aws

개발자 가이드

AWS Step Functions

AWS Step Functions: 개발자 가이드

Table of Contents

무엇입니까 AWS Step Functions?

AWS Step Functions 분산 애플리케이션을 구축하고, 프로세스를 자동화하고, 마이크로서비스를 오케스트레이션하고, 데이터 및 기계 학습 (ML) 파이프라인을 생성하는 데 도움이 되는 시각적 워크플로 서비스입니다.

Step Functions의 그래픽 콘솔에서 애플리케이션의 워크플로를 일련의 이벤트 기반 단계로 볼 수 있습니다.

Step Functions는 스테이트 머신과 태스크를 기반으로 합니다. Step Functions에서는 상태 머신을 워크플로라고 하며, 워크플로우는 일련의 이벤트 기반 단계입니다. 워크플로의 각 단계를 상태라고 합니다. 예를 들어 작업 상태는 다른 AWS 서비스 서비스나 API 호출과 같이 다른 AWS 서비스가 수행하는 작업 단위를 나타냅니다.

Step Functions의 내장 컨트롤을 사용하면 워크플로의 각 단계 상태를 검사하여 애플리케이션이 예상대로 순서대로 실행되는지 확인할 수 있습니다. 사용 사례에 따라 Step Functions가 Lambda와 같은 AWS 서비스를 호출하여 작업을 수행하도록 할 수 있습니다. 기계 학습 모델을 처리하고 게시하는 워크플로를 만들 수 있습니다. Step Functions의 제어 AWS 서비스 (예: ETL) 추출, 변환 및 로드 (ETL) 워크플로를 생성할 수 있습니다. AWS Glue또한 사람의 상호 작용이 필요한 애플리케이션을 위해 오래 실행되는 자동화된 워크플로를 만들 수 있습니다.

> **Tip**
>
> Step Functions의 사용 방법을 알아보려면 AWS Step Functions 워크숍의 대화형 모듈을 따르거나 이 가이드의 시작하기 섹션을 읽고 신용 카드 신청 워크플로를 생성하십시오.

주제

- AWS SDK 및 최적화된 통합
- 표준 및 Express 워크플로
- 사용 사례
- 서비스 통합
- 지원되는 리전
- Step Functions를 처음 사용하시나요?

AWS SDK 및 최적화된 통합

다른 AWS 서비스를 호출하려면 Step Functions의 AWS SDK 통합을 사용하거나 Step Functions의 최적화된 통합 중 하나를 사용할 수 있습니다.

- AWS SDK 통합을 통해 상태 머신에서 200개가 넘는 AWS 서비스를 직접 호출하여 9,000개 이상의 API 작업에 액세스할 수 있습니다.
- Step Functions의 최적화된 통합은 상태 시스템에서의 사용을 간소화하도록 사용자 지정되었습니다.

표준 및 Express 워크플로

Step Functions에는 두 가지 워크플로 유형이 있습니다. 표준 워크플로는 워크플로를 정확히 1회 실행하며 최대 1년 동안 실행될 수 있습니다. 즉, 표준 워크플로의 각 단계는 정확히 1회 실행됩니다. 하지만 Express at-least-once 워크플로는 워크플로가 실행되며 최대 5분 동안 실행할 수 있습니다. 즉, Express 워크플로의 단계 하나 이상은 1회 넘게 실행되지만 워크플로의 각 단계는 최소 1회 이상 실행됩니다.

실행은 워크플로를 실행하여 작업을 수행하는 인스턴스입니다. 표준 워크플로에서는 실행 내역과 시각적 디버깅을 보여주므로 감사가 가능한 장기 실행 워크플로에 적합합니다. Express 워크플로는 스트리밍 데이터 처리 및 IoT 데이터 수집과 같은 high-event-rate 워크로드에 적합합니다.

표준 워크플로 사양

- 실행 속도 1초당 2,000
- 상태 전환 속도 1초당 4,000
- 상태 전환을 기준으로 요금 책정
- 실행 내역 및 시각적 디버깅 표시
- 모든 서비스 통합 및 패턴 지원

Express 워크플로 사양

- 실행 속도 1초당 100,000
- 무제한에 가까운 상태 전환 속도
- 실행 횟수 및 기간을 기준으로 요금 책정

- Amazon에 실행 기록 보내기 CloudWatch
- 활성화된 로그 수준을 기반으로 실행 내역과 시각적 디버깅 표시
- 모든 서비스 통합 및 대부분의 패턴 지원

Step Functions 요금을 포함한 표준 및 Express 워크플로에 대한 자세한 내용은 다음을 참조하세요.

- 표준 워크플로와 Express 워크플로 비교
- AWS Step Functions 요금

사용 사례

Step Functions는 애플리케이션의 구성 요소와 로직을 관리하므로 코드 작성을 줄이고 애플리케이션을 빠르게 빌드하고 업데이트하는 데 집중할 수 있습니다. 이 섹션에서는 Step Functions를 사용하는 일반적인 사용 사례를 설명합니다.

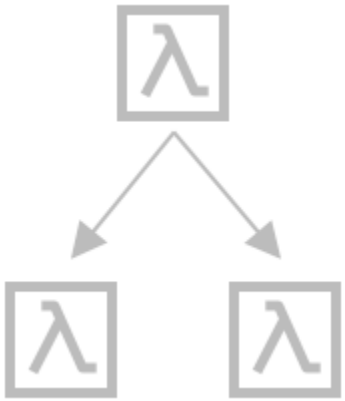
사용 사례 #1: 함수 오케스트레이션

Lambda 함수 그룹(단계)을 특정 순서로 실행하는 워크플로를 만듭니다. Lambda 함수 하나의 출력이 다음 Lambda 함수 입력으로 전달됩니다. 워크플로의 마지막 단계에서 결과를 제공합니다. Step Functions를 사용하면 워크플로의 각 단계가 서로 상호 작용하는 방식을 확인할 수 있으므로 각 단계에서 의도한 함수를 수행하는지 확인할 수 있습니다.

함수 그룹이 포함된 상태 시스템을 만드는 방법을 보여주는 자습서는 다음을 참조하세요.

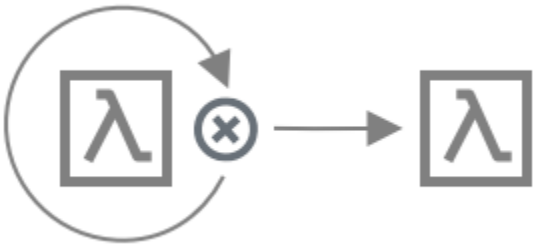
- 시작하기 AWS Step Functions

사용 사례 #2: 분기



고객이 신용 한도 증가를 요청합니다. `Choice` 상태를 사용하면 Step Functions에서 `Choice` 상태 입력에 따라 결정을 내리도록 할 수 있습니다. 요청이 고객의 사전 승인된 신용 한도를 초과하는 경우 Step Functions에서 고객의 요청을 승인하기 위해 관리자에게 보내도록 할 수 있습니다. 요청이 고객의 사전 승인된 신용 한도보다 적은 경우 Step Functions에서 요청을 자동으로 승인하도록 할 수 있습니다.

사용 사례 #3: 오류 처리



`Retry`

이 사용 사례에서는 고객이 사용자 이름을 요청합니다. 처음에는 고객 요청이 실패합니다. `Retry` 문을 사용하면 Step Functions에서 고객 요청을 다시 시도하도록 할 수 있습니다. 두 번째로 고객 요청이 성공했습니다.

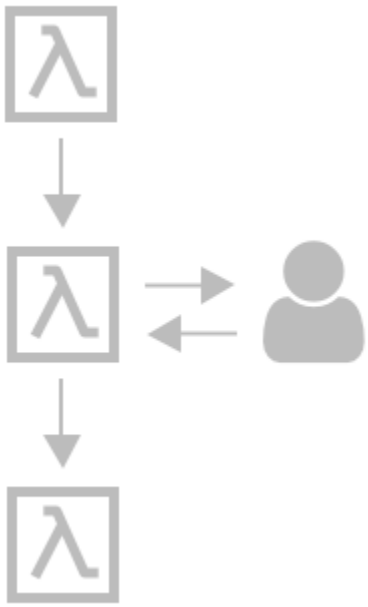
`Catch`

비슷한 사용 사례에서는 고객이 사용할 수 없는 사용자 이름을 요청합니다. `Catch` 문을 사용하면 Step Functions에서 사용 가능한 사용자 이름을 제안하도록 할 수 있습니다. 고객이 사용 가능한 사용자 이름을 사용하는 경우 Step Functions가 워크플로의 다음 단계로 이동하여 확인 이메일을 보내도록 할 수 있습니다. 고객이 사용 가능한 사용자 이름을 사용하지 않는 경우 Step Functions가 워크플로의 다른 단계로 이동하여 가입 프로세스를 다시 시작하도록 합니다.

`Retry` 및 `Catch` 문에 대한 자세한 예제는 다음을 참조하세요.

- Step Functions에서 오류 처리

사용 사례 #4: 휴먼인더루프

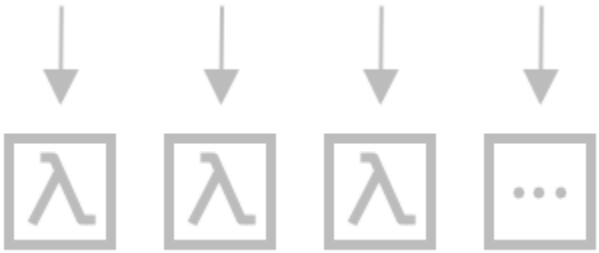


고객 중 한 명이 은행 앱을 사용하여 친구에게 송금합니다. 고객이 확인 이메일을 기다립니다. 콜백과 작업 토큰을 사용하면 Step Functions가 고객이 송금하고 고객의 친구가 돈을 받으면 다시 보고하도록 Lambda에게 지시하도록 할 수 있습니다. Lambda에서 고객의 친구가 돈을 받았음을 다시 보고하면 Step Functions가 워크플로의 다음 단계로 이동하여 고객에게 확인 이메일을 보내도록 할 수 있습니다.

작업 토큰이 포함된 콜백을 보여주는 샘플 프로젝트를 보려면 다음을 참조하세요.

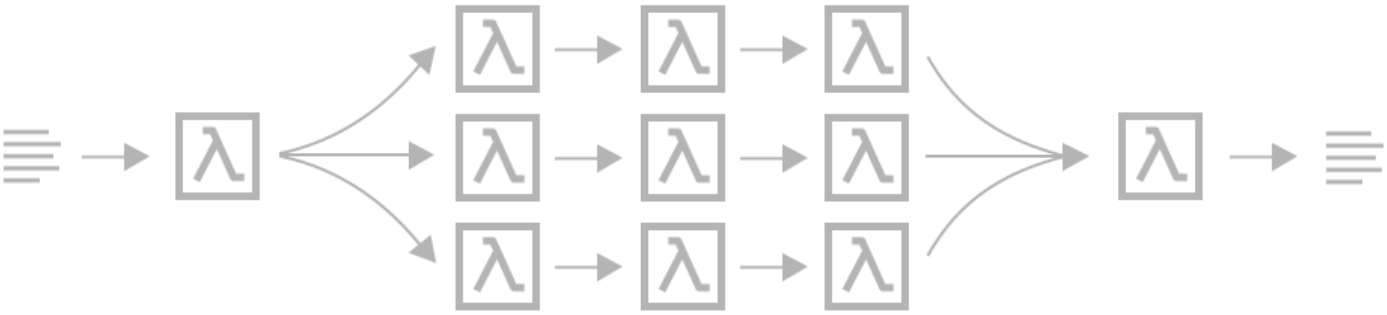
- 콜백 패턴 예제(Amazon SQS, Amazon SNS, Lamda)

사용 사례 #5: 병렬 처리



고객은 비디오 파일을 5가지 디스플레이 해상도로 변환하여 시청자가 여러 장치에서 비디오를 시청할 수 있습니다. `Parallel` 상태를 사용하면 Step Functions가 비디오 파일을 입력하므로 Lambda에서 이 비디오 파일을 5가지의 디스플레이 해상도로 동시에 처리할 수 있습니다.

사용 사례 #6: 동적 병렬 처리



고객이 품목 3개를 주문하며 사용자는 각 품목 배송을 준비해야 합니다. 각 품목의 재고 여부를 확인하고 각 품목을 수집한 다음 각 품목을 포장하여 배송합니다. `Map` 상태를 사용하면 Step Functions는 Lambda가 고객의 각 품목을 동시에 처리하도록 합니다. 고객의 모든 품목을 배송하기 위해 포장하면 Step Functions는 워크플로의 다음 단계로 이동하여 추적 정보가 포함된 확인 이메일을 고객에게 보냅니다.

`Map` 상태를 사용하여 동적 병렬 처리를 보여주는 샘플 프로젝트를 보려면 다음을 참조하세요.

- Map 상태를 사용하여 데이터 동적 처리

서비스 통합

Step Functions는 여러 AWS 서비스와 통합됩니다. Step Functions를 이러한 서비스와 결합하려면 다음 서비스 통합 패턴을 사용합니다.

응답 요청(기본값)

- 서비스를 직접적으로 호출하고 Step Functions가 HTTP 응답을 가져온 후에 다음 상태로 진행하도록 합니다.

작업 실행(.sync)

- 서비스를 직접적으로 호출하고 작업이 완료될 때까지 Step Functions가 기다리도록 합니다.

태스크 토큰이 포함된 콜백을 기다립니다 (. waitForTask토큰)

- 작업 토큰이 포함된 서비스를 직접적으로 호출하고 작업 토큰이 콜백과 함께 반환될 때까지 Step Functions가 기다리도록 합니다.

다음 표에서는 Step Functions에 사용 가능한 서비스 통합과 서비스 통합 패턴을 보여줍니다.

표준 워크플로와 익스프레스 워크플로는 동일한 통합을 지원하지만 동일한 통합 패턴은 지원하지 않습니다.

- 최적화된 통합 패턴 지원은 통합마다 다릅니다.
- 익스프레스 워크플로는 작업 실행 (.sync) 또는 콜백 대기 (. waitForTask토큰).
- 자세한 정보는 표준 워크플로와 Express 워크플로 비교을 참조하세요.

Standard Workflows

지원되는 서비스 통합

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
최적화된 통합	Amazon API Gateway	✓		✓
	Amazon Athena	✓	✓	
	AWS Batch	✓	✓	
	Amazon Bedrock	✓	✓	✓
	AWS CodeBuild	✓	✓	
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓	✓	✓
	Amazon EKS	✓	✓	✓
	Amazon EMR	✓	✓	

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
	Amazon EMR on EKS	✓	✓	
	Amazon EMR Serverless	✓	✓	
	Amazon EventBridge	✓		✓
	AWS Glue	✓	✓	
	AWS Glue DataBrew	✓	✓	
	AWS Lambda	✓		✓
	AWS Elemental MediaConvert	✓	✓	
	Amazon SageMaker	✓	✓	
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓
AWS SDK 통합	200개 초과	✓		✓

Express Workflows

지원되는 서비스 통합

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
최적화된 통합	Amazon API Gateway	✓		
	Amazon Athena	✓		
	AWS Batch	✓		
	Amazon Bedrock	✓		
	AWS CodeBuild	✓		
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓		
	Amazon EKS	✓		
	Amazon EMR	✓		
	Amazon EMR on EKS	✓		
	Amazon EMR Serverless	✓		
	Amazon EventBridge	✓		
	AWS Glue	✓		
	AWS Glue DataBrew	✓		
	AWS Lambda	✓		
	AWS Elemental MediaConvert	✓		
	Amazon SageMaker	✓		

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		
AWS SDK 통합	200개 초과	✓		

지원되는 리전

대부분의 AWS 지역에서 Step Functions를 지원합니다. Step Functions를 사용할 수 있는 전체 AWS 지역 목록은 AWS 지역 표를 참조하십시오.

Step Functions를 처음 사용하시나요?

Step Functions를 처음 사용하는 경우 다음 항목을 통해 Step Functions를 다른 AWS 서비스와 결합하는 방법을 비롯하여 Step Functions를 사용하는 다양한 부분을 이해할 수 있습니다.

- Step Functions 자습서
- Step Functions를 위한 샘플 프로젝트
- AWS Step Functions Python용 데이터 사이언스 SDK

시작하기 AWS Step Functions

Step Functions는 애플리케이션 워크플로를 일련의 이벤트 기반 단계로 정의할 수 있는 서버리스 오케스트레이션 서비스입니다. 워크플로의 각 단계를 상태라고 합니다. 워크플로를 정의하는 데 태스크 상태, Choice, Parallel 및 맵과 같은 상태를 가장 일반적으로 사용합니다. Task상태 내에서 Step Functions가 지원하는 AWS SDK 통합을 사용하고 AWS 서비스 워크플로에서 여러 개를 오케스트레이션할 수 있습니다.

주제

주요 개념

자습서를 시작하기 전에 다음 주요 Step Functions 용어의 문맥을 검토하세요.

용어	설명
워크플로	비즈니스 프로세스를 반영하는 경우가 많은 일련의 단계입니다.
상태	입력을 기반으로 결정을 내리고, 해당 입력으로 작업을 수행하고, 출력을 다른 상태로 전달할 수 있는 상태 머신의 개별 단계입니다. 자세한 정보는 상태을 참조하세요.

용어	설명
Workflow Studio	더 빠르게 워크플로 프로토타입을 제작하고 워크플로를 빌드할 수 있도록 도와주는 시각적 워크플로 디자이너입니다. 자세한 정보는 AWS Step Functions 워크플로 스튜디오을 참조하세요.
상태 시스템	워크플로의 개별 상태나 단계를 나타내는 JSON 텍스트와 필드(예: `StartAt`, `TimeoutSeconds` 및 `Version`)를 사용하여 정의한 워크플로입니다. 자세한 정보는 상태 시스템 구조을 참조하세요.
Amazon States Language	상태 머신을 정의하는 데 사용되는 JSON 기반의 구조화된 언어입니다. ASL을 사용하면 작업을 수행할 수 있는 상태 모음 (`Task`상태) 을 정의하고, 다음으로 전환할 상태 (`Choice`상태) 를 결정하고, 오류가 발생한 실행 중지 (상태) 를 결정합니다. `Fail` 자세한 정보는 Amazon States Language을 참조하세요.
입력 및 출력 구성	워크플로의 상태는 JSON 데이터를 입력으로 받고 일반적으로 JSON 데이터를 다음 상태에 대한 출력으로 전달합니다. Step Functions는 상태 간 데이터 흐름을 제어하는 필터를 제공합니다. 자세한 정보는 Step Functions에서 입력 및 출력 처리을 참조하세요.
서비스 통합	워크플로우에서 AWS 서비스 API 작업을 호출할 수 있습니다. 자세한 정보는 다른 서비스와 AWS Step Functions 함께 사용을 참조하세요.
서비스 통합 유형	• AWS SDK 통합 — 스테이트 머신에서 직접 2백 AWS 서비스 9,000개 이상의 API 작업을 호출하는 표준 방법입니다. • 최적화된 통합 — 특정 서비스와의 통화 및 데이터 교환을 간소화하는 맞춤형 통합. 예를 들어, Lambda Invoke는 이스케이프된 JSON `Payload` 문자열의 응답 필드를 JSON 객체로 자동 변환합니다.

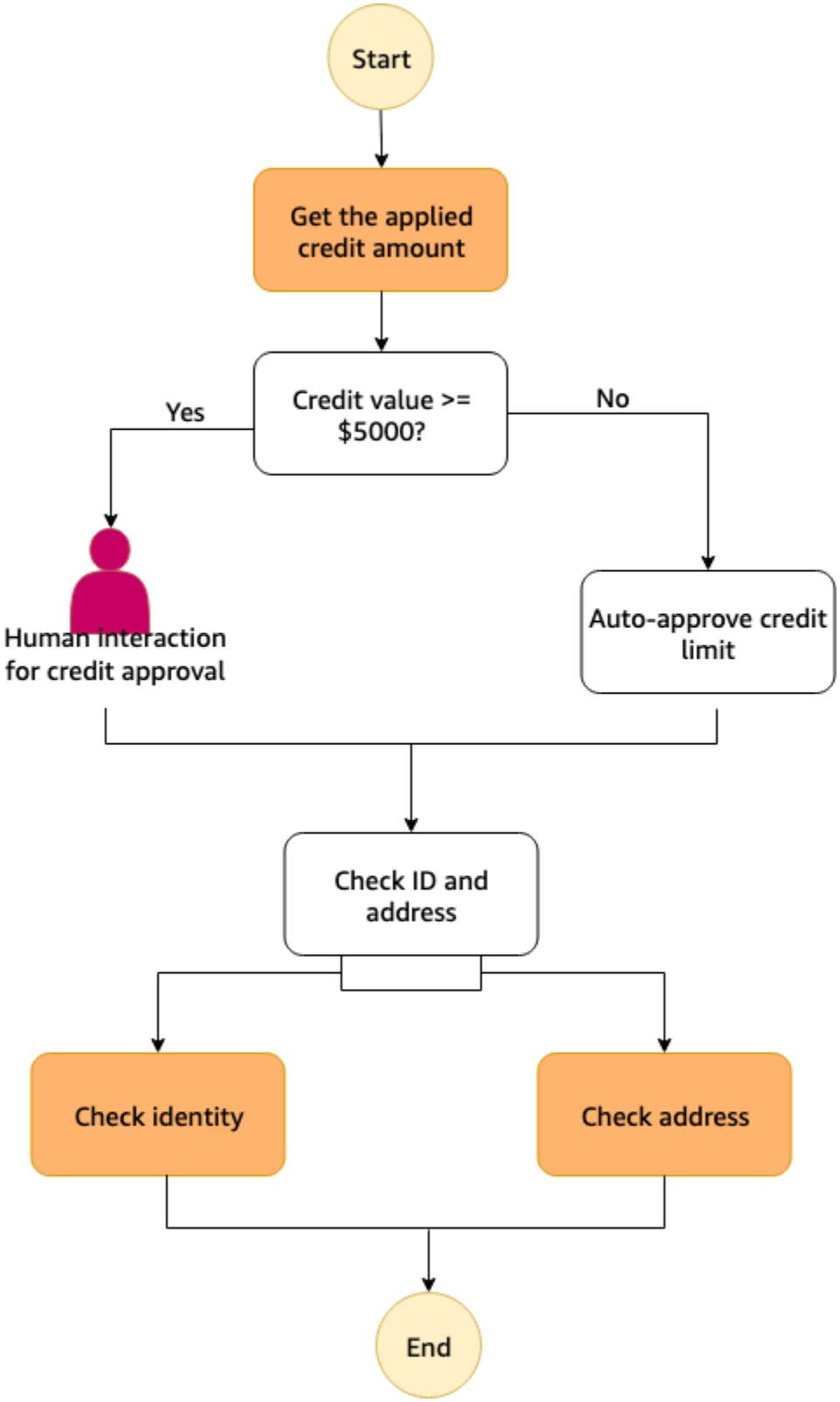
용어	설명
서비스 통합 패턴	를 호출할 때는 다음 AWS 서비스서비스 통합 패턴 중 하나를 사용합니다. • 응답 요청 (기본값) - HTTP 응답을 받은 후 즉시 서비스를 호출하고 다음 상태로 이동합니다. • 작업 실행(.sync) - 서비스를 직접적으로 호출하고 Step Functions가 작업이 완료될 때까지 기다리도록 합니다. • 태스크 토큰 (.wait Token) 으로 콜백 대기 — 태스크 ForTask 토큰으로 서비스를 호출하고 콜백과 함께 태스크 토큰이 반환될 때까지 Step Functions가 대기하도록 합니다.
Execution	상태 시스템 실행은 워크플로를 실행하여 작업을 수행하는 인스턴스입니다. 자세한 정보는 Step Functions에서 실행을 참조하세요.

이 시리즈의 자습서

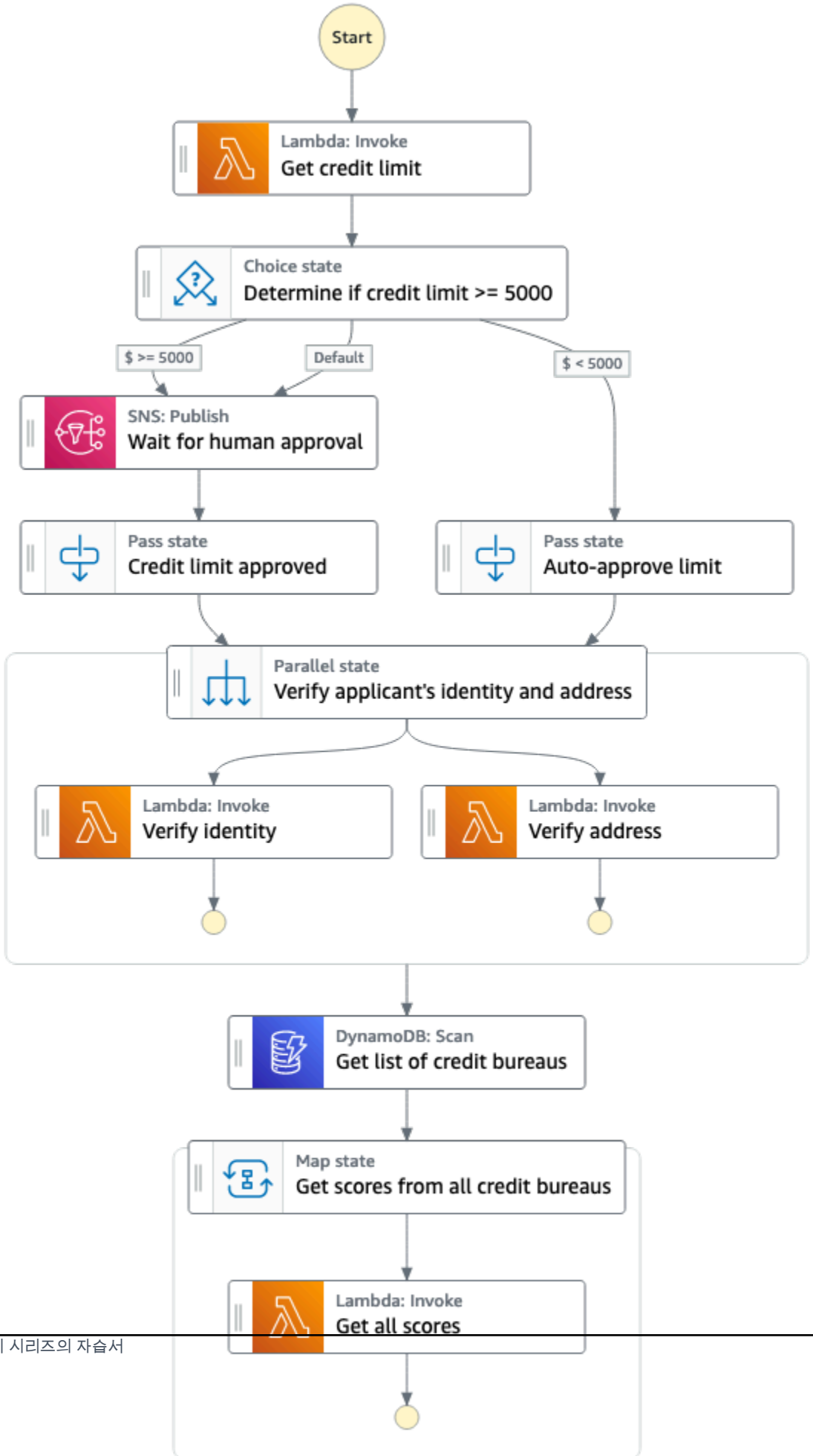
이 자습서를 완료하면 신용 카드 신청 처리를 시뮬레이션하는 워크플로를 갖게 됩니다. 공통 상태를 사용하고 워크플로를 다른 상태와 통합하는 방법을 배우게 됩니다. AWS 서비스

Step Functions를 사용하여 데이터 처리, IT 자동화, 기계 학습 및 미디어 인코딩과 같은 다양한 유형의 워크플로를 만들 수 있습니다.

다음 순서도는 기업이 신용 카드 신청을 처리하는 단계를 보여줍니다. 요청된 크레딧 금액이 $5000 미만인 경우 신용 한도가 자동으로 승인됩니다. 요청이 한도를 초과할 경우 워크플로우에서 요청자의 신원을 확인하고 신용 점수를 검토하는 담당자를 추가합니다.

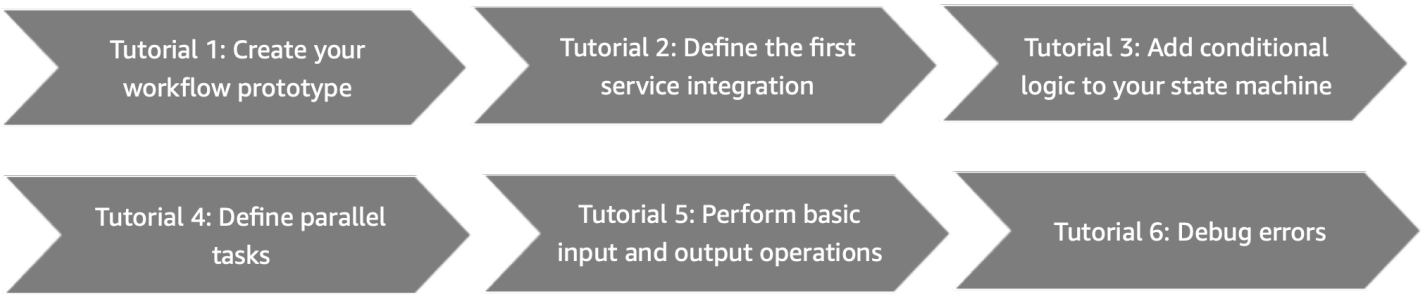


다음 다이어그램은 Step Functions 워크플로우에서 신용대출 신청 업무 프로세스 단계가 상태별로 표시되는 방식을 보여줍니다.


Start
Lambda: Invoke
Get credit limit
Choice state
Determine if credit limit >= 5000
$ >= 5000
Default
$ < 5000
SNS: Publish
Wait for human approval
Pass state
Credit limit approved
Pass state
Auto-approve limit
Parallel state
Verify applicant's identity and address
Lambda: Invoke
Verify identity
Lambda: Invoke
Verify address
DynamoDB: Scan
Get list of credit bureaus
Map state
Get scores from all credit bureaus
Lambda: Invoke
Get all scores

다음 자습서 시리즈에서는 신용 카드 처리 워크플로를 구축해 보겠습니다.

Step Functions의 주요 기능을 익히려면 다음 자습서를 완료하는 것이 좋습니다.



시작하기 전에 사전 조건이 모두 달성되어야 합니다.

시작하기 위한 사전 요구 사항 AWS Step Functions

AWS Step Functions 처음 사용하기 전에 다음 작업을 완료하십시오.

가입하여 AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

등록한 AWS 계정후에는 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 보호하고 AWS IAM Identity Center활성화하고 생성하십시오 AWS 계정 루트 사용자.

보안을 유지하세요 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 AWS Management Console소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하십시오.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자 (콘솔) 에 대한 가상 MFA 디바이스 활성화를 참조하십시오.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 ID 소스로 사용하는 방법에 대한 자습서는 사용 설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM IDentity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 사용 설명서의 AWS 액세스 포털에 로그인을 참조하십시오.AWS 로그인

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

자습서 1: 상태 시스템의 프로토타입 만들기

이 자습서에서는 Step Functions의 Workflow Studio를 사용하여 신용 카드 처리 워크플로의 프로토타입을 만듭니다. 작업 및 흐름 탭에서 각각 필요한 API 작업과 상태를 선택하고 Workflow Studio의 끌어서 놓기 기능을 사용하여 워크플로 프로토타입을 만들 수 있습니다. 이후 자습서에서는 이 워크플로에서 사용할 AWS 서비스 및 Step Functions 상태를 구성하는 방법을 알아봅니다.

상태 시스템 프로토타입 만들기

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택합니다. 디자인 모드에서 Workflow Studio가 열립니다.
4. Workflow Studio의 작업 탭에서 AWS Lambda Invoke API 작업을 끌어 첫 번째 상태를 여기에 놓기 레이블이 지정된 빈 상태에 놓습니다. 다음과 같이 구성합니다.
   - 구성 탭의 상태 이름에 **`Get credit limit`**를 입력합니다.
5. 흐름 탭에서 Choice 상태를 끌어 Get credit limit 상태 아래에 놓습니다. Choice 상태 이름을 **`Credit applied >= 5000?`**으로 바꿉니다.
6. 다음 상태를 Credit applied >= 5000? 상태 브랜치로 끌어서 놓습니다.
   a. Amazon SNS Publish - 작업 탭에서 Amazon SNS Publish API 작업을 끌어서 놓습니다. 이 상태 이름을 **`Wait for human approval`**로 바꿉니다.
   b. Pass 상태 — 흐름 탭에서 Pass 상태를 끌어서 놓습니다. 이 브랜치 이름을 **`Auto-approve limit`**로 바꿉니다.
7. Pass 상태를 끌어 사용자 Wait for human approval 상태 아래에 놓습니다. 이 Pass 상태 이름을 **`Credit limit approved`**로 바꿉니다.
8. 다음과 같이 Parallel 상태를 끌어 Choice 상태 뒤에 놓습니다.
   a. Parallel 상태를 Credit limit approved 상태 뒤에 놓습니다.
   b. Parallel 상태 이름을 **`Verify applicant's identity and address`**로 바꿉니다.
   c. Parallel 상태의 두 브랜치 모두에서 AWS Lambda Invoke API 작업 두 개를 끌어서 놓습니다.
   d. 이러한 상태 이름을 각각 **`Verify identity`** 및 **`Verify address`**로 바꿉니다.
   e. Auto-approve limit 상태를 선택하고 다음 상태에 Verify applicant's identity and address를 선택합니다.

9. DynamoDB Scan 상태를 끌어 Verify applicant's identity and address 상태 아래에 놓습니다. DynamoDB Scan 상태 이름을 **`Get list of credit bureaus`**로 변경합니다.
10. Map 상태를 끌어 Get list of credit bureaus 뒤에 놓습니다. 다음과 같이 Map 상태를 구성합니다.
    a. 이름을 **`Get scores from all credit bureaus`**로 바꿉니다.
    b. 처리 모드에서 기본 선택 항목인 인라인을 그대로 둡니다.
    c. AWS Lambda Invoke API 작업을 끌어 여기에 상태를 놓기 레이블이 지정된 빈 상태에 놓습니다.
    d. AWS Lambda Invoke 상태 이름을 **`Get all scores`**로 바꿉니다.
11. 이 창을 열어 두고 다음 자습서로 진행하여 추가 작업을 수행합니다.

다음 단계

다음 자습서에서는 Get credit limit 상태에서 사용하는 Lambda 함수를 통합하는 방법을 알아봅니다.

자습서 2: Lambda 함수를 사용하여 첫 번째 서비스 통합 정의

이 자습서에서는 워크플로의 첫 번째 서비스 통합을 정의하는 방법을 알아봅니다. Get credit limit라는 `Task` 상태를 사용하여 Lambda 함수를 간접적으로 호출합니다. `Task`상태 내에서 Step Functions가 AWS 지원하는 SDK 통합을 사용할 수 있습니다.

워크플로의 첫 번째 서비스 통합을 정의하려면 먼저 Lambda 함수를 만듭니다. 그런 다음 워크플로를 업데이트하여 Lambda 함수와의 서비스 통합을 지정합니다. 이 자습서에서 사용되는 Lambda 함수는 신청자가 신청한 신용 한도를 나타내는 무작위로 생성된 정수를 반환합니다.

주제

1단계: Lambda 함수 만들기 및 테스트

AWS Management Console 또는 선호하는 편집기에서 함수 코드를 작성할 수 있습니다. 다음 단계에서는 `RandomNumberforCredit`이라는 Node.js Lambda 함수를 만듭니다.

> **Important**
>
> 자습서 1에서 생성한 워크플로 프로토타입이 이 자습서에서 생성할 Lambda 함수와 AWS 리전 동일한지 확인하십시오.

1. 새 탭이나 창에서 Lambda 콘솔을 열고 **RandomNumberforCredit**이라는 Node.js 16.x Lambda 함수를 만듭니다. 콘솔을 사용하여 Lambda 함수를 만드는 방법은 AWS Lambda 개발자 안내서의 콘솔에서 Lambda 함수 생성을 참조하세요.
2. RandomNumberforCredit페이지에서 index.mjs를 선택하고 코드 소스 영역의 기존 코드를 다음 코드로 대체합니다.

```
export const handler = async function(event, context) {

    const credLimit = Math.floor(Math.random() * 10000);
    return (credLimit);

};
```

3. 함수 개요 섹션에서 Lambda 함수의 Amazon 리소스 이름을 복사하고 텍스트 파일에 저장합니다. Get credit limit 상태에 대한 서비스 통합을 지정하는 동안에 함수 ARN이 필요합니다. ARN 예제는 다음과 같습니다.

```
arn:aws:lambda:us-east-2:123456789012:function:HelloWorld
```

4. 배포를 선택한 다음 테스트를 선택하여 변경 사항을 배포하고 Lambda 함수 출력을 확인합니다.

2단계: 워크플로 업데이트 - Get credit limit 상태 구성

Step Functions 콘솔에서 워크플로를 업데이트하여 1단계에서 만든 `RandomNumberforCredit` Lambda 함수와의 서비스 통합을 지정합니다.

1. 자습서 1에서 만든 워크플로 프로토타입이 포함된 Step Functions 콘솔 창을 엽니다.
2. Get credit limit 상태를 선택하고 구성 탭에서 다음을 수행합니다.
   a. 통합 유형에 기본 선택 항목인 최적화를 그대로 둡니다.

Step Functions를 사용하면 다른 사람과 AWS 서비스 통합하고 워크플로에서 오케스트레이션할 수 있습니다. 서비스 통합과 해당 유형에 대한 자세한 내용은 다른 서비스와 AWS Step Functions 함께 사용 섹션을 참조하세요.

b. 함수 이름의 경우 드롭다운 목록에서 RandomNumberforCreditLambda 함수를 선택합니다.

c. 나머지 항목에 대해서는 기본 선택 항목을 그대로 둡니다.

3. 이 창을 열어 두고 다음 자습서로 진행하여 추가 작업을 수행합니다.

> **Note**
>
> 이 자습서에서 워크플로의 Task 상태 내에서 Lambda 함수와 통합하는 방법을 배웠습니다. 다음 구문과 같이 서비스 이름과 API 호출을 지정하여 해당 Task 주에서 지원되는 다른 AWS SDK 통합을 사용할 수도 있습니다.
>
> ```
> arn:aws:states:::aws-sdk:serviceName:apiAction
> ```
>
> 자세한 정보는 다른 서비스와 AWS Step Functions 함께 사용을 참조하세요.

다음 단계

다음 자습서에서는 워크플로에 조건부 논리를 구현합니다. Step Functions 상태 시스템의 조건부 논리는 대부분의 일반 프로그래밍 언어의 if-else 문과 유사하게 동작합니다. 워크플로에서 조건부 논리를 사용하여 조건부 정보를 기반으로 실행 경로를 결정합니다.

자습서 3: 워크플로에 if-else 조건 구현

Choice 상태를 사용하여 워크플로에 if-else 조건을 구현할 수 있습니다. 지정된 조건이 true 또는 false로 평가되는지에 따라 워크플로 실행 경로를 결정합니다.

이 자습서에서는 자습서 2에서 사용된 RandomNumberforCredit Lambda 함수가 반환한 적용된 크레딧 금액이 특정 임계값 한도를 초과하는지 확인하는 조건부 논리를 추가합니다. 금액이 임계값 한도를 초과하는 경우 사람이 신청을 승인해야 합니다. 그렇지 않으면 신청이 자동 승인되고 다음 단계로 이동합니다.

작업 토큰이 반환될 때까지 워크플로 실행을 일시 중지하여 인적 상호 작용 단계를 모방합니다. 이렇게 하려면 이 자습서에서 사용할 AWS SDK 통합, 즉 Amazon Simple Notification Service에 작업 토큰을

전달합니다. SendTaskSuccess API 직접 호출을 통해 작업 토큰을 다시 수신할 때까지 워크플로 실행이 일시 중지됩니다. 작업 토큰 사용에 대한 자세한 내용은 작업 토큰을 사용하여 콜백 대기를 참조하세요.

워크플로 프로토타입에 사람의 승인 및 자동 승인 단계를 이미 정의했으므로 이 자습서에서는 먼저 콜백 토큰을 수신하는 Amazon SNS 주제를 만듭니다. 그런 다음 Lambda 함수를 만들어 콜백 기능을 구현합니다. 마지막으로 이러한 AWS 서비스 통합의 세부 정보를 추가하여 워크플로 프로토타입을 업데이트합니다.

주제

1단계: 콜백 토큰을 수신하는 Amazon SNS 주제 만들기

인적 상호 작용 단계를 구현하려면 Amazon Simple Notification Service 주제에 게시하고 콜백 작업 토큰을 이 주제에 전달합니다. 콜백 작업은 작업 토큰이 페이로드와 함께 반환될 때까지 워크플로 실행을 일시 중지합니다.

1. Amazon SNS 콘솔을 열고 표준 주제 유형을 만듭니다. 주제를 만드는 방법은 Amazon Simple Notification Service 개발자 가이드의 Amazon SNS 주제 생성을 참조하세요.
2. 주제 이름을 **`TaskTokenTopic`**으로 지정합니다.
3. 주제 ARN을 복사하여 텍스트 파일로 저장해야 합니다. 사용자 Wait for human approval 상태에 대한 서비스 통합을 지정하는 동안에 주제 ARN이 필요합니다. 주제 ARN 예제는 다음과 같습니다.

   ```
   arn:aws:sns:us-east-2:123456789012:TaskTokenTopic
   ```

4. 주제에 대한 이메일 기반 구독을 만든 다음 구독을 확인합니다. 주제 구독 방법은 Amazon Simple Notification Service 개발자 안내서의 주제 구독 생성을 참조하세요.

2단계: Lambda 함수를 만들어 콜백 처리

콜백 기능을 처리하려면 Lambda 함수를 정의하고 1단계에서 만든 Amazon SNS 주제를 이 함수의 트리거로 추가합니다. 작업 토큰을 사용하여 Amazon SNS 주제에 게시하면 Lambda 함수가 게시된 메시지의 페이로드와 함께 간접적으로 호출됩니다.

2.1단계: Lambda 함수를 만들어 콜백 처리

이 함수에서는 크레딧 한도 승인 요청을 처리하고 `SendTaskSuccess` API 직접 호출을 사용하여 요청 결과를 성공으로 반환합니다. 이 Lambda 함수는 Amazon SNS 주제로부터 받은 작업 토큰도 반환합니다.

간소화되도록 인적 상호 작용 단계에 사용된 Lambda 함수는 모든 작업을 자동으로 승인하고 `SendTaskSuccess` API 직접 호출을 사용하여 작업 토큰을 반환합니다. Lambda 함수 이름을 **`callback-human-approval`**로 지정할 수 있습니다.

1. 새 탭이나 창에서 Lambda 콘솔을 열고 **`callback-human-approval`**이라는 Node.js 16.x Lambda 함수를 만듭니다. 콘솔을 사용하여 Lambda 함수를 만드는 방법은 AWS Lambda 개발자 안내서의 콘솔에서 Lambda 함수 생성을 참조하세요.
2. callback-human-approval 페이지에서 코드 소스 영역의 기존 코드를 다음 코드로 바꿉니다.

```
// Sample Lambda function that will automatically approve any task whenever a
 message is published to an Amazon SNS topic by Step Functions.

console.log('Loading function');
const AWS = require('aws-sdk');
const resultMessage = "Successful";

exports.handler = async (event, context) => {
    const stepfunctions = new AWS.StepFunctions();

    let message = JSON.parse(event.Records[0].Sns.Message);
    let taskToken = message.TaskToken;

    console.log('Message received from SNS:', message);
    console.log('Task token: ', taskToken);
```

```
    // Return task token to Step Functions

    let params = {
        output: JSON.stringify(resultMessage),
        taskToken: taskToken
    };

    console.log('JSON Returned to Step Functions: ', params);
    let myResult = await stepfunctions.sendTaskSuccess(params).promise();
    console.log('State machine - callback completed..');

    return myResult;

};
```

3. 이 창을 열어 두고 다음 섹션의 단계를 수행하여 추가 작업을 수행합니다.

2.2단계: Amazon SNS 주제를 Lambda 함수의 트리거로 추가

이 자습서의 1단계에서 만든 Amazon SNS 주제를 이 자습서의 2.1단계에서 만든 Lambda 함수의 트리거로 추가하면 Amazon SNS 주제에 게시할 때마다 Lambda 함수가 트리거됩니다. 작업 토큰을 사용하여 Amazon SNS 주제에 게시하면 Lambda 함수가 게시된 메시지의 페이로드와 함께 간접적으로 호출됩니다. Lambda 함수의 트리거 구성에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 트리거 구성을 참조하세요.

1. `callback-human-approval` Lambda 함수의 함수 개요 섹션에서 트리거 추가를 선택합니다.
2. 트리거 드롭다운 목록에서 SNS를 트리거로 선택합니다.
3. SNS 주제에 이 자습서의 1단계에서 만든 Amazon SNS 주제 이름을 입력하고 나타나는 드롭다운 목록에서 주제를 선택합니다.
4. 추가(Add)를 선택합니다.
5. 이 창을 열어 두고 다음 섹션의 단계를 수행하여 추가 작업을 수행합니다.

2.3단계: Lambda 함수 IAM 역할에 필요한 권한 제공

`SendTaskSucess` API 직접 호출을 사용하여 작업 토큰을 반환할 수 있도록 Step Functions에 액세스할 수 있는 권한을 `callback-human-approval` Lambda 함수에 제공해야 합니다.

1. callback-human-approval 페이지에서 구성 탭을 선택한 다음 권한을 선택합니다.

2. 실행 역할에서 역할 이름을 선택하여 AWS Identity and Access Management 콘솔의 역할 페이지로 이동합니다.
3. 필수 권한을 추가하라면 권한 추가, 정책 연결을 차례로 선택합니다.
4. 검색 상자에 **AWSStepFunctions**을 입력한 다음 Enter 키를 누릅니다.
5. AWSStepFunctionsFullAccess를 선택한 다음 아래로 스크롤하여 정책 연결을 선택합니다. 그러면 `callback-human-approval` Lambda 함수 역할에 필요한 권한이 포함된 정책이 추가됩니다.

3단계: 워크플로 업데이트 - Choice 상태에 if-else 조건부 논리 추가

Step Functions 콘솔에서 `Choice` 상태를 사용하여 워크플로의 조건부 논리를 정의합니다. `RandomNumberforCredit` Lambda 함수에서 반환한 출력이 5000 미만이면 요청된 크레딧이 자동 승인됩니다. 반환된 출력이 5000보다 크거나 같으면 워크플로 실행은 크레딧 한도 승인을 위한 인적 상호 작용 단계로 진행됩니다.

`Choice` 상태에서는 비교 연산자를 사용하여 입력 변수를 특정 값과 비교합니다. 상태 시스템 실행을 시작하는 동안 입력 변수를 실행 입력으로 지정하거나 이전 단계의 출력을 현재 단계의 입력으로 사용할 수 있습니다. 기본적으로 단계 출력은 `Payload` 변수에 저장됩니다. `Payload` 변수 값을 사용하여 `Choice` 상태를 비교하려면 다음 절차와 같이 `$` 구문을 사용합니다.

정보가 한 상태에서 다른 상태로 이동하는 방식과 워크플로에 입력과 출력을 지정하는 방법은 자습서 7: 입력 및 출력 구성 및 Step Functions에서 입력 및 출력 처리를 참조하세요.

> **Note**
>
> `Choice` 상태에서 비교하기 위해 상태 시스템 실행 입력에 지정된 입력 변수를 사용하는 경우 `$.variable_name` 구문을 사용하여 비교를 수행합니다. 예를 들어 변수를 비교하려면(예: `myAge`) `$.myAge` 구문을 사용합니다.

이 단계에서 `Choice` 상태는 Get credit limit 상태로부터 입력을 수신하므로 `Choice` 상태 구성에 `$` 구문을 사용합니다. `Choice` 상태 구성의 `$.variable_name` 구문을 사용하여 이전 단계의 출력을 참조할 때 상태 시스템 실행 결과가 어떻게 달라지는지 살펴보려면 자습서 8의 잘못된 경로 Choice 상태 오류 디버깅 섹션을 참조하세요.

`Choice` 상태를 사용하여 if-else 조건부 논리 추가하기

1. 자습서 1: 상태 시스템의 프로토타입 만들기에서 만든 워크플로 프로토타입이 포함된 Step Functions 콘솔 창을 엽니다.

2. Credit applied >= 5000? 상태를 선택하고 구성 탭에서 다음과 같이 조건부 로직을 지정합니다.

   a. 선택 규칙의 규칙 #1 타일에서 편집 아이콘을 선택하여 첫 번째 선택 규칙을 정의합니다.

   b. 조건 추가를 선택합니다.

   c. 규칙 #1 조건 대화 상자의 변수에 **$**를 입력합니다.

   d. 연산자에 미만을 선택합니다.

   e. 값에 숫자 상수를 선택한 다음 값 드롭다운 목록 옆에 있는 필드에 **5000**을 입력합니다.

   f. 조건 저장을 선택합니다.

   g. 다음 상태: 드롭다운 목록에 Auto-approve limit를 선택합니다.

   h. 새 선택 규칙 추가를 선택한 다음, 2.b~2.f 하위 단계를 반복하여 크레딧 금액이 5000보다 크거나 같을 때 두 번째 선택 규칙을 정의합니다. 연산자에서 크거나 같음을 선택합니다.

   i. 다음 상태: 드롭다운 목록에서 Wait for human approval을 선택합니다.

   j. 기본 규칙 상자에서 편집 아이콘을 선택하여 기본 선택 규칙을 정의한 다음 기본 상태 드롭다운 목록에서 Wait for human approval을 선택합니다. 기본 규칙을 정의하여 Choice 상태 조건 중 어느 것도 true 또는 false로 평가되지 않으면 전환할 다음 상태를 지정합니다.

3. Wait for human approval 상태를 다음과 같이 구성합니다.

   a. 구성 탭에서 주제에 Amazon SNS 주제 이름인 TaskTokenTopic을 입력하고 드롭다운 목록에 나타나는 이름을 선택합니다.

   b. 메시지의 드롭다운 목록에서 메시지 입력을 선택합니다. 메시지 필드에서 Amazon SNS 주제에 게시할 메시지를 지정합니다. 이 자습서에서는 작업 토큰을 메시지로 게시합니다.

      작업 토큰을 사용하면 외부 프로세스가 완료되고 작업 토큰이 반환될 때까지 표준 유형 Step Functions 워크플로를 일시 중지할 수 있습니다. `.waitForTaskToken` 서비스 통합 패턴을 지정하여 Task 상태를 콜백 작업으로 지정하면 작업 토큰이 생성되고 작업이 시작될 때 컨텍스트 객체에 배치됩니다. 컨텍스트 객체는 실행 중에 사용할 수 있는 내부 JSON 구조로, 여기에 상태 시스템과 실행에 대한 정보가 포함됩니다. 컨텍스트 객체에 대한 자세한 내용은 컨텍스트 객체를 참조하세요.

   c. 표시되는 상자에 다음을 메시지로 입력합니다.

      ```
      {
        "TaskToken.$": "$$.Task.Token"
      }
      ```

   d. 콜백 대기 확인란을 선택합니다.

   e. 나타나는 대화 상자에서 완료를 선택합니다.

4. 이 창을 열어 두고 다음 자습서로 진행하여 추가 작업을 수행합니다.

다음 단계

다음 자습서에서는 여러 작업을 동시에 수행하는 방법을 알아봅니다.

자습서 4: 동시에 수행할 여러 작업 정의

지금까지 워크플로를 순차적으로 실행하는 방법을 배웠습니다. 하지만 `Parallel` 상태를 사용하여 단계 2개 이상을 동시에 실행할 수 있습니다. `Parallel` 상태로 인해 인터프리터가 각 브랜치를 동시에 실행합니다.

`Parallel` 상태의 두 브랜치 모두 같은 입력을 수신하지만 각 브랜치는 해당 상태에 맞는 입력 부분을 처리합니다. Step Functions 함수는 각 브랜치 실행이 완료될 때까지 기다린 후 다음 단계를 진행합니다.

이 자습서에서는 Parallel 상태를 사용하여 신청자의 신원과 주소를 동시에 확인합니다.

주제

- 1단계: Lambda 함수를 만들어 필수 검사 수행
- 2단계: 워크플로 업데이트 - 수행할 병렬 작업 추가

1단계: Lambda 함수를 만들어 필수 검사 수행

이 신용 카드 신청 워크플로는 Parallel 상태 내에서 Lambda 함수 2개를 간접적으로 호출하여 신청자의 신원과 주소를 확인합니다. 이러한 검사는 Parallel 상태를 통해 동시에 수행됩니다. 상태 시스템은 두 병렬 브랜치 모두 실행을 완료한 후에만 실행을 완료합니다.

check-identity 및 check-address Lambda 함수 만들기

1. 새 탭이나 창에서 Lambda 콘솔을 열고 `check-identity` 및 `check-address`라는 Node.js 16.x Lambda 함수 2개를 만듭니다. 콘솔을 사용하여 Lambda 함수를 만드는 방법은 AWS Lambda 개발자 안내서의 콘솔에서 Lambda 함수 생성을 참조하세요.
2. check-identity 함수 페이지를 열고 코드 소스 영역의 기존 코드를 다음 코드로 바꿉니다.

```
const ssnRegex = /^\d{3}-?\d{2}-?\d{4}$/;
const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
```

```
class ValidationError extends Error {
    constructor(message) {
        super(message);
        this.name = "CustomValidationError";
    }
}

exports.handler = async (event) => {
    const {
        ssn,
        email
    } = event;
    console.log(`SSN: ${ssn} and email: ${email}`);

    const approved = ssnRegex.test(ssn) && emailRegex.test(email);

    if (!approved) {
        throw new ValidationError("Check Identity Validation Failed");
    }

    return {
        statusCode: 200,
        body: JSON.stringify({
            approved,
            message: `Identity validation ${approved ? 'passed' : 'failed'}`
        })
    }
};
```

3. check-address 함수 페이지를 열고 코드 소스 영역의 기존 코드를 다음 코드로 바꿉니다.

```
class ValidationError extends Error {
  constructor(message) {
      super(message);
      this.name = "CustomAddressValidationError";
  }
}

exports.handler = async event => {
  const {
      street,
      city,
      state,
      zip
```

```
  } = event;
  console.log(`Address information: ${street}, ${city}, ${state} - ${zip}`);

  const approved = [street, city, state, zip].every(i => i?.trim().length > 0);

  if (!approved) {
      throw new ValidationError("Check Address Validation Failed");
  }

  return {
      statusCode: 200,
      body: JSON.stringify({
          approved,
          message: `Address validation ${ approved ? 'passed' : 'failed'}`
      })
  }
};
```

4. 함수 개요 섹션에서 두 Lambda 함수의 Amazon 리소스 이름(ARN)을 각각 복사하고 텍스트 파일에 저장합니다. Verify applicant's identity and address 상태에 대한 서비스 통합을 지정하는 동안에 함수 ARN이 필요합니다. ARN 예제는 다음과 같습니다.

```
arn:aws:lambda:us-east-2:123456789012:function:HelloWorld
```

2단계: 워크플로 업데이트 - 수행할 병렬 작업 추가

Step Functions 콘솔에서 워크플로를 업데이트하여 1단계에서 만든 check-identity 및 check-address Lambda 함수와의 서비스 통합을 지정합니다.

워크플로에 Parallel 작업 추가하기

1. 자습서 1: 상태 시스템의 프로토타입 만들기에서 만든 워크플로 프로토타입이 포함된 Step Functions 콘솔 창을 엽니다.
2. Verify identity를 선택하고 구성 탭에서 다음을 수행합니다.

   a. 통합 유형에 기본 선택 항목인 최적화를 그대로 둡니다.

> **Note**
>
> Step Functions를 사용하면 다른 AWS 서비스와 통합하고 워크플로에서 오케스트레이션할 수 있습니다. 서비스 통합과 유형에 대한 자세한 내용은 다른 서비스와 AWS Step Functions 함께 사용을 참조하세요.

b. 함수 이름의 드롭다운 목록에서 check-identity Lambda 함수를 선택합니다.

c. 페이로드에 페이로드 입력을 선택한 다음 예제 페이로드를 다음과 같은 페이로드로 바꿉니다.

```
{
    "email": "janedoe@example.com",
    "ssn": "012-00-0000"
}
```

3. Verify address를 선택하고 구성 탭에서 다음을 수행합니다.

a. 통합 유형에 기본 선택 항목인 최적화를 그대로 둡니다.

b. 함수 이름의 드롭다운 목록에서 check-address Lambda 함수를 선택합니다.

c. 페이로드에 페이로드 입력을 선택한 다음 예제 페이로드를 다음과 같은 페이로드로 바꿉니다.

```
{
    "street": "123 Any St",
    "city": "Any Town",
    "state": "AT",
    "zip": "01000"
}
```

4. 다음(Next)을 선택합니다.

자습서 5: 항목 컬렉션을 동시에 반복하기

이전 자습서에서는 `Parallel` 상태를 사용하여 별도의 단계 브랜치를 동시에 실행하는 방법을 배웠습니다. `Map` 상태를 사용하면 데이터 세트의 항목마다 일련의 워크플로 단계를 실행할 수 있습니다. `Map` 상태 반복은 동시에 실행되므로 데이터 세트를 빠르게 처리할 수 있습니다.

워크플로에 Map 상태를 포함하면 인라인 모드와 분산 모드 등 두 Map 상태 처리 모드 중 하나를 사용하여 데이터 처리와 같은 작업을 수행할 수 있습니다. Map 상태를 구성하려면 Map 상태 처리 모드 및 정의를 지정하는 JSON 객체가 포함된 `ItemProcessor`를 정의합니다. 이 자습서에서는 동시 반복을 최대 40개까지 지원하는 기본 인라인 모드에서 Map 상태를 실행합니다. 분산 모드에서 Map 상태를 실행하면 병렬 하위 워크플로 실행이 최대 10,000개까지 지원됩니다.

워크플로 실행이 Map 상태로 전환되면 상태 입력에 지정된 JSON 배열이 반복됩니다. 배열 항목마다 반복은 Map 상태가 포함된 워크플로의 컨텍스트에서 실행됩니다. 모든 반복이 완료되면 Map 상태는 `ItemProcessor`에서 처리한 각 항목의 출력이 포함된 배열을 반환합니다.

이 자습서에서는 인라인 모드에서 Map 상태를 사용하여 신용 조사 기관 세트를 반복해 신청자의 신용 점수를 가져오는 방법을 알아봅니다. 이렇게 하려면 먼저 Amazon DynamoDB 테이블에 저장된 모든 신용 조사 기관의 이름을 가져온 다음 Map 상태를 사용하여 신용 조사 기관 목록을 반복해 각 신용 조사 기관에서 보고한 신청자의 신용 점수를 가져옵니다.

주제

1단계: 모든 신용 조사 기관의 이름을 저장할 DynamoDB 테이블 만들기

이 단계에서는 DynamoDB 콘솔을 사용하여 **`GetCreditBureau`** 테이블을 만듭니다. 테이블은 문자열 속성 이름을 파티션 키로 사용합니다. 이 테이블에 신청자의 신용 점수를 가져오려는 모든 신용 조사 기관의 이름을 저장합니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/dynamodb/에서 DynamoDB 콘솔을 엽니다.
2. 콘솔의 탐색 창에서 테이블을 선택한 다음 테이블 생성을 선택합니다.
3. 다음과 같이 테이블 세부 정보를 입력합니다.
   a. 테이블 이름에 **`GetCreditBureau`**을 입력합니다.
   b. 파티션 키에 **`Name`**을 입력합니다.
   c. 모든 기본 선택 항목을 그대로 두고 테이블 생성을 선택합니다.
4. 테이블이 생성되면 테이블 목록에서 GetCreditBureau 테이블을 선택합니다.

5. 작업, 항목 생성을 차례대로 선택합니다.
6. 값에 신용 조사 기관 이름을 입력합니다. 예: **CredTrack**.
7. 항목 생성(Create Item)을 선택합니다.
8. 이 프로세스를 반복하여 다른 신용 조사 기관 이름에 대한 항목을 만듭니다. 예: **KapFinn**, **CapTrust**.

2단계: 상태 시스템 업데이트 - DynamoDB 테이블에서 결과 가져오기

Step Functions 콘솔에서 Task 상태를 추가하고 AWS SDK 통합을 사용하여 1단계에서 만든 DynamoDB 테이블에서 신용 조사 기관 이름을 가져옵니다. 이 단계 출력을 이 자습서의 워크플로에서 나중에 추가할 Map 상태의 입력으로 사용합니다.

1. CreditCardWorkflow 상태 시스템을 열어 업데이트합니다.
2. Get list of credit bureaus 상태를 선택합니다.
3. API 파라미터에 테이블 이름 값을 **GetCreditBureau**로 지정합니다.

3단계: 모든 신용 조사 기관의 신용 점수를 반환하는 Lambda Functions 만들기

이 단계에서는 모든 신용 조사 기관 이름을 입력으로 수신하고 각 신용 조사 기관의 신청자 신용 점수를 반환하는 Lambda 함수를 만듭니다. 이 Lambda 함수는 이 자습서의 4단계에서 워크플로에 추가할 Map 상태에서 간접적으로 호출됩니다.

1. Node.js 16.x Lambda 함수를 만들고 이름을 **get-credit-score**로 지정합니다.
2. get-credit-score 페이지에서 다음 코드를 코드 소스 영역에 붙여넣습니다.

```
function getScore(arr) {
  let temp;
  let i = Math.floor((Math.random() * arr.length));
  temp = arr[i];
  console.log(i);
  console.log(temp);
  return temp;
}

const arrScores = [700, 820, 640, 460, 726, 850, 694, 721, 556];
```

```
exports.handler = (event, context, callback) => {
  let creditScore = getScore(arrScores);
  callback(null, "Credit score pulled is: " + creditScore + ".");
};
```

3. Lambda 함수를 배포합니다.

4단계: 상태 시스템 업데이트 - 신용 점수를 반복적으로 가져오도록 Map 상태 추가

Step Functions 콘솔에서 get-credit-score Lambda 함수를 간접적으로 호출하는 Map 상태를 추가하여 Get list of credit bureaus 상태에서 반환한 모든 신용 조사 기관의 신청자 신용 점수를 확인합니다.

1. CreditCardWorkflow 상태 시스템을 열어 업데이트합니다.
2. Get scores from all credit bureaus 상태를 선택합니다.
3. 구성 탭에서 항목 배열에 대한 경로 제공을 선택한 다음 **`$.Items`**를 입력합니다.
4. Map 상태 내에서 Get all scores 단계를 선택합니다.
5. 구성 탭에서 통합 유형에 최적화가 선택되어 있는지 확인합니다.
6. 함수 이름에 get-credit-score Lambda 함수의 이름을 입력하고 표시되는 드롭다운 목록에서 선택합니다.
7. 페이로드에 페이로드 없음을 선택합니다.

자습서 6: 워크플로 저장 및 상태 시스템 실행

이제 워크플로 프로토타입에서 사용 중인 모든 리소스의 구성을 마쳤으므로 이를 Step Functions 상태 머신으로 저장하고 실행을 시작할 수 있습니다. AWS 서비스

주제

1단계: 자동으로 생성된 상태 시스템 정의 검토 및 상태 시스템 저장

흐름 탭에서 상태를 끌어 Workflow Studio의 캔버스로 놓아 워크플로 프로토타입을 빌드하면 Step Functions에서 워크플로의 Amazon States Language(ASL) 정의를 실시간으로 자동 구성합니다. 필요에 따라 코드 편집기에서 이 정의를 편집할 수 있습니다.

ASL 정의를 검토하고 상태 시스템 저장하기

1. (선택 사항) Inspector에서 정의를 선택하여 상태 시스템의 Amazon States Language(ASL) 정의를 봅니다. 이 정의는 작업 및 흐름 탭과 Inspector 패널에서의 선택 항목에 따라 자동으로 생성됩니다.

   > **Tip**
   >
   > 정의를 편집하려면 페이지 상단에 있는 코드를 선택하여 코드 편집기를 열면 됩니다. 이 자습서에서는 자동 생성된 정의를 계속 사용합니다.

2. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택합니다. MyStateMachine 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 튜토리얼에서는 이름 **`CreditCardWorkflow`**를 입력합니다.

3. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 상태 머신 설정의 모든 기본 선택 항목을 그대로 둡니다.

   > **Note**
   >
   > (선택 사항) Step Functions는 `RandomNumberforCredit` Lambda 함수를 간접적으로 호출하고 Amazon SNS 주제에 게시하는 데 필요한 최소 권한이 있는 상태 시스템의 실행 역할을 자동으로 만듭니다.

   상태 시스템에 대한 올바른 권한을 사용하여 이전에 IAM 역할을 만들었고 이를 사용하려면 권한에서 기존 역할 선택을 선택한 다음 목록에서 역할을 선택합니다. 또는 역할 ARN 입력을 선택한 다음 IAM 역할에 대한 ARN을 제공합니다.

4. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

2단계: 나머지 IAM 정책 추가

Step Functions는 `Parallel` 상태에서 사용되는 Lambda 함수를 간접적으로 호출할 수 있는 권한을 자동으로 생성하지 않으므로 필요한 정책을 추가해야 합니다.

나머지 정책 추가하기

1. CreditCardWorkflow페이지에서 상태 머신이 IAM 콘솔로 이동하는 데 사용할 IAM 역할을 선택합니다. 이 페이지에서 나머지 Lambda 함수에 필요한 권한을 추가합니다.
2. 권한 추가를 선택하고 정책 연결을 선택합니다.
3. 검색 상자에 **`AWSLambdaRole`**을 입력한 다음 Enter 키를 누릅니다.
4. 선택한 AWSLambdaRole다음 Attach 정책을 선택합니다. 이제 이 정책이 상태 시스템의 실행 역할에 추가됩니다. 이 정책을 사용하면 상태 시스템의 모든 Lambda 함수를 간접적으로 호출할 수 있습니다.

3단계: 상태 시스템 실행

상태 시스템 실행은 워크플로를 실행하여 작업을 수행하는 인스턴스입니다.

상태 시스템 실행하기

1. CreditCardWorkflow페이지에서 실행 시작을 선택합니다.

   실행 시작 대화 상자가 표시됩니다.
2. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

> **Note**
>
> Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

> **Note**
>
> 이 상태 시스템을 실행하기 위해 어떠한 입력도 제공할 필요가 없습니다. 하지만 필요한 경우 다른 상태 시스템에 대한 실행 시작 대화 상자의 입력 영역에서 실행 입력을 지정할 수 있습니다. 상태 머신에 실행 입력을 제공하는 방법에 대한 예는 4단계: AWS Step Functions Workflow Studio 사용 방법 배우기 자습서의 새 실행 시작을 참조하십시오.

b. 실행 시작을 선택합니다.

3. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

자습서 7: 입력 및 출력 구성

Step Functions 실행에서 JSON 텍스트를 입력으로 수신하고 해당 입력을 워크플로의 첫 번째 상태에 전달합니다. 워크플로의 개별 상태는 JSON 데이터를 입력으로 수신하고 일반적으로 JSON 데이터를 출력으로 다음 상태에 전달합니다. 워크플로에서 상태 하나 이상에 대한 입력 또는 출력을 구성하지 않으면 기본적으로 데이터는 워크플로의 한 상태에서 다음 상태로 전달됩니다. 정보가 상태에서 다른 상태로 흐르는 방식을 이해하고 이 데이터를 필터링하고 조작하는 방법을 알아보는 것이 Step Functions에서 워크플로를 효율적으로 설계하고 구현하는 데 중요합니다.

Step Functions는 상태 간 입력 및 출력 데이터 흐름을 제어하는 필터 여러 개를 제공합니다. 워크플로에서 사용할 수 있는 필터는 다음과 같습니다.

> **Note**
>
> 사용 사례에 따라 워크플로에 이러한 필드를 모두 적용하지 않아도 됩니다.

`InputPath`

작업 입력으로 사용할 전체 입력 페이로드의 WHAT 부분을 선택합니다. 이 필드를 지정하면 Step Functions에서 먼저 이 필드를 적용합니다.

`####`

작업을 간접적으로 호출하기 전과 같이 HOW를 지정합니다. `Parameters` 필드를 사용하여 입력으로 AWS 서비스 통합에 전달되는 키-값 페어 컬렉션을 만들 수 있습니다(예: AWS Lambda 함수). 이러한 값은 정적이거나 상태 입력 또는 워크플로 컨텍스트 객체에서 동적으로 선택한 값일 수도 있습니다.

`ResultSelector`

작업 출력에서 선택할 WHAT을 결정합니다. `ResultSelector` 필드를 사용하여 상태 결과를 바꾸는 키-값 페어 컬렉션을 만들고 해당 컬렉션을 `ResultPath`에 전달할 수 있습니다.

`ResultPath`

작업 출력을 배치할 WHERE를 결정합니다. `ResultPath`를 사용하여 상태 출력이 입력 사본인지, 생성되는 결과인지 아니면 이 둘의 조합인지 확인합니다.

`OutputPath`

다음 상태로 전송할 WHAT을 결정합니다. `OutputPath`를 사용하여 원치 않는 정보를 필터링하고 관심 있는 JSON 데이터 부분만 전달할 수 있습니다.

> **Tip**
>
> `Parameters` 및 `ResultSelector` 필터는 JSON을 구성하는 방식으로 작동하는 반면, `InputPath` 및 `OutputPath` 필터는 JSON 데이터 객체 내의 특정 노드를 필터링하는 방식으로 작동합니다. `ResultPath` 필터는 출력을 추가할 수 있는 필드를 만드는 방식으로 작동합니다.

이 자습서에서는 다음 작업을 수행하는 방법을 알아봅니다.

- 필터를 사용하여 원시 입력의 특정 부분을 선택합니다. InputPath
- 파라미터 필터를 사용하여 선택한 입력 조작
- ResultSelector, ResultPath 및 OutputPath 필터를 사용하여 출력 구성

워크플로에 입력과 출력을 구성하는 방법에 대한 자세한 내용은 Step Functions에서 입력 및 출력 처리을 참조하세요.

필터를 사용하여 원시 입력의 특정 부분을 선택합니다. InputPath

`InputPath` 필터를 사용하여 입력 페이로드의 특정 부분을 선택합니다.

`InputPath`를 지정하지 않으면 값이 `$`로 기본 설정됩니다. 이로 인해 상태 작업이 특정 부분이 아닌 전체 원시 입력을 참조합니다.

InputPath 필터 사용 방법을 알아보려면 다음 단계를 수행합니다.

- 1단계: 상태 시스템 만들기
- 2단계: 상태 시스템 실행
- 3단계: InputPath 필터를 사용하여 실행 입력의 특정 부분 선택

1단계: 상태 시스템 만들기

> **Important**
>
> 상태 머신이 이전에 생성한 Lambda 함수와 동일한 AWS 계정 및 지역에 속하는지 확인하십시오.

1. 자습서 4에서 배운 `Parallel` 상태 예제를 사용하여 새 상태 시스템을 만듭니다. 워크플로 프로토타입이 다음 프로토타입과 비슷한지 확인합니다.
2. `check-identity` 및 `check-address` Lambda 함수의 통합을 구성합니다. Lambda 함수를 만들고 상태 시스템에서 사용하는 방법은 1단계: Lambda 함수를 만들어 필수 검사 수행 및 2단계: 워크플로 업데이트 - 수행할 병렬 작업 추가 섹션을 참조하세요.
3. 페이로드의 경우 기본 선택 항목인 상태 입력을 페이로드로 사용을 그대로 둬야 합니다.
4. 다음을 선택하고 자습서 5의 1단계: 상태 시스템 저장 1~3단계를 수행하여 새 상태 시스템을 만듭니다. 이 자습서에서는 상태 시스템 이름을 **`WorkflowInputOutput`**으로 지정합니다.

2단계: 상태 시스템 실행

1. WorkflowInputOutput페이지에서 실행 시작을 선택합니다.
2. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

> **Note**
>
> Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

3. 입력 영역에 다음 JSON 데이터를 실행 입력으로 추가합니다.

```
{
  "data": {
    "firstname": "Jane",
    "lastname": "Doe",
    "identity": {
      "email": "jdoe@example.com",
      "ssn": "123-45-6789"
    },
    "address": {
      "street": "123 Main St",
      "city": "Columbus",
      "state": "OH",
      "zip": "43219"
    }
  }
}
```

4. 실행 시작을 선택합니다.
5. `check-identity` 및 `check-address` Lambda 함수에서 필수 ID 및 주소 확인을 수행하는 데 사용해야 하는 실행 입력 부분을 지정하지 않았으므로 상태 시스템 실행에서 오류가 발생합니다.
6. 이 자습서의 3단계를 계속 진행하여 오류를 수정합니다.

3단계: **InputPath** 필터를 사용하여 실행 입력의 특정 부분 선택

1. 실행 세부 정보 페이지에서 상태 머신 편집을 선택합니다.
2. 2단계: 상태 시스템 실행에 제공된 실행 입력에 언급된 대로 신청자 ID를 확인하려면 Verify identity 작업 정의를 다음과 같이 편집합니다.

```
...
{
    "StartAt": "Verify identity",
    "States": {
      "Verify identity": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "InputPath": "$.data.identity",
        "Parameters": {
          "Payload.$": "$",
          "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:check-
identity:$LATEST"
        },
        "End": true
      }
    }
  }
  ...
```

따라서 다음 JSON 데이터를 `check-identity` 함수의 입력으로 사용할 수 있게 됩니다.

```
{
   "email": "jdoe@example.com",
   "ssn": "123-45-6789"
 }
```

3. 실행 입력에 언급된 대로 신청자 주소를 확인하려면 다음과 같이 `Verify address` 작업 정의를 편집합니다.

```
...
{
    "StartAt": "Verify address",
    "States": {
      "Verify address": {
        "Type": "Task",
```

```
        "Resource": "arn:aws:states:::lambda:invoke",
        "InputPath": "$.data.address",
        "Parameters": {
          "Payload.$": "$",
          "FunctionName": "arn:aws:lambda:us-east-1:123456789012:function:check-
address:$LATEST"
        },
        "End": true
      }
    }
  }
  ...
```

따라서 다음 JSON 데이터를 check-address 함수의 입력으로 사용할 수 있게 됩니다.

```
{
    "street": "123 Main St",
    "city": "Columbus",
    "state": "OH",
    "zip": "43219"
}
```

4. 실행 시작을 선택합니다. 이제 상태 시스템 실행이 성공적으로 완료됩니다.

파라미터 필터를 사용하여 선택한 입력 조작

InputPath 필터를 사용하면 제공하는 원시 JSON 입력을 제한할 수 있지만 Parameters 필터를 사용하면 키-값 페어 컬렉션을 입력으로 전달할 수 있습니다. 이러한 키-값 페어는 상태 시스템 정의에서 정의한 정적 값이거나 InputPath를 사용하여 원시 입력에서 선택한 값일 수 있습니다.

워크플로에서 Parameters는 InputPath 다음에 적용됩니다. Parameters는 기본 작업이 입력 페이로드를 수락하는 방법을 지정하는 데 도움이 됩니다. 예를 들어 check-address Lambda 함수가 JSON 데이터 대신 문자열 파라미터를 입력으로 수락하면 Parameters 필터를 사용하여 입력을 변환할 수 있습니다.

