 메타데이터 게시 지원](#)

이제 AWS Glue 크롤러를 사용하여 주석 및 원시 유형과 같은 메타데이터를 JDBC 데이터 스토어용 데이터 카탈로그에 게시할 수 있습니다. [자세한 내용은 데이터 카탈로그 테이블에 크롤러별로 설정된 매개 변수, Crawler속성 및 구조를 참조하십시오. JdbcTarget](#)

2022년 11월 18일

[Snowflake 데이터 스토어 크롤링 지원](#)

이제 AWS Glue를 사용하여 Snowflake 테이블 및 뷰를 크롤링하고 메타데이터를 데이터 카탈로그에 테이블 항목으로 게시할 수 있습니다. Amazon S3의 Snowflake 외부 테이블의 경우 크롤러는 Amazon S3 위치 및 외부 테이블의 파일 형식 유형을 크롤링하고 테이블 파라미터로 채웁니다. 자세한 내용은 [어떤 데이터 스토어를 크롤할 수 있나요?](#), [AWS Glue 연결 속성 및 크롤러가 데이터 카탈로그 테이블에 설정한 파라미터](#)를 참조하세요.

2022년 11월 18일

[Spark 애플리케이션의 향상된 셔플 관리 지원](#)

이제 Apache Spark의 새로운 클라우드 셔플 스토리지 플러그인을 지원할 수 있습니다. 자세한 내용은 [Amazon S3의 AWS Glue Spark 셔플 플러그인](#) 및 [Apache Spark용 클라우드 셔플 스토리지 플러그인](#)을 참조하세요.

2022년 11월 15일

[크롤링 Amazon S3 이벤트 알림을 가속화할 때 데이터 카탈로그 대상에 대한 지원을 추가했습니다.](#)

Amazon S3 대상에 대한 기존 지원과 더불어 이제 Amazon S3 이벤트 알림을 사용한 데이터 카탈로그 대상의 크롤링 가속화에 대한 지원이 제공됩니다. 자세한 내용은 [Amazon S3 이벤트 알림을 사용하여 크롤링 가속화](#)를 참조하세요.

2022년 10월 13일

[크롤러가 생성할 수 있는 최대 테이블 수 지정 가능](#)

이제 크롤러가 생성할 수 있는 최대 테이블 수를 지정할 수 있습니다. 자세한 내용을 알아보려면 [크롤러가 생성할 수 있는 최대 테이블 수를 지정하는 방법](#)을 참조하세요.

2022년 9월 6일

[AWS Glue에서 Python 셸 작업의 Python 3.9 지원](#)

AWS Glue의 Python 셸 작업에서 Python 3.9와 호환되는 스크립트 실행과 사용할 사전 패키징된 라이브러리 세트 선택에 대해 지원이 제공됩니다. 자세한 내용은 [AWS Glue의 Python 셸 작업](#)을 참조하세요.

2022년 8월 11일

[Support는 여분의 용량으로 긴급하지 않거나 시간에 민감하지 않은 AWS Glue 작업을 실행할 수 있도록 지원](#)

이제 사전 프로덕션 작업, 테스트 및 일회성 데이터 로드와 같이 긴급하지 않은 작업에 대한 유연한 작업 실행 구성 지원이 제공됩니다. 자세한 내용은 [AWS Glue의 작업 추가](#)를 참조하십시오.

2022년 8월 9일

[스트리밍 작업에 대한 새 작업자 유형 지원](#)

이제 볼륨이 낮은 스트리밍 작업에 대한 G.025X 작업자 유형의 사용이 지원됩니다. 자세한 내용은 [AWS Glue의 작업 추가](#)를 참조하십시오.

2022년 7월 14일

[AWS Glue 연결에서의 Kafka SASL 사용 지원](#)

이제 AWS Glue 연결에서 Kafka SASL 사용이 지원됩니다. 자세한 내용은 [클라이언트 인증을 위한 AWS Glue Kafka 연결 속성](#)을 참조하세요.

2022년 7월 5일

Protobuf 스키마에 대한 Apache Kafka 커넥터 지원	Protobuf 스키마에 대한 Apache Kafka 커넥터가 이제 지원됩니다. 자세한 내용은 AWS Glue Schema Registry 를 참조하세요.	2022년 6월 9일
AWS Glue 작업에서 Auto Scaling 지원(GA)	AWS Glue 버전 3.0 작업에 Auto Scaling을 사용하여 컴퓨팅 리소스 크기를 동적으로 조정하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue에서 Auto Scaling 사용 을 참조하세요.	2022년 4월 14일
AWS Glue 작업 스크립트 AWS Glue 개발 및 테스트에 대한 설명서 업데이트	AWS Glue에 대한 사용 가능한 개발 및 테스트 방법에 대한 정보가 재구성 및 추가되었으며, Docker를 통해 개발하기 위한 지침을 포함합니다. 자세한 내용은 AWS Glue 작업 스크립트 개발 및 테스트 를 참조하세요.	2022년 3월 14일
AWS Glue Schema Registry의 지원되는 데이터 형식으로 프로토콜 버퍼(Protobuf) 추가	AVRO 및 JSON 외에 지원되는 데이터 형식으로 Protobuf에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue Schema Registry 를 참조하세요.	2022년 2월 25일
Delta Lake 테이블 크롤링에 대한 지원	Delta Lake 테이블을 AWS Glue 크롤링하는 데 사용하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 Delta Lake 데이터 스토어에 대한 구성 옵션을 지정하는 방법 을 참조하세요.	2022년 2월 24일

AWS Glue 직무 인사이트 지원	AWS Glue Job Insights를 사용하여 작업 디버깅 및 작업 최적화를 간소화하는 방법에 대한 정보가 추가되었습니다. AWS Glue 자세한 내용은 Monitoring with AWS Glue job insights (작업 인사이트를 사용한 모니터링)를 참조하세요.	2022년 2월 8일
VPC 엔드포인트를 사용한 Amazon S3 기반 데이터 카탈로그 테이블의 크롤링 지원	Amazon S3 데이터 스토어 외에도 Amazon S3 기반 데이터 카탈로그 테이블이 보안, 감사 또는 제어를 위해 Amazon Virtual Private Cloud 환경(Amazon VPC)에 의해서만 액세스되도록 구성할 수 있습니다. 자세한 내용은 Crawling an Amazon S3 Data Store or Amazon S3 backed Data Catalog tables using a VPC Endpoint (VPC 엔드포인트를 사용하여 Amazon S3 데이터 스토어 또는 Amazon S3 기반 데이터 카탈로그 테이블 크롤링)를 참조하세요.	2022년 2월 3일
Lake Formation 관리형 테이블 지원	ACID 트랜잭션, 자동 데이터 압축, 시간 이동 쿼리를 지원하는 Lake Formation 관리형 테이블에 대한 AWS Glue 지원 정보를 추가했습니다. 자세한 내용은 AWS Glue API 및 AWS Lake Formation 개발자 가이드 를 참조하세요.	2021년 11월 30일