다음 예제에서 Parameters 필터는 3단계: InputPath 필터를 사용하여 실행 입력의 특정 부분 선택의 InputPath를 사용하여 선택한 입력을 수신하고 입력 항목에 내장 함수 States.Format을 적용하여 addressString 문자열을 만듭니다. 내장 함수를 사용하면 지정된 입력에 대해 기본적인 데이터 처리 작업을 수행할 수 있습니다. 자세한 내용은 내장 함수 섹션을 참조하세요.

```
"Parameters": {
```

```
    "addressString.$": "States.Format('{}. {}, {} - {}', $.street, $.city, $.state,
 $.zip)"
  }
```

따라서 다음 문자열이 생성되어 check-address Lambda 함수에 입력으로 제공됩니다.

```
{
    "addressString": "123 Main St. Columbus, OH - 43219"
}
```

ResultSelector, ResultPath 및 OutputPath 필터를 사용하여 출력 구성

check-address Lambda 함수가 WorkflowInputOutput 상태 시스템에서 간접적으로 호출되면 함수는 주소를 확인한 후에 출력 페이로드를 반환합니다. 실행 세부 정보 페이지에서 주소 확인 단계를 선택하고 단계 세부 정보 창의 태스크 결과 내에서 출력 페이로드를 봅니다.

```
{
    "ExecutedVersion": "$LATEST",
    "Payload": {
      "statusCode": 200,
      "body": "{\"approved\":true,\"message\":\"identity validation passed\"}"
    },
    "SdkHttpMetadata": {
      "AllHttpHeaders": {
        "X-Amz-Executed-Version": [
          "$LATEST"
        ],
        ...
        ...
    "StatusCode": 200
  }
```

ResultSelector 사용

이제 ID 및 주소 확인 검사 결과를 워크플로의 다음 상태에 제공해야 하는 경우 출력 JSON에서 Payload.body 노드를 선택하고 ResultSelector 필터의 StringToJson 내장 함수를 사용하여 필요에 따라 데이터 형식을 지정할 수 있습니다.

ResultSelector는 작업 출력에서 필요한 내용을 선택합니다. 다음 예제에서 ResultSelector는 $.Payload.body의 문자열을 가져와 States.StringToJson 내장 함수를 적용하여 문자열을 JSON으로 변환하고 결과 JSON을 ID 노드에 놓습니다.

```
"ResultSelector": {
    "identity.$": "States.StringToJson($.Payload.body)"
  }
```

결과적으로 다음 JSON 데이터가 생성됩니다.

```
{
    "identity": {
      "approved": true,
      "message": "Identity validation passed"
    }
  }
```

이러한 입력 및 출력 필터를 사용할 때 잘못된 JSON 경로 표현식을 지정하여 발생하는 런타임 오류가 발생할 수도 있습니다. 자세한 내용은 단원을 참조하십시오.

ResultPath 사용

`ResultPath` 필드를 사용하여 상태의 작업 처리 결과를 저장할 초기 입력 페이로드의 위치를 지정할 수 있습니다. `ResultPath`를 지정하지 않으면 기본값으로 `$`가 설정됩니다. 이로 인해 초기 입력 페이로드가 원시 작업 결과로 바뀝니다. `ResultPath`를 `null`로 지정하면 원시 결과가 삭제되고 초기 입력 페이로드가 유효 출력이 됩니다.

`ResultSelector` 필드를 사용하여 만든 JSON 데이터에 `ResultPath` 필드를 적용하면 다음 예제와 같이 작업 결과가 입력 페이로드의 결과 노드 내에 추가됩니다.

```
{
  "data": {
    "firstname": "Jane",
    "lastname": "Doe",
    "identity": {
      "email": "jdoe@example.com",
      "ssn": "123-45-6789"
    },
    "address": {
      "street": "123 Main St",
      "city": "Columbus",
      "state": "OH",
      "zip": "43219"
    },
  "results": {
```

```
    "identity": {
      "approved": true
    }
  }
}
```

OutputPath 사용

`ResultPath`를 적용한 후에 상태 출력의 일부를 선택하여 다음 상태로 전달할 수 있습니다. 이렇게 하면 원치 않는 정보를 필터링하고 관심 있는 JSON 부분만 전달할 수 있습니다.

다음 예제에서 `OutputPath` 필드는 결과 노드 내에 상태 출력을 저장합니다(`"OutputPath": "$.results"`). 따라서 다음 상태로 전달할 수 있는 상태의 최종 출력은 다음과 같습니다.

```
{
    "addressResult": {
      "approved": true,
      "message": "address validation passed"
    },
    "identityResult": {
      "approved": true,
      "message": "identity validation passed"
    }
}
```

콘솔 기능을 사용하여 입력 및 출력 데이터 흐름 시각화

Step Functions 콘솔의 데이터 흐름 시뮬레이터 또는 실행 세부 정보 페이지의 고급 보기 옵션을 사용하여 워크플로의 상태 간의 입력 및 출력 데이터 흐름을 시각화할 수 있습니다.

자습서 8: 콘솔에서 오류 디버깅

Step Functions로 작업할 때 다음과 같은 이유로 인해 런타임 오류가 발생할 수 있습니다.

- `Choice` 상태의 변수 필드에 잘못 지정된 JSON 경로
- 상태 시스템 정의 문제(예: `Choice` 상태에 정의된 규칙과 일치하는 규칙 없음)
- 입력과 출력을 조작하기 위해 필터를 적용하는 중에 잘못된 JSON 경로 표현식
- Lambda 함수 예외로 인한 작업 실패
- IAM 권한 오류

이 자습서에서는 Step Functions 콘솔을 사용하여 이러한 몇몇 오류를 디버깅하는 방법을 알아봅니다. 자세한 내용은 Step Functions에서 오류 처리 섹션을 참조하세요.

주제

- 잘못된 경로 Choice 상태 오류 디버깅
- 입력 및 출력 필터를 적용하는 동안 JSON 경로 표현식 오류 디버깅

잘못된 경로 Choice 상태 오류 디버깅

`Choice` 상태의 변수 필드에 잘못되었거나 확인할 수 없는 JSON 경로를 지정하거나 `Choice` 상태에 일치 규칙을 정의하지 않으면 워크플로를 실행하는 동안 오류가 발생합니다.

잘못된 경로 오류를 설명하기 위해 이 자습서에서는 워크플로의 `Choice` 상태 오류를 소개합니다. CreditCardWorkflow 상태 시스템을 사용하고 오류가 발생하도록 해당 정의를 편집하겠습니다.

1. Step Functions 콘솔을 열고 CreditCardWorkflow 상태 시스템을 선택합니다.
2. 편집을 선택하여 상태 시스템 정의를 편집합니다. 다음 코드에 강조 표시된 변경 사항을 상태 시스템 정의에 적용합니다.

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Get credit limit",
  "States": {
    "Get credit limit": {
      ...
      ...
    },
    "Credit applied >= 5000?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.Payload",
          "NumericLessThan": 5000,
          "Next": "Auto-approve limit"
        },
        {
          "Variable": "$.Payload",
          "NumericGreaterThanEquals": 5000,
          "Next": "Wait for human approval"
        }
```

```
        ],
        "Default": "Wait for human approval"
    },
    ...
    ...
  }
}
```

3. 저장을 선택한 다음 일단 저장을 선택합니다.
4. 상태 시스템을 실행합니다.
5. 상태 시스템 실행의 실행 세부 정보 페이지에서 다음 중 하나를 수행합니다.

   a. 오류 메시지에서 원인을 선택하여 실행 실패 원인을 확인합니다.

   b. 오류 메시지에서 단계 세부 정보 표기를 선택하여 오류가 발생한 단계를 확인합니다.
6. 단계 세부 정보 섹션의 입력 및 출력 탭에서 고급 보기 토글 버튼을 선택하여 선택한 상태의 입력 및 출력 데이터 전송 경로를 확인합니다.
7. 그래프 보기에서 Credit applied >= 5000?이 선택되어 있는지 확인하고 다음을 수행합니다.

   a. 입력 상자에서 상태의 입력 값을 봅니다.

   b. 정의 탭을 선택하고 변수 필드에 지정된 JSON 경로를 확인합니다.

   Credit applied >= 5000? 상태의 입력값은 숫자 값이지만 입력 값의 JSON 경로를 `$.Payload`로 지정했습니다. 상태 시스템 실행 중에는 이 JSON 경로가 존재하지 않으므로 `Choice` 상태에서 이 경로를 확인할 수 없습니다.
8. 상태 시스템을 편집하여 변수 필드 값을 **$**로 지정합니다.

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Get credit limit",
  "States": {
    "Get credit limit": {
      ...
      ...
    },
    "Credit applied >= 5000?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$",
          "NumericLessThan": 5000,
```

```
          "Next": "Auto-approve limit"
        },
        {
          "Variable": "$",
          "NumericGreaterThanEquals": 5000,
          "Next": "Wait for human approval"
        }
        ],
        "Default": "Wait for human approval"
    },
    ...
    ...
  }
}
```

입력 및 출력 필터를 적용하는 동안 JSON 경로 표현식 오류 디버깅

입력 및 출력 필터를 사용할 때 잘못된 JSON 경로 식 지정으로 인해 런타임 오류가 발생할 수 있습니다.

다음 예제에서는 자습서 5에서 만든 WorkflowInputOutput 상태 시스템을 사용하고 `ResultSelector` 필터를 사용하여 작업 출력의 일부를 선택하는 시나리오를 보여줍니다.

1. `ResultSelector` 필터를 적용하여 ID 확인 단계를 위한 작업 출력 일부를 선택합니다. 이렇게 하려면 상태 시스템 정의를 다음과 같이 편집합니다.

```
{
  "StartAt": "Verify identity",
  "States": {
    "Verify identity": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:check-
identity",
        "Payload": {
          "email": "jdoe@example.com",
          "ssn": "123-45-6789"
        }
      },
      ...
      ...
```

```
      "ResultSelector": {
        "identity.$": "$.Payload.body.message"
 }",
      "End": true
    }
  }
}
```

2. 상태 시스템을 실행합니다.
3. 상태 시스템 실행의 실행 세부 정보 페이지에서 다음을 수행합니다.

   a. 오류 메시지에서 원인을 선택하여 실행 실패 원인을 확인합니다.

   b. 오류 메시지에서 단계 세부 정보 표기를 선택하여 오류가 발생한 단계를 확인합니다.
4. 오류 메시지에서는 $.Payload.body 노드의 콘텐츠가 이스케이프된 JSON 문자열입니다. JSON 경로 표기법을 사용하여 문자열을 참조할 수 없으므로 오류가 발생했습니다.
5. $.Payload.body.message 노드를 참조하려면 다음을 수행합니다.

   a. 먼저 States.StringToJSON 내장 함수를 사용하여 문자열을 JSON 형식으로 변환합니다.

   b. 내장 함수 내에 있는 $.Payload.body.message 노드의 JSON 경로를 지정합니다.

   ```
   "ResultSelector": {
    "identity.$":"States.StringToJson($.Payload.body.message)"
   }
   ```
6. 상태 시스템을 다시 실행합니다.

사용 사례

AWS Step Functions를 사용하면 비즈니스 요구 사항을 애플리케이션으로 빠르게 변환하는 데 도움이 되는 시각적 워크플로를 빌드할 수 있습니다. Step Functions는 자동으로 상태, 체크포인트 및 재시작을 관리하고 오류와 예외를 자동으로 처리하는 기능을 기본 제공합니다. Step Functions에서 자동으로 제공하는 기능을 더 잘 이해하려면 다음 사용 사례를 읽어보세요.

주제

- 데이터 처리
- 기계 학습
- 마이크로서비스 오케스트레이션
- IT 및 보안 자동화

데이터 처리

점점 다양해진 소스로 인해 데이터 양이 증가함에 따라 조직에서는 정보에 입각한 비즈니스 결정을 더욱 빠르게 내릴 수 있도록 이 데이터를 신속하게 처리해야 합니다. 대규모 데이터를 처리하려면 조직은 모바일 장치, 애플리케이션, 위성, 마케팅 및 영업, 운영 데이터 저장소, 인프라 등에서 수신된 정보를 관리하기 위해 리소스를 탄력적으로 프로비저닝해야 합니다.

Step Functions는 데이터 처리 워크플로를 성공적으로 관리하는 데 필요한 확장성, 안정성 및 가용성을 제공합니다. 수평적으로 규모를 조정하고 내결함성 워크플로를 제공하는 Step Functions를 사용하여 수백만 개의 동시 실행을 관리할 수 있습니다. Step Functions의 `Parallel` 상태 유형 또는 # 상태 유형을 사용한 동적 병렬 처리와 같은 병렬 실행을 사용하여 데이터를 더 빠르게 처리합니다. 워크플로의 일부로 # 상태를 사용하여 Amazon S3 버킷과 같은 정적 데이터 스토어의 객체를 반복할 수 있습니다. 또한 Step Functions를 사용하면 복잡한 프로세스를 관리하지 않고도 실패한 실행을 쉽게 재시도하거나 특정 오류 처리 방법을 선택할 수 있습니다.

데이터 처리 요구 사항에 따라 Step Functions는 일괄 처리용 AWS Batch, 빅 데이터 처리용 Amazon EMR, 데이터 준비용 AWS Glue, 데이터 분석용 Athena, 컴퓨팅용 AWS Lambda와 같은 AWS에서 제공하는 다른 데이터 처리 서비스와 직접 통합됩니다.

고객이 Step Functions를 사용하여 수행하는 데이터 처리 워크플로 유형의 예제는 다음과 같습니다.

파일, 비디오 및 이미지 처리

- 비디오 파일 컬렉션을 가져와 휴대폰, 노트북 또는 TV와 같이 표시할 장치에 적합한 다른 크기나 해상도로 변환합니다.
- 사용자가 업로드한 대규모 사진 컬렉션을 가져와 썸네일이나 다양한 해상도의 이미지로 변환한 후 사용자의 웹사이트에 표시할 수 있습니다.
- CSV 파일과 같은 반정형 데이터를 가져와 인보이스와 같은 비정형 데이터와 결합하여 매월 비즈니스 이해 관계자에게 전송되는 비즈니스 보고서를 생성합니다.
- 위성에서 수집한 지구 관측 데이터를 가져와 서로 정렬되는 형식으로 변환한 다음 유용한 정보를 추가적으로 얻을 수 있도록 지구에서 수집한 다른 데이터 소스를 추가합니다.
- 다양한 운송 수단의 제품 운송 로그를 가져와 몬테카를로 시뮬레이션을 사용하여 최적화를 찾은 다음 제품을 배송하는 데 귀사를 사용하는 조직과 담당자에게 다시 보고서를 보냅니다.

추출, 전환, 적재(ETL) 작업 조정:

- AWS Glue를 사용하여 일련의 데이터 준비 단계를 통해 영업 기회 기록을 마케팅 지표 데이터 세트와 결합하고 조직 전체에서 사용할 수 있는 비즈니스 인텔리전스 보고서를 생성합니다.
- 빅 데이터 처리용 Amazon EMR 클러스터를 만들고 시작 및 종료합니다.

배치 프로세싱 및 고성능 컴퓨팅(HPC) 워크로드:

- 원시 전장 유전체를 변이 검출로 처리하는 유전체학 2차 분석 파이프라인을 빌드합니다. 원시 파일을 참조 서열에 정렬하고 동적 병렬화를 사용하여 지정된 염색체 목록에서 변이를 검출합니다.
- 다양한 전기 및 화학 화합물을 사용하여 다양한 레이아웃을 시뮬레이션해 차세대 모바일 장치나 기타 전자 제품의 생산 효율성을 찾습니다. 다양한 시뮬레이션을 통해 워크로드를 대규모로 일괄 처리하여 최적의 설계를 얻습니다.

기계 학습

기계 학습을 사용하면 조직은 수집된 데이터를 빠르게 분석하여 패턴을 식별한 다음 사람의 개입을 최소화하면서 의사 결정을 내릴 수 있습니다. 기계 학습은 학습 데이터라고 하는 초기 데이터 세트에서 시작됩니다. 이 학습 데이터는 기계 학습 모델의 예측 정확도를 높이는 데 도움이 되고 이 모델의 학습 토대 역할을 합니다. 모델이 비즈니스 요구 사항을 충족할 만큼 정확하다고 간주되면 프로덕션에 배포됩니다. AWS Step Functions 데이터 과학 소프트웨어 개발 키트(SDK)는 Amazon SageMaker 및 Step Functions를 사용하여 데이터를 전처리하고 학습한 다음 모델을 게시하는 워크플로를 쉽게 만들 수 있는 오픈 소스 라이브러리입니다.

기존 데이터 세트 전처리는 조직에서 주로 사용하는 교육 데이터를 만드는 방법입니다. 이 방법은 이미지 내 객체에 레이블 지정, 텍스트에 주석 달기 또는 오디오 처리와 같은 방법으로 정보를 추가합니다. 데이터를 전처리하려면 AWS Glue를 사용하거나 Jupyter Notebook 앱을 실행하는 SageMaker 노트북 인스턴스를 만들면 됩니다. 데이터가 준비되면 Amazon S3에 업로드하여 쉽게 액세스할 수 있습니다. 기계 학습 모델이 학습되면 배포 준비가 될 때까지 각 모델의 파라미터를 조정하여 정확도를 향상시킬 수 있습니다.

Step Functions를 사용하면 SageMaker에서 엔드 투 엔드 기계 학습 워크플로를 오케스트레이션할 수 있습니다. 이러한 워크플로에는 데이터 전처리, 후처리, 특성 추출, 데이터 검증 및 모델 평가가 포함될 수 있습니다. 모델을 프로덕션에 배포한 후에는 새로운 방식을 개선 및 테스트하여 비즈니스 성과를 지속적으로 개선할 수 있습니다. Python에서 직접 프로덕션 준비 워크플로를 만들거나 Step Functions Data Science SDK를 사용하여 해당 워크플로를 복사하고 새로운 옵션을 검사하며 개선된 워크플로를 프로덕션에서 사용할 수 있습니다.

고객이 Step Functions를 사용하는 일부 유형의 기계 학습 워크플로는 다음과 같습니다.

부정 탐지

- 신용 사기와 같은 사기 거래를 식별하여 발생을 방지합니다.
- 학습된 기계 학습 모델을 사용하여 계정 탈취를 감지하고 방지합니다.
- 가짜 계정 생성을 비롯한 홍보성 악용사례를 식별하여 신속하게 조치를 취할 수 있습니다.

개인화 및 권장 사항

- 고객의 관심을 끌 것으로 예상되는 제품을 기반으로 대상 고객에게 제품을 추천합니다.
- 고객이 자신의 계정을 무료 등급에서 유료 구독으로 업그레이드할지 여부를 예측합니다.

데이터 강화

- 전처리의 일환으로 데이터 강화를 사용하여 더욱 정확한 기계 학습 모델에 더 나은 학습 데이터를 제공합니다.
- 텍스트 및 오디오 발췌문에 주석을 달아 풍자 및 속어와 같은 구문 정보를 추가합니다.
- 이미지에서 추가 객체에 레이블을 지정하여 객체가 사과, 농구, 바위 또는 동물인지 여부와 같이 모델에서 학습할 수 있는 중요한 정보를 제공합니다.

마이크로서비스 오케스트레이션

마이크로서비스 아키텍처는 애플리케이션을 느슨하게 결합된 서비스로 나눕니다. 이점에는 확장성 향상, 탄력성 향상, 시장 출시 시간 단축 등이 있습니다. 각 마이크로서비스는 독립적이므로 전체 애플리케이션 규모를 조정할 필요 없이 단일 서비스나 기능을 쉽게 스케일 업할 수 있습니다. 개별 서비스가 느슨하게 결합되어 있어 개별 팀이 전체 애플리케이션을 이해할 필요 없이 단일 비즈니스 프로세스에 집중할 수 있습니다. 또한 마이크로서비스를 사용하면 비즈니스 요구 사항에 맞는 개별 구성 요소를 선택할 수 있으므로 전체 워크플로를 다시 작성하지 않고도 선택 항목을 유연하게 변경할 수 있습니다. 다양한 팀에서 자신이 원하는 프로그래밍 언어와 프레임워크로 마이크로서비스를 사용할 수 있으며 이 마이크로서비스는 애플리케이션 프로그래밍 인터페이스(API)를 통해 애플리케이션 내 다른 마이크로서비스와 계속 통신할 수 있습니다.

Step Functions는 마이크로서비스 워크플로를 관리하는 여러 가지 방법을 제공합니다. 장기 실행 워크플로의 경우 AWS Fargate 통합과 함께 표준 워크플로를 사용하여 컨테이너에서 실행되는 애플리케이션을 오케스트레이션할 수 있습니다. 즉각적인 대응이 필요한 단기간 대용량 워크플로의 경우에는 동기 Express 워크플로가 이상적입니다. 이러한 워크플로는 워크플로 지속 시간이 짧고 응답을 반환하기 전에 일련의 단계를 완료해야 하는 웹 기반 또는 모바일 애플리케이션에 사용될 수 있습니다. Amazon API Gateway에서 동기 Express 워크플로를 직접 트리거할 수 있으며 워크플로가 완료되거나 시간 초과될 때까지 연결이 열린 상태로 유지됩니다. 즉각적인 응답이 필요하지 않은 단기간의 워크플로를 위해 Step Functions는 비동기 Express 워크플로를 제공합니다.

다음은 Step Functions를 사용하는 일부 API 오케스트레이션의 예제입니다.

동기 또는 실시간 워크플로

- 직원의 성 업데이트와 같이 레코드의 값을 변경하면 변경 내용이 화면에 즉시 표시되도록 할 수 있습니다.
- 결제 중에 품목 추가, 제거 또는 수량 변경 등 주문을 업데이트한 다음 업데이트 내용을 다시 고객에게 즉시 반영합니다.
- 빠른 처리 작업을 실행하고 결과를 다시 요청자에게 즉시 반환합니다.

컨테이너 오케스트레이션

- Amazon Elastic Kubernetes Service를 사용하여 Kubernetes에서 또는 Fargate를 사용하여 Amazon Elastic Container Service(ECS)에서 작업을 실행하고 동일한 워크플로의 일부로 Amazon SNS로 알림 전송과 같은 AWS 다른 서비스와 통합할 수 있습니다.

IT 및 보안 자동화

IT 자동화는 소프트웨어 업그레이드 및 패치, 취약성을 해결하기 위한 보안 업데이트 배포, 인프라 선택, 데이터 동기화, 지원 티켓 라우팅 등과 같이 점점 더 복잡하고 시간을 많이 소모하는 작업을 관리하는 데 도움이 될 수 있습니다. 반복적이고 시간을 소모하는 작업을 자동화하면 조직에서 일상적인 작업을 대규모로 신속하고 일관되게 완료할 수 있습니다. 이를 통해 특성 개발, 복잡한 지원 요청 및 혁신과 같은 전략적 작업에 집중하는 동시에 증가하는 수요를 충족할 수 있습니다.

Step Functions를 사용하면 수동 개입 없이 비즈니스 요구 사항이 충족되도록 자동으로 규모를 조정하는 워크플로를 만들 수 있습니다. 워크플로에서 오류가 발생하는 경우 수동 개입이 필요하지 않은 경우가 많습니다. Step Functions를 사용하면 자동으로 실패한 작업을 재시도하고 워크플로의 오류를 관리할 수 있는 지수 백오프를 수행할 수 있습니다.

워크플로를 진행하려면 먼저 사람이 개입해야 하는 상황이 발생할 수 있습니다. 예를 들어 대폭적인 크레딧 증가를 승인하려면 사람의 승인이 필요할 수 있습니다. 이를 관리하기 위해 Step Functions에서 분기 로직을 정의하여 정의된 양을 초과하는 요청만 사람이 승인해야 하고 다른 모든 요청은 자동으로 완료되도록 할 수 있습니다. 사람이 승인해야 하는 경우 Step Functions를 사용하면 특정 단계에서 워크플로를 일시 중지하고 응답을 기다린 다음 응답을 수신하면 워크플로를 계속할 수 있습니다.

다음은 고객이 Step Functions를 사용하는 자동화 워크플로 유형의 몇 가지 예제입니다.

IT 자동화

- SSH 포트 열기, 디스크 공간 부족 또는 Amazon S3 버킷에 대한 공개 액세스 권한 부여와 같은 사고를 자동으로 해결합니다.
- AWS CloudFormation StackSets 배포 자동화

보안 자동화

- 사용자와 사용자 액세스 키가 노출된 시나리오에 대한 대응을 자동화합니다.
- 작업을 특정 ARN으로 제한 또는 다른 조치 적용과 같은 정의된 정책 조치에 따라 보안 사고 대응을 자동으로 해결합니다.
- 수신 후 몇 초 내에 직원에게 피싱 이메일을 경고합니다.

사람 승인

- 기계 학습 모델 교육을 자동화하고 데이터 과학자가 모델을 수동으로 승인하도록 한 다음 수신된 응답에 따라 모델을 자동으로 배포하거나 거부합니다.

- 수동 검토를 위해 부정적인 감정을 가진 고객들이 즉시 에스컬레이션되도록 감성 분석을 기반으로 접수된 고객 피드백의 라우팅을 자동화합니다.

Step Functions 작동 방식

이 단원에서는 AWS Step Functions를 익히고 작동 방식을 이해하도록 도와주는 중요한 개념을 설명합니다.

주제

표준 워크플로와 Express 워크플로 비교

상태 시스템을 만들 때 유형을 표준 또는 Express 중 하나로 선택합니다. 상태 시스템의 기본 유형은 표준입니다. 유형이 표준인 상태 시스템을 표준 워크플로라고 하고 유형이 Express인 상태 시스템을 Express 워크플로라고 합니다.

표준 및 Express 워크플로 모두에서 Amazon States Language를 사용하여 상태 시스템을 정의합니다. 상태 시스템 실행은 선택한 유형에 따라 다르게 작동합니다.

> **Important**
>
> 상태 시스템을 만든 후에는 선택한 유형을 변경할 수 없습니다.

> **Note**
>
> 선택한 편집기와 같이 Step Functions 콘솔 외부에서 상태 시스템을 정의하는 경우 상태 시스템 정의를 .asl.json 확장명으로 저장해야 합니다.

표준 워크플로는 장기 실행되고(최대 1년) 내구성이 뛰어나며 감사 가능한 워크플로에 적합합니다. 실행 완료 후 최대 90일까지 Step Functions API를 사용하여 전체 실행 내역을 검색할 수 있습니다. 표준 워크플로는 ASL에 `Retry` 동작이 지정되지 않는 한 작업과 상태가 절대 두 번 이상 실행되지 않는 정확하게 1회 실행 모델을 따릅니다. 따라서 표준 워크플로는 Amazon EMR 클러스터 시작 또는 결제 처리와 같이 멱등성이 없는 작업을 오케스트레이션하는 데 적합합니다. 표준 워크플로 실행 요금은 처리된 상태 전환 횟수에 따라 청구됩니다.

Express 워크플로는 IoT 데이터 수집, 스트리밍 데이터 처리 및 변환, 모바일 애플리케이션 백엔드 등의 대용량 이벤트 처리 워크로드에 적합합니다. 이러한 워크플로는 최대 5분 동안 실행할 수 있습니다. Express 워크플로는 실행이 두 번 이상 실행될 수 있는 최소 1회 이상 실행되는 모델을 사용합니다. 따라서 Express 워크플로는 입력 데이터 변환 및 Amazon DynamoDB의 PUT 작업을 통한 저장과 같이 멱등성이 있는 작업을 오케스트레이션하는 데 적합합니다. Express 워크플로 실행 요금은 실행 횟수, 실행 기간 및 실행이 실행되는 동안에 사용된 메모리에 따라 청구됩니다.

표준 및 Express 워크플로는 Amazon API Gateway(규모에 따라 조정되는 완전 관리형 API)을 통한 HTTP 요청, IoT 규칙, 140개가 넘는 Amazon EventBridge 이벤트 소스와 같은 이벤트에 대한 응답으로 자동 시작됩니다.

> **Tip**
>
> Express 워크플로 예제를 AWS 계정에 배포하려면 AWS Step Functions 워크숍의 모듈 7 - API Gateway, Parallel 상태, Express 워크플로를 참조하세요.

표준 및 Express 워크플로 실행을 위한 콘솔 환경은 콘솔에서의 표준 및 Express 워크플로 실행을 참조하세요.

표준 워크플로와 Express 워크플로 비교

	표준 워크플로	Express 워크플로: 동기 및 비동기
최대 지속 시간	1년	5분

	표준 워크플로	Express 워크플로: 동기 및 비동기
지원되는 실행 시작 비율	지원되는 실행 시작률과 관련된 할당량은 API 작업 제한과 관련된 할당량을 참조하세요.	지원되는 실행 시작률과 관련된 할당량은 API 작업 제한과 관련된 할당량을 참조하세요.
지원되는 상태 전환 비율	지원되는 상태 전환율과 관련된 할당량은 상태 제한과 관련된 할당량을 참조하세요.	제한 없음
요금	상태 전환 횟수를 기준으로 가격이 책정됩니다. 상태 전환은 실행이 완료되는 단계마다 계산됩니다.	실행 횟수, 실행 기간, 메모리 사용량에 따라 가격이 책정됩니다.
실행 내역	Step Functions API를 사용하여 실행을 나열하고 설명할 수 있습니다. 콘솔을 통해 실행을 시각적으로 디버깅할 수 있습니다. 상태 시스템에서 로깅을 활성화하여 CloudWatch Logs에서 실행을 검사할 수도 있습니다. 콘솔에서의 표준 워크플로 실행을 디버깅하는 방법에 대한 자세한 내용은 콘솔에서의 표준 및 Express 워크플로 실행 및 실행 보기 및 디버깅을 참조하세요.	무제한 실행 내역, 즉 5분 동안 생성할 수 있는 만큼 실행 내역 항목이 유지됩니다. 상태 시스템에서 로깅을 활성화하여 CloudWatch Logs 또는 Step Functions 콘솔에서 실행을 검사할 수 있습니다. 콘솔에서의 Express 워크플로 실행을 디버깅하는 방법에 대한 자세한 내용은 콘솔에서의 표준 및 Express 워크플로 실행 및 실행 보기 및 디버깅을 참조하세요.
실행 시맨틱	워크플로를 정확하게 1회 실행합니다.	비동기 Express 워크플로: 최소 1회 이상 실행되는 워크플로. 동기 Express 워크플로: 최대 1회 실행되는 워크플로.

	표준 워크플로	Express 워크플로: 동기 및 비동기
서비스 통합	모든 서비스 통합 및 패턴을 지원합니다.	모든 서비스 통합을 지원합니다. **Note** Express 워크플로는 작업 실행(`.sync`) 또는 콜백(`.waitForTaskToken`) 서비스 통합 패턴을 지원하지 않습니다.
Step Functions 활동	Step Functions 활동을 지원합니다.	Step Functions 활동을 지원하지 않습니다.

동기 및 비동기 Express 워크플로

선택할 수 있는 Express 워크플로 유형에는 비동기 Express 워크플로 및 동기 Express 워크플로 등 두 가지가 있습니다.

- 비동기 Express 워크플로는 워크플로가 시작되었다는 확인을 반환되지만 워크플로가 완료될 때까지 기다리지 않습니다. 결과를 얻으려면 서비스의 CloudWatch Logs를 폴링해야 합니다. 메시징 서비스 또는 다른 서비스에서 사용하지 않는 데이터 처리와 같은 즉각적인 응답 출력이 필요하지 않은 경우에 비동기 Express 워크플로를 사용할 수 있습니다. Step Functions의 중첩된 워크플로에서 또는 `StartExecution` API 직접 호출을 사용하여 이벤트에 대한 응답으로 비동기 Express 워크플로를 시작할 수 있습니다.
- 동기 Express 워크플로는 워크플로를 시작하고 완료될 때까지 기다린 다음 결과를 반환합니다. 동기 Express 워크플로를 사용하여 마이크로서비스를 오케스트레이션할 수 있습니다. 동기 Express 워크플로를 사용하면 오류 처리, 재시도 또는 병렬 작업 실행을 위한 추가 코드를 개발하지 않고도 애플리케이션을 개발할 수 있습니다. Amazon API Gateway, AWS Lambda에서 간접적으로 호출하거나 `StartSyncExecution` API 직접 호출을 사용하여 동기 Express 워크플로를 실행할 수 있습니다.

> **Note**
>
> 콘솔에서 Step Functions Express 워크플로를 동기적으로 실행하면 60초 후에 `StartSyncExecution` 요청이 만료됩니다. Express 워크플로를 최대 5분 동안 동기적으로 실행하려면 Step Functions 콘솔 대신 AWS SDK 또는 AWS Command Line Interface(AWS CLI)를 사용하여 `StartSyncExecution` 요청을 수행하세요.

동기 Express 실행 API 직접 호출은 기존 계정 용량 한도에 영향을 주지 않습니다. Step Functions는 온디맨드 용량을 제공하고 지속적인 워크로드에 따라 자동으로 규모를 조정합니다. 용량이 확보될 때까지 워크로드 급증이 제한될 수 있습니다.

실행 보장

표준 워크플로	비동기 Express 워크플로	동기 Express 워크플로	
정확하게 1회 실행되는 워크플로 실행	최소 1회 이상 실행되는 워크플로 실행	최대 1회 실행되는 워크플로 실행	
실행 상태는 상태 전환 간에 내부적으로 유지됩니다.	실행 상태는 상태 전환 간에 유지되지 않습니다.	실행 상태는 상태 전환 간에 유지되지 않습니다.	
현재 실행 중인 워크플로와 동일한 이름으로 실행을 시작하면 자동으로 멱등성 응답을 반환합니다. 새 워크플로는 시작되지 않으며 현재 실행 중인 워크플로가 완료되면 예외가 발생합니다.	멱등성이 자동으로 관리되지 않습니다. 같은 이름의 워크플로를 여러 개 시작하면 동시에 실행됩니다. 상태 시스템 로직이 멱등하지 않으면 내부 워크플로 상태가 손실될 수 있습니다.	멱등성이 자동으로 관리되지 않습니다. Step Functions는 실행이 시작되면 대기하고 완료되면 상태 시스템 결과를 반환합니다. 예외가 발생하면 워크플로는 다시 시작되지 않습니다.	
실행 내역 데이터는 90일 후에 제거됩니다.	실행 내역은 Step Functions에서 캡처되	실행 내역은 Step Functions에서 캡처되	

표준 워크플로	비동기 Express 워크플로	동기 Express 워크플로	
오래된 실행 데이터를 제거한 후에 워크플로 이름을 다시 사용할 수 있습니다. 규정 준수, 조직 또는 규제 기관 요구 사항을 충족하려면 할당량 요청을 보내 실행 내역 보존 기간을 30일로 줄이면 됩니다. 이렇게 하려면 AWS Support Center Console를 사용하고 새 사례를 만듭니다.	지 않습니다. Amazon CloudWatch Logs를 통해 로깅을 활성화해야 합니다.	지 않습니다. Amazon CloudWatch Logs를 통해 로깅을 활성화해야 합니다.	

Express 워크플로를 사용하여 비용 최적화

Step Functions는 상태 시스템을 빌드하는 데 사용하는 워크플로 유형에 따라 표준 및 Express 워크플로의 요금을 결정합니다. 서버리스 워크플로 비용을 최적화하려면 다음 권장 사항 중 하나 또는 둘 다 따르면 됩니다.

주제

- 팁 #1: 표준 워크플로 내에서 Express 워크플로 중첩
- 팁 #2: 표준 워크플로를 Express 워크플로로 전환

표준 또는 Express 워크플로 유형 선택이 결제에 미치는 영향은 AWS Step Functions 요금을 참조하세요.

팁 #1: 표준 워크플로 내에서 Express 워크플로 중첩

Step Functions는 기간과 단계 수가 한정적인 워크플로를 실행합니다. 일부 워크플로는 짧은 시간 내에 실행을 완료할 수 있습니다. 다른 워크플로에서는 장기간 실행되는 워크플로와 이벤트 속도가 빠른 워

크플로를 함께 사용해야 할 수도 있습니다. Step Functions를 사용하면 여러 단순한 소규모 워크플로에서 복잡한 대규모 워크플로를 빌드할 수 있습니다.

예를 들어 주문 처리 워크플로를 빌드하려면 멱등성이 없는 모든 작업을 표준 워크플로에 포함하면 됩니다. 여기에는 인적 상호 작용을 통한 주문 승인 및 결제 처리와 같은 작업이 포함될 수 있습니다. 그런 다음 결제 알림 전송 및 제품 인벤토리 업데이트와 같은 일련의 멱등성이 있는 작업을 Express 워크플로에 결합할 수 있습니다. 표준 워크플로 내에서 이 Express 워크플로를 중첩할 수 있습니다. 이 예제에서는 표준 워크플로를 상위 상태 시스템이라고 합니다. 중첩된 Express 워크플로를 하위 상태 시스템이라고 합니다.

팁 #2: 표준 워크플로를 Express 워크플로로 전환

다음 요구 사항을 충족하는 경우 기존 표준 워크플로를 Express 워크플로로 변환할 수 있습니다.

- 워크플로는 실행을 5분 이내에 완료해야 합니다.
- 워크플로는 최소 1회 이상 실행 모델을 준수합니다. 즉, 워크플로의 각 단계가 정확히 2회 이상 실행될 수 있습니다.
- 워크플로는 `.waitForTaskToken` 또는 `.sync` 서비스 통합 패턴을 사용하지 않습니다.

> **Important**
>
> Express 워크플로는 Amazon CloudWatch Logs를 사용하여 실행 내역을 기록합니다. CloudWatch Logs를 사용하면 추가 비용이 발생합니다.

콘솔을 사용하여 표준 워크플로를 Express 워크플로로 전환하기

1. Step Functions 콘솔을 엽니다.
2. 상태 시스템 페이지에서 표준 유형 상태 시스템을 선택하여 엽니다.

   > **Tip**
   >
   > 상태 시스템 목록을 필터링하고 표준 워크플로만 보려면 모든 유형 드롭다운 목록에서 표준을 선택하세요.

3. 신규로 복사를 선택합니다.

   Workflow Studio가 디자인 모드에서 열리고 선택한 상태 시스템의 워크플로가 표시됩니다.

4. (선택 사항) 워크플로 설계를 업데이트합니다.
5. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 시스템 이름인 MyStateMachine 옆에 있는 편집 아이콘을 선택합니다. 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.
6. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   유형에 Express를 선택했는지 확인합니다. 상태 머신 설정의 다른 모든 기본 선택을 그대로 둡니다.

   > **Note**
   >
   > 이전에 AWS CDK 또는 AWS SAM에서 정의한 표준 워크플로를 변환하는 경우 `Type` 및 `Resource` 이름 값을 변경해야 합니다.

7. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

   > **Note**
   >
   > Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

워크플로 비용 최적화를 관리할 때의 모범 사례와 지침에 대한 자세한 내용은 비용 효과적인 AWS Step Functions 워크플로 빌드를 참조하세요.

상태

개별 상태는 입력을 기반으로 결정하고 입력에서 작업을 수행하며 출력을 다른 상태로 전달할 수 있습니다. AWS Step Functions에서는 Amazon States Language(ASL)로 워크플로를 정의합니다. Step Functions 콘솔은 애플리케이션 로직을 시각화하는 데 도움이 되도록 상태 시스템의 그래픽 표현을 제공합니다.

> **Note**
>
> 선택한 편집기와 같이 Step Functions 콘솔 외부에서 상태 시스템을 정의하는 경우 상태 시스템 정의를 .asl.json 확장명으로 저장해야 합니다.

상태는 상태 시스템의 요소입니다. 상태는 이름으로 참조되는데, 이름은 문자열일 수 있으며 전체 상태 머신 범위 내에서 고유해야 합니다.

상태는 상태 머신에서 다음과 같은 다양한 기능을 수행할 수 있습니다.

- 상태 머신에서 몇 가지 작업 수행(작업 상태)
- 실행 브랜치 간 선택(Choice 상태)
- 오류로 실행 중지 또는 성공(Fail 또는 Succeed 상태)
- 입력을 출력으로 전달 또는 일부 수정된 데이터를 워크플로에 전달(Pass 상태)
- 특정 시간 동안 또는 지정된 날짜 및 시간까지 지연 제공(Wait 상태)
- 브랜치 병렬 실행 시작(Parallel 상태)
- 동적으로 단계 반복(Map 상태)

다음 예는 AWS Lambda 함수를 수행하는 `HelloWorld`라는 상태입니다.

```
"HelloWorld": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloFunction",
  "Next": "AfterHelloWorldState",
  "Comment": "Run the HelloWorld Lambda function"
}
```

상태는 다음과 같은 많은 일반 기능을 공유합니다.

- 상태 유형을 나타내는 `Type` 필드
- 상태에 대한 사람이 읽을 수 있는 메모나 설명이 있는 선택적 `Comment` 필드
- 각 상태(`Succeed` 또는 `Fail` 상태 제외)에는 `Next` 필드가 있어야 하며, `End` 필드를 지정하여 터미널 상태로 전환할 수도 있습니다.

> **Note**
>
> `Choice` 상태에는 둘 이상의 `Next` 상태가 있을 수 있지만, 각 선택 규칙 내에는 하나만 있을 수 있습니다. `Choice` 상태는 `End`를 사용할 수 없습니다.

특정 상태 유형에는 추가 필드가 필요하거나 일반 필드 사용법을 재정의할 수 있습니다.

표준 워크플로를 만들고 실행한 후에는 Step Functions 콘솔의 실행 세부 정보 페이지를 확인하여 각 상태, 입력 및 출력, 활성 시기 및 기간에 대한 정보에 액세스할 수 있습니다. 자세한 내용은 Step Functions 콘솔에서 실행 보기 및 디버깅 섹션을 참조하세요.

Express 워크플로 실행을 만들고 실행한 후에 Express 워크플로에 로깅을 활성화하면 Step Functions 콘솔 또는 Amazon CloudWatch Logs에서 실행에 대한 정보에 액세스할 수 있습니다. 자세한 내용은 Step Functions 콘솔에서 실행 보기 및 디버깅 섹션을 참조하세요.

주제

Amazon States Language

Amazon States Language는 작업을 수행할 수 있는 상태 시스템(상태 모음)을 정의하거나(`Task` 상태) 다음으로 전환할 상태를 결정하거나(`Choice` 상태) 오류를 표시하면서 실행을 중지하는(`Fail` 상태) 데 사용되는 JSON 기반의 구조화된 언어입니다.

자세한 내용은 Amazon States Language Specification 및 Statelint(Amazon States Language 코드를 검증하는 도구)를 참조하십시오.

Amazon States Language를 사용하여 Step Functions 콘솔에서 상태 시스템을 만들려면 시작하기를 참조하세요.

> **Note**
>
> 선택한 편집기와 같이 Step Functions 콘솔 외부에서 상태 시스템을 정의하는 경우 상태 시스템 정의를 .asl.json 확장명으로 저장해야 합니다.

Amazon States Language 사양 예제

```
{
  "Comment": "An example of the Amazon States Language using a choice state.",
  "StartAt": "FirstState",
  "States": {
    "FirstState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FUNCTION_NAME",
      "Next": "ChoiceState"
    },
    "ChoiceState": {
      "Type" : "Choice",
      "Choices": [
        {
          "Variable": "$.foo",
          "NumericEquals": 1,
          "Next": "FirstMatchState"
        },
        {
          "Variable": "$.foo",
          "NumericEquals": 2,
          "Next": "SecondMatchState"
        }
      ],
      "Default": "DefaultState"
    },

    "FirstMatchState": {
      "Type" : "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:OnFirstMatch",
      "Next": "NextState"
    },
```

```
    "SecondMatchState": {
      "Type" : "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:OnSecondMatch",
      "Next": "NextState"
    },

    "DefaultState": {
      "Type": "Fail",
      "Error": "DefaultStateError",
      "Cause": "No Matches!"
    },

    "NextState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FUNCTION_NAME",
      "End": true
    }
  }
}
```

주제

- 상태 시스템 구조
- 내장 함수
- 일반 상태 필드

상태 시스템 구조

다음 필드가 들어 있는 구조를 나타내는 JSON 텍스트를 사용하여 정의된 상태 시스템입니다.

Comment(선택 사항)

육안으로 읽을 수 있는 머신 상태 설명입니다.

StartAt(필수)

상태 객체 중 하나의 이름과 대/소문자를 포함하여 정확하게 일치해야 하는 문자열입니다.

TimeoutSeconds(선택 사항)

상태 시스템이 시작할 수 있는 실행의 최대 시간(초)입니다. 지정된 시간보다 오래 실행될 경우 실행이 실패하고 `States.Timeout` 오류 이름이 표시됩니다.

`Version`(선택 사항)

상태 시스템에 사용되는 Amazon States Language의 버전입니다(기본값: "1.0").

`States`(필수)

쉼표로 구분된 상태 집합을 포함하는 객체입니다.

`States` 필드에는 다음과 같은 상태가 포함됩니다.

```
{
    "State1" : {
    },

    "State2" : {
    },
    ...
}
```

상태 시스템은 상태 시스템에 들어 있는 상태 및 상태 간 관계에 의해 정의됩니다.

다음은 예입니다.

```
{
  "Comment": "A Hello World example of the Amazon States Language using a Pass state",
  "StartAt": "HelloWorld",
  "States": {
    "HelloWorld": {
      "Type": "Pass",
      "Result": "Hello World!",
      "End": true
    }
  }
}
```

상태 시스템 실행이 시작되면 `StartAt` 필드(`"HelloWorld"`)에서 참조하는 상태로 시스템이 시작됩니다. 이 상태에 `"End": true` 필드가 있으면 실행이 중지되고 결과가 반환됩니다. 그렇지 않으면 시스템에서 `"Next":` 필드를 찾고 해당 상태로 다음 작업을 계속합니다. 터미널 상태(`"Type": "Succeed"`, `"Type": "Fail"` 또는 `"End": true`인 상태)에 도달하거나 런타임 오류가 발생할 때까지 이 프로세스가 반복됩니다.

상태 시스템 내 상태에는 다음 규칙이 적용됩니다.

- 상태는 닫힌 블록 내에서 어느 순서로든 발생할 수 있지만 상태가 나열된 순서가 실행 순서에 영향을 미치지 않습니다. 상태의 내용에 따라 이 순서가 결정됩니다.
- 하나의 상태 시스템 내에서 하나의 상태만 `start` 상태로 지정되어 있을 수 있습니다. 이 상태는 최상위 구조의 `StartAt` 필드 값에 따라 지정됩니다. 이 상태는 실행이 시작될 때 제일 먼저 실행되는 상태입니다.
- `End` 필드가 `true`인 상태는 `end`(또는 `terminal`) 상태로 간주됩니다. 상태 시스템 논리에 따라(예: 상태 시스템에 실행 브랜치가 여러 개 있는 경우) `end` 상태가 하나 이상 있을 수도 있습니다.
- 상태 시스템이 하나의 상태로만 구성되어 있는 경우 `start` 상태 및 `end` 상태가 모두 될 수 있습니다.

내장 함수

Amazon States Language는 `Task` 상태를 사용하지 않고도 기본적인 데이터 처리 작업을 수행하는 데 도움이 되는 몇 가지 내장 함수를 제공합니다. 내장 함수는 프로그래밍 언어의 함수와 비슷해 보이는 구성입니다. 이 함수를 사용하여 페이로드 빌더에서 `Resource` 상태의 `Task` 필드를 오가는 데이터를 처리할 수 있습니다.

Amazon States Language에서 내장 함수는 수행하려는 데이터 처리 작업의 유형에 따라 다음과 같은 범주로 그룹화됩니다.

- 배열의 내장 함수
- 데이터 인코딩 및 디코딩을 위한 내장 함수
- 해시 계산을 위한 내장 함수
- JSON 데이터 조작을 위한 내장 함수
- 수학 연산을 위한 내장 함수
- 문자열 작업을 위한 내장 연산
- 고유 식별자 생성을 위한 내장 함수
- 일반 연산을 위한 내장 함수

> **Note**
>
> - 내장 함수를 사용하려면 다음 예제와 같이 상태 시스템 정의의 키 값에 `.$`를 지정해야 합니다.

```
"KeyId.$": "States.Array($.Id)"
```

- 워크플로의 필드 내에 내장 함수를 최대 10개까지 중첩할 수 있습니다. 다음 예제에서는 중첩된 내장 함수 9개가 포함된 *myArn*이라고 하는 필드를 보여줍니다.

```
"myArn.$": "States.Format('{}.{}.{}',
 States.ArrayGetItem(States.StringSplit(States.ArrayGetItem(States.StringSplit($.ImageRe
 '/'), 2), '.'), 0),
 States.ArrayGetItem(States.StringSplit(States.ArrayGetItem(States.StringSplit($.ImageRe
 '/'), 2), '.'), 1))"
```

Tip

로컬 개발 환경에서 Step Functions를 사용하는 경우 워크플로에 모든 내장 함수가 포함될 수 있도록 버전 1.12.0 이상을 사용해야 합니다.

내장 함수를 지원하는 필드

다음 표에서는 각 상태에 내장 함수를 지원하는 필드를 보여줍니다.

내장 함수를 지원하는 필드

State	Pass	작업	Choice	Wait	Succeed	Fail	Parallel	지도
InputPath								
파라미터	✓	✓					✓	✓
ResultSelector		✓					✓	✓
ResultPath								

State	Pass	작업	Choice	Wait	Succeed	Fail	Parallel	지도
OutputPath								
변수								
<Comparison Operator> Path								
TimeoutSecondsPath								
HeartbeatSecondsPath								
보안 인증		✓						

배열의 내장 함수

배열을 조작할 때 다음 내장 함수를 사용하세요.

`States.Array`

`States.Array` 내장 함수는 인수를 0개 이상 사용합니다. 인터프리터는 제공된 순서대로 인수 값이 포함된 JSON 배열을 반환합니다. 다음 입력을 예로 들어보겠습니다.

```
{
  "Id": 123456
}
```

다음을 사용할 수 있습니다.

```
"BuildId.$": "States.Array($.Id)"
```

그러면 응답이 반환됩니다.

```
“BuildId”: [123456]
```

States.ArrayPartition

States.ArrayPartition 내장 함수를 사용하여 큰 배열을 분할할 수 있습니다. 이 내장 함수를 사용하여 데이터를 분할한 다음 작은 청크의 페이로드를 전송할 수도 있습니다.

이 내장 함수에서는 인수 2개를 사용합니다. 첫 번째 인수는 배열이고 두 번째 인수는 청크 크기를 정의합니다. 인터프리터는 입력 배열을 청크 크기로 지정된 크기의 여러 배열로 청크합니다. 배열에 남아 있는 항목 수가 청크 크기보다 작으면 마지막 배열 청크 길이가 이전 배열 청크 길이보다 짧을 수 있습니다.

입력 검증

- 배열을 함수의 첫 번째 인수에 대한 입력 값으로 지정해야 합니다.
- 청크 크기 값을 나타내는 두 번째 인수에는 0이 아닌 양의 정수를 지정해야 합니다.

  두 번째 인수에 정수가 아닌 값을 지정하면 Step Functions에서 가장 가까운 정수로 반올림합니다.
- 입력 배열은 Step Functions의 페이로드 크기 한도인 256KB를 초과할 수 없습니다.

다음 입력 배열을 예로 들어보겠습니다.

```
{"inputArray": [1,2,3,4,5,6,7,8,9] }
```

States.ArrayPartition 함수를 사용하여 배열을 값이 4개인 청크로 나눌 수 있습니다.

```
"inputArray.$": "States.ArrayPartition($.inputArray,4)"
```

그러면 다음과 같은 배열 청크가 반환됩니다.

```
{"inputArray": [ [1,2,3,4], [5,6,7,8], [9]] }
```

이전 예제에서 States.ArrayPartition 함수는 배열 3개를 출력합니다. 처음 두 배열에는 청크 크기로 정의된 대로 각각 값 4개가 포함됩니다. 세 번째 배열은 나머지 값을 포함하며 정의된 청크 크기보다 작습니다.

States.ArrayContains

States.ArrayContains 내장 함수를 사용하여 배열에 특정 값이 있는지 확인할 수 있습니다. 예를 들어 이 함수를 사용하여 Map 상태 반복에 오류가 있었는지 여부를 감지할 수 있습니다.

이 내장 함수에서는 인수 2개를 사용합니다. 첫 번째 인수는 배열이고, 두 번째 인수는 배열 내에서 검색할 값입니다.

입력 검증

- 배열을 함수의 첫 번째 인수에 대한 입력값으로 지정해야 합니다.
- 유효한 JSON 객체를 두 번째 인수로 지정해야 합니다.
- 입력 배열은 Step Functions의 페이로드 크기 한도인 256KB를 초과할 수 없습니다.

다음 입력 배열을 예로 들어보겠습니다.

```
{
   "inputArray": [1,2,3,4,5,6,7,8,9],
   "lookingFor": 5
}
```

States.ArrayContains 함수를 사용하여 inputArray 내에서 lookingFor 값을 찾을 수 있습니다.

```
"contains.$": "States.ArrayContains($.inputArray, $.lookingFor)"
```

lookingFor에 저장된 값이 inputArray에 포함되므로 States.ArrayContains에서 다음과 같은 결과를 반환합니다.

```
{"contains": true }
```

States.ArrayRange

States.ArrayRange 내장 함수를 사용하여 특정 범위의 요소를 포함하는 새 배열을 만들 수 있습니다. 새 배열에는 요소가 최대 1,000개까지 포함될 수 있습니다.

이 함수는 인수 3개를 사용합니다. 첫 번째 인수는 새 배열의 첫 번째 요소이고 두 번째 인수는 새 배열의 마지막 요소입니다. 세 번째 인수는 새 배열의 요소 간 증분 값입니다.

입력 검증

- 모든 인수에 정수 값을 지정해야 합니다.

  인수에 정수가 아닌 값을 지정하면 Step Functions에서 가장 가까운 정수로 반올림합니다.
- 세 번째 인수에는 0이 아닌 값을 지정해야 합니다.
- 새로 생성된 배열에는 항목이 1,000개 넘게 포함될 수 없습니다.

예를 들어 다음 States.ArrayRange 함수를 사용하여 첫 번째 값이 1이고 최종 값이 9이며 첫 번째 값과 최종 값 사이의 값이 각 항목에 대해 2씩 증가하는 배열을 만듭니다.

```
"array.$": "States.ArrayRange(1, 9, 2)"
```

그러면 다음과 같은 배열이 반환됩니다.

```
{"array": [1,3,5,7,9] }
```

States.ArrayGetItem

이 내장 함수는 지정된 인덱스 값을 반환합니다. 이 함수는 인수 2개를 사용합니다. 첫 번째 인수는 값의 배열이고 두 번째 인수는 반환할 값의 배열 인덱스입니다.

예를 들어 다음 inputArray 및 index 값을 사용합니다.

```
{
   "inputArray": [1,2,3,4,5,6,7,8,9],
   "index": 5
}
```

States.ArrayGetItem 함수를 사용하여 다음 값에서 배열 내 index 위치 5의 값을 반환할 수 있습니다.

```
"item.$": "States.ArrayGetItem($.inputArray, $.index)"
```

이 예제의 경우 States.ArrayGetItem에서 다음 결과를 반환합니다.

```
{ "item": 6 }
```

States.ArrayLength

States.ArrayLength 내장 함수는 배열 길이를 반환합니다. 여기에는 인수 하나가 있으며 이 인수는 길이를 반환할 배열입니다.

다음 입력 배열을 예로 들어보겠습니다.

```
{
   "inputArray": [1,2,3,4,5,6,7,8,9]
}
```

States.ArrayLength를 사용하여 inputArray 길이를 반환할 수 있습니다.

```
"length.$": "States.ArrayLength($.inputArray)"
```

이 예제의 경우 States.ArrayLength에서 배열 길이를 나타내는 다음 JSON 객체를 반환합니다.

```
{ "length": 9 }
```

States.ArrayUnique

States.ArrayUnique 내장 함수는 배열에서 중복된 값을 제거하고 고유한 요소만 포함하는 배열을 반환합니다. 이 함수는 정렬되지 않을 수 있는 배열을 유일한 인수로 사용합니다.

예를 들어 다음 inputArray에는 일련의 중복 값이 포함되어 있습니다.

```
{"inputArray": [1,2,3,3,3,3,3,3,4] }
```

States.ArrayUnique 함수를 사용하고 중복된 값을 제거하려는 배열을 지정할 수 있습니다.

```
"array.$": "States.ArrayUnique($.inputArray)"
```

States.ArrayUnique 함수에서 고유한 요소만 포함되어 있는 다음 배열을 반환하고 모든 중복 값을 제거합니다.

```
{"array": [1,2,3,4] }
```

데이터 인코딩 및 디코딩을 위한 내장 함수

다음 내장 함수를 사용하여 Base64 인코딩 체계를 기반으로 데이터를 인코딩하거나 디코딩할 수 있습니다.

States.Base64Encode

States.Base64Encode 내장 함수를 사용하여 MIME Base64 인코딩 체계를 기반으로 데이터를 인코딩할 수 있습니다. 이 함수를 사용하면 AWS Lambda 함수를 사용하지 않고도 데이터를 다른 AWS 서비스에 전달할 수 있습니다.

이 함수는 최대 10,000자의 데이터 문자열을 유일한 인수로 사용하여 인코딩합니다.

예를 들어 다음 input 문자열을 고려해보겠습니다.

```
{"input": "Data to encode" }
```

States.Base64Encode 함수를 사용하여 input 문자열을 MIME Base64 문자열로 인코딩할 수 있습니다.

```
"base64.$": "States.Base64Encode($.input)"
```

States.Base64Encode 함수에서 응답으로 다음 인코딩된 데이터를 반환합니다.

```
{"base64": "RGF0YSB0byBlbmNvZGU=" }
```

States.Base64Decode

States.Base64Decode 내장 함수를 사용하여 MIME Base64 디코딩 체계를 기반으로 데이터를 디코딩할 수 있습니다. 이 함수를 사용하면 Lambda 함수를 사용하지 않고도 데이터를 다른 AWS 서비스에 전달할 수 있습니다.

이 함수는 최대 10,000자의 Base64로 인코딩된 데이터 문자열을 유일한 인수로 사용하여 디코딩합니다.

다음 입력을 예로 들어보겠습니다.

```
{"base64": "RGF0YSB0byBlbmNvZGU=" }
```

States.Base64Decode 함수를 사용하여 base64 문자열을 인간이 읽을 수 있는 문자열로 디코딩할 수 있습니다.

```
"data.$": "States.Base64Decode($.base64)"
```

States.Base64Decode function에서 응답으로 다음과 같은 디코딩된 데이터를 반환합니다.

```
{"data": "Decoded data" }
```

해시 계산을 위한 내장 함수

States.Hash

States.Hash 내장 함수를 사용하여 지정된 입력의 해시 값을 계산할 수 있습니다. 이 함수를 사용하면 Lambda 함수를 사용하지 않고도 데이터를 다른 AWS 서비스에 전달할 수 있습니다.

이 함수는 인수 2개를 사용합니다. 첫 번째 인수는 해시 값을 계산하려는 데이터입니다. 두 번째 인수는 해시 계산을 수행하는 데 사용할 해싱 알고리즘입니다. 제공하는 데이터는 10,000자 이하의 객체 문자열이어야 합니다.

지정하는 해싱 알고리즘은 다음 알고리즘 중 하나일 수 있습니다.

- MD5
- SHA-1
- SHA-256
- SHA-384
- SHA-512

예를 들어 이 함수를 사용하면 지정된 Algorithm를 사용해 Data 문자열의 해시 값을 계산할 수 있습니다.

```
{
   "Data": "input data",
   "Algorithm": "SHA-1"
}
```

States.Hash 함수를 사용하여 해시 값을 계산할 수 있습니다.

```
"output.$": "States.Hash($.Data, $.Algorithm)"
```

States.Hash 함수에서 응답으로 다음 해시 값을 반환합니다.

```
{"output": "aaff4a450a104cd177d28d18d7485e8cae074b7" }
```

JSON 데이터 조작을 위한 내장 함수

다음 함수를 사용하여 JSON 객체에서 기본적인 데이터 처리 작업을 수행할 수 있습니다.

States.JsonMerge

States.JsonMerge 내장 함수를 사용하여 두 JSON 객체를 단일 객체로 병합할 수 있습니다. 이 함수는 인수 3개를 사용합니다. 처음 두 인수는 병합하려는 JSON 객체입니다. 세 번째 인수는 false의 부울 값입니다. 이 부울 값은 심층 병합 모드가 활성화되어 있는지 여부를 결정합니다.

현재 Step Functions는 단순 병합 모드만 지원합니다. 따라서 부울 값을 false로 지정해야 합니다. 단순 모드에서 두 JSON 객체 모두에 같은 키가 있는 경우 후자 객체 키가 첫 번째 객체의 같은 키를 재정의합니다. 또한 단순 병합을 사용할 때는 JSON 객체 내에 중첩된 객체는 병합되지 않습니다.

예를 들어 States.JsonMerge 함수를 사용하여 a 키를 공유하는 다음 JSON 객체를 병합할 수 있습니다.

```
{
   "json1": { "a": {"a1": 1, "a2": 2}, "b": 2 },
   "json2": { "a": {"a3": 1, "a4": 2}, "c": 3 }
}
```

json1 및 json2 객체를 States.JsonMerge 함수의 입력으로 지정하여 병합할 수 있습니다.

```
"output.$": "States.JsonMerge($.json1, $.json2, false)"
```

States.JsonMerge에서 결과로 다음과 같은 병합된 JSON 객체를 반환합니다. 병합된 JSON 객체 output에서는 json2 객체의 a 키가 json1 객체의 a 키를 대체합니다. 또한 단순 모드에서는 중첩된 객체 병합을 지원하지 않으므로 json1 객체의 a 키에 있는 중첩된 객체는 삭제됩니다.

```
{
```

```
   "output": {
      "a": {"a3": 1, "a4": 2},
      "b": 2,
      "c": 3
   }
}
```

States.StringToJson

States.StringToJson 함수는 이스케이프된 JSON 문자열에 대한 참조 경로를 유일한 인수로 사용합니다.

인터프리터는 JSON 파서를 적용하고 입력의 파싱된 JSON 형식을 반환합니다. 예를 들어 이 함수를 사용하여 다음 입력 문자열을 이스케이프 처리할 수 있습니다.

```
{
 "escapedJsonString": "{\"foo\": \"bar\"}"
}
```

States.StringToJson 함수를 사용하고 escapedJsonString을 입력 인수로 지정합니다.

```
States.StringToJson($.escapedJsonString)
```

States.StringToJson 함수에서 다음과 같은 결과를 반환합니다.

```
{ "foo": "bar" }
```

States.JsonToString

States.JsonToString 함수는 이스케이프 처리되지 않은 문자열로 반환할 JSON 데이터가 포함된 Path라는 인수 하나만 사용합니다. 인터프리터는 Path로 지정된 데이터를 나타내는 JSON 텍스트가 포함된 문자열을 반환합니다. 예를 들어 이스케이프 처리된 값이 포함된 다음 JSON Path를 제공할 수 있습니다.

```
{
  "unescapedJson": {
     "foo": "bar"
  }
}
```

unescapedJson 내에 포함된 데이터와 함께 States.JsonToString 함수를 제공합니다.

```
States.JsonToString($.unescapedJson)
```

States.JsonToString 함수에서 다음 응답을 반환합니다.

```
{\"foo\": \"bar\"}
```

수학 연산을 위한 내장 함수

다음 함수를 사용하여 수학 연산을 수행할 수 있습니다.

States.MathRandom

States.MathRandom 내장 함수를 사용하여 지정된 시작 번호(포함)와 끝 번호(제외) 사이의 난수를 반환할 수 있습니다.

이 함수를 사용하여 특정 작업을 리소스 2개 사이에서 분산할 수 있습니다.

이 함수는 인수 3개를 사용합니다. 첫 번째 인수는 시작 번호이고 두 번째 인수는 끝 번호입니다. 마지막 인수는 시드 값을 제어합니다. 시드 값 인수는 선택 사항입니다. 이 함수를 동일한 시드 값과 함께 사용하면 같은 숫자가 반환됩니다.

> **Important**
>
> States.MathRandom 함수에서 암호로 보호되는 난수를 반환하지 않으므로 보안에 민감한 애플리케이션에는 사용하지 않는 것이 좋습니다.

입력 검증

- 시작 번호 인수와 끝 번호 인수에 정수 값을 지정해야 합니다.

  시작 번호 또는 끝 번호 인수에 정수가 아닌 값을 지정하면 Step Functions에서 가장 가까운 정수로 반올림합니다.

예를 들어 1~999 사이의 난수를 생성하려면 다음 입력값을 사용하면 됩니다.

```
{
```

```
   "start": 1,
   "end": 999
}
```

난수를 생성하려면 `start` 및 `end` 값을 `States.MathRandom` 함수에 제공합니다.

```
"random.$": "States.MathRandom($.start, $.end)"
```

`States.MathRandom` 함수에서 응답으로 다음 난수를 반환합니다.

```
{"random": 456 }
```

States.MathAdd

`States.MathAdd` 내장 함수를 사용하여 두 숫자의 합을 반환할 수 있습니다. 예를 들어 이 함수를 사용하여 Lambda 함수를 호출하지 않고도 루프 내에서 값을 증가시킬 수 있습니다.

입력 검증

- 모든 인수에 정수 값을 지정해야 합니다.

  인수 하나 또는 둘 다에 정수가 아닌 값을 지정하면 Step Functions에서 가장 가까운 정수로 반올림합니다.
- -2147483648~2147483647 범위의 정수 값을 지정해야 합니다.

예를 들어 다음 값을 사용하여 111에서 1을 뺄 수 있습니다.

```
{
   "value1": 111,
   "step": -1
}
```

그런 다음, `States.MathAdd` 함수를 사용하여 `value1`을 시작 값으로, `step`을 `value1`씩 증가할 값으로 정의합니다.

```
"value1.$": "States.MathAdd($.value1, $.step)"
```

`States.MathAdd` 함수에서 응답으로 다음 수를 반환합니다.

```
{"value1": 110 }
```

문자열 작업을 위한 내장 연산

States.StringSplit

States.StringSplit 함수를 사용하여 문자열을 값 배열로 분할할 수 있습니다. 이 함수는 인수 2개를 사용합니다. 첫 번째 인수는 문자열이고 두 번째 인수는 함수에서 문자열을 나누는 데 사용할 구분 문자입니다.

Example - 단일 구분 문자를 사용하여 입력 문자열 분할

이 예제에서는 States.StringSplit을 사용하여 쉼표로 구분된 일련의 값이 포함된 다음 inputString을 나눕니다.

```
{
    "inputString": "1,2,3,4,5",
    "splitter": ","
}
```

States.StringSplit 함수를 사용하여 inputString을 첫 번째 인수로, 구분 문자 splitter를 두 번째 인수로 정의합니다.

```
"array.$": "States.StringSplit($.inputString, $.splitter)"
```

States.StringSplit 함수에서 결과로 다음 문자열 배열을 반환합니다.

```
{"array":  ["1","2","3","4","5"] }
```

Example - 여러 구분 문자를 사용하여 입력 문자열 분할

이 예제에서는 States.StringSplit을 사용하여 여러 구분 문자가 포함된 다음 inputString을 나눕니다.

```
{
  "inputString": "This.is+a,test=string",
  "splitter": ".+,="
}
```

States.StringSplit 함수를 다음과 같이 사용합니다.

```
{
```

```
  "myStringArray.$": "States.StringSplit($.inputString, $.splitter)"
}
```

States.StringSplit 함수에서 결과로 다음 문자열 배열을 반환합니다.

```
{"myStringArray": [
  "This",
  "is",
  "a",
  "test",
  "string"
]}
```

고유 식별자 생성을 위한 내장 함수

States.UUID

States.UUID 내장 함수를 사용하여 난수로 생성된 버전 4 범용 고유 식별자(v4 UUID)를 반환할 수 있습니다. 예를 들어 이 함수를 사용하여 UUID 파라미터가 필요한 다른 AWS 서비스 또는 리소스를 호출하거나 항목을 DynamoDB 테이블에 삽입할 수 있습니다.

States.UUID 함수는 지정된 인수 없이 호출됩니다.

```
"uuid.$": "States.UUID()"
```

함수는 다음 예제와 같이 임의로 생성된 UUID를 반환합니다.

```
{"uuid": "ca4c1140-dcc1-40cd-ad05-7b4aa23df4a8" }
```

일반 연산을 위한 내장 함수

States.Format

States.Format 내장 함수를 사용하여 리터럴 값과 보간 값 모두에서 문자열을 구성할 수 있습니다. 이 함수는 인수를 하나 이상 사용합니다. 첫 번째 인수 값은 문자열이어야 하며 0개 이상의 문자 시퀀스 {} 인스턴스를 포함할 수 있습니다. 내장 간접 호출에 남아 있는 인수는 {} 발생 횟수만큼 많아야 합니다. 인터프리터는 첫 번째 인수에 정의된 문자열을 반환합니다. 이 때 각 {}는 내장 간접 호출에서 위치적으로 해당하는 인수의 값으로 대체됩니다.

예를 들어 다음과 같은 개별 name 및 template 문장의 입력 값을 사용하여 이름을 삽입할 수 있습니다.

```
{
 "name": "Arnav",
 "template": "Hello, my name is {}."
}
```

States.Format 함수를 사용하여 template 문자열과 {} 문자 대신 삽입할 문자열을 지정합니다.

```
States.Format('Hello, my name is {}.', $.name)
```

또는

```
States.Format($.template, $.name)
```

States.Format 함수는 이전 입력 중 하나를 사용하여 응답으로 완성된 문자열을 반환합니다.

```
Hello, my name is Arnav.
```

내장 함수에 예약된 문자

'{} 및 \ 문자는 내장 함수용으로 예약되어 있으며 값에 이러한 문자를 표시하려면 백슬래시('\')로 이스케이프 처리해야 합니다.

\ 문자가 이스케이프 문자로 제공되지 않고 값의 일부로 표시되어야 하는 경우에는 백슬래시로 문자를 이스케이프 처리해야 합니다. 내장 함수에서는 다음과 같은 이스케이프 처리된 문자 시퀀스가 사용됩니다.

- 리터럴 문자열 \'은 '를 나타냅니다.
- 리터럴 문자열 \{은 {를 나타냅니다.
- 리터럴 문자열 \}은 }를 나타냅니다.
- 리터럴 문자열 \\은 \를 나타냅니다.

JSON에서 문자열 리터럴 값에 포함된 백슬래시를 다른 백슬래시로 이스케이프 처리해야 합니다. JSON에 해당하는 목록은 다음과 같습니다.

- 이스케이프 처리된 문자열 \\\'은 \'를 나타냅니다.
- 이스케이프 처리된 문자열 \\\{은 \{를 나타냅니다.
- 이스케이프 처리된 문자열 \\\}은 \}를 나타냅니다.
- 이스케이프 처리된 문자열 \\\\은 \\를 나타냅니다.

> **Note**
>
> 내장 간접 호출 문자열에서 열린 이스케이프 백슬래시 \가 발견되면 인터프리터는 런타임 오류를 반환합니다.

일반 상태 필드

`Type`(필수)

상태의 유형입니다.

`Next`

현재 상태가 종료될 때 실행되는 다음 상태의 이름입니다. `Choice`와 같은 일부 상태 유형에서는 여러 개의 전환 상태를 허용합니다.

현재 상태가 워크플로의 마지막 상태이거나 Succeed 또는 Fail과 같은 터미널 상태인 경우 `Next` 필드를 지정할 필요가 없습니다.

`End`

`true`로 설정하면 이 상태가 터미널 상태(실행을 끝냄)로 지정됩니다. 상태 시스템마다 터미널 상태의 수에는 제한이 없습니다. 한 상태에서는 `Next` 또는 `End` 중 하나만 사용될 수 있습니다. 일부 상태 유형(예: `Choice`) 또는 터미널 상태(예: Succeed 및Fail)는 `End` 필드를 지원하거나 사용하지 않습니다.

`Comment`(선택 사항)

육안으로 읽을 수 있는 상태 설명이 들어 있습니다.

`InputPath`(선택 사항)

처리를 위해 상태 작업에 전달되도록 상태의 입력 부분을 선택하는 경로입니다. 생략하는 경우, 전체 입력을 지정하는 `$` 값이 지정됩니다. 자세한 내용은 입/출력 처리를 참조하십시오.

OutputPath(선택 사항)

다음 상태에 전달할 상태 출력 부분을 선택하는 경로입니다. 생략하면 전체 출력을 지정하는 $ 값이 지정됩니다. 자세한 내용은 입/출력 처리를 참조하십시오.

Pass

Pass상태("Type": "Pass")는 작업을 수행하지 않고 입력을 출력으로 전달합니다. Pass 상태는 상태 시스템을 생성하고 디버그할 때 유용합니다.

또한 Pass 상태를 사용하면 필터를 사용하여 JSON 상태 입력을 변환한 다음 변환된 데이터를 워크플로의 다음 상태로 전달할 수 있습니다. 입력 변환에 대한 자세한 내용은 InputPath, 파라미터 및 ResultSelector를 참조하세요.

Pass 상태에서는 일반 상태 필드 외에도 다음 필드를 허용합니다.

Result(선택 사항)

다음 상태로 전달되는 가상 작업의 출력을 나타냅니다. 상태 시스템 정의에 ResultPath 필드를 포함하면 Result가 ResultPath 필드에서 지정한 대로 배치되고 다음 상태로 전달됩니다.

ResultPath(선택 사항)

Result에 지정된 가상 작업의 출력(입력에 상대적)을 배치할 위치를 지정합니다. 그런 다음 OutputPath 필드(있는 경우)에 지정된 대로 입력이 필터링된 후 상태의 출력으로 사용됩니다. 자세한 내용은 입/출력 처리를 참조하십시오.

Parameters(선택 사항)

입력으로 전달될 키-값 페어 컬렉션을 만듭니다. Parameters를 정적 값으로 지정하거나 경로를 사용하여 입력에서 선택할 수 있습니다. 자세한 내용은 InputPath, 파라미터 및 ResultSelector 섹션을 참조하세요.

Pass 상태 예제

다음은 테스트 목적 등으로 상태 시스템에 일부 수정된 데이터를 입력하는 Pass 상태의 예입니다.

```
"No-op": {
  "Type": "Pass",
  "Result": {
    "x-datum": 0.381018,
    "y-datum": 622.2269926397355
```

```
  },
  "ResultPath": "$.coords",
  "End": true
}
```

이 상태의 입력이 다음과 같은 경우를 예로 들어보겠습니다.

```
{
  "georefOf": "Home"
}
```

그러면 다음과 같이 출력됩니다.

```
{
  "georefOf": "Home",
  "coords": {
    "x-datum": 0.381018,
    "y-datum": 622.2269926397355
  }
}
```

태스크 상태

Task 상태("Type": "Task")는 상태 머신에 의해 수행되는 하나의 작업 단위를 나타냅니다. 작업은 액티비티 또는 AWS Lambda 함수를 사용하거나, 지원되는 다른 API와 통합하거나 AWS 서비스, Stripe와 같은 타사 API를 호출하여 작업을 수행합니다.

Amazon States Language는 상태 유형을 Task로 설정하고 태스크에 활동의 Amazon 리소스 이름(ARN)이나 Lambda 함수나 타사 API 엔드포인트를 제공하여 작업을 표시합니다. 다음 Task 상태 정의는 *HelloFunction*라는 Lambda 함수를 간접적으로 호출합니다.

```
"Lambda Invoke": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "Parameters": {
    "Payload.$": "$",
    "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:HelloFunction:
$LATEST"
  },
  "End": true
}
```

이 주제의 내용

Task 유형

Step Functions는 Task 상태 정의에 지정할 수 있는 다음 작업 유형을 지원합니다.

- 활동
- Lambda 함수
- 지원되는 A. AWS 서비스
- HTTP 태스크

Task 상태 정의의 `Resource` 필드에 해당 ARN을 제공하여 작업 유형을 지정합니다. 다음 예제에서는 `Resource` 필드 구문을 보여줍니다. 타사 API를 호출하는 유형을 제외한 모든 태스크 유형은 다음 구문을 사용합니다. HTTP 태스크의 구문에 대한 자세한 정보는 타사 API 호출 섹션을 참조하세요.

작업 상태 정의에서 다음 구문의 기울임꼴 텍스트를 AWS 리소스별 정보로 바꾸십시오.

```
arn:partition:service:region:account:task_type:name
```

다음 목록에서는 이 구문의 개별 구성 요소를 설명합니다.

- `partition`가장 일반적으로 사용할 AWS Step Functions 파티션입니다. `aws`
- `service`작업을 실행하는 AWS 서비스 데 사용되는 값을 나타내며 다음 값 중 하나일 수 있습니다.
  - 활동용 `states`
  - Lambda 함수용 `lambda`. Amazon SNS 또는 Amazon DynamoDB와 같은 다른 AWS 서비스제품과 통합하는 경우 또는 를 사용하십시오. `sns` `dynamodb`
- `region`Step Functions 활동 또는 상태 머신 유형, Lambda 함수 또는 AWS 기타 리소스가 생성된 AWS 지역 코드입니다.
- `account`리소스를 정의한 AWS 계정 ID입니다.
- `task_type`은 실행할 작업 유형입니다. 다음 값 중 하나일 수 있습니다.

  - `activity` - 활동
  - `function` – Lambda 함수
  - *`servicename`* - 지원되는 연결 서비스의 이름(Step Functions를 위한 최적화된 통합 참조)
- `name`은 등록된 리소스 이름(활동 이름, Lambda 함수 이름 또는 서비스 API 작업)입니다.

> **Note**
>
> Step Functions는 파티션 또는 리전 전체에서 ARN 참조를 지원하지 않습니다. 예를 들어 `aws-cn`이 `aws` 파티션에서 작업을 간접적으로 호출할 수 없으며 그 반대의 경우에는 마찬가지입니다.

다음 단원에서는 각 작업 유형에 대한 자세한 내용이 제공됩니다.

활동

활동은 사용자가 구현하고 호스팅하며 특정 작업을 수행하는 작업자(프로세스나 스레드)를 나타냅니다. 이러한 활동은 표준 워크플로에서만 지원되며 Express 워크플로에서는 지원되지 않습니다.

활동 `Resource` ARN은 다음 구문을 사용합니다.

```
arn:partition:states:region:account:activity:name
```

> **Note**
>
> 처음 사용하기 전에 Step Functions를 사용하여 (API 작업 또는 Step Functions 콘솔을 사용하여) 활동을 생성해야 합니다. CreateActivity

활동 생성 및 작업자 구현에 대한 자세한 내용은 활동을 참조하십시오.

Lambda 함수

Lambda 작업은 를 사용하여 함수를 실행합니다. AWS Lambda Lambda 함수를 지정하려면 `Resource` 필드에서 Lambda 함수의 ARN을 사용합니다.

Lambda 함수를 지정하는 데 사용하는 통합 유형(최적화된 통합 또는 AWS SDK 통합)에 따라 Lambda 함수의 `Resource` 필드 구문이 달라집니다.

다음 Resource 필드 구문은 Lambda 함수와 최적화된 통합의 예제입니다.

```
"arn:aws:states:::lambda:invoke"
```

다음 Resource 필드 구문은 Lambda 함수와의 AWS SDK 통합 예제입니다.

```
"arn:aws:states:::aws-sdk:lambda:invoke"
```

다음 Task 상태 정의에서는 *HelloWorld*라는 Lambda 함수와 최적화된 통합의 예제를 보여줍니다.

```
"LambdaState": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "OutputPath": "$.Payload",
  "Parameters": {
    "Payload.$": "$",
    "FunctionName": "arn:aws:lambda:us-east-1:function:HelloWorld:$LATEST"
  },
  "Next": "NextState"
}
```

필드에 지정된 Lambda 함수가 완료되면 출력이 필드 (" ") 에서 Next 식별된 상태로 전송됩니다. Resource NextState

지원되는 A. AWS 서비스

연결된 리소스를 참조할 때 Step Functions는 지원되는 서비스의 API 작업을 직접 호출합니다. Resource 필드에서 서비스와 작업을 지정합니다.

연결된 서비스 Resource ARN은 다음 구문을 사용합니다.

```
arn:partition:states:region:account:servicename:APIname
```

> **Note**
>
> 연결된 리소스에 대한 동기 연결을 만들려면 .sync을 ARN의 *APIname* 항목에 추가하세요. 자세한 정보는 다른 서비스와 함께 사용을 참조하세요.

예:

```
{
 "StartAt": "BATCH_JOB",
 "States": {
   "BATCH_JOB": {
     "Type": "Task",
     "Resource": "arn:aws:states:::batch:submitJob.sync",
     "Parameters": {
       "JobDefinition": "preprocessing",
       "JobName": "PreprocessingBatchJob",
       "JobQueue": "SecondaryQueue",
       "Parameters.$": "$.batchjob.parameters",
       "RetryStrategy": {
          "attempts": 5
        }
     },
     "End": true
    }
  }
}
```

Task 상태 필드

Task 상태에서는 일반 상태 필드 외에도 다음 필드가 있습니다.

`Resource`(필수)

URI, 특히 실행할 특정 작업을 고유하게 식별하는 ARN입니다.

`Parameters` (선택 사항)

연결된 리소스의 API 작업으로 정보를 전달합니다. 파라미터는 정적 JSON과 JsonPathJSON을 혼합하여 사용할 수 있습니다. 자세한 정보는 파라미터를 서비스 API에 전달을 참조하세요.

`Credentials` (선택 사항)

지정된 Resource 역할을 간접적으로 호출하기 전에 상태 시스템의 실행 역할이 간주되어야 하는 대상 역할을 지정합니다. 또는 JSONPath 값 또는 실행 입력을 기반으로 런타임 시 IAM 역할 ARN으로 확인되는 내장 함수를 지정할 수도 있습니다. JSONPath 값을 지정하는 경우 `$.` 표기법으로 접두사를 추가해야 합니다.

Task 상태에서 이 필드 사용에 대한 예제는 Task 상태의 보안 인증 정보 필드 예제 섹션을 참조하세요. 이 필드를 사용하여 상태 머신에서 계정 간 AWS 리소스에 액세스하는 예제는 을 참조하십시오. 자습서: 교차 AWS 계정 리소스 액세스

> **Note**
>
> 이 필드는 Lambda 함수를 Task 유형 사용하는 사용자 및 지원되는 서비스에서 지원됩니다. AWS

`ResultPath` (선택 사항)

`Resource`에 지정된 작업 실행 결과를 배치할 위치(입력에서의 위치)를 지정합니다. 그러면 `OutputPath` 필드(있는 경우)에 지정된 대로 입력이 필터링된 후 상태의 출력으로 사용됩니다. 자세한 내용은 입/출력 처리를 참조하십시오.

`ResultSelector` (선택 사항)

키 값 페어 컬렉션을 전달합니다. 여기서 값은 정적이거나 결과에서 선택됩니다. 자세한 정보는 ResultSelector을 참조하세요.

`Retry` (선택 사항)

상태에 런타임 오류가 발생하는 경우 사용될 재시도 정책을 정의하는 객체 배열(Retrier). 자세한 정보는 Retry 및 Catch를 사용하는 상태 시스템 예제을 참조하세요.

`Catch` (선택 사항)

폴백(fallback) 상태를 정의하는 객체 배열(Catcher). 이 상태는, 상태에 런타임 오류가 발생하거나 해당 재시도 정책이 소진되거나 정의되지 않은 경우 실행됩니다. 자세한 내용은 폴백 상태를 참조하십시오.

`TimeoutSeconds` (선택 사항)

States.Timeout 오류로 인해 제한 시간이 초과되어 실패하기 전에 활동이나 작업이 실행될 수 있는 최대 시간을 지정합니다. 제한 시간 값은 0이 아닌 양의 정수여야 합니다. 기본 값은 99999999입니다.

작업이 시작된 후에(예: `ActivityStarted` 또는 `LambdaFunctionStarted` 이벤트가 실행 이벤트 기록에 로깅될 때) 제한 시간 카운트가 시작합니다. 활동의 경우 `GetActivityTask`에서 토큰을 받고 `ActivityStarted`가 실행 이벤트 기록에서 로깅될 때 카운트가 시작합니다.

작업이 시작되면 Step Functions는 지정된 `TimeoutSeconds` 기간 내에 작업이나 활동 작업자의 성공 또는 실패 응답을 기다립니다. 작업이나 활동 작업자가 이 시간 내에 응답하지 않으면 Step Functions는 워크플로 실행을 실패로 표시합니다.

`TimeoutSecondsPath` (선택 사항)

참조 경로를 사용하여 상태 입력에서 시간 제한 값을 동적으로 제공하려면 `TimeoutSecondsPath`를 사용합니다. 확인되면 참조 경로에서 값이 양의 정수인 필드를 선택해야 합니다.

> **Note**
>
> `Task` 상태에는 `TimeoutSeconds` 및 `TimeoutSecondsPath` 모두 포함될 수 없습니다.

`HeartbeatSeconds` (선택 사항)

작업 실행 중에 활동 작업자가 하트비트 신호를 보내는 빈도를 결정합니다. 하트비트는 작업이 계속 실행 중이며 완료되는 데 시간이 더 필요하다는 것을 나타냅니다. 하트비트는 `TimeoutSeconds` 기간 내에 활동이나 작업의 제한 시간이 초과되지 않도록 합니다.

`HeartbeatSeconds`는 0이 아닌 양의 정수 값이고 `TimeoutSeconds` 필드 값보다 작아야 합니다. 기본 값은 99999999입니다. 작업의 하트비트 사이에서 지정된 시간(초)보다 많은 시간이 경과하면 Task 상태가 실패하고 States.Timeout 오류가 표시됩니다.

활동의 경우 `GetActivityTask`에서 토큰을 받고 `ActivityStarted`가 실행 이벤트 기록에서 로깅될 때 카운트가 시작합니다.

`HeartbeatSecondsPath` (선택 사항)

참조 경로를 사용하여 상태 입력에서 하트비트 값을 동적으로 제공하려면 `HeartbeatSecondsPath`를 사용합니다. 확인되면 참조 경로에서 값이 양의 정수인 필드를 선택해야 합니다.

> **Note**
>
> `Task` 상태에는 `HeartbeatSeconds` 및 `HeartbeatSecondsPath` 모두 포함될 수 없습니다.

`Task` 상태가 `End` 필드를 `true`로 설정해야 하거나(상태가 실행을 끝내는 경우), `Task` 상태의 완료 시에 실행되는 `Next` 필드에 상태를 제공해야 합니다.

Task 상태 정의 예제

다음 예제에서는 요구 사항에 따라 Task 상태 정의를 지정하는 방법을 보여줍니다.

- Task 상태 제한 시간 및 하트비트 간격 지정
  - 정적 제한 시간 및 하트비트 알림 예제
  - 동적 작업 제한 시간 및 하트비트 알림 예제
- 보안 인증 정보 필드 사용
  - 하드 코딩된 IAM 역할 ARN 지정
  - JSONPath를 IAM 역할 ARN으로 지정
  - 내장 함수를 IAM 역할 ARN으로 지정

Task 상태 제한 시간 및 하트비트 간격

실행 시간이 긴 활동의 경우 시간 초과 값 및 하트비트 간격을 설정하는 것이 좋습니다. 제한 시간 및 하트비트 값을 지정하거나 동적으로 설정하여 수행할 수 있습니다.

정적 제한 시간 및 하트비트 알림 예제

`HelloWorld`가 완료되면 다음 단계(`NextState`)가 실행됩니다.

이 작업이 300초 내에 완료되지 않거나 60초 간격으로 하트비트 알림을 전송하지 않으면 작업이 `failed`로 표시됩니다.

```
"ActivityState": {
  "Type": "Task",
  "Resource": "arn:aws:states:us-east-1:123456789012:activity:HelloWorld",
  "TimeoutSeconds": 300,
  "HeartbeatSeconds": 60,
  "Next": "NextState"
}
```

동적 작업 제한 시간 및 하트비트 알림 예제

이 예시에서는 AWS Glue 작업이 완료되면 다음 상태가 실행됩니다.

이 작업이 AWS Glue 작업에서 동적으로 설정한 간격 내에 완료되지 못하면 작업은 `failed`로 표시됩니다.

```
"GlueJobTask": {
  "Type": "Task",
  "Resource": "arn:aws:states:::glue:startJobRun.sync",
  "Parameters": {
    "JobName": "myGlueJob"
  },
  "TimeoutSecondsPath": "$.params.maxTime",

  "Next": "NextState"
}
```

Task 상태의 보안 인증 정보 필드 예제

하드 코딩된 IAM 역할 ARN 지정

다음 예제에서는 Echo라는 교차 계정 Lambda 함수에 액세스하기 위해 상태 시스템 실행 역할이 간주되어야 하는 대상 IAM 역할을 지정합니다. 이 예제에서 대상 역할 ARN은 하드 코딩된 값으로 지정됩니다.

```
{
  "StartAt": "Cross-account call",
  "States": {
    "Cross-account call": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Credentials": {
        "RoleArn": "arn:aws:iam::111122223333:role/LambdaRole"
      },
      "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-2:111122223333:function:Echo"
      },
      "End": true
    }
  }
}
```

JSONPath를 IAM 역할 ARN으로 지정

다음 예제에서는 런타임 시 IAM 역할 ARN으로 확인되는 JSONPath 값을 지정합니다.

```
{
  "StartAt": "Lambda",
```

```
  "States": {
    "Lambda": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Credentials": {
        "RoleArn.$": "$.roleArn"
      },
      ...
    }
  }
}
```

내장 함수를 IAM 역할 ARN으로 지정

다음 예제에서는 런타임 시 IAM 역할 ARN으로 확인되는 `States.Format` 내장 함수를 사용합니다.

```
{
  "StartAt": "Lambda",
  "States": {
    "Lambda": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Credentials": {
        "RoleArn.$": "States.Format('arn:aws:iam::{}:role/ROLENAME', $.accountId)"
      },
      ...
    }
  }
}
```

활동

활동은 AWS Step Functions 특성으로, 이 특성을 통해 Amazon Elastic Compute Cloud(Amazon EC2), Amazon Elastic Container Service(Amazon ECS), 모바일 디바이스 등 거의 모든 곳에서 호스팅될 수 있는 작업자가 작업을 수행하는 상태 시스템에서 작업을 수행할 수 있습니다.

개요

AWS Step Functions에서 활동은 상태 시스템에서 (활동 작업자라고 함) 어디서든 실행하는 코드를 특정 작업과 연결하는 방법입니다. Step Functions 콘솔을 사용하거나 `CreateActivity`를 직접적으로 호출하여 활동을 만들 수 있습니다. 이렇게 하면 작업 상태에 대한 Amazon 리소스 이름(ARN)이 제공됩니다. 이 ARN을 사용하여 활동 작업자의 작업에 대한 작업 상태를 폴링하십시오.

> **Note**
>
> 활동은 버전 관리되지 않으며 이전 버전과 호환이 가능합니다. 이전 버전과 호환되지 않는 방식으로 활동을 변경해야 하는 경우 Step Functions에서 고유한 이름을 사용하여 새 활동을 만드세요.

활동 작업자는 Amazon EC2 인스턴스, AWS Lambda 함수, 모바일 디바이스에서 실행하는 애플리케이션이 될 수 있습니다. HTTP를 연결할 수 있는 모든 애플리케이션이라면 위치와 무관하게 호스팅됩니다. Step Functions에서 활동 작업 상태에 이르면 워크플로는 활동 작업자가 작업을 폴링할 때까지 기다립니다. 활동 작업자는 `GetActivityTask`를 사용하고 관련된 활동의 ARN을 전송하여 Step Functions를 폴링합니다. `GetActivityTask`는 `input`(작업에 대한 JSON 입력 문자열) 및 `taskToken`(고유한 작업 식별자)을 포함하여 응답을 반환합니다. 활동 작업자는 업무를 완료 후 `SendTaskSuccess` 또는 `SendTaskFailure`를 사용하여 성공 또는 실패 보고서를 작성할 수 있습니다. 이 두 개의 호출에서는 이 작업의 결과를 연결하기 위해 `GetActivityTask`에서 제공한 `taskToken`을 사용합니다.

활동 작업 관련 API

Step Functions를 통해 API는 활동을 생성 및 열거하고 작업을 요청하며 작업자 결과를 기반으로 상태 시스템 흐름을 관리합니다.

다음은 활동과 관련된 Step Functions API입니다.

- `CreateActivity`
- `GetActivityTask`
- `ListActivities`
- `SendTaskFailure`
- `SendTaskHeartbeat`
- `SendTaskSuccess`

> **Note**
>
> `GetActivityTask`를 사용하여 활동 작업에 대해 폴링하면 일부 구현에서 지연 시간이 발생할 수 있습니다. 활동 작업을 폴링할 때 지연 시간 방지 섹션을 참조하세요.

활동 작업이 완료할 때까지 대기

작업 정의에서 `TimeoutSeconds`를 설정하여 상태 대기 시간을 설정하십시오. 작업 활성화 및 대기 상태를 유지하려면 `TimeoutSeconds`에 설정된 시간 내에서 `SendTaskHeartbeat`를 사용하여 활동 작업자의 하트비트를 정기적으로 전송하십시오. 긴 제한 시간을 구성하고 하트비트를 능동적으로 전송하면 Step Functions의 활동은 실행이 완료될 때까지 최대 1년을 대기할 수 있습니다.

예를 들어, 시간이 오래 걸리는 프로세스의 결과를 대기하는 워크플로우가 필요한 경우 다음과 같이 하십시오.

1. 콘솔을 사용하거나 `CreateActivity`를 사용하여 활동을 생성합니다. 활동 ARN을 메모하십시오.
2. 상태 시스템 정의에서 이 ARN의 활동 작업 상태를 참조하여 `TimeoutSeconds`를 설정하십시오.
3. 이 활동 ARN을 참조하여 `GetActivityTask`를 통해 작업에 대해 폴링하는 활동 작업자를 구현하십시오.
4. 작업이 시간 초과되지 않도록 하려면 상태 시스템 작업 정의에서 `HeartbeatSeconds`에 설정한 시간 내에서 `SendTaskHeartbeat`를 정기적으로 사용하십시오.
5. 상태 시스템의 실행을 시작하십시오.
6. 활동 작업자의 프로세스를 시작하십시오.

이 활동 작업 상태에 이르면 실행이 일시 정지하고 활동 작업자가 작업에 대해 폴링할 때까지 대기합니다. 활동 작업자에게 `taskToken`이 제공되면 워크플로우는 `SendTaskSuccess` 또는 `SendTaskFailure`이 상태를 제공할 때까지 대기합니다. 이 실행에서 `TimeoutSeconds`에 구성된 시간보다 먼저 이 가운데 하나 또는 `SendTaskHeartbeat` 호출을 수신하지 않으면 실행이 실패하고 실행 기록에 `ExecutionTimedOut` 이벤트가 포함됩니다.

다음 단계

활동 작업자를 사용하는 상태 머신을 생성하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- Step Functions를 사용하여 Activity 상태 시스템 만들기
- Ruby 활동 작업자 예제

Ruby 활동 작업자 예제

다음은 AWS SDK for Ruby를 사용하여 모범 사례 사용 방법과 활동 작업자 구현 방법을 보여주는 활동 작업자 예제입니다.

이 코드는 폴러 및 활동 작업자용 스레드 수를 구성할 수 있는 소비자-생산자 패턴을 구현합니다. 폴러 스레드는 지속적으로 활동 작업을 폴링합니다. 검색된 활동 작업은 활동 스레드가 픽업할 수 있도록 경계가 있는 차단 대기열을 통해 전달됩니다.

- AWS SDK for Ruby에 대한 자세한 내용은 AWS SDK for Ruby API 참조를 확인하세요.
- 이 코드 및 관련 리소스를 다운로드하려면 GitHub에서 step-functions-ruby-activity-worker 리포지토리를 참조하십시오.

아래 Ruby 코드는 이 Ruby 활동 작업자 예제에 대한 주 진입점입니다.

```
require_relative '../lib/step_functions/activity'
credentials = Aws::SharedCredentials.new
region = 'us-west-2'
activity_arn = 'ACTIVITY_ARN'

activity = StepFunctions::Activity.new(
  credentials: credentials,
  region: region,
  activity_arn: activity_arn,
  workers_count: 1,
  pollers_count: 1,
  heartbeat_delay: 30
)

# The start method takes as argument the block that is the actual logic of your custom
activity.
activity.start do |input|
  { result: :SUCCESS, echo: input['value'] }
end
```

이 코드에는 기본값이 포함되어 있으며, 사용자가 각 값을 본인의 활동을 참조하도록 변경하고 특정 구현에 맞춰 조정할 수 있습니다. 이 코드는 실제 구현 로직을 입력으로 사용하며 사용자가 특정 활동 및 자격 증명을 참조하고 스레드 수 및 하트비트 지연을 구성할 수 있습니다. 코드에 대한 자세한 내용과 다운로드 방법은 Step Functions Ruby Activity Worker를 참조하세요.

항목	설명
`require_relative`	다음 활동 작업자 코드 예제의 상대 경로입니다.

항목	설명
region	AWS 활동의 리전입니다.
workers_count	활동 작업자용 스레드 수입니다. 대부분의 구현에서는 10~20개 스레드면 충분할 것입니다. 활동 처리 시간이 길수록 더 많은 스레드가 필요할 수 있습니다. 초당 프로세스 활동 수에 활동 처리 지연 시간(초)의 99 백분위를 곱하면 추정치를 구할 수 있습니다.
pollers_count	폴러용 스레드 수입니다. 대부분의 구현에서는 10~20개 스레드면 충분할 것입니다.
heartbeat_delay	하트비트 간 지연 시간(초)입니다.
input	활동의 구현 로직입니다.

다음은 코드에서 `../lib/step_functions/activity`로 참조되는 Ruby 활동 작업자입니다.

```
require 'set'
require 'json'
require 'thread'
require 'logger'
require 'aws-sdk'

module Validate
  def self.positive(value)
    raise ArgumentError, 'Argument has to be positive' if value <= 0
    value
  end

  def self.required(value)
    raise ArgumentError, 'Argument is required' if value.nil?
    value
  end
end

module StepFunctions
  class RetryError < StandardError
```

```
    def initialize(message)
      super(message)
    end
  end

  def self.with_retries(options = {}, &block)
    retries = 0
    base_delay_seconds = options[:base_delay_seconds] || 2
    max_retries = options[:max_retries] || 3
    begin
      block.call
    rescue => e
      puts e
      if retries < max_retries
        retries += 1
        sleep base_delay_seconds**retries
        retry
      end
      raise RetryError, 'All retries of operation had failed'
    end
  end

  class Activity
    def initialize(options = {})
      @states = Aws::States::Client.new(
        credentials: Validate.required(options[:credentials]),
        region: Validate.required(options[:region]),
        http_read_timeout: Validate.positive(options[:http_read_timeout] || 60)
      )
      @activity_arn = Validate.required(options[:activity_arn])
      @heartbeat_delay = Validate.positive(options[:heartbeat_delay] || 60)
      @queue_max = Validate.positive(options[:queue_max] || 5)
      @pollers_count = Validate.positive(options[:pollers_count] || 1)
      @workers_count = Validate.positive(options[:workers_count] || 1)
      @max_retry = Validate.positive(options[:workers_count] || 3)
      @logger = Logger.new(STDOUT)
    end

    def start(&block)
      @sink = SizedQueue.new(@queue_max)
      @activities = Set.new
      start_heartbeat_worker(@activities)
      start_workers(@activities, block, @sink)
      start_pollers(@activities, @sink)
```

```
      wait
    end

    def queue_size
      return 0 if @sink.nil?
      @sink.size
    end

    def activities_count
      return 0 if @activities.nil?
      @activities.size
    end

    private

    def start_pollers(activities, sink)
      @pollers = Array.new(@pollers_count) do
        PollerWorker.new(
          states: @states,
          activity_arn: @activity_arn,
          sink: sink,
          activities: activities,
          max_retry: @max_retry
        )
      end
      @pollers.each(&:start)
    end

    def start_workers(activities, block, sink)
      @workers = Array.new(@workers_count) do
        ActivityWorker.new(
          states: @states,
          block: block,
          sink: sink,
          activities: activities,
          max_retry: @max_retry
        )
      end
      @workers.each(&:start)
    end

    def start_heartbeat_worker(activities)
      @heartbeat_worker = HeartbeatWorker.new(
        states: @states,
```

```
        activities: activities,
        heartbeat_delay: @heartbeat_delay,
        max_retry: @max_retry
      )
      @heartbeat_worker.start
    end

    def wait
      sleep
    rescue Interrupt
      shutdown
    ensure
      Thread.current.exit
    end

    def shutdown
      stop_workers(@pollers)
      wait_workers(@pollers)
      wait_activities_drained
      stop_workers(@workers)
      wait_activities_completed
      shutdown_workers(@workers)
      shutdown_worker(@heartbeat_worker)
    end

    def shutdown_workers(workers)
      workers.each do |worker|
        shutdown_worker(worker)
      end
    end

    def shutdown_worker(worker)
      worker.kill
    end

    def wait_workers(workers)
      workers.each(&:wait)
    end

    def wait_activities_drained
      wait_condition { @sink.empty? }
    end

    def wait_activities_completed
```

```
    wait_condition { @activities.empty? }
  end

  def wait_condition(&block)
    loop do
      break if block.call
      sleep(1)
    end
  end

  def stop_workers(workers)
    workers.each(&:stop)
  end

  class Worker
    def initialize
      @logger = Logger.new(STDOUT)
      @running = false
    end

    def run
      raise 'Method run hasn\'t been implemented'
    end

    def process
      loop do
        begin
          break unless @running
          run
        rescue => e
          puts e
          @logger.error('Unexpected error has occurred')
          @logger.error(e)
        end
      end
    end

    def start
      return unless @thread.nil?
      @running = true
      @thread = Thread.new do
        process
      end
    end
```

```
  def stop
    @running = false
  end

  def kill
    return if @thread.nil?
    @thread.kill
    @thread = nil
  end

  def wait
    @thread.join
  end
end

class PollerWorker < Worker
  def initialize(options = {})
    @states = options[:states]
    @activity_arn = options[:activity_arn]
    @sink = options[:sink]
    @activities = options[:activities]
    @max_retry = options[:max_retry]
    @logger = Logger.new(STDOUT)
  end

  def run
    activity_task = StepFunctions.with_retries(max_retry: @max_retry) do
      begin
        @states.get_activity_task(activity_arn: @activity_arn)
      rescue => e
        @logger.error('Failed to retrieve activity task')
        @logger.error(e)
      end
    end
    return if activity_task.nil? || activity_task.task_token.nil?
    @activities.add(activity_task.task_token)
    @sink.push(activity_task)
  end
end

class ActivityWorker < Worker
  def initialize(options = {})
    @states = options[:states]
```

```
        @block = options[:block]
        @sink = options[:sink]
        @activities = options[:activities]
        @max_retry = options[:max_retry]
        @logger = Logger.new(STDOUT)
      end

      def run
        activity_task = @sink.pop
        result = @block.call(JSON.parse(activity_task.input))
        send_task_success(activity_task, result)
      rescue => e
        send_task_failure(activity_task, e)
      ensure
        @activities.delete(activity_task.task_token) unless activity_task.nil?
      end

      def send_task_success(activity_task, result)
        StepFunctions.with_retries(max_retry: @max_retry) do
          begin
            @states.send_task_success(
              task_token: activity_task.task_token,
              output: JSON.dump(result)
            )
          rescue => e
            @logger.error('Failed to send task success')
            @logger.error(e)
          end
        end
      end

      def send_task_failure(activity_task, error)
        StepFunctions.with_retries do
          begin
            @states.send_task_failure(
              task_token: activity_task.task_token,
              cause: error.message
            )
          rescue => e
            @logger.error('Failed to send task failure')
            @logger.error(e)
          end
        end
      end
```

```
    end

    class HeartbeatWorker < Worker
      def initialize(options = {})
        @states = options[:states]
        @activities = options[:activities]
        @heartbeat_delay = options[:heartbeat_delay]
        @max_retry = options[:max_retry]
        @logger = Logger.new(STDOUT)
      end

      def run
        sleep(@heartbeat_delay)
        @activities.each do |token|
          send_heartbeat(token)
        end
      end

      def send_heartbeat(token)
        StepFunctions.with_retries(max_retry: @max_retry) do
          begin
            @states.send_task_heartbeat(token)
          rescue => e
            @logger.error('Failed to send heartbeat for activity')
            @logger.error(e)
          end
        end
      rescue => e
        @logger.error('Failed to send heartbeat for activity')
        @logger.error(e)
      end
    end
  end
end
```

Choice

`Choice` 상태(`"Type": "Choice"`)는 조건부 논리를 상태 시스템에 추가합니다.

`Choice` 상태에는 대부분의 일반 상태 필드 외에도 다음과 같은 필드가 추가로 포함되어 있습니다.

`Choices`(필수)

상태 시스템이 다음으로 전환하는 상태를 결정하는 선택 규칙의 어레이. 선택 규칙의 비교 연산자를 사용하여 입력 변수를 특정 값과 비교할 수 있습니다. 예를 들어 선택 규칙을 사용하면 입력 변수가 100보다 큰지 또는 작은지 비교할 수 있습니다.

`Choice` 상태가 실행되면 각 선택 규칙을 true 또는 false로 평가합니다. 이 평가 결과에 따라 Step Functions는 워크플로의 다음 상태로 전환됩니다.

`Choice` 상태에 규칙을 최소 하나 이상 정의해야 합니다.

`Default`(선택 사항, 권장됨)

`Choices`의 전환 중 하나도 수행되지 않는 경우 전환될 상태의 이름.

> **Important**
>
> `Choice` 상태는 `End` 필드를 지원하지 않습니다. 또한, `Next`는 `Choices` 필드 내에서만 사용됩니다.

> **Tip**
>
> `Choice` 상태를 사용하는 워크플로 예제를 AWS 계정에 배포하려면 AWS Step Functions 워크숍의 모듈 5 - 선택 상태 및 Map 상태를 참조하세요.

선택 규칙

`Choice` 상태에는 값이 비어 있지 않은 배열인 `Choices` 필드가 있어야 합니다. 이 배열의 각 요소는 선택 규칙이라는 객체이며, 이 객체에는 다음이 포함됩니다.

- 비교값 - 비교할 입력 변수, 비교 유형 및 변수와 비교할 값을 지정하는 필드 2개. 선택 규칙에서는 두 변수를 비교할 수 있습니다. 선택 규칙 내에서 `Path`를 지원되는 비교 연산자 이름에 추가하여 변수 값을 상태 입력의 다른 값과 비교할 수 있습니다. 비교할 때 `Variable` 및 경로 필드의 값은 유효한 참조 경로여야 합니다.
- **`Next`** 필드 - 이 필드 값은 상태 시스템의 상태 이름과 일치해야 합니다.

다음은 숫자 값이 1과 동일한지 여부를 검사하는 예제입니다.

```
{
  "Variable": "$.foo",
  "NumericEquals": 1,
  "Next": "FirstMatchState"
}
```

다음은 문자열이 MyString과 동일한지 여부를 검사하는 예제입니다.

```
{
  "Variable": "$.foo",
  "StringEquals": "MyString",
  "Next": "FirstMatchState"
}
```

다음은 문자열이 MyStringABC보다 큰지를 검사하는 예제입니다.

```
{
  "Variable": "$.foo",
  "StringGreaterThan": "MyStringABC",
  "Next": "FirstMatchState"
}
```

다음은 문자열이 null인지 여부를 검사하는 예제입니다.

```
{
 "Variable": "$.possiblyNullValue",
 "IsNull": true
}
```

다음 예제에서는 이전 IsPresent 선택 규칙으로 인해 $.keyThatMightNotExist가 존재하는 경우에만 StringEquals가 평가되는 방식을 보여줍니다.

```
"And": [
 {
 "Variable": "$.keyThatMightNotExist",
 "IsPresent": true
 },
 {
 "Variable": "$.keyThatMightNotExist",
 "StringEquals": "foo"
```

```
 }
]
```

다음은 와일드카드가 있는 패턴이 일치하는지 여부를 검사하는 예제입니다.

```
{
 "Variable": "$.foo",
 "StringMatches": "log-*.txt"
}
```

다음은 타임스탬프가 2001-01-01T12:00:00Z과 동일한지 여부를 검사하는 예제입니다.

```
{
  "Variable": "$.foo",
  "TimestampEquals": "2001-01-01T12:00:00Z",
  "Next": "FirstMatchState"
}
```

다음은 변수를 상태 입력의 다른 값과 비교하는 예제입니다.

```
{
 "Variable": "$.foo",
 "StringEqualsPath": "$.bar"
}
```

Step Functions는 Choices 필드에 명시된 순서에 따라 각 선택 규칙을 검사합니다. 그런 다음 비교 연산자에 따라 변수가 값과 일치하는 첫 번째 선택 규칙의 Next 필드에 지정된 상태로 전환합니다.

다음 비교 연산자가 지원됩니다.

- And
- BooleanEquals,BooleanEqualsPath
- IsBoolean
- IsNull
- IsNumeric
- IsPresent
- IsString
- IsTimestamp

- Not
- NumericEquals,NumericEqualsPath
- NumericGreaterThan,NumericGreaterThanPath
- NumericGreaterThanEquals,NumericGreaterThanEqualsPath
- NumericLessThan,NumericLessThanPath
- NumericLessThanEquals,NumericLessThanEqualsPath
- Or
- StringEquals,StringEqualsPath
- StringGreaterThan,StringGreaterThanPath
- StringGreaterThanEquals,StringGreaterThanEqualsPath
- StringLessThan,StringLessThanPath
- StringLessThanEquals,StringLessThanEqualsPath
- StringMatches
- TimestampEquals,TimestampEqualsPath
- TimestampGreaterThan,TimestampGreaterThanPath
- TimestampGreaterThanEquals,TimestampGreaterThanEqualsPath
- TimestampLessThan,TimestampLessThanPath
- TimestampLessThanEquals,TimestampLessThanEqualsPath

이러한 연산자마다 해당 값이 적절한 유형(문자열, 숫자, 부울 또는 타임스탬프)이어야 합니다. Step Functions는 숫자 필드를 문자열 값과 일치시키려고 시도하지 않습니다. 단, 타임스탬프 필드는 논리적으로 문자열이므로 타임스탬프로 간주되는 필드는 StringEquals 연산자와 연결될 수 있습니다.

> **Note**
>
> 상호 운영성을 위해, IEEE 754-2008 binary64 데이터 유형에 표시되는 크기나 정밀도를 넘는 값을 사용하여 수 비교를 수행하지 마십시오. 특히, $[-2^{53}+1, 2^{53}-1]$ 범위를 넘는 정수를 사용하면 예상되는 방식으로 비교가 수행되지 않을 수 있습니다.
> 타임스탬프(예: 2016-08-18T17:33:00Z)는 RFC3339 프로필 ISO 8601을 준수하되 다음과 같은 추가 제한도 따라야 합니다.
>
> - 대문자 T로 날짜와 시간 부분을 구분해야 합니다.
> - 대문자 Z로 숫자로 된 시간대 오프셋이 없음을 표기해야 합니다.

문자열 비교 동작을 이해하려면 Java compareTo 설명서를 참조하십시오.
And 및 Or 연산자는 비어 있지 않은 선택 규칙 어레이여야 하며 Next 필드를 자체적으로 포함하지 않아야 합니다. 마찬가지로, Not 연산자는 단일 선택 규칙이어야 하며 Next 필드를 포함하지 않아야 합니다.
And, Not 및 Or를 사용하면 복잡한 중첩 선택 규칙을 생성할 수 있습니다. 단, Next 필드는 최상위 선택 규칙에만 표시될 수 있습니다.
StringMatches 비교 연산자를 사용하여 문자열을 와일드카드("*")가 하나 이상 있는 패턴과 비교할 수 있습니다. 와일드카드 문자는 \\ (Ex: "*") 표준을 통해 이스케이프됩니다. 매칭 중에는 "*" 이외의 어떠한 문자도 특별한 의미를 갖지 않습니다.

Choice 상태 예제

다음은 전환될 Choice 상태 및 다른 상태의 예입니다.

Note

[$.type] 필드를 지정해야 합니다. 상태 입력에 $.type 필드가 들어 있지 않으면 실행이 실패하고 실행 기록에 오류가 표시됩니다. 리터럴 값과 일치하는 문자열만 StringEquals 필드에 지정할 수 있습니다. 예: "StringEquals": "Buy".

```
"ChoiceStateX": {
  "Type": "Choice",
  "Choices": [
    {
      "Not": {
        "Variable": "$.type",
        "StringEquals": "Private"
      },
      "Next": "Public"
    },
    {
      "Variable": "$.value",
      "NumericEquals": 0,
      "Next": "ValueIsZero"
    },
    {
```

```
      "And": [
        {
          "Variable": "$.value",
          "NumericGreaterThanEquals": 20
        },
        {
          "Variable": "$.value",
          "NumericLessThan": 30
        }
      ],
      "Next": "ValueInTwenties"
    }
  ],
  "Default": "DefaultState"
},

"Public": {
  "Type" : "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Foo",
  "Next": "NextState"
},

"ValueIsZero": {
  "Type" : "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Zero",
  "Next": "NextState"
},

"ValueInTwenties": {
  "Type" : "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Bar",
  "Next": "NextState"
},

"DefaultState": {
  "Type": "Fail",
  "Cause": "No Matches!"
}
```

이 예제에서 상태 시스템은 다음 입력 값으로 시작됩니다.

```
{
  "type": "Private",
```

```
  "value": 22
}
```

Step Functions는 `value` 필드에 따라 `ValueInTwenties` 상태로 전환됩니다.

`Choice` 상태의 `Choices`에 일치하는 상태가 없으면 `Default` 필드에 제공된 상태가 대신 실행됩니다. `Default` 상태가 지정되어 있지 않으면 오류가 표시되며 실행이 실패합니다.

Wait

`Wait` 상태(`"Type": "Wait"`)는 상태 시스템이 지정된 시간 동안 계속되지 않도록 지연시킵니다. 상대적인 시간, 상태가 시작된 후 지정된 시간(초) 또는 타임스탬프로서 지정되는 절대적인 종료 시간에서 선택할 수 있습니다.

`Wait` 상태에서는 일반 상태 필드 외에도 다음 필드 하나가 있습니다.

`Seconds`

`Next` 필드에 지정된 상태를 시작하기 전에 기다려야 하는 시간(초). 시간을 0~99999999까지의 양의 정수 값으로 지정해야 합니다.

`Timestamp`

`Next` 필드에 지정된 상태를 시작할 때까지 기다려야 하는 절대 시간.

타임스탬프는 ISO 8601의 RFC3339 프로필을 준수해야 하며, 대문자 `T`는 날짜와 시간 부분을 구분해야 하고, 대문자 `Z`는 숫자 시간대 오프셋이 없음을 나타냅니다(예: `2024-08-18T17:33:00Z`).

> **Note**
>
> 현재 대기 시간을 타임스탬프로 지정하면 Step Functions는 시간 값을 초까지 고려하여 밀리초를 잘라냅니다.

`SecondsPath`

`Next` 필드에 지정된 상태를 시작하기 전에 기다려야 하는 시간(초)으로, 상태의 입력 데이터의 경로를 사용하여 지정됩니다.

이 필드에 정수 값을 지정해야 합니다.

`TimestampPath`

`Next` 필드에 지정된 상태를 시작할 때까지 기다려야 하는 절대 시간으로, 상태의 입력 데이터의 경로를 사용하여 지정됩니다.

> **Note**
>
> `Seconds`, `Timestamp`, `SecondsPath` 또는 `TimestampPath` 중 하나를 정확하게 지정해야 합니다. 또한 표준 워크플로와 Express 워크플로에 지정할 수 있는 최대 대기 시간은 각각 1년 및 5분입니다.

Wait 상태 예제

다음 `Wait` 상태는 상태 시스템을 10초간 지연시킵니다.

```
"wait_ten_seconds": {
  "Type": "Wait",
  "Seconds": 10,
  "Next": "NextState"
}
```

다음 예제에서 `Wait` 상태는 절대 시간이 2024년 3월 14일 오전 1시 59분(UTC 기준)이 될 때까지 기다립니다.

```
"wait_until" : {
  "Type": "Wait",
  "Timestamp": "2024-03-14T01:59:00Z",
  "Next": "NextState"
}
```

대기 시간을 반드시 하드 코딩할 필요는 없습니다. 예를 들어 다음 입력 데이터를 예로 들 수 있습니다.

```
{
  "expirydate": "2024-03-14T01:59:00Z"
}
```

입력 데이터로부터 값을 선택하려면 참조 경로를 사용하여 입력의 "expirydate" 값을 선택할 수 있습니다.

```
"wait_until" : {
    "Type": "Wait",
    "TimestampPath": "$.expirydate",
    "Next": "NextState"
}
```

Succeed

Succeed 상태("Type": "Succeed")는 실행을 성공적으로 중지합니다. Succeed 상태는 실행 중지 작업만 하는 Choice 상태 브랜치에 대한 유용한 대상입니다.

Succeed 상태는 터미널 상태이므로 다음 예제와 같이 Next 필드가 없으며 End 필드도 필요하지 않습니다.

```
"SuccessState": {
  "Type": "Succeed"
}
```

Fail

Fail 상태("Type": "Fail")는 Catch 블록에서 포착하지 않는 한 상태 시스템 실행을 중지하고 이를 실패로 표시합니다.

Fail 상태를 통해 공통 상태 필드 집합의 Type 및 Comment 필드만 사용할 수 있습니다. 또한 Fail 상태는 다음 필드를 허용합니다.

Cause(선택 사항)

오류 원인을 설명하는 사용자 지정 문자열입니다. 운영 또는 진단 목적으로 이 필드를 지정할 수 있습니다.

CausePath(선택 사항)

참조 경로를 사용하여 상태 입력에서 동적으로 오류 원인에 대한 자세한 설명을 제공하려면 CausePath를 사용하세요. 해결되면 참조 경로에서 문자열 값이 포함된 필드를 선택해야 합니다.

문자열을 반환하는 내장 함수를 사용하여 CausePath를 지정할 수도 있습니다. 이러한 내장 함수는 States.Format, States.JsonToString, States.ArrayGetItem, States.Base64Encode, States.Base64Decode, States.Hash 및 States.UUID입니다.

> **Important**
>
> - `Cause` 또는 `CausePath`를 지정할 수 있지만 Fail 상태 정의에서는 둘 다 지정할 수 없습니다.
> - 정보 보안 모범 사례로 민감한 정보나 내부 시스템 세부 정보 모두 원인 설명에서 제거하는 것이 좋습니다.

`Error`(선택 사항)

Retry 또는 Catch 필드를 사용하여 오류를 처리하도록 제공할 수 있는 오류 이름입니다. 운영 또는 진단 목적으로 오류 이름을 제공할 수도 있습니다.

`ErrorPath`(선택 사항)

참조 경로를 사용하여 상태 입력에서 동적으로 오류 이름을 제공하려면 `ErrorPath`를 사용하세요. 해결되면 참조 경로에서 문자열 값이 포함된 필드를 선택해야 합니다.

문자열을 반환하는 내장 함수를 사용하여 `ErrorPath`를 지정할 수도 있습니다. 이러한 내장 함수는 States.Format, States.JsonToString, States.ArrayGetItem, States.Base64Encode, States.Base64Decode, States.Hash 및 States.UUID입니다.

> **Important**
>
> - `Error` 또는 `ErrorPath`를 지정할 수 있지만 Fail 상태 정의에서는 둘 다 지정할 수 없습니다.
> - 정보 보안 모범 사례로 민감한 정보나 내부 시스템 세부 정보 모두 오류 이름에서 제거하는 것이 좋습니다.

`Fail` 상태는 항상 상태 시스템을 종료하므로 `Next` 필드가 없으며 `End` 필드도 필요하지 않습니다.

Fail 상태 정의 예제

다음 Fail 상태 정의 예제에서는 정적 `Error` 및 `Cause` 필드 값을 지정합니다.

```
"FailState": {
  "Type": "Fail",
  "Cause": "Invalid response.",
```

```
  "Error": "ErrorA"
}
```

다음 Fail 상태 정의 예제에서는 참조 경로를 동적으로 사용하여 `Error` 및 `Cause` 필드 값을 확인합니다.

```
"FailState": {
  "Type": "Fail",
  "CausePath": "$.Cause",
  "ErrorPath": "$.Error"
}
```

다음 Fail 상태 정의 예제에서는 States.Format 내장 함수를 사용하여 `Error` 및 `Cause` 필드 값을 동적으로 지정합니다.

```
"FailState": {
  "Type": "Fail",
  "CausePath": "States.Format('This is a custom error message for {}, caused by {}.',
 $.Error, $.Cause)",
  "ErrorPath": "States.Format('{}', $.Error)"
}
```

Parallel

`Parallel` 상태(`"Type": "Parallel"`)를 사용하여 별도의 실행 브랜치를 상태 시스템에 추가할 수 있습니다.

`Parallel` 상태에는 일반 상태 필드 외에도 다음 추가 필드가 포함됩니다.

`Branches`(필수)

병렬로 실행할 상태 시스템을 지정하는 객체 어레이. 이러한 각 상태 시스템 객체에는 `States` 및 `StartAt`이라는 필드가 들어 있습니다. 이러한 필드의 의미는 상태 시스템의 최상위에 있는 필드와 동일합니다.

`ResultPath` (선택 사항)

브랜치의 출력을 배치할 위치(입력에서)를 지정합니다. 그러면 `OutputPath` 필드(있는 경우)에 지정된 대로 입력이 필터링된 후 상태의 출력으로 사용됩니다. 자세한 내용은 입/출력 처리를 참조하십시오.

`ResultSelector` (선택 사항)

키 값 페어 컬렉션을 전달합니다. 여기서 값은 정적이거나 결과에서 선택됩니다. 자세한 설명은 ResultSelector 섹션을 참조하세요.

`Retry` (선택 사항)

상태에 런타임 오류가 발생하는 경우 사용될 재시도 정책을 정의하는 Retriers라는 객체 어레이. 자세한 설명은 Retry 및 Catch를 사용하는 상태 시스템 예제 섹션을 참조하세요.

`Catch` (선택 사항)

상태에 런타임 오류가 발생하고 해당 재시도 정책이 소진되었거나 정의되지 않은 경우 실행되는 폴백 상태를 정의하는 Catcher라는 객체 배열. 자세한 내용은 폴백 상태를 참조하십시오.

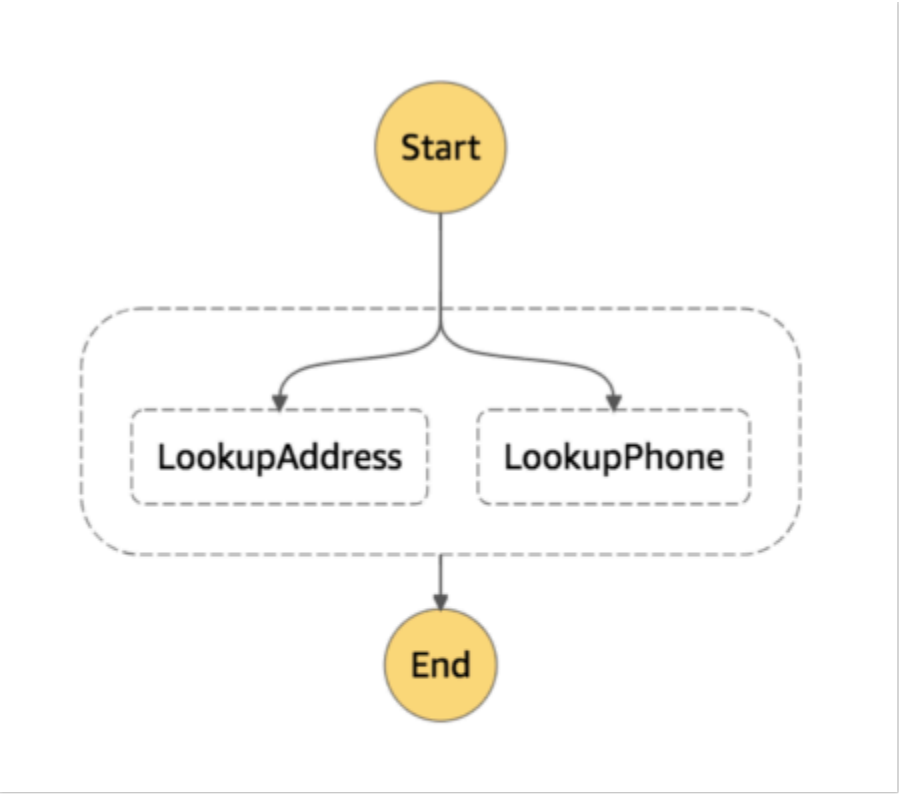
`Parallel`상태는 AWS Step Functions 해당 브랜치의 `StartAt` 필드에 이름이 지정된 상태에서 시작하여 각 브랜치를 최대한 동시에 실행하고 모든 브랜치가 종료 (최종 상태에 도달) 될 때까지 기다렸다가 `Parallel` 상태 `Next` 필드를 처리합니다.

Parallel 상태 예제

```
{
  "Comment": "Parallel Example.",
  "StartAt": "LookupCustomerInfo",
  "States": {
    "LookupCustomerInfo": {
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
         "StartAt": "LookupAddress",
         "States": {
           "LookupAddress": {
             "Type": "Task",
             "Resource":
               "arn:aws:lambda:us-east-1:123456789012:function:AddressFinder",
             "End": true
           }
         }
       },
       {
         "StartAt": "LookupPhone",
         "States": {
```

```
            "LookupPhone": {
              "Type": "Task",
              "Resource":
                "arn:aws:lambda:us-east-1:123456789012:function:PhoneFinder",
              "End": true
            }
          }
        }
      ]
    }
  }
}
```

이 예제에서는 LookupAddress 및 LookupPhone 브랜치가 병렬로 실행됩니다. 다음은 워크플로가 Step Functions 콘솔에서 어떻게 표시되는지를 보여줍니다.



각 브랜치는 독립적이어야 합니다. Parallel 상태의 한 브랜치에 있는 상태에는 해당 브랜치의 외부 필드를 대상으로 하는 Next 필드가 있을 수 없으며 해당 브랜치로의 브랜치 전환 외부의 다른 상태도 있을 수 없습니다.

Parallel 상태의 입/출력 처리

Parallel 상태는 각 브랜치에 자체 입력 데이터의 사본을 제공합니다(InputPath 필드의 수정 작업에 따라 달라질 수 있음). 이 상태는 해당 브랜치의 출력을 포함하는 각 브랜치의 요소 하나가 있는 배열인 출력을 생성합니다. 모든 요소의 형식이 동일해야 하지는 않습니다. 일반적인 방법으로 ResultPath 필드를 사용하여 출력 배열을 입력 데이터에 삽입할 수 있습니다(전체는 Parallel 상태 출력으로 전송됨, 입/출력 처리 참조).

```
{
  "Comment": "Parallel Example.",
  "StartAt": "FunWithMath",
  "States": {
    "FunWithMath": {
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
          "StartAt": "Add",
          "States": {
            "Add": {
              "Type": "Task",
              "Resource": "arn:aws:states:us-east-1:123456789012:activity:Add",
              "End": true
            }
          }
        },
        {
          "StartAt": "Subtract",
          "States": {
            "Subtract": {
              "Type": "Task",
              "Resource": "arn:aws:states:us-east-1:123456789012:activity:Subtract",
              "End": true
            }
          }
        }
      ]
    }
```

```
  }
}
```

FunWithMath 상태에서 어레이 [3, 2]를 입력으로 제공한 경우, Add 및 Subtract 상태는 모두 입력으로 해당 어레이를 받습니다. Add 및 Subtract 작업의 출력은 배열 요소 3과 2의 합계와 차이(즉, 5 및 1)인 반면 Parallel 상태 출력은 배열입니다.

```
[ 5, 1 ]
```

> **Tip**
>
> 상태 시스템에서 사용하는 Parallel 또는 Map 상태가 배열을 반환하는 경우 ResultSelector 필드를 사용하여 배열을 평면 배열로 변환할 수 있습니다. 자세한 설명은 배열의 배열 평면화 섹션을 참조하세요.

오류 처리

오류가 처리되지 않거나 Fail 상태로 전환되는 등의 이유로 브랜치에 오류가 발생하면 전체 Parallel 상태가 오류 발생으로 간주되고 모든 브랜치가 중지됩니다. Parallel 상태 자체에서 오류를 처리하지 않으면 Step Functions가 실행을 중지하고 오류를 표시합니다.

> **Note**
>
> 병렬 상태가 무너지면 간접적으로 호출된 Lambda 함수가 계속 실행되며 작업 토큰을 실행하는 활동 작업자는 중지되지 않습니다.
>
> - 장시간 실행되는 활동을 중지하려면 하트비트를 사용하여 Step Functions에서 브랜치를 중지했는지 확인하고 작업을 실행하고 있는 작업자를 중지하세요. 상태가 실패한 경우 SendTaskHeartbeat, SendTaskSuccess 또는 SendTaskFailure를 호출하면 오류가 발생합니다. 하트비트 오류를 참조하십시오.
> - 실행 중인 Lambda 함수를 중지할 수 없습니다. 대비책을 구현했으면 Lambda 함수가 완료된 후 정리 작업이 실행되도록 Wait 상태를 사용하세요.

맵

Map 상태를 사용하여 데이터 세트의 항목마다 일련의 워크플로 단계를 실행할 수 있습니다. Map 상태의 반복은 동시에 실행되므로 데이터 세트를 빠르게 처리할 수 있습니다. Map 상태는 JSON 배열, Amazon S3 객체 목록 또는 CSV 파일 등 다양한 입력 유형을 사용할 수 있습니다.

Step Functions는 워크플로에서 Map 상태를 사용할 수 있도록 인라인 모드와 분산 모드 등 두 가지 유형의 처리 모드를 제공합니다.

이러한 모드에 대한 내용과 각 모드에서 Map 상태를 사용하는 방법은 다음 주제를 참조하세요.

- Map 상태 처리 모드
  - 인라인 모드에서 Map 상태 사용
  - 분산 모드에서 Map 상태 사용

> **Tip**
>
> Map 상태를 사용하는 워크플로 예제를 AWS 계정에 배포하려면 AWS Step Functions 워크숍의 모듈 5 - 선택 상태 및 Map 상태를 참조하세요.

Map 상태 처리 모드

Step Functions는 데이터 세트의 항목을 처리하려는 방식에 따라 Map 상태에 대한 다음과 같은 처리 모드를 제공합니다.

- 인라인 — 제한적 동시성 모드입니다. 이 모드에서는 각 Map 상태 반복이 Map 상태가 포함된 워크플로의 컨텍스트에서 실행됩니다. Step Functions는 이러한 반복 실행 내역을 상위 워크플로 실행 내역에 추가합니다. 기본적으로 Map 상태는 인라인 모드에서 실행됩니다.

  이 모드에서 Map 상태는 입력으로 JSON 배열만 수락합니다. 또한 이 모드는 동시 반복을 최대 40회까지 지원합니다.

  자세한 내용은 인라인 모드에서 Map 상태 사용 섹션을 참조하세요.
- 분산 — 동시성이 높은 모드입니다. 이 모드에서 Map 상태는 각 반복을 하위 워크플로 실행으로 실행하므로 병렬 하위 워크플로를 동시에 최대 10,000개까지 실행할 수 있습니다. 각 하위 워크플로 실행에는 상위 워크플로와 별개인 자체 실행 내역이 있습니다.

이 모드에서 Map 상태는 입력으로 JSON 배열이나 CSV 파일과 같은 Amazon S3 데이터 소스를 허용할 수 있습니다.

자세한 내용은 분산 모드에서 Map 상태 사용 섹션을 참조하세요.

사용해야 하는 모드는 데이터 세트의 항목을 처리하려는 방식에 따라 달라집니다. 워크플로 실행 내역 항목이 25,000개를 초과하지 않거나 동시 반복이 40회 넘게 필요하지 않은 경우에 인라인 모드에서 Map 상태를 사용합니다.

다음 조건 조합을 모두 충족하는 대규모 병렬 워크로드를 오케스트레이션해야 하는 경우 분산 모드에서 Map 상태를 사용합니다.

- 데이터 세트 크기가 256KB를 초과합니다.
- 워크플로의 실행 이벤트 내역 항목이 25,000개를 초과합니다.
- 병렬 반복을 40회 넘게 동시에 실행해야 합니다.

주제

인라인 모드와 분산 모드의 차이점

다음 표에는 인라인 모드와 분산 모드 간의 차이점이 강조 표시되어 있습니다.

인라인 모드	분산 모드
Supported data sources	
워크플로의 이전 단계에서 전달된 JSON 배열을 입력으로 허용합니다.	다음 데이터 소스를 입력으로 허용합니다. • 워크플로의 이전 단계에서 전달된 JSON 배열 • 배열이 포함된 Amazon S3 버킷에 있는 JSON 파일 • Amazon S3 버킷에 있는 CSV 파일

인라인 모드	분산 모드
	• Amazon S3 객체 목록 • Amazon S3 인벤토리
Map iterations	
이 모드에서는 각 Map 상태 반복이 Map 상태가 포함된 워크플로의 컨텍스트에서 실행됩니다. Step Functions는 이러한 반복 실행 내역을 상위 워크플로 실행 내역에 추가합니다.	이 모드에서 Map 상태는 각 반복을 하위 워크플로 실행으로 실행하므로 병렬 하위 워크플로를 동시에 최대 10,000개까지 실행할 수 있습니다. 각 하위 워크플로 실행에는 상위 워크플로와 별개인 자체 실행 내역이 있습니다.
Maximum concurrency for parallel iterations	
가능한 한 동시에 반복을 최대 40회까지 실행할 수 있습니다.	하위 워크플로 실행을 동시에 최대 10,000개까지 실행하여 한 번에 수백만 개의 데이터 항목을 처리할 수 있습니다.
Input payload and event history sizes	
입력 페이로드 크기를 256KB로 제한하고 실행 이벤트 내역 항목을 25,000개로 제한합니다.	Map 상태는 Amazon S3 데이터 소스에서 직접 입력을 읽을 수 있으므로 페이로드 크기 제한을 해결할 수 있습니다. 이 모드에서는 Map 상태에서 시작한 하위 워크플로 실행이 상위 워크플로 실행 내역과 별도인 자체 실행 내역을 유지하므로 실행 내역 제한을 해결할 수도 있습니다.
Monitoring and observability	

인라인 모드	분산 모드
콘솔에서 또는 `GetExecutionHistory` API 작업을 간접적으로 호출하여 워크플로 실행 내역을 검토할 수 있습니다. 또한 CloudWatch 및 X-Ray를 통해 실행 내역을 볼 수 있습니다.	분산 모드에서 `Map` 상태를 실행하면 Step Functions에서 맵 실행 리소스를 만듭니다. 맵 실행은 Distributed Map 상태가 시작하는 일련의 하위 워크플로 실행을 의미합니다. Step Functions 콘솔에서 맵 실행을 볼 수 있습니다. `DescribeMapRun` API 작업을 간접적으로 호출할 수도 있습니다. 또한 맵 실행에서 CloudWatch에 지표를 내보냅니다. 자세한 내용은 Distributed Map 상태 실행의 맵 실행 검사 섹션을 참조하세요.

인라인 모드에서 Map 상태 사용

기본적으로 `Map` 상태는 인라인 모드에서 실행됩니다. 인라인 모드에서 Map 상태는 입력으로 JSON 배열만 수락합니다. 워크플로의 이전 단계에서 이 배열을 수신합니다. 이 모드에서는 각 `Map` 상태 반복이 `Map` 상태가 포함된 워크플로의 컨텍스트에서 실행됩니다. Step Functions는 이러한 반복 실행 내역을 상위 워크플로 실행 내역에 추가합니다.

이 모드에서 `Map` 상태는 동시 반복을 최대 40개까지 지원합니다.

인라인으로 설정된 `Map` 상태를 Inline Map 상태라고 합니다. 워크플로 실행 내역 항목이 25,000개를 초과하지 않거나 동시 반복이 40회 넘게 필요하지 않은 경우에 인라인 모드에서 `Map` 상태를 사용합니다.

Inline Map 상태 사용에 대한 소개는 자습서 Inline Map 상태를 사용하여 작업 반복을 참조하세요.

목차

이 주제의 주요 개념

인라인 모드

Map 상태의 제한된 동시성 모드입니다. 이 모드에서는 각 Map 상태 반복이 Map 상태가 포함된 워크플로의 컨텍스트에서 실행됩니다. Step Functions는 이러한 반복의 실행 내역을 상위 워크플로 실행 내역에 추가합니다. Map 상태는 기본적으로 인라인 모드에서 실행됩니다.

이 모드는 입력으로 JSON 배열만 수락하고 동시 반복을 최대 40회까지 지원합니다.

Inline Map 상태

인라인 모드로 설정된 Map 상태입니다.

맵 워크플로

Map 상태가 반복마다 실행되는 단계 세트입니다.

Map 상태 반복

Map 상태 내에서 정의된 워크플로 반복입니다.

Inline Map 상태 필드

워크플로에서 Inline Map 상태를 사용하려면 다음 필드를 하나 이상을 지정합니다. 일반 상태 필드 외에 다음 필드를 지정합니다.

`Type`(필수)

상태 유형을 설정합니다(예: Map).

`ItemProcessor`(필수)

Map 상태 처리 모드와 정의를 지정하는 다음 JSON 객체를 포함합니다.

정의에는 각 배열 항목을 처리하기 위해 반복하는 단계 세트가 포함되어 있습니다.

- `ProcessorConfig` - Map 상태의 처리 모드를 지정하는 선택적 JSON 객체입니다. 이 객체에는 `Mode` 하위 필드가 포함되어 있습니다. 이 필드의 기본값은 인라인 모드의 Map 상태를 사용하는 `INLINE`입니다.

  이 모드에서 반복이 실패하면 Map 상태가 실패합니다. Map 상태가 실패하면 모든 반복이 중지합니다.

- `StartAt` - 워크플로의 첫 번째 상태를 나타내는 문자열을 지정합니다. 이 문자열은 대소문자를 구분하며 상태 객체 중 하나의 이름과 일치해야 합니다. 이 상태는 데이터 세트의 항목마다 가장 먼저 실행됩니다. `Map` 상태에 제공하는 모든 실행 입력은 먼저 `StartAt` 상태에 전달됩니다.
- `States` - 쉼표로 구분된 상태 집합이 포함된 JSON 객체입니다. 이 객체에서 Map workflow를 정의합니다.

> **Note**
>
> - `ItemProcessor` 필드 내의 상태는 서로 전환될 수만 있습니다. `ItemProcessor` 필드 외부의 상태는 필드 내의 상태로 전환될 수 없습니다.
> - `ItemProcessor` 필드는 현재 지원 중단된 `Iterator` 필드를 대체합니다. `Iterator` 필드를 사용하는 `Map` 상태를 계속 포함할 수 있지만 이 필드를 `ItemProcessor`로 바꾸는 것이 좋습니다.
>
>   Step Functions Local은 현재 이 `ItemProcessor` 필드를 지원하지 않습니다. Step Functions Local에서 `Iterator` 필드를 사용하는 것이 좋습니다.

`ItemsPath`(선택 사항)

JsonPath 구문을 사용하여 참조 경로를 지정합니다. 이 경로에서 상태 입력 내의 항목 배열을 포함하는 JSON 노드를 선택합니다. 자세한 내용은 ItemsPath 섹션을 참조하세요.

`ItemSelector`(선택 사항)

입력 배열 항목 값을 각 `Map` 상태 반복으로 전달하기 전에 재정의합니다.

이 필드에서 키-값 페어 컬렉션이 포함된 유효한 JSON을 지정합니다. 이러한 페어에는 다음 중 하나가 포함할 수 있습니다.

- 상태 시스템 정의에서 정의하는 정적 값
- 경로를 사용하여 상태 입력에서 선택한 값
- 컨텍스트 객체에서 액세스한 값

자세한 내용은 ItemSelector 섹션을 참조하세요.

`ItemSelector` 필드는 현재 지원 중단된 `Parameters` 필드를 대체합니다. `Parameters` 필드를 사용하는 `Map` 상태를 계속 포함할 수 있지만 이 필드를 `ItemSelector`로 바꾸는 것이 좋습니다.

`MaxConcurrency`(선택 사항)

동시에 실행할 수 있는 `Map` 상태 반복 횟수의 상한을 제공하는 정수 값을 지정합니다. 예를 들어 `MaxConcurrency` 값이 10이면 한 번에 실행되는 `Map` 상태 동시 반복 횟수는 10회로 제한됩니다.

> **Note**
>
> 동시 반복은 제한될 수 있습니다. 이 경우 일부 반복은 이전 반복이 완료될 때까지 시작되지 않습니다. 입력 배열에 항목이 40개 넘게 있으면 이러한 경우가 발생할 가능성이 높아집니다.
> 동시성을 높이려면 분산 모드에서 Map 상태 사용을 고려하세요.

기본값은 `0`이며 이 값은 동시성을 제한하지 않습니다. Step Functions는 반복을 가능한 한 동시에 간접적으로 호출합니다.

`MaxConcurrency` 값이 `1`이면 `ItemProcessor` 배열 요소마다 한 번 간접적으로 호출됩니다. 배열의 항목은 입력에 나타나는 순서대로 처리됩니다. Step Functions는 이전 반복이 완료될 때까지 새 반복을 시작하지 않습니다.

`MaxConcurrencyPath`(선택 사항)

참조 경로를 사용하여 상태 입력에서 동적으로 최대 동시성 값을 제공하려면 `MaxConcurrencyPath`를 사용합니다. 확인되면 참조 경로에서 값이 음수가 아닌 정수인 필드를 선택해야 합니다.

> **Note**
>
> `Map` 상태에는 `MaxConcurrency` 및 `MaxConcurrencyPath` 모두 포함될 수 없습니다.

`ResultPath`(선택 사항)

`Map` 상태 반복 출력을 저장할 위치를 입력에서 지정합니다. 그러면 Map 상태에서 `OutputPath` 필드(지정된 경우)에 지정된 대로 입력을 필터링합니다. 그런 다음 필터링된 입력을 상태 출력으로 사용합니다. 자세한 내용은 입/출력 처리를 참조하십시오.

`ResultSelector`(선택 사항)

키 값 페어 컬렉션을 전달합니다. 여기서 값은 정적이거나 결과에서 선택된 값입니다. 자세한 내용은 ResultSelector 섹션을 참조하세요.

> **Tip**
>
> 상태 시스템에서 사용하는 Parallel 또는 Map 상태가 배열을 반환하는 경우 ResultSelector 필드를 사용하여 배열을 평면 배열로 변환할 수 있습니다. 자세한 내용은 배열의 배열 평면화 섹션을 참조하세요.

`Retry`(선택 사항)

Retrier라고 하는 객체 배열로, 재시도 정책을 정의합니다. 런타임 오류가 발생하면 상태에서 재시도 정책을 사용합니다. 자세한 내용은 Retry 및 Catch를 사용하는 상태 시스템 예제 섹션을 참조하세요.

> **Note**
>
> Inline Map 상태에 Retriers를 정의하면 재시도 정책이 실패한 반복이 아닌 모든 `Map` 상태 반복에 적용됩니다. 예를 들어 `Map` 상태에 2회 성공한 반복과 한 번 실패한 반복이 있다고 가정해보겠습니다. `Map` 상태의 `Retry` 필드를 정의한 경우 재시도 정책이 실패한 반복에만 아닌 `Map` 상태 반복 3회 모두에 적용됩니다.

`Catch`(선택 사항)

폴백(fallback) 상태를 정의하는 객체 배열(Catcher). 런타임 오류가 발생하고 재시도 정책이 없거나 삭제되면 상태에서 Catcher를 실행합니다. 자세한 내용은 폴백 상태를 참조하십시오.

사용되지 않는 필드

> **Note**
>
> 다음 필드를 사용하는 `Map` 상태를 계속 포함할 수 있지만 `Iterator`를 `ItemProcessor` 및 `ItemSelector`가 있는 `Parameters`로 바꾸는 것이 좋습니다.

`Iterator`

각 배열 요소를 처리하는 일련의 단계를 정의하는 JSON 객체를 지정합니다.

Parameters

키-값 페어 컬렉션을 지정합니다. 여기서 값에는 다음 중 하나가 포함될 수 있습니다.

- 상태 시스템 정의에서 정의하는 정적 값
- 경로를 사용하여 입력에서 선택한 값

Inline Map 상태 예제

인라인 모드에서 실행되는 Map 상태에 대한 다음 입력 데이터가 있다고 가정합니다.

```
{
  "ship-date": "2016-03-14T01:59:00Z",
  "detail": {
    "delivery-partner": "UQS",
    "shipped": [
      { "prod": "R31", "dest-code": 9511, "quantity": 1344 },
      { "prod": "S39", "dest-code": 9511, "quantity": 40 },
      { "prod": "R31", "dest-code": 9833, "quantity": 12 },
      { "prod": "R40", "dest-code": 9860, "quantity": 887 },
      { "prod": "R40", "dest-code": 9511, "quantity": 1220 }
    ]
  }
}
```

이전 입력에서 지정된 대로 다음 예제의 Map 상태는 shipped 필드 내 배열 항목마다 ship-val이라는 AWS Lambda 함수를 한 번 간접적으로 호출합니다.

```
"Validate All": {
    "Type": "Map",
    "InputPath": "$.detail",
    "ItemProcessor": {
        "ProcessorConfig": {
            "Mode": "INLINE"
        },
        "StartAt": "Validate",
        "States": {
            "Validate": {
                "Type": "Task",
                "Resource": "arn:aws:states:::lambda:invoke",
                "OutputPath": "$.Payload",
                "Parameters": {
```

```
                    "FunctionName": "arn:aws:lambda:us-
east-2:123456789012:function:ship-val:$LATEST"
                },
                "End": true
            }
        }
    },
    "End": true,
    "ResultPath": "$.detail.shipped",
    "ItemsPath": "$.shipped"
}
```

Map 상태를 반복할 때마다 ItemsPath 필드로 선택한 배열의 항목이 입력으로 ship-val Lambda 함수에 전송됩니다. 다음 값은 Map 상태에서 Lambda 함수 간접 호출에 보내는 입력의 예제입니다.

```
{
  "prod": "R31",
  "dest-code": 9511,
  "quantity": 1344
}
```

완료되고 나면 Map 상태 출력은 JSON 배열이 되며, 여기서 각 항목은 반복에 대한 출력입니다. 이 경우 이 배열에는 ship-val Lambda 함수의 출력이 포함됩니다.

ItemSelector가 있는 Inline Map 상태 예제

앞의 예제의 ship-val Lambda 함수에 전송 주체에 대한 정보가 필요하다고 가정해보겠습니다. 이 정보는 각 반복의 배열에 있는 항목에 추가됩니다. 입력의 정보를 Map 상태의 현재 반복에 지정된 정보와 함께 포함할 수 있습니다. 다음 예제에서 ItemSelector 필드를 확인하세요.

```
"Validate-All": {
  "Type": "Map",
  "InputPath": "$.detail",
  "ItemsPath": "$.shipped",
  "MaxConcurrency": 0,
  "ResultPath": "$.detail.shipped",
  "ItemSelector": {
    "parcel.$": "$$.Map.Item.Value",
    "courier.$": "$.delivery-partner"
  },
  "ItemProcessor": {
    "StartAt": "Validate",
```

```
    "States": {
      "Validate": {
        "Type": "Task",
 "Resource": "arn:aws:lambda:us-east-1:123456789012:function:ship-val",
        "End": true
      }
    }
  },
  "End": true
}
```

ItemSelector 블록은 반복에 대한 입력을 JSON 노드로 대체합니다. 이 노드에는 컨텍스트 개체의 현재 항목 데이터와 Map 상태입력 delivery-partner 필드의 주체 정보가 모두 포함됩니다. 다음은 단일 반복에 대한 입력의 예제입니다. Map 상태는 이 입력을 ship-val Lambda 함수 간접 호출에 전달합니다.

```
{
  "parcel": {
    "prod": "R31",
    "dest-code": 9511,
    "quantity": 1344
   },
   "courier": "UQS"
}
```

이전 Inline Map 상태 예제에서 ResultPath 필드는 출력을 입력과 같은 형식으로 생성합니다. 하지만 각 요소가 각 반복의 ship-val Lambda 간접 호출의 출력인 배열로 detail.shipped 필드를 덮어씁니다.

Inline Map 상태 및 해당 필드 사용 방법에 대한 자세한 내용은 다음을 참조하세요.

- Inline Map 상태를 사용하여 작업 반복
- Step Functions에서 입력 및 출력 처리
- ItemsPath
- Map 상태의 컨텍스트 객체 데이터

Inline **Map** 상태 입력 및 출력 처리

지정된 Map 상태의 경우 InputPath에서 상태 입력의 하위 집합을 선택합니다.

Map 상태 입력에는 JSON 배열이 포함되어야 합니다. Map 상태는 배열의 항목마다 ItemProcessor 섹션을 한 번 실행합니다. ItemsPath 필드를 지정하면 Map 상태는 입력 위치를 선택하여 반복할 배열을 찾습니다. 지정하지 않으면 ItemsPath 값은 $가 되고, ItemProcessor 섹션에서는 배열이 유일한 입력일 것으로 예상합니다. ItemsPath 필드를 지정하는 경우 해당 값은 참조 경로여야 합니다. Map 상태는 InputPath를 적용한 후 이 경로를 유효 입력에 적용합니다. ItemsPath는 값이 JSON 배열인 필드를 식별해야 합니다.

각 반복에 대한 입력은 기본적으로 ItemsPath 값으로 식별되는 배열 필드의 단일 요소입니다. ItemSelector 필드를 사용하여 이 값을 재정의할 수 있습니다.

완료되고 나면 Map 상태 출력은 JSON 배열이 되며, 여기서 각 항목은 반복에 대한 출력입니다.

Inline Map 상태 입력 및 출력에 대한 자세한 내용은 다음을 참조하세요.