대화형 세션 및 노트북에 대한 새로운 AWS 관리형 정책이 추가되었습니다.	IAM에 대한 새로운 관리형 정책은 대화형 세션 및 AWS Glue 노트북과 함께 사용할 때 보안을 강화했습니다. 자세한 내용은 AWS Glue에 대한 AWS 관리형 정책 을 참조하세요.	2021년 11월 30일
이제 스트리밍 작업에서 Glue Schema Registry가 지원됨	Glue Schema Registry의 일부인 테이블에 액세스하는 스트리밍 작업을 생성할 수 있습니다. 자세한 내용은 AWS Glue Schema Registry 및 AWS Glue에서 스트리밍 ETL 작업 추가 를 참조하세요.	2021년 11월 15일
새로운 기계 학습 기능 지원	중분 일치 및 일치 점수를 포함하여 일치 항목 찾기 기계 학습 변환의 새로운 기능에 대한 정보를 추가했습니다. 자세한 내용은 중분 일치 항목 찾기 및 일치 신뢰도 점수를 사용하여 일치 항목의 품질 추정을 참조 하세요.	2021년 10월 31일
(프라이빗 평가판) AWS Glue 유연한 작업 지원	시작 및 완료 시간이 다를 수 있는 시간에 민감하지 않은 작업에 해당되는 유연한 실행 클래스를 사용한 AWS Glue Spark 작업 구성에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue의 작업 추가 를 참조하십시오.	2021년 10월 29일

<u>Amazon S3 이벤트 알림을 사용하여 크롤링 가속화 지원</u>	Amazon S3 이벤트 알림을 사용하여 크롤링을 가속화하는 방법에 대한 정보를 추가했습니다. 자세한 내용은 <u>Amazon S3 이벤트 알림을 사용하여 크롤링 가속화</u> 를 참조하세요.	2021년 10월 15일
<u>액세스 제어 및 VPC와 관련된 추가 보안 구성 옵션</u>	AWS Glue에서 새로운 액세스 제어 권한을 구성하는 방법 및 VPC 구성에 대한 정보를 추가했습니다. 자세한 내용은 <u>AWS 태그 인 AWS Glue, 조건 키 또는 컨텍스트 키를 사용하여 설정을 제어하는 ID 기반 정책 (IAM 정책), VPC를 통과하도록 모든 AWS 호출 구성을 참조하십시오</u> .	2021년 10월 13일
<u>VPC 엔드포인트 정책 지원</u>	AWS Glue의 Virtual Private Cloud(VPC) 엔드포인트 정책 지원에 대한 정보를 추가했습니다. 자세한 내용은 <u>AWS Glue 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)</u> 를 참조하세요.	2021년 10월 11일
<u>이제 중국에서 Glue Studio 사용 가능</u>	이제 중국 베이징 및 닝샤 리전에서 AWS Glue Studio를 사용할 수 있습니다.	2021년 10월 11일

[AWS Glue Studio에서는 대화형 작업 편집을 위한 노트북 작성 기능을 제공합니다.](#)

노트북을 사용하여 코드를 작성 및 실행하고, 결과를 시각화하고, 인사이트를 공유할 수 있습니다. 일반적으로 데이터 과학자는 실험 및 데이터 탐색 작업에 노트북을 사용합니다. 자세한 내용은 [노트북 사용](#)을 참조하세요.

2021년 10월 1일

[이제 스트리밍 소스에 직접 액세스 가능](#)

시각적 편집기에서 ETL 작업에 데이터 원본을 추가할 때 데이터 카탈로그 데이터베이스와 테이블을 사용하지 않고 데이터 스트림에 액세스하기 위한 정보를 제공할 수 있습니다.

2021년 9월 30일

[AWS Glue 버전 지원 정책이 문서화됨](#)

AWS Glue 버전 지원 정책 및 특정 AWS Glue 버전의 수명 종료 단계에 대한 정보를 추가했습니다. 자세한 내용은 [AWS Glue 버전 지원 정책](#)을 참조하세요.

2021년 9월 24일

[이제 데이터 미리 보기에서 사용자 지정 커넥터 사용 가능](#)

사용자 지정 커넥터를 사용하여 데이터 원본 노드를 편집할 때 데이터 미리 보기(Data preview) 탭을 선택하여 데이터 집합을 미리 볼 수 있습니다. 자세한 내용은 [사용자 지정 커넥터](#)를 참조하세요.

2021년 9월 24일

[AWS Glue 대화형 세션 지원
\(비공개 미리 보기\)](#)

(비공개 미리 보기) 모든 Jupyter Notebook에서 AWS Glue 대화형 세션을 사용하여 클라우드에서 Spark 워크로드를 실행하는 방법에 대한 정보가 추가되었습니다. 2.0 이상 버전을 사용하는 경우 ETL (AWS Glue 추출, 변환 및 로드) 코드를 개발하는 데 선호되는 방법은 대화형 세션입니다. AWS Glue 자세한 내용은 [Jupyter Notebook용 AWS Glue 대화형 세션 설정 및 실행을 참조하십시오](#).

2021년 8월 24일

[블루프린트에서 워크플로 생성
지원\(GA\)](#)

블루프린트에서 일반적인 추출, 변환, 로드 사용 사례를 코딩한 다음 블루프린트에서 워크플로를 생성하는 방법에 대한 정보가 추가되었습니다. 데이터 분석가가 복잡한 ETL 프로세스를 쉽게 생성하고 실행할 수 있습니다. 자세한 정보는 [AWS Glue에서 블루프린트와 워크플로를 사용하여 복잡한 ETL 활동 수행을 참조하세요](#).