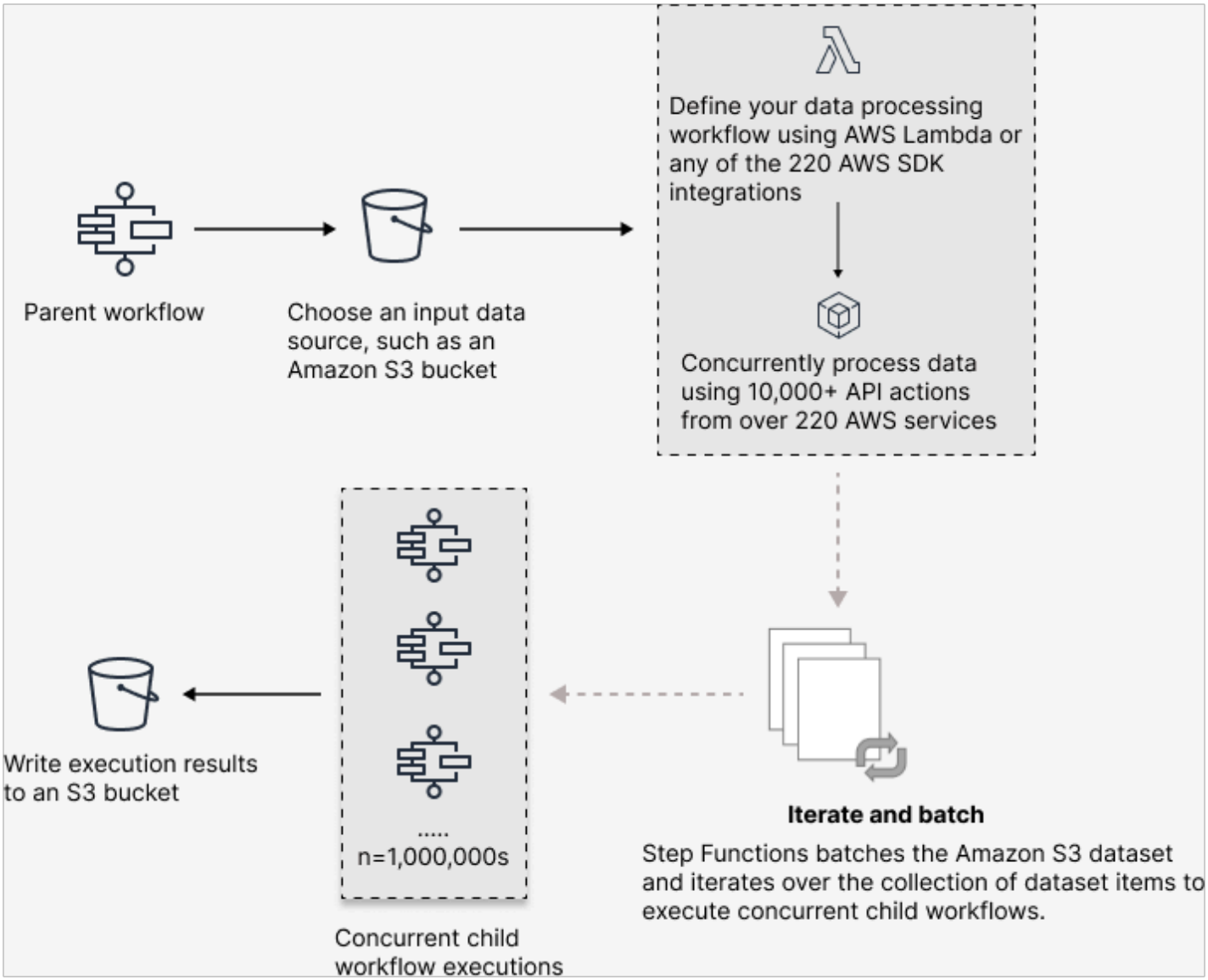
- Inline Map 상태를 사용하여 작업 반복
- ItemSelector가 있는 Inline Map 상태 예제
- Step Functions에서 입력 및 출력 처리
- Map 상태의 컨텍스트 객체 데이터
- Map 상태를 사용하여 데이터 동적 처리

분산 모드에서 Map 상태를 사용하여 대규모 병렬 워크로드 오케스트레이션

Step Functions를 사용하면 대규모 병렬 워크로드를 오케스트레이션하여 반정형 데이터의 온디맨드 처리와 같은 작업을 수행할 수 있습니다. 이러한 병렬 워크로드를 사용하면 Amazon S3에 저장된 대규모 데이터 소스를 동시에 처리할 수 있습니다. 예를 들어 대량의 데이터가 포함된 단일 JSON 또는 CSV 파일을 처리할 수 있습니다. 또는 대규모 Amazon S3 객체 집합을 처리할 수 있습니다.

워크플로에 대규모 병렬 워크로드를 설정하려면 분산 모드에 Map 상태를 포함합니다. Map 상태는 데이터 세트의 항목을 동시에 처리합니다. Distributed로 설정된 Map 상태를 Distributed Map 상태라고 합니다. 분산 모드에서 Map 상태를 사용하면 동시 처리가 가능해집니다. 분산 모드에서 Map 상태는 데이터 세트의 항목을 하위 워크플로 실행이라고 하는 반복으로 처리합니다. 병렬로 실행할 수 있는 하위 워크플로 실행 수를 지정할 수 있습니다. 각 하위 워크플로 실행에는 상위 워크플로와 별개인 자체 실행 내역이 있습니다. 지정하지 않는 경우 Step Functions는 병렬 하위 워크플로 실행 10,000개를 동시에 실행합니다.

다음 그림에서는 워크플로에서 대규모 병렬 워크로드를 설정하는 방법을 설명합니다.



이 주제의 내용

주요 용어

분산 모드

Map 상태 처리 모드입니다. 이 모드에서는 각 Map 상태 반복은 하위 워크플로 실행으로 실행되므로 동시성이 높습니다. 각 하위 워크플로 실행에는 상위 워크플로의 실행 내역과 별개인 자체 실행 내역이 있습니다. 이 모드에서 대규모 Amazon S3 데이터 소스의 입력을 읽을 수 있습니다.

Distributed Map 상태

분산 처리 모드로 설정된 Map 상태입니다.

맵 워크플로

Map 상태가 실행하는 일련의 단계입니다.

상위 워크플로

Distributed Map 상태가 하나 이상 포함된 워크플로입니다.

하위 워크플로 실행

Distributed Map 상태 반복입니다. 하위 워크플로 실행에는 상위 워크플로의 실행 내역과 별개인 자체 실행 내역이 있습니다.

맵 실행

분산 모드에서 Map 상태를 실행하면 Step Functions에서 맵 실행 리소스를 만듭니다. 맵 실행은 Distributed Map 상태가 시작하는 일련의 하위 워크플로 실행과 이러한 실행을 제어하는 런타임 설정을 의미합니다. Step Functions에서 Amazon 리소스 이름(ARN)을 맵 실행에 할당합니다. Step Functions 콘솔에서 맵 실행을 검사할 수 있습니다. `DescribeMapRun` API 작업을 간접적으로 호출할 수도 있습니다. Map Run은 메트릭도 CloudWatch 내보냅니다.

자세한 정보는 맵 실행 검사을 참조하세요.

Distributed Map 상태 정의 예제

다음 조건 조합을 모두 충족하는 대규모 병렬 워크로드를 오케스트레이션해야 하는 경우 분산 모드에서 Map 상태를 사용합니다.

- 데이터 세트 크기가 256KB를 초과합니다.
- 워크플로의 실행 이벤트 내역 항목이 25,000개를 초과합니다.
- 병렬 반복을 40회 넘게 동시에 실행해야 합니다.

다음 Distributed Map 상태 정의 예제에서는 데이터 세트를 Amazon S3 버킷에 저장된 CSV 파일로 지정합니다. 또한 CSV 파일의 각 행에서 데이터를 처리하는 Lambda 함수를 지정합니다. 이 예제에서는 CSV 파일을 사용하므로 CSV 열 헤더 위치도 지정합니다. 이 예제의 전체 상태 시스템 정의를 보려면 Distributed Map을 사용하여 대규모 CSV 데이터 복사 자습서를 참조하세요.

```
{
  "Map": {
    "Type": "Map",
    "ItemReader": {
      "ReaderConfig": {
        "InputType": "CSV",
        "CSVHeaderLocation": "FIRST_ROW"
      },
      "Resource": "arn:aws:states:::s3:getObject",
      "Parameters": {
        "Bucket": "Database",
        "Key": "csv-dataset/ratings.csv"
      }
    },
    "ItemProcessor": {
      "ProcessorConfig": {
        "Mode": "DISTRIBUTED",
        "ExecutionType": "EXPRESS"
      },
      "StartAt": "LambdaTask",
      "States": {
        "LambdaTask": {
          "Type": "Task",
          "Resource": "arn:aws:states:::lambda:invoke",
          "OutputPath": "$.Payload",
          "Parameters": {
            "Payload.$": "$",
            "FunctionName": "arn:aws:lambda:us-
east-2:123456789012:function:processCSVData"
          },
          "End": true
        }
      }
    },
    "Label": "Map",
    "End": true,
    "ResultWriter": {
      "Resource": "arn:aws:states:::s3:putObject",
```

```
      "Parameters": {
        "Bucket": "myOutputBucket",
        "Prefix": "csvProcessJobs"
      }
    }
  }
}
```

Distributed Map을 실행할 수 있는 권한

워크플로에 Distributed Map 상태가 포함된 경우 Step Functions에는 상태 시스템 역할에서 Distributed Map 상태의 StartExecution API 작업을 호출할 수 있는 적절한 권한이 있어야 합니다.

다음 IAM 정책 예제에서는 Distributed Map 상태를 실행하는 데 필요한 최소 권한을 상태 시스템 역할에 부여합니다.

> **Note**
>
> *stateMachineName*을 Distributed Map 상태를 사용하는 상태 시스템의 이름으로 바꿔야 합니다. 예를 들어 arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine*입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "arn:aws:states:region:accountID:stateMachine:stateMachineName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:StopExecution"
      ],
```

```
      "Resource": "arn:aws:states:region:accountID:execution:stateMachineName:*"
    }
  ]
}
```

또한 Amazon S3 버킷과 같이 분산 맵 상태에서 사용되는 AWS 리소스에 액세스하는 데 필요한 최소한의 권한이 있는지 확인해야 합니다. 자세한 내용은 Distributed Map 상태를 사용하기 위한 IAM 정책을 참조하세요.

Distributed Map 상태 필드

워크플로에서 Distributed Map 상태를 사용하려면 다음 필드를 하나 이상을 지정합니다. 일반 상태 필드 외에 다음 필드를 지정합니다.

`Type`(필수)

상태 유형을 설정합니다(예: `Map`).

`ItemProcessor`(필수)

`Map` 상태 처리 모드와 정의를 지정하는 다음 JSON 객체를 포함합니다.

- `ProcessorConfig` - `Map` 상태 구성을 지정하는 JSON 객체입니다. 이 객체에는 다음 하위 필드가 포함됩니다.
  - `Mode` - 분산 모드에서 `Map` 상태를 사용하려면 **`DISTRIBUTED`**로 설정합니다.

    > **Note**
    >
    > 현재 Express 워크플로 내에서 `Map` 상태를 사용하는 경우 `Mode`를 `DISTRIBUTED`로 설정할 수 없습니다. 하지만 표준 워크플로 내에서 `Map` 상태를 사용하는 경우 `Mode`를 `DISTRIBUTED`로 설정할 수 있습니다.

  - `ExecutionType` - 맵 워크플로 실행 유형을 STANDARD 또는 EXPRESS로 지정합니다. `Mode` 하위 필드에 `DISTRIBUTED`를 지정한 경우 이 필드를 제공해야 합니다. 워크플로 유형에 대한 자세한 내용은 표준 워크플로와 Express 워크플로 비교 섹션을 참조하세요.
- `StartAt` - 워크플로의 첫 번째 상태를 나타내는 문자열을 지정합니다. 이 문자열은 대소문자를 구분하며 상태 객체 중 하나의 이름과 일치해야 합니다. 이 상태는 데이터 세트의 항목마다 가장 먼저 실행됩니다. `Map` 상태에 제공하는 모든 실행 입력은 먼저 `StartAt` 상태에 전달됩니다.
- `States` - 쉼표로 구분된 상태 집합이 포함된 JSON 객체입니다. 이 객체에서 Map workflow를 정의합니다.

`ItemReader`

데이터 세트와 해당 위치를 지정합니다. `Map` 상태는 지정된 데이터 세트에서 입력 데이터를 수신합니다.

분산 모드에서는 이전 상태에서 전달된 JSON 페이로드나 대규모 Amazon S3 데이터 소스를 데이터 세트로 사용할 수 있습니다. 자세한 정보는 ItemReader을 참조하세요.

`ItemsPath` (선택 사항)

JsonPath구문을 사용하여 상태 입력 내에 항목 배열을 포함하는 JSON 노드를 선택하는 참조 경로를 지정합니다.

분산 모드에서는 이전 단계의 JSON 배열을 상태 입력으로 사용하는 경우에만 이 필드를 지정합니다. 자세한 정보는 ItemsPath을 참조하세요.

`ItemSelector` (선택 사항)

개별 데이터 세트 항목 값을 각 `Map` 상태 반복에 전달하기 전에 재정의합니다.

이 필드에서 키-값 페어 컬렉션이 포함된 유효한 JSON 입력을 지정합니다. 이러한 페어는 상태 시스템 정의에서 정의한 정적 값, 경로를 사용하여 상태 입력에서 선택한 값 또는 컨텍스트 객체에서 액세스한 값 중 하나일 수 있습니다. 자세한 정보는 ItemSelector을 참조하세요.

`ItemBatcher` (선택 사항)

데이터 세트 항목을 일괄 처리하려면 지정합니다. 그러면 각 하위 워크플로 실행에서 이러한 항목 배치를 입력으로 수신합니다. 자세한 정보는 ItemBatcher을 참조하세요.

`MaxConcurrency` (선택 사항)

동시에 실행할 수 있는 하위 워크플로 실행 수를 지정합니다. 인터프리터는 병렬 하위 워크플로 실행을 지정된 수까지만 허용합니다. 동시성 값을 지정하지 않거나 0으로 설정하면 Step Functions는 동시성을 제한하지 않고 병렬 하위 워크플로 실행 10,000개를 실행합니다.

> **Note**
>
> Parallel 하위 워크플로 실행에 대해 더 높은 동시성 제한을 지정할 수 있지만 다운스트림 AWS 서비스의 용량 (예:) 을 초과하지 않는 것이 좋습니다. AWS Lambda

`MaxConcurrencyPath` (선택 사항)

참조 경로를 사용하여 상태 입력에서 동적으로 최대 동시성 값을 제공하려면 `MaxConcurrencyPath`를 사용합니다. 확인되면 참조 경로에서 값이 음수가 아닌 정수인 필드를 선택해야 합니다.

> **Note**
>
> Map 상태에는 `MaxConcurrency` 및 `MaxConcurrencyPath` 모두 포함될 수 없습니다.

`ToleratedFailurePercentage` (선택 사항)

맵 실행에서 허용할 수 있는 실패 항목 비율을 정의합니다. 이 비율을 초과하면 맵 실행이 자동으로 실패합니다. Step Functions는 실패하거나 시간 초과된 총 항목 수를 총 항목 수로 나눈 결과로 실패 항목 백분율을 계산합니다. 0~100 범위의 값을 지정해야 합니다. 자세한 정보는 Distributed Map 상태의 허용 실패 임계값을 참조하세요.

`ToleratedFailurePercentagePath` (선택 사항)

참조 경로를 사용하여 상태 입력에서 허용 실패 백분율 값을 동적으로 제공하려면 `ToleratedFailurePercentagePath`를 사용합니다. 확인되면 참조 경로에서 값이 0~100인 필드를 선택해야 합니다.

`ToleratedFailureCount` (선택 사항)

맵 실행에서 허용할 수 있는 실패 항목 수를 정의합니다. 이 수를 초과하면 맵 실행이 자동으로 실패합니다. 자세한 정보는 Distributed Map 상태의 허용 실패 임계값을 참조하세요.

`ToleratedFailureCountPath` (선택 사항)

참조 경로를 사용하여 상태 입력에서 허용 실패 계수 값을 동적으로 제공하려면 `ToleratedFailureCountPath`를 사용합니다. 확인되면 참조 경로에서 값이 음수가 아닌 정수인 필드를 선택해야 합니다.

`Label` (선택 사항)

Map 상태를 고유하게 식별하는 문자열입니다. 맵 실행마다 Step Functions는 레이블을 맵 실행 ARN에 추가합니다. 다음은 `demoLabel`이라는 사용자 지정 레이블이 있는 맵 실행 ARN의 예제입니다.

```
arn:aws:states:us-east-1:123456789012:mapRun:demoWorkflow/
demoLabel:3c39a231-69bb-3d89-8607-9e124eddbb0b
```

레이블을 지정하지 않으면 Step Functions에서 자동으로 고유한 레이블을 생성합니다.

> **Note**
>
> 레이블 길이는 40자를 초과할 수 없고 상태 시스템 정의 내에서 고유해야 하며 다음과 같은 문자를 포함할 수 없습니다.
>
> - 공백 문자
> - 와일드카드 문자 (`? *`)
> - 괄호 문자(`< > { } [ ]`)
> - 특수 문자 (`: ; , \ | ^ ~ $ # % & ` "`)
> - 제어 문자(`\\u0000` - `\\u001f` 또는 `\\u007f` - `\\u009f`).
>
> Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

`ResultWriter` (선택 사항)

Step Functions에서 모든 하위 워크플로 실행 결과를 기록하는 Amazon S3 위치를 지정합니다.

Step Functions는 실행 입력 및 출력, ARN, 실행 상태와 같은 모든 하위 워크플로 실행 데이터를 통합합니다. 그런 다음 같은 상태의 실행을 지정된 Amazon S3 위치에 있는 각 파일로 내보냅니다. 자세한 정보는 ResultWriter을 참조하세요.

`Map` 상태 결과를 내보내지 않으면 모든 하위 워크플로 실행 결과 배열이 반환됩니다. 예:

```
[1, 2, 3, 4, 5]
```

`ResultPath` (선택 사항)

반복 출력을 배치할 위치를 입력에서 지정합니다. 그러면 입력은 `OutputPath` 필드(있는 경우)에서 지정한 대로 필터링된 후 상태 출력으로 전달됩니다. 자세한 내용은 입/출력 처리를 참조하십시오.

`ResultSelector` (선택 사항)

키-값 페어 컬렉션을 전달합니다. 여기서 값은 정적이거나 결과에서 선택된 값입니다. 자세한 정보는 ResultSelector을 참조하세요.

> **Tip**
>
> 상태 시스템에서 사용하는 Parallel 또는 Map 상태가 배열을 반환하는 경우 ResultSelector 필드를 사용하여 배열을 평면 배열로 변환할 수 있습니다. 자세한 정보는 배열의 배열 평면화을 참조하세요.

`Retry` (선택 사항)

Retrier라고 하는 객체 배열로, 재시도 정책을 정의합니다. 상태에서 런타임 오류가 발생하면 실행에서 재시도 정책을 사용합니다. 자세한 정보는 Retry 및 Catch를 사용하는 상태 시스템 예제을 참조하세요.

> **Note**
>
> Distributed Map 상태에 대한 Retrier를 정의하면 재시도 정책이 `Map` 상태가 시작된 모든 하위 워크플로 실행에 적용됩니다. 예를 들어 `Map` 상태에서 하위 워크플로 실행 3개를 시작했는데 그 중 하나가 실패했다고 가정해보겠습니다. 실패가 발생하면 실행에서 `Map` 상태의 `Retry` 필드(정의된 경우) 를 사용합니다. 재시도 정책은 실패한 실행뿐만 아니라 모든 하위 워크플로 실행에 적용됩니다. 하위 워크플로 실행이 하나 이상 실패하면 맵 실행이 실패합니다.
>
> `Map` 상태를 재시도하면 새 맵 실행이 생성됩니다.

`Catch` (선택 사항)

폴백(fallback) 상태를 정의하는 객체 배열(Catcher). 상태에서 런타임 오류가 발생하면 Step Functions는 `Catch`에 정의된 Catcher를 사용합니다. 오류가 발생하면 실행은 먼저 `Retry`에 정의된 모든 Retrier를 사용합니다. 재시도 정책이 정의되지 않았거나 삭제되면 실행에서 Catcher(정의된 경우)를 사용합니다. 자세한 내용은 폴백 상태를 참조하십시오.

다음 단계

계속해서 Distributed Map 상태에 대해 자세히 알아보려면 다음 리소스를 참조하세요.

- 입/출력 처리

  Distributed Map 상태에서 수신하는 입력과 생성되는 출력을 구성하기 위해 Step Functions는 다음 필드를 제공합니다.

- ItemReader
- ItemsPath
- ItemSelector
- ItemBatcher
- ResultWriter
- 입력 CSV 파일 파싱

Step Functions는 이러한 필드 외에도 Distributed Map의 허용 실패 임계값을 정의할 수 있는 기능을 제공합니다. 이 값을 사용하면 최대 실패 항목 수 또는 실패 항목 비율을 맵 실행의 실패 임계값으로 지정할 수 있습니다. 허용 실패 임계값 구성에 대한 자세한 내용은 Distributed Map 상태의 허용 실패 임계값 섹션을 참조하세요.

- Distributed Map 상태 사용

  Distributed Map 상태 사용을 시작하려면 다음 자습서와 샘플 프로젝트를 참조하세요.

  - Distributed Map 상태를 사용하여 시작하기
  - Lambda 함수를 사용하여 전체 데이터 배치 처리
  - Lambda 함수를 사용하여 개별 데이터 항목 처리
  - 샘플 프로젝트: Distributed Map을 사용하여 CSV 파일 처리
  - 샘플 프로젝트: Distributed Map을 사용하여 Amazon S3 버킷의 데이터 처리

- Distributed Map 상태 실행 예제

  Step Functions 콘솔은 Distributed Map 상태 실행과 관련된 모든 정보를 표시하는 맵 실행 세부 정보 페이지를 제공합니다. 이 페이지에 표시된 정보를 검사하는 방법에 대한 자세한 내용은 맵 실행 검사 섹션을 참조하세요.

Distributed Map 상태의 허용 실패 임계값

대규모 병렬 워크로드를 오케스트레이션할 때 허용 실패 임계값을 정의할 수도 있습니다. 이 값을 사용하면 최대 실패 항목 수 또는 실패 항목 비율을 맵 실행의 실패 임계값으로 지정할 수 있습니다. 지정한 값에 따라 임계값을 초과하면 맵 실행이 자동으로 실패합니다. 두 값을 모두 지정하는 경우 두 값 중 하나를 초과하면 워크플로가 실패합니다.

임계값을 지정하면 전체 맵 실행이 실패하기 전에 특정 개수의 항목이 실패할 수 있습니다. 지정된 임계값이 초과되어 맵 실행이 실패하면 Step Functions에서 `States.ExceedToleratedFailureThreshold` 오류를 반환합니다.

> **Note**
>
> Step Functions는 허용 실패 임계값이 초과되더라도 맵 실행이 실패하기 전에 맵 실행에서 하위 워크플로를 계속 실행할 수 있습니다.

Workflow Studio에서 임계값을 지정하려면 런타임 설정 필드의 추가 구성에서 허용된 실패 임계값 설정을 선택합니다.

허용 실패 백분율

허용할 수 있는 실패 항목 백분율을 정의합니다. 이 값을 초과하면 맵 실행이 실패합니다. Step Functions는 실패하거나 시간 초과된 총 항목 수를 총 항목 수로 나눈 결과로 실패 항목 백분율을 계산합니다. 0~100 범위의 값을 지정해야 합니다. 기본 백분율 값은 0입니다. 즉, 하위 워크플로 실행 중 하나라도 실패하거나 시간 초과되면 워크플로가 실패합니다. 백분율을 100으로 지정하면 모든 하위 워크플로 실행이 실패하더라도 워크플로는 실패하지 않습니다.

또는 백분율을 Distributed Map 상태 입력에서 기존 키-값 페어의 참조 경로로 지정할 수 있습니다. 이 경로는 런타임 시 0~100 사이의 양의 정수로 해석되어야 합니다. `ToleratedFailurePercentagePath` 하위 필드에 참조 경로를 지정합니다.

다음 입력을 예로 들어보겠습니다.

```
{
  "percentage": 15
}
```

다음과 같이 해당 입력에 대한 참조 경로를 사용하여 백분율을 지정할 수 있습니다.

```
{
  ...
  "Map": {
    "Type": "Map",
    ...
    "ToleratedFailurePercentagePath": "$.percentage"
    ...
  }
}
```

> **Important**
>
> Distributed Map 상태 정의에서 `ToleratedFailurePercentage` 또는 `ToleratedFailurePercentagePath`를 지정할 수 있지만 둘 다 지정할 수는 없습니다.

허용 실패 횟수

허용할 수 있는 실패 항목 수를 정의합니다. 이 값을 초과하면 맵 실행이 실패합니다.

또는 수를 Distributed Map 상태 입력에서 기존 키-값 페어의 참조 경로로 지정할 수 있습니다. 이 경로는 런타임 시의 양의 정수로 해석되어야 합니다. `ToleratedFailureCountPath` 하위 필드에 참조 경로를 지정합니다.

다음 입력을 예로 들어보겠습니다.

```
{
  "count": 10
}
```

다음과 같이 해당 입력에 대한 참조 경로를 사용하여 개수를 지정할 수 있습니다.

```
{
  ...
  "Map": {
    "Type": "Map",
    ...
    "ToleratedFailureCountPath": "$.count"
    ...
  }
}
```

> **Important**
>
> Distributed Map 상태 정의에서 `ToleratedFailureCount` 또는 `ToleratedFailureCountPath`를 지정할 수 있지만 둘 다 지정할 수는 없습니다.

Transitions

새로운 상태 시스템 실행을 시작하면 시스템은 최상위 `StartAt` 필드에서 참조하는 상태로 시작합니다. 이 필드(문자열로 지정)는 대/소문자를 포함하여 워크플로 상태 이름과 정확하게 일치해야 합니다.

상태를 실행한 후 AWS Step Functions는 `Next` 필드 값을 사용하여 진행할 다음 상태를 결정합니다.

또한 `Next` 필드는 상태 이름을 문자열로 지정합니다. 이 문자열은 대소문자를 구분하며 상태 시스템 설명에 지정된 상태 이름과 정확히 일치해야 합니다.

예를 들어 다음 상태에는 `NextState`에 대한 전환이 포함되어 있습니다.

```
"SomeState" : {
  ...,
  "Next" : "NextState"
}
```

대부분의 상태에서는 `Next` 필드를 통한 단일 전환 규칙만 허용합니다. 그러나 특정 흐름 제어 상태(예: `Choice` 상태)를 사용하면 여러 전환 규칙(각각에 자체 `Next` 필드 포함)을 지정할 수 있습니다. 전환 지정 방법에 대한 자세한 내용을 포함하여 사용자가 지정할 수 있는 각 상태 유형에 대한 자세한 내용은 Amazon States 언어에서 참조할 수 있습니다.

상태에는 다른 상태로부터 들어오는 전환이 여러 개 있을 수 있습니다.

터미널 상태(`"Type": Succeed`, `"Type": Fail` 또는 `"End": true`인 상태)에 도달하거나 런타임 오류가 발생할 때까지 프로세스가 반복됩니다.

실행을 redrive하면 상태 전환으로 간주됩니다. 또한 redrive에서 재실행되는 모든 상태도 상태 전환으로 간주됩니다.

상태 시스템 내 상태에는 다음 규칙이 적용됩니다.

- 상태는 둘러싸는 블록 내에서 어떠한 순서로도 발생할 수 있습니다. 그러나 나열된 순서는 실행 순서에는 영향을 미치지 않습니다. 이 순서는 상태의 내용에 따라 결정됩니다.
- 상태 시스템 내에는 `start` 상태로 지정된 상태 하나만 있을 수 있습니다. `start` 상태는 최상위 구조의 `StartAt` 필드 값으로 정의됩니다.
- 상태 시스템 논리에 따라(예: 상태 시스템에 논리 브랜치가 여러 개 있는 경우) `end` 상태 수가 하나를 넘을 수 있습니다.

- 상태 시스템이 상태 하나로만 구성된 경우 시작 상태와 종료 상태 모두가 될 수 있습니다.

Distributed Map 상태에서 전환

분산 모드에서 Map 상태를 사용하는 경우 Distributed Map 상태가 시작하는 하위 워크플로 실행마다 상태 전환 하나에 대한 요금이 청구됩니다. 인라인 모드에서 Map 상태를 사용하는 경우 Inline Map 상태를 반복할 때마다 상태 전환 요금이 청구되지 않습니다.

분산 모드에서 Map 상태를 사용하여 비용을 최적화하고 Map 상태 정의에 중첩된 워크플로를 포함할 수 있습니다. 또한 Distributed Map 상태는 Express 유형의 하위 워크플로 실행을 시작할 때 더 많은 가치를 더합니다. Step Functions는 Express 하위 워크플로 실행의 응답과 상태를 저장하므로 CloudWatch Logs에 실행 데이터를 저장할 필요성이 줄어듭니다. 또한 Distributed Map 상태에서 사용할 수 있는 흐름 제어 (예: 오류 임계값 정의 또는 항목 그룹 일괄 처리)에 액세스할 수 있습니다. Step Functions 요금에 대한 자세한 내용은 AWS Step Functions 요금을 참조하세요.

상태 머신 데이터

상태 머신 데이터는 다음과 같은 양식을 사용합니다.

- 상태 머신에의 초기 입력
- 상태 간 전송되는 데이터
- 상태 머신의 출력

이 섹션에서는 AWS Step Functions에서 상태 머신 데이터의 형식 지정 및 사용 방법에 대해 설명합니다.

주제

- 데이터 형식
- 상태 머신 입/출력
- 상태 입/출력

데이터 형식

상태 시스템 데이터는 JSON 텍스트로 표시됩니다. JSON에서 지원하는 모든 데이터 형식을 사용하여 상태 시스템에 값을 제공할 수 있습니다.

Note

- JSON 텍스트 형식의 수는 JavaScript 시맨틱을 따릅니다. 이러한 수는 일반적으로 배정밀도 IEEE-854 값에 적합합니다.
- 다음은 유효한 JSON 텍스트입니다.
  - 따옴표로 구분된 독립형 문자열
  - 객체
  - 배열
  - 숫자
  - 부울 값
  - `null`
- 상태 출력이 다음 상태의 입력이 됩니다. 그러나 입력 및 출력 처리를 사용하면 입력 데이터의 하위 집합에서 작업할 상태를 제한할 수 있습니다.

상태 머신 입/출력

두 가지 방법 중 하나로 초기 입력 데이터를 AWS Step Functions 상태 시스템에 제공할 수 있습니다. 실행을 시작할 때 데이터를 `StartExecution` 작업에 전달할 수 있습니다. Step Functions 콘솔에서 데이터를 상태 시스템으로 전달할 수도 있습니다. 초기 데이터는 상태 머신의 `StartAt` 상태에 전달됩니다. 아무 입력도 제공되지 않으면 빈 객체(`{}`)가 기본값이 됩니다.

마지막 상태(`terminal`)에 따라 실행의 출력이 반환됩니다. 이 출력 값은 실행 결과에 JSON 텍스트로 표시됩니다.

표준 워크플로의 경우 `DescribeExecution` 작업과 같은 외부 호출자를 사용하여 실행 내역에서 실행 결과를 검색할 수 있습니다. Step Functions 콘솔에서 실행 결과를 볼 수 있습니다.

Express 워크플로의 경우 로깅을 활성화하면 CloudWatch Logs에서 결과를 검색하거나 Step Functions 콘솔에서 실행을 보고 디버깅할 수 있습니다. 자세한 정보는 CloudWatch Logs를 사용하여 로깅 및 Step Functions 콘솔에서 실행 보기 및 디버깅 섹션을 참조하세요.

상태 시스템과 관련된 할당량도 고려해야 합니다. 자세한 정보는 할당량 섹션을 참조하세요.

상태 입/출력

각 상태의 입력은 이전 상태의 JSON 텍스트로 구성되거나 `StartAt` 상태의 경우 입력이 실행으로 전환됩니다. 특정 흐름 제어 상태는 입력을 출력으로 그대로 보냅니다.

다음 예에서는 상태 머신이 두 개의 수를 함께 추가합니다.

1. AWS Lambda 함수를 정의합니다.

```
function Add(input) {
  var numbers = JSON.parse(input).numbers;
  var total = numbers.reduce(
    function(previousValue, currentValue, index, array) {
      return previousValue + currentValue; });
  return JSON.stringify({ result: total });
}
```

2. 상태 시스템을 정의합니다.

```
{
  "Comment": "An example that adds two numbers together.",
  "StartAt": "Add",
  "Version": "1.0",
  "TimeoutSeconds": 10,
  "States":
    {
        "Add": {
          "Type": "Task",
          "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Add",
          "End": true
        }
    }
}
```

3. 다음 JSON 텍스트를 사용하여 실행을 시작합니다.

```
{ "numbers": [3, 4] }
```

   Add 상태가 JSON 텍스트를 수신하고 이를 Lambda 함수에 전달합니다.

   Lambda 함수에서 계산 결과를 상태에 반환합니다.

상태가 출력에 다음 값을 반환합니다.

```
{ "result": 7 }
```

[Add] 또한 상태 머신의 마지막 상태이므로 이 값이 상태 머신의 출력으로 반환됩니다.

마지막 상태에서 출력이 반환되지 않는 경우 상태 머신이 빈 객체({})를 반환합니다.

자세한 내용은 Step Functions에서 입력 및 출력 처리 섹션을 참조하세요.

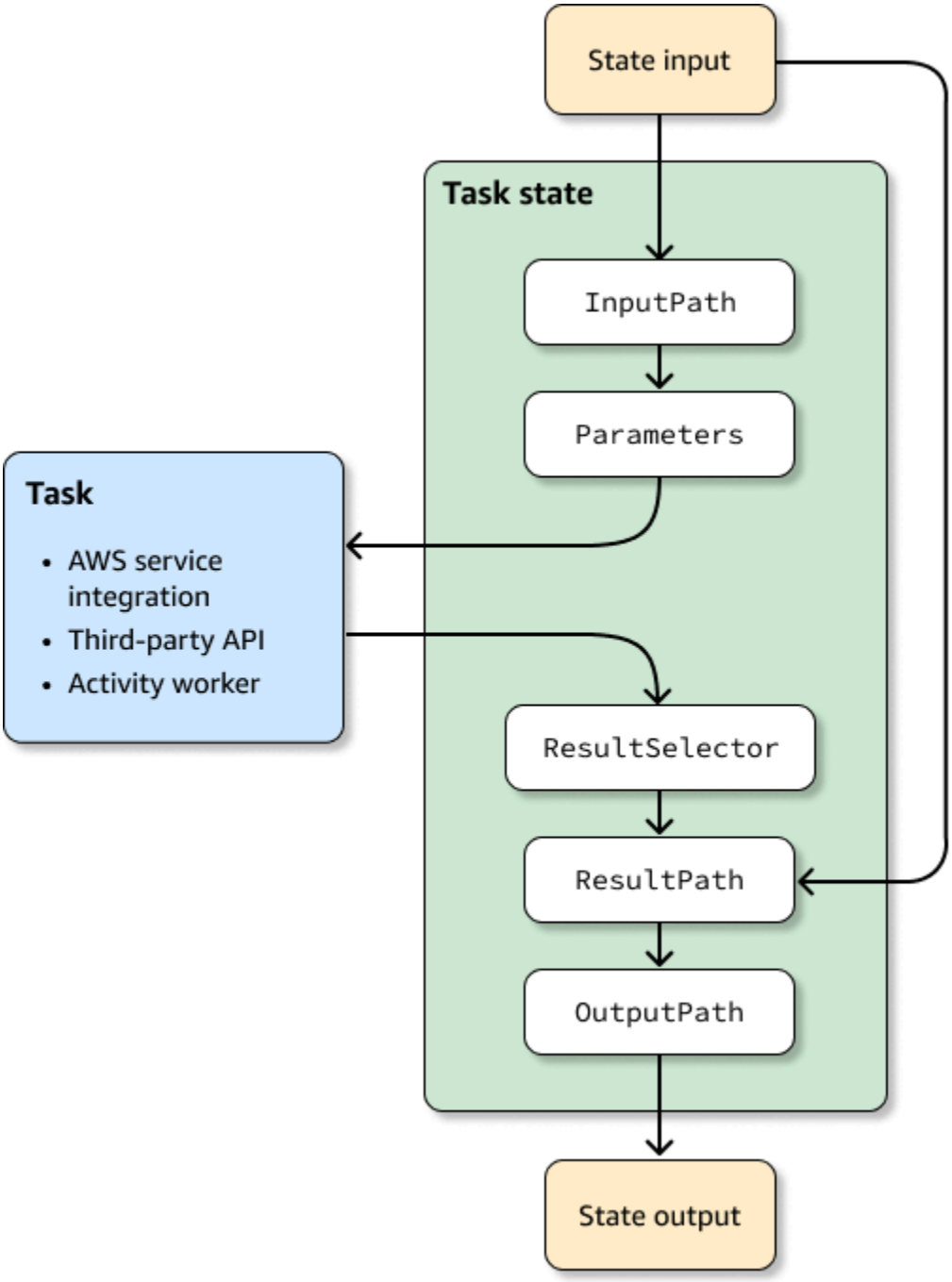
Step Functions에서 입력 및 출력 처리

Step Functions 실행에서 JSON 텍스트를 입력으로 수신하고 해당 입력을 워크플로의 첫 번째 상태에 전달합니다. 개별 상태는 JSON을 입력으로 수신하고 일반적으로 다음 상태에서 JSON을 출력으로 넘겨줍니다. 이 정보가 상태에서 상태로 어떻게 흐르는지 이해하고 이 데이터를 필터링하고 조작하는 방법에 대해 알아보는 것이 AWS Step Functions의 워크플로우를 효율적으로 설계하고 실행하는 데 중요합니다.

Amazon States Language에서 이러한 필드는 상태 간에서 JSON 흐름을 필터링하고 제어합니다.

- `InputPath`
- `Parameters`
- `ResultSelector`
- `ResultPath`
- `OutputPath`

다음 다이어그램은 작업 상태에서 JSON 정보가 이동하는 방식을 보여줍니다. `InputPath`Task상태 작업 (예: AWS Lambda 함수) 에 전달할 JSON 입력 부분을 선택합니다. `ResultPath`그런 다음 출력에 전달할 상태 입력과 작업 결과의 조합을 선택합니다. `OutputPath`JSON 출력을 필터링하여 출력에 전달되는 정보를 추가로 제한할 수 있습니다.



`InputPath`, `Parameters`, `ResultSelector`, `ResultPath` 및 `OutputPath` 각각은 워크플로의 각 상태를 이동할 때 JSON을 조작합니다.

이들은 각각 path를 사용하여 입력 또는 결과에서 JSON 부분을 선택할 수 있습니다. 경로는 JSON 텍스트 내의 노드를 식별하는 $로 시작하는 문자열입니다. Step Functions 경로는 JsonPath구문을 사용합니다.

> **Tip**
>
> Step Functions 콘솔의 데이터 흐름 시뮬레이터를 사용하면 JSON 경로 구문을 테스트하고 상태 내에서 데이터가 조작되는 방식을 더 잘 이해하며 데이터가 상태 간에 전달되는 방식을 확인할 수 있습니다.

> **Tip**
>
> 입력 및 출력 처리가 포함된 워크플로 예제를 배포하려면 AWS 계정모듈 6 - AWS Step Functions 워크숍의 입력 및 출력 처리를 참조하십시오.

주제

경로

Amazon States Language에서 경로는 JSON 텍스트 내 구성 요소를 식별하는 데 사용할 수 있는 `$`로 시작하는 문자열입니다. 경로는 JsonPath구문을 따릅니다. `InputPath`, `ResultPath` 및 `OutputPath`의 값을 지정할 때 입력 하위 집합에 액세스하기 위한 경로를 지정할 수 있습니다 자세한 내용은 Step Functions에서 입력 및 출력 처리단원을 참조하세요.

> **Note**
>
> 상태 정의의 `Parameters` 필드 내에서 경로를 사용하여 입력 또는 컨텍스트 객체의 JSON 노드를 지정할 수 있습니다. 파라미터를 서비스 API에 전달를 참조하세요.

필드 이름에 JsonPath ABNF 규칙 `member-name-shorthand` 정의에 포함되지 않은 문자가 포함된 경우 대괄호 표기법을 사용해야 합니다. 따라서 구두점(_ 제외)과 같은 특수 문자를 인코딩하려면 대괄호 표기법을 사용해야 합니다. 예를 들어 `$.abc.['def ghi']`입니다.

참조 경로

참조 경로는 JSON 구조에서 단일 노드만 식별할 수 있는 방식으로 구문이 제한되는 경로입니다.

- 객체 필드는 점(`.`) 및 대괄호(`[ ]`) 표기를 통해서만 액세스할 수 있습니다.
- `length()`가 지원되지 않은 등의 함수
- 기호가 아닌 어휘 연산자(예: `subsetof`)는 지원되지 않습니다.
- 정규 표현식을 기준으로 또는 JSON 구조의 다른 값을 참조하는 필터링은 지원되지 않습니다.
- `@`, `,`, `:`, 및 `?` 연산자는 지원되지 않습니다.

예를 들어 상태 입력 데이터에 다음과 같은 값이 포함된 경우:

```
{
  "foo": 123,
  "bar": ["a", "b", "c"],
  "car": {
      "cdr": true
  }
}
```

다음과 같은 참조 경로가 반환합니다.

```
$.foo => 123
$.bar => ["a", "b", "c"]
$.car.cdr => true
```

특정 상태에서는 경로 및 참조 경로를 사용하여 상태 머신의 흐름을 제어하거나 상태 설정 또는 옵션을 구성합니다. 자세한 내용은 데이터 흐름 시뮬레이터를 사용한 워크플로 입력 및 출력 경로 처리 모델링 및 JSONPath의 효과적인 사용을 참조하십시오. AWS Step Functions

배열의 배열 평면화

상태 시스템의 Parallel 또는 맵 상태에서 배열의 배열을 반환하는 경우 ResultSelector 필드를 사용하여 배열을 평면 배열로 변환할 수 있습니다. Parallel 또는 Map 상태 정의 내에 이 필드를 포함하여 이러한 상태의 결과를 조작할 수 있습니다.

배열을 평면화하려면 다음 예제와 같이 ResultSelector 필드에서 JMESPath 구문 [*]을 사용합니다.

```
"ResultSelector": {
    "flattenArray.$": "$[*][*]"
  }
```

배열을 평면화하는 방법을 보여주는 예제는 다음 자습서의 3단계를 참조하세요.

- Lambda 함수를 사용하여 전체 데이터 배치 처리
- Lambda 함수를 사용하여 개별 데이터 항목 처리

InputPath, 파라미터 및 ResultSelector

InputPath, Parameters 및 ResultSelector 필드는 JSON이 워크플로를 통해 이동할 때 JSON을 조작하는 방법을 제공합니다. InputPath는 경로를 사용하여 JSON 표기법을 필터링해 전달되는 입력을 제한할 수 있습니다(경로 참조). Parameters 필드를 사용하면 키-값 페어 컬렉션을 전달할 수 있습니다. 여기서 값은 상태 머신 정의에서 정의하는 정적 값 또는 경로를 사용하여 입력에서 선택하는 값입니다. ResultSelector 필드는 ResultPath가 적용되기 전의 상태 결과를 조작할 수 있는 방법을 제공합니다.

AWS Step Functions InputPath필드를 먼저 적용한 다음 Parameters 필드를 적용합니다. 우선 InputPath를 사용하여 원시 입력을 필터링한 다음 Parameters를 적용하여 입력을 추가로 조작하거나 새 값을 추가할 수 있습니다. 그런 다음 ResultSelector 필드를 사용하여 ResultPath가 적용되기 전의 상태 출력을 조작할 수 있습니다.

> **Tip**
>
> Step Functions 콘솔의 데이터 흐름 시뮬레이터를 사용하면 JSON 경로 구문을 테스트하고 상태 내에서 데이터가 조작되는 방식을 더 잘 이해하며 데이터가 상태 간에 전달되는 방식을 확인할 수 있습니다.

InputPath

InputPath를 사용하여 상태 입력의 부분을 선택합니다.

예를 들어, 상태에 대한 입력에 다음이 포함된다고 가정합니다.

```
{
  "comment": "Example for InputPath.",
  "dataset1": {
    "val1": 1,
    "val2": 2,
    "val3": 3
  },
  "dataset2": {
    "val1": "a",
    "val2": "b",
    "val3": "c"
  }
}
```

InputPath를 적용할 수 있습니다.

```
"InputPath": "$.dataset2",
```

위의 InputPath를 통해 입력으로 전달되는 JSON은 다음과 같습니다.

```
{
  "val1": "a",
  "val2": "b",
  "val3": "c"
}
```

Note

경로는 값의 모음을 출력할 수 있습니다. 다음 예제를 살펴보세요.

```
{ "a": [1, 2, 3, 4] }
```

경로 $.a[0:2]를 적용하는 경우 결과는 다음과 같습니다.

```
[ 1, 2 ]
```

파라미터

이 섹션에서는 파라미터 필드를 사용할 수 있는 다양한 방법을 설명합니다.

키-값 페어

Parameters 필드를 사용하여 입력으로 전달되는 키-값 페어 컬렉션을 만듭니다. 각각의 값은 상태 머신 정의에 포함한 정적 값 또는 path를 통해 입력 또는 컨텍스트 객체에서 선택한 값일 수 있습니다. path를 사용하여 값을 선택한 키-값 페어의 경우 키 이름이 .$ 기호로 끝나야 합니다.

예를 들어 다음과 같은 입력을 제공한다고 가정합니다.

```
{
  "comment": "Example for Parameters.",
  "product": {
    "details": {
       "color": "blue",
       "size": "small",
       "material": "cotton"
    },
    "availability": "in stock",
    "sku": "2317",
    "cost": "$23"
  }
}
```

정보의 일부를 선택하기 위해 상태 머신 정의에 이러한 파라미터를 지정할 수 있습니다.

```
"Parameters": {
        "comment": "Selecting what I care about.",
        "MyDetails": {
          "size.$": "$.product.details.size",
          "exists.$": "$.product.availability",
          "StaticValue": "foo"
        }
      },
```

이전 입력 및 Parameters 필드를 지정하면 다음과 같은 JSON이 전달됩니다.

```
{
  "comment": "Selecting what I care about.",
  "MyDetails": {
      "size": "small",
      "exists": "in stock",
      "StaticValue": "foo"
  }
```

```
},
```

입력 외에도 컨텍스트 객체라고 알려진 특수 JSON 객체에 액세스할 수 있습니다. 컨텍스트 객체에는 상태 머신 실행에 대한 정보가 포함됩니다. 컨텍스트 객체를 참조하세요.

연결된 리소스

또한 Parameters 필드는 정보를 연결된 리소스로 전달할 수 있습니다. 예를 들어 작업 상태가 작업을 오케스트레이션하고 있는 경우 관련 API 매개변수를 해당 서비스의 API 작업에 직접 전달할 수 있습니다. AWS Batch 자세한 내용은 다음을 참조하세요.

- 파라미터를 서비스 API에 전달
- 다른 서비스와 함께 사용

Amazon S3

상태 간에 전달하는 Lambda 함수 데이터가 262,144바이트를 초과하는 경우 Amazon S3를 사용하여 데이터를 저장하고 다음 방법 중 하나를 구현하는 것이 좋습니다.

- Map 상태에서 Amazon S3 데이터 소스에서 직접 입력을 읽을 수 있도록 워크플로에서 Distributed Map 상태를 사용합니다. 자세한 정보는 분산 모드에서 Map 상태 사용을 참조하세요.
- Payload 파라미터에 있는 버킷의 Amazon 리소스 이름(ARN)을 파싱하여 버킷 이름과 키 값을 가져옵니다. 자세한 정보는 대용량 페이로드를 전달하는 대신 Amazon S3 ARN 사용을 참조하세요.

또는 구현을 조정하여 실행에서 더 작은 용량의 페이로드를 전달할 수 있습니다.

ResultSelector

ResultSelector 필드를 사용하여 ResultPath가 적용되기 전의 상태 결과를 조작합니다. 이 ResultSelector 필드를 사용하면 키 값 페어 컬렉션을 만들 수 있습니다. 여기서 값은 정적이거나 상태 결과에서 선택한 값입니다. ResultSelector 필드를 사용하면 ResultPath 필드에 전달하려는 상태 결과 부분을 선택할 수 있습니다.

> **Note**
>
> ResultPath 필드를 사용하여 ResultSelector 필드 출력을 원래 입력에 추가할 수 있습니다.

ResultSelector는 다음 상태의 선택적 필드입니다.

- 맵
- 태스크 상태
- Parallel

예를 들어 Step Functions 서비스 통합에서는 결과의 페이로드 외에 메타데이터를 반환합니다. ResultSelector는 결과의 일부를 선택하고 ResultPath를 사용하여 상태 입력과 병합할 수 있습니다. 이 예제에서는 resourceType 및 ClusterId만 선택하고 이를 Amazon EMR createCluster.sync의 상태 입력과 병합하려고 합니다. 다음을 예로 들어보겠습니다.

```
{
  "resourceType": "elasticmapreduce",
  "resource": "createCluster.sync",
  "output": {
    "SdkHttpMetadata": {
      "HttpHeaders": {
        "Content-Length": "1112",
        "Content-Type": "application/x-amz-JSON-1.1",
        "Date": "Mon, 25 Nov 2019 19:41:29 GMT",
        "x-amzn-RequestId": "1234-5678-9012"
      },
      "HttpStatusCode": 200
    },
    "SdkResponseMetadata": {
      "RequestId": "1234-5678-9012"
    },
    "ClusterId": "AKIAIOSFODNN7EXAMPLE"
  }
}
```

그런 다음 ResultSelector를 사용하여 resourceType 및 ClusterId를 선택할 수 있습니다.

```
"Create Cluster": {
  "Type": "Task",
  "Resource": "arn:aws:states:::elasticmapreduce:createCluster.sync",
  "Parameters": {
    <some parameters>
  },
  "ResultSelector": {
    "ClusterId.$": "$.output.ClusterId",
```

```
    "ResourceType.$": "$.resourceType"
  },
  "ResultPath": "$.EMROutput",
  "Next": "Next Step"
}
```

지정된 입력에서 `ResultSelector`를 사용하면 다음이 생성됩니다.

```
{
  "OtherDataFromInput": {},
  "EMROutput": {
    "ResourceType": "elasticmapreduce",
    "ClusterId": "AKIAIOSFODNN7EXAMPLE"
  }
}
```

배열의 배열 평면화

상태 시스템의 Parallel 또는 맵 상태에서 배열의 배열을 반환하는 경우 ResultSelector 필드를 사용하여 배열을 평면 배열로 변환할 수 있습니다. Parallel 또는 Map 상태 정의 내에 이 필드를 포함하여 이러한 상태의 결과를 조작할 수 있습니다.

배열을 평면화하려면 다음 예제와 같이 `ResultSelector` 필드에서 JMESPath 구문 `[*]`을 사용합니다.

```
"ResultSelector": {
    "flattenArray.$": "$[*][*]"
  }
```

배열을 평면화하는 방법을 보여주는 예제는 다음 자습서의 3단계를 참조하세요.

- Lambda 함수를 사용하여 전체 데이터 배치 처리
- Lambda 함수를 사용하여 개별 데이터 항목 처리

ResultPath

상태 출력은 상태 입력의 복사본, 상태 결과(예, `Task` 상태의 Lambda 함수 출력), 또는 상태 입력 및 결과의 조합일 수 있습니다. `ResultPath`를 사용하여 상태 출력으로 이들의 어떤 조합을 전달할지 관리합니다.

다음 상태 유형은 결과를 생성하고 `ResultPath:`를 포함할 수 있습니다.

- Pass
- 태스크 상태
- Parallel
- 맵

`ResultPath`를 사용하여 작업 입력과 작업 결과를 결합하거나 이들 중 하나를 선택합니다. `ResultPath`에 제공한 경로는 어떤 정보가 출력으로 전달될지 관리합니다.

> **Note**
>
> `ResultPath`는 JSON에서 단일 노드만 식별하는 범위가 제한된 참조 경로의 사용으로 한정됩니다. Amazon States Language의 참조 경로를 참조하세요.

이 예제는 Lambda를 사용하는 Step Functions 상태 시스템 만들기 자습서에 설명된 상태 시스템과 Lambda 함수를 기반으로 합니다. `ResultPath` 필드의 다양한 경로를 시도하여 자습서를 실습하고 다른 출력을 테스트합니다.

ResultPath 사용 시:

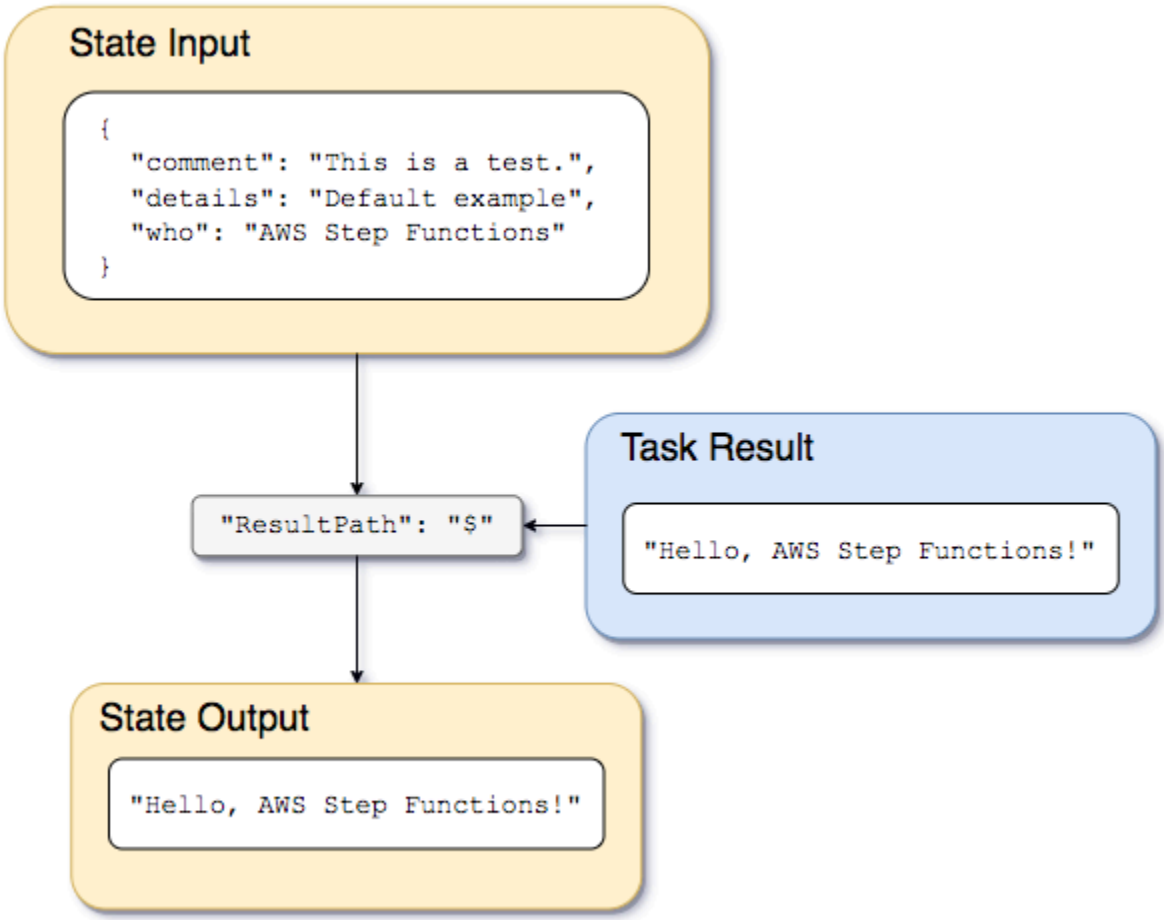
- 입력을 결과로 바꾸는 ResultPath 데 사용
- 결과 삭제 및 원래의 입력 유지
- 결과를 입력에 포함하는 ResultPath 데 사용합니다.
- 입력의 노드를 결과로 업데이트하는 ResultPath 데 사용합니다.
- ResultPath a에 오류와 입력을 모두 포함하는 데 사용합니다. Catch

> **Tip**
>
> Step Functions 콘솔의 데이터 흐름 시뮬레이터를 사용하면 JSON 경로 구문을 테스트하고 상태 내에서 데이터가 조작되는 방식을 더 잘 이해하며 데이터가 상태 간에 전달되는 방식을 확인할 수 있습니다.

입력을 결과로 바꾸는 ResultPath 데 사용

`ResultPath`를 지정하지 않는 경우, 기본값은 `"ResultPath": "$"`에 지정된 것처럼 행동합니다. 상태가 입력을 결과로 대체하도록 하기 때문에 상태 입력은 작업 결과의 결과로 완전히 대체됩니다.

다음 도표는 `ResultPath`가 입력을 작업의 결과로 완전히 대체하는 방법을 보여줍니다.



Lambda를 사용하는 Step Functions 상태 시스템 만들기에 설명된 상태 시스템과 Lambda 함수를 사용하고 서비스 통합 유형을 Lambda 함수의 AWS SDK 통합으로 변경합니다. 이렇게 하려면 다음 예제와 같이 `Task` 상태의 `Resource` 필드에 Lambda 함수 Amazon 리소스 이름(ARN)을 지정합니다. AWS SDK 통합을 사용하면 `Task` 상태 결과에 메타데이터 없이 Lambda 함수 출력만 포함됩니다.

```
{
   "StartAt":"CallFunction",
   "States":{
      "CallFunction": {
         "Type":"Task",
```

```
         "Resource":"arn:aws:lambda:us-east-2:123456789012:function:HelloFunction",
         "End": true
      }
   }
}
```

그런 다음 다음 입력을 전달합니다.

```
{
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 함수에서 다음 결과를 제공합니다.

```
"Hello, AWS Step Functions!"
```

> **Tip**
>
> Step Functions 콘솔에서 이 결과를 볼 수 있습니다. 이렇게 하려면 콘솔의 실행 세부 정보 페이지에 있는 그래프 보기에서 Lambda 함수를 선택합니다. 그런 다음 단계 세부 정보 창에서 출력 탭을 선택하여 이 결과를 확인합니다.

`ResultPath`가 상태에서 지정되지 않은 경우 또는 `"ResultPath": "$"`가 설정된 경우 상태 입력은 Lambda 함수 결과로 대체되고 상태 출력은 다음과 같습니다.

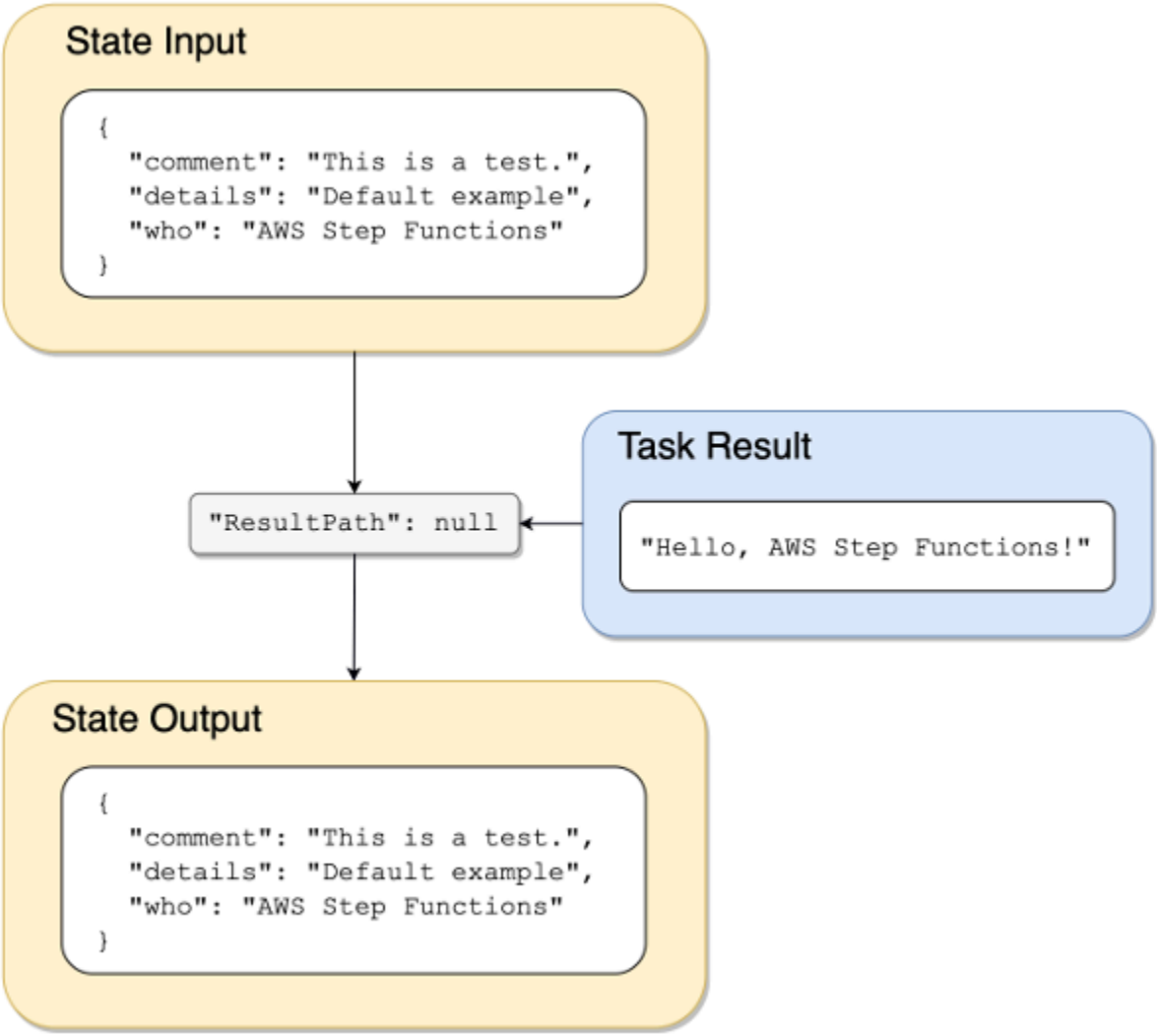
```
"Hello, AWS Step Functions!"
```

> **Note**
>
> `ResultPath`는 결과를 출력으로 전달하기 전에 입력에 결과의 내용을 포함할 경우 사용됩니다. 하지만 `ResultPath`가 지정되지 않으면 기본적으로 전체 입력을 대체합니다.

결과 삭제 및 원래의 입력 유지

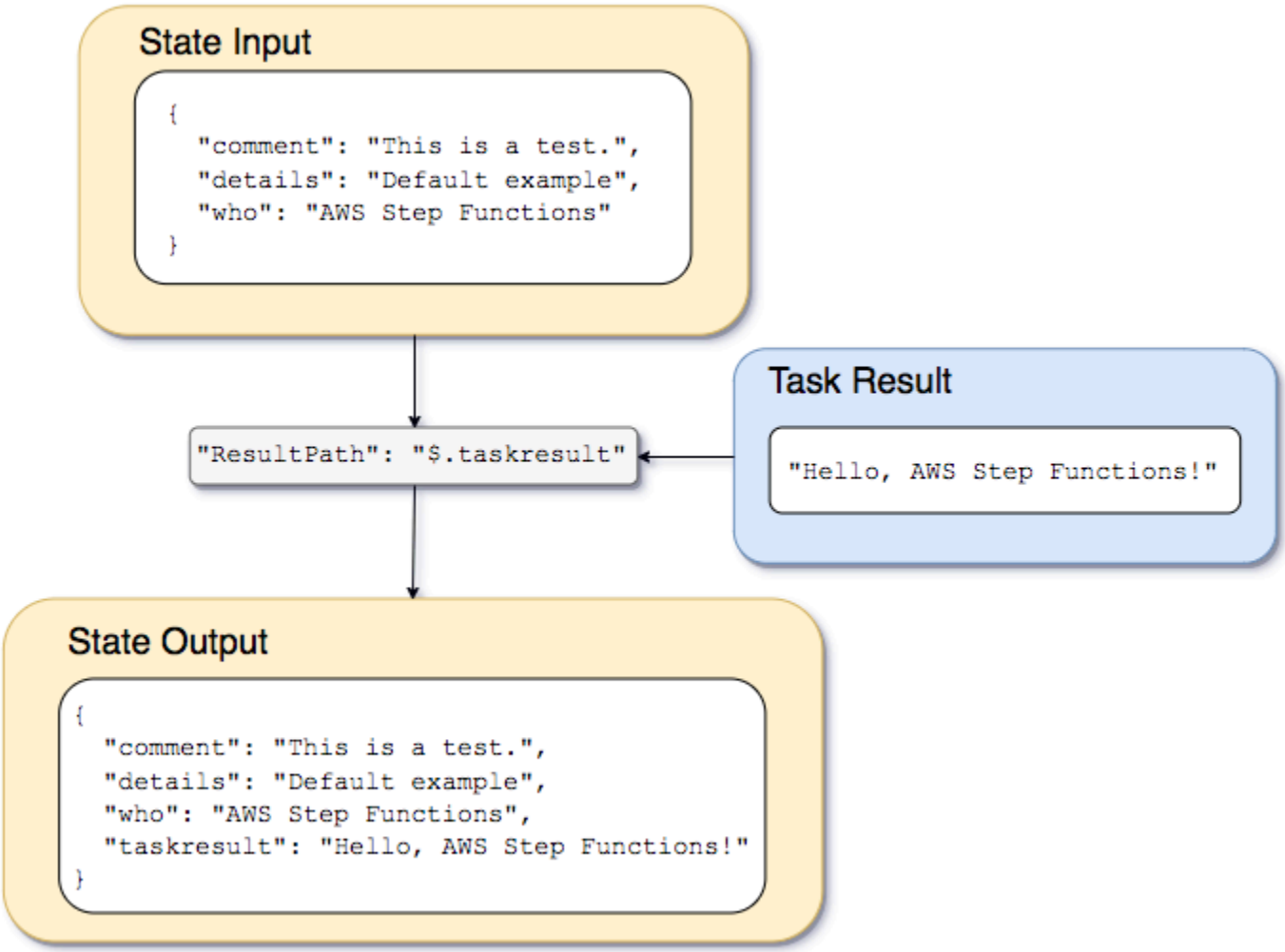
`ResultPath`를 `null`로 설정하면 원래의 입력이 출력에 전달됩니다. `"ResultPath": null`를 사용하면 상태의 입력 페이로드가 결과에 관계없이 출력에 직접 복사됩니다.

다음 다이어그램은 null `ResultPath`가 입력을 직접 출력에 복사하는 방법을 보여줍니다.



결과를 입력에 포함하는 ResultPath 데 사용합니다.

다음 그림은 `ResultPath`가 입력에 결과를 포함하는 방법을 보여줍니다.



Lambda를 사용하는 Step Functions 상태 시스템 만들기 자습서에 설명된 상태 시스템과 Lambda 함수를 사용하여 다음 입력을 전달할 수 있습니다.

```
{
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 함수 결과는 다음과 같습니다.

```
"Hello, AWS Step Functions!"
```

입력을 보존하고 Lambda 함수 결과를 삽입한 후 통합된 JSON을 다음 상태에 전달하려면 `ResultPath`를 다음과 같이 설정하면 됩니다.

```
"ResultPath": "$.taskresult"
```

이렇게 하면 Lambda 함수 결과가 원래 입력에 포함됩니다.

```
{
  "comment": "This is a test of input and output of a Task state.",
  "details": "Default behavior example",
  "who": "AWS Step Functions",
  "taskresult": "Hello, AWS Step Functions!"
}
```

Lambda 함수 출력은 `taskresult` 값으로 기존 입력에 추가됩니다. 새로이 입력된 값을 포함하여 이 입력은 다음 상태로 전달됩니다.

결과를 입력의 하위 노드로 입력할 수 있습니다. `ResultPath` 속성을 다음과 같이 설정합니다.

```
"ResultPath": "$.strings.lambdaresult"
```

다음 입력을 사용하여 실행을 시작합니다.

```
{
  "comment": "An input comment.",
  "strings": {
    "string1": "foo",
    "string2": "bar",
    "string3": "baz"
  },
  "who": "AWS Step Functions"
}
```

Lambda 함수 결과는 입력에서 `strings` 노드의 하위 값으로 삽입됩니다.

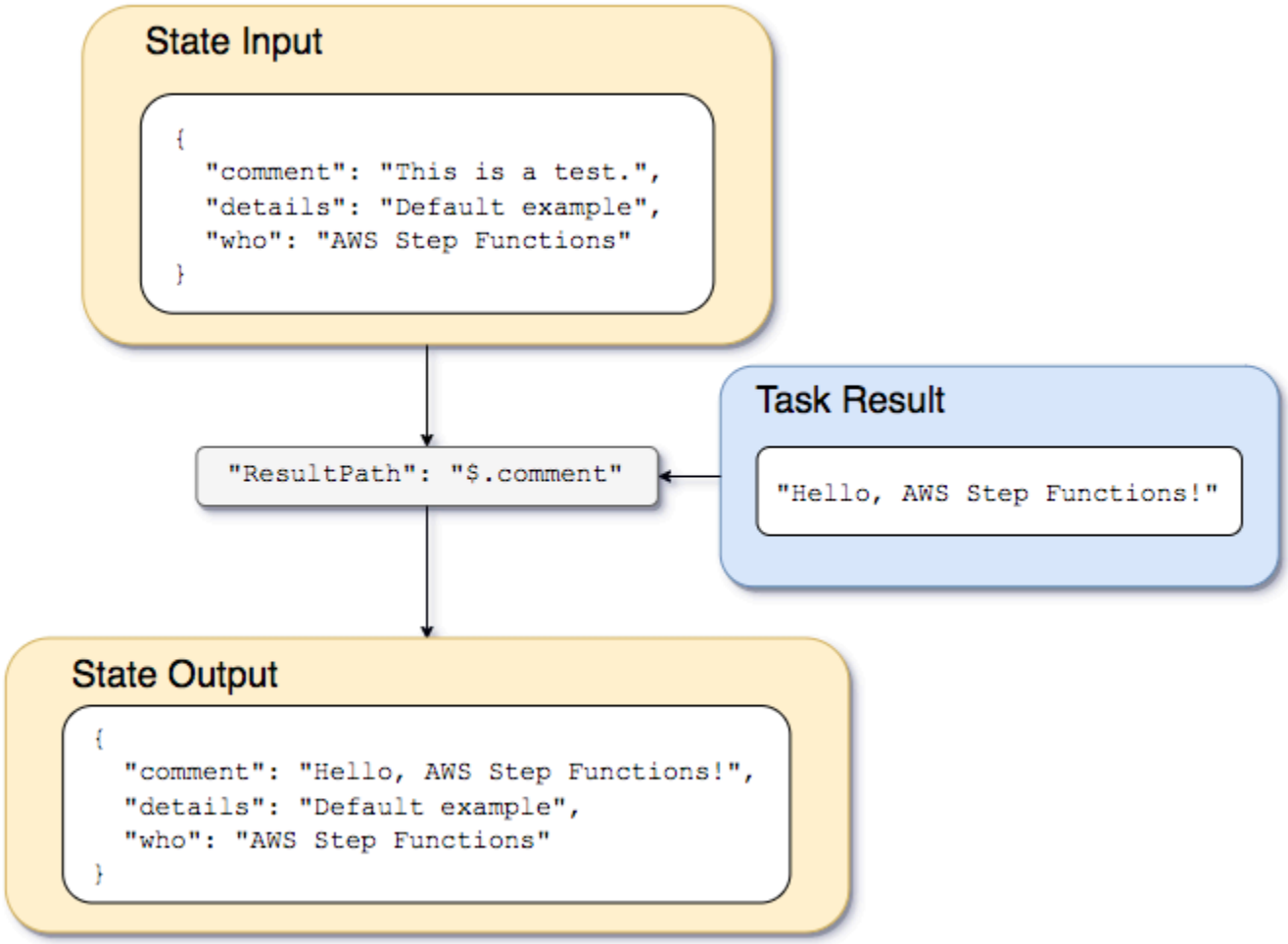
```
{
  "comment": "An input comment.",
  "strings": {
    "string1": "foo",
    "string2": "bar",
    "string3": "baz",
```

```
    "lambdaresult": "Hello, AWS Step Functions!"
  },
  "who": "AWS Step Functions"
}
```

다음 상태 출력은 기존 입력 JSON과 결과를 하위 노드로 포함합니다.

입력의 노드를 결과로 업데이트하는 ResultPath 데 사용합니다.

다음 도표는 `ResultPath`가 작업 결과의 값으로 입력에서 기존 JSON 노드의 값을 업데이트하는 방법을 보여줍니다.



Lambda를 사용하는 Step Functions 상태 시스템 만들기 자습서에 설명된 상태 시스템과 Lambda 함수의 예제를 사용하여 다음 입력을 전달할 수 있습니다.

```
{
```

```
  "comment": "This is a test of the input and output of a Task state.",
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Lambda 함수 결과는 다음과 같습니다.

```
Hello, AWS Step Functions!
```

JSON에서 입력을 보존하고 결과를 새로운 노드로 입력하는 대신에 기존 노드를 덮어쓰기할 수 있습니다.

예를 들어, `"ResultPath": "$"`를 삭제하고 설정하면 전체 노드를 덮어쓰는 것처럼 개별 노드를 결과로 덮어쓰도록 지정할 수 있습니다.

```
"ResultPath": "$.comment"
```

`comment` 노드는 상태 입력에 이미 존재하므로`ResultPath`를 `"$.comment"`로 설정하면 입력의 노드가 Lambda 함수 결과로 바꿉니다. `OutputPath`의 추가 필터링 없이, 다음과 같이 출력으로 전달됩니다.

```
{
  "comment": "Hello, AWS Step Functions!",
  "details": "Default behavior example",
  "who": "AWS Step Functions",
}
```

`comment` 노드 값 `"This is a test of the input and output of a Task state."`는 상태 출력에서 Lambda 함수 결과인 `"Hello, AWS Step Functions!"`로 대체됩니다.

ResultPath a에 오류와 입력을 모두 포함하는 데 사용합니다. **Catch**

이 Step Functions 상태 시스템을 사용하여 오류 조건 처리 자습서에서는 상태 머신이 오류를 잡아내는 방법을 보여줍니다. 경우에 따라 오류가 있는 기존 입력을 보존하고자 할 수 있습니다. `Catch`에 `ResultPath`를 사용하면 원래 입력을 바꾸는 대신 오류를 포함시킬 수 있습니다.

```
"Catch": [{
  "ErrorEquals": ["States.ALL"],
  "Next": "NextTask",
```

```
  "ResultPath": "$.error"
}]
```

이전 Catch 상태가 오류를 발견하면 상태 입력 내 error 노드에 결과를 포함합니다. 예를 들어 다음 입력을 고려해 보십시오.

```
{"foo": "bar"}
```

오류 발견 시 상태 출력은 다음과 같습니다.

```
{
  "foo": "bar",
  "error": {
    "Error": "Error here"
  }
}
```

오류를 처리하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- Step Functions에서 오류 처리
- Step Functions 상태 시스템을 사용하여 오류 조건 처리

OutputPath

OutputPath를 사용하면 다음 상태로 전달할 상태 출력 부분을 선택할 수 있습니다. 이렇게 하면 원치 않는 정보를 필터링하고 관심 있는 JSON 부분만 전달할 수 있습니다.

OutputPath를 지정하지 않으면 기본값 $가 사용됩니다. 이 값은 전체 JSON 노드(상태 입력, 작업 결과 및 ResultPath에 따라 결정됨)를 다음 상태로 전달합니다.

> **Tip**
>
> Step Functions 콘솔의 데이터 흐름 시뮬레이터를 사용하면 JSON 경로 구문을 테스트하고 상태 내에서 데이터가 조작되는 방식을 더 잘 이해하며 데이터가 상태 간에 전달되는 방식을 확인할 수 있습니다.

자세한 내용은 다음을 참조하십시오.

- Amazon States Language의 경로
- InputPath, ResultPath 및 OutputPath 예제
- 정적 JSON을 파라미터로 전달
- Step Functions에서 입력 및 출력 처리

InputPath, ResultPath 및 OutputPath 예제

Fail 상태 또는 Succeed 상태 이외의 모든 상태에는 `InputPath`, `ResultPath` 또는 `OutputPath`와 같은 입력 및 출력 처리 필드가 포함될 수 있습니다. 또한 Wait 및 Choice 상태는 `ResultPath` 필드를 지원하지 않습니다. 이러한 필드를 사용하면 JsonPath를 사용하여 워크플로를 통해 이동하는 JSON 데이터를 필터링할 수 있습니다.

또한 `Parameters` 필드를 사용하여 워크플로를 통해 이동하는 JSON 데이터를 조작할 수 있습니다. `Parameters` 사용에 대한 자세한 내용은 InputPath, 파라미터 및 ResultSelector을 참조하십시오.

예를 들어 Lambda를 사용하는 Step Functions 상태 시스템 만들기 자습서에 설명된 AWS Lambda 함수와 상태 시스템을 시작합니다. 상태 시스템을 수정하여 다음 `InputPath`, `ResultPath`, `OutputPath`를 포함할 수 있습니다.

```
{
  "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda
 function",
  "StartAt": "HelloWorld",
  "States": {
    "HelloWorld": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:HelloFunction",
      "InputPath": "$.lambda",
      "ResultPath": "$.data.lambdaresult",
      "OutputPath": "$.data",
      "End": true
    }
  }
}
```

다음 입력을 사용하여 실행을 시작합니다.

```
{
  "comment": "An input comment.",
```

```
  "data": {
    "val1": 23,
    "val2": 17
  },
  "extra": "foo",
  "lambda": {
    "who": "AWS Step Functions"
  }
}
```

comment 및 extra 노드를 폐기하지만 data 노드의 정보를 보존하면서 동시에 Lambda 함수의 출력을 포함하려고 한다고 가정해보겠습니다.

업데이트된 상태 시스템에서 Task 상태를 변경하여 작업으로 입력을 전달합니다.

```
"InputPath": "$.lambda",
```

상태 시스템 정의의 이 라인은 작업 입력을 상태 입력의 lambda 노드로만 한정합니다. Lambda 함수는 입력으로 JSON 객체 {"who": "AWS Step Functions"}만 수신합니다.

```
"ResultPath": "$.data.lambdaresult",
```

이 ResultPath는 상태 시스템에게 Lambda 함수 결과를 기존 상태 시스템 입력의 data 노드 하위 개념으로 lambdaresult라는 노드에 입력하라고 지시합니다. 원래 입력과 OutputPath를 사용한 결과에서 다른 어떠한 조작도 수행하지 않으므로 이제 상태 출력에는 원래 입력과 함께 Lambda 함수의 결과가 포함됩니다.

```
{
  "comment": "An input comment.",
  "data": {
    "val1": 23,
    "val2": 17,
    "lambdaresult": "Hello, AWS Step Functions!"
  },
  "extra": "foo",
  "lambda": {
    "who": "AWS Step Functions"
  }
}
```

하지만 목적은 data 노드만 보존하고 Lambda 함수의 결과를 포함하는 것이었습니다. OutputPath는 이런 결합된 JSON을 상태 출력으로 보내기 전에 필터링합니다.

```
"OutputPath": "$.data",
```

기존 입력에서 data 노드(ResultPath가 입력한 lambdaresult 하위 개념 포함)만 선택하여 출력으로 전달될 수 있게 합니다. 상태 출력은 다음과 같이 필터링됩니다.

```
{
  "val1": 23,
  "val2": 17,
  "lambdaresult": "Hello, AWS Step Functions!"
}
```

이 Task작업 상태에서:

1. InputPath는 lambda 노드만 입력에서 Lambda 함수로 보냅니다.
2. ResultPath는 기존 입력의 data 노드의 하위 개념으로 결과를 입력합니다.
3. OutputPath는 상태 입력에서 data 노드만 상태 출력으로 전달하도록 이제 Lambda 함수의 결과를 포함된 상태 입력을 필터링합니다.

Example JsonPath를 사용하여 원래 상태 시스템 입력, 결과 및 최종 출력을 조작하는 예제

보험 신청자의 신원과 주소를 확인하는 다음 상태 시스템을 고려해보겠습니다.

> **Note**
>
> 전체 예제를 보려면 Step Functions에서 JSON 경로를 사용하는 방법을 참조하세요.

```
{
  "Comment": "Sample state machine to verify an applicant's ID and address",
  "StartAt": "Verify info",
  "States": {
    "Verify info": {
      "Type": "Parallel",
      "End": true,
      "Branches": [
```

```
        {
          "StartAt": "Verify identity",
          "States": {
            "Verify identity": {
              "Type": "Task",
              "Resource": "arn:aws:states:::lambda:invoke",
              "Parameters": {
                "Payload.$": "$",
                "FunctionName": "arn:aws:lambda:us-east-2:111122223333:function:check-
identity:$LATEST"
              },
              "End": true
            }
          }
        },
        {
          "StartAt": "Verify address",
          "States": {
            "Verify address": {
              "Type": "Task",
              "Resource": "arn:aws:states:::lambda:invoke",
              "Parameters": {
                "Payload.$": "$",
                "FunctionName": "arn:aws:lambda:us-east-2:111122223333:function:check-
address:$LATEST"
              },
              "End": true
            }
          }
        }
      ]
    }
  }
}
```

다음 입력을 사용하여 이 상태 시스템을 실행하면 검증을 수행하는 Lambda 함수에서 확인이 필요한 데이터만 입력으로 예상하므로 실행이 실패합니다. 따라서 적절한 JsonPath를 사용하여 확인할 정보가 포함된 노드를 지정해야 합니다.

```
{
  "data": {
    "firstname": "Jane",
    "lastname": "Doe",
```

```
    "identity": {
      "email": "jdoe@example.com",
      "ssn": "123-45-6789"
    },
    "address": {
      "street": "123 Main St",
      "city": "Columbus",
      "state": "OH",
      "zip": "43219"
    },
    "interests": [
      {
        "category": "home",
        "type": "own",
        "yearBuilt": 2004
      },
      {
        "category": "boat",
        "type": "snowmobile",
        "yearBuilt": 2020
      },
      {
        "category": "auto",
        "type": "RV",
        "yearBuilt": 2015
      },
    ]
  }
}
```

check-identity Lambda 함수에서 사용해야 하는 노드를 지정하려면 다음과 같이 InputPath 필드를 사용합니다.

```
"InputPath": "$.data.identity"
```

그리고 *check-address* Lambda 함수에서 사용해야 하는 노드를 지정하려면 다음과 같이 InputPath 필드를 사용합니다.

```
"InputPath": "$.data.address"
```

이제 원본 상태 시스템 입력 내에 검증 결과를 저장하려면 다음과 같이 ResultPath 필드를 사용합니다.

```
"ResultPath": "$.results"
```

하지만 ID 및 확인 결과만 필요하고 원래 입력을 삭제하려면 다음과 같이 `OutputPath` 필드를 사용합니다.

```
"OutputPath": "$.results"
```

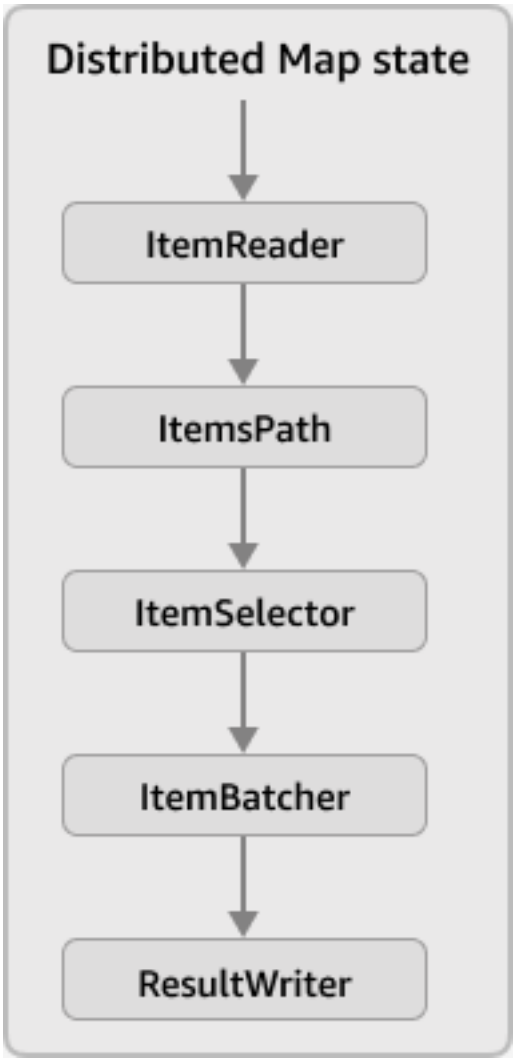
자세한 내용은 Step Functions에서 입력 및 출력 처리 섹션을 참조하세요.

Map 상태 입력 및 출력 필드

`Map` 상태는 JSON 배열, Amazon S3 객체 목록 또는 Amazon S3 버킷의 CSV 파일 행과 같은 데이터 세트의 항목 컬렉션을 동시에 반복합니다. 컬렉션의 항목마다 일련의 단계를 반복합니다. 이러한 필드를 사용하여 `Map` 상태에서 수신하는 입력과 상태에서 생성하는 출력을 구성할 수 있습니다. Step Functions는 Distributed Map 상태의 각 필드를 다음 목록과 그림에 표시된 순서대로 적용합니다.

> **Note**
>
> 사용 사례에 따라 이러한 필드를 모두 적용하지 않아도 됩니다.

1. ItemReader
2. ItemsPath
3. ItemSelector
4. ItemBatcher
5. ResultWriter



> **Note**
>
> 현재 Step Functions 콘솔의 데이터 흐름 시뮬레이터에서는 이러한 Map 상태 입력 및 출력 필드를 사용할 수 없습니다.

ItemReader

`ItemReader` 필드는 데이터 세트와 해당 위치를 지정하는 JSON 객체입니다. Distributed Map 상태는 이 데이터 세트를 입력으로 사용합니다. 다음 예제에서는 데이터 세트가 Amazon S3 버킷에 저장된 CSV 파일인 경우의 `ItemReader` 필드 구문을 보여줍니다.

```
"ItemReader": {
    "ReaderConfig": {
```

```
        "InputType": "CSV",
        "CSVHeaderLocation": "FIRST_ROW"
    },
    "Resource": "arn:aws:states:::s3:getObject",
    "Parameters": {
        "Bucket": "myBucket",
        "Key": "csvDataset/ratings.csv"
    }
}
```

> **Tip**
>
> Workflow Studio의 항목 소스 필드에 데이터 세트와 해당 위치를 지정합니다.

목차

ItemReader 필드 콘텐츠

데이터 세트에 따라 `ItemReader` 필드 콘텐츠가 달라집니다. 예를 들어 데이터 세트가 워크플로의 이전 단계에서 전달된 JSON 배열이면 `ItemReader` 필드는 생략됩니다. 데이터 세트가 Amazon S3 데이터 소스이면 이 필드에는 다음 하위 필드가 포함됩니다.

`ReaderConfig`

다음 세부 정보를 지정하는 JSON 객체:

- `InputType`

  CSV 파일, 객체, JSON 파일 또는 Amazon S3 인벤토리 목록과 같은 Amazon S3 데이터 소스 유형을 지정합니다. Workflow Studio의 항목 소스 필드 아래에 있는 Amazon S3 항목 소스 드롭다운 목록에서 입력 유형을 선택할 수 있습니다.

- `CSVHeaderLocation`

> **Note**
>
> CSV 파일을 데이터 세트로 사용하는 경우에만 이 필드를 지정해야 합니다.

다음 값 중 하나를 수락하여 열 헤더 위치를 지정합니다.

> **Important**
>
> 현재 Step Functions는 CSV 헤더를 최대 10KB까지 지원합니다.

- `FIRST_ROW` - 파일의 첫 번째 줄이 헤더이면 이 옵션을 사용합니다.
- `GIVEN` - 상태 시스템 정의 내에 헤더를 지정하려면 이 옵션을 사용합니다. 예를 들어 CSV 파일에는 다음 데이터가 포함됩니다.

```
1,307,3.5,1256677221
1,481,3.5,1256677456
1,1091,1.5,1256677471
...
```

다음 JSON 배열을 CSV 헤더로 제공합니다.

```
"ItemReader": {
    "ReaderConfig": {
        "InputType": "CSV",
        "CSVHeaderLocation": "GIVEN",
        "CSVHeaders": [
            "userId",
            "movieId",
            "rating",
            "timestamp"
        ]
    }
}
```

> **Tip**
>
> Workflow Studio의 항목 소스 필드에 있는 추가 구성에서 이 옵션을 찾을 수 있습니다.

- `MaxItems`

  `Map` 상태로 전달되는 데이터 항목 수를 제한합니다. 예를 들어 행 1,000개가 포함된 CSV 파일을 제공하고 한도를 100개로 지정한다고 가정해보겠습니다. 그러면 인터프리터는 행 100개만 `Map` 상태에 전달합니다. `Map` 상태는 헤더 행 다음부터 순차적으로 항목을 처리합니다.

  기본적으로 `Map` 상태는 지정된 데이터 세트의 모든 항목을 반복합니다.

  > **Note**
  >
  > 현재는 한도를 최대 100,000,000개까지 지정할 수 있습니다. Distributed Map 상태는 이 한도를 초과하는 항목을 읽지 않습니다.

  > **Tip**
  >
  > Workflow Studio의 항목 소스 필드에 있는 추가 구성에서 이 옵션을 찾을 수 있습니다.

  또는 Distributed Map 상태 입력에서 기존 키-값 페어의 참조 경로를 지정할 수 있습니다. 이 경로는 양의 정수로 확인되어야 합니다. `MaxItemsPath` 하위 필드에 참조 경로를 지정합니다.

  > **Important**
  >
  > `MaxItems` 또는 `MaxItemsPath` 하위 필드를 지정할 수 있지만 둘 다 함께 지정할 수는 없습니다.

`Resource`

지정된 데이터 세트에 따라 Step Functions에서 간접적으로 호출해야 하는 Amazon S3 API 작업입니다.

`Parameters`

데이터 세트가 저장되는 Amazon S3 버킷 이름과 객체 키를 지정하는 JSON 객체입니다.

> **Important**
>
> Amazon S3 버킷이 상태 시스템과 동일한 AWS 계정 및 AWS 리전에 있는지 확인하세요.

데이터 세트 예제

다음 옵션 중 하나를 데이터 세트로 지정할 수 있습니다.

- 이전 단계의 JSON 배열
- Amazon S3 객체 목록
- Amazon S3 버킷에 있는 JSON 파일
- Amazon S3 버킷에 있는 CSV 파일
- Amazon S3 인벤토리 목록

> **Important**
>
> Step Functions에는 사용하는 Amazon S3 데이터 세트에 액세스할 수 있는 적절한 권한이 필요합니다. 데이터 세트에 대한 IAM 정책은 데이터 세트에 대한 IAM 정책을 참조하세요.

이전 단계의 JSON 배열

Distributed Map 상태는 워크플로의 이전 단계에서 전달된 JSON 입력을 허용할 수 있습니다. 이 입력은 배열이거나 특정 노드 내의 배열을 포함해야 합니다. 배열이 포함된 노드를 선택하려면 `ItemsPath` 필드를 사용하면 됩니다.

배열의 개별 항목을 처리하기 위해 Distributed Map 상태는 배열 항목마다 하위 워크플로 실행을 시작합니다. 다음 탭에서는 `Map` 상태에 전달된 입력과 하위 워크플로 실행에 대한 해당 입력의 예제를 보여줍니다.

> **Note**
>
> 데이터 세트가 이전 단계의 JSON 배열이면 Step Functions에서 `ItemReader` 필드를 생략합니다.

Input passed to the Map state

항목 3개로 구성된 다음 JSON 배열을 생각해보세요.

```
"facts": [
    {
        "verdict": "true",
        "statement_date": "6/11/2008",
        "statement_source": "speech"
    },
    {
        "verdict": "false",
        "statement_date": "6/7/2022",
        "statement_source": "television"
    },
    {
        "verdict": "mostly-true",
        "statement_date": "5/18/2016",
        "statement_source": "news"
    }
]
```

Input passed to a child workflow execution

Distributed Map 상태는 하위 워크플로 실행 3개를 시작합니다. 각 실행은 배열 항목을 입력으로 수신합니다. 다음 예제에서는 하위 워크플로 실행에서 수신한 입력을 보여줍니다.

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

Amazon S3 객체 예제

Distributed Map 상태는 Amazon S3 버킷에 저장된 객체를 반복할 수 있습니다. 워크플로 실행이 `Map` 상태에 도달하면 Step Functions는 ListObjectsV2 API 작업을 간접적으로 호출하여 Amazon S3 객체 메타데이터 배열을 반환합니다. 이 배열의 각 항목에는 버킷에 저장된 데이터의 데이터(예: ETag 및 Key)가 포함됩니다.

배열의 개별 항목을 처리하기 위해 Distributed Map 상태는 하위 워크플로 실행을 시작합니다. 예를 들어 Amazon S3 버킷에 이미지 100개가 포함되어 있다고 가정해보겠습니다. 그러면 `ListObjectsV2` API 작업을 간접적으로 호출한 후에 반환된 배열에는 항목 100개가 포함됩니다. 그런 다음 Distributed Map 상태는 하위 워크플로 실행 100개를 시작하여 각 배열 항목을 처리합니다.

> **Note**
>
> - 현재 Step Functions에는 Amazon S3 콘솔을 사용하여 특정 Amazon S3 버킷에 만든 각 폴더에 대한 항목도 포함되어 있습니다. 이로 인해 Distributed Map 상태에서 추가 하위 워크플로 실행을 시작합니다. 폴더에 추가 하위 워크플로 실행을 만들지 않으려면 AWS CLI를 사용하여 폴더를 만드는 것이 좋습니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 Amazon S3 상위 수준 명령을 참조하세요.
> - Step Functions에는 사용하는 Amazon S3 데이터 세트에 액세스할 수 있는 적절한 권한이 필요합니다. 데이터 세트에 대한 IAM 정책은 데이터 세트에 대한 IAM 정책을 참조하세요.

다음 탭에서는 이 데이터 세트의 하위 워크플로 실행에 전달된 `ItemReader` 필드 구문과 입력의 예제를 보여줍니다.

ItemReader syntax

이 예시에서는 `myBucket`이라는 Amazon S3 버킷의 `processData`라는 접두사 내에 이미지, JSON 파일 및 객체가 포함된 데이터를 구성했습니다.

```
"ItemReader": {
    "Resource": "arn:aws:states:::s3:listObjectsV2",
    "Parameters": {
        "Bucket": "myBucket",
        "Prefix": "processData"
    }
}
```

Input passed to a child workflow execution

Distributed Map 상태는 Amazon S3 버킷에 있는 항목 수만큼 하위 워크플로 실행을 시작합니다. 다음 예제에서는 하위 워크플로 실행에서 수신한 입력을 보여줍니다.

```
{
  "Etag": "\"05704fbdccb224cb01c59005bebbad28\"",
```

```
  "Key": "processData/images/n02085620_1073.jpg",
  "LastModified": 1668699881,
  "Size": 34910,
  "StorageClass": "STANDARD"
}
```

Amazon S3 버킷에 있는 JSON 파일

Distributed Map 상태는 Amazon S3 버킷에 저장된 JSON 파일을 데이터 세트로 허용할 수 있습니다. JSON 파일에는 배열이 포함되어야 합니다.

워크플로 실행이 `Map` 상태에 도달하면 Step Functions는 GetObject API 작업을 간접적으로 호출하여 지정된 JSON 파일을 가져옵니다. 그러면 `Map` 상태가 배열의 각 항목을 반복하고 항목마다 하위 워크플로 실행을 시작합니다. 예를 들어 JSON 파일에 배열 항목 1,000개가 포함된 경우 `Map` 상태는 하위 워크플로 실행 1,000개를 시작합니다.

> **Note**
>
> - 하위 워크플로 실행을 시작하는 데 사용되는 실행 입력은 256KB를 초과할 수 없습니다. 하지만 항목 크기를 줄이기 위해 선택적 `ItemSelector` 필드를 적용한 경우 Step Functions는 CSV 또는 JSON 파일에서 항목을 최대 8MB까지 읽을 수 있습니다.
> - 현재 Step Functions는 Amazon S3 인벤토리 보고서에서 개별 파일 최대 크기로 10GB를 지원합니다. 하지만 Step Functions는 각 개별 파일이 10GB 미만이면 10GB 넘게 처리할 수 있습니다.
> - Step Functions에는 사용하는 Amazon S3 데이터 세트에 액세스할 수 있는 적절한 권한이 필요합니다. 데이터 세트에 대한 IAM 정책은 데이터 세트에 대한 IAM 정책을 참조하세요.

다음 탭에서는 이 데이터 세트의 하위 워크플로 실행에 전달된 `ItemReader` 필드 구문과 입력의 예제를 보여줍니다.

이 예제의 경우 *factcheck.json*이라는 JSON 파일이 있다고 가정해보겠습니다. 이 파일은 Amazon S3 버킷의 *jsonDataset* 접두사 내에 저장되었습니다. 다음은 JSON 데이터 세트의 예제입니다.

```
[
  {
    "verdict": "true",
    "statement_date": "6/11/2008",
```

```
    "statement_source": "speech"
  },
  {
    "verdict": "false",
    "statement_date": "6/7/2022",
    "statement_source": "television"
  },
  {
    "verdict": "mostly-true",
    "statement_date": "5/18/2016",
    "statement_source": "news"
  },
  ...
]
```

ItemReader syntax

```
"ItemReader": {
    "Resource": "arn:aws:states:::s3:getObject",
    "ReaderConfig": {
        "InputType": "JSON"
    },
    "Parameters": {
        "Bucket": "myBucket",
        "Key": "jsonDataset/factcheck.json"
    }
}
```

Input to a child workflow execution

Distributed Map 상태는 JSON 파일에 있는 배열 항목 수만큼 하위 워크플로 실행을 시작합니다. 다음 예제에서는 하위 워크플로 실행에서 수신한 입력을 보여줍니다.

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

Amazon S3 버킷에 있는 CSV 파일

Distributed Map 상태는 Amazon S3 버킷에 저장된 CSV 파일을 데이터 세트로 허용할 수 있습니다. CSV 파일을 데이터 세트로 사용하는 경우에는 CSV 열 헤더를 지정해야 합니다. CSV 헤더를 지정하는 방법은 ItemReader 필드 콘텐츠를 참조하세요.

CSV 파일에서 데이터를 만들고 유지하기 위한 표준화된 형식이 없으므로 Step Functions는 다음 규칙에 따라 CSV 파일을 파싱합니다.

- 쉼표(,)는 개별 필드를 구분하는 구분 기호입니다.
- 줄 바꿈은 개별 레코드를 구분하는 구분 기호입니다.
- 필드는 문자열로 취급됩니다. 데이터 유형 변환의 경우 ItemSelector에서 `States.StringToJson` 내장 함수를 사용합니다.
- 문자열을 묶을 때 큰따옴표(“ ”)는 필요하지 않습니다. 그러나 큰따옴표로 묶인 문자열에는 구분 기호로 작동하지 않는 쉼표와 줄 바꿈이 포함될 수 있습니다.
- 큰따옴표를 반복해서 사용하지 마세요.
- 행의 필드 수가 헤더의 필드 수보다 적으면 Step Functions에서 누락된 값에 빈 문자열을 제공합니다.
- 행의 필드 수가 헤더의 필드 수보다 많으면 Step Functions는 추가 필드를 건너뜁니다.

Step Functions에서 CSV 파일을 파싱하는 방법에 대한 자세한 내용은 Example of parsing an input CSV file를 참조하세요.

워크플로 실행이 `Map` 상태에 도달하면 Step Functions는 GetObject API 작업을 간접적으로 호출하여 지정된 CSV 파일을 가져옵니다. 그러면 `Map` 상태가 CSV 파일의 각 행을 반복하고 하위 워크플로 실행을 시작하여 각 행의 항목을 처리합니다. 예를 들어 입력으로 행 100개가 포함된 CSV 파일을 제공한다고 가정해보겠습니다. 그러면 인터프리터에서 각 행을 `Map` 상태에 전달합니다. `Map` 상태는 헤더 행 다음부터 순차적으로 항목을 처리합니다.

> **Note**
>
> - 하위 워크플로 실행을 시작하는 데 사용되는 실행 입력은 256KB를 초과할 수 없습니다. 하지만 항목 크기를 줄이기 위해 선택적 `ItemSelector` 필드를 적용한 경우 Step Functions는 CSV 또는 JSON 파일에서 항목을 최대 8MB까지 읽을 수 있습니다.
> - 현재 Step Functions는 Amazon S3 인벤토리 보고서에서 개별 파일 최대 크기로 10GB를 지원합니다. 하지만 Step Functions는 각 개별 파일이 10GB 미만이면 10GB 넘게 처리할 수 있습니다.

- Step Functions에는 사용하는 Amazon S3 데이터 세트에 액세스할 수 있는 적절한 권한이 필요합니다. 데이터 세트에 대한 IAM 정책은 데이터 세트에 대한 IAM 정책을 참조하세요.

다음 탭에서는 이 데이터 세트의 하위 워크플로 실행에 전달된 `ItemReader` 필드 구문과 입력의 예제를 보여줍니다.

ItemReader syntax

예를 들어 라는 *`ratings.csv`*라는 CSV 파일이 있다고 가정해보겠습니다. 그런 다음 Amazon S3 버킷의 *`csvDataset`* 접두사 내에 이 파일을 저장했습니다.

```
{
  "ItemReader": {
    "ReaderConfig": {
      "InputType": "CSV",
      "CSVHeaderLocation": "FIRST_ROW"
    },
    "Resource": "arn:aws:states:::s3:getObject",
    "Parameters": {
      "Bucket": "myBucket",
      "Key": "csvDataset/ratings.csv"
    }
  }
}
```

Input to a child workflow execution

Distributed Map 상태는 헤더 행을 제외하고(파일에 있는 경우) CSV 파일에 있는 행 수만큼 하위 워크플로 실행을 시작합니다. 다음 예제에서는 하위 워크플로 실행에서 수신한 입력을 보여줍니다.

```
{
  "rating": "3.5",
  "movieId": "307",
  "userId": "1",
  "timestamp": "1256677221"
}
```

S3 인벤토리 예제

Distributed Map 상태는 Amazon S3 버킷에 저장된Amazon S3 인벤토리 매니페스트 파일을 데이터 세트로 허용할 수 있습니다.

워크플로 실행이 `Map` 상태에 도달하면 Step Functions는 GetObject API 작업을 간접적으로 호출하여 지정된 Amazon S3 인벤토리 매니페스트 파일을 가져옵니다. 그런 다음 `Map` 상태는 인벤토리의 객체를 반복하여 Amazon S3 인벤토리 객체 메타데이터 배열을 반환합니다.

> **Note**
>
> - 현재 Step Functions는 Amazon S3 인벤토리 보고서에서 개별 파일 최대 크기로 10GB를 지원합니다. 하지만 Step Functions는 각 개별 파일이 10GB 미만이면 10GB 넘게 처리할 수 있습니다.
> - Step Functions에는 사용하는 Amazon S3 데이터 세트에 액세스할 수 있는 적절한 권한이 필요합니다. 데이터 세트에 대한 IAM 정책은 데이터 세트에 대한 IAM 정책을 참조하세요.

다음은 CSV 형식의 인벤토리 파일에 대한 예제입니다. 이 파일에는 `sourceBucket`이라는 Amazon S3 버킷에 저장되어 있는 `csvDataset` 및 `imageDataset` 객체가 포함되어 있습니다.

```
"sourceBucket","csvDataset/","0","2022-11-16T00:27:19.000Z"
"sourceBucket","csvDataset/titles.csv","3399671","2022-11-16T00:29:32.000Z"
"sourceBucket","imageDataset/","0","2022-11-15T20:00:44.000Z"
"sourceBucket","imageDataset/n02085620_10074.jpg","27034","2022-11-15T20:02:16.000Z"
...
```

> **Important**
>
> 현재 Step Functions는 사용자 정의 Amazon S3 인벤토리 보고서를 데이터 세트로 지원하지 않습니다. 또한 Amazon S3 인벤토리 보고서의 출력 형식이 CSV인지 확인해야 합니다. Amazon S3 인벤토리 및 설정 방법에 대한 자세한 내용은 Amazon S3 사용 설명서의 Amazon S3 인벤토리를 참조하세요.

다음 인벤토리 매니페스트 파일 예제에서는 인벤토리 객체 메타데이터의 CSV 헤더를 보여줍니다.

```
{
```

```
  "sourceBucket" : "sourceBucket",
  "destinationBucket" : "arn:aws:s3:::inventory",
  "version" : "2016-11-30",
  "creationTimestamp" : "1668560400000",
  "fileFormat" : "CSV",
  "fileSchema" : "Bucket, Key, Size, LastModifiedDate",
  "files" : [ {
    "key" : "source-bucket/destination-prefix/
data/20e55de8-9c21-45d4-99b9-46c732000228.csv.gz",
    "size" : 7300,
    "MD5checksum" : "a7ff4a1d4164c3cd55851055ec8f6b20"
  } ]
}
```

다음 탭에서는 이 데이터 세트의 하위 워크플로 실행에 전달된 ItemReader 필드 구문과 입력의 예제를 보여줍니다.

ItemReader syntax

```
{
  "ItemReader": {
    "ReaderConfig": {
      "InputType": "MANIFEST"
    },
    "Resource": "arn:aws:states:::s3:getObject",
    "Parameters": {
      "Bucket": "destinationBucket",
      "Key": "destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-MMZ/
manifest.json"
    }
  }
}
```

Input to a child workflow execution

```
{
  "LastModifiedDate": "2022-11-16T00:29:32.000Z",
  "Bucket": "sourceBucket",
  "Size": "3399671",
  "Key": "csvDataset/titles.csv"
}
```

Amazon S3 인벤토리 보고서를 구성하는 동안에 선택한 필드에 따라 `manifest.json` 파일 콘텐츠는 표시된 예제와 다를 수 있습니다.

데이터 세트에 대한 IAM 정책

Step Functions 콘솔을 사용하여 워크플로를 만들면 Step Functions에서 워크플로 정의의 리소스를 기반으로 IAM 정책을 자동으로 생성할 수 있습니다. 이러한 정책에는 상태 시스템 역할에서 Distributed Map 상태에 대한 `StartExecution` API 작업을 간접적으로 호출하도록 허용하는 데 필요한 최소 권한이 포함되어 있습니다. 또한 이러한 정책에는 Step Functions에서 Amazon S3 버킷과 객체, Lambda 함수와 같은 AWS 리소스에 액세스하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 권한만 포함하는 것이 좋습니다. 예를 들어 워크플로에 분산 모드의 `Map` 상태가 포함된 경우 정책 범위를 데이터 세트가 포함된 특정 Amazon S3 버킷과 폴더로 좁힙니다.

> **Important**
>
> Distributed Map 상태 입력에 있는 기존 키-값 페어에 대한 참조 경로를 사용하여 Amazon S3 버킷과 객체 또는 접두사를 지정하는 경우 워크플로에 대한 IAM 정책을 업데이트해야 합니다. 정책 범위를 런타임 시 경로에서 확인하는 버킷과 객체 이름으로 좁히세요.

다음 IAM 정책 예제에서는 ListObjectsV2 및 GetObject API 작업을 사용하여 Amazon S3 데이터 세트에 액세스하는 데 필요한 최소 권한을 부여합니다.

Example Amazon S3 객체를 데이터 세트로 사용하는 IAM 정책

다음 예제에서는 *myBucket*이라는 Amazon S3 버킷의 *processImages* 내에 구성된 객체에 액세스할 수 있는 최소 권한을 부여하는 IAM 정책을 보여줍니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket"
            ],
            "Condition": {
```

```
                "StringLike": {
                    "s3:prefix": [
                        "processImages"
                    ]
                }
            }
        }
    ]
}
```

Example CSV 파일을 데이터 세트로 사용하는 IAM 정책

다음 예제에서는 *ratings.csv*라는 CSV 파일에 액세스할 수 있는 최소 권한을 부여하는 IAM 정책을 보여줍니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/csvDataset/ratings.csv"
            ]
        }
    ]
}
```

Example Amazon S3 인벤토리를 데이터 세트로 사용하는 IAM 정책

다음 예제에서는 Amazon S3 인벤토리 보고서에 액세스할 수 있는 최소 권한을 부여하는 IAM 정책을 보여줍니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-
MMZ/manifest.json",
                "arn:aws:s3:::destination-prefix/source-bucket/config-ID/data/*"
            ]
        }
    ]
}
```

ItemsPath

`ItemsPath` 필드를 사용하여 `Map` 상태에 제공된 JSON 입력 내에서 배열을 선택합니다. `Map` 상태는 배열의 항목마다 일련의 단계를 반복합니다. 기본적으로 `Map` 상태는 `ItemsPath`를 전체 입력을 선택하는 `$`로 설정합니다. `Map` 상태에 대한 입력이 JSON 배열이면 배열의 각 항목을 반복에 입력으로 전달하면서 해당 항목마다 반복을 실행합니다.

> **Note**
>
> 워크플로의 이전 상태에서 전달된 JSON 입력을 사용하는 경우에만 Distributed Map 상태에서 `ItemsPath`를 사용할 수 있습니다.

`ItemsPath` 필드를 사용하여 반복에 사용된 JSON 배열을 가리키는 입력의 위치를 지정할 수 있습니다. `ItemsPath` 값은 참조 경로여야 하고 해당 경로는 JSON 배열을 가리켜야 합니다. 예를 들어 다음과 같은 두 개의 배열이 포함된 `Map` 상태를 가정해 보겠습니다.

```
{
  "ThingsPiratesSay": [
    {
      "say": "Avast!"
    },
    {
      "say": "Yar!"
    },
    {
      "say": "Walk the Plank!"
    }
  ],
  "ThingsGiantsSay": [
    {
```

```
      "say": "Fee!"
    },
    {
      "say": "Fi!"
    },
    {
      "say": "Fo!"
    },
    {
      "say": "Fum!"
    }
  ]
}
```

이 경우 ItemsPath로 배열을 선택하여 Map 상태 반복에 사용할 배열을 지정할 수 있습니다. 다음 상태 시스템 정의에서는 ItemsPath를 사용하여 입력에서 ThingsPiratesSay 배열을 지정합니다. 그런 다음 ThingsPiratesSay 배열의 항목마다 SayWord Pass 상태 반복을 실행합니다.

```
{
  "StartAt": "PiratesSay",
  "States": {
    "PiratesSay": {
      "Type": "Map",
      "ItemsPath": "$.ThingsPiratesSay",
      "ItemProcessor": {
         "StartAt": "SayWord",
         "States": {
           "SayWord": {
             "Type": "Pass",
             "End": true
           }
         }
      },
      "End": true
    }
  }
}
```

입력을 처리할 때는 Map 상태는 InputPath 다음에 ItemsPath를 적용합니다. InputPath에서 입력을 필터링한 후에 상태에 유효한 입력에서 작동합니다.

Map 상태에 대한 자세한 내용은 다음을 참조하십시오.

- Map 상태
- Map 상태 처리 모드
- Inline Map 상태를 사용하여 작업 반복
- Inline Map 상태 입력 및 출력 처리

ItemSelector

기본적으로 Map 상태의 유효 입력은 원시 상태 입력에 있는 개별 데이터 항목 집합입니다. 이 ItemSelector 필드를 사용하면 데이터 항목 값이 Map 상태로 전달되기 전에 이 값을 재정의할 수 있습니다. 값을 재정의하려면 키-값 페어 컬렉션이 포함된 유효한 JSON 입력을 지정합니다. 이러한 페어는 상태 시스템 정의에 제공된 정적 값, 경로를 사용하여 상태 입력에서 선택한 값 또는 컨텍스트 객체에서 액세스한 값일 수 있습니다.

경로나 컨텍스트 객체를 사용하여 키-값 페어를 지정하는 경우 키 이름은 .$로 끝나야 합니다.

> **Note**
>
> ItemSelector 필드는 Map 상태 내에서 Parameters 필드를 대체합니다. Map 상태 정의의 Parameters 필드를 사용하여 사용자 지정 입력을 만드는 경우 ItemSelector으로 바꾸는 것이 좋습니다.

Inline Map 상태와 Distributed Map 상태 모두에서 ItemSelector 필드를 지정할 수 있습니다.

예를 들어 imageData 노드 내에 항목 3개로 구성된 배열을 포함하는 다음 JSON 입력을 고려해보세요. *Map* 상태 반복마다 배열 항목이 입력으로 반복에 전달됩니다.

```
[
  {
    "resize": "true",
    "format": "jpg"
  },
  {
    "resize": "false",
    "format": "png"
  },
  {
    "resize": "true",
    "format": "jpg"
```

```
  }
]
```

다음 예제와 같이 `ItemSelector` 필드를 사용하여 사용자 지정 JSON 입력을 정의하여 원래 입력을 재정의할 수 있습니다. 그러면 Step Functions에서 이 사용자 지정 입력을 각 *Map* 상태 반복에 전달합니다. 사용자 지정 입력에는 `size` 정적 값과 `Map` 상태에 대한 컨텍스트 객체 데이터 값이 포함됩니다. `$$.Map.Item.Value` 컨텍스트 객체에는 각 개별 데이터 항목 값이 포함됩니다.

```
{
  "ItemSelector": {
    "size": 10,
    "value.$": "$$.Map.Item.Value"
  }
}
```

다음 예제에서는 Inline Map 상태를 한 번 반복하면 수신되는 입력을 보여줍니다.

```
{
  "size": 10,
  "value": {
    "resize": "true",
    "format": "jpg"
  }
}
```

> **Tip**
>
> `ItemSelector` 필드를 사용하는 Distributed Map 상태의 전체 예제는 Distributed Map 상태를 사용하여 시작하기를 참조하세요.

ItemBatcher

`ItemBatcher` 필드는 단일 하위 워크플로 실행에서 항목 그룹을 처리하도록 지정하는 JSON 개체입니다. 대용량 CSV 파일이나 JSON 배열 또는 대규모 Amazon S3 객체 집합을 처리할 때 일괄 처리를 사용합니다.

다음 예제에서는 `ItemBatcher` 필드 구문을 보여줍니다. 다음 구문에서 각 하위 워크플로 실행에서 처리해야 하는 최대 항목 수는 100으로 설정됩니다.

```
{
  "ItemBatcher": {
    "MaxItemsPerBatch": 100
  }
}
```

기본적으로 데이터 세트의 각 항목은 입력으로 개별 하위 워크플로 실행에 전달됩니다. 예를 들어 다음 배열이 포함된 JSON 파일을 입력으로 지정한다고 가정해보겠습니다.

```
[
  {
    "verdict": "true",
    "statement_date": "6/11/2008",
    "statement_source": "speech"
  },
  {
    "verdict": "false",
    "statement_date": "6/7/2022",
    "statement_source": "television"
  },
  {
    "verdict": "true",
    "statement_date": "5/18/2016",
    "statement_source": "news"
  },
  ...
]
```

지정된 입력의 경우 각 하위 워크플로 실행은 배열 항목을 입력으로 수신합니다. 다음은 하위 워크플로 실행 입력을 보여주는 예제입니다.

```
{
  "verdict": "true",
  "statement_date": "6/11/2008",
  "statement_source": "speech"
}
```

처리 작업의 성능과 비용을 최적화하려면 항목 수와 항목 처리 시간의 균형을 맞추는 배치 크기를 선택합니다. 일괄 처리를 사용하면 Step Functions에서 항목을 항목 배열에 추가합니다. 그런 다음 배열을 입력으로 각 하위 워크플로 실행에 전달합니다. 다음은 하위 워크플로 실행에 입력으로 전달된 두 항목의 배치를 보여주는 예제입니다.

```
{
  "Items": [
    {
      "verdict": "true",
      "statement_date": "6/11/2008",
      "statement_source": "speech"
    },
    {
      "verdict": "false",
      "statement_date": "6/7/2022",
      "statement_source": "television"
    }
  ]
}
```

Tip

워크플로에서 `ItemBatcher` 필드를 사용하는 방법에 대해 자세히 알아보려면 다음 자습서와 워크숍을 사용해보세요.

- Lambda 함수 내에서 전체 데이터 배치 처리
- 하위 워크플로 실행 내에서 일괄적으로 항목 반복
- AWS Step Functions 워크숍의 모듈 14 - 데이터 처리에서 분산 맵을 사용한 대규모 병렬화

목차

항목 일괄 처리를 지정하는 필드

항목을 일괄 처리하려면 일괄 처리할 최대 항목 수, 최대 배치 크기 또는 둘 다를 지정합니다. 항목을 일괄 처리하려면 다음 값 중 하나를 지정해야 합니다.

배치당 최대 항목 수

각 하위 워크플로 실행에서 처리하는 최대 항목 수를 지정합니다. 인터프리터는 `Items` 배열에서 일괄 처리되는 항목 수를 이 값으로 제한합니다. 배치 번호와 크기 모두 지정하면 인터프리터는 지정된 배치 크기 한도가 초과되지 않도록 배치의 항목 수를 줄입니다.

이 값을 지정하지 않고 최대 배치 크기 값을 제공하면 Step Functions는 최대 배치 크기(바이트 단위)를 초과하지 않고 각 하위 워크플로 실행에서 최대한 많은 항목을 처리합니다.

예를 들어 노드 1130개가 포함된 입력 JSON 파일을 사용하여 실행을 실행한다고 가정해보겠습니다. 각 배치의 최대 항목 값을 100으로 지정하면 Step Functions에서 배치 12개를 만듭니다. 이 중 배치 11개 각각에는 항목 100개가 포함되고 12번째 배치에는 잔여 항목 30개가 포함됩니다.

또는 Distributed Map 상태 입력의 기존 키-값 페어에 대한 참조 경로로 각 배치의 최대 항목 수를 지정할 수 있습니다. 이 경로는 양의 정수로 확인되어야 합니다.

다음 입력을 예로 들어보겠습니다.

```
{
  "maxBatchItems": 500
}
```

다음과 같이 참조 경로를 사용하여 일괄 처리할 최대 항목 수를 지정할 수 있습니다.

```
{
  ...
  "Map": {
    "Type": "Map",
    "MaxConcurrency": 2000,
    "ItemBatcher": {
      "MaxItemsPerBatchPath": "$.maxBatchItems"
    }
    ...
    ...
  }
}
```

> **Important**
>
> `MaxItemsPerBatch` 또는 `MaxItemsPerBatchPath` 하위 필드를 지정할 수 있지만 둘 다 함께 지정할 수는 없습니다.

배치당 최대 KB

최대 배치 크기(바이트 단위)를 최대 256KB까지 지정합니다. 최대 배치 수와 크기 모두 지정하면 Step Functions는 지정된 배치 크기 한도가 초과되지 않도록 배치의 항목 수를 줄입니다.

또는 Distributed Map 상태 입력의 기존 키-값 페어에 대한 참조 경로로 최대 배치 크기를 지정할 수 있습니다. 이 경로는 양의 정수로 확인되어야 합니다.

> **Note**
>
> 일괄 처리를 사용하지만 최대 배치 크기를 지정하지 않으면 인터프리터는 각 하위 워크플로 실행에서 최대 256KB까지 처리할 수 있는 항목 수를 모두 처리합니다.

다음 입력을 예로 들어보겠습니다.

```
{
  "batchSize": 131072
}
```

다음과 같이 참조 경로를 사용하여 최대 배치 크기를 지정할 수 있습니다.

```
{
  ...
  "Map": {
    "Type": "Map",
    "MaxConcurrency": 2000,
    "ItemBatcher": {
      "MaxInputBytesPerBatchPath": "$.batchSize"
    }
    ...
    ...
  }
}
```

> **Important**
>
> `MaxInputBytesPerBatch` 또는 `MaxInputBytesPerBatchPath` 하위 필드를 지정할 수 있지만 둘 다 함께 지정할 수는 없습니다.

배치 입력

필요한 경우 각 하위 워크플로 실행에 전달되는 각 배치에 포함할 고정 JSON 입력을 지정할 수도 있습니다. Step Functions는 이 입력을 각 개별 하위 워크플로 실행의 입력과 병합합니다. 항목 배열의 사실 확인 날짜 고정 입력을 예로 들어 보겠습니다.

```
"ItemBatcher": {
    "BatchInput": {
        "factCheck": "December 2022"
    }
}
```

각 하위 워크플로 실행에서 입력으로 다음을 수신합니다.

```
{
  "BatchInput": {
    "factCheck": "December 2022"
  },
  "Items": [
    {
      "verdict": "true",
      "statement_date": "6/11/2008",
      "statement_source": "speech"
    },
    {
      "verdict": "false",
      "statement_date": "6/7/2022",
      "statement_source": "television"
    },
    ...
  ]
}
```

ResultWriter

`ResultWriter` 필드는 Step Functions가 Distributed Map 상태에서 시작한 하위 워크플로 실행 결과를 기록하는 Amazon S3 위치를 지정하는 JSON 객체입니다. 기본적으로 Step Functions는 이러한 결과를 내보내지 않습니다.

> **Important**
>
> 맵 실행 결과를 내보내는 데 사용하는 Amazon S3 버킷이 상태 시스템과 동일한 AWS 계정 및 AWS 리전에 있는지 확인하세요. 그렇지 않으면 상태 시스템 실행이 실패하고 `States.ResultWriterFailed` 오류가 표시됩니다.

출력 페이로드 크기가 256KB를 초과하는 경우 Amazon S3 버킷에 결과를 내보내는 것이 도움이 됩니다. Step Functions는 실행 입력 및 출력, ARN, 실행 상태와 같은 모든 하위 워크플로 실행 데이터를 통합합니다. 그런 다음 같은 상태의 실행을 지정된 Amazon S3 위치에 있는 각 파일로 내보냅니다. 다음 예제에서는 하위 워크플로 실행 결과를 내보내는 경우의 `ResultWriter` 필드 구문을 보여줍니다. 이 예제에서는 `csvProcessJobs` 접두사 내 `myOutputBucket`이라는 버킷에 결과를 저장합니다.

```
{
  "ResultWriter": {
    "Resource": "arn:aws:states:::s3:putObject",
    "Parameters": {
      "Bucket": "myOutputBucket",
      "Prefix": "csvProcessJobs"
    }
  }
}
```

> **Tip**
>
> Workflow Studio에서 Amazon S3로 Map 상태 결과 내보내기를 선택하여 하위 워크플로 실행 결과를 내보낼 수 있습니다. 그런 다음 결과를 내보내려는 Amazon S3 버킷의 이름과 접두사를 입력합니다.

Step Functions에는 결과를 내보내려는 버킷과 폴더에 액세스할 수 있는 적절한 권한이 필요합니다. 필요한 IAM 정책에 대한 자세한 내용은 ResultWriter에 대한 IAM 정책을 참조하세요.

하위 워크플로 실행 결과를 내보내면 Distributed Map 상태 실행에서 맵 실행 ARN과 Amazon S3 내보내기 위치에 대한 데이터를 다음 형식으로 반환합니다.

```
{
  "MapRunArn": "arn:aws:states:us-
east-2:123456789012:mapRun:csvProcess/Map:ad9b5f27-090b-3ac6-9beb-243cd77144a7",
```

```
  "ResultWriterDetails": {
    "Bucket": "myOutputBucket",
    "Key": "csvProcessJobs/ad9b5f27-090b-3ac6-9beb-243cd77144a7/manifest.json"
  }
}
```

Step Functions는 동일한 상태의 실행을 해당 파일로 각각 내보냅니다. 예를 들어 하위 워크플로 실행 실행에서 성공이 500개, 실패가 200개인 경우 Step Functions는 지정된 Amazon S3 위치에 성공 및 실패 결과에 대한 파일 2개를 만듭니다. 이 예제에서 성공 결과 파일에는 성공 결과 500개가 포함되고 실패 결과 파일에는 실패 결과 200개가 포함됩니다.

지정된 실행 시도의 경우 Step Functions는 실행 출력에 따라 지정된 Amazon S3 위치에 다음 파일을 만듭니다.

- `manifest.json` - 내보내기 위치, 맵 실행 ARN, 결과 파일에 대한 정보와 같은 맵 실행 메타데이터를 포함합니다.

  맵 실행을 redriven한 경우 `manifest.json` 파일에는 맵 실행의 모든 시도에서 성공한 모든 하위 워크플로 실행에 대한 참조가 포함됩니다. 하지만 이 파일에는 특정 redrive의 실패 및 보류 중인 실행에 대한 참조가 포함됩니다.
- `SUCCEEDED_n.json` - 모든 성공적인 하위 워크플로 실행에 대한 통합 데이터를 포함합니다. n은 파일의 인덱스 번호를 나타냅니다. 인덱스 번호는 0부터 시작합니다. 예: `SUCCEEDED_1.json`.
- `FAILED_n.json` - 모든 실패, 제한 시간 및 중단된 하위 워크플로 실행에 대한 통합 데이터를 포함합니다. 실패한 실행을 복구하려면 이 파일을 사용합니다. n은 파일 인덱스를 나타냅니다. 인덱스 번호는 0부터 시작합니다. 예: `FAILED_1.json`.
- `PENDING_n.json` - 맵 실행이 실패하거나 중단되어 시작되지 않은 모든 하위 워크플로 실행에 대한 통합 데이터를 포함합니다. n은 파일 인덱스를 나타냅니다. 인덱스 번호는 0부터 시작합니다. 예: `PENDING_1.json`.

Step Functions는 개별 결과 파일을 최대 5GB까지 지원합니다. 파일 크기가 5GB를 초과하면 Step Functions에서 나머지 실행 결과를 기록할 다른 파일을 만들고 파일 이름에 인덱스 번호를 추가합니다. 예를 들어 `Succeeded_0.json` 파일 크기가 5GB를 초과하면 Step Functions에서 `Succeeded_1.json` 파일을 만들어 나머지 결과를 기록합니다.

하위 워크플로 실행 결과를 내보내도록 지정하지 않은 경우 상태 시스템 실행은 다음 예제와 같이 하위 워크플로 실행 결과 배열을 반환합니다.

> **Note**
>
> 반환된 출력 크기가 256KB를 초과하면 상태 시스템 실행이 실패하고 `States.DataLimitExceeded` 오류가 반환됩니다.

```
[
  {
    "statusCode": 200,
    "inputReceived": {
      "show_id": "s1",
      "release_year": "2020",
      "rating": "PG-13",
      "type": "Movie"
    }
  },
  {
    "statusCode": 200,
    "inputReceived": {
      "show_id": "s2",
      "release_year": "2021",
      "rating": "TV-MA",
      "type": "TV Show"
    }
  },
  ...
]
```

ResultWriter에 대한 IAM 정책

Step Functions 콘솔을 사용하여 워크플로를 만들면 Step Functions에서 워크플로 정의의 리소스를 기반으로 IAM 정책을 자동으로 생성할 수 있습니다. 이러한 정책에는 상태 시스템 역할에서 Distributed Map 상태에 대한 `StartExecution` API 작업을 간접적으로 호출하도록 허용하는 데 필요한 최소 권한이 포함되어 있습니다. 또한 이러한 정책에는 Step Functions에서 Amazon S3 버킷과 객체, Lambda 함수와 같은 AWS 리소스에 액세스하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 권한만 포함하는 것이 좋습니다. 예를 들어 워크플로에 분산 모드의 Map 상태가 포함된 경우 정책 범위를 데이터 세트가 포함된 특정 Amazon S3 버킷과 폴더로 좁힙니다.

> **Important**
>
> Distributed Map 상태 입력에 있는 기존 키-값 페어에 대한 참조 경로를 사용하여 Amazon S3 버킷과 객체 또는 접두사를 지정하는 경우 워크플로에 대한 IAM 정책을 업데이트해야 합니다. 정책 범위를 런타임 시 경로에서 확인하는 버킷과 객체 이름으로 좁히세요.

다음 IAM 정책 예제에서는 `PutObject` API 작업을 사용하여 하위 워크플로 실행 결과를 Amazon S3 버킷의 *`csvJobs`* 폴더에 쓰는 데 필요한 최소 권한을 부여합니다.

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

하위 워크플로 실행 결과를 기록하는 Amazon S3 버킷이 AWS Key Management Service (AWS KMS) 키를 통해 암호화된 경우 IAM 정책에 필요한 AWS KMS 권한을 포함해야 합니다. 자세한 내용은 AWS KMS key 암호화된 Amazon S3 버킷에 대한 IAM 권한 섹션을 참조하세요.

입력 CSV 파일 파싱

CSV 파일에서 데이터를 만들고 유지하기 위한 표준화된 형식이 없으므로 Step Functions는 다음 규칙에 따라 CSV 파일을 파싱합니다.

- 쉼표(,)는 개별 필드를 구분하는 구분 기호입니다.
- 줄 바꿈은 개별 레코드를 구분하는 구분 기호입니다.

- 필드는 문자열로 취급됩니다. 데이터 유형 변환의 경우 ItemSelector에서 `States.StringToJson` 내장 함수를 사용합니다.
- 문자열을 묶을 때 큰따옴표(“ ”)는 필요하지 않습니다. 그러나 큰따옴표로 묶인 문자열에는 구분 기호로 작동하지 않는 쉼표와 줄 바꿈이 포함될 수 있습니다.
- 큰따옴표를 반복해서 사용하지 마세요.
- 행의 필드 수가 헤더의 필드 수보다 적으면 Step Functions에서 누락된 값에 빈 문자열을 제공합니다.
- 행의 필드 수가 헤더의 필드 수보다 많으면 Step Functions는 추가 필드를 건너뜁니다.

Example 입력 CSV 파일 파싱

행 하나를 입력으로 포함하는 *myCSVInput.csv*라는 CSV 파일을 제공했다고 가정해보겠습니다. 그런 다음 *my-bucket*이라는 Amazon S3 버킷에 이 파일을 저장했습니다. CSV 파일은 다음과 같습니다.

```
abc,123,"This string contains commas, a double quotation marks (""), and a newline (
)",{""MyKey"":""MyValue""},"[1,2,3]"
```

다음 상태 시스템에서 이 CSV 파일을 읽고 ItemSelector를 사용하여 일부 필드의 데이터 유형을 변환합니다.

```
{
  "StartAt": "Map",
  "States": {
    "Map": {
      "Type": "Map",
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "DISTRIBUTED",
          "ExecutionType": "STANDARD"
        },
        "StartAt": "Pass",
        "States": {
          "Pass": {
            "Type": "Pass",
            "End": true
          }
        }
      },
```

```
      "End": true,
      "Label": "Map",
      "MaxConcurrency": 1000,
      "ItemReader": {
        "Resource": "arn:aws:states:::s3:getObject",
        "ReaderConfig": {
          "InputType": "CSV",
          "CSVHeaderLocation": "GIVEN",
          "CSVHeaders": [
            "MyLetters",
            "MyNumbers",
            "MyString",
            "MyObject",
            "MyArray"
          ]
        },
        "Parameters": {
          "Bucket": "my-bucket",
          "Key": "myCSVInput.csv"
        }
      },
      "ItemSelector": {
        "MyLetters.$": "$$.Map.Item.Value.MyLetters",
        "MyNumbers.$": "States.StringToJson($$.Map.Item.Value.MyNumbers)",
        "MyString.$": "$$.Map.Item.Value.MyString",
        "MyObject.$": "States.StringToJson($$.Map.Item.Value.MyObject)",
        "MyArray.$": "States.StringToJson($$.Map.Item.Value.MyArray)"
      }
    }
  }
}
```

이 상태 시스템을 실행하면 다음 출력이 생성됩니다.

```
[
  {
    "MyNumbers": 123,
    "MyObject": {
      "MyKey": "MyValue"
    },
    "MyString": "This string contains commas, a double quote (\"), and a newline (\n)",
    "MyLetters": "abc",
    "MyArray": [
```

```
      1,
      2,
      3
    ]
  }
]
```

컨텍스트 객체

컨텍스트 객체는 실행 중에 사용할 수 있는 내부 JSON 구조로, 여기에 상태 시스템과 실행에 대한 정보가 포함됩니다. 이를 사용하면 워크플로에서 특정 실행에 대한 정보에 액세스할 수 있습니다. 다음 필드에서 컨텍스트 객체에 액세스할 수 있습니다.

- `InputPath`
- `OutputPath`
- `ItemsPath`(Map 상태에서)
- `Variable`(Choice 상태에서)
- `ResultSelector`
- `Parameters`
- 변수와 변수 비교 연산자

컨텍스트 객체

컨텍스트 객체에는 상태 머신, 상태, 실행 및 작업에 대한 정보가 포함됩니다. 이 JSON 객체에는 각 데이터 유형에 대한 노드가 포함되며 다음과 같은 형식입니다.

```
{
    "Execution": {
        "Id": "String",
        "Input": {},
        "Name": "String",
        "RoleArn": "String",
        "StartTime": "Format: ISO 8601",
        "RedriveCount": Number,
        "RedriveTime": "Format: ISO 8601"
    },
    "State": {
        "EnteredTime": "Format: ISO 8601",
        "Name": "String",
```

```
        "RetryCount": Number
    },
    "StateMachine": {
        "Id": "String",
        "Name": "String"
    },
    "Task": {
        "Token": "String"
    }
}
```

실행 중에 컨텍스트 객체는 이 객체에 액세스하는 Parameters 필드에 대한 관련 데이터로 채워집니다. Parameters 필드가 작업 상태 범위를 벗어나는 경우 Task 필드의 값은 null입니다

아직 실행을 redriven하지 않은 경우 RedriveCount 컨텍스트 객체 값은 0입니다. 또한 RedriveTime 컨텍스트 객체는 실행을 redriven한 경우에만 사용 가능합니다. redriven a Map Run한 경우 표준 유형의 하위 워크플로에만 RedriveTime 컨텍스트 객체를 사용할 수 있습니다. Express 유형의 하위 워크플로가 포함된 redriven 맵 실행의 경우에는 RedriveTime을 사용할 수 없습니다.

진행 중인 실행의 콘텐츠에는 다음 형식의 세부 사항이 포함됩니다.

```
{
    "Execution": {
        "Id": "arn:aws:states:us-
east-1:123456789012:execution:stateMachineName:executionName",
        "Input": {
           "key": "value"
        },
        "Name": "executionName",
        "RoleArn": "arn:aws:iam::123456789012:role...",
        "StartTime": "2019-03-26T20:14:13.192Z"
    },
    "State": {
        "EnteredTime": "2019-03-26T20:14:13.192Z",
        "Name": "Test",
        "RetryCount": 3
    },
    "StateMachine": {
        "Id": "arn:aws:states:us-east-1:123456789012:stateMachine:stateMachineName",
        "Name": "stateMachineName"
    },
    "Task": {
        "Token": "h7XRiCdLtd/83p1E0dMccoxlzFhglsdkzpK9mBVKZsp7d9yrT1W"
```

```
    }
}
```

> **Note**
>
> Map 상태와 관련된 컨텍스트 객체 데이터는 Map 상태의 컨텍스트 객체 데이터 단원을 참조하십시오.

컨텍스트 객체 액세스

컨텍스트 객체에 액세스하려면 경로를 통해 상태 입력을 선택할 때와 같이 먼저 `.$`를 끝에 추가하여 파라미터 이름을 지정합니다. 그런 다음, 입력 대신 컨텍스트 객체 데이터에 액세스하기 위해 경로 앞에 `$$.`를 추가합니다. 이는 경로를 사용하여 컨텍스트 개체에서 노드를 AWS Step Functions 선택하도록 지시합니다.

다음 예제에서는 실행 ID, 이름, 시작 시간, redrive 개수와 같은 컨텍스트 개체에 액세스하는 방법을 보여줍니다.

- 실행 ARN을 검색하여 다운스트림 서비스에 전달
- Pass 상태의 실행 시작 시간과 이름에 액세스
- 실행의 redrive 횟수에 액세스

실행 ARN을 검색하여 다운스트림 서비스에 전달

이 예제 Task 상태는 경로를 사용하여 실행 Amazon 리소스 이름(ARN)을 검색하고 Amazon Simple Queue Service(Amazon SQS) 메시지에 전달합니다.

```
{
  "Order Flight Ticket Queue": {
    "Type": "Task",
    "Resource": "arn:aws:states:::sqs:sendMessage",
    "Parameters": {
      "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/flight-purchase",
      "MessageBody": {
        "From": "YVR",
        "To": "SEA",
        "Execution.$": "$$.Execution.Id"
      }
```

```
    },
    "Next": "NEXT_STATE"
  }
}
```

통합 서비스를 호출할 때 작업 토큰을 사용하는 방법에 대한 자세한 내용은 작업 토큰을 사용하여 콜백 대기 단원을 참조하십시오.

Pass 상태의 실행 시작 시간과 이름에 액세스

```
{
  "Comment": "Accessing context object in a state machine",
  "StartAt": "Get execution context data",
  "States": {
    "Get execution context data": {
      "Type": "Pass",
      "Parameters": {
        "startTime.$": "$$.Execution.StartTime",
        "execName.$": "$$.Execution.Name"
      },
      "ResultPath": "$.executionContext",
      "End": true
    }
  }
}
```

실행의 redrive 횟수에 액세스

다음 Task 상태 정의 예제에서는 실행의 redrive 횟수에 액세스하는 방법을 보여줍니다.

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "OutputPath": "$.Payload",
  "Parameters": {
    "Payload": {
      "Number.$": "$$.Execution.RedriveCount"
    },
    "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:functionName"
  },
  "End": true
}
```

Map 상태의 컨텍스트 객체 데이터

Map 상태를 처리하고 있을 때 컨텍스트 객체에서 사용 가능한 추가 항목은 Index와 Value입니다. Map 상태 반복마다 Index는 현재 처리 중인 배열 항목의 인덱스 번호를 포함하는 반면 Value는 처리 중인 배열 항목을 포함합니다. Map 상태에서 컨텍스트 객체에는 다음 데이터가 포함됩니다.

```
"Map": {
   "Item": {
      "Index": Number,
      "Value": "String"
   }
}
```

Map 상태에서만 이러한 데이터를 사용할 수 있으며 ItemSelector 필드에 지정할 수 있습니다.

> **Note**
>
> 컨텍스트 객체의 파라미터는 ItemProcessor 섹션에 포함된 상태가 아니라 기본 Map 상태의 ItemSelector 블록에서 정의할 수 있습니다.

상태 머신이 단순한 Map 상태로 지정되는 것을 고려할 때 컨텍스트 객체의 정보를 다음과 같이 주입할 수 있습니다.

```
{
  "StartAt": "ExampleMapState",
  "States": {
    "ExampleMapState": {
      "Type": "Map",
      "ItemSelector": {
        "ContextIndex.$": "$$.Map.Item.Index",
        "ContextValue.$": "$$.Map.Item.Value"
      },
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "INLINE"
        },
        "StartAt": "TestPass",
        "States": {
          "TestPass": {
            "Type": "Pass",
            "End": true
```

```
          }
        }
      },
      "End": true
    }
  }
}
```

다음 입력으로 이전 상태 머신을 실행하면 Index와 Value가 출력에 삽입됩니다.

```
[
  {
    "who": "bob"
  },
  {
    "who": "meg"
  },
  {
    "who": "joe"
  }
]
```

실행 출력은 다음과 같이 반복 3회마다 Index 및 Value 항목 값을 반환합니다.

```
[
  {
    "ContextIndex": 0,
    "ContextValue": {
      "who": "bob"
    }
  },
  {
    "ContextIndex": 1,
    "ContextValue": {
      "who": "meg"
    }
  },
  {

    "ContextIndex": 2,
    "ContextValue": {
      "who": "joe"
    }
```

```
  }
]
```

데이터 흐름 시뮬레이터

Step Functions 콘솔에서 워크플로를 설계, 구현 및 디버깅할 수 있습니다. JsonPath 입력 및 출력 처리를 사용하여 워크플로의 데이터 흐름을 제어할 수도 있습니다. 데이터 흐름 시뮬레이터를 사용하면 워크플로의 태스크 상태 상태에서 런타임 시 데이터를 처리하는 순서를 시뮬레이션할 수 있습니다. 시뮬레이터를 사용하면 한 상태에서 다른 상태로 이동하는 데이터를 필터링하고 조작하는 방법을 이해할 수 있습니다. Step Functions에서 JSON 데이터의 흐름을 처리하고 제어하는 데 사용하는 다음 각 필드를 시뮬레이션합니다.

`InputPath`

작업 입력으로 사용할 전체 입력 페이로드의 WHAT 부분을 선택합니다. 이 필드를 지정하면 Step Functions에서 먼저 이 필드를 적용합니다.

`####`

작업을 간접적으로 호출하기 전과 같이 HOW를 지정합니다. `Parameters` 필드를 사용하여 입력으로 AWS 서비스 통합에 전달되는 키-값 페어 컬렉션을 만들 수 있습니다(예: AWS Lambda 함수). 이러한 값은 정적이거나 상태 입력 또는 워크플로 컨텍스트 객체에서 동적으로 선택한 값일 수도 있습니다.

`ResultSelector`

작업 출력에서 선택할 WHAT을 결정합니다. `ResultSelector` 필드를 사용하여 상태 결과를 바꾸는 키-값 페어 컬렉션을 만들고 해당 컬렉션을 `ResultPath`에 전달할 수 있습니다.

`ResultPath`

작업 출력을 배치할 WHERE를 결정합니다. `ResultPath`를 사용하여 상태 출력이 입력 사본인지, 생성되는 결과인지 아니면 이 둘의 조합인지 확인합니다.

`OutputPath`

다음 상태로 전송할 WHAT을 결정합니다. `OutputPath`를 사용하여 원치 않는 정보를 필터링하고 관심 있는 JSON 데이터 부분만 전달할 수 있습니다.

이 주제의 내용

- 데이터 흐름 시뮬레이터 사용
- 데이터 흐름 시뮬레이터 사용에 대한 고려사항

데이터 흐름 시뮬레이터 사용

시뮬레이터는 입력 및 출력 데이터 처리 필드를 적용하기 전후의 데이터를 실시간으로 나란히 비교합니다. 시뮬레이터를 사용하려면 JSON 입력을 지정합니다. 그런 다음 각 입력 및 출력 처리 필드를 통해 평가합니다. 시뮬레이터는 자동으로 JSON 입력을 검증하고 모든 구문 오류를 강조 표시합니다.

데이터 흐름 시뮬레이터 사용하기

다음 단계에서는 JSON 입력을 제공하고 `InputPath` 및 `####` 필드를 적용합니다. 사용 가능한 다른 필드를 적용하고 해당 결과를 볼 수도 있습니다.

1. Step Functions 콘솔을 엽니다.
2. 탐색 창에서 데이터 흐름 시뮬레이터를 선택합니다.
3. 상태 입력 영역에서 미리 채워진 예제 JSON 데이터를 다음 JSON 데이터로 바꿉니다. 그리고 다음(Next)을 선택합니다.

```
{
  "data": {
    "firstname": "Jane",
    "lastname": "Doe",
    "identity": {
      "email": "jdoe@example.com",
      "ssn": "123-45-6789"
    },
    "address": {
      "street": "123 Main St",
      "city": "Columbus",
      "state": "OH",
      "zip": "43219"
    }
  }
}
```

4. InputPath의 경우 **`$.data.address`**를 입력하여 입력 JSON 데이터의 주소 노드를 선택합니다.

   InputPath 이후의 상태 입력 상자에 다음 결과가 표시됩니다.

```
{
  "street": "123 Main St",
  "city": "Columbus",
  "state": "OH",
```

```
  "zip": "43219"
}
```

5. 다음(Next)을 선택합니다.
6. Parameters 필드를 적용하여 결과 JSON 데이터를 문자열로 변환합니다. 데이터를 변환하려면 다음을 수행합니다.

   - 파라미터 상자에 다음 코드를 입력하여 addressString 문자열을 만듭니다.

```
{
    "addressString.$": "States.Format('{}. {}, {} - {}', $.street, $.city,
 $.state, $.zip)"
  }
```

7. 파라미터 이후의 필터링된 입력 상자에서 Parameters 필드 적용 결과를 봅니다.

데이터 흐름 시뮬레이터 사용에 대한 고려사항

데이터 흐름 시뮬레이터를 사용하기 전에 다음을 포함하지만 이에 국한되지 않는 제한사항을 고려합니다.

- 지원되지 않는 필터 표현식

  시뮬레이터의 필터 표현식은 Step Functions 서비스에서와 다르게 동작합니다. 여기에는 [?(expression)] 구문을 사용하는 표현식이 포함됩니다. 다음은 지원되지 않는 표현식의 목록입니다. 이러한 표현식을 사용하면 표현식에서 평가 후 예상 결과를 반환되지 않을 수 있습니다.

  - $..book[?(@.isInStock==true)]
  - $..book[?(@.price > 10.0)].title
- 단일 항목 배열에 대한 잘못된 JsonPath 평가

  단일 배열 항목을 반환하는 JsonPath 표현식으로 데이터를 필터링하면 시뮬레이터에서 배열 없이 항목을 반환합니다. 예를 들어 items라는 데이터 배열을 고려합니다.

```
{
  "items": [
    {
      "name": "shoe",
      "color": "blue",
      "comment": "nice shoe"
```

```
    },
    {
      "name": "hat",
      "color": "red"
    },
    {
      "name": "shirt",
      "color": "yellow"
    }
  ]
}
```

이 `items` 배열을 제공했을 때 InputPath 필드에 **`$..comment`**를 입력하면 다음과 같은 결과를 예상합니다.

```
[
  "nice shoe"
]
```

하지만 데이터 흐름 시뮬레이터는 대신 다음과 같은 출력을 반환합니다.

```
"nice shoe"
```

여러 항목이 포함된 배열의 JsonPath 평가의 경우 시뮬레이터는 예상 출력을 반환합니다.

버전 및 별칭을 사용하여 지속적인 배포 관리

Step Functions를 사용하여 상태 시스템 버전 및 별칭을 통해 워크플로의 지속적인 배포를 관리할 수 있습니다. 버전은 실행 가능한 상태 시스템의 번호가 매겨져 있고 변경할 수 없는 스냅샷입니다. 별칭은 상태 시스템 버전을 최대 2개까지 가리킵니다.

상태 시스템 버전을 여러 개 유지하고 프로덕션 워크플로에서 상태 시스템 배포를 관리할 수 있습니다. 별칭을 사용하면 서로 다른 워크플로 버전 간에 트래픽을 라우팅하고 점진적으로 해당 워크플로를 프로덕션 환경에 배포할 수 있습니다.

또한 버전이나 별칭을 사용하여 상태 시스템 실행을 시작할 수 있습니다. 상태 시스템 실행을 시작할 때 버전이나 별칭을 사용하지 않으면 Step Functions에서 상태 시스템 정의의 최신 버전을 사용합니다.

> **상태 시스템 개정**
>
> 상태 시스템에는 개정이 하나 이상 있을 수 있습니다. UpdateStateMachine API 작업을 사용하여 상태 시스템을 업데이트하면 새 상태 시스템 개정이 생성됩니다. 개정은 상태 시스템 정의 및 구성의 변경할 수 없는 읽기 전용 스냅샷입니다. 개정에서 상태 시스템 실행을 시작할 수 없으며 개정에는 ARN이 없습니다. 개정에는 범용 고유 식별자(UUID)인 `revisionId`가 있습니다.

목차

상태 시스템 버전

버전은 번호가 매겨져 있으며 변경할 수 없는 상태 시스템 스냅샷입니다. 해당 상태 시스템에 적용된 최신 버전에서 버전을 게시합니다. 버전마다 고유한 Amazon 리소스 이름(ARN)이 있습니다. 이 ARN은 콜론(:)으로 구분된 상태 시스템 ARN과 버전 번호의 조합입니다. 다음 예제에서는 상태 시스템 버전 ARN의 형식을 보여줍니다.

```
arn:partition:states:region:account-id:stateMachine:myStateMachine:1
```

상태 시스템 버전을 사용하려면 첫 번째 버전을 게시해야 합니다. 버전을 게시한 후 버전 ARN을 StartExecution사용하여 API 작업을 호출할 수 있습니다. 버전을 편집할 수 없지만 상태 시스템을 업데이트하고 새 버전을 게시할 수 있습니다. 상태 시스템 버전 여러 개를 게시할 수도 있습니다.

새 상태 시스템 버전을 게시하면 Step Functions에서 버전 번호를 할당합니다. 버전 번호는 1부터 시작하며 새 버전이 나올 때마다 단조롭게 증가합니다. 지정된 상태 시스템에는 버전 번호가 다시 사용되지 않습니다. 상태 시스템 버전 10을 삭제한 다음 새 버전을 게시하면 Step Functions에서 상태 시스템을 버전 11로 게시합니다.

다음 속성은 상태 시스템의 모든 버전에서 동일합니다.

- 상태 시스템의 모든 버전은 동일한 유형(표준 또는 Express)을 공유합니다.
- 버전 간에 상태 시스템 이름이나 만든 날짜를 변경할 수 없습니다.
- 태그는 전역적으로 상태 시스템에 적용됩니다. TagResource및 UntagResourceAPI 작업을 사용하여 상태 머신의 태그를 관리할 수 있습니다.

상태 시스템에도 각 버전 및 revision에 속하는 속성이 포함되지만 이러한 속성은 두 가지 지정된 버전이나 개정마다 다를 수 있습니다. 이러한 속성에는 상태 시스템 정의, IAM 역할, 추적 구성 및 로깅 구성이 포함됩니다.

내용

- 상태 시스템 버전 게시(콘솔)
- Step Functions API 작업을 사용하여 버전 관리
- 콘솔에서 상태 시스템 버전 실행

상태 시스템 버전 게시(콘솔)

상태 시스템 버전을 최대 1,000개까지 게시할 수 있습니다. 이 소프트 한도 증가를 요청하려면 AWS Management Console의 지원 센터 페이지를 사용합니다. 콘솔에서 사용하지 않는 버전을 수동으로 삭제하거나 DeleteStateMachineVersionAPI 작업을 호출하여 삭제할 수 있습니다.

상태 머신 버전 게시하기

1. Step Functions 콘솔을 열고 기존 상태 시스템을 선택합니다.
2. 상태 시스템 세부 정보 페이지에서 편집을 선택합니다.
3. 필요에 따라 상태 시스템 정의를 편집한 다음 저장을 선택합니다.
4. Publish version(버전 게시)을 선택합니다.
5. (선택 사항) 나타나는 대화 상자의 설명 필드에 상태 시스템 버전에 대한 간략한 설명을 입력합니다.
6. 게시를 선택합니다.

> **Note**
>
> 새 상태 시스템 버전을 게시하면 Step Functions에서 버전 번호를 할당합니다. 버전 번호는 1부터 시작하며 새 버전이 나올 때마다 단조롭게 증가합니다. 지정된 상태 시스템에는 버전 번호가 다시 사용되지 않습니다. 상태 시스템 버전 10을 삭제한 다음 새 버전을 게시하면 Step Functions에서 상태 시스템을 버전 11로 게시합니다.

Step Functions API 작업을 사용하여 버전 관리

Step Functions는 상태 시스템 버전을 게시하고 관리하도록 다음 API 작업을 제공합니다.

- PublishStateMachineVersion— 상태 revision 머신의 최신 버전을 게시합니다.
- UpdateStateMachine— 상태 컴퓨터를 업데이트하고 동일한 `true` 요청에서 `publish` 매개 변수를 로 설정한 경우 새 상태 컴퓨터 버전을 게시합니다.
- CreateStateMachine— `publish` 매개 변수를 로 설정한 경우 상태 머신의 첫 번째 개정판을 게시합니다. `true`
- ListStateMachineVersions— 지정된 상태 시스템 ARN의 버전을 나열합니다.
- DescribeStateMachine— 지정된 버전 ARN에 대한 스테이트 머신 버전 세부 정보를 반환합니다. `stateMachineArn`
- DeleteStateMachineVersion— 스테이트 머신 버전을 삭제합니다.

를 *`myStateMachine`* 사용하여 호출한 상태 머신의 현재 버전에서 새 버전을 게시하려면 다음 `publish-state-machine-version` 명령을 사용합니다. AWS Command Line Interface

```
aws stepfunctions publish-state-machine-version --state-machine-arn arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
```

응답은 `stateMachineVersionArn`을 반환합니다. 예를 들어 이전 명령은 `arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:1`에 대한 응답을 반환합니다.

> **Note**
>
> 새 상태 시스템 버전을 게시하면 Step Functions에서 버전 번호를 할당합니다. 버전 번호는 1부터 시작하며 새 버전이 나올 때마다 단조롭게 증가합니다. 지정된 상태 시스템에는 버전 번호가 다시 사용되지 않습니다. 상태 시스템 버전 10을 삭제한 다음 새 버전을 게시하면 Step Functions에서 상태 시스템을 버전 11로 게시합니다.

콘솔에서 상태 시스템 버전 실행

상태 시스템 버전을 시작하려면 먼저 현재 상태 시스템 revision에서 버전을 게시해야 합니다. 버전을 게시하려면 Step Functions 콘솔을 사용하거나 PublishStateMachineVersionAPI 작업을 호출하십시오. 이름이 지정된 선택적 매개 변수를 UpdateStateMachineAlias사용하여 API 작업을 `publish` 호출하여 상태 머신을 업데이트하고 해당 버전을 게시할 수도 있습니다.