2021년 8월 23일

[AWS Glue 버전 3.0을 지원합니다.](#)

Apache Spark ETL 작업을 위한 Apache Spark 3.0 엔진 업그레이드, 기타 최적화 및 업그레이드를 지원하는 AWS Glue 버전 3.0에 대한 지원 정보가 추가되었습니다. 자세한 내용은 [AWS Glue 릴리스 정보](#)와 [AWS Glue 버전 3.0으로 AWS Glue 작업 마이그레이션](#)을 참조하세요. 이 릴리스의 다른 기능에는 AWS Glue 셔플 관리자, SIMD 벡터화 CSV 리더 및 카탈로그 파티션 조건자가 있습니다. 자세한 내용은 [Amazon S3의 AWS Glue Spark 셔플 관리자](#), [AWS Glue의 ETL 입력 및 출력의 포맷 옵션](#) 및 [카탈로그 파티션 조건자를 사용한 서버 측 필터링](#)을 참조하세요.

2021년 8월 18일

[AWS GovCloud \(US\) Region](#)

AWS Glue Studio이제 에서 사용할 수 있습니다. AWS GovCloud (US) Region

2021년 8월 18일

[AWS Glue Studio에서 Python 셀 작성 사용 가능](#)

새 작업을 생성할 때 이제 Python 셀 작업을 생성하도록 선택할 수 있습니다. 자세한 내용은 [작업 생성 프로세스 시작](#) 및 [AWS Glue Studio에서 Python 셀 작업 편집](#)을 참조하세요.

2021년 8월 13일

Amazon EventBridge 이벤트를 사용한 워크플로 시작 지원	이벤트 중심 아키텍처에서 AWS Glue가 이벤트 소비자가 될 수 있는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 Amazon EventBridge 이벤트로 AWS Glue 워크플로 시작 및 워크플로를 시작한 EventBridge 이벤트 보기 를 참조하십시오.	2021년 7월 14일
AWS Glue Schema Registry의 지원되는 데이터 포맷으로 JSON 추가	AVRO 외에 지원되는 데이터 포맷으로서의 JSON에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue Schema Registry 를 참조하세요.	2021년 6월 30일
데이터 카탈로그 테이블 없이 AWS Glue 스트리밍 작업 생성	<code>create_data_frame_from_options</code> Python 함수 또는 Scala 스크립트용 <code>getSource</code> 는 데이터 카탈로그 테이블을 요구하는 대신 데이터 스트림을 직접 참조하는 스트리밍 ETL 작업을 지원합니다.	2021년 6월 15일
AWS Glue기계 학습 변환은 이제 키를 지원합니다 AWS Key Management Service .	콘솔, CLI 또는 API를 사용하여 AWS Glue Machine Learning 변환을 구성할 때 보안 구성 또는 AWS KMS AWS Glue 키를 지정할 수 있습니다. 자세한 내용은 Machine Learning 변환에 데이터 암호화 사용 및 AWS Glue Machine Learning API 를 참조하세요.	2021년 6월 15일

관리형 정책 업데이트 AWSGlueConsoleFullAccess AWS	<p>AWSGlueConsoleFullAccess AWS 관리형 정책의 사소한 업데이트에 대한 정보가 추가되었습니다. 자세한 내용은 AWS 관리형 정책 AWS Glue 업데이트를 참조하십시오.</p>	2021년 6월 10일
작업을 생성하고 편집하는 동안 작업의 데이터 세트 보기	<p>작업 다이어그램의 노드에 대한 새로운 [데이터 미리 보기 (Data preview)] 탭을 사용하여 해당 노드에서 처리된 데이터의 샘플을 볼 수 있습니다. 자세한 내용은 시각적 작업 편집기에서 데이터 미리 보기 사용을 참조하세요.</p>	2021년 6월 7일
크롤러 출력의 테이블 위치를 나타내는 값 지정 지원	<p>크롤러의 출력을 구성할 때 테이블 위치를 나타내는 값 지정에 대한 정보가 추가되었습니다. 자세한 내용은 테이블 위치를 지정하는 방법을 참조하세요.</p>	2021년 6월 4일
Amazon S3 데이터 스토어를 크롤링할 때 데이터 집합의 파일 샘플 크롤링 지원	<p>Amazon S3를 크롤링할 때 파일 샘플을 크롤링하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 크롤러 속성을 참조하십시오.</p>	2021년 5월 10일

[AWS Glue 최적화 parquet 라이터 지원](#)

AWS Glue 최적화된 파켓 라이터를 사용하여 parquet 분류에 따라 테이블을 만들거나 DynamicFrames 업데이트하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue ETL 작업의 데이터 카탈로그에서 테이블 생성, 스키마 업데이트 및 새 파티션 추가 및 AWS Glue의 ETL 입력 및 출력 포맷 옵션](#)을 참조하세요.

2021년 5월 4일

[Kafka 클라이언트 인증 암호 지원](#)

AWS Glue의 스트리밍 ETL 작업이 Apache Kafka 스트림 생성자를 사용한 SSL 클라이언트 인증서 인증을 지원하는 방법에 대한 정보가 추가되었습니다. 이제 인증할 때 AWS Glue가 사용하는 Apache Kafka 클러스터에 대한 AWS Glue 연결을 정의하는 동안 사용자 정의 인증서를 제공할 수 있습니다. 자세한 내용은 [AWS Glue 연결 속성 및 연결 API](#)를 참조하세요.

2021년 4월 28일

[스트리밍 ETL 작업에서 다른 계정의 Amazon Kinesis Data Streams 데이터 사용 지원](#)

다른 계정의 Amazon Kinesis Data Streams에서 데이터를 사용하기 위해 스트리밍 ETL 작업을 생성하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue에서 스트리밍 ETL 작업 추가](#)를 참조하십시오.