콘솔을 사용하거나 StartExecutionAPI 작업을 호출하고 버전 ARN을 제공하여 버전 실행을 시작할 수 있습니다. 별칭을 사용하여 버전 실행을 시작할 수도 있습니다. 라우팅 구성에 따라 별칭에서 트래픽을 특정 버전으로 라우팅합니다.

버전을 사용하지 않고 상태 시스템 실행을 시작하면 Step Functions는 상태 시스템 최신 개정을 실행에 사용합니다. Step Functions에서 실행을 버전과 연결하는 방식은 실행을 버전 또는 별칭과 연결을 참조하세요.

상태 시스템 버전을 사용하여 실행 시작하기

1. Step Functions 콘솔을 열고 버전을 하나 이상 게시한 기존 상태 시스템을 선택합니다. 버전을 게시하는 방법은 상태 시스템 버전 게시(콘솔)를 참조하세요.
2. 상태 시스템 세부 정보 페이지에서 버전 탭을 선택합니다.
3. 버전 섹션에서 다음을 수행합니다.
   a. 실행을 시작할 버전을 선택합니다.
   b. 실행 시작을 선택합니다.

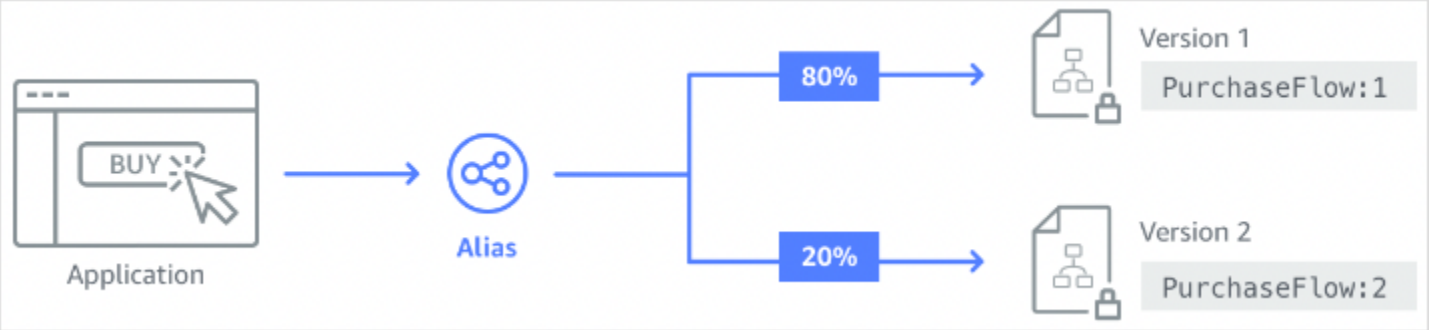
4. (선택 사항) 실행 시작 대화 상자에 실행 이름을 입력합니다.
5. (선택 사항) 실행 입력을 입력한 다음 실행 시작을 선택합니다.

상태 시스템 별칭

별칭은 같은 상태 시스템의 버전을 최대 2개까지 가리킵니다. 상태 시스템 별칭을 여러 개 만들 수 있습니다. 별칭마다 고유한 Amazon 리소스 이름(ARN)이 있습니다. 별칭 ARN은 상태 시스템 ARN과 별칭 이름의 조합이며 콜론(:)으로 구분됩니다. 다음 예제에서는 상태 시스템 별칭 ARN의 형식을 보여줍니다.

```
arn:partition:states:region:account-id:stateMachine:myStateMachine:aliasName
```

별칭을 사용하여 두 상태 시스템 버전 중 하나 간에 트래픽을 라우팅할 수 있습니다. 단일 버전을 가리키는 별칭을 만들 수도 있습니다. 별칭은 상태 시스템 버전만 가리킬 수 있습니다. 별칭을 사용하여 다른 별칭을 가리킬 수는 없습니다. 상태 시스템의 다른 버전을 가리키도록 별칭을 업데이트할 수도 있습니다.



내용

- 상태 시스템 별칭 만들기(콘솔)
- Step Functions API 작업으로 별칭 관리
- 별칭 라우팅 구성
- 별칭을 사용하여 상태 시스템 실행(콘솔)

상태 시스템 별칭 만들기(콘솔)

Step Functions 콘솔을 사용하거나 CreateStateMachineAliasAPI 작업을 호출하여 각 상태 머신에 대해 최대 100개의 별칭을 생성할 수 있습니다. 이 소프트 한도 증가를 요청하려면 AWS Management

Console의 지원 센터 페이지를 사용합니다. 콘솔에서 또는 API 작업을 호출하여 사용하지 않는 별칭을 삭제합니다. DeleteStateMachineAlias

상태 시스템 별칭 만들기

1. Step Functions 콘솔을 열고 기존 상태 시스템을 선택합니다.
2. 상태 시스템 세부 정보 페이지에서 별칭 탭을 선택합니다.
3. 새 별칭 생성을 선택합니다.
4. 별칭 생성(Create alias) 페이지에서 다음을 수행합니다.
   a. 별칭 이름을 입력합니다.
   b. (선택 사항) 경보에 대한 설명(Description)을 입력합니다.
5. 별칭에 대한 라우팅을 구성하려면 별칭 라우팅 구성을 참조하세요.
6. 별칭 생성을 선택합니다.

Step Functions API 작업으로 별칭 관리

Step Functions는 상태 시스템 별칭을 생성 및 관리하거나 별칭에 대한 정보를 가져오는 데 사용할 수 있는 다음 API 작업을 제공합니다.

- CreateStateMachineAlias— 상태 머신의 별칭을 생성합니다.
- DescribeStateMachineAlias— 상태 시스템 별칭에 대한 세부 정보를 반환합니다.
- ListStateMachineAliases— 지정된 상태 시스템 ARN의 별칭을 나열합니다.
- UpdateStateMachineAlias— 또는 를 수정하여 기존 상태 시스템 별칭의 구성을 업데이트합니다. `description` `routingConfiguration`
- DeleteStateMachineAlias— 상태 시스템 버전을 삭제합니다.

를 *myStateMachine* 사용하여 이름을 지정한 *PROD* 상태 머신의 버전 1을 가리키는 별칭을 만들려면 AWS Command Line Interface 명령을 사용합니다. `create-state-machine-alias`

```
aws stepfunctions create-state-machine-alias --name PROD  --routing-
configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:1\",\"weight\":100}]"
```

별칭 라우팅 구성

별칭을 사용하여 상태 시스템 두 버전 간에 실행 트래픽을 라우팅할 수 있습니다. 새 버전의 상태 시스템을 시작하려는 경우를 예로 들어 보겠습니다. 별칭에 대한 라우팅을 구성하여 새 버전 배포와 관련된 위험을 줄일 수 있습니다. 라우팅을 구성하면 대부분의 트래픽을 테스트를 거친 상태 시스템의 이전 버전으로 보낼 수 있습니다. 그러면 새 버전을 롤포워드해도 안전한지 확인할 때까지 새 버전에 더 적은 비율을 보낼 수 있습니다.

라우팅 구성을 정의하려면 별칭에서 가리키는 상태 시스템 버전 두 개 모두 게시해야 합니다. 별칭에서 실행을 시작하면 Step Functions는 라우팅 구성에 지정된 버전 중에서 실행할 상태 시스템 버전을 임의로 선택합니다. 이 선택은 별칭 라우팅 구성에서 각 버전에 할당하는 트래픽 비율을 기반으로 합니다.

별칭에 대한 라우팅 구성 구성하기

- 별칭 만들기 페이지의 라우팅 구성에서 다음을 수행합니다.

  a. 버전에서 별칭이 가리키는 첫 번째 상태 시스템 버전을 선택합니다.

  b. 두 버전 간 트래픽 분할 확인란을 선택합니다.

  > **Tip**
  > 단일 버전을 가리키려면 두 버전 간 트래픽 분할 확인란을 선택 취소하세요.

  c. 버전에서 별칭이 가리켜야 하는 두 번째 버전을 선택합니다.

  d. 트래픽 비율 필드에서 각 버전으로 라우팅할 트래픽 비율을 지정합니다. 예를 들어 실행 트래픽의 60%를 첫 번째 버전으로, 40%를 두 번째 버전으로 라우팅하려면 **`60`** 및 **`40`**을 입력합니다.

     합친 트래픽 비율은 100%이어야 합니다.

별칭을 사용하여 상태 시스템 실행(콘솔)

콘솔에서 별칭을 사용하거나 별칭의 ARN으로 StartExecutionAPI 작업을 호출하여 상태 시스템 실행을 시작할 수 있습니다. 그러면 Step Functions에서 별칭으로 지정된 버전을 실행합니다. 상태 시스템 실행을 시작할 때 버전이나 별칭을 지정하지 않으면 Step Functions는 기본적으로 최신 버전을 사용합니다.

별칭을 사용하여 상태 시스템 실행 시작하기

1. Step Functions 콘솔을 열고 별칭을 만든 기존 상태 시스템을 선택합니다. 별칭을 만드는 방법은 상태 시스템 별칭 만들기(콘솔)를 참조하세요.
2. 상태 시스템 세부 정보 페이지에서 별칭 탭을 선택합니다.
3. 별칭 섹션에서 다음을 수행합니다.
   a. 실행을 시작할 별칭을 선택합니다.
   b. 실행 시작을 선택합니다.
4. (선택 사항) 실행 시작 대화 상자에 실행 이름을 입력합니다.
5. 필요한 경우 실행 입력을 입력한 다음 실행 시작을 선택합니다.

버전 및 별칭에 대한 권한 부여

버전 또는 별칭을 사용하여 Step Functions API 작업을 간접적으로 호출하려면 적절한 권한이 필요합니다. API 작업을 간접적으로 호출하도록 버전이나 별칭에 권한을 부여하기 위해 Step Functions는 버전 ARN 또는 별칭 ARN을 사용하는 대신 상태 시스템의 ARN을 사용합니다. 특정 버전이나 별칭에 대한 권한의 범위를 좁힐 수도 있습니다. 자세한 내용은 권한 범위 축소 섹션을 참조하세요.

*myStateMachine*이라는 상태 시스템의 다음 IAM 정책 예제를 사용하면 CreateStateMachineAlias API 작업을 간접적으로 호출하여 상태 시스템 별칭을 만들 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "states:CreateStateMachineAlias",
      "Resource": "arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine"
    }
  ]
}
```

상태 시스템 버전이나 별칭을 사용하여 API 작업에 대한 액세스를 허용하거나 거부할 수 있는 권한을 설정할 때는 다음 사항을 고려하세요.

- CreateStateMachine 및 UpdateStateMachine API 작업의 publish 파라미터를 사용하여 새 상태 시스템 버전을 게시하는 경우 PublishStateMachineVersion API 작업에 대한 ALLOW 권한도 필요합니다.
- DeleteStateMachine API 작업은 상태 시스템과 관련된 모든 버전 및 별칭을 삭제합니다.

이 주제의 내용

버전 또는 별칭에 대한 권한 범위 축소

한정자를 사용하여 버전이나 별칭에 필요한 권한을 부여할 수 있는 권한의 범위를 더 좁힐 수 있습니다. 한정자는 버전 번호나 별칭 이름을 나타냅니다. 한정자를 사용하여 상태 시스템을 검증할 수 있습니다. 다음 예제는 한정자로 PROD라는 별칭을 사용하는 상태 시스템 ARN입니다.

```
arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:PROD
```

검증된 ARN 및 검증되지 않은 ARN에 대한 자세한 내용은 실행을 버전 또는 별칭과 연결을 참조하세요.

IAM 정책의 Condition 문에서 states:StateMachineQualifier라는 선택적 컨텍스트 키를 사용하여 권한 범위를 좁힙니다. 예를 들어 myStateMachine이라는 상태 시스템에 대한 다음 IAM 정책은 PROD라는 별칭이나 버전 1을 사용하여 DescribeStateMachine API 작업을 간접적으로 호출할 수 있는 액세스를 거부합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "states:DescribeStateMachine",
      "Resource": "arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "states:StateMachineQualifier": [
            "PROD",
            "1"
          ]
        }
      }
```

```
    }
  ]
}
```

다음 목록에서는 StateMachineQualifier 컨텍스트 키를 사용하여 권한 범위를 좁힐 수 있는 API 작업을 지정합니다.

- CreateStateMachineAlias
- DeleteStateMachineAlias
- DeleteStateMachineVersion
- DescribeStateMachine
- DescribeStateMachineAlias
- ListExecutions
- ListStateMachineAliases
- StartExecution
- StartSyncExecution
- UpdateStateMachineAlias

상태 시스템 실행을 버전 또는 별칭과 연결

Step Functions는 API 작업을 호출하는 데 사용하는 Amazon 리소스 이름 (ARN) 을 기반으로 한 버전 또는 별칭과 실행을 연결합니다. StartExecution Step Functions는 실행 시작 시간에 이 작업을 수행합니다.

정규화된 또는 정규화되지 않은 ARN을 사용하여 상태 시스템 실행을 시작할 수 있습니다.

- 정규화된 ARN - 버전 번호 또는 별칭 이름이 접미사로 추가된 상태 시스템 ARN을 의미합니다.

  다음 정규화된 ARN 예제는 myStateMachine이라는 상태 시스템의 버전 3을 참조합니다.

  ```
  arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:3
  ```

  다음 정규화된 ARN 예제는 myStateMachine이라는 상태 시스템의 PROD 별칭을 참조합니다.

  ```
  arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:PROD
  ```

- 정규화되지 않은 ARN - 버전 번호나 별칭 이름 접미사가 없는 상태 시스템 ARN을 의미합니다.

```
arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
```

예를 들어 정규화된 ARN에서 버전 3을 참조하면 Step Functions는 실행을 이 버전과 연결합니다. 버전 3을 가리키는 별칭과 실행을 연결하지 않습니다.

정규화된 ARN에서 별칭을 참조하면 Step Functions는 실행을 해당 별칭 및 별칭이 가리키는 버전과 연결합니다. 실행을 별칭 하나에만 연결할 수 있습니다.

> **Note**
>
> 정규화되지 않은 ARN으로 실행을 시작하면 Step Functions는 버전에서 같은 상태 시스템 revision을 사용하더라도 해당 실행을 버전과 연결하지 않습니다. 예를 들어 버전 3에서 최신 버전을 사용하지만 정규화되지 않은 ARN으로 실행을 시작하면 Step Functions는 해당 실행을 버전 3과 연결하지 않습니다.

이 주제의 내용

버전 또는 별칭으로 시작된 실행 보기

Step Functions는 버전 또는 별칭으로 시작된 실행을 볼 수 있는 다음과 같은 방법을 제공합니다.

API 작업 사용

및 API 작업을 호출하여 버전 또는 별칭과 관련된 모든 실행을 볼 수 있습니다. DescribeExecutionListExecutions 이러한 API 작업은 실행을 시작하는 데 사용된 버전이나 별칭의 ARN을 반환합니다. 또한 이러한 작업은 실행 상태와 ARN을 포함한 기타 세부 정보도 반환합니다.

상태 시스템 별칭 ARN 또는 버전 ARN을 제공하여 특정 별칭이나 버전과 연결된 실행을 나열할 수도 있습니다.

다음 ListExecutionsAPI 작업의 예제 응답은 이라는 상태 시스템 실행을 시작하는 데 사용되는 별칭의 ARN을 보여줍니다. *myFirstExecution*

다음 코드 스니펫에서 *####*로 표시된 텍스트는 리소스별 정보를 나타냅니다.

```
{
    "executions": [
        {
            "executionArn": "arn:aws:states:us-
east-1:123456789012:execution:myStateMachine:myFirstExecution",
            "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine",
            "stateMachineAliasArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:myStateMachine:PROD",
            "name": "myFirstExecution",
            "status": "SUCCEEDED",
            "startDate": "2023-04-20T23:07:09.477000+00:00",
            "stopDate": "2023-04-20T23:07:09.732000+00:00"
        }
    ]
}
```

Step Functions 콘솔 사용

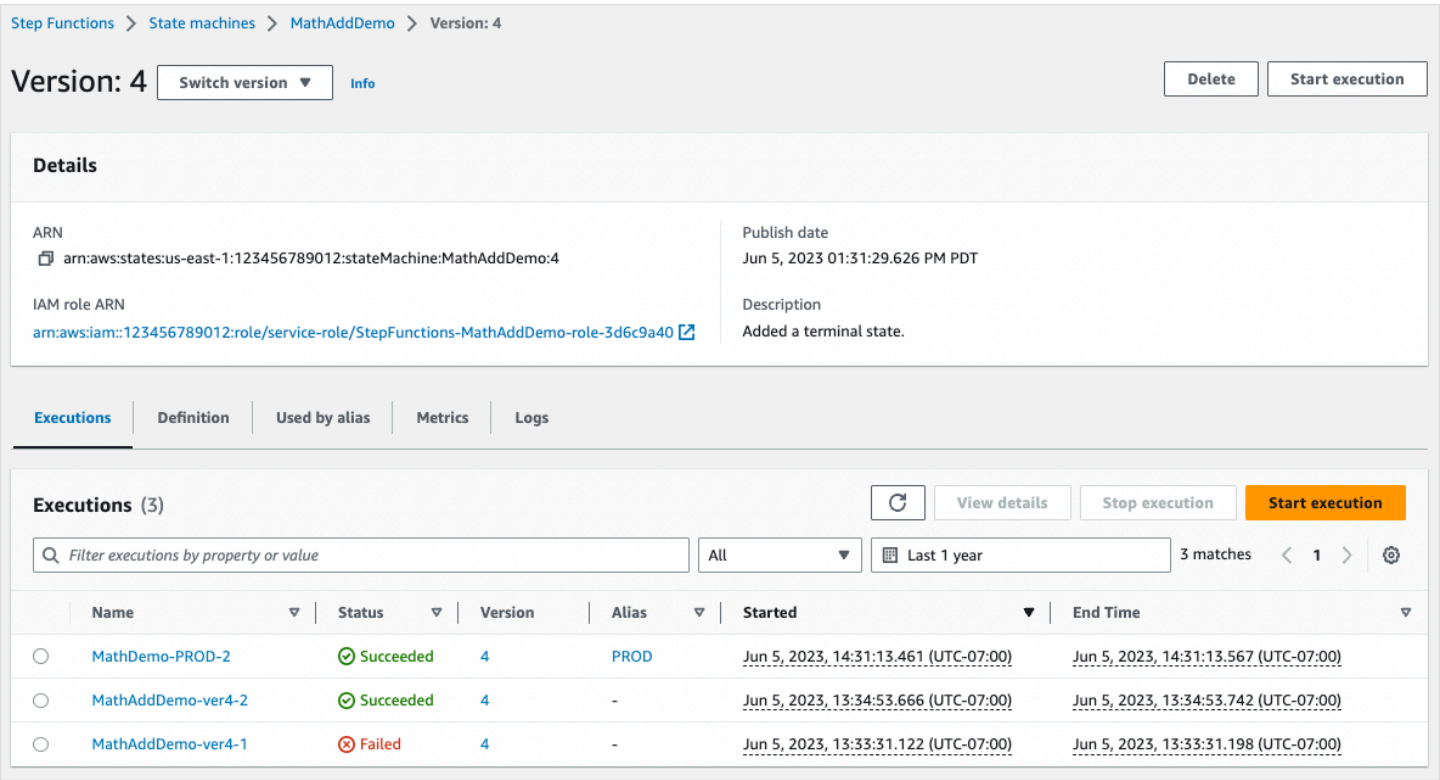
Step Functions 콘솔에서도 버전이나 별칭으로 시작된 실행을 볼 수 있습니다. 다음 절차에서는 특정 버전으로 시작된 실행을 보는 방법을 보여줍니다.

1. Step Functions 콘솔을 열고 버전을 게시했거나 별칭을 만든 기존 상태 시스템을 선택합니다. 이 예제에서는 특정 상태 시스템 버전으로 시작된 실행을 보는 방법을 보여줍니다.
2. 버전 탭을 선택한 다음 버전 목록에서 버전을 선택합니다.

   > **Tip**
   >
   > 속성 또는 값 상자별로 필터링하여 특정 버전을 검색합니다.

3. 버전 세부 정보 페이지에서 선택한 버전으로 시작된 모든 진행 중인 상태 시스템 실행과 이전 상태 시스템 실행 목록을 확인할 수 있습니다.

다음 이미지에서는 버전 세부 정보 콘솔 페이지를 보여줍니다. 이 페이지에는 *MathAddDemo*라는 상태 시스템의 버전 4에서 시작된 실행이 나열됩니다. 또한 이 목록에는 *PROD*라는 별칭으로 시작된 실행도 표시됩니다. 이 별칭은 실행 트래픽을 버전 4로 라우팅했습니다.



CloudWatch 지표 사용

Qualified ARN으로 시작하는 상태 시스템 실행마다 Step Functions는 현재 내보내는 지표와 동일한 이름 및 값을 가진 추가 지표를 내보냅니다. 이러한 추가 지표에는 실행을 시작하는 데 사용하는 버전 식별자 및 별칭 이름 각각에 대한 차원이 포함됩니다. 이러한 지표를 사용하면 버전 수준에서 상태 시스템 실행을 모니터링하고 롤백 시나리오가 필요한 시기를 결정할 수 있습니다. 이러한 지표를 기반으로 Amazon CloudWatch 경보를 생성할 수도 있습니다.

Step Functions는 별칭 또는 버전으로 시작하는 실행에 대한 다음 지표를 내보냅니다.

- `ExecutionTime`
- `ExecutionsAborted`
- `ExecutionsFailed`
- `ExecutionsStarted`
- `ExecutionsSucceeded`
- `ExecutionsTimedOut`

버전 ARN으로 실행을 시작한 경우 Step Functions는 `StateMachineArn`을 사용하여 지표를 게시하고 `StateMachineArn` 및 `Version` 차원을 사용하여 두 번째 지표를 게시합니다.

별칭 ARN으로 실행을 시작한 경우 Step Functions는 다음 지표를 내보냅니다.

- 정규화되지 않은 ARN 및 버전에 대한 지표 2개
- `StateMachineArn` 및 `Alias` 차원이 포함된 지표

별칭 및 버전 배포 예제

다음 카나리 배포 기법 예제에서는 AWS Command Line Interface를 사용하여 새 상태 시스템 버전을 배포하는 방법을 보여줍니다. 이 예시에서는 생성된 별칭이 실행 트래픽의 20%를 새 버전으로 라우팅합니다. 그런 다음 나머지 80%를 이전 버전으로 라우팅합니다. 새 상태 시스템 버전을 배포하고 별칭으로 실행 트래픽을 전환하려면 다음 단계를 완료합니다.

1. 현재 상태 시스템 개정의 버전을 게시합니다.

   AWS CLI에서 publish-state-machine-version 명령을 사용하여 *myStateMachine*:이라는 상태 시스템의 현재 개정에서 버전을 게시합니다.

   ```
   aws stepfunctions publish-state-machine-version --state-machine-arn
    arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine
   ```

   응답은 게시한 버전의 `stateMachineVersionArn`을 반환합니다. 예: `arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:1`.

2. 상태 시스템 버전을 가리키는 별칭을 만듭니다.

   create-state-machine-alias 명령을 사용하여 *myStateMachine* 버전 1을 가리키는 *PROD*라는 별칭을 만듭니다.

   ```
   aws stepfunctions create-state-machine-alias --name PROD --routing-
   configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
   east-1:123456789012:stateMachine:myStateMachine:1\",\"weight\":100}]"
   ```

3. 별칭으로 시작된 실행에서 올바르게 게시된 버전을 사용하는지 확인합니다.

   start-execution 명령에 **PROD** 별칭의 ARN을 제공하여 *myStateMachine*의 새 실행을 시작합니다.

   ```
   aws stepfunctions start-execution
     --state-machine-arn arn:aws:states:us-
   east-1:123456789012:stateMachineAlias:myStateMachine:PROD
     --input "{}"
   ```

StartExecution 요청에 상태 시스템 ARN을 제공하면 실행 시작 시 별칭에 지정된 버전 대신 최신 revision의 상태 시스템이 사용됩니다.

4. 상태 시스템 정의를 업데이트하고 새 버전을 게시합니다.

   *myStateMachine*을 업데이트하고 새 버전을 게시합니다. 이렇게 하려면 update-state-machine 명령의 선택적 publish 파라미터를 사용합니다.

   ```
   aws stepfunctions update-state-machine
       --state-machine-arn arn:aws:states:us-
   east-1:123456789012:stateMachine:myStateMachine
       --definition $UPDATED_STATE_MACHINE_DEFINITION
       --publish
   ```

   응답에서 새 버전의 stateMachineVersionArn을 반환합니다. 예: arn:aws:states:us-east-1:*123456789012*:stateMachine:*myStateMachine*:2.

5. 두 버전을 모두 가리키도록 별칭을 업데이트하고 별칭의 라우팅 구성을 설정합니다.

   update-state-machine-alias 명령을 사용하여 별칭 PROD의 라우팅 구성을 업데이트합니다. 실행 트래픽의 80%는 버전 1로 이동하고 나머지 20%는 버전 2로 이동하도록 별칭을 구성합니다.

   ```
   aws stepfunctions update-state-machine-alias --state-machine-alias-arn
    arn:aws:states:us-east-1:123456789012:stateMachineAlias:myStateMachine:PROD
    --routing-configuration "[{\"stateMachineVersionArn\":
   \"arn:aws:states:us-east-1:123456789012:stateMachine:myStateMachine:1\",
   \"weight\":80}, {\"stateMachineVersionArn\":\"arn:aws:states:us-
   east-1:123456789012:stateMachine:myStateMachine:2\",\"weight\":20}]"
   ```

6. 버전 1을 버전 2로 바꿉니다.

   새 상태 시스템 버전이 올바르게 작동하는지 확인한 후에 새 상태 시스템 버전을 배포할 수 있습니다. 이렇게 하려면 별칭을 다시 업데이트하여 실행 트래픽의 100%를 새 버전에 할당합니다.

   update-state-machine-alias 명령을 사용하여 별칭 PROD의 라우팅 구성을 버전 2의 100%로 설정합니다.

   ```
   aws stepfunctions update-state-machine-alias --state-machine-alias-arn
    arn:aws:states:us-east-1:123456789012:stateMachineAlias:myStateMachine:PROD
    --routing-configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
   east-1:123456789012:stateMachine:myStateMachine:2\",\"weight\":100}]"
   ```

> **Tip**
>
> 버전 2의 배포를 롤백하려면 트래픽의 100%가 새로 배포된 버전으로 이동하도록 별칭의 라우팅 구성을 편집합니다.
>
> ```
> aws stepfunctions update-state-machine-alias
>   --state-machine-alias-arn arn:aws:states:us-
> east-1:123456789012:stateMachineAlias:myStateMachine:PROD
>   --routing-configuration "[{\"stateMachineVersionArn\":\"arn:aws:states:us-
> east-1:123456789012:stateMachine:myStateMachine:1\",\"weight\":100}]"
> ```

버전과 별칭을 사용하여 다른 유형의 배포를 수행할 수 있습니다. 예를 들어 새 버전의 상태 시스템을 순차적으로 배포할 수 있습니다. 이렇게 하려면 새 버전을 가리키는 별칭의 라우팅 구성에서 가중치를 적용하는 비율을 점차 늘립니다.

버전과 별칭을 사용하여 블루/그린 배포를 수행할 수도 있습니다. 이렇게 하려면 상태 시스템의 현재 버전 1을 실행하는 `green` 별칭을 만듭니다. 그런 다음 새 버전(예: *2*)을 실행하는 또 다른 `blue` 별칭을 만듭니다. 새 버전을 테스트하려면 실행 트래픽을 `blue` 별칭으로 전송합니다. 새 버전이 올바르게 작동한다고 확신하면 새 버전을 가리키도록 `green` 별칭을 업데이트합니다.

상태 시스템 버전 점진적 배포 수행

롤링 배포는 이전 버전의 애플리케이션을 새 버전의 애플리케이션으로 천천히 바꾸는 배포 전략입니다. 상태 시스템 버전 롤링 배포를 수행하려면 실행 트래픽 양을 점진적으로 늘리면서 새 버전으로 전송합니다. 트래픽 양과 증가 속도는 사용자가 구성하는 파라미터입니다.

다음 옵션 중 하나를 사용하여 버전 롤링 배포를 수행할 수 있습니다.

- Step Functions 콘솔 - 같은 상태 시스템 버전 2개를 가리키는 별칭을 만듭니다. 이 별칭의 경우 트래픽이 두 버전 간에 이동하도록 라우팅 구성을 구성합니다. 콘솔을 사용하여 버전을 출시하는 방법에 대한 자세한 내용은 버전 및 에일리어스을 참조하세요.
- AWS CLI 및 SDK용 스크립트 - AWS CLI 또는 AWS SDK를 사용하여 쉘 스크립트를 만듭니다. 자세한 내용은 AWS CLI 및 AWS SDK 사용에 대한 다음 섹션을 참조하세요.
- AWS CloudFormation 템플릿 - `AWS::StepFunctions::StateMachineVersion` 및 `AWS::StepFunctions::StateMachineAlias` 리소스를 사용하여 여러 상태 시스템 버전을 게시하고 이러한 버전 중 하나 또는 두 개를 가리키는 별칭을 만듭니다.

AWS CLI를 사용하여 새 상태 시스템 버전 배포

이 섹션의 예제 스크립트에서는 AWS CLI를 사용하여 트래픽을 이전 상태 시스템 버전에서 새 상태 시스템 버전으로 점진적으로 이동하는 방법을 보여줍니다. 이 예제 스크립트를 사용하거나 요구 사항에 따라 업데이트할 수 있습니다.

이 스크립트에서는 별칭을 사용하여 새 상태 시스템 버전을 배포하기 위한 카나리 배포를 보여줍니다. 다음 단계에서는 스크립트에서 수행하는 작업을 간략하게 설명합니다.

1. `publish_revision` 파라미터가 true로 설정되면 최신 revision을 상태 시스템의 다음 버전으로 게시합니다. 배포가 성공하면 이 버전이 새로운 실시간 버전이 됩니다.

   `publish_revision` 파라미터를 false로 설정하면 스크립트는 마지막으로 게시된 상태 시스템 버전을 배포합니다.
2. 아직 존재하지 않으면 별칭을 만듭니다. 별칭이 존재하지 않으면 이 별칭의 모든 트래픽이 새 버전을 가리키도록 한 다음 스크립트를 종료합니다.
3. 별칭의 라우팅 구성을 업데이트하여 소량의 트래픽을 이전 버전에서 새 버전으로 이동합니다. `canary_percentage` 파라미터를 사용하여 이 canary 비율을 설정합니다.
4. 기본적으로 구성 가능한 CloudWatch 경보를 60초 간격으로 모니터링합니다. 이러한 경보 중 하나라도 발생하면 모든 트래픽이 이전 버전을 가리키도록 하여 배포를 즉시 롤백합니다.

   `alarm_polling_interval`에 정의된 시간 간격(초 단위)이 지난 후에 경보를 계속 모니터링합니다. `canary_interval_seconds`에 정의된 시간 간격이 경과할 때까지 계속 모니터링합니다.
5. `canary_interval_seconds` 중에 알람이 발생한 경우 모든 트래픽을 새 버전으로 이동합니다.
6. 새 버전이 성공적으로 배포되면 `history_max` 파라미터에 지정된 숫자보다 오래된 버전을 모두 삭제합니다.

```
#!/bin/bash
#
# AWS StepFunctions example showing how to create a canary deployment with a
# State Machine Alias and versions.
#
# Requirements: AWS CLI installed and credentials configured.
#
# A canary deployment deploys the new version alongside the old version, while
# routing only a small fraction of the overall traffic to the new version to
# see if there are any errors. Only once the new version has cleared a testing
# period will it start receiving 100% of traffic.
```

```
#
# For a Blue/Green or All at Once style deployment, you can set the
# canary_percentage to 100. The script will immediately shift 100% of traffic
# to the new version, but keep on monitoring the alarms (if any) during the
# canary_interval_seconds time interval. If any alarms raise during this period,
# the script will automatically rollback to the previous version.
#
# Step Functions allows you to keep a maximum of 1000 versions in version history
# for a state machine. This script has a version history deletion mechanism at
# the end, where it will delete any versions older than the limit specified.
#
# For a fuller example, that also demonstrates linear (or rolling) deployments,
# please see
# https://github.com/aws-samples/aws-stepfunctions-examples/blob/main/gradual-deploy/
sfndeploy.py

set -euo pipefail

# ****************************************************************************
# you can safely change the variables in this block to your values
state_machine_name="my-state-machine"
alias_name="alias-1"
region="us-east-1"

# array of cloudwatch alarms to poll during the test period.
# to disable alarm checking, set alarm_names=()
alarm_names=("alarm1" "alarm name with a space")

# true to publish the current revision as the next version before deploy.
# false to deploy the latest version from the state machine's version history.
publish_revision=true

# true to force routing configuration update even if the current routing
# for the alias does not have a 100% routing config.
# false will abandon deploy attempt if current routing config not 100% to a
# single version.
# Be careful when you combine this flag with publish_revision - if you just
# rerun the script you might deploy the newly published revision from the
# previous run.
force=false

# percentage of traffic to route to the new version during the test period
canary_percentage=10
```

```
# how many seconds the canary deployment lasts before full deploy to 100%
canary_interval_seconds=300

# how often to poll the alarms
alarm_polling_interval=60

# how many versions to keep in history. delete versions prior to this.
# set to 0 to disable old version history deletion.
history_max=0
# ******************************************************************************

#######################################
# Update alias routing configuration.
#
# If you don't specify version 2 details, will only create 1 routing entry. In
# this case the routing entry weight must be 100.
#
# Globals:
#   alias_arn
# Arguments:
#   1. version 1 arn
#   2. version 1 weight
#   3. version 2 arn (optional)
#   4. version 2 weight (optional)
#######################################
function update_routing() {
  if [[ $# -eq 2 ]]; then
    local routing_config="[{\"stateMachineVersionArn\": \"$1\", \"weight\":$2}]"
  elif [[ $# -eq 4 ]]; then
    local routing_config="[{\"stateMachineVersionArn\": \"$1\", \"weight\":$2},
 {\"stateMachineVersionArn\": \"$3\", \"weight\":$4}]"
  else
    echo "You have to call update_routing with either 2 or 4 input arguments." >&2
    exit 1
  fi

  ${aws} update-state-machine-alias --state-machine-alias-arn ${alias_arn} --routing-
configuration "${routing_config}"
}

# ******************************************************************************
# pre-run validation
if [[ (("${#alarm_names[@]}" -gt 0)) ]]; then
```

```
  alarm_exists_count=$(aws cloudwatch describe-alarms --alarm-names "${alarm_names[@]}"
 --alarm-types "CompositeAlarm" "MetricAlarm" --query "length([MetricAlarms,
 CompositeAlarms][])" --output text)

  if [[ (("${#alarm_names[@]}" -ne "${alarm_exists_count}")) ]]; then
    echo All of the alarms to monitor do not exist in CloudWatch: $(IFS=,; echo
 "${alarm_names[*]}") >&2
    echo Only the following alarm names exist in CloudWatch:
    aws cloudwatch describe-alarms --alarm-names "${alarm_names[@]}" --alarm-types
 "CompositeAlarm" "MetricAlarm" --query "join(', ', [MetricAlarms, CompositeAlarms]
[].AlarmName)" --output text
    exit 1
  fi
fi

if [[ (("${history_max}" -gt 0)) && (("${history_max}" -lt 2)) ]]; then
  echo The minimum value for history_max is 2. This is the minimum number of older
 state machine versions to be able to rollback in the future. >&2
  exit 1
fi
# ******************************************************************************
# main block follows

account_id=$(aws sts get-caller-identity --query Account --output text)

sm_arn="arn:aws:states:${region}:${account_id}:stateMachine:${state_machine_name}"

# the aws command we'll be invoking a lot throughout.
aws="aws stepfunctions"

# promote the latest revision to the next version
if [[ "${publish_revision}" = true ]]; then
  new_version=$(${aws} publish-state-machine-version --state-machine-arn=$sm_arn --
query stateMachineVersionArn --output text)
  echo Published the current revision of state machine as the next version with arn:
 ${new_version}
else
  new_version=$(${aws} list-state-machine-versions --state-machine-arn ${sm_arn} --max-
results 1 --query "stateMachineVersions[0].stateMachineVersionArn" --output text)
  echo "Since publish_revision is false, using the latest version from the state
 machine's version history: ${new_version}"
fi

# find the alias if it exists
```

```
alias_arn_expected="${sm_arn}:${alias_name}"
alias_arn=$(${aws} list-state-machine-aliases --state-machine-arn
 ${sm_arn} --query "stateMachineAliases[?stateMachineAliasArn==\`
${alias_arn_expected}\`].stateMachineAliasArn" --output text)

if [[ "${alias_arn_expected}" == "${alias_arn}" ]]; then
  echo Found alias ${alias_arn}

  echo Current routing configuration is:
  ${aws} describe-state-machine-alias --state-machine-alias-arn "${alias_arn}" --query
 routingConfiguration
else
  echo Alias does not exist. Creating alias ${alias_arn_expected} and routing 100%
 traffic to new version ${new_version}

  ${aws} create-state-machine-alias --name "${alias_name}" --routing-configuration
 "[{\"stateMachineVersionArn\": \"${new_version}\", \"weight\":100}]"

  echo Done!
  exit 0
fi

# find the version to which the alias currently points (the current live version)
old_version=$(${aws} describe-state-machine-alias --state-machine-alias-arn $alias_arn
 --query "routingConfiguration[?weight==\`100\`].stateMachineVersionArn" --output text)

if [[ -z "${old_version}" ]]; then
  if [[ "${force}" = true ]]; then
    echo Force setting is true. Will force update to routing config for alias to point
 100% to new version.
    update_routing "${new_version}" 100

    echo Alias ${alias_arn} now pointing 100% to ${new_version}.
    echo Done!
    exit 0
  else
    echo Alias ${alias_arn} does not have a routing config entry with 100% of the
 traffic. This means there might be a deploy in progress, so not starting another
 deploy at this time. >&2
    exit 1
  fi
fi

if [[ "${old_version}" == "${new_version}" ]]; then
```

```
  echo The alias already points to this version. No update necessary.
  exit 0
fi

echo Switching ${canary_percentage}% to new version ${new_version}
(( old_weight = 100 - ${canary_percentage} ))
update_routing "${new_version}" ${canary_percentage} "${old_version}" ${old_weight}

echo New version receiving ${canary_percentage}% of traffic.
echo Old version ${old_version} is still receiving ${old_weight}%.

if [[ ${#alarm_names[@]} -eq 0 ]]; then
  echo No alarm_names set. Skipping cloudwatch monitoring.
  echo Will sleep for ${canary_interval_seconds} seconds before routing 100% to new
 version.
  sleep ${canary_interval_seconds}
  echo Canary period complete. Switching 100% of traffic to new version...
else
  echo Checking if alarms fire for the next ${canary_interval_seconds} seconds.

  (( total_wait = canary_interval_seconds + $(date +%s) ))

  now=$(date +%s)
  while [[ ((${now} -lt ${total_wait})) ]]; do
    alarm_result=$(aws cloudwatch describe-alarms --alarm-names "${alarm_names[@]}"
 --state-value ALARM --alarm-types "CompositeAlarm" "MetricAlarm" --query "join(', ',
 [MetricAlarms, CompositeAlarms][].AlarmName)" --output text)

    if [[ ! -z "${alarm_result}" ]]; then
      echo The following alarms are in ALARM state: ${alarm_result}. Rolling back
 deploy. >&2
      update_routing "${old_version}" 100

      echo Rolled back to ${old_version}
      exit 1
    fi

    echo Monitoring alarms...no alarms have triggered.
    sleep ${alarm_polling_interval}
    now=$(date +%s)
  done

  echo No alarms detected during canary period. Switching 100% of traffic to new
 version...
```

```
fi

update_routing "${new_version}" 100

echo Version ${new_version} is now receiving 100% of traffic.

if [[ (("${history_max}" -eq 0 ))]]; then
  echo Version History deletion is disabled. Remember to prune your history, the
 default limit is 1000 versions.
  echo Done!
  exit 0
fi

echo Keep the last ${history_max} versions. Deleting any versions older than that...

# the results are sorted in descending order of the version creation time
version_history=$(${aws} list-state-machine-versions --state-
machine-arn ${sm_arn} --max-results 1000 --query "join(\`\"\\n\"\`,
 stateMachineVersions[].stateMachineVersionArn)" --output text)

counter=0

while read line; do
  ((counter=${counter} + 1))

  if [[ (( ${counter} -gt ${history_max})) ]]; then
    echo Deleting old version ${line}
    ${aws} delete-state-machine-version --state-machine-version-arn ${line}
  fi
done <<< "${version_history}"

echo Done!
```

AWS SDK를 사용하여 새 상태 시스템 버전 배포

aws-stepfunctions-examples의 예제 스크립트에서는 Python용 AWS SDK를 사용하여 트래픽을 점진적으로 상태 시스템의 이전 버전에서 새 버전으로 이동하는 방법을 보여줍니다. 이 예제 스크립트를 사용하거나 요구 사항에 따라 업데이트할 수 있습니다.

스크립트에서는 다음과 같은 배포 전략을 보여줍니다.

- Canary: 트래픽이 두 증분으로 나뉘어 이동합니다.

첫 번째 증분에서는 작은 비율의 트래픽(예: 10%)이 새 버전으로 이동합니다. 두 번째 증분에서는 지정된 시간 간격(초)이 초과되기 전에 남은 트래픽이 새 버전으로 이동합니다. 지정된 시간 간격 동안 CloudWatch 경보가 발생하지 않은 경우에만 남은 트래픽이 새 버전으로 전환됩니다.

- 선형 또는 롤링 - 각 증분 사이의 시간(초)이 같도록 하여 동일한 증분으로 트래픽을 새 버전으로 이동합니다.

  예를 들어 증분 비율을 **600**초 `--interval`을 사용하여 **20**으로 지정하면 이 배포는 새 버전에서 모든 트래픽을 수신할 때까지 600초 간격으로 트래픽을 20%씩 증가합니다.

  CloudWatch 경보가 발생하면 이 배포는 즉시 새 버전을 롤백합니다.
- 한 번에 모두 또는 블루/그린 - 모든 트래픽을 새 버전으로 즉시 이동합니다. 이 배포는 새 버전을 모니터링하고 CloudWatch 경보가 발생하면 자동으로 이전 버전으로 롤백합니다.

AWS CloudFormation을 사용하여 새 상태 시스템 버전 배포

다음 CloudFormation 템플릿 예제에서는 *MyStateMachine* 상태 시스템의 두 버전을 게시합니다. 이러한 두 버전을 모두 가리키는 *PROD* 별칭을 만든 다음 버전 2를 배포합니다.

이 예제에서는 이 버전에서 모든 트래픽을 수신할 때까지 트래픽의 10%가 5분 간격으로 버전 2로 이동합니다. 이 예제에서도 CloudWatch 알람을 설정하는 방법을 보여줍니다. 설정한 경보 중 하나라도 이 ALARM 상태로 전환되면 배포가 실패하고 즉시 롤백됩니다.

```
MyStateMachine:
  Type: AWS::StepFunctions::StateMachine
  Properties:
    Type: STANDARD
    StateMachineName: MyStateMachine
    RoleArn: arn:aws:iam::123456789012:role/myIamRole
    Definition:
      StartAt: PassState
      States:
        PassState:
          Type: Pass
          Result: Result
          End: true

MyStateMachineVersionA:
  Type: AWS::StepFunctions::StateMachineVersion
  Properties:
    Description: Version 1
```

```
    StateMachineArn: !Ref MyStateMachine

MyStateMachineVersionB:
  Type: AWS::StepFunctions::StateMachineVersion
  Properties:
    Description: Version 2
    StateMachineArn: !Ref MyStateMachine

PROD:
  Type: AWS::StepFunctions::StateMachineAlias
  Properties:
    Name: PROD
    Description: The PROD state machine alias taking production traffic.
    DeploymentPreference:
      StateMachineVersionArn: !Ref MyStateMachineVersionB
      Type: LINEAR
      Percentage: 10
      Interval: 5
      Alarms:
        # A list of alarms that you want to monitor. If any of these alarms trigger,
 rollback the deployment immediately by pointing 100 percent of traffic to the previous
 version.
       - !Ref CloudWatchAlarm1
       - !Ref CloudWatchAlarm2
```

Step Functions에서 실행

AWS Step Functions 상태 시스템이 실행되고 해당 작업을 수행하면 상태 시스템 실행이 발생합니다. 각 Step Functions 상태 시스템에는 Step Functions 콘솔에서 또는 AWS SDK, Step Functions API 작업 또는 AWS Command Line Interface(AWS CLI)를 사용하여 시작할 수 있는 동시 실행이 여러 개 있을 수 있습니다. 실행은 JSON 입력을 받고 JSON 출력을 생산합니다. 다음과 같은 방법으로 Step Functions 실행을 시작할 수 있습니다.

- StartExecution API 작업을 직접적으로 호출합니다.
- Step Functions 콘솔에서 새 실행을 시작합니다.
- 이벤트에 대한 응답으로 실행을 시작하려면 Amazon EventBridge를 사용합니다.
- 일정에 따라 상태 시스템 실행을 시작하려면 Amazon EventBridge Scheduler를 사용합니다.
- Amazon API Gateway를 사용하여 실행을 시작합니다.
- Task 상태에서 중첩된 워크플로 실행을 시작합니다.

Step Functions를 사용하는 다양한 방법에 대한 자세한 내용은 개발 옵션을 참조하세요.

작업 상태에서 워크플로 실행 시작

AWS Step Functions는 상태 머신의 Task 상태에서 직접 워크플로 실행을 시작할 수 있습니다. 이를 통해 워크플로를 더 작은 상태 머신으로 나누고 이러한 다른 상태 머신의 실행을 시작할 수 있습니다. 이러한 새로운 워크플로 실행을 시작하면 다음을 수행할 수 있습니다.

- 상위 수준의 워크플로를 하위 수준의 작업별 워크플로우와 분리합니다.
- 별도의 상태 머신을 여러 번 호출하여 반복적인 요소를 피하십시오.
- 빠른 개발을 위해 모듈식 재사용 가능한 워크플로 라이브러리를 만듭니다.
- 복잡성을 줄이고 상태 시스템을보다 쉽게 편집하고 문제를 해결할 수 있습니다.

Step Functions는 자체 API를 통합 서비스로 직접적으로 호출하여 이러한 워크플로 실행을 시작할 수 있습니다. 해당 Task 상태에서 StartExecution API 작업을 호출하고 필요한 매개 변수를 전달하면 됩니다. 모든 서비스 통합 패턴을 사용하여 Step Functions API를 직접적으로 호출할 수 있습니다.

> **Tip**
>
> 중첩된 워크플로 예제를 AWS 계정에 배포하려면 모듈 13 - 중첩된 Express 워크플로를 참조하세요.

상태 시스템의 새 실행을 시작하려면 다음 예제와 유사한 Task 상태를 사용합니다.

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution",
   "Parameters":{
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Input":{
         "Comment":"Hello world!"
      },
   },
   "Retry":[
      {
        "ErrorEquals":[
            "StepFunctions.ExecutionLimitExceeded"
        ]
```

```
      }
   ],
   "End":true
}
```

이 Task 상태는 HelloWorld 상태 머신의 새로운 실행을 시작하고 JSON 주석을 입력으로 전달합니다.

> **Note**
>
> StartExecution API 작업 할당량은 시작할 수 있는 실행 수를 제한할 수 있습니다. StepFunctions.ExecutionLimitExceeded의 Retry를 사용하여 실행이 시작되었는지 확인하십시오. 다음 항목을 참조하십시오.
>
> - API 작업 제한과 관련된 할당량
> - Step Functions에서 오류 처리

워크플로 실행을 연결합니다

시작된 워크플로 실행을 시작된 실행과 연결하려면 컨텍스트 개체의 실행 ID를 실행 입력으로 전달하십시오. 실행 중인 Task 상태에서 컨텍스트 개체의 ID에 액세스할 수 있습니다. 매개 변수 이름에 .$를 추가하고 $$.Execution.Id를 사용하여 컨텍스트 개체에서 ID를 참조하여 실행 ID를 전달하십시오.

```
"AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
```

실행을 시작할 때 AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID로 이름이 지정된 특수 매개 변수를 사용할 수 있습니다. 포함된 경우 이 연결은 Step Functions 콘솔의 단계 세부 정보 섹션에 링크를 제공합니다. 제공되면 실행 시작부터 시작된 워크플로 실행까지 워크플로 실행을 쉽게 추적할 수 있습니다. 이전 예제를 사용하여 다음과 같이 실행 ID를 HelloWorld 상태 머신의 시작된 실행과 연결합니다.

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution",
   "Parameters":{
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
```

```
      "Input": {
        "Comment": "Hello world!",
        "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
       }
   },
   "End":true
}
```

자세한 내용은 다음을 참조하세요.

- 다른 서비스와 함께 사용
- 파라미터를 서비스 API에 전달
- 컨텍스트 객체 액세스
- AWS Step Functions

AWS Step Functions에서 Amazon EventBridge 스케줄러 사용

Amazon EventBridge 스케줄러는 하나의 중앙 관리형 서비스에서 작업을 생성, 실행 및 관리할 수 있는 서버리스 스케줄러입니다. EventBridge 스케줄러를 사용하면 반복 패턴에 대해 cron 및 rate 표현식을 사용하여 일정을 만들거나 일회성 간접 호출을 구성할 수 있습니다. 전송을 위한 유연한 기간을 설정하고, 재시도 제한을 정의하고, 실패한 API 간접 호출의 최대 보존 시간을 설정할 수 있습니다.

예를 들어 EventBridge 스케줄러를 사용하면 보안 관련 이벤트가 발생할 때 또는 데이터 처리 작업을 자동화하기 위해 일정에 따라 상태 시스템 실행을 시작할 수 있습니다.

이 페이지에서는 EventBridge 스케줄러를 사용하여 일정에 따라 Step Functions 상태 시스템 실행을 시작하는 방법을 설명합니다.

주제

- 실행 역할 설정
- 일정 생성
- 관련 리소스

실행 역할 설정

새 일정을 생성할 때 EventBridge 스케줄러에 사용자를 대신하여 대상 API 작업을 간접적으로 호출할 수 있는 권한이 있어야 합니다. 실행 역할을 사용하여 EventBridge 스케줄러에 이러한 권한을

부여합니다. 일정의 실행 역할에 연결하는 권한 정책은 필요한 권한을 정의합니다. 이러한 권한은 EventBridge 스케줄러가 간접적으로 호출하려는 대상 API에 따라 달라집니다.

다음 절차와 같이 EventBridge 스케줄러 콘솔을 사용하여 일정을 생성하면 EventBridge 스케줄러가 선택한 대상을 기준으로 실행 역할을 자동으로 설정합니다. EventBridge 스케줄러 SDK, AWS CLI 또는 AWS CloudFormation 중 하나를 사용하여 일정을 생성하려면 EventBridge 스케줄러가 대상을 간접적으로 호출하는 데 필요한 권한을 부여하는 기존 실행 역할이 있어야 합니다. 일정에 대한 실행 역할을 수동으로 설정하는 방법에 대한 자세한 내용은 EventBridge 스케줄러 사용 설명서의 Setting up an execution role을 참조하세요.

일정 생성

콘솔을 사용하여 일정 생성

1. https://console.aws.amazon.com/scheduler/home에서 Amazon EventBridge 스케줄러 콘솔을 엽니다.
2. 일정 페이지에서 일정 생성을 선택합니다.
3. 일정 세부 정보 지정 페이지의 일정 이름 및 설명 섹션에서 다음을 수행합니다.

   a. 일정 이름에 일정의 이름을 입력합니다. 예: **`MyTestSchedule`**.

   b. (선택 사항) 설명에 일정에 대한 설명을 입력합니다. 예: **`My first schedule`**.

   c. 일정 그룹 드롭다운 목록에서 일정 그룹을 선택합니다. 그룹이 없는 경우 기본값을 선택합니다. 일정 그룹을 생성하려면 자체 일정 생성을 선택합니다.

   일정 그룹을 사용하여 일정 그룹에 태그를 추가합니다.

4. • 일정 옵션을 선택합니다.

발생	수행할 작업	
일회성 일정 일회성 일정은 사용자가 지정하는 날짜와 시간에 한 번만 대상을 간접적으로 호출합니다.	날짜 및 시간에 대해 다음을 수행합니다. • `YYYY/MM/DD` 형식으로 유효한 날짜를 입력합니다. • 24시간 `hh:mm` 형식으로 타임스탬프를 입력합니다.	

발생	수행할 작업	
	• 시간대에서 시간대를 선택하세요.	
반복되는 일정 반복 일정은 cron 표현식 또는 rate 표현식을 사용하여 지정한 속도로 대상을 간접적으로 호출합니다.	a. Schedule type(일정 유형)에서 다음 중 하나를 수행합니다. • cron 식을 사용하여 일정을 정의하려면 Cron-based schedule(Cron 기반 일정)을 선택하고 cron 식을 입력합니다. • rate 식을 사용하여 일정을 정의하려면 Rate-based schedule(Rate 기반 일정)을 선택하고 rate 식을 입력합니다. cron 및 rate 표현식에 대한 자세한 내용은 EventBridge Scheduler 사용 설명서의 EventBridge 스케줄러의 스케줄 유형을 참조하세요. b. 유연한 기간에서 끄기를 선택하여 옵션을 끄거나 미리 정의된 기간 중 하나를 선택합니다. 예를 들어, 15분을 선택하고 1시간에 한 번씩 대상을 간접적으로 호출하도록 반복 일정을 설정하면 일정은 매시간 시작 후 15분 이내에 실행됩니다.	

5. (선택 사항) 이전 단계에서 반복 일정을 선택한 경우 기간 섹션에서 다음을 수행합니다.

   a. 시간대에서 시간대를 선택합니다.

   b. 시작 날짜 및 시간에 YYYY/MM/DD 형식으로 유효한 날짜를 입력한 다음 24시간 hh:mm 형식으로 타임스탬프를 지정합니다.

   c. 종료 날짜 및 시간에 YYYY/MM/DD 형식으로 유효한 날짜를 입력한 다음 24시간 hh:mm 형식으로 타임스탬프를 지정합니다.

6. 다음(Next)을 선택합니다.

7. 대상 선택 페이지에서 EventBridge 스케줄러가 간접적으로 호출하는 AWS API 작업을 선택합니다.

   a. AWS Step Functions StartExecution을 선택합니다.

   b. StartExecution 섹션에서 상태 시스템을 선택하거나 새 상태 머신 생성을 선택합니다.

      현재 일정에 따라 동기 Express 워크플로를 실행할 수 없습니다.

   c. 실행의 JSON 페이로드를 입력합니다. 상태 시스템에 JSON 페이로드가 필요하지 않더라도 다음 예제와 같이 여전히 JSON 형식의 입력을 포함해야 합니다.

```
{
    "Comment": "sampleJSONData"
}
```

8. 다음(Next)을 선택합니다.

9. 설정 페이지에서 다음 작업을 수행하십시오.

   a. 일정을 켜려면 일정 상태에서 일정 활성화를 토글합니다.

   b. 일정에 대한 재시도 정책을 구성하려면 재시도 정책 및 데드-레터 큐(DLQ)에서 다음을 수행합니다.

      - 재시도를 토글합니다.
      - 최대 이벤트 수명에 EventBridge 스케줄러가 처리되지 않은 이벤트를 보관해야 하는 최대 시간과 분을 입력합니다.
      - 최대 시간은 24시간입니다.
      - 최대 재시도 횟수에는 대상이 오류를 반환할 경우 EventBridge 스케줄러가 일정을 재시도하는 최대 횟수를 입력합니다.

        최댓값은 185회입니다.

재시도 정책을 사용하면 일정이 대상을 간접적으로 호출하지 못할 경우 EventBridge 스케줄러가 일정을 다시 실행합니다. 구성된 경우 일정에 대한 최대 보존 기간과 재시도 횟수를 설정해야 합니다.

c. EventBridge 스케줄러가 전송되지 않은 이벤트를 저장하는 위치를 선택합니다.

DLQ(Dead Letter Queue) 옵션	수행할 작업
저장 안 함	[None]을 선택합니다.
일정을 생성하는 위치와 같은 AWS 계정에 이벤트 저장	a. 내 AWS 계정의 Amazon SQS 대기열을 DLQ로 선택을 선택합니다. b. Amazon SQS 대기열의 Amazon 리소스 이름(ARN)을 선택합니다.
일정을 생성하는 위치와 다른 AWS 계정에 이벤트 저장	a. 다른 AWS 계정의 Amazon SQS 대기열을 DLQ로 지정을 선택합니다. b. Amazon SQS 대기열의 Amazon 리소스 이름(ARN)을 입력합니다.

d. 고객 관리형 키를 사용하여 대상 입력을 암호화하려면 암호화에서 암호화 설정 사용자 지정(고급)을 선택합니다.

이 옵션을 선택하는 경우 기존 KMS 키 ARN을 입력하거나 AWS KMS key 생성을 선택하여 AWS KMS 콘솔로 이동합니다. EventBridge 스케줄러가 저장 데이터를 암호화하는 방법에 대한 자세한 내용은 Amazon EventBridge 스케줄러 사용 설명서의 Encryption at rest를 참조하세요.

e. EventBridge 스케줄러가 새 실행 역할을 생성하도록 하려면 이 일정에 대한 새 역할 생성을 선택합니다. 그런 다음 역할 이름을 입력합니다. 이 옵션을 선택하면 EventBridge 스케줄러가 템플릿 대상에 필요한 필수 권한을 역할에 연결합니다.

10. 다음(Next)을 선택합니다.
11. 일정 검토 및 생성 페이지에서 일정의 세부 정보를 검토합니다. 각 섹션에서 편집을 선택하여 해당 단계로 돌아가서 세부 정보를 편집합니다.
12. 일정 생성을 선택합니다.

    일정 페이지에서 새 일정과 기존 일정 목록을 볼 수 있습니다. 상태 열에서 새 일정이 활성화됨 상태인지 확인합니다.

EventBridge 스케줄러에서 상태 시스템을 간접적으로 호출했는지 확인하려면 상태 시스템의 Amazon CloudWatch 로그를 확인합니다.

관련 리소스

EventBridge 스케줄러에 대한 자세한 내용은 다음을 참조하세요.

- EventBridge 스케줄러 사용 설명서
- EventBridge 스케줄러 API 참조
- EventBridge 스케줄러 요금

콘솔에서의 표준 및 Express 워크플로 실행

상태 시스템을 만들 때 유형을 표준 또는 Express 중 하나로 선택합니다. 상태 시스템의 기본 유형은 표준입니다. 유형이 표준인 상태 시스템을 표준 워크플로라고 하고 유형이 Express인 상태 시스템을 Express 워크플로라고 합니다.

표준 및 Express 워크플로 모두에서 Amazon States Language를 사용하여 상태 시스템을 정의합니다. 상태 시스템 실행은 선택한 유형에 따라 다르게 작동합니다.

> **Important**
>
> 상태 시스템을 만든 후에는 선택한 유형을 변경할 수 없습니다.

표준 및 Express 워크플로에 대한 자세한 내용은 표준 워크플로와 Express 워크플로 비교를 참조하세요.

표준 워크플로 실행 내역은 Step Functions에 기록되지만 Express 워크플로 실행 내역은 Step Functions에 기록되지 않습니다. 익스프레스 워크플로 실행 기록을 기록하려면 Amazon에 로그를 전

송하도록 구성해야 CloudWatch 합니다. 자세한 설명은 CloudWatch Logs를 사용하여 로깅 섹션을 참조하세요.

로깅이 Express 워크플로에 구성되면 Step Functions 콘솔에서 실행을 볼 수 있습니다. Express 워크플로 실행과 표준 워크플로 실행을 볼 수 있는 콘솔 환경은 다음과 같은 차이점과 제한을 제외하면 비슷합니다.

> **Note**
>
> Express 워크플로의 실행 데이터는 CloudWatch Logs Insights를 사용하여 표시되므로 로그를 스캔하면 요금이 부과됩니다. 기본적으로 로그 그룹에는 지난 3시간 동안에 완료된 실행만 나열됩니다. 더 많은 실행 이벤트가 포함된 더 큰 시간 범위를 지정하면 비용이 증가합니다. 자세한 내용은 CloudWatch 요금 페이지의 로그 탭 아래에 있는 벤디드 로그를 참조하십시오. CloudWatch Logs를 사용하여 로깅

내용

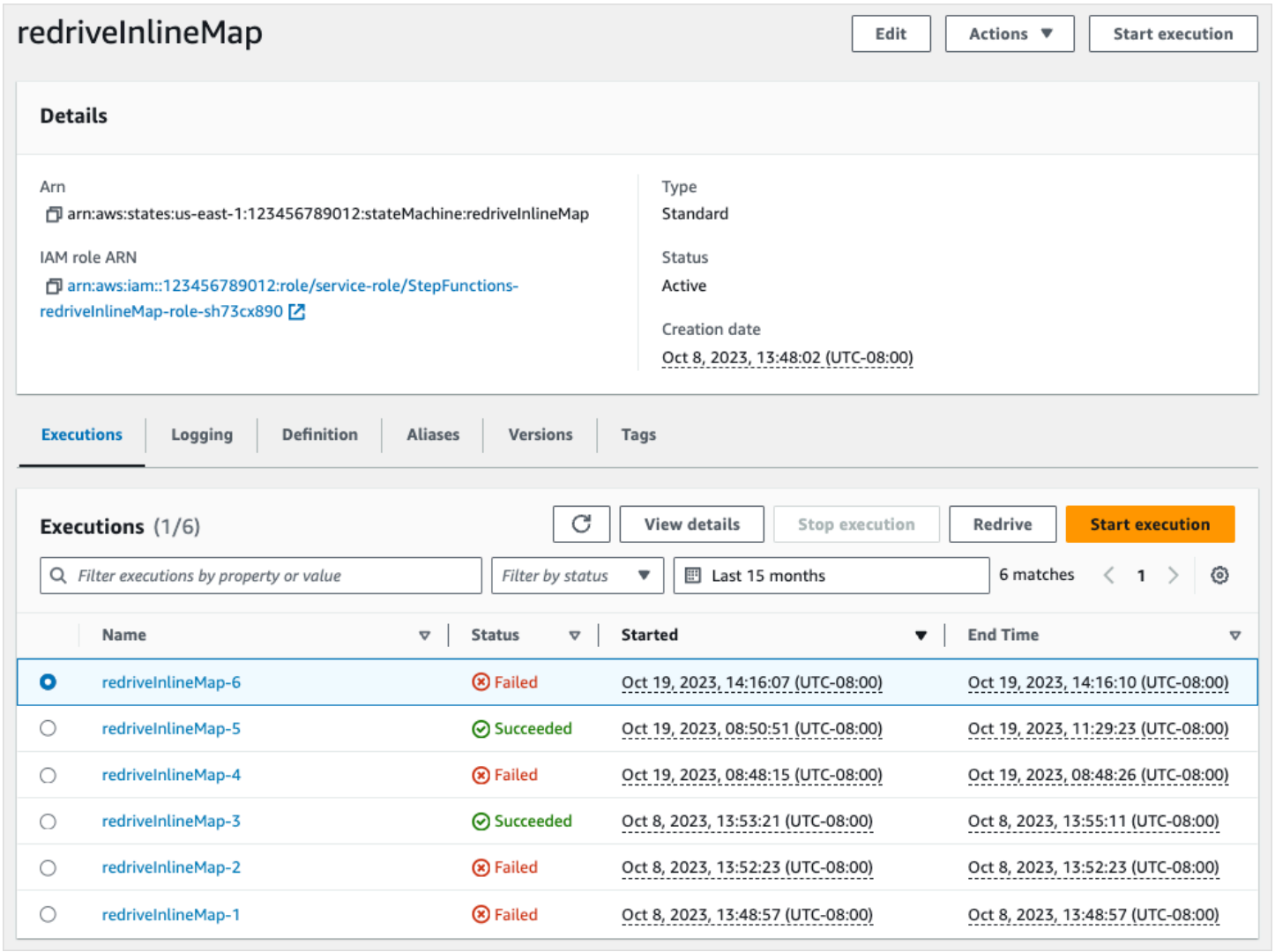
콘솔 환경 차이

모든 표준 및 Express 워크플로의 경우 Step Functions 콘솔의 상태 시스템 세부 정보 페이지에서 상태 시스템 및 IAM 역할 ARN과 같은 세부 정보를 볼 수 있습니다.

상태 시스템 세부 정보 페이지의 실행 탭 아래에서도 상태 시스템 실행 내역 목록을 확인할 수 있습니다. 속성 또는 값을 기준으로 실행 필터링 상자를 사용하여 선택한 상태 시스템의 특정 실행, 버전 또는 별칭을 검색합니다. 전체 드롭다운을 사용하여 실행 내역을 상태를 기준으로 필터링합니다. 실행 내역을 선택하고 세부 정보 보기 버튼을 선택하여 실행 세부 정보 페이지를 열 수도 있습니다.

표준 워크플로

표준 워크플로 실행 내역은 지난 90일 동안에 완료된 실행에 대해 항상 사용 가능합니다.



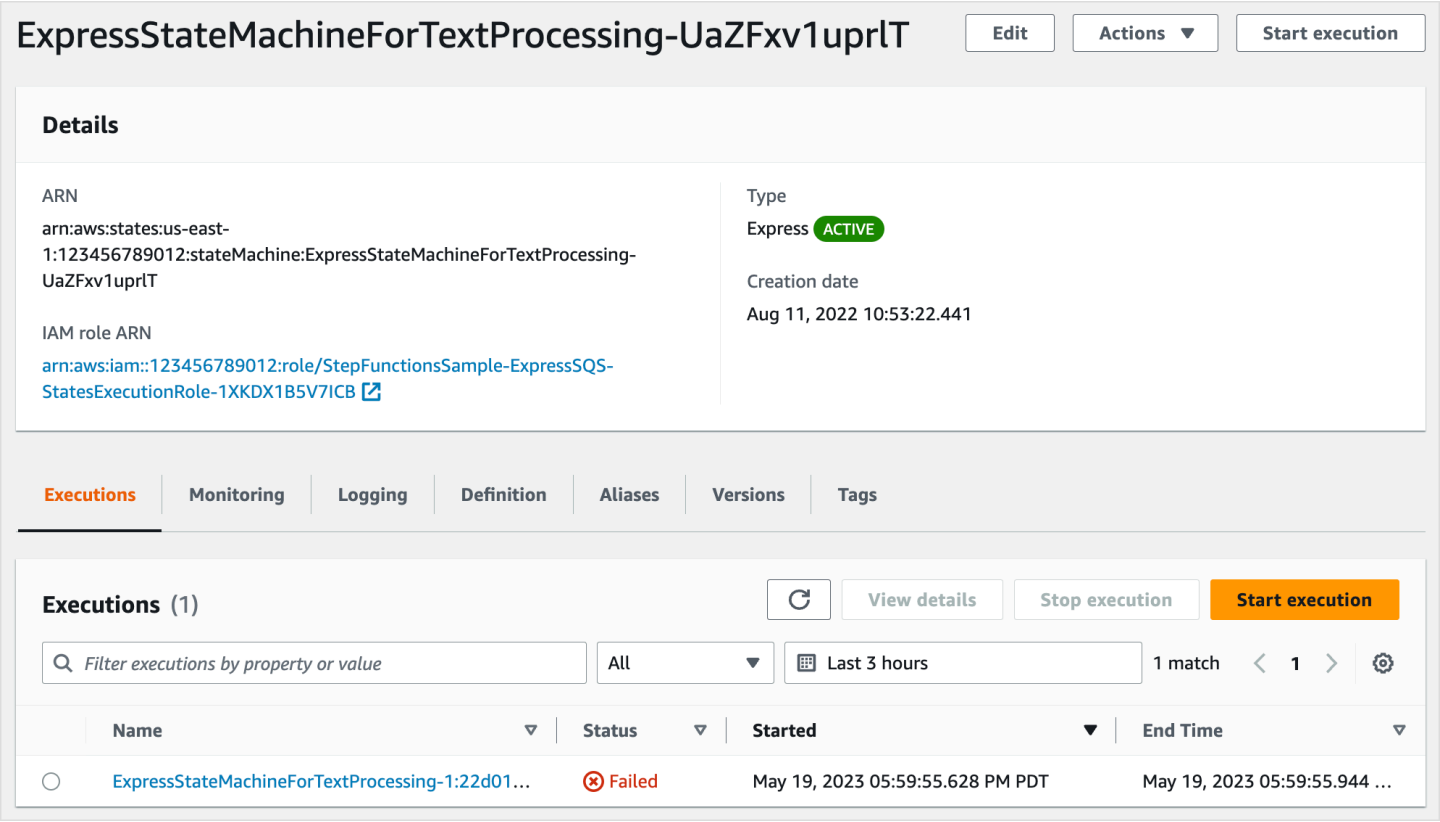
Express 워크플로

Express 워크플로의 실행 기록을 표시하기 위해 Step Functions 콘솔은 Logs 로그 그룹을 통해 수집된 CloudWatch 로그 데이터를 검색합니다.

또한 Express 워크플로 실행을 보려면 새 콘솔 환경을 활성화해야 합니다. 이렇게 하려면 실행 탭의 배너 안에 표시된 활성화 버튼을 선택합니다. 이 버튼을 선택하면 다시 표시되지 않습니다.

> **Tip**
>
> 콘솔 환경을 활성화하거나 비활성화하려면 Express 실행 내역 활성화 토글 버튼을 사용하세요.

기본적으로 최근 3시간 동안에 완료된 실행의 내역이 제공됩니다. 이 시간 범위를 조정하거나 사용자 지정 범위를 지정할 수 있습니다. 더 많은 실행 이벤트가 포함된 더 큰 시간 범위를 지정하면 로그 스캔 비용이 증가합니다. 자세한 내용은 CloudWatch 가격 책정 페이지의 로그 탭 아래에 있는 벤디드 로그를 참조하십시오. CloudWatch Logs를 사용하여 로깅



Express 워크플로 실행 보기에 대한 고려 사항 및 제한 사항

Step Functions 콘솔에서 Express 워크플로 실행을 볼 때 다음 고려 사항과 제한 사항에 유의하세요.

- Express 워크플로우 실행 세부 정보의 가용성은 Amazon CloudWatch Logs에 의존합니다.
- 로깅 수준이 ERROR 또는 FATAL인 경우 Express 워크플로 실행 세부 정보 일부가 제공됩니다.
- 업데이트된 후에는 이전 실행의 상태 시스템 정의를 볼 수 없습니다.

Express 워크플로우 실행 세부 정보의 가용성은 Amazon CloudWatch Logs에 의존합니다.

> **Note**
>
> Express 워크플로 실행을 보도록 새 콘솔 환경을 활성화하지 않으면 Step Functions 콘솔에서 실행 내역과 해당 실행 세부 정보가 제공되지 않습니다. 새 콘솔 환경을 활성화하려면 실행 탭의 배너 안에 표시된 활성화 버튼을 선택하세요.

Express 워크플로의 경우 실행 기록 및 세부 실행 정보가 CloudWatch Logs Insights를 통해 수집됩니다. 이 정보는 상태 머신을 생성할 때 지정하는 CloudWatch 로그 로그 그룹에 보관됩니다. 상태 시스템 실행 내역은 Step Functions 콘솔의 실행 탭에 표시됩니다. 상태 시스템의 각 실행에 대한 세부 정보는 선택한 실행의 실행 세부 정보 페이지에 표시됩니다.

> **Warning**
>
> 익스프레스 워크플로의 CloudWatch 로그를 삭제하면 해당 로그가 실행 탭에 나열되지 않습니다.

모든 실행 이벤트 유형을 로깅할 수 있도록 기본 로그 수준인 ALL을 사용하는 것이 좋습니다. 기존 상태 시스템을 편집할 때 필요에 따라 로그 수준을 업데이트할 수 있습니다. 자세한 내용은 CloudWatch Logs를 사용하여 로깅 및 로그 수준 섹션을 참조하세요.

로깅 수준이 ERROR 또는 FATAL인 경우 Express 워크플로 실행 세부 정보 일부가 제공됩니다.

기본적으로 Express 워크플로 실행의 로깅 수준은 ALL로 설정됩니다. 로그 수준을 변경해도 완료된 실행의 실행 내역과 실행 세부 정보는 영향을 받지 않습니다. 하지만 모든 새 실행은 업데이트된 로그 수준에 따라 로그를 내보냅니다. 자세한 내용은 CloudWatch Logs를 사용하여 로깅 및 로그 수준 섹션을 참조하세요.

예를 들어 로그 수준을 ALL에서 ERROR 또는 FATAL로 변경하면 Step Functions 콘솔의 실행 탭에는 실패한 실행만 나열됩니다. 콘솔은 이벤트 보기 탭에 실패한 상태 시스템 단계에 대한 이벤트 세부 정보만 보여줍니다.

모든 실행 이벤트 유형을 로깅할 수 있도록 기본 로그 수준인 ALL을 사용하는 것이 좋습니다. 상태 시스템을 편집할 때 필요에 따라 로그 수준을 업데이트할 수 있습니다.

업데이트된 후에는 이전 실행의 상태 시스템 정의를 볼 수 없습니다.

과거 실행에 대한 상태 시스템 정의는 Express 워크플로에 저장되지 않습니다. 상태 시스템 정의를 변경하면 최신 정의를 사용하는 실행에 대한 상태 시스템 정의만 볼 수 있습니다.

예를 들어 상태 시스템 정의에서 단계를 하나 이상 제거하면 Step Functions는 정의와 이전 실행 이벤트 간의 불일치를 감지합니다. 이전 정의는 Express 워크플로에 저장되지 않으므로 Step Functions는 이전 버전의 상태 시스템 정의에서 실행된 실행에 대한 상태 시스템 정의를 표시할 수 없습니다. 따라서 이전 버전의 상태 시스템 정의에서 실행된 실행에는 실행 입력 및 출력, 정의, 그래프 보기 및 테이블 보기 탭을 사용할 수 없습니다.

Step Functions 콘솔에서 실행 보기 및 디버깅

Step Functions 콘솔의 실행 세부 정보 페이지에는 표준 및 Express 워크플로의 이전 상태 시스템 실행과 진행 중인 상태 시스템 실행에 대한 정보가 표시됩니다. 이 정보는 대시보드 형식으로 표시됩니다. 예를 들어 상태 시스템의 Amazon States Language 정의, 실행 상태, ARN 및 총 상태 전환 수를 찾을 수 있습니다. 또한 상태 시스템의 모든 개별 상태에 대한 실행 세부 정보를 볼 수 있습니다.