2021년 3월 30일

SQL 변환 사용 가능	SQL 변환 노드를 사용하여 SQL 쿼리 형식으로 고유한 변환을 작성할 수 있습니다. 자세한 내용은 SQL 쿼리를 사용하여 데이터 변환 을 참조하세요.	2021년 3월 23일
블루프린트에서 워크플로 생성 지원(공개 평가판)	(공개 미리 보기) 블루프린트에서 일반적인 추출, 변환, 로드 사용 사례를 코딩한 다음 블루프린트에서 워크플로를 생성하는 방법에 대한 정보가 추가되었습니다. 데이터 분석가가 복잡한 ETL 프로세스를 쉽게 생성하고 실행할 수 있습니다. 자세한 정보는 AWS Glue에서 블루프린트와 워크플로를 사용하여 복잡한 ETL 활동 수행 을 참조하세요.	2021년 3월 22일
데이터 대상에 커넥터 사용 가능	이제 데이터 대상에 사용자 지정 또는 AWS Marketplace 커넥터를 사용할 수 있습니다. 자세한 내용은 사용자 정의 커넥터로 작업 작성 을 참조하세요.	2021년 3월 15일
AWS Glue 기계 학습 변환을 위한 열 중요도 지표 지원	AWS Glue 기계 학습 변환 작업 시 열 중요도 지표 보기에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue 콘솔에서 Machine Learning 변환 작업을 참조 하세요.	2021년 2월 5일

[이제 AWS Glue Studio에서 작업 예약 사용 가능](#)

AWS Glue Studio에서 시간 기반의 작업 실행 일정을 정의할 수 있습니다. 콘솔을 사용하여 기본 일정을 생성하거나 Unix와 같은 [cron](#) 구문을 사용하여 더 복잡한 일정을 정의할 수 있습니다. 자세한 내용은 [작업 실행 예약](#)을 참조하세요.

2020년 12월 21일

[AWS Glue 사용자 정의 커넥터 출시](#)

AWS Glue 사용자 정의 커넥터를 사용하면 AWS Marketplace에서 커넥터를 검색하고 구독할 수 있습니다. Apache Spark Datasource, Athena 연합 쿼리 및 JDBC API용으로 구축된 커넥터를 플러그 인하기 위한 AWS Glue Spark 런타임 인터페이스도 출시되었습니다. 자세한 내용은 [AWS Glue Studio에서 커넥터 및 연결 사용](#)을 참조하세요.

2020년 12월 21일

[AWS Glue 버전 2.0에서 스트리밍 ETL 작업 실행 지원](#)

Glue 버전 2.0에서 스트리밍 ETL 작업 실행 지원에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue에서 스트리밍 ETL 작업 추가](#)를 참조하십시오.

2020년 12월 18일

[제한된 실행으로 워크로드 분할 지원](#)

워크로드 분할을 사용하여 데이터 집합 크기의 상한 또는 ETL 작업 실행에서 처리되는 파일 수를 구성하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [제한된 실행으로 워크로드 분할](#)을 참조하세요.

2020년 11월 23일

향상된 파티션 관리 지원	새 API를 사용하여 기존 테이블에 파티션 인덱스를 추가하거나 삭제하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 파티션 인덱스 작업 을 참조하세요.	2020년 11월 23일
AWS Glue Schema Registry 지원	AWS Glue Schema Registry를 사용하여 스키마를 중앙에서 검색, 제어 및 발전시키는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue Schema Registry 를 참조하세요.	2020년 11월 19일
스트리밍 ETL 작업에서 Grok 입력 형식 지원	로그 파일과 같은 스트리밍 소스에 Grok 패턴을 적용하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 스트리밍 소스에 Grok 패턴 적용 을 참조하세요.	2020년 11월 17일
AWS Glue 콘솔에서 워크플로에 태그 추가 지원	AWS Glue 콘솔을 사용하여 워크플로를 생성할 때 태그를 추가하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue 콘솔을 사용하여 워크플로 생성 및 구축 을 참조하세요.	2020년 10월 27일
중분 크롤러 실행 지원	마지막 실행 이후 추가된 Amazon S3 폴더만 크롤링하는 중분 크롤러 실행 지원에 대한 정보가 추가되었습니다. 자세한 내용은 중분 크롤 을 참조하세요.	2020년 10월 21일

[스트리밍 ETL 데이터 소스에 대한 스키마 감지 지원. Avro 스트리밍 ETL 데이터 소스 및 자체 관리형 Kafka 지원](#)

AWS Glue의 스트리밍 추출, 변환, 로드 작업은 이제 수신 레코드의 스키마를 자동으로 감지하고 레코드별로 스키마 변경을 처리할 수 있습니다. 이제 자체 관리형 Kafka 데이터 원본이 지원됩니다. 스트리밍 ETL 작업은 이제 데이터 원본에서 Avro 포맷을 지원합니다. 자세한 내용은 [AWS Glue의 스트리밍 ETL](#), [스트리밍 ETL 작업에 대한 작업 속성 정의](#) 및 [Avro 스트리밍 소스에 대한 참고 사항 및 제한 사항](#)을 참조하세요.

2020년 10월 7일

[MongoDB 및 DocumentDB 데이터 소스 크롤링 지원](#)

MongoDB 및 Amazon DocumentDB(MongoDB 호환) 데이터 원본 크롤링 지원에 대한 정보가 추가되었습니다. 자세한 내용은 [크롤러 정의](#)를 참조하세요.

2020년 10월 5일

[FIPS 규정 준수 지원](#)

AWS Glue를 사용하여 데이터에 액세스할 때 FIPS 140-2 검증된 암호화 모듈이 필요한 고객을 위한 FIPS 엔드포인트에 대한 정보가 추가되었습니다. 자세한 내용은 [FIPS 규정 준수를 참조](#)하세요.

2020년 9월 23일

[AWS Glue Studio에서는 작업 생성 및 모니터링을 위한 사용하기 쉬운 시각적 인터페이스를 제공합니다.](#)

이제 간단한 그래프 기반 인터페이스를 사용하여 데이터를 이동 및 변환하고 AWS Glue에서 실행하는 작업을 작성할 수 있습니다. 그런 다음 AWS Glue Studio의 작업 실행 대시보드를 사용하여 ETL 실행을 모니터링하고 작업이 의도한 대로 작동하는지 확인할 수 있습니다. 자세한 내용은 [AWS Glue Studio 사용 설명서](#)를 참조하십시오.

2020년 9월 23일

[쿼리 성능 향상을 위한 테이블 인덱스 생성 지원](#)

테이블에서 파티션의 하위 집합을 검색할 수 있도록 테이블 인덱스 생성에 대한 정보가 추가되었습니다. 자세한 내용은 [파티션 인덱스 작업](#)을 참조하십시오.