내용

실행 세부 정보 페이지 - 인터페이스 개요

실행 세부 정보 페이지에서 표준 및 Express 워크플로 모두의 진행 중인 상태 시스템 실행과 이전 상태 시스템 실행에 대한 세부 정보를 찾을 수 있습니다. 실행을 시작하는 동안에 실행 ID를 지정한 경우 이

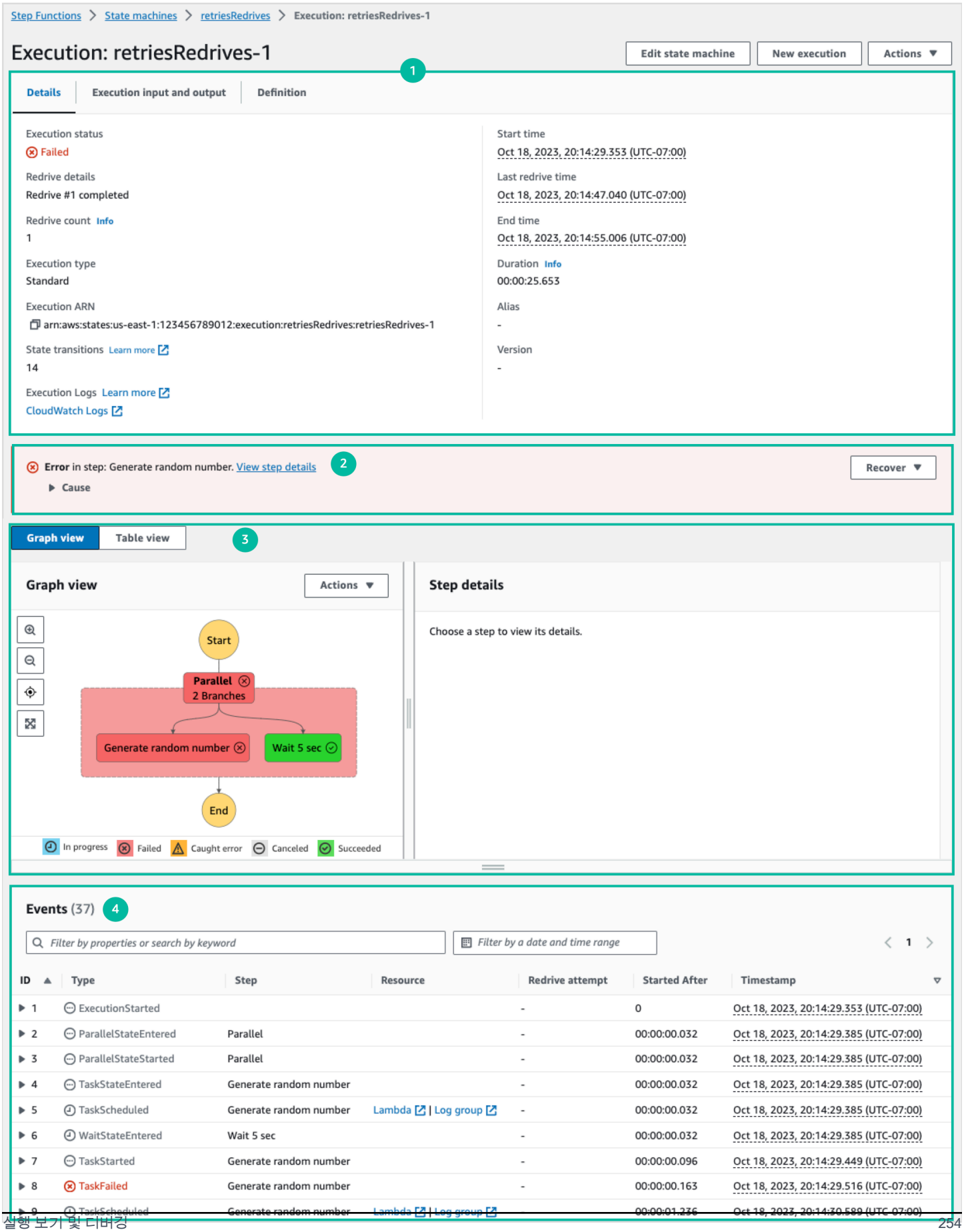
페이지 제목은 해당 실행 ID로 표시됩니다. 그렇지 않으면 제목은 Step Functions에서 자동으로 생성하는 고유한 실행 ID로 지정됩니다.

실행 지표 외에도 실행 세부 정보 페이지에는 상태 시스템과 해당 실행을 관리하기 위한 다음과 같은 옵션이 있습니다.

Button	이 버튼을 선택하면 수행되는 내용
상태 머신 편집	상태 시스템의 Amazon States Language 정의를 편집합니다.
새 실행	상태 시스템의 새 실행을 시작합니다.
작업	다음 옵션 중에서 선택합니다. • 실행 중지 - 진행 중인 실행을 중지합니다. 완료된 실행에는 이 옵션을 사용할 수 없습니다. • Redrive - 지난 14일 동안 성공적으로 완료되지 않은 표준 워크플로 실행을 Redrive합니다. 여기에는 실패, 중단 또는 시간 초과된 실행이 포함됩니다. 자세한 설명은 실행 Redriving 섹션을 참조하세요. • 내보내기 - 다른 사용자와 공유하거나 오프라인 분석을 수행하도록 실행 세부 정보를 JSON 형식으로 내보냅니다. • 피드백 전송 - 인터페이스에 대한 피드백을 공유합니다.

> **버전 또는 별칭으로 시작되는 실행 보기**
>
> Step Functions 콘솔에서 버전이나 별칭으로 시작되는 실행을 볼 수도 있습니다. 자세한 내용은 버전 및 별칭의 실행 목록을 참조하세요.

실행 세부 정보 콘솔 페이지에는 다음 섹션이 포함됩니다.


Step Functions > State machines > retriesRedrives > Execution: retriesRedrives-1
Execution: retriesRedrives-1
Edit state machine
New execution
Actions
1
Details
Execution input and output
Definition
Execution status
Failed
Redrive details
Redrive #1 completed
Redrive count Info
1
Execution type
Standard
Execution ARN
arn:aws:states:us-east-1:123456789012:execution:retriesRedrives:retriesRedrives-1
State transitions Learn more
14
Execution Logs Learn more
CloudWatch Logs
Start time
Oct 18, 2023, 20:14:29.353 (UTC-07:00)
Last redrive time
Oct 18, 2023, 20:14:47.040 (UTC-07:00)
End time
Oct 18, 2023, 20:14:55.006 (UTC-07:00)
Duration Info
00:00:25.653
Alias
-
Version
-
Error in step: Generate random number. View step details
2
Cause
Recover
Graph view
Table view
3
Graph view
Actions
Start
Parallel
2 Branches
Generate random number
Wait 5 sec
End
In progress
Failed
Caught error
Canceled
Succeeded
Step details
Choose a step to view its details.
Events (37)
4
Filter by properties or search by keyword
Filter by a date and time range
1
ID
Type
Step
Resource
Redrive attempt
Started After
Timestamp
1 ExecutionStarted - 0 Oct 18, 2023, 20:14:29.353 (UTC-07:00)
2 ParallelStateEntered Parallel - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
3 ParallelStateStarted Parallel - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
4 TaskStateEntered Generate random number - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
5 TaskScheduled Generate random number Lambda | Log group - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
6 WaitStateEntered Wait 5 sec - 00:00:00.032 Oct 18, 2023, 20:14:29.385 (UTC-07:00)
7 TaskStarted Generate random number - 00:00:00.096 Oct 18, 2023, 20:14:29.449 (UTC-07:00)
8 TaskFailed Generate random number - 00:00:00.163 Oct 18, 2023, 20:14:29.516 (UTC-07:00)
9 TaskScheduled Generate random number Lambda | Log group - 00:00:01.236 Oct 18, 2023, 20:14:30.589 (UTC-07:00)

1. 실행 요약
2. 오류 메시지
3. 보기 모드
4. 단계 세부 정보
5. 이벤트

실행 요약

실행 요약 섹션은 실행 세부 정보 페이지 상단에 나타납니다. 이 섹션에서는 워크플로의 실행 세부 정보를 간략하게 설명합니다. 이 정보는 다음과 같은 탭 3개로 구분됩니다.

세부 정보

실행 상태, ARN, 실행 시작 및 종료 시간의 타임스탬프와 같은 정보가 표시됩니다. 상태 시스템 실행 중에 발생한 총 상태 전환 수도 볼 수 있습니다. 상태 머신에 대한 추적 또는 로그를 활성화한 경우 X-Ray trace 맵 및 Amazon CloudWatch Execution Logs 링크를 볼 수도 있습니다.

상태 시스템 실행이 다른 상태 시스템에서 시작된 경우 이 탭에서 부모 상태 시스템의 링크를 볼 수 있습니다.

상태 시스템 실행이 redriven된 경우 이 탭에는 redrive 관련 정보(예: Redrive 횟수)가 표시됩니다.

실행 입력 및 출력

스테이트 머신 실행 입력 및 출력을 side-by-side 표시합니다.

정의

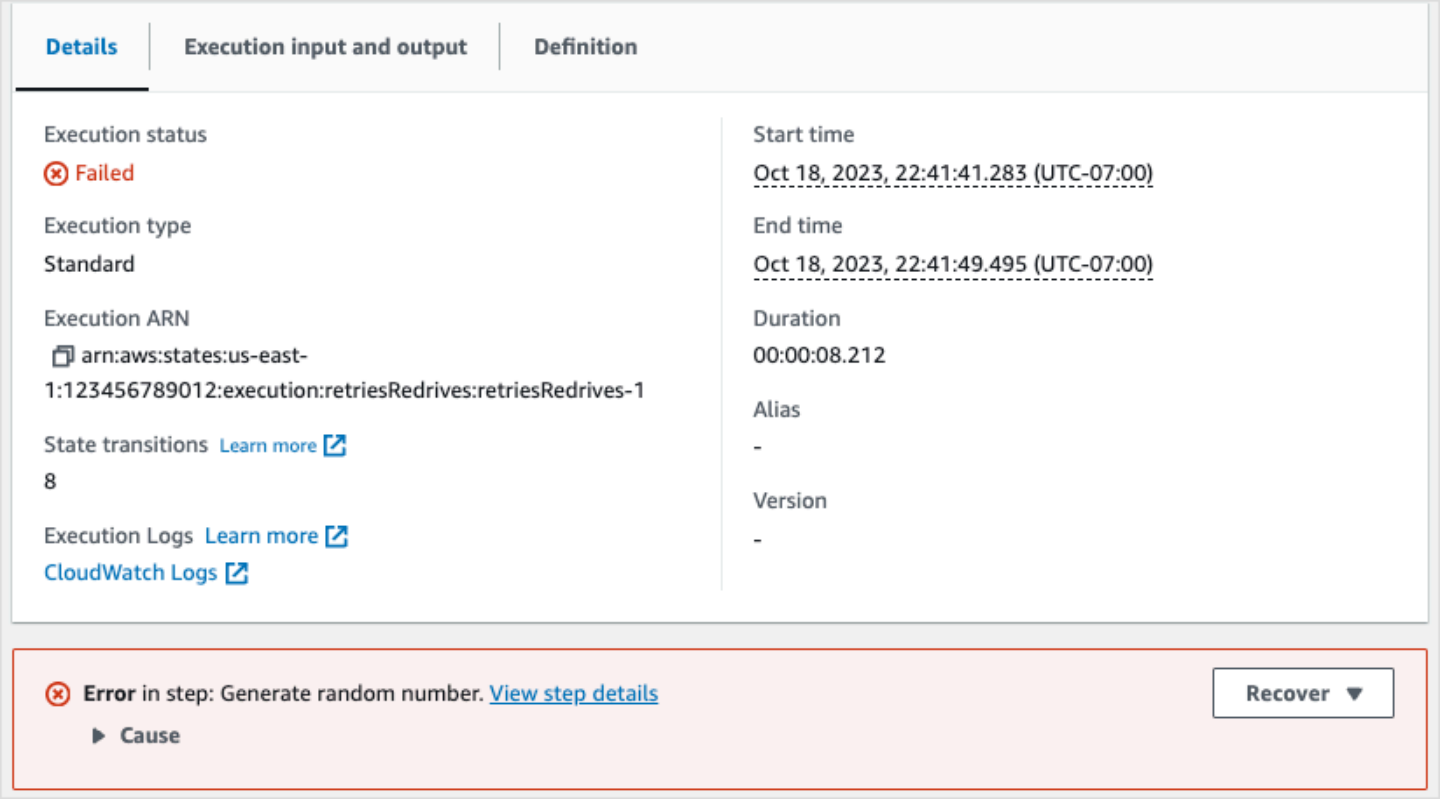
상태 시스템의 Amazon States Language 정의를 표시합니다.

오류 메시지

상태 시스템 실행이 실패하면 실행 세부 정보 페이지에 오류 메시지가 표시됩니다. 오류 메시지에서 원인 또는 단계 세부 정보 보기를 선택하면 실행 실패 원인이나 오류가 발생한 단계를 볼 수 있습니다.

단계 세부 정보 보기를 선택하면 Step Functions에서 단계 세부 정보, 그래프 보기 및 테이블 보기 탭에 오류가 발생한 단계를 강조 표시합니다. 단계가 재시도를 정의한 Task, Map 또는 Parallel 상태이면 단계 세부 정보 창에 단계의 재시도 탭이 표시됩니다. 또한 실행을 redriven한 경우 단계 세부 정보 창의 재시도 및 redrives 탭에서 재시도 및 redrive 실행 세부 정보를 볼 수 있습니다.

이 오류 메시지의 복구 드롭다운 버튼에서 실패한 실행을 redrive하거나 새 실행을 시작할 수 있습니다. 자세한 설명은 실행 Redriving 섹션을 참조하세요.



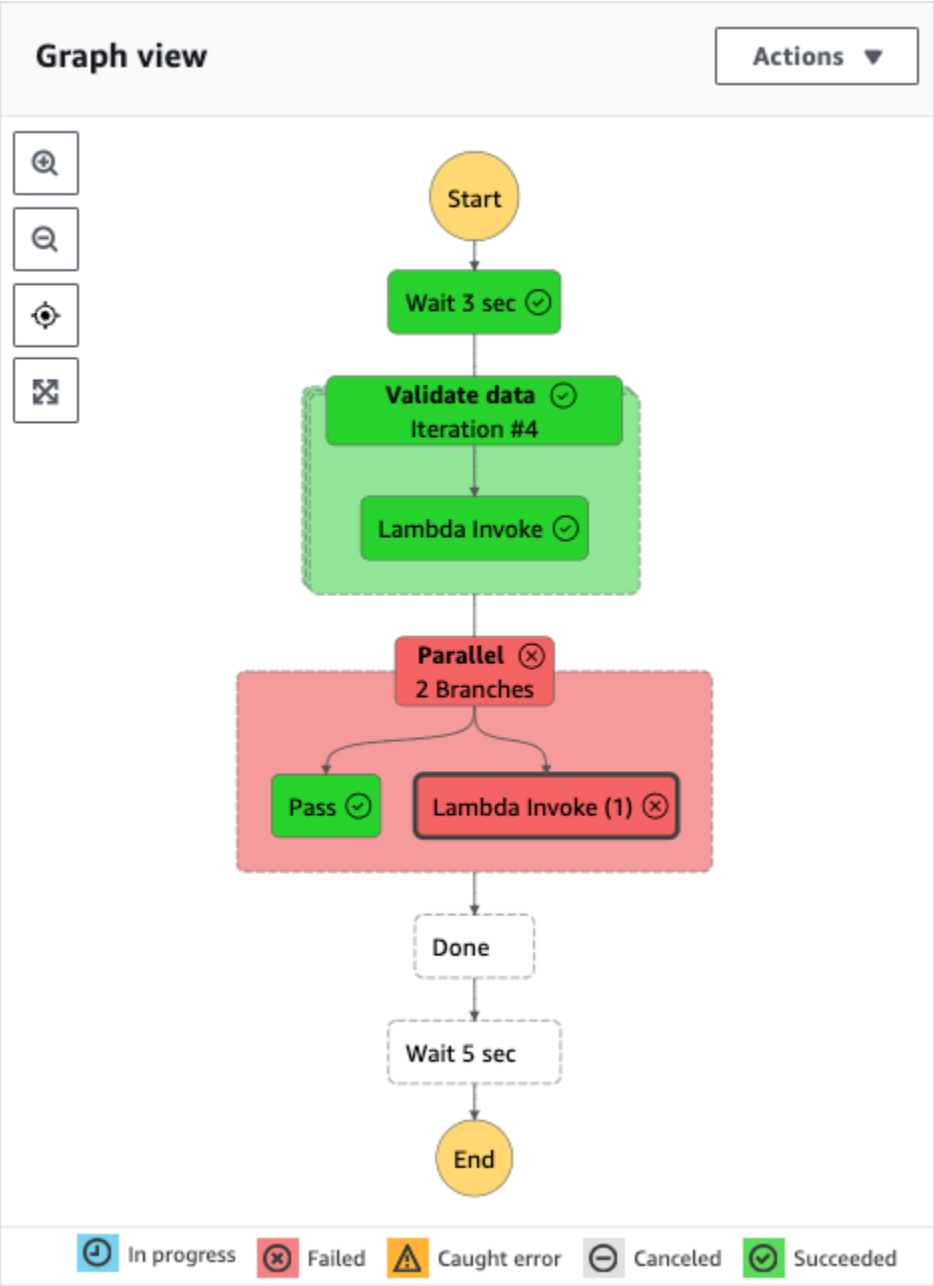
보기 모드

보기 모드 섹션에는 상태 시스템에 대한 두 가지 시각화가 포함되어 있습니다. 워크플로의 그래픽 표현, 워크플로의 상태를 간략하게 보여주는 표 또는 상태 시스템 실행과 관련된 이벤트 목록을 볼 수 있습니다.

> **Note**
>
> 해당 내용을 보려면 탭을 선택합니다.

Graph view

그래프 보기 모드에는 워크플로의 그래픽 표현이 표시됩니다. 하단에는 상태 시스템 실행 상태를 나타내는 범례가 포함되어 있습니다. 전체 워크플로를 확대, 축소, 가운데 정렬하거나 전체 화면 모드에서 워크플로를 볼 수 있게 해주는 버튼도 포함되어 있습니다.



이 보기에서는 워크플로의 아무 단계나 선택하여 단계 세부 정보 구성 요소에서 해당 실행에 대한 세부 정보를 볼 수 있습니다. 그래프 보기에서 단계를 선택한 경우 테이블 보기에도 해당 단계가 표시됩니다. 반대의 경우도 마찬가지입니다. 테이블 보기에서 단계를 선택한 경우 그래프 보기에도 같은 단계가 표시됩니다.

상태 시스템에 `Map` 상태, `Parallel` 상태 또는 둘 다 포함된 경우 그래프 보기에서 워크플로의 상태 시스템 이름을 볼 수 있습니다. 또한 `Map` 상태의 경우 그래프 보기를 사용하면 Map 상태 실행 데이터의 다양한 반복 간에 이동할 수 있습니다. 예를 들어 Map 상태에 반복 5회가 있고 3번째와 4번째 반복의 실행 데이터를 보려는 경우 다음을 수행합니다.

1. 반복 데이터를 보려는 Map 상태를 선택합니다.
2. 3번째 반복의 경우 맵 반복 뷰어의 드롭다운 목록에서 #2를 선택합니다. 이는 반복이 0부터 계산되기 때문입니다. 마찬가지로 Map 상태의 4번째 반복의 경우 드롭다운 목록에서 #3을 선택합니다.

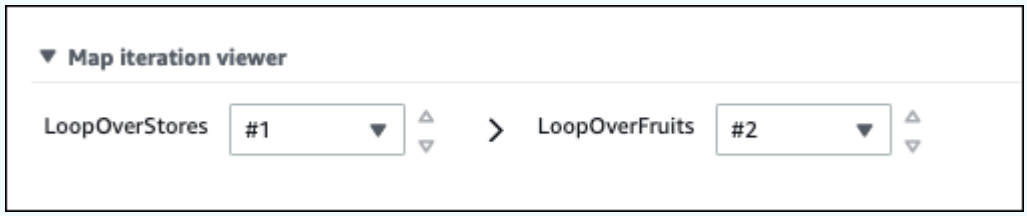
   또는

   △

   및

   ▽

   컨트롤을 사용하여 Map 상태의 여러 반복으로 이동할 수 있습니다.

   > **Note**
   >
   > 상태 시스템에 중첩된 Map 상태가 포함된 경우 다음 예제와 같이 상위 및 하위 Map 상태 반복에 대한 드롭다운 목록이 표시됩니다.
   >
   > 
   >

3. (선택 사항) Map 상태 반복 하나 이상에서 실행을 실패하거나 실행이 중지된 경우 드롭다운 목록의 실패 또는 중단됨 아래에서 해당 반복 번호를 선택하여 해당 데이터를 볼 수 있습니다.

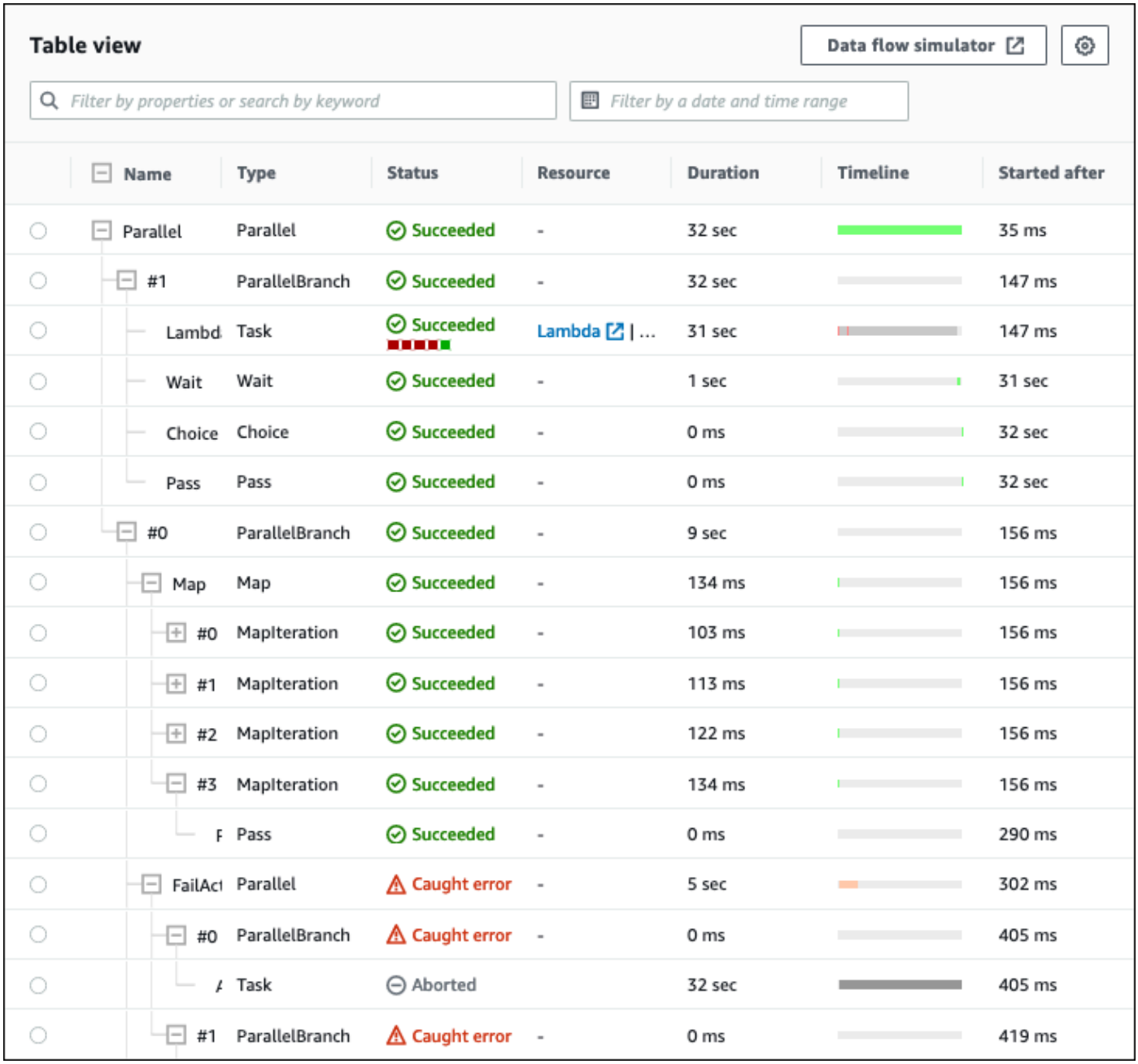
마지막으로 내보내기 및 레이아웃 버튼을 사용하여 워크플로 그래프를 SVG 또는 PNG 이미지로 내보낼 수 있습니다. 워크플로의 가로 보기 또는 세로 보기로 전환할 수도 있습니다.

Table view

테이블 보기 모드에는 워크플로 상태가 표 형식으로 표시됩니다. 이 보기 모드에서는 이름, 사용된 리소스 (예: AWS Lambda 함수) 의 이름, 상태가 성공적으로 실행되었는지 여부 등 워크플로에서 실행된 각 상태의 세부 정보를 볼 수 있습니다.

이 보기에서는 워크플로의 아무 상태나 선택하여 단계 세부 정보 구성 요소에서 해당 실행에 대한 세부 정보를 볼 수 있습니다. 테이블보기에서 단계를 선택한 경우 그래프 보기에도 해당 단계가 표시됩니다. 반대의 경우도 마찬가지입니다. 그래프 보기에서 단계를 선택한 경우 테이블 보기에도 같은 단계가 표시됩니다.

보기에 필터를 적용하여 테이블 보기 모드에서 표시되는 데이터 양을 제한할 수도 있습니다. 상태 또는 Redrive 시도와 같은 특정 속성에 대한 필터를 만들 수 있습니다. 자세한 설명은 자습서: Step Functions 콘솔을 사용하여 상태 시스템 실행 검사 섹션을 참조하세요.

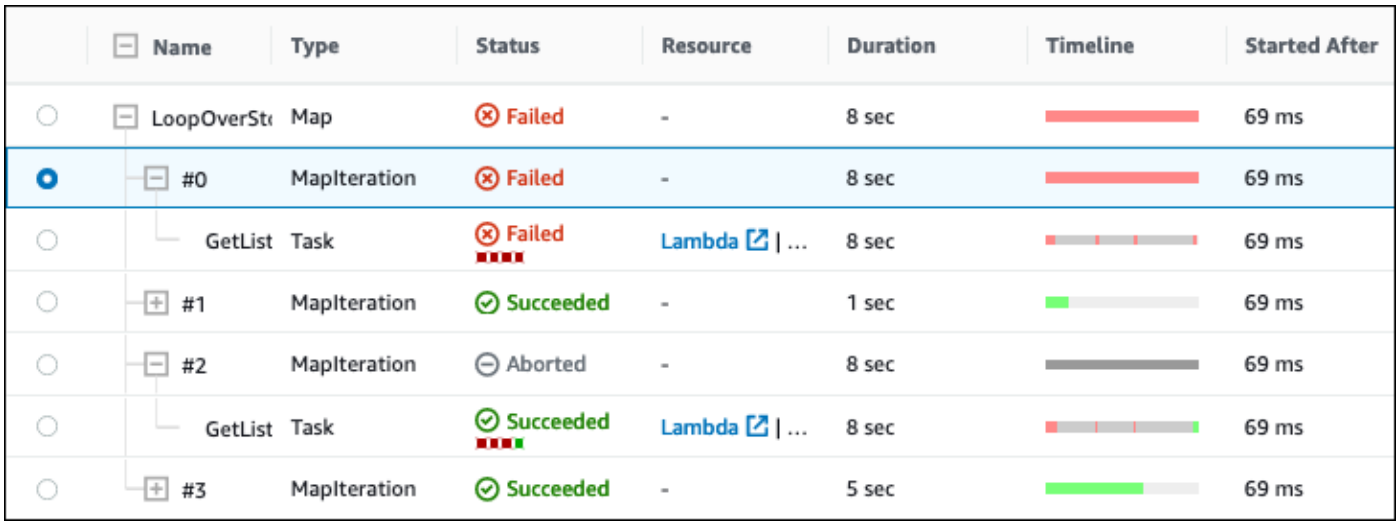


기본적으로 이 모드에는 이름, 유형, 상태, 리소스 및 다음 후에 시작됨 열이 표시됩니다. 기본 설정 대화 상자를 사용하여 보려는 열을 구성할 수 있습니다. 이 대화 상자에서 선택한 내용은 다시 변경될 때까지 향후 상태 시스템 실행을 위해 유지됩니다.

타임라인 열을 추가하면 전체 실행의 런타임과 관련된 각 상태의 실행 기간이 표시됩니다. 이는 색상으로 구분된 선형 타임라인으로 표시됩니다. 이를 통해 특정 상태 실행에서 성능 관련 문제를 식

별할 수 있습니다. 타임라인의 각 상태를 색상으로 구분한 세그먼트를 통해 진행 중, 실패 또는 중단됨과 같은 상태 실행 상태를 식별할 수 있습니다.

예를 들어 상태 시스템의 상태에 대한 실행 재시도를 정의한 경우 이러한 재시도는 타임라인에 표시됩니다. 빨간색 세그먼트는 실패한 `Retry` 시도를 나타내고 밝은 회색 세그먼트는 각 시도 `BackoffRate` 간의 `Retry`를 나타냅니다.



상태 시스템에 `Map` 상태, `Parallel` 상태 또는 둘 다 포함된 경우 테이블 보기에서 워크플로의 해당 상태 이름을 볼 수 있습니다. `Map` 및 `Parallel` 상태의 경우 테이블 보기 모드는 반복 및 병렬 브랜치에 대한 실행 데이터를 트리 보기 내의 노드로 표시합니다. 단계 세부 정보 섹션에서 이러한 상태의 각 노드를 선택하여 개별 세부 정보를 볼 수 있습니다. 예를 들어 상태 실패 원인인 특정 Map 상태 반복에 대한 데이터를 검토할 수 있습니다. Map 상태의 노드를 확장한 다음 상태 열에서 각 반복에 대한 상태를 봅니다.

단계 세부 정보

그래프 보기 또는 테이블 보기에서 상태를 선택하면 오른쪽에 단계 세부 정보 섹션이 열립니다. 이 섹션에는 선택한 상태에 대한 자세한 정보를 제공하는 다음 탭이 있습니다.

입력

선택한 상태의 입력 세부 정보를 표시합니다. 입력에 오류가 있으면 탭 헤더에 ⚠ 가 표시됩니다. 또한 이 탭에서 오류 원인을 볼 수 있습니다.

고급 보기 토글 버튼을 선택하여 데이터가 선택한 상태를 통과할 때의 입력 데이터 전송 경로를 확인할 수도 있습니다. 이를 통해 `InputPath`, `Parameters`, `ResultSelector`, `OutputPath` 및

ResultPath와 같은 필드 하나 이상이 데이터에 적용될 때 입력이 처리된 방식을 식별할 수 있습니다.

출력

선택한 상태의 출력을 표시합니다. 출력에 오류가 있으면 탭 헤더에  가 표시됩니다. 또한 이 탭에서 오류 원인을 볼 수 있습니다.

고급 보기 토글 버튼을 선택하여 데이터가 선택한 상태를 통과할 때의 출력 데이터 전송 경로를 확인할 수도 있습니다. 이를 통해 InputPath, Parameters, ResultSelector, OutputPath 및 ResultPath와 같은 필드 하나 이상이 데이터에 적용될 때 입력이 처리된 방식을 식별할 수 있습니다.

세부 정보

상태 유형, 실행 상태 및 실행 기간과 같은 정보를 표시합니다.

예를 들어 리소스를 사용하는 Task 상태의 경우 이 탭은 리소스 호출을 위한 리소스 정의 페이지 및 Amazon CloudWatch 로그 페이지로 연결되는 링크를 제공합니다. AWS Lambda 또한 Task 상태의 TimeoutSeconds 및 HeartbeatSeconds 필드에 대한 값(지정된 경우)도 보여줍니다.

Map 상태의 경우 이 탭에는 Map 상태의 총 반복 횟수에 대한 정보가 표시됩니다. 이터레이션은 실패, 중단, 성공 또는 으로 분류됩니다. InProgress

정의

선택한 상태에 해당하는 Amazon States Language 정의를 표시합니다.

재시도

> **Note**
>
> 이 탭은 상태 시스템 Task 또는 Parallel 상태에서 Retry 필드를 정의한 경우에만 나타납니다.

원래 실행 시도에서 선택한 상태에 대한 초기 재시도 및 후속 재시도를 표시합니다. 초기 시도와 이후의 모든 실패 시도의 경우 유형 옆에 있는  를 선택하면 드롭다운 상자에 나타나는 실패 이유를 볼 수 있습니다. 재시도가 성공하면 드롭다운 상자에 나타나는 출력을 볼 수 있습니다.

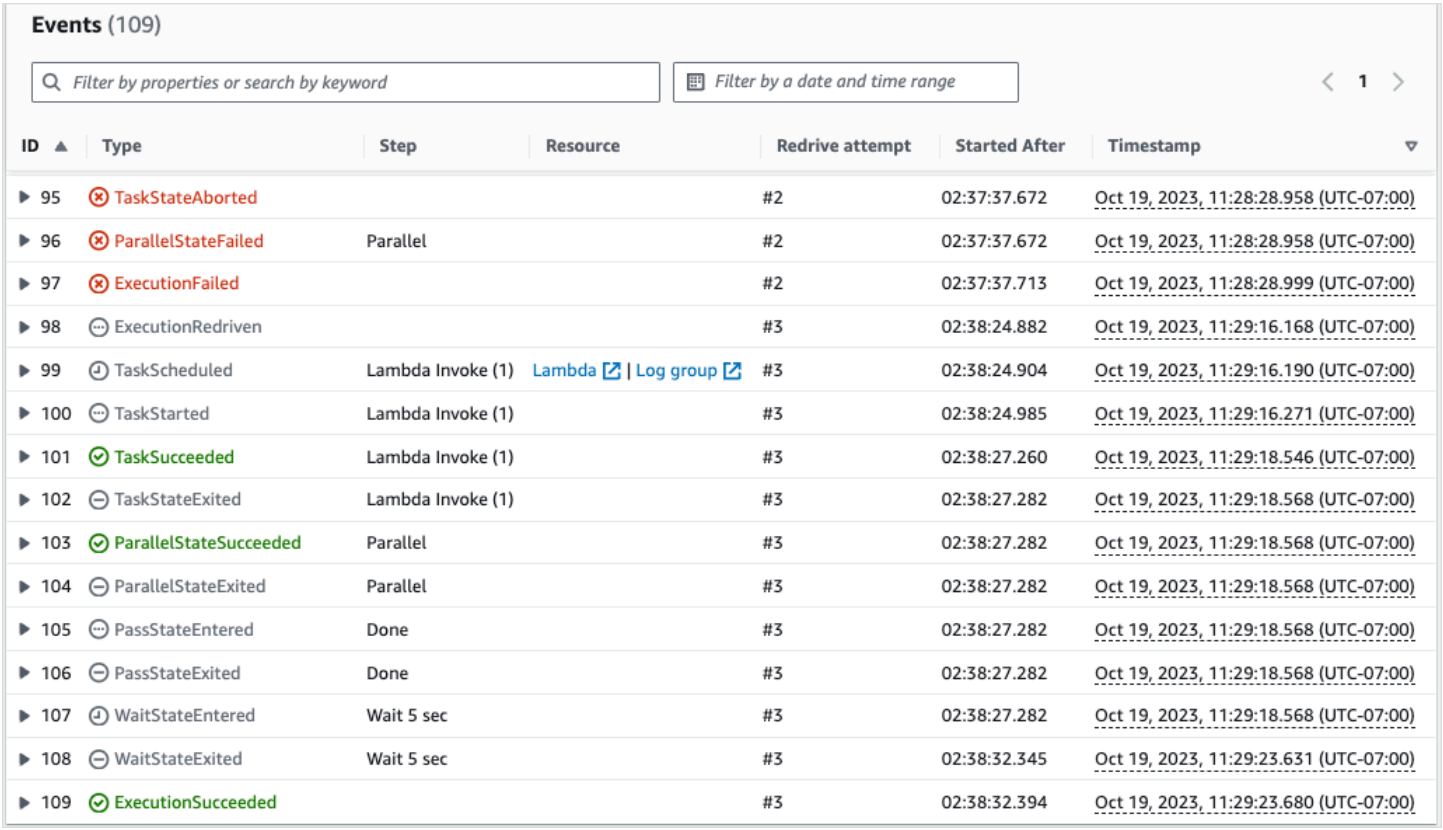
실행을 redriven한 경우 이 탭 헤더에는 재시도 및 redrives 이름이 표시되고 각 redrive에 대한 재시도 세부 정보가 표시됩니다.

이벤트

실행에서 선택한 상태와 관련된 이벤트의 필터링된 목록을 표시합니다. 이 탭에 표시되는 정보는 이벤트 테이블에 표시되는 전체 실행 이벤트 내역에 포함됩니다.

이벤트

이벤트 테이블에는 선택한 실행의 전체 내역이 여러 페이지에 걸친 이벤트 목록으로 표시됩니다. 각 페이지에는 이벤트가 최대 25개까지 포함됩니다. 또한 이 섹션에는 총 이벤트 수가 표시되므로 최대 이벤트 내역 수인 이벤트 25,000개를 초과했는지 확인할 수 있습니다.



기본적으로 이벤트 테이블의 결과는 이벤트의 타임스탬프를 기준으로 오름차순으로 표시됩니다. 타임스탬프 열 헤더를 클릭하여 실행 이벤트 내역 정렬 방식을 내림차순으로 변경할 수 있습니다.

이벤트 테이블에서 각 이벤트는 실행 상태를 나타내도록 색상으로 구분되어 있습니다. 예를 들어 실패한 이벤트는 빨간색으로 표시됩니다. 이벤트에 대한 추가 세부 정보를 보려면 이벤트 ID 옆에 있는 ▶ 를 선택합니다. 열리면 이벤트 세부 정보에 이벤트의 입력, 출력 및 리소스 간접 호출이 표시됩니다.

또한 이벤트 테이블에서 필터를 적용하여 표시되는 실행 이벤트 내역 결과를 제한할 수 있습니다. ID 또는 Redrive 시도와 같은 속성을 선택할 수 있습니다. 자세한 내용은 자습서: Step Functions 콘솔을 사용하여 상태 시스템 실행 검사을(를) 참조하세요.

자습서: Step Functions 콘솔을 사용하여 상태 시스템 실행 검사

이 자습서에서는 실행 세부 정보 페이지에 표시된 실행 정보를 검사하고 실행 실패 원인을 보는 방법을 알아봅니다. 그런 다음 다양한 Map 상태 실행 반복에 액세스하는 방법을 알아봅니다. 마지막으로 테이블 보기에서 열을 구성하고 관심 있는 정보만 표시되도록 적절한 필터를 적용하는 방법을 알아봅니다.

이 자습서에서는 과일 세트 가격을 구하는 표준 유형 상태 시스템을 만듭니다. 이를 위해 스테이트 머신은 네 가지 과일의 무작위 목록, 각 과일의 가격, 과일의 평균 비용을 반환하는 세 가지 AWS Lambda 함수를 사용합니다. Lambda 함수는 과일 가격이 임계값보다 작거나 같은 경우 오류가 발생하도록 설계되었습니다.

> **Note**
>
> 다음 절차에는 표준 워크플로 실행 세부 정보를 검사하는 방법에 대한 지침이 포함되어 있지만 Express 워크플로 실행 세부 정보도 검사할 수 있습니다. 표준 및 Express 워크플로 유형의 실행 세부 정보 간의 차이점은 콘솔에서의 표준 및 Express 워크플로 실행을 참조하세요.

내용

1단계: 필수 Lambda 함수 만들기 및 테스트

1. Lambda 콘솔을 열고 1단계: Lambda 함수 생성 섹션의 1~4단계를 수행합니다. Lambda 함수 **`GetListOfFruits`**의 이름을 지정해야 합니다.
2. Lambda 함수를 만든 후 페이지 오른쪽 상단에 표시된 함수의 Amazon 리소스 이름(ARN)을 복사합니다. ARN을 복사하려면  를 클릭합니다. 다음은 예제 ARN입니다. 여기서 *`function-name`*은 Lambda 함수 이름입니다(이 경우 `GetListOfFruits`).

```
arn:aws:lambda:us-east-1:123456789012:function:function-name
```

3. Lambda 함수의 다음 코드를 페이지의 코드 소스 영역에 복사합니다. GetListOfFruits

```
function getRandomSubarray(arr, size) {
    var shuffled = arr.slice(0), i = arr.length, temp, index;
    while (i--) {
        index = Math.floor((i + 1) * Math.random());
        temp = shuffled[index];
        shuffled[index] = shuffled[i];
        shuffled[i] = temp;
    }
    return shuffled.slice(0, size);
}

exports.handler = async function(event, context) {

    const fruits = ['Abiu','Açaí','Acerola','Ackee','African
 cucumber','Apple','Apricot','Avocado','Banana','Bilberry','Blackberry','Blackcurrant','Jos


     const errorChance = 45;

    const waitTime = Math.floor( 100 * Math.random() );

    await new Promise( r => setTimeout(() => r(), waitTime));

    const num = Math.floor( 100 * Math.random() );
    // const num = 51;
     if (num <= errorChance) {
         throw(new Error('Error'));
     }

    return getRandomSubarray(fruits, 4);
};
```

4. 배포를 선택한 다음 테스트를 선택하여 변경 사항을 배포하고 Lambda 함수 출력을 확인합니다.
5. 다음 단계에 따라 각각 **GetFruitPrice** 및 **CalculateAverage**라는 Lambda 함수 2개를 추가로 만듭니다.

   a. 다음 코드를 GetFruitPriceLambda 함수의 코드 소스 영역에 복사합니다.

```
exports.handler = async function(event, context) {

    const errorChance = 0;
    const waitTime = Math.floor( 100 * Math.random() );

    await new Promise( r => setTimeout(() => r(), waitTime));

    const num = Math.floor( 100 * Math.random() );
    if (num <= errorChance) {
        throw(new Error('Error'));
    }

    return Math.floor(Math.random()*100)/10;
};
```

b. 다음 코드를 CalculateAverageLambda 함수의 코드 소스 영역에 복사합니다.

```
function getRandomSubarray(arr, size) {
    var shuffled = arr.slice(0), i = arr.length, temp, index;
    while (i--) {
        index = Math.floor((i + 1) * Math.random());
        temp = shuffled[index];
        shuffled[index] = shuffled[i];
        shuffled[i] = temp;
    }
    return shuffled.slice(0, size);
}

const average = arr => arr.reduce( ( p, c ) => p + c, 0 ) / arr.length;

exports.handler = async function(event, context) {
        const errors = [
        "Error getting data from DynamoDB",
        "Error connecting to DynamoDB",
        "Network error",
        "MemoryError - Low memory"
        ]

    const errorChance = 0;

    const waitTime = Math.floor( 100 * Math.random() );
```

```
    await new Promise( r => setTimeout(() => r(), waitTime));

    const num = Math.floor( 100 * Math.random() );
    if (num <= errorChance) {
        throw(new Error(getRandomSubarray(errors, 1)[0]));
    }

    return average(event);
};
```

c. 이 두 Lambda 함수의 ARN을 복사한 다음 배포하고 테스트해야 합니다.

2단계: 상태 시스템 만들기 및 실행

Step Functions 콘솔을 사용하여 1단계에서 만든 Lambda 함수를 간접적으로 호출하는 상태 시스템을 만듭니다. 이 상태 시스템에는 Map 상태 3개가 정의되어 있습니다. 각 Map 상태에는 Lambda 함수 중 하나를 간접적으로 호출하는 Task 상태가 포함되어 있습니다. 또한 각 Task 상태에 Retry 필드가 정의되며 상태마다 재시도 횟수가 정의됩니다. Task 상태에서 런타임 오류가 발생하면 해당 Task에 정의된 재시도 횟수까지 다시 실행됩니다.

1. Step Functions 콘솔을 열고 코드로 워크플로 작성을 선택합니다.

> **Important**
>
> 상태 머신이 이전에 생성한 Lambda 함수와 동일한 AWS 계정 및 지역에 속하는지 확인하십시오.

2. 유형의 경우 기본 선택인 표준을 그대로 유지합니다.
3. 다음 Amazon States Language 정의를 복사하여 정의 아래에 붙여넣습니다. 표시된 ARN을 앞서 만든 Lambda 함수의 ARN으로 교체해야 합니다.

```
{
    "StartAt": "LoopOverStores",
    "States": {
        "LoopOverStores": {
            "Type": "Map",
            "Iterator": {
                "StartAt": "GetListOfFruits",
                "States": {
                    "GetListOfFruits": {
                        "Type": "Task",
```

```
                        "Resource": "arn:aws:states:::lambda:invoke",
                        "OutputPath": "$.Payload",
                        "Parameters": {
                            "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:GetListofFruits:$LATEST",
                            "Payload": {
                                "storeName.$": "$"
                            }
                        },
                        "Retry": [
                            {
                                "ErrorEquals": [
                                    "States.ALL"
                                ],
                                "IntervalSeconds": 2,
                                "MaxAttempts": 1,
                                "BackoffRate": 1.3
                            }
                        ],
                        "Next": "LoopOverFruits"
                    },
                    "LoopOverFruits": {
                        "Type": "Map",
                        "Iterator": {
                            "StartAt": "GetFruitPrice",
                            "States": {
                                "GetFruitPrice": {
                                    "Type": "Task",
                                    "Resource": "arn:aws:states:::lambda:invoke",
                                    "OutputPath": "$.Payload",
                                    "Parameters": {
                                        "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:GetFruitPrice:$LATEST",
                                        "Payload": {
                                            "fruitName.$": "$"
                                        }
                                    },
                                    "Retry": [
                                        {
                                            "ErrorEquals": [
                                                "States.ALL"
                                            ],
                                            "IntervalSeconds": 2,
                                            "MaxAttempts": 3,
```

```
                                        "BackoffRate": 1.3
                                    }
                                ],
                                "End": true
                            }
                        }
                    },
                    "ItemsPath": "$",
                    "End": true
                }
            }
        },
        "ItemsPath": "$.stores",
        "Next": "LoopOverStoreFruitsPrice",
        "ResultPath": "$.storesFruitsPrice"
    },
    "LoopOverStoreFruitsPrice": {
        "Type": "Map",
        "End": true,
        "Iterator": {
            "StartAt": "CalculateAverage",
            "States": {
                "CalculateAverage": {
                    "Type": "Task",
                    "Resource": "arn:aws:states:::lambda:invoke",
                    "OutputPath": "$.Payload",
                    "Parameters": {
                        "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:Calculate-average:$LATEST",
                        "Payload.$": "$"
                    },
                    "Retry": [
                        {
                            "ErrorEquals": [
                                "States.ALL"
                            ],
                            "IntervalSeconds": 2,
                            "MaxAttempts": 2,
                            "BackoffRate": 1.3
                        }
                    ],
                    "End": true
                }
            }
```

```
            },
            "ItemsPath": "$.storesFruitsPrice",
            "ResultPath": "$.storesPriceAverage",
            "MaxConcurrency": 1
        }
    }
}
```

4. 상태 시스템 이름을 입력합니다. 이 페이지의 다른 옵션을 기본 선택 사항으로 유지하고 상태 시스템 생성을 선택합니다.
5. 제목이 상태 시스템 이름인 페이지를 엽니다. 4단계: 상태 시스템 실행 섹션의 1~4단계를 수행하되 실행 입력으로는 다음 데이터를 사용합니다.

```
{
    "stores": [
      "Store A",
      "Store B",
      "Store C",
      "Store D"
    ]
}
```

3단계: 상태 시스템 실행 세부 정보 보기

제목이 실행 ID인 페이지에서 실행 결과를 검토하고 오류를 디버깅할 수 있습니다.

1. (선택 사항) 실행 세부 정보 페이지에 표시된 탭 중에서 선택하여 각 탭에 있는 정보를 확인합니다. 예를 들어 상태 시스템 입력과 실행 출력을 보려면 실행 요약 섹션에서 실행 입력 및 출력을 선택합니다.
2. 상태 시스템 실행이 실패하면 오류 메시지에서 원인 또는 단계 세부 정보 표시를 선택합니다. 오류에 대한 세부 정보는 단계 세부 정보 섹션에 표시됩니다. 오류가 발생한 단계 (이름이 지정된 Task GetListofFruits상태) 가 그래프 보기와 테이블 보기에서 강조 표시되어 있는 것을 알 수 있습니다.

> **Note**
>
> GetListofFruits단계가 Map 상태 내에 정의되어 있고 단계가 성공적으로 실행되지 않았으므로 **Map**상태 단계가 실패로 표시됩니다.

4단계: 다양한 보기 모드 살펴보기

선호하는 모드를 선택하여 상태 시스템 워크플로나 실행 이벤트 내역을 볼 수 있습니다. 이러한 보기 모드에서 수행할 수 있는 일부 작업은 다음과 같습니다.

그래프 보기 — 다양한 **Map** 상태 반복으로 전환

Map 상태에 반복 5회가 있고 3번째와 4번째 반복의 실행 세부 정보를 보려는 경우 다음을 수행합니다.

1. 반복 데이터를 보려는 Map 상태를 선택합니다.
2. 맵 반복 뷰어에서 보려는 반복을 선택합니다. 반복은 0부터 계산됩니다. 반복 5회 중 3번째 반복을 선택하려면 Map 상태 이름 옆에 있는 드롭다운 목록에서 #2를 선택합니다.

> **Note**
>
> 상태 시스템에 중첩된 Map 상태가 포함된 경우 Step Functions는 부모 및 하위 Map 상태 반복을 별도의 드롭다운 목록 2개로 표시합니다.
>
> 
>

3. (선택 사항) Map 상태 반복 하나 이상이 실행되지 않았거나 중단된 상태에서 중지된 경우 실패한 반복에 대한 세부 정보를 볼 수 있습니다. 이러한 세부 정보를 확인하려면 드롭다운 목록의 실패 또는 중단됨에서 영향을 받는 반복 번호를 선택합니다.

테이블 보기 — 다양한 **Map** 상태 반복으로 전환

Map 상태에 반복 5회가 있고 반복 번호 3 및 4의 실행 세부 정보를 보려는 경우 다음을 수행합니다.

1. 다른 반복 데이터를 보려는 Map 상태를 선택합니다.
2. Map 상태 반복이 표시된 트리 뷰에서 반복 번호 3에 해당하는 #2 반복의 행을 선택합니다. 마찬가지로, 반복 번호 4에 해당하는 #3 행을 선택합니다.

테이블 보기 - 표시할 열 구성

를 선택합니다. 그런 다음 기본 설정 대화 상자의 표시할 열 선택에서 표시할 열을 선택합니다.

기본적으로 이 모드에는 이름, 유형, 상태, 리소스 및 다음 후에 시작됨 열이 표시됩니다.

테이블 보기 - 결과 필터링

상태 또는 날짜 및 시간 범위와 같은 속성을 기반으로 필터를 하나 이상 적용하면 표시되는 정보의 양을 제한합니다. 예를 들어 실행을 실패한 단계를 보려면 다음 필터를 적용합니다.

1. 속성으로 필터링 또는 키워드로 검색을 선택한 다음 속성에서 상태를 선택합니다.
2. 연산자에서 Status =를 선택합니다.
3. Status = Failed를 선택합니다.
4. (선택 사항) 필터 지우기를 선택하여 적용된 필터를 제거합니다.

이벤트 보기 - 결과 필터링

유형 또는 날짜 및 시간 범위와 같은 속성을 기반으로 필터를 하나 이상 적용하면 표시되는 정보의 양을 제한합니다. 예를 들어 실행을 실패한 Task 상태를 보려면 다음 필터를 적용합니다.

1. 속성으로 필터링 또는 키워드로 검색을 선택한 다음 속성에서 유형을 선택합니다.
2. 연산자에서 Type =를 선택합니다.
3. 유형 =을 선택합니다 TaskFailed.
4. (선택 사항) 필터 지우기를 선택하여 적용된 필터를 제거합니다.

이벤트 보기 - TaskFailed이벤트 세부 정보 검사

TaskFailed이벤트 ID

▶

옆에 있는 항목을 선택하여 드롭다운 상자에 나타나는 입력, 출력 및 리소스 호출을 비롯한 세부 정보를 검사할 수 있습니다.

실행 Redriving

redrive를 사용하여 지난 14일 동안 성공적으로 완료되지 않은 표준 워크플로 실행을 다시 시작할 수 있습니다. 여기에는 실패, 중단 또는 시간 초과된 실행이 포함됩니다.

redrive를 실행하면 실패한 단계부터 실패한 실행을 계속하고 동일한 입력을 사용합니다. Step Functions는 성공한 단계의 결과와 실행 내역을 보존하며 redrive가 실행되면 이러한 단계가 다시 실행되지 않습니다. 예를 들어 워크플로에 Pass 상태와 태스크 상태 상태 등 2가지 상태가 있다고 가정해보겠습니다. Task 상태에서 워크플로 실행이 실패하고 실행을 redrive하면 실행 일정이 조정된 후 Task 상태가 다시 실행됩니다.

Redriven 실행에서는 원래 실행 시도에 사용된 것과 동일한 상태 시스템 정의와 실행 ARN을 사용합니다. 원래 실행 시도가 버전, 별칭 또는 둘 다에 연결된 경우 redriven 실행은 같은 버전, 별칭 또는 둘 다와 연결됩니다. 다른 버전을 가리키도록 별칭을 업데이트하더라도 redriven 실행에서는 원래 실행 시도와 연결된 버전을 계속 사용합니다. redriven 실행은 같은 상태 시스템 정의를 사용하므로 상태 시스템 정의를 업데이트한 경우에는 새 실행을 시작해야 합니다.

실행을 redrive하면 상태 시스템 수준 제한 시간(정의된 경우)이 0으로 재설정됩니다. 상태 시스템 수준 제한 시간에 대한 자세한 내용은 `TimeoutSeconds`를 참조하세요.

redrives 실행은 상태 전환으로 간주됩니다. 상태 전환이 결제에 미치는 영향은 Step Functions 요금을 참조하세요.

주제

실패한 실행에 대한 Redrive 적격성

원래 실행 시도가 다음 조건을 충족하는 경우에 실행을 redrive할 수 있습니다.

- 2023년 11월 15일 이후에 실행을 시작했습니다. 이 날짜 이전에 시작한 실행은 redrive에 적합하지 않습니다.
- 실행 상태는 `SUCCEEDED`가 아닙니다.
- 워크플로 실행이 redrivable할 수 있는 14일을 초과하지 않았습니다. Redrivable 기간은 특정 실행을 redrive할 수 있는 기간을 의미합니다. 이 기간은 상태 시스템에서 실행을 완료한 날부터 시작합니다.

- 워크플로 실행이 최대 오픈 시간인 1년을 초과하지 않았습니다. 상태 시스템 실행 할당량은 상태 시스템 실행과 관련된 할당량을 참조하세요.
- 실행 이벤트 내역 수는 24,999개 미만입니다. Redriven 실행은 해당 이벤트 내역을 기존 이벤트 내역에 추가합니다. 워크플로 실행에 이벤트가 24,999개 미만 있어 `ExecutionRedriven` 내역 이벤트와 다른 내역 이벤트를 최소 하나 이상 수용할 수 있는지 확인합니다.

개별 상태의 Redrive 동작

워크플로에서 실패한 상태에 따라 모든 실패 상태의 redrive 동작이 달라집니다. 다음 표에서는 모든 상태에 대한 redrive 동작을 설명합니다.

상태 이름	Redrive 실행 동작
Pass	이전 단계가 실패하거나 상태 시스템 시간이 초과되면 Pass 상태가 종료되고 redrive에서 실행되지 않습니다.
태스크 상태	Task 상태를 다시 예약하고 시작합니다. Task 상태를 재실행하는 실행을 redrive하면 상태에 대한 `TimeoutSeconds` (정의된 경우)는 0으로 재설정됩니다. 제한 시간에 대한 자세한 내용은 Task 상태를 참조하세요.
Choice	Choice 상태 규칙을 다시 평가합니다.
Wait	상태에서 과거의 타임스탬프를 참조하는 `Timestamp` 또는 `TimestampPath` 를 지정하면 redrive로 인해 Wait 상태가 종료되고 `Next` 필드에 지정된 상태로 전환됩니다.
Succeed	Succeed 상태로 전환되는 상태 시스템 실행을 redrive하지 마세요.
Fail	Fail 상태로 다시 전환되었다가 다시 실패합니다.
Parallel	실패하거나 중단된 해당 브랜치만 다시 예약하고 redrives합니다.

상태 이름	Redrive 실행 동작
	`States.DataLimitExceeded` 오류로 인해 상태가 실패한 경우 원래 실행 시도에서 성공한 브랜치를 포함하여 Parallel 상태가 다시 실행됩니다.
Inline Map 상태	실패하거나 중단된 해당 반복만 다시 예약하고 redrives합니다. `States.DataLimitExceeded` 오류로 인해 상태가 실패한 경우 원래 실행 시도에서 성공한 반복을 포함하여 Inline Map 상태가 다시 실행됩니다.
Distributed Map 상태	맵 실행에서 실패한 하위 워크플로 실행을 redrives합니다. 자세한 설명은 맵 실행 Redriving 섹션을 참조하세요. `States.DataLimitExceeded` 오류로 인해 상태가 실패한 경우 Distributed Map 상태가 다시 실행됩니다. 여기에는 원래 실행 시도에서 성공한 하위 워크플로가 포함됩니다.

실행을 redrive할 수 있는 IAM 권한

Step Functions에는 실행을 redrive할 수 있는 적절한 권한이 필요합니다. 다음 IAM 정책 예제에서는 실행을 redriving하는 데 필요한 최소 권한을 상태 시스템에 부여합니다. #### 텍스트를 리소스별 정보로 바꿔야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:RedriveExecution"
            ],
```

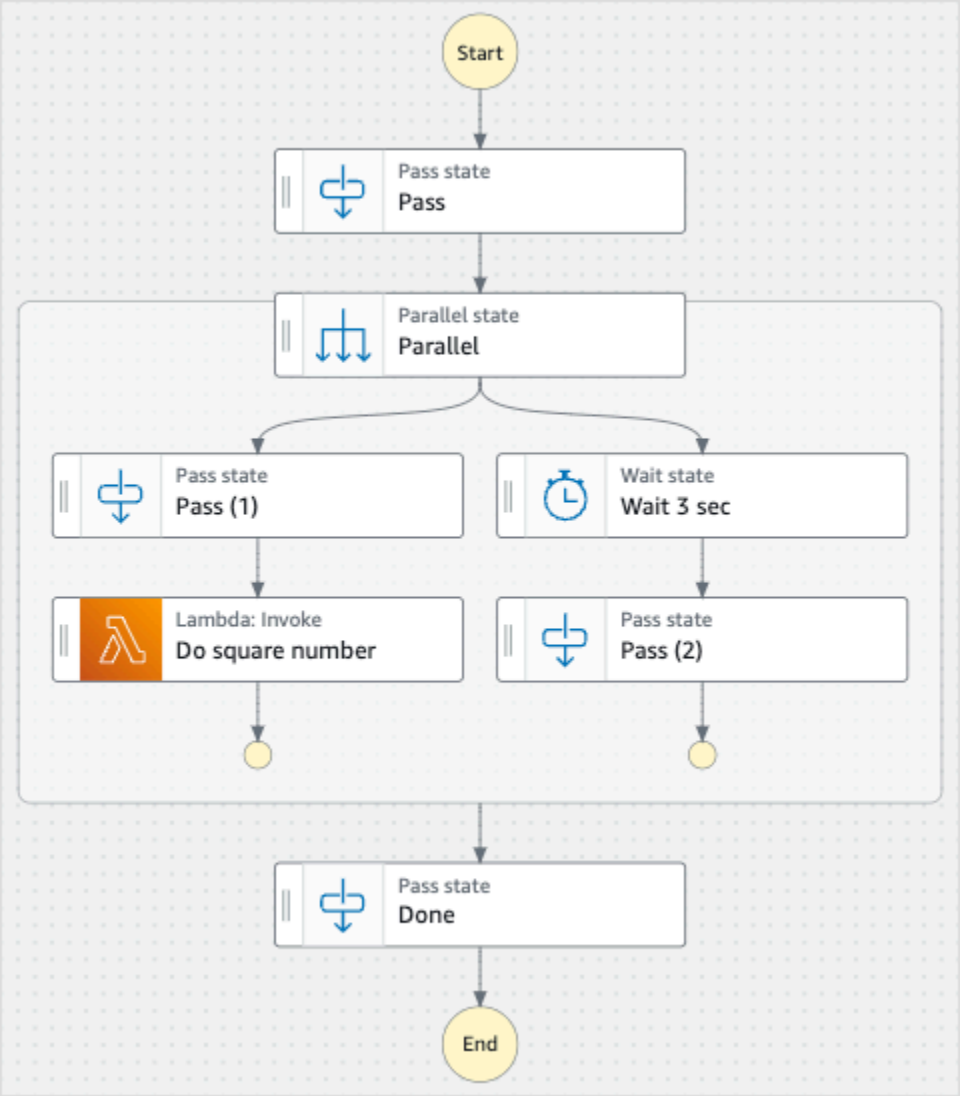
```
            "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:*"
        }
    ]
}
```

맵 실행을 redrive하는 데 필요한 권한의 예제는 Distributed Map을 redriving하기 위한 IAM 정책 예제를 참조하세요.

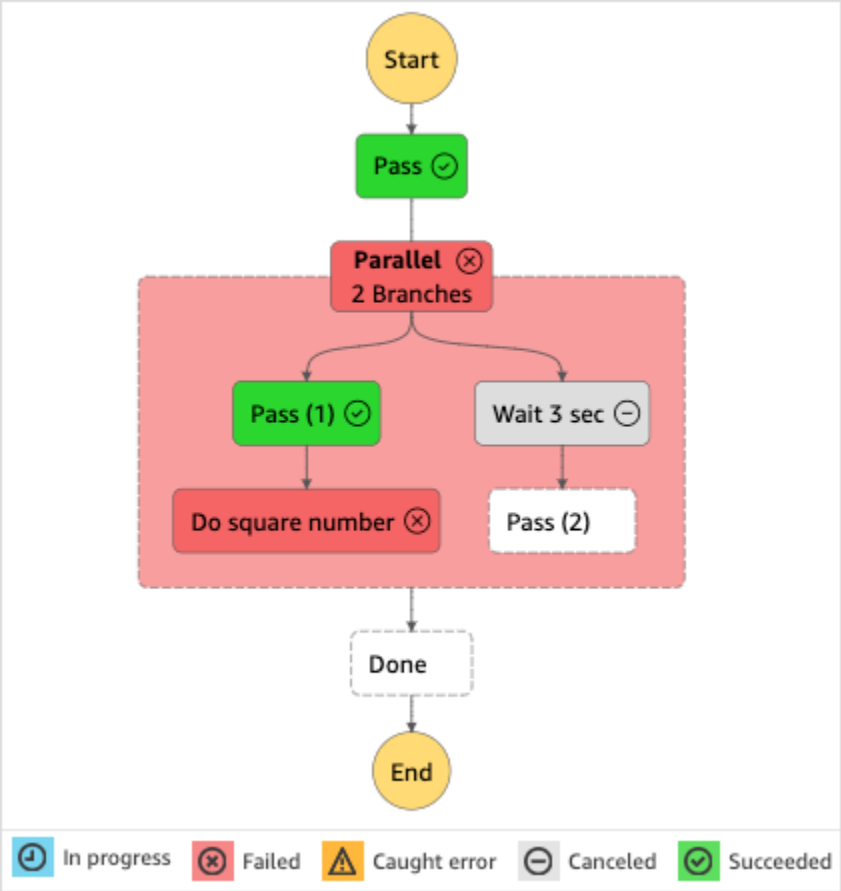
콘솔에서 실행 Redriving

Step Functions 콘솔에서 적격한 실행을 redrive할 수 있습니다.

예를 들어 다음 이미지가 상태 시스템의 워크플로 그래프를 나타낸다고 가정해보겠습니다.



이 상태 시스템을 실행한다고 가정해보겠습니다. 다음 이미지에서는 상태 시스템 실행에 대한 그래프를 보여줍니다.



이 이미지에 표시된 것처럼 Parallel 상태 내의 Do square number라고 하는 Lambda 간접 호출 단계에서 오류를 반환했습니다. 이로 인해 Parallel 상태가 실패했습니다. 실행이 진행 중이거나 시작되지 않은 브랜치가 중지되고 상태 시스템 실행이 실패합니다.

콘솔에서 실행을 redrive하기

1. Step Functions 콘솔을 열고 실행을 실패한 기존 상태 시스템을 선택합니다.
2. 상태 시스템 세부 정보 페이지의 실행에서 실패한 실행 인스턴스를 선택합니다.
3. Redrive를 선택합니다.
4. Redrive 대화 상자에서 Redrive 실행을 선택합니다.

> **Tip**
>
> 실패한 실행의 실행 세부 정보 페이지에서 다음 중 하나를 수행하여 실행을 redrive하세요.
>
> - 복구를 선택한 다음 장애로부터 Redrive을 선택합니다.

- 작업을 선택한 다음 Redrive를 선택합니다.

redrive에서 같은 상태 시스템 정의와 ARN을 사용합니다. 원래 실행 시도에서 실패한 단계부터 실행을 계속 실행합니다. 이 예제에서는 Parallel 상태 내의 Do square number 단계와 Wait 3 sec 브랜치입니다. Parallel 상태에서 이러한 실패한 단계 실행을 다시 시작한 후에는 redrive에서 Done 단계의 실행을 계속합니다.

5. 실행을 선택하여 실행 세부 정보 페이지를 엽니다.

   이 페이지에서 redriven 실행 결과를 볼 수 있습니다. 예를 들어 실행 요약 섹션에서 실행이 redriven된 횟수를 나타내는 Redrive 횟수를 확인할 수 있습니다. 이벤트 섹션에서 원래 실행 시도 이벤트에 추가된 redrive 관련 실행 이벤트를 확인할 수 있습니다. 예제는 `ExecutionRedriven` 이벤트를 참조하세요.

API를 사용하여 실행 Redriving

RedriveExecutionAPI를 사용하여 redrive 적격한 실행을 수행할 수 있습니다. 이 API는 실패, 중단 또는 시간 초과된 단계부터 실패한 표준 워크플로 실행을 다시 시작합니다.

AWS Command Line Interface (AWS CLI) 에서 다음 명령을 실행하여 redrive 스테이트 머신 실행에 실패했습니다. `####` 텍스트를 리소스별 정보로 바꿔야 합니다.

```
aws stepfunctions redrive-execution --execution-arn arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:foo
```

redriven 실행 검사

콘솔에서 또는 API를 사용하여 redriven 실행을 검사할 수 있습니다.
GetExecutionHistoryDescribeExecution

콘솔에서 redriven 실행 검사

1. Step Functions 콘솔을 열고 실행을 redriven한 기존 상태 시스템을 선택합니다.
2. 실행 세부 정보 페이지를 엽니다.

   이 페이지에서 redriven 실행 결과를 볼 수 있습니다. 예를 들어 실행 요약 섹션에서 실행이 redriven된 횟수를 나타내는 Redrive 횟수를 확인할 수 있습니다. 이벤트 섹션에서 원래 실행 시도

이벤트에 추가된 redrive 관련 실행 이벤트를 확인할 수 있습니다. 예제는 `ExecutionRedriven` 이벤트를 참조하세요.

API를 사용하여 redriven 실행 검사

상태 시스템 실행을 redriven한 경우 다음 API 중 하나를 사용하여 redriven 실행에 대한 세부 정보를 볼 수 있습니다. *####* 텍스트를 리소스별 정보로 바꿔야 합니다.

- GetExecutionHistory — 지정된 실행 기록을 이벤트 목록으로 반환합니다. 이 API는 redrive 실행 시도에 대한 세부 정보도 반환합니다(가능한 경우).

  에서 AWS CLI 다음 명령을 실행합니다.

  ```
  aws stepfunctions get-execution-history --execution-arn arn:aws:states:us-east-2:123456789012:execution:myStateMachine:foo
  ```

- DescribeExecution — 상태 머신 실행에 대한 정보를 제공합니다. 이는 실행과 관련된 상태 시스템, 실행 입력 및 출력, 실행 redrive 세부 정보(있는 경우) 및 관련 실행 메타데이터일 수 있습니다.

  에서 AWS CLI 다음 명령을 실행합니다.