2020년 9월 9일

[AWS Glue 버전 2.0에서 Apache Spark ETL 작업을 실행할 때 시작 시간 단축 지원.](#)

시작 시간 단축, 로깅 변경, 작업 수준에서 추가 Python 모듈 지정 지원과 함께 Apache Spark ETL 작업 실행을 위한 업그레이드된 인프라를 제공하는 AWS Glue 버전 2.0 지원에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue 릴리스 정보 및 단축된 시작 시간으로 Spark ETL 작업 실행](#)을 참조하십시오.

2020년 8월 10일

<u>동시 워크플로 실행 수 제한 지원</u>	특정 워크플로에 대한 동시 워크플로 실행 수를 제한하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 <u>AWS Glue 콘솔을 사용하여 워크플로 생성 및 구축</u> 을 참조하세요.	2020년 8월 10일
<u>VPC 엔드포인트를 사용한 Amazon S3 데이터 스토어 크롤링 지원</u>	보안, 감사 또는 제어 목적으로 Amazon Virtual Private Cloud 환경(Amazon VPC)에서만 액세스할 수 있도록 Amazon S3 데이터 스토어를 구성하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 <u>VPC 엔드포인트를 사용하여 Amazon S3 데이터 스토어 크롤링</u> 을 참조하세요.	2020년 8월 7일
<u>워크플로 실행 재개 지원</u>	하나 이상의 노드(작업 또는 크롤러)가 성공적으로 완료되지 않아 부분적으로만 완료된 워크플로 실행을 재개하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 <u>워크플로 실행 복구 및 재개</u> 를 참조하세요.	2020년 7월 27일
<u>AWS Glue의 Kafka 연결에서 프라이빗 CA 인증서 사용 지원</u>	AWS Glue에서 Kafka 연결에 대한 프라이빗 CA 인증서 사용을 지원하는 새로운 연결 옵션에 대한 정보가 추가되었습니다. 자세한 내용은 <u>AWS Glue에서 ETL 관련 연결 유형 및 옵션과 AWS Glue가 사용하는 특정 파라미터</u> 를 참조하세요.	2020년 7월 20일

다른 계정의 DynamoDB 데이터 읽기 지원	다른 AWS 계정의 DynamoDB 테이블에서 데이터 읽기에 대한 AWS Glue 지원에 대한 정보가 추가되었습니다. 자세한 내용은 다른 계정의 DynamoDB 데이터에서 읽기 를 참조하세요.	2020년 7월 17일
AWS Glue 버전 1.0 이상에서 DynamoDB 라이더 연결 지원	DynamoDB 라이더 지원에 대한 정보와 DynamoDB에서 읽거나 쓸 수 있는 새로운 연결 옵션 또는 업데이트된 연결 옵션이 추가되었습니다. 자세한 내용은 AWS Glue의 ETL 연결 유형 및 옵션 을 참조하세요.	2020년 7월 17일
AWS Glue 및 Lake Formation을 모두 사용하여 리소스 링크 및 교차 계정 액세스 제어 지원	리소스 링크라는 새 데이터 카탈로그 객체에 대한 내용과 AWS Glue 및 AWS Lake Formation이 모두 있는 계정 간에 데이터 카탈로그 리소스 공유를 관리하는 방법에 대한 내용이 추가되었습니다. 자세한 내용은 교차 계정 액세스 권한 부여 및 테이블 리소스 링크 를 참조하세요.	2020년 7월 7일
DynamoDB 데이터 스토어를 크롤링할 때 레코드 샘플링 지원	DynamoDB 데이터 저장소를 크롤링할 때 구성할 수 있는 새 속성에 대한 정보가 추가되었습니다. 자세한 내용은 크롤러 속성 을 참조하십시오.	2020년 6월 12일

워크플로우 실행 중지 지원	특정 워크플로우에 대한 워크플로우 실행을 중지하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 워크플로우 실행 중지 를 참조하십시오.	2020년 5월 14일
Spark 스트리밍 ETL 작업 지원	스트리밍 데이터 원본을 사용하여 추출, 변환 및 로드(ETL) 작업을 생성하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue에서 스트리밍 ETL 작업 추가 를 참조하십시오.	2020년 4월 27일
ETL 작업 실행 후 데이터 카탈로그에서 테이블 생성, 스키마 업데이트, 새 파티션 추가 지원	데이터 카탈로그에서 테이블 생성, 스키마 업데이트 및 새 파티션 추가를 활성화하여 ETL 작업의 결과를 확인하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue ETL 작업의 데이터 카탈로그에서 테이블 생성, 스키마 업데이트 및 새 파티션 추가 를 참조하십시오.	2020년 4월 2일
AWS Glue에서 Apache Avro 데이터 형식을 ETL 입력 및 출력으로 버전 지정 지원	AWS Glue에서 Apache Avro 데이터 포맷을 ETL 입력 및 출력으로 버전을 지정하는 것에 대한 정보가 추가되었습니다. 기본 버전은 1.7입니다. <code>version</code> 포맷 옵션을 사용해 Avro 버전 1.8을 지정하여 논리적 읽기/쓰기를 활성화할 수 있습니다. 자세한 내용은 AWS Glue에서 ETL 입력 및 출력의 포맷 옵션 을 참조하십시오.	2020년 3월 31일

[Amazon S3에 Parquet 데이터를 쓸 수 있도록 EMRFS S3 최적화 커미터 지원](#)

AWS Glue 작업을 생성하거나 업데이트할 때 Amazon S3에 Parquet 데이터를 쓸 수 있도록 새 플래그를 설정해 EMRFS S3 최적화 커미터를 활성화하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue가 사용하는 특정 파라미터](#)를 참조하십시오.

2020년 3월 30일

[리소스 태그로 AWS 관리되는 리소스로서의 기계 학습 변환 지원](#)

AWS 리소스 태그를 사용하여 기계 학습 변환에 대한 액세스를 관리하고 제어하는 방법에 대한 정보가 추가되었습니다. AWS Glue 에서 작업, 트리거, 엔드포인트, 크롤러 및 기계 학습 변환에 AWS 리소스 태그를 할당할 수 있습니다. AWS Glue 자세한 내용은 [AWS Glue의 AWS 태그](#)를 참조하세요.

2020년 3월 2일

[재정의할 수 없는 작업 인수 지원](#)

트리거에서 또는 작업을 실행할 때 재정의할 수 없는 특수 작업 파라미터에 대한 지원 정보가 추가되었습니다. 자세한 내용은 [AWS Glue의 작업 추가](#)를 참조하십시오.

2020년 2월 12일

[Amazon S3의 데이터 집합을 사용할 수 있도록 새로운 변환 지원](#)

Apache Spark 애플리케이션에서 Amazon S3의 데이터 집합을 사용할 수 있도록 새로운 변환(병합, 제거 및 전환) 및 Amazon S3 스토리지 클래스 제외 사항에 대한 정보가 추가되었습니다. Python에서 이러한 변환에 대한 지원에 대한 자세한 내용은 [Amazon S3의 데이터 세트 사용을 참조하십시오](#) [mergeDynamicFrame](#). [스칼라의 경우 mergeDynamicFrames](#) 및 [Scala API를 참조하십시오](#). [AWS Glue GlueContext](#)

2020년 1월 16일

[ETL 작업에서 새 파티션 정보로 데이터 카탈로그 업데이트 지원](#)

ETL (추출, 변환, 로드) 스크립트를 코딩하여 새 파티션 AWS Glue Data Catalog 정보로 업데이트하는 방법에 대한 정보가 추가되었습니다. 이 기능을 사용하면 새 파티션을 확인하기 위해 작업 완료 후 크롤러를 다시 실행할 필요가 없습니다. 자세한 내용은 [새 파티션으로 데이터 카탈로그 업데이트](#)를 참조하세요.

2020년 1월 15일

[새 튜토리얼: 노트북 사용 SageMaker](#)

Amazon SageMaker 노트북을 사용하여 ETL 및 기계 학습 스크립트를 개발하는 방법을 보여주는 자습서가 추가되었습니다. [자습서: 개발 엔드포인트와 함께 Amazon SageMaker Notebook 사용](#)을 참조하십시오.

2020년 1월 3일

MongoDB 및 Amazon DocumentDB(MongoDB 호환)에서의 읽기 지원	<p>MongoDB 및 Amazon DocumentDB(MongoDB와 호환)에서의 읽기 및 쓰기에 대한 새 연결 유형 및 연결 옵션에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue의 ETL 연결 유형 및 옵션을 참조하세요.</p>	2019년 12월 17일
다양한 수정 및 설명	<p>전체적으로 수정 및 설명을 추가했습니다. 알려진 문제 장에서 항목을 제거했습니다. 데이터 카탈로그 암호화 설정을 지정하고 보안 구성을 생성할 때 AWS Glue가 대칭 고객 마스터 키(CMK)만 지원하는 경고가 추가되었습니다. AWS Glue가 Amazon DynamoDB에 쓰기를 지원하지 않는다는 메모를 추가했습니다.</p>	2019년 12월 9일
사용자 지정 JDBC 드라이버 지원	<p>MySQL 버전 8 및 Oracle Database 버전 18과 같이 AWS Glue에서 기본적으로 지원하지 않는 JDBC 드라이버로 데이터 원본과 대상에 연결하는 데 대한 정보가 추가되었습니다. 자세한 내용은 JDBC connectionType 값을 참조하십시오.</p>	2019년 11월 25일

[다양한 개발 엔드포인트에 SageMaker 노트북 연결 지원](#)

SageMaker 노트북을 다양한 개발 엔드포인트에 연결하는 방법에 대한 정보가 추가되었습니다. 새 개발 엔드포인트로 전환하기 위한 새로운 콘솔 작업과 새 SageMaker IAM 정책을 설명하는 업데이트입니다. 자세한 내용은 [AWS Glue 콘솔에서 노트북 사용하기 및 Amazon SageMaker Notebook용 IAM 정책 생성을 참조하십시오](#).

2019년 11월 21일

[기계 학습 변환에서 AWS Glue 버전 지원](#)

AWS Glue의 어떤 버전이 기계 학습 변환과 호환되는지 나타내기 위해 기계 학습 변환에서 AWS Glue 버전을 정의하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue 콘솔에서 기계 학습 변환 작업을 참조하세요](#).

2019년 11월 21일

[작업 북마크 되돌리기 지원](#)

이전 작업 실행으로 작업 북마크를 되돌리는 방법에 대한 정보를 추가했습니다. 이제 후속 작업 실행 시 북마크로 지정된 작업 실행의 데이터만 다시 처리합니다. 두 북마크 간에 작업을 실행할 수 있도록 허용하는 `job-bookmark-pause` 옵션에 대한 두 개의 새로운 하위 옵션 설명이 있습니다. 자세한 내용은 [작업 북마크를 사용해 처리된 데이터 추적 및 AWS Glue에서 사용되는 특정 파라미터](#)를 참조하세요.

2019년 10월 22일

[데이터 스토어에 연결하기 위한 사용자 지정 JDBC 인증서 지원](#)

AWS Glue 데이터 소스 또는 대상에 SSL을 연결하기 위해 AWS Glue의 사용자 지정 JDBC 인증서 지원에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue 콘솔에서 연결 작업을 참조](#)하세요.

2019년 10월 10일

[Python Wheel 지원](#)

Python 셸 작업에 대한 종속성으로 AWS Glue의 wheel 파일(egg 파일과 함께) 지원에 대한 정보가 추가되었습니다. 자세한 내용은 [자체 Python 라이브러리 제공](#)을 참조하십시오.

2019년 9월 26일

[AWS Glue에서 개발 엔드포인트의 버전 관리 지원](#)

개발 엔드포인트에서 Glue version을 정의하는 방법에 대한 정보가 추가되었습니다. Glue version은 AWS Glue에서 지원하는 Apache Spark 및 Python의 버전을 결정합니다. 자세한 내용은 [개발 엔드포인트 추가](#)를 참조하십시오.

2019년 9월 19일

[Spark UI를 사용한 AWS Glue 모니터링 지원](#)

Apache Spark UI를 사용하여 AWS Glue 작업 시스템에서 실행 중인 AWS Glue ETL 작업과 AWS Glue 개발 엔드포인트의 Spark 애플리케이션을 모니터링하고 디버그하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [Spark UI를 사용한 AWS Glue 모니터링](#)을 참조하세요.

2019년 9월 19일

퍼블릭 AWS Glue ETL 라이브러리를 사용해 로컬 ETL 스크립트를 개발할 수 있도록 지원 개선	AWS Glue 버전 1.0이 현재 지원되고 있다는 것을 반영하기 위해 AWS Glue ETL 라이브러리 콘텐츠가 업데이트되었습니다. 자세한 내용은 AWS Glue ETL 라이브러리를 사용해 로컬에서 ETL 스크립트 개발 및 테스트 를 참조하세요.	2019년 9월 18일
작업 실행 시 Amazon S3 스토리지 클래스를 제외할 수 있도록 지원	Amazon S3에서 파일 또는 파티션을 읽어오는 AWS Glue ETL 작업을 실행할 때 Amazon S3 스토리지 클래스를 제외하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 Amazon S3 스토리지 클래스 제외 를 참조하세요.	2019년 8월 29일
퍼블릭 AWS Glue ETL 라이브러리를 사용해 로컬 ETL 스크립트를 개발할 수 있도록 지원	네트워크에 연결할 필요 없이 로컬에서 Python 및 Scala ETL 스크립트를 개발하고 테스트하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 AWS Glue ETL 라이브러리를 사용해 로컬에서 ETL 스크립트 개발 및 테스트 를 참조하세요.	2019년 8월 28일
알려진 문제	AWS Glue의 알려진 문제에 대한 정보가 추가되었습니다. 자세한 내용은 알려진 AWS Glue 문제 단원을 참조하십시오.	2019년 8월 28일

AWS Glue에서 기계 학습 변환 지원	사용자 지정 변환 생성을 위해 AWS Glue에서 제공하는 기계 학습 기능에 대해 정보를 추가했습니다. 작업을 생성할 때 이러한 변환을 만들 수 있습니다. 자세한 내용은 AWS Glue에서의 기계 학습 변환 을 참조하십시오.	2019년 8월 8일
공유 Amazon Virtual Private Cloud 지원	공유 Amazon Virtual Private Cloud에 대한 AWS Glue 지원 관련 정보가 추가되었습니다. 자세한 내용은 공유 Amazon VPC 를 참조하십시오.	2019년 8월 6일
AWS Glue에서 버전 관리 지원	작업 속성에서 Glue version을 정의하는 방법에 대한 정보가 추가되었습니다. AWS Glue 버전은 AWS Glue에서 지원하는 Apache Spark 및 Python의 버전을 결정합니다. 자세한 내용은 AWS Glue의 작업 추가 를 참조하십시오.	2019년 7월 24일
개발 엔드포인트의 추가 구성 옵션 지원	메모리 집약적 워크로드가 있는 개발 엔드포인트의 구성 옵션에 대한 정보를 추가했습니다. 실행기당 더 많은 메모리를 제공하는 두 가지 새로운 구성 중에서 선택할 수 있습니다. 자세한 내용은 AWS Glue 콘솔상의 개발 엔드포인트 작업을 참조하십시오 .	2019년 7월 24일

[워크플로를 사용한 ETL\(추출, 전송, 로드\) 활동 수행 지원](#)

AWS Glue가 단일 엔터티로 실행하고 추적할 수 있는 복잡한 다중 작업 추출, 변환, 로드 활동을 설계하기 위해 워크플로라는 새로운 구조 사용에 대한 정보를 추가했습니다. 자세한 내용은 [AWS Glue에서 워크플로를 사용하여 복잡한 ETL 활동 수행](#)을 참조하세요.

2019년 6월 20일

[Python 셸 작업의 Python 3.6 지원](#)

Python 셸 작업의 Python 3.6 지원에 대한 정보를 추가했습니다. Python 2.7 또는 Python 3.6을 작업 속성으로 지정할 수 있습니다. 자세한 내용은 [AWS Glue에서 Python 셸 작업 추가](#)를 참조하세요.

2019년 6월 5일

[Virtual Private Cloud\(VPC\) 엔드포인트 지원](#)

VPC에서 인터페이스 엔드포인트를 통해 AWS Glue에 직접 연결하는 방법에 대한 정보가 추가되었습니다. VPC 인터페이스 엔드포인트를 사용하는 경우 VPC와 AWS Glue 간의 통신은 AWS 네트워크에서 완전하고 안전하게 수행됩니다. 자세한 내용은 [VPC 엔드포인트와 함께 AWS Glue 사용](#)을 참조하세요.

2019년 6월 4일

[AWS Glue 작업에 대한 실시간 지속 로깅 지원](#)

드라이버 로그, 각 실행자 로그 및 Spark 작업 진행률 표시줄을 CloudWatch 포함하여 실시간 Apache Spark 작업 로그를 활성화하고 보는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue 작업에 대한 지속 로깅](#)을 참조하십시오.

2019년 5월 28일

[크롤러 소스로 기존 데이터 카탈로그 테이블 지원](#)

크롤러 소스로 기존 데이터 카탈로그 테이블의 목록 지정에 대한 정보가 추가되었습니다. 이제 크롤러는 테이블 스키마에 대한 변경 사항을 감지하고 테이블 정의를 업데이트하며 새 데이터를 사용할 수 있게 되면 새 파티션을 등록할 수 있습니다. 자세한 내용은 [크롤러 속성](#)을 참조하십시오.

2019년 5월 10일

[메모리 집약적 작업에 대한 추가 구성 옵션 지원](#)

메모리 집약적 워크로드가 있는 Apache Spark 작업의 구성 옵션에 대한 정보를 추가했습니다. 실행기당 더 많은 메모리를 제공하는 두 가지 새로운 구성 중에서 선택할 수 있습니다. 자세한 내용은 [AWS Glue의 작업 추가](#)를 참조하십시오.

2019년 4월 5일

[CSV 사용자 지정 분류자 지원](#)

사용자 지정 CSV 분류자를 사용하여 다양한 유형의 CSV 데이터 스키마 추론에 대한 정보를 추가했습니다. 자세한 내용은 [사용자 지정 분류자 작성](#)을 참조하십시오.

2019년 3월 26일

[AWS 리소스 태그 지원](#)

리소스에 대한 액세스를 관리하고 제어하는 데 도움이 되는 AWS 리소스 태그 사용에 대한 정보가 추가되었습니다. AWS Glue 에서 작업, 트리거, 엔드포인트 및 크롤러에 AWS 리소스 태그를 할당할 수 있습니다. AWS Glue 자세한 내용은 [AWS Glue의AWS 태그](#)를 참조하세요.

2019년 3월 20일

[Spark SQL 작업에 데이터 카탈로그 지원](#)

를 외부 Apache Hive AWS Glue Metastore로 사용하도록 작업 및 개발 엔드포인트를 구성하는 방법에 대한 정보가 추가되었습니다. AWS Glue Data Catalog 이렇게 하면 작업 및 개발 엔드포인트가 AWS Glue Data Catalog에 저장된 테이블에 대해 Apache Spark SQL 쿼리를 직접 실행할 수 있습니다. 자세한 내용은 [Spark SQL 작업에 대한AWS Glue Data Catalog 지원](#)을 참조하세요.

2019년 3월 14일

[Python 셸 작업 지원](#)

Python 셸 작업과 새로운 필드 Maximum capacity(최대 용량)에 대한 정보가 추가되었습니다. 자세한 내용은 [AWS Glue 에서 Python 셸 작업 추가](#)를 참조하세요.

2019년 1월 18일

[데이터베이스와 테이블을 변경할 때 알림 지원](#)

데이터베이스, 테이블 및 파티션 API 직접 호출의 변경 사항에 대해 생성된 이벤트 정보가 추가되었습니다. 이벤트에서 이러한 CloudWatch 이벤트에 응답하도록 액션을 구성할 수 있습니다. 자세한 내용은 [CloudWatch 이벤트 AWS Glue 자동화를 참조하십시오](#).

2019년 1월 16일

[연결 암호 암호화 지원](#)

연결 객체에 사용되는 암호를 암호화하는 방법에 대한 정보를 추가했습니다. 자세한 내용은 [Encrypting Connection Passwords](#) 단원을 참조하십시오.

2018년 12월 11일

[리소스 수준 권한 및 리소스 기반 정책에 대한 지원](#)

AWS Glue에서 리소스 수준 권한 및 리소스 기반 정책을 사용하는 방법에 대한 정보를 추가했습니다. 자세한 내용은 [AWS Glue의 보안](#)에 수록된 주제를 참조하십시오.

2018년 10월 15일

[SageMaker 노트북 지원](#)

AWS Glue개발 엔드포인트가 있는 SageMaker 노트북 사용에 대한 정보가 추가되었습니다. 자세한 내용은 [노트북 관리](#)를 참조하십시오.

2018년 10월 5일

[암호화 지원](#)

AWS Glue의 암호화를 사용하는 방법에 대한 정보를 추가했습니다. 자세한 내용은 [저장 데이터 암호화](#), [전송 데이터 암호화](#) 및 [AWS Glue의 암호화 설정](#)을 참조하세요.

2018년 8월 24일

Apache Spark 작업 지표 지원	ETL 작업의 디버깅 및 프로파일링 향상을 위해 Apache Spark 측정치 사용에 대한 정보를 추가했습니다. 드라이버 및 실행기의 읽고 쓴 바이트 수, 메모리 사용량 및 CPU 부하 같은 실행 시간 측정치와, AWS Glue 콘솔의 실행기간의 데이터 셔플을 쉽게 추적할 수 있습니다. 자세한 내용은 CloudWatch 지표를 AWS Glue 사용한 모니터링, 작업 모니터링 및 디버깅, AWS Glue 콘솔에서의 작업 사용을 참조하십시오.	2018년 7월 13일
데이터 원본으로서 DynamoDB 지원	DynamoDB 크롤링 및 이를 ETL 작업 데이터 원본으로 사용하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 크롤러를 사용하여 테이블 카탈로그 작성 및 연결 파라미터 를 참조하십시오.	2018년 7월 10일
노트북 서버 생성 절차 업데이트	개발 엔드포인트와 연결된 Amazon EC2 인스턴스에서 노트북 서버를 생성하는 방법에 대한 정보가 업데이트되었습니다. 자세한 내용은 개발 엔드포인트와 연결된 노트북 서버 생성 단원을 참조하십시오.	2018년 7월 9일
RSS에서 현재 사용 가능한 업데이트	이제 AWS Glue Developer Guide에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.	2018년 6월 25일

작업 지연 알림 지원	작업 중의 지연 임계값을 구성하는 내용이 추가됩니다. 자세한 내용은 AWS Glue의 작업 추가 를 참조하십시오.	2018년 5월 25일
크롤러를 구성하여 새 열 추가	크롤러의 새 구성 옵션에 대한 정보가 추가되었습니다. MergeNewColumns 더 자세한 내용은 크롤러 구성하기 를 참조하십시오.	2018년 5월 7일
작업 제한 시간 지원	작업 중의 타임아웃 임계값을 설정하는 내용이 추가됩니다. 자세한 내용은 AWS Glue의 작업 추가 를 참조하십시오.	2018년 10월 4일
추가 실행 상태를 기반으로 Scala ETL 스크립트 및 트리거 작업 지원	ETL 프로그래밍 언어로써 사용되는 Scala에 대한 자세한 내용이 포함되었습니다. 또한, 트리거 API는 현재 (모든 조건과 더불어) 조건이 맞으면 API의 시작을 지원합니다. 작업도 ("succeeded" 작업 실행과 더불어) "failed" 또는 "stopped" 작업 실행을 기반으로 시작됩니다.	2018년 1월 12일

이전 업데이트

다음 표에서는 2018년 1월 이전 AWS Glue 개발자 안내서의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다.

변경 사항	설명	날짜
XML 데이터 원본 및 새로운 크롤러 구성 옵션 지원	파티션 변경에 따른 XML 데이터 원본 및 새로운 크롤러 옵션을 분류하는 추가된 정보.	2017년 11월 16일
새로운 변환, 추가 Amazon RDS 데이터베이스 엔진 지원 및 개발 엔드포인트 개선	맵 및 필터 변환, Amazon RDS Microsoft SQL Server 및 Amazon RDS Oracle에 대한 지원, 개발 엔드포인트를 위한 새로운 기능에 대한 정보가 추가되었습니다.	2017년 9월 29일
AWS Glue 최초 릴리스	AWS Glue Developer Guide가 처음으로 릴리스되었습니다.	2017년 8월 14일

AWS 용어집

최신 AWS 용어는 참조의 [AWS 용어집](#)을 참조하십시오. AWS 용어집

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.