  ```
  aws stepfunctions describe-execution --execution-arn arn:aws:states:us-east-2:123456789012:execution:myStateMachine:foo
  ```

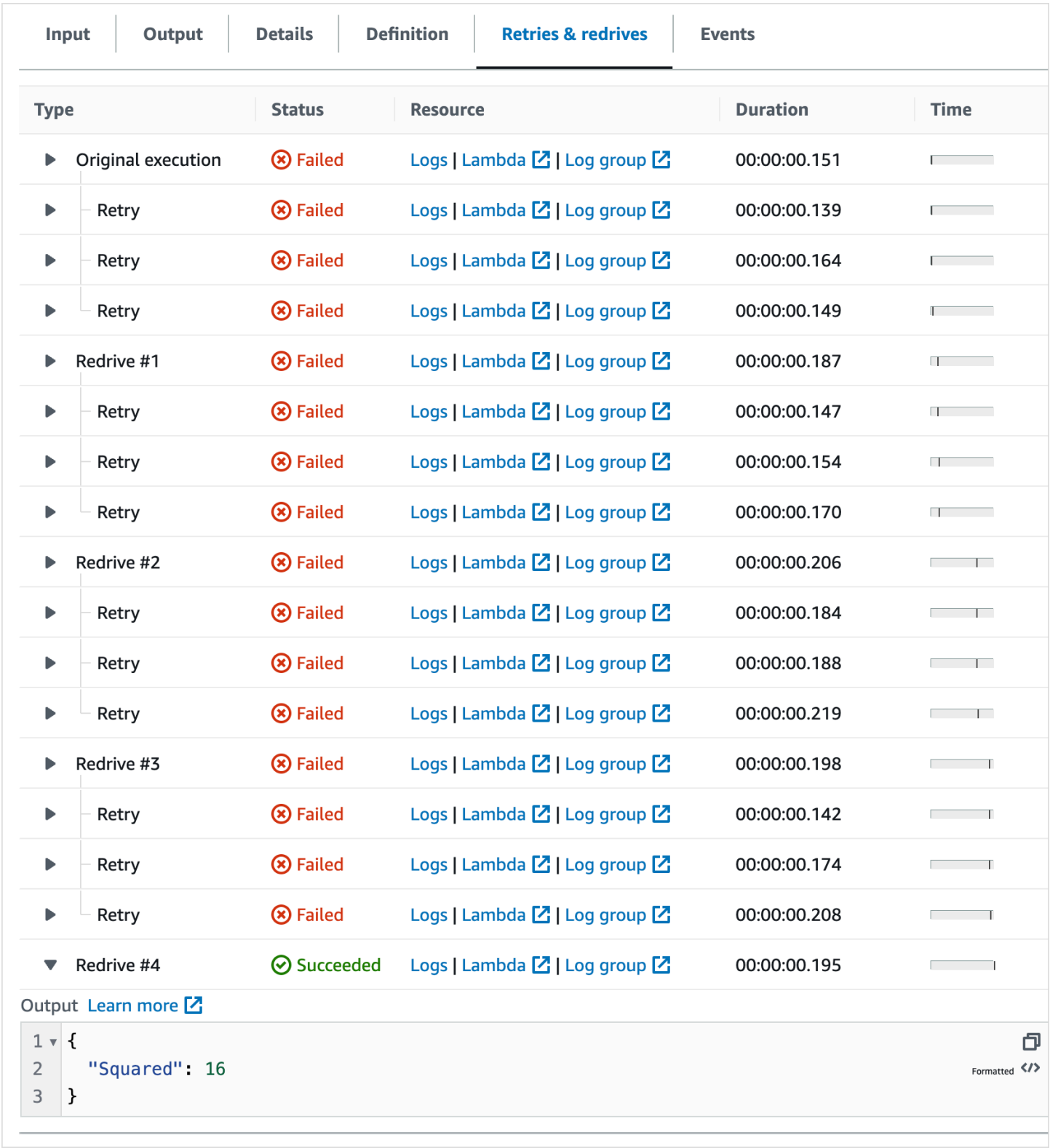
redriven 실행 재시도 동작

redriven 실행에서 재시도를 정의한 태스크 상태, Parallel 또는 Inline Map 상태를 재시도하는 경우 이러한 상태에 대한 재시도 횟수는 0으로 재설정됩니다. 이렇게 하면 redrive에서의 최대 시도 횟수가 허용됩니다. redriven 실행의 경우 콘솔을 사용하여 이러한 상태의 개별 재시도를 추적할 수 있습니다.

콘솔에서 개별 재시도 검사하기

1. Step Functions 콘솔의 실행 세부 정보 페이지에서 redrive에서 재시도된 상태를 선택합니다.
2. 재시도 및 redrives 탭을 선택합니다.
3. 각 재시도 옆에 있는 ▶ 를 선택하여 세부 정보를 봅니다. 재시도가 성공하면 드롭다운 상자에 나타나는 출력에서 결과를 볼 수 있습니다.

다음 이미지에서는 원래 실행 시도 상태와 해당 실행 redrives에 대해 수행된 재시도의 예제를 보여줍니다. 이 이미지에서는 원래 시도와 redrive 실행 시도에서 재시도가 3회 수행됩니다. 네 번째 redrive 시도에서 실행이 성공하고 출력 16이 반환됩니다.

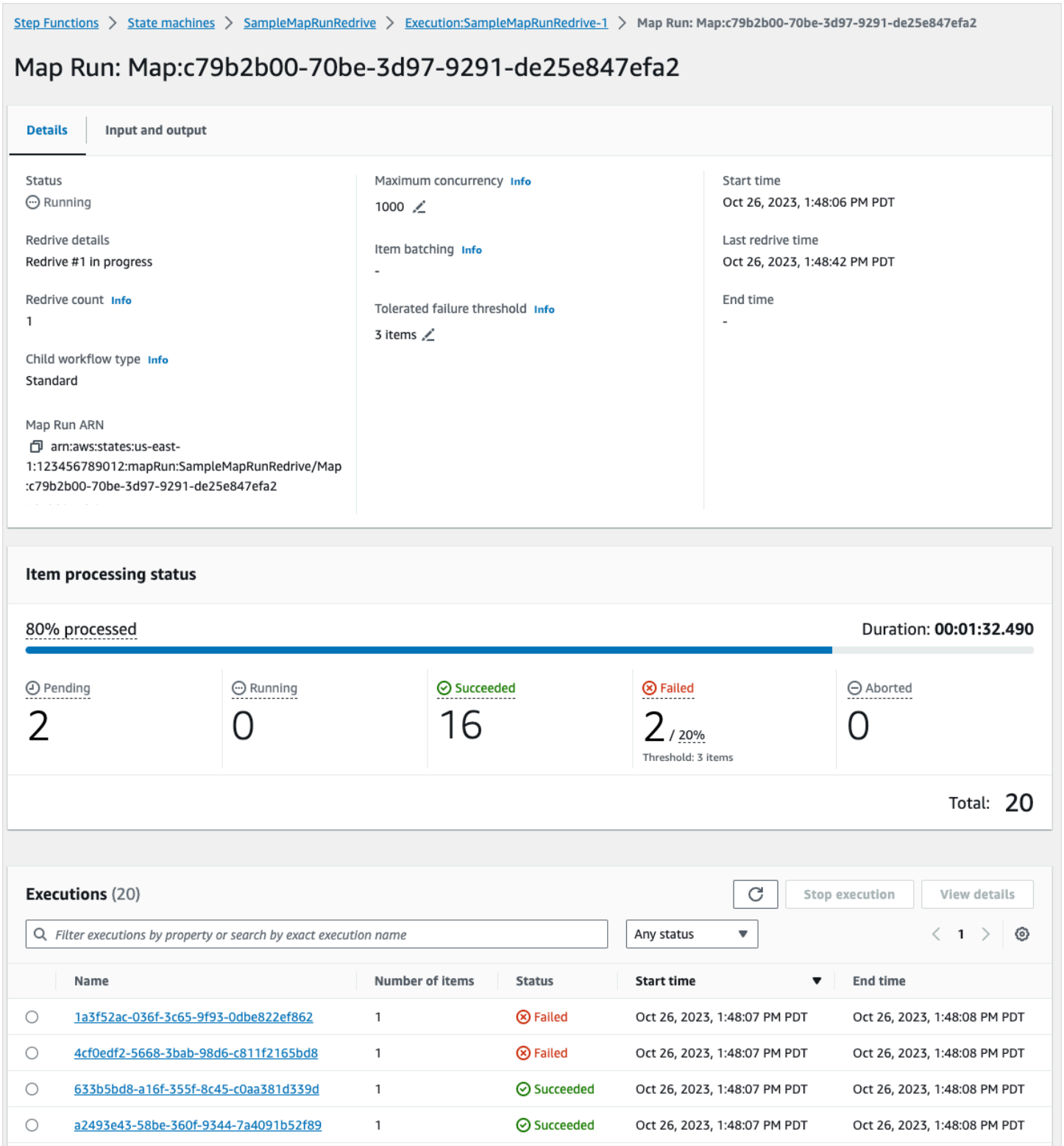


Distributed Map 상태 실행의 맵 실행 검사

분산 모드에서 `Map` 상태를 실행하면 Step Functions에서 맵 실행 리소스를 만듭니다. 맵 실행은 Distributed Map 상태가 시작하는 일련의 하위 워크플로 실행과 이러한 실행을 제어하는 런타임 설정을 의미합니다. Step Functions에서 Amazon 리소스 이름(ARN)을 맵 실행에 할당합니다. Step Functions 콘솔에서 맵 실행을 검사할 수 있습니다. `DescribeMapRun` API 작업을 간접적으로 호출할 수도 있습니다. Map Run은 메트릭도 CloudWatch 내보냅니다.

Step Functions 콘솔은 Distributed Map 상태 실행과 관련된 모든 정보가 표시되는 맵 실행 세부 정보 페이지를 제공합니다. 예를 들어 Distributed Map 상태 실행의 상태, 맵 실행 ARN 및 Distributed Map 상태에서 시작된 하위 워크플로 실행에서 처리된 항목의 상태를 볼 수 있습니다. 또한 모든 하위 워크플로 실행 목록을 보고 세부 정보에 액세스할 수 있습니다. 또한 맵 실행이 redriven됐으면 맵 실행 요약 섹션에서 맵 실행의 redrive 세부 정보를 볼 수 있습니다. 예를 들면 지난 redrive 시간입니다. 이 정보는 콘솔에 대시보드 형식으로 표시됩니다.

맵 실행 세부 정보 페이지에는 다음 섹션이 포함됩니다.



내용

- 항목 처리 상태
- 실행 목록
- 맵 실행 Redriving
  - 맵 실행에서 하위 워크플로에 대한 Redrive 적격성
  - 하위 워크플로 실행 redrive 동작
  - 맵 실행 redrive에서 사용된 입력 시나리오
  - 맵 실행을 redrive할 수 있는 IAM 권한
  - 콘솔에서 맵 실행 Redriving
  - API를 사용하여 맵 실행 Redriving

맵 실행 요약

맵 실행 요약 섹션은 맵 실행 세부정보 페이지 상단에 표시됩니다. 이 섹션에는 Distributed Map 상태의 실행 세부 정보가 간략하게 표시됩니다. 이 정보는 다음 탭으로 구분됩니다.

세부 정보

Distributed Map 상태의 실행 상태, 맵 실행 ARN 및 Distributed Map 상태에서 시작된 하위 워크플로 실행 유형과 같은 정보를 표시합니다. 맵 실행의 허용 실패 임계값, 하위 워크플로 실행에 지정된 최대 동시성과 같은 추가 구성을 볼 수 있습니다. 이러한 구성을 편집할 수도 있습니다.

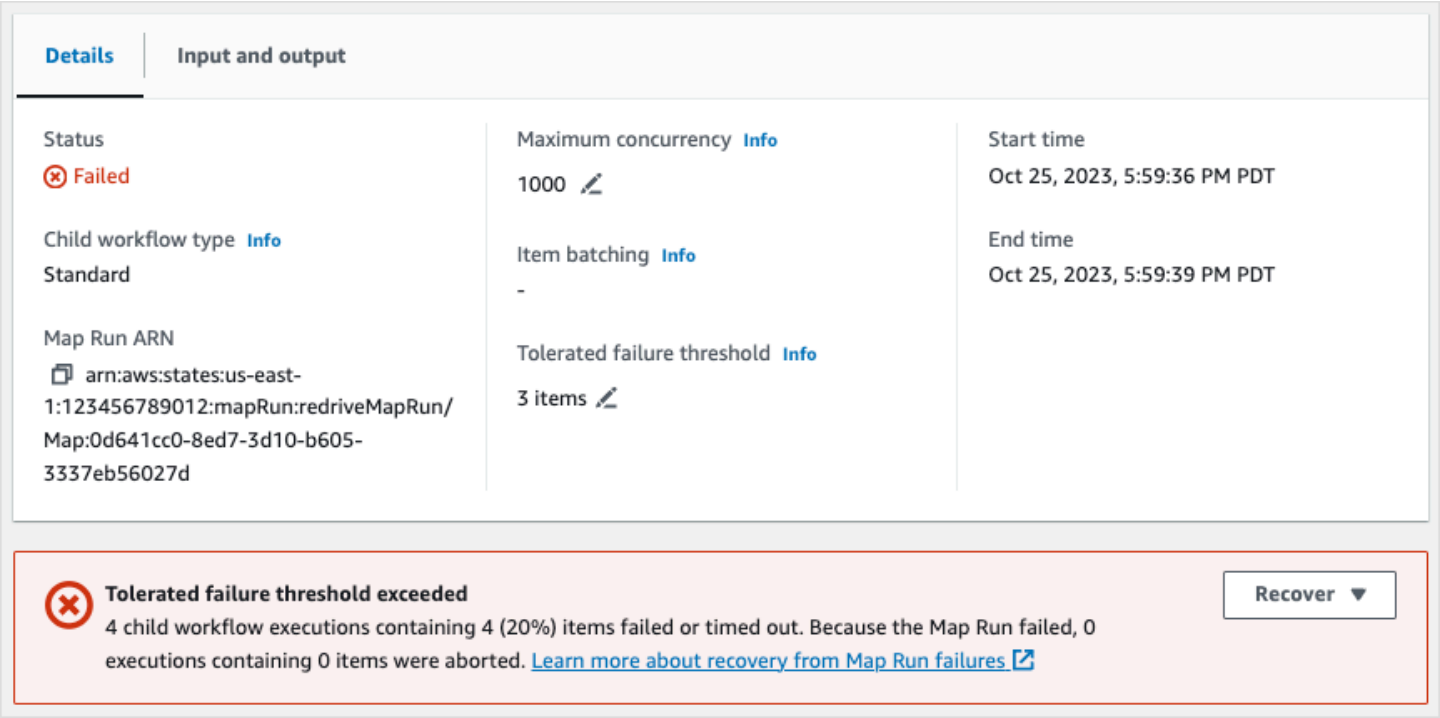
입력 및 출력

Distributed Map 상태에서 수신한 입력과 입력에서 생성하는 해당 출력을 표시합니다. 예를 들어 입력 데이터 세트와 해당 위치, 해당 데이터 세트의 개별 데이터 항목에 적용된 입력 필터를 볼 수 있습니다. Distributed Map 상태 실행의 출력을 내보내면 이 탭에는 실행 결과가 포함된 Amazon S3 버킷의 경로가 표시됩니다. 그렇지 않으면 상위 워크플로의 실행 세부 정보 페이지로 이동하여 실행 결과를 봅니다.

오류 메시지

맵 실행이 실패하면 맵 실행 세부 정보 페이지에 실패 이유와 함께 오류 메시지가 표시됩니다.

이 오류 메시지의 복구 드롭다운 버튼에서 이 맵 실행에서 시작된 실패한 하위 워크플로 실행을 redrive하거나 상위 워크플로의 새 실행을 시작할 수 있습니다. 자세한 설명은 맵 실행 Redriving 섹션을 참조하세요.



항목 처리 상태

항목 처리 상태 섹션에는 맵 실행에서 처리된 항목의 상태가 표시됩니다. 예를 들어 보류 중은 하위 워크플로 실행에서 아직 항목 처리를 시작하지 않았음을 나타냅니다.

항목 상태는 항목을 처리하는 하위 워크플로 실행 상태에 따라 달라집니다. 하위 워크플로 실행이 실패하거나 시간 초과되거나 사용자가 실행을 취소하면 Step Functions는 해당 하위 워크플로 실행 내의 항목 처리 결과에 대한 정보를 수신하지 않습니다. 해당 실행에서 처리한 모든 항목은 하위 워크플로 실행 상태를 공유합니다.

예를 들어 각 실행에서 항목 50개를 일괄 처리하는 하위 워크플로 실행 2개에서 항목 100개를 처리하려고 한다고 가정해보겠습니다. 실행 중 하나가 실패하고 다른 하나는 성공하면 성공한 항목 50개와 실패한 항목 50개가 있게 됩니다.

다음 표에는 모든 항목에 사용할 수 있는 처리 상태 유형이 설명되어 있습니다.

상태 표시기	설명
보류중	하위 워크플로 실행에서 처리를 시작하지 않은 항목을 나타냅니다. 항목 처리가 시작되기 전에 맵 실행이 중지 또는 실패하거나 사용자가 실행

상태 표시기	설명
	을 취소한 경우 항목은 보류 중 상태로 유지됩니다. 예를 들어 처리 보류 중인 항목 10개가 있는 맵 실행이 실패하면 이러한 항목 10개는 보류 중 상태로 유지됩니다.
[실행 중]	하위 워크플로 실행에서 현재 처리 중인 항목을 나타냅니다.
성공	하위 워크플로 실행에서 항목을 성공적으로 처리했음을 나타냅니다. 성공한 하위 워크플로 실행에는 실패한 항목이 있을 수 없습니다. 실행 중에 데이터 세트의 항목 하나가 실패하면 전체 하위 워크플로 실행이 실패합니다.
실패	하위 워크플로 실행이 항목을 처리하지 못했거나 실행 시간이 초과되었음을 나타냅니다. 하위 워크플로 실행에서 처리한 항목 중 하나라도 실패하면 전체 하위 워크플로 실행이 실패합니다. 항목 1,000개를 처리한 하위 워크플로 실행을 예로 들어보겠습니다. 실행 중에 해당 데이터 세트의 항목 중 하나가 실패하면 Step Functions는 전체 하위 워크플로 실행을 실패로 간주합니다. 맵 실행을 redrive하면 이 상태의 항목 수가 0으로 재설정됩니다.

상태 표시기	설명
중단됨	하위 워크플로 실행에서 항목 처리를 시작했지만 사용자가 실행을 취소했거나 맵 실행이 실패하여 Step Functions에서 실행을 중지했음을 나타냅니다. 항목 50개를 처리하는 실행 중인 하위 워크플로 실행을 예로 들어보겠습니다. 실패 또는 사용자의 실행 취소로 인해 맵 실행이 중지되면 하위 워크플로 실행과 모든 항목 50개의 상태가 중단됨으로 변경됩니다. Express 유형의 하위 워크플로 실행을 사용하는 경우 실행을 중지할 수 없습니다. Express 유형의 하위 워크플로 실행을 시작하는 맵 실행을 redrive하면 이 상태의 항목 수가 0으로 재설정됩니다. Express 하위 워크플로가 다시 시작되지 않고 StartExecutionAPI 작업을 사용하여 다시 시작되기 때문입니다. redriven

실행 목록

실행 섹션에는 특정 맵 실행의 모든 하위 워크플로 실행이 나열됩니다. 정확한 실행 이름으로 검색 필드를 사용하면 특정 하위 워크플로 실행을 검색할 수 있습니다. 또한 모든 상태 드롭다운을 사용하여 상태별로 하위 워크플로 실행 내역을 필터링할 수 있습니다. 특정 실행에 대한 세부 정보를 확인하려면 목록에서 하위 워크플로 실행을 선택하고 세부 정보 보기 버튼을 선택하여 실행 세부 정보 페이지를 엽니다.

> **Important**
>
> 하위 워크플로 실행의 보존 정책은 90일입니다. 이 보존 기간보다 오래된 완료된 하위 워크플로 실행은 실행 테이블에 표시되지 않습니다. Distributed Map 상태 또는 상위 워크플로가 보존 기간보다 오래 계속 실행되는 경우에도 마찬가지입니다. `ResultWriter`를 사용하여 Distributed Map 상태 출력을 Amazon S3 버킷으로 내보내면 이러한 하위 워크플로 실행 결과를 포함하여 실행 세부 정보를 볼 수 있습니다.

> **Tip**
>
> 모든 하위 워크플로 실행의 최신 목록을 보려면 새로 고침 버튼
>
> 
>
> 을 선택하세요.

맵 실행 Redriving

상위 워크플로를 redriving하면 맵 실행에서 실패한 하위 워크플로 실행을 다시 시작할 수 있습니다. redriven 상위 워크플로에서 Distributed Map을 비롯한 실패한 모든 상태를 redrives합니다. 상위 워크플로에서 실행을 완료한 시점의 상태에 대한 `<stateType>Entered` 이벤트에 해당하는 `<stateType>Exited` 이벤트가 없으면 상위 워크플로는 실패한 상태를 리드라이브합니다. 예를 들어 이벤트 내역에 `MapStateEntered` 이벤트에 대한 `MapStateExited` 이벤트가 포함되어 있지 않으면 상위 워크플로를 redrive하여 맵 실행에서 실패한 모든 하위 워크플로 실행을 redrive할 수 있습니다.

상태 시스템에 ItemReader, ResultWriter 또는 둘 다에 액세스하는 데 필요한 권한이 없으면 맵 실행이 원래 실행 시도에서 시작되지 않거나 실패합니다. 맵 실행이 상위 워크플로의 원래 실행 시도에서 시작되지 않은 경우 상위 워크플로를 redriving하면 맵 실행이 처음으로 시작됩니다. 이 문제를 해결하려면 상태 시스템 역할에 필요한 권한을 추가한 다음 상위 워크플로를 redrive합니다. 필요한 권한을 추가하지 않고 상위 워크플로를 redrive한 경우 새 맵 실행을 시작하려고 시도하면 다시 실패합니다. 필요할 수 있는 권한은 Distributed Map 상태를 사용하기 위한 IAM 정책 섹션을 참조하세요.

주제

- 맵 실행에서 하위 워크플로에 대한 Redrive 적격성
- 하위 워크플로 실행 redrive 동작
- 맵 실행 redrive에서 사용된 입력 시나리오
- 맵 실행을 redrive할 수 있는 IAM 권한
- 콘솔에서 맵 실행 Redriving
- API를 사용하여 맵 실행 Redriving

맵 실행에서 하위 워크플로에 대한 Redrive 적격성

다음 조건이 충족되면 맵 실행에서 실패한 하위 워크플로를 redrive할 수 있습니다.

- 2023년 11월 15일 이후에 상위 워크플로 실행을 시작했습니다. 이 날짜 이전에 시작한 실행은 redrive에 적합하지 않습니다.
- 특정 맵 실행의 엄격한 한도인 redrives 1,000회를 초과하지 않았습니다. 이 한도를 초과하면 `States.Runtime` 오류가 발생합니다.
- 상위 워크플로는 redrivable입니다. 상위 워크플로가 redrivable이 아니면 맵 실행에서 하위 워크플로 실행을 redrive할 수 없습니다. 워크플로의 redrive 적격성에 대한 자세한 내용은 실패한 실행에 대한 Redrive 적격성 섹션을 참조하세요.
- 맵 실행에서 표준 유형의 하위 워크플로 실행이 실행 이벤트 내역 한도인 25,000개를 초과하지 않았습니다. 이벤트 내역 한도를 초과한 하위 워크플로 실행은 허용 실패 임계값에 포함되며 실패로 간주됩니다. 실행의 redrive 적격성에 대한 자세한 내용은 실패한 실행에 대한 Redrive 적격성 섹션을 참조하세요.

맵 실행이 원래 실행 시도에서 실패했더라도 다음과 같은 경우에는 새 맵 실행이 시작되고 기존 맵 실행은 redriven되지 않습니다.

- `States.DataLimitExceeded` 오류로 인해 맵 실행이 실패했습니다.
- JSON 데이터 보간 오류인 `States.Runtime`으로 인해 맵 실행이 실패했습니다. 예를 들어 OutputPath에 존재하지 않는 JSON 노드를 선택했습니다.

상위 워크플로가 중지되거나 시간 초과된 후에도 맵 실행은 계속 실행될 수 있습니다. 다음과 같은 시나리오에서는 redrive는 즉시 발생하지 않습니다.

- 맵 실행에서 여전히 진행 중인 표준 유형의 하위 워크플로 실행을 취소하거나 Express 유형의 하위 워크플로 실행이 완료될 때까지 기다리고 있을 수 있습니다.
- 결과를 내보내도록 맵 실행을 구성한 경우 맵 실행에서 여전히 결과를 ResultWriter에 쓰고 있을 수 있습니다.

이러한 경우에는 실행 중인 맵 실행이 redrive를 시도하기 전에 작업을 완료합니다.

하위 워크플로 실행 redrive 동작

맵 실행의 redriven 하위 워크플로 실행은 다음 표의 설명과 같은 동작을 나타냅니다.

Express 하위 워크플로	표준 하위 워크플로
원래 실행 시도에서 실패하거나 제한 시간이 초과된 모든 하위 워크플로 실행은 StartExecutionAPI 작업을 사용하여 시작됩니다. ItemProcessor의 첫 번째 상태가 먼저 실행됩니다.	원래 실행 시도에서 실패, 시간 초과 또는 취소된 모든 하위 워크플로 실행은 RedriveExecution API 작업을 통해 redriven됩니다. 이러한 하위 워크플로는 ItemProcessor 실행에 실패한 마지막 상태의 것입니다. redriven
실행 실패는 항상 redriven될 수 있습니다. Express 하위 워크플로 실행은 항상 API 작업을 사용하여 새 실행으로 시작되기 때문입니다. StartExecution	실패한 표준 하위 워크플로 실행은 항상 redriven될 수 없습니다. 실행이 redrivable되지 않으면 다시 시도되지 않습니다. 실행의 마지막 오류나 출력은 영구적입니다. 이는 실행이 기록 이벤트 25,000개를 초과하거나 redrivable 기간 14일이 만료된 경우에 가능합니다. 상위 워크플로 실행이 14일 이내에 종료되었지만 하위 워크플로 실행은 14일 이전에 종료된 경우에는 표준 하위 워크플로 실행이 redrivable되지 않을 수 있습니다.
Express 하위 워크플로 실행은 원래 실행 시도와 동일한 실행 ARN을 사용하지만 개별 redrives를 명확하게 식별할 수 없습니다.	표준 하위 워크플로 실행은 원래 실행 시도와 동일한 실행 ARN을 사용합니다. redrives콘솔에서 및 같은 API를 사용하여 개인을 명확하게 식별할 수 있습니다. GetExecutionHistoryDescribeExecution 자세한 정보는 redriven 실행 검사을 참조하세요.

맵 실행을 redriven하고 동시 한도에 도달하면 해당 맵 실행의 하위 워크플로 실행이 보류 중 상태로 전환됩니다. 맵 실행의 실행 상태도 redrive 보류 중 상태로 전환됩니다. 지정된 동시성 한도에서 더 많은 하위 워크플로 실행을 허용할 수 있을 때까지 실행은 redrive 보류 중 상태로 유지됩니다.

예를 들어 워크플로에 있는 분산 맵의 동시성 한도가 3000이고 재실행할 하위 워크플로 수가 6000개라고 가정해보겠습니다. 이로 인해 하위 워크플로가 3000개가 동시에 실행되고 나머지 워크플로 3000개는 리드라이브 보류 중 상태로 유지됩니다. 하위 워크플로 3000개의 첫 번째 배치가 실행을 완료하면 나머지 하위 워크플로 3000개가 실행됩니다.

맵 실행이 실행을 완료하거나 중단되면 redrive 보류 중 상태의 하위 워크플로 실행 수가 0으로 재설정됩니다.

맵 실행 redrive에서 사용된 입력 시나리오

원래 실행 시도에서 분산 맵에 입력을 제공한 방식에 따라 redriven 맵 실행은 다음 표의 설명대로 입력을 사용합니다.

원래 실행 시도의 입력	맵 실행 redrive에 사용된 입력
이전 상태에서 전달된 입력 또는 실행 입력	redriven 맵 실행에서 같은 입력을 사용합니다.
다음 중 조건 중 하나에 해당되므로 ItemReader를 통해 전달된 입력과 맵 실행에서 하위 워크플로 실행을 시작하지 않았습니다. • States.ItemReaderFailed 오류가 발생하여 맵 실행이 실패했습니다. • States.ResultWriterFailed 오류가 발생하여 맵 실행이 실패했습니다. • 맵 실행이 시작되기 전에 상위 워크플로 실행이 시간 초과되었거나 취소되었습니다.	redriven 맵 실행에서 Amazon S3 버킷의 입력을 사용합니다.
를 사용하여 입력이 전달되었습니다. ItemReader 하위 워크플로 실행을 시작하거나 시작하려고 시도한 후에 맵 실행이 실패했습니다.	redriven 맵 실행은 원래 실행 시도에서 제공된 입력과 동일한 입력을 사용합니다.

맵 실행을 redrive할 수 있는 IAM 권한

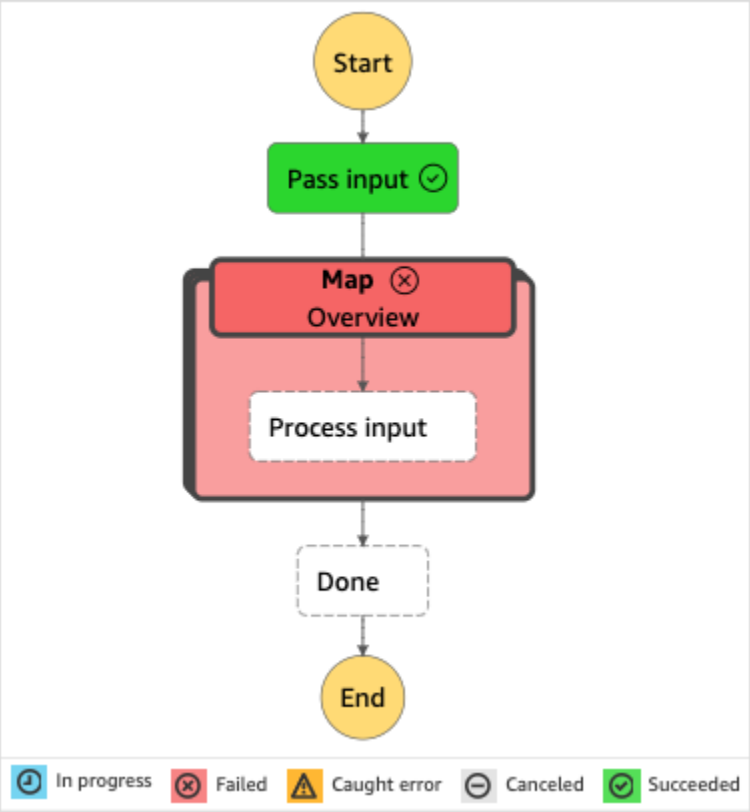
Step Functions에는 맵 실행을 redrive할 수 있는 적절한 권한이 필요합니다. 다음 IAM 정책 예제에서는 맵 실행을 redriving하는 데 필요한 최소 권한을 상태 시스템에 부여합니다. #### 텍스트를 리소스별 정보로 바꿔야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "states:RedriveExecution"
      ],
      "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine/myMapRunLabel:*"
    }
  ]
}
```

콘솔에서 맵 실행 Redriving

다음 이미지에서는 분산 맵이 포함된 상태 시스템의 실행 그래프를 보여줍니다. 맵 실행이 실패하여 이 실행이 실패했습니다. 맵 실행을 redrive하려면 상위 워크플로를 redrive해야 합니다.



콘솔에서 맵 실행 redrive하기

1. Step Functions 콘솔을 열고 실행이 실패한 분산 맵이 포함된 기존 상태 시스템을 선택합니다.
2. 상태 시스템 세부 정보 페이지의 실행에서 이 상태 시스템의 실패한 실행 인스턴스를 선택합니다.
3. Redrive를 선택합니다.
4. Redrive 대화 상자에서 Redrive 실행을 선택합니다.

> **Tip**
>
> 실행 세부 정보 또는 맵 실행 세부 정보 페이지에서 맵 실행을 redrive할 수도 있습니다. 실행 세부 정보 페이지에서 다음 중 하나를 수행하여 실행을 redrive합니다.
>
> - 복구를 선택한 다음 장애로부터 Redrive을 선택합니다.
> - 작업을 선택한 다음 Redrive를 선택합니다.
>
> 맵 실행 세부 정보 페이지에서 복구를 선택한 다음 장애로부터 Redrive을 선택합니다.

redrive에서 같은 상태 시스템 정의와 ARN을 사용합니다. 원래 실행 시도에서 실패한 단계부터 실행을 계속 실행합니다. 이 예제에서는 Map이라는 분산 맵 단계와 내부의 프로세스 입력 단계입니다. 실패한 맵 실행 하위 워크플로 실행을 다시 시작한 후에도 redrive는 완료 단계 실행을 계속합니다.

5. 실행 세부 정보 페이지에서 맵 실행을 선택하여 redriven 맵 실행 세부 정보를 확인합니다.

   이 페이지에서 redriven 실행 결과를 볼 수 있습니다. 예를 들어 맵 실행 요약 섹션에서 맵 실행이 redriven된 횟수를 나타내는 Redrive 횟수를 확인할 수 있습니다. 이벤트 섹션에서 원래 실행 시도 이벤트에 추가된 redrive 관련 실행 이벤트를 확인할 수 있습니다. 예제는 `MapRunRedriven` 이벤트를 참조하세요.

맵 실행을 redriven 수행한 후에는 콘솔에서 또는 GetExecutionHistory및 DescribeExecutionAPI 작업을 사용하여 redrive 세부 정보를 검토할 수 있습니다. redriven 실행 검사 방법에 대한 자세한 내용은 redriven 실행 검사 섹션을 참조하세요.

API를 사용하여 맵 실행 Redriving

상위 워크플로에서 RedriveExecution API를 사용하여 적합한 맵 실행을 redrive할 수 있습니다. 이 API는 맵 실행에서 실패한 하위 워크플로 실행을 다시 시작합니다.

AWS Command Line Interface (AWS CLI)에서 다음 명령을 실행하여 실패한 상태 시스템 실행을 redrive합니다. `####` 텍스트를 리소스별 정보로 바꿔야 합니다.

```
aws stepfunctions redrive-execution --execution-arn arn:aws:states:us-
east-2:123456789012:execution:myStateMachine:foo
```

Map Run을 redriven 실행한 후에는 콘솔에서 또는 DescribeMapRunAPI 작업을 사용하여 redrive 세부 정보를 검토할 수 있습니다. Map Run에서 표준 워크플로 실행의 redrive 세부 정보를 검토하려면 GetExecutionHistory또는 DescribeExecutionAPI 작업을 사용할 수 있습니다. redriven 실행 검사 방법에 대한 자세한 내용은 the section called “redriven 실행 검사” 섹션을 참조하세요.

상위 워크플로에서 로깅을 활성화한 경우 Step Functions 콘솔의 맵 실행에서 Express 워크플로 실행의 redrive 세부 정보를 검사할 수 있습니다. 자세한 내용은 CloudWatch Logs를 사용하여 로깅을(를) 참조하세요.

Step Functions에서 오류 처리

`Pass` 및 `Wait` 상태를 제외한 모든 상태에서 런타임 오류가 발생할 수 있습니다. 다음 예제와 같은 다양한 이유로 오류가 발생할 수 있습니다.

- 상태 머신 정의 문제(예: `Choice` 상태에 일치하는 규칙 없음)
- 작업 실패(예: AWS Lambda 함수에서 예외 발생)
- 일시적인 문제(예: 네트워크 파티션 이벤트)

기본적으로, 상태가 오류를 보고하는 경우 AWS Step Functions에서 실행에 전체적으로 오류를 발생시킵니다.

> **Tip**
>
> 오류 처리가 포함된 워크플로 예제를 AWS 계정에 배포하려면 AWS Step Functions 워크숍의 모듈 8 - 오류 처리를 참조하세요.

주제

- 오류 이름
- 오류 후 재시도
- 폴백 상태
- Retry 및 Catch를 사용하는 상태 시스템 예제

오류 이름

Step Functions는 오류 이름이라고 하는 대소문자를 구분하는 문자열을 사용하여 Amazon States Language의 오류를 식별합니다. Amazon States Language는 잘 알려진 오류의 이름을 지정하는 기본 제공 문자열 집합을 정의하며,모든 문자열은 `States.` 접두사로 시작합니다.

`States.ALL`

확인된 오류 이름과 일치하는 와일드카드.

> **Note**
>
> 이 오류 유형은 `States.DataLimitExceeded` 터미널 오류 유형과 런타임 오류 유형을 포착할 수 없습니다. 이러한 오류 유형에 대한 자세한 내용은 `States.DataLimitExceeded` 및 `States.Runtime`을 참조하세요.

`States.DataLimitExceeded`

Step Functions는 다음 조건에서 `States.DataLimitExceeded` 예외를 보고합니다.

- 커넥터 출력이 페이로드 크기 할당량보다 큰 경우
- 상태 출력이 페이로드 크기 할당량보다 큰 경우
- `Parameters` 처리 후 상태 입력이 페이로드 크기 할당량보다 큰 경우

할당량에 대한 자세한 내용은 할당량을 참조하세요.

> **Note**
>
> 이 오류는 `States.ALL` 오류 유형에서 포착할 수 없는 터미널 오류입니다.

`States.ExceedToleratedFailureThreshold`

실패한 항목 수가 상태 시스템 정의에 지정된 임계값을 초과하므로 `Map` 상태가 실패했습니다. 자세한 내용은 Distributed Map 상태의 허용 실패 임계값 섹션을 참조하세요.

`States.HeartbeatTimeout`

`Task` 상태에서 `HeartbeatSeconds` 값보다 오랜 시간 동안 하트비트를 보내지 못했습니다.

> **Note**
>
> 이 오류는 Catch 및 Retry 필드 내에서만 나타납니다.

States.IntrinsicFailure

페이로드 템플릿 내에서 내장 함수를 간접적으로 호출하려는 시도가 실패했습니다.

States.ItemReaderFailed

ItemReader 필드에 지정된 항목 소스에서 Map 상태를 읽을 수 없으므로 이 상태가 실패했습니다. 자세한 내용은 ItemReader 섹션을 참조하세요.

States.NoChoiceMatched

Choice 상태에서 선택 규칙에 정의된 조건과 입력을 일치시키지 못했고 기본 전환이 지정되지 않았습니다.

States.ParameterPathFailure

경로를 사용하여 상태 Parameters 필드 내에서 이름이 .$로 끝나는 필드를 교체하려는 시도가 실패했습니다.

States.Permissions

지정된 코드를 실행할 수 있는 권한이 부족하여 Task 상태가 실패했습니다.

States.ResultPathMatchFailure

Step Functions가 상태 ResultPath 필드를 상태에서 수신한 입력에 적용하지 못했습니다.

States.ResultWriterFailed

ResultWriter 필드에 지정된 대상에 결과를 쓸 수 없으므로 Map 상태가 실패했습니다. 자세한 내용은 ResultWriter 섹션을 참조하세요.

States.Runtime

처리할 수 없는 일부 예외로 인해 실행이 실패했습니다. 이러한 문제는 런타임 시 null JSON 페이로드에 InputPath 또는 OutputPath의 적용을 시도하는 등의 오류로 발생하는 경우가 많습니다. States.Runtime 오류를 검색할 수 없으며 이 오류로 인해 실행이 항상 실패합니다. States.ALL에서 Retry 또는 Catch를 사용해도 States.Runtime 오류를 포착하지 못합니다.

States.TaskFailed

실행 중에 Task 상태에 오류가 발생했습니다. Retry 또는 Catch에서 사용될 때 States.TaskFailed는 States.Timeout을 제외한 모든 알려진 오류 이름과 일치하는 와일드카드 역할을 합니다.

States.Timeout

Task 상태가 TimeoutSeconds 값보다 오랜 시간을 실행했거나 HeartbeatSeconds 값보다 오랜 시간 동안 하트비트를 보내지 못했습니다.

또한 상태 시스템이 지정된 TimeoutSeconds 값보다 오래 실행되면 실행이 실패하고 States.Timeout 오류가 발생합니다.

상태는 다른 이름으로 오류를 보고할 수 있습니다. 하지만 이러한 오류 이름은 States. 접두사로 시작할 수 없습니다.

가장 좋은 방법은 프로덕션 코드가 AWS Lambda 서비스 예외 (Lambda.ServiceException 및 Lambda.SdkClientException)를 처리하는지 확인하는 것입니다. 자세한 내용은 Lambda 서비스 예외 처리 섹션을 참조하세요.

> **Note**
>
> Lambda에서 처리되지 않은 오류는 오류 출력에서 Lambda.Unknown으로 보고됩니다. 메모리 부족 오류 및 함수 시간 초과도 여기에 포함됩니다. Lambda.Unknown, States.ALL 또는 States.TaskFailed를 일치시켜 이러한 오류를 처리할 수 있습니다. Lambda에서 최대 간접 호출 수에 도달하면 오류는 Lambda.TooManyRequestsException입니다. Lambda 함수 오류에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 오류 처리 및 자동 재시도를 참조하세요.

오류 후 재시도

Task, Parallel 및 Map 상태에 Retry라는 이름의 필드가 있을 수 있습니다. 이 필드의 값은 Retrier라고 알려진 객체의 배열이어야 합니다. 개별 Retrier는 특정 재시도 횟수를 나타내며, 보통 점점 시간 간격이 증가합니다.

이러한 상태 중 하나에서 오류를 보고하고 Retry 필드가 있으면 Step Functions에서 배열에 나열된 순서대로 Retire 전체를 스캔합니다. Retirer ErrorEquals 필드 값에 오류 이름이 나타나면 상태 시스템은 Retry 필드에 정의된 대로 재시도를 시도합니다.

redriven 실행에서 Retire가 정의된 태스크 상태, Parallel, 또는 Inline Map 상태를 다시 실행하면 redrive에 대한 최대 시도 횟수가 허용되도록 이러한 상태의 재시도 횟수가 0으로 재설정됩니다. redriven 실행의 경우 콘솔을 사용하여 이러한 상태의 개별 재시도를 추적할 수 있습니다. 자세한 내용은 실행 Redriving에서 redriven 실행 재시도 동작 섹션을 참조하세요.

Retrier에는 다음 필드가 포함됩니다.

> **Note**
>
> 재시도는 상태 변환으로 취급됩니다. 상태 전환이 결제에 미치는 영향은 Step Functions 요금을 참조하세요.

`ErrorEquals`(필수)

오류 이름과 일치하는 문자열 배열(비어 있지 않음). 상태에서 오류를 보고하면 Step Functions는 Retrier 전체를 스캔합니다. 오류 이름이 이 어레이에 표시되면, 이 Retrier에 설명된 재시도 정책이 실행됩니다.

`IntervalSeconds`(선택 사항)

처음 재시도하기 전에 기다리는 시간(초)을 나타내는 정수입니다(기본값: 1). `IntervalSeconds`의 최대값은 99999999입니다.

`MaxAttempts`(선택 사항)

양수로, 최대 재시도 횟수를 나타냅니다(기본값 3). 지정된 횟수보다 많이 오류가 발생하는 경우 재시도가 중지되고 일반 오류 처리가 다시 시작됩니다. 값을 `0`으로 지정하면 오류가 결코 재시도되지 않습니다. `MaxAttempts`의 최대값은 99999999입니다.

`BackoffRate`(선택 사항)

각 재시도 후의 간격이 늘어나도록 `IntervalSeconds`에서 지정된 재시도 횟수를 곱하는 승수입니다. 기본적으로 `BackoffRate` 값은 `2.0`씩 증가합니다.

예를 들어 `IntervalSeconds`가 3, `MaxAttempts`가 3, `BackoffRate`가 2라고 가정해보겠습니다. 첫 번째 재시도는 오류가 발생한 지 3초 후에 수행됩니다. 두 번째 재시도는 첫 번째 재시도 후 6초 후에 수행됩니다. 반면 세 번째 재시도는 두 번째 재시도 후 12초 후에 수행됩니다.

`MaxDelaySeconds`(선택 사항)

재시도 간격을 최대로 늘릴 수 있는 최대값(초)을 설정하는 양의 정수입니다. 이 필드를 `BackoffRate` 필드와 함께 사용하면 유용합니다. 이 필드에 지정된 값은 각 연속 재시도에 적용되

는 백오프 비율 승수로 인한 기하급수적 대기 시간을 제한합니다. `MaxDelaySeconds`에 0보다 크고 31622401보다 작은 값을 지정해야 합니다.

이 값을 지정하지 않으면 Step Functions는 재시도 간의 대기 시간을 제한하지 않습니다.

`JitterStrategy`(선택 사항)

연속 재시도 간의 대기 시간에 지터를 포함할지 여부를 결정하는 문자열입니다. 지터는 무작위 지연 간격으로 동시 재시도를 분산시켜 줄입니다. 이 문자열은 `FULL` 또는 `NONE`을 해당 값으로 수락합니다. 기본값은 `NONE`입니다.

예를 들어 `MaxAttempts`를 3으로, `IntervalSeconds`를 2로, `BackoffRate`를 2로 설정했다고 가정해 보겠습니다. 첫 번째 재시도는 오류가 발생한 지 2초 후에 수행됩니다. 두 번째 재시도는 첫 번째 재시도 후 4초 후에 수행되고 세 번째 재시도는 두 번째 재시도 후 8초 후에 수행됩니다. `JitterStrategy`를 `FULL`로 설정하면 첫 번째 재시도 간격은 0~2초 사이에서 무작위로, 두 번째 재시도 간격은 0~4초 사이에서 무작위로, 세 번째 재시도 간격은 0~8초 사이에서 무작위로 설정됩니다.

Retry 필드 예제

이 섹션에는 다음 `Retry` 필드 예제가 포함되어 있습니다.

- Retry with BackoffRate
- Retry with MaxDelaySeconds
- Retry all errors except States.Timeout
- Complex retry scenario

> **Tip**
>
> 오류 처리 워크플로 예제를 AWS 계정에 배포하려면 AWS Step Functions 워크숍의 오류 처리 모듈을 참조하세요.

예제 1 - BackoffRate를 사용하여 재시도

다음 예제의 `Retry`는 3초 동안 기다린 후 첫 번째 재시도를 두 번 수행합니다. 지정한 `BackoffRate`에 따라 Step Functions는 최대 재시도 횟수에 도달할 때까지 각 재시도 간의 간격을 늘립니다. 다음 예제에서는 첫 번째 재시도 후 3초 동안 기다린 후 두 번째 재시도가 시작합니다.

```
"Retry": [ {
   "ErrorEquals": [ "States.Timeout" ],
   "IntervalSeconds": 3,
   "MaxAttempts": 2,
   "BackoffRate": 1
} ]
```

예제 2 - MaxDelaySeconds를 사용하여 재시도

다음 예시에서는 재시도를 3번 수행하고 BackoffRate로 인한 대기 시간을 5초로 제한합니다. 첫 번째 재시도는 3초 동안 기다린 후에 수행됩니다. MaxDelaySeconds에서 설정한 최대 대기 시간 한도로 인해 두 번째 및 세 번째 재시도는 이전 재시도 후 5초 후에 수행됩니다.

```
"Retry": [ {
    "ErrorEquals": [ "States.Timeout" ],
    "IntervalSeconds": 3,
    "MaxAttempts": 3,
    "BackoffRate":2,
    "MaxDelaySeconds": 5,
    "JitterStrategy": "FULL"
} ]
```

MaxDelaySeconds를 설정하지 않으면 두 번째 재시도는 첫 번째 재시도 후 6초 후에 수행되고 세 번째 재시도는 두 번째 재시도 후 12초 후에 수행됩니다.

예제 3 - States.Timeout을 제외한 모든 오류 재시도

Retrier의 ErrorEquals 필드에 표시되는 예약된 이름 States.ALL은 모든 오류 이름을 나타내는 와일드카드입니다. 이 이름은 ErrorEquals 어레이에 하나만 표시되어야 하며, Retry 어레이의 마지막 Retrier에 표시되어야 합니다. States.TaskFailed 이름은 와일드카드 역할도 하며 States.Timeout를 제외한 모든 오류와 일치합니다.

다음은 States.Timeout을 제외한 모든 오류를 재시도하는 Retry 필드의 예제입니다.

```
"Retry": [ {
   "ErrorEquals": [ "States.Timeout" ],
   "MaxAttempts": 0
}, {
   "ErrorEquals": [ "States.ALL" ]
} ]
```

예제 4 — 복잡한 재시도 시나리오

Retrier의 파라미터는 단일 상태 실행의 맥락에서 해당 Retrier에 대한 모든 방문에 적용됩니다.

다음 Task 상태를 고려하십시오.

```
"X": {
   "Type": "Task",
   "Resource": "arn:aws:states:us-east-1:123456789012:task:X",
   "Next": "Y",
   "Retry": [ {
      "ErrorEquals": [ "ErrorA", "ErrorB" ],
      "IntervalSeconds": 1,
      "BackoffRate": 2.0,
      "MaxAttempts": 2
   }, {
      "ErrorEquals": [ "ErrorC" ],
      "IntervalSeconds": 5
   } ],
   "Catch": [ {
      "ErrorEquals": [ "States.ALL" ],
      "Next": "Z"
   } ]
}
```

이 작업은 연속 4번 실패하며 오류 이름 ErrorA, ErrorB, ErrorC 및 ErrorB가 출력됩니다. 결과로 다음이 발생합니다.

- 첫 번째 오류 2개는 첫 번째 Retrier와 일치하며 이로 인해 1초 및 2초 동안 대기합니다.
- 세 번째 오류는 두 번째 Retrier와 일치하며 이로 인해 5초 동안 대기합니다.
- 네 번째 오류는 첫 번째 Retrier와 일치합니다. 하지만 해당 특정 오류에 대한 최대 재시도 횟수 2회(MaxAttempts)에 이미 도달했습니다. 따라서 해당 Retrier가 실패하고 실행은 Catch 필드를 통해 워크플로를 Z 상태로 리디렉션합니다.

폴백 상태

Task, Map 및 Parallel 상태 각각에는 Catch라는 필드가 있을 수 있습니다. 이 필드의 값은 catchers라는 객체의 어레이이어야 합니다.

Catcher에는 다음 필드가 포함됩니다.

`ErrorEquals`(필수)

오류 이름에 연결되는 문자열 배열(비어 있지 않음)로, 동일한 이름의 Retrier 필드와 동일하게 지정됩니다.

`Next`(필수)

상태 머신의 상태 이름 중 하나와 정확히 일치하는 문자열입니다.

`ResultPath`(선택 사항)

Catcher에서 `Next` 필드에 지정된 상태로 전송하는 입력을 결정하는 경로입니다.

상태에서 오류를 보고하고 `Retry` 필드가 없거나 재시도를 통해 오류가 해결되지 않으면 Step Function는 배열에 나열된 순서대로 Catcher 전체를 스캔합니다. Catcher의 `ErrorEquals` 필드 값에 오류 이름이 표시되면 상태 머신이 `Next` 필드에 이름이 지정된 상태로 전환됩니다.

Catcher의 `ErrorEquals` 필드에 표시되는 예약된 이름 `States.ALL`은 모든 오류 이름을 나타내는 와일드카드입니다. 이 이름은 `ErrorEquals` 어레이에 하나만 표시되어야 하며, `Catch` 어레이의 마지막 Catcher에 표시되어야 합니다. `States.TaskFailed` 이름은 와일드카드 역할도 하며 `States.Timeout`를 제외한 모든 오류와 일치합니다.

다음은 Lambda 함수에서 처리되지 않은 Java 예외를 출력하면 `RecoveryState` 상태로 전환되는 `Catch` 필드의 예제입니다. 그렇지 않으면 필드가 다음과 같이 `EndState` 상태로 전환됩니다.

```
"Catch": [ {
   "ErrorEquals": [ "java.lang.Exception" ],
   "ResultPath": "$.error-info",
   "Next": "RecoveryState"
}, {
   "ErrorEquals": [ "States.ALL" ],
   "Next": "EndState"
} ]
```

> **Note**
>
> 각 Catcher는 처리할 오류를 여러 개 지정할 수 있습니다.

오류 출력

Step Functions가 catch 이름에 지정된 상태로 전환되면 객체에는 일반적으로 `Cause` 필드가 포함됩니다. 이 필드 값은 육안으로 읽을 수 있는 오류 설명입니다. 이 객체를 오류 출력이라고 합니다.

예 예에서 첫 번째 Catcher에는 `ResultPath` 필드가 들어 있습니다. 이 예는 상태의 최상위에 있는 `ResultPath` 필드와 유사하게 작동하여 다음과 같은 두 가지 작업을 수행할 수 있습니다.

- 상태 실행 결과를 가져와 상태 입력 전체 또는 일부분을 덮어씁니다.
- 결과를 가져와 입력에 추가합니다. Catcher에서 오류를 처리하는 경우 상태 실행 결과가 오류 출력이 됩니다.

따라서 이 예제의 첫 번째 Catcher의 경우 입력에 `error-info` 필드가 아직 없으면 Catcher에서 오류 출력을 이 필드로 입력에 추가합니다. 그러면 Catcher가 전체 입력을 `RecoveryState`로 전송합니다. 두 번째 Catcher의 경우 오류 출력이 입력을 덮어쓰고 Catcher는 오류 출력만 `EndState`에 전송합니다.

> **Note**
>
> `ResultPath` 필드를 지정하지 않으면, 기본값이 `$`로 설정되며 전체 입력을 선택하여 덮어씁니다.

상태에 `Retry` 및 `Catch` 필드가 모두 있으면 Step Functions는 적절한 Retrier를 먼저 사용합니다. 재시도 정책에서 오류를 해결하지 못하면 Step Functions는 일치하는 Catcher 전환을 적용합니다.

페이로드 및 서비스 통합으로 인한 오류

Catcher는 문자열 페이로드를 출력으로 반환합니다. Amazon Athena 또는 AWS CodeBuild와 같은 서비스 통합을 사용하는 경우 `Cause` 문자열을 JSON으로 변환할 수 있습니다. 다음은 내장 함수가 있는 `Pass` 상태에서 `Cause` 문자열을 JSON으로 변환하는 방법을 보여주는 예제입니다.

```
"Handle escaped JSON with JSONtoString": {
  "Type": "Pass",
  "Parameters": {
    "Cause.$": "States.StringToJson($.Cause)"
  },
  "Next": "Pass State with Pass Processing"
```

```
},
```

Retry 및 Catch를 사용하는 상태 시스템 예제

다음 예제에 정의된 상태 시스템에는 Lambda 함수가 2개 있습니다. 하나는 항상 실패하는 함수이고 다른 하나는 상태 시스템에 정의된 시간 제한이 발생할 수 있도록 충분히 오래 기다리는 함수입니다.

다음은 항상 실패하는 Node.js Lambda 함수 정의로, `error` 메시지를 반환합니다. 다음 상태 시스템 예제에서 이 Lambda 함수 이름은 `FailFunction`입니다. Lambda 함수를 만드는 방법은 1단계: Lambda 함수 생성 섹션을 참조하세요.

```
exports.handler = (event, context, callback) => {
    callback("error");
};
```

다음은 10초 동안 대기하는 Node.js Lambda 함수 정의입니다. 다음 상태 시스템 예제에서 이 Lambda 함수 이름은 `sleep10`입니다.

> **Note**
>
> Lambda 콘솔에서 이 Lambda 함수를 만들 때 고급 설정 섹션의 제한 시간 값을 3초(기본값)에서 11초로 변경해야 합니다.

```
exports.handler = (event, context, callback) => {
    setTimeout(function(){
    }, 11000);
};
```

Retry를 사용하여 실패 처리

이 상태 머신은 `Retry` 필드를 사용하여 실패하고 오류 이름 `HandledError`를 출력하는 함수를 재시도합니다. 재시도 간 지수 백오프를 사용하여 이 함수를 두 번 재시도합니다.

```
{
   "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda
 function",
   "StartAt": "HelloWorld",
```

```
   "States": {
      "HelloWorld": {
         "Type": "Task",
         "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction",
         "Retry": [ {
            "ErrorEquals": ["HandledError"],
            "IntervalSeconds": 1,
            "MaxAttempts": 2,
            "BackoffRate": 2.0
         } ],
      "End": true
      }
   }
}
```

이 변형에서는 사전 정의된 오류 코드 States.TaskFailed를 사용하며 이 오류 코드는 Lambda 함수에서 출력하는 오류와 일치합니다.

```
{
   "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda
 function",
   "StartAt": "HelloWorld",
   "States": {
      "HelloWorld": {
         "Type": "Task",
         "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction",
         "Retry": [ {
            "ErrorEquals": ["States.TaskFailed"],
            "IntervalSeconds": 1,
            "MaxAttempts": 2,
            "BackoffRate": 2.0
         } ],
         "End": true
      }
   }
}
```

> **Note**
>
> 모범 사례로서, Lambda 함수를 참조하는 작업이 Lambda 서비스 예외를 처리해야 합니다. 자세한 내용은 Lambda 서비스 예외 처리 섹션을 참조하세요.

Catch를 사용하여 실패 처리

이 예제에서는 Catch 필드를 사용합니다. Lambda 함수에서 오류를 출력하면 오류가 포착되고 상태 시스템이 fallback 상태로 전환됩니다.

```
{
   "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda
 function",
   "StartAt": "HelloWorld",
   "States": {
      "HelloWorld": {
         "Type": "Task",
         "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction",
         "Catch": [ {
            "ErrorEquals": ["HandledError"],
            "Next": "fallback"
         } ],
         "End": true
      },
      "fallback": {
         "Type": "Pass",
         "Result": "Hello, AWS Step Functions!",
         "End": true
      }
   }
}
```

이 변형에서는 사전 정의된 오류 코드 States.TaskFailed를 사용하며 이 오류 코드는 Lambda 함수에서 출력하는 오류와 일치합니다.

```
{
   "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda
 function",
   "StartAt": "HelloWorld",
   "States": {
      "HelloWorld": {
         "Type": "Task",
         "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction",
         "Catch": [ {
            "ErrorEquals": ["States.TaskFailed"],
            "Next": "fallback"
         } ],
```

```
      "End": true
      },
      "fallback": {
         "Type": "Pass",
         "Result": "Hello, AWS Step Functions!",
         "End": true
      }
   }
}
```

Retry를 사용하여 제한 시간 처리

이 상태 시스템은 Retry 필드를 사용하여 TimeoutSeconds에 지정된 제한 시간 값에 따라 제한 시간이 초과된 Task 상태를 재시도합니다. Step Functions는 이 Task 상태에서 Lambda 함수 간접 호출을 두 번 재시도하며 재시도 간에 지수 백오프를 적용합니다.

```
{
   "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda
 function",
   "StartAt": "HelloWorld",
   "States": {
      "HelloWorld": {
         "Type": "Task",
         "Resource": "arn:aws:lambda:us-east-1:123456789012:function:sleep10",
         "TimeoutSeconds": 2,
         "Retry": [ {
            "ErrorEquals": ["States.Timeout"],
            "IntervalSeconds": 1,
            "MaxAttempts": 2,
            "BackoffRate": 2.0
         } ],
         "End": true
      }
   }
}
```

Catch를 사용하여 제한 시간 처리

이 예제에서는 Catch 필드를 사용합니다. 시간 초과가 발생하면 상태 머신이 fallback 상태로 전환됩니다.

```
{
```

```
  "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda
function",
  "StartAt": "HelloWorld",
  "States": {
     "HelloWorld": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:us-east-1:123456789012:function:sleep10",
        "TimeoutSeconds": 2,
        "Catch": [ {
           "ErrorEquals": ["States.Timeout"],
           "Next": "fallback"
        } ],
        "End": true
     },
     "fallback": {
        "Type": "Pass",
        "Result": "Hello, AWS Step Functions!",
        "End": true
     }
  }
}
```

> **Note**
>
> `ResultPath`를 사용하여 상태 입력과 오류를 저장할 수 있습니다. ResultPath a에 오류와 입력을 모두 포함하는 데 사용합니다. `Catch`을(를) 참조하세요.

다른 서비스에서 AWS Step Functions 간접 호출

상태 시스템을 간접적으로 호출하도록 다른 여러 서비스를 구성할 수 있습니다. 상태 시스템의 워크플로 유형에 따라 상태 시스템을 비동기적이나 동기적으로 간접 호출할 수 있습니다. 상태 시스템을 동기적으로 간접 호출하려면 `StartSyncExecution` API 직접 호출 또는 Amazon API Gateway와 Express 워크플로 통합을 사용합니다. 비동기 호출을 사용하면 Step Functions는 작업 토큰이 반환될 때까지 워크플로 실행을 일시 중지합니다. 하지만 작업 토큰을 기다리면 워크플로가 동기화됩니다.

Step Functions를 간접적으로 호출하도록 구성할 수 있는 서비스는 다음과 같습니다.

- AWS Lambda(`StartExecution` 호출 사용)
- Amazon API Gateway

- Amazon EventBridge
- AWS CodePipeline
- AWS IoT 규칙 엔진
- AWS Step Functions

Step Functions 간접 호출에는 `StartExecution` 할당량이 적용됩니다. 자세한 내용은 다음을 참조하세요.

- 할당량

Step Functions에서 읽기 일관성

AWS Step Functions 에서 상태 시스템 업데이트가 드디어 일관성을 갖추었습니다. 몇 초 이내의 모든 `StartExecution` 호출에서는 업데이트된 정의 및 `roleArn`(IAM 역할에 대한 Amazon 리소스 이름)이 사용됩니다. `UpdateStateMachine` 호출 직후 시작된 실행에서는 이전의 상태 시스템 정의 및 `roleArn`이 사용될 수 있습니다.

자세한 내용은 다음을 참조하세요.

- AWS Step Functions API 참조의 `UpdateStateMachine`
- 시작하기 AWS Step Functions에서 워크플로 업데이트

Step Functions에서 태그 지정

AWS Step Functions은 상태 머신(표준 및 Express) 및 활동의 태깅을 지원합니다. 이 기능은 리소스와 관련된 비용을 추적 및 관리하고 AWS Identity and Access Management(IAM) 정책에서 보안을 개선하는 데 도움이 될 수 있습니다. Step Functions 리소스에 태그를 지정하면 AWS Resource Groups에서 관리할 수 있습니다. 리소스 그룹에 대한 자세한 내용은 AWS Resource Groups 사용 설명서를 참조하세요.

태그 기반 권한 부여의 경우 다음 예제와 같은 상태 시스템 실행 리소스는 상태 시스템과 연결된 태그를 상속합니다.

```
arn:<partition>:states:<Region>:<account-id>:execution:<StateMachineName>:<ExecutionId>
```

DescribeExecution 또는 실행 리소스 ARN을 지정하는 다른 API를 직접적으로 호출하면 태그 기반 권한 부여를 수행하는 동안 Step Functions에서 상태 시스템과 연결된 태그를 사용하여 요청을 수락하거나 거부합니다. 이렇게 하면 상태 시스템 수준에서 상태 시스템 실행에 대한 액세스를 허용하거나 거부할 수 있습니다.

리소스 태그 지정에 관련된 제한을 검토하려면 태그 지정과 관련된 제한 단원을 참조하십시오.

주제

- 비용 할당을 위한 태그 지정
- 보안을 위한 태그 지정
- Step Functions 콘솔에서 태그 보기 및 관리
- Step Functions API 작업으로 태그 관리

비용 할당을 위한 태그 지정

비용 할당을 위해 Step Functions 리소스를 구성하고 식별하려면 상태 시스템이나 활동의 목적을 식별하는 메타데이터 태그를 추가하면 됩니다. 이 기능은 리소스가 많을 때 특히 유용합니다. 비용 할당 태그를 사용하여 비용 구조를 반영하도록 AWS 청구서를 구성할 수 있습니다. 이렇게 하려면 태그 키와 값이 포함될 AWS 계정 청구서를 가져오도록 등록합니다. 자세한 내용은 AWS Billing 사용 설명서에서 월간 비용 할당 보고서 설정을 참조하세요.

예를 들어 다음과 같이 Step Functions 리소스의 비용 센터와 목적을 나타내는 태그를 추가할 수 있습니다.

리소스	키	값
StateMachine1	Cost Center	34567
	Application	Image processing
StateMachine2	Cost Center	34567
	Application	Rekognition processing
Activity1	Cost Center	12345
	Application	Legacy database

이 태그 지정 체계에서는 동일한 코스트 센터에서 관련된 작업을 수행하는 2개의 상태 머신을 그룹화할 수 있고, 관련이 없는 활동은 다른 비용 할당 태그를 사용해 태그 지정할 수 있습니다.

보안을 위한 태그 지정

IAM은 태그를 기반으로 리소스에 대한 액세스를 제어할 수 있습니다. 태그를 기반으로 액세스를 제어하려면 IAM 정책의 조건 요소에 리소스 태그에 대한 정보를 제공합니다.

예를 들어 `environment` 키 및 `production` 값과 함께 태그가 포함된 모든 Step Functions 리소스에 대한 액세스를 제한할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "states:TagResource",
                "states:DeleteActivity",
                "states:DeleteStateMachine",
                "states:StopExecution"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/environment": "production"}
            }
        }
    ]
}
```

자세한 내용은 IAM 사용 설명서의 태그를 사용한 액세스 제어를 참조하세요.

Step Functions 콘솔에서 태그 보기 및 관리

Step Functions를 사용하면 Step Functions 콘솔에서 상태 시스템의 태그를 보고 관리할 수 있습니다. 상태 머신의 세부 정보 페이지에서 태그를 선택합니다. 여기에서 상태 머신과 관련된 기존 태그를 볼 수 있습니다.

> **Note**
>
> 활동에 대한 태그를 관리하려면 Step Functions API 작업으로 태그 관리 단원을 참조하십시오.

상태 머신과 관련된 태그를 추가하거나 삭제하려면 태그 관리 버튼을 선택합니다.

1. 상태 머신의 세부 정보 페이지로 이동합니다.
2. 실행 및 정의 옆에 있는 태그를 선택합니다.
3. 태그 관리를 선택합니다.
   - 기존의 태그를 수정하려면 키 및 값 페어를 편집합니다.
   - 기존 태그를 제거하려면 태그 제거를 선택합니다.
   - 새 태그를 추가하려면 태그 추가를 선택하고 키 및 값을 입력합니다.
4. Save를 선택합니다.

Step Functions API 작업으로 태그 관리

Step Functions API를 사용하여 태그를 관리하려면 다음 API 작업을 사용합니다.

- ListTagsForResource
- TagResource
- UntagResource

AWS Step Functions 워크플로 스튜디오

Workflow Studio AWS Step Functions for는 서비스를 AWS 오케스트레이션하여 서버리스 워크플로를 만들 수 있는 로우 코드 시각적 워크플로 디자이너입니다. 이 drag-and-drop 기능이나 내장된 코드 편집기를 사용하여 워크플로를 생성 및 편집하고, 각 상태에 대한 입력 및 출력 필터링 또는 변환 방식을 제어하고, 오류 처리를 구성할 수 있습니다. 상태를 끌어서 놓아 워크플로를 빌드하면 Workflow Studio에서 작업을 검증하고 코드를 자동 생성합니다. 코드 편집기 내에서 생성된 코드를 검토하거나 상태 시스템 정의를 업데이트할 수 있습니다. 작업을 마치면 워크플로를 저장하고 실행한 다음 Step Functions 콘솔에서 결과를 검사할 수 있습니다. 워크플로를 시각적으로 추가하고 수정하여 애플리케이션의 여러 서비스를 오케스트레이션할 수 있습니다.

Step Functions Workflow Studio를 사용하려면 사용하려는 모든 리소스에 대한 올바른 권한을 제공하는 AWS 계정, 및 자격 증명이 필요합니다. 자세한 정보는 시작하기 위한 사전 요구 사항 AWS Step Functions을 참조하세요.

> **Note**
>
> Workflow Studio는 Internet Explorer 11을 지원하지 않습니다. Internet Explorer 11을 사용하고 있고 Workflow Studio를 사용할 때 문제가 발생하면 다른 브라우저를 사용해 봅니다.

Step Functions에서 워크플로를 만들거나 편집할 때 Step Functions 콘솔에서 Workflow Studio에 액세스할 수 있습니다. AWS Application Composer 내에서 Workflow Studio에 액세스할 수도 있습니다. Application Composer의 Workflow Studio는 IaC 도구(예: AWS CloudFormation 템플릿)를 사용하여 구축한 서버리스 애플리케이션에 워크플로를 쉽게 통합할 수 있는 시각적 IaC 환경을 제공합니다. Application Composer에서 Workflow Studio를 사용하면 AWS CloudFormation 템플릿을 사용하여 워크플로를 구축할 수 있습니다. Application Composer 내에서 새 워크플로를 추가하고, 기존 워크플로를 수정하고, 개별 워크플로 단계를 다른 애플리케이션 리소스에 연결할 수 있습니다. Application Composer는 필요한 CloudFormation 리소스와 구성을 자동으로 생성하고 업데이트합니다. 이를 통해 워크플로에 사용되는 모든 리소스를 한 곳에서 만들고 관리할 수 있습니다. 또한 워크플로 프로토타이핑에서 프로덕션 배포까지의 과정을 가속화하는 데도 도움이 됩니다.

Application Composer에서 Workflow Studio를 사용하면 로컬 프로젝트에 직접 연결할 수도 있습니다. 이를 통해 시각적 캔버스와 함께 통합 개발 환경(IDE)에서 작업할 수 있습니다. 자세한 정보는 Application Composer에서 Workflow Studio 사용을 참조하세요.

주제

인터페이스 개요

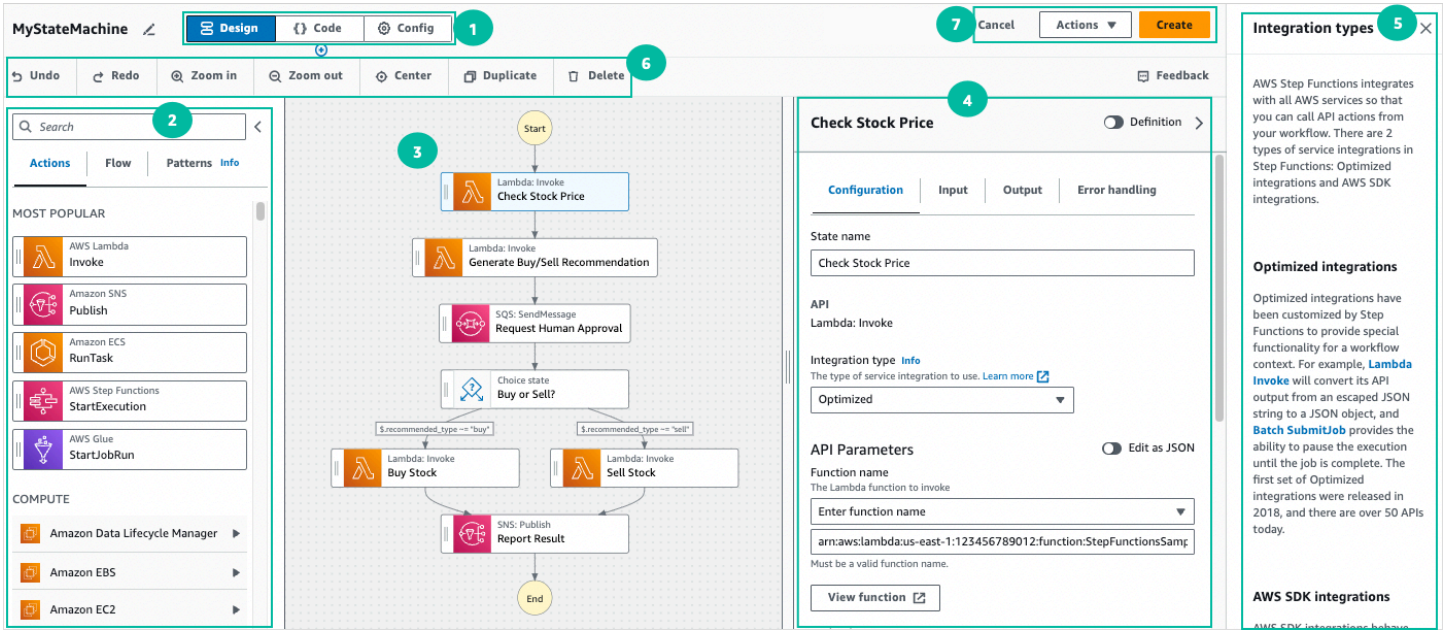
Workflow Studio AWS Step Functions for는 오케스트레이션을 통해 서버리스 워크플로를 만들 수 있는 로우 코드 시각적 워크플로 디자이너입니다. AWS 서비스끌어서 놓기 기능이 있는 Workflow Studio를 사용하면 워크플로 프로토타입을 간편하게 빌드, 편집 및 시각화할 수 있습니다. 또한 Workflow Studio는 Step Functions 콘솔 내에서 Amazon States Language(ASL) 를 사용하여 워크플로 정의를 작성 및 편집할 수 있는 코드 편집기를 기본 제공합니다.

워크플로를 빌드 및 시각화하고 정의를 편집하며 구성을 관리하는 데 도움이 되도록 Workflow Studio는 디자인, 코드 및 구성 등 3가지 모드를 제공합니다. 다음 섹션에서는 이러한 단계를 자세히 설명합니다.

이 주제의 내용

디자인 모드

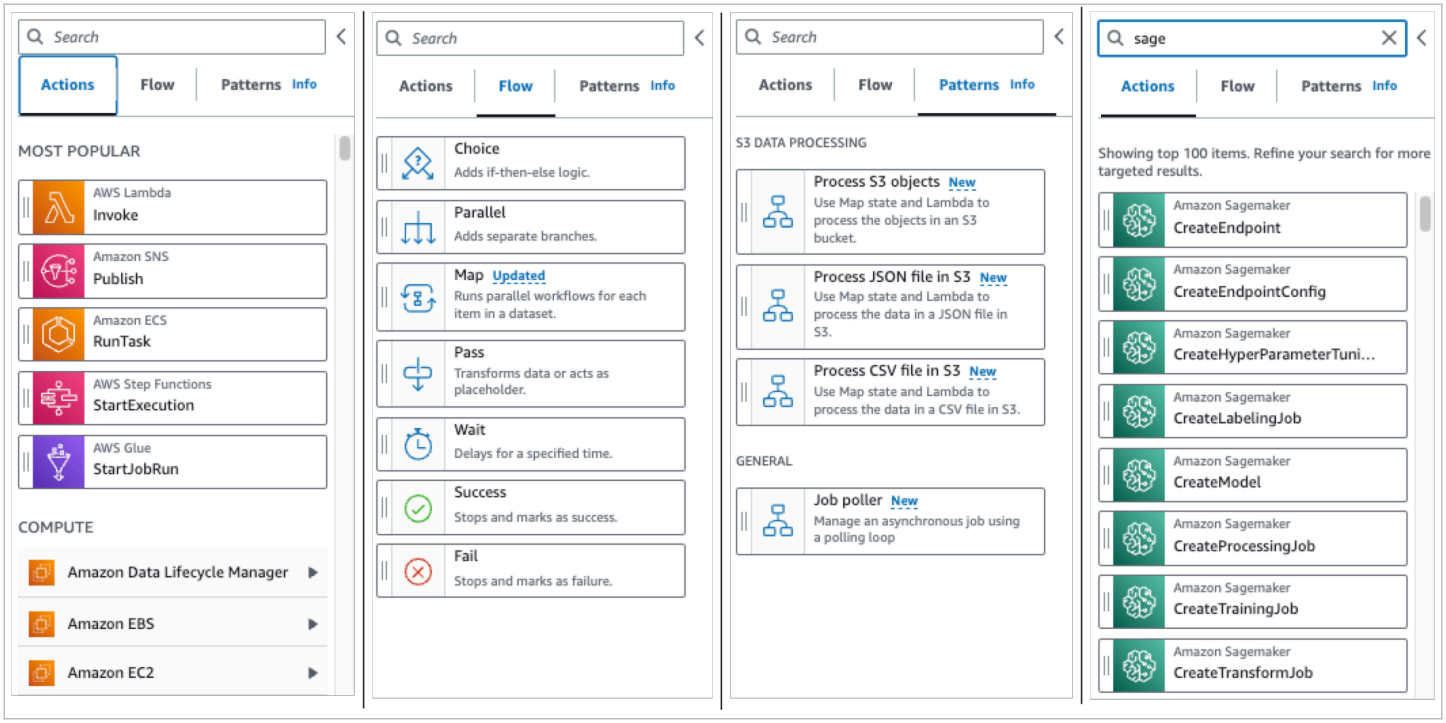
Workflow Studio의 디자인 모드는 프로토타입을 빌드할 때 워크플로를 시각화하는 그래픽 인터페이스를 제공합니다. 다음 이미지에서는 디자인 모드에서 사용할 수 있는 다양한 구성 요소를 보여줍니다.



1. 모드 버튼 — 모드 버튼을 사용하여 Workflow Studio의 디자인, 코드 또는 구성 모드로 전환합니다. 워크플로 ASL 정의에 있는 JSON이 유효하지 않으면 모드를 전환할 수 없습니다.
2. 상태 브라우저에는 다음과 같은 탭 3개가 포함됩니다.
   - Actions 탭은 캔버스의 워크플로 그래프로 끌어서 놓을 수 있는 AWS API 목록을 제공합니다. 각 작업은 태스크 상태 상태를 나타냅니다.
   - 흐름 탭은 캔버스의 워크플로 그래프로 끌어서 놓을 수 있는 흐름 상태 목록을 제공합니다.
   - 패턴 탭에는 다양한 사용 사례에 사용할 수 있는 재사용 가능한 여러 ready-to-use 구성 요소가 있습니다. 예를 들어 이러한 패턴을 사용하여 Amazon S3 버킷의 데이터를 반복 처리할 수 있습니다.
3. Canvas에서 상태를 끌어 워크플로 그래프에 놓고 상태 순서를 변경하고 구성하거나 볼 상태를 선택할 수 있습니다.
4. 이 Inspector 패널에서 캔버스에서 선택한 모든 상태의 속성을 보고 편집할 수 있습니다. 정의 토글을 켜면 현재 선택한 상태가 강조 표시된 상태로 워크플로의 Amazon States Language 코드가 표시됩니다.
5. 도움이 필요할 때 정보 링크를 클릭하면 컨텍스트 정보가 포함된 패널이 열립니다. 이 패널에는 Step Functions 설명서의 관련 주제로 연결되는 링크도 포함되어 있습니다.
6. 디자인 도구 모음 - 실행 취소, 삭제 및 확대와 같은 일반적인 작업을 수행할 수 있는 일련의 버튼이 포함됩니다.
7. 유틸리티 버튼 - 워크플로 저장 또는 해당 ASL 정의를 JSON 또는 YAML 파일로 내보내기와 같은 작업을 수행하는 데 일련의 버튼입니다.

상태 브라우저

상태 브라우저에서 상태를 선택하여 워크플로 그래프로 끌어서 놓을 수 있습니다. Actions AWS 탭은 API 목록을 제공하고 Flow 탭은 흐름 상태 목록을 제공합니다. 패턴 탭은 다양한 사용 ready-to-use 사례에 사용할 수 있는 재사용 가능한 여러 구성 요소를 제공합니다. 상단에 있는 검색 상자를 사용하여 상태 브라우저의 모든 상태를 검색할 수 있습니다.



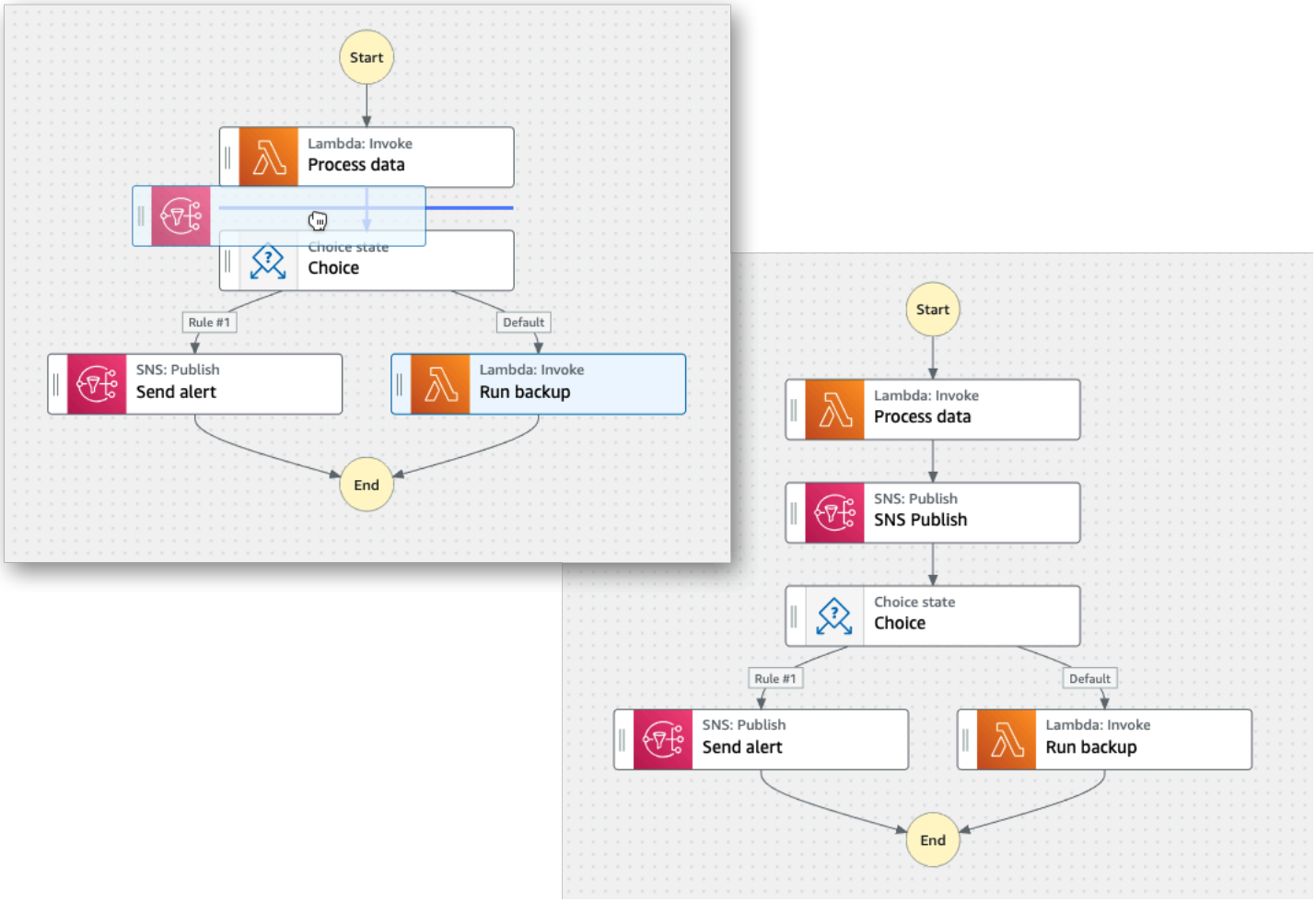
워크플로를 지시하고 제어하는 데 사용할 수 있는 흐름 상태는 7개가 있습니다. 이전 상태에서 모든 입력을 가져오며 대부분의 경우 이전 상태의 입력을 필터링하고 출력을 다음 상태로 필터링할 수 있습니다. 흐름 상태는 다음과 같습니다.

- Choice: 워크플로에 실행 브랜치 사이에 선택 항목을 추가합니다. Inspector의 구성 탭에서 워크플로가 전환될 상태를 결정하는 규칙을 구성할 수 있습니다.
- Parallel: 워크플로에 병렬 실행 브랜치를 추가합니다.
- 맵: 입력 배열의 요소마다 단계를 동적으로 반복합니다. `Parallel` 흐름 상태와 달리 상태는 `Map` 상태는 상태 입력에 있는 배열의 여러 항목에 같은 단계를 실행합니다.
- Pass: 입력을 출력으로 전달할 수 있습니다. (선택 사항) 고정 데이터를 출력에 추가할 수 있습니다.
- Wait: 워크플로를 일정 시간 동안 또는 지정된 시간이나 날짜까지 일시 중지합니다.
- Succeed: 워크플로를 성공적으로 중지합니다.
- Fail: 실패 시 워크플로를 중지합니다.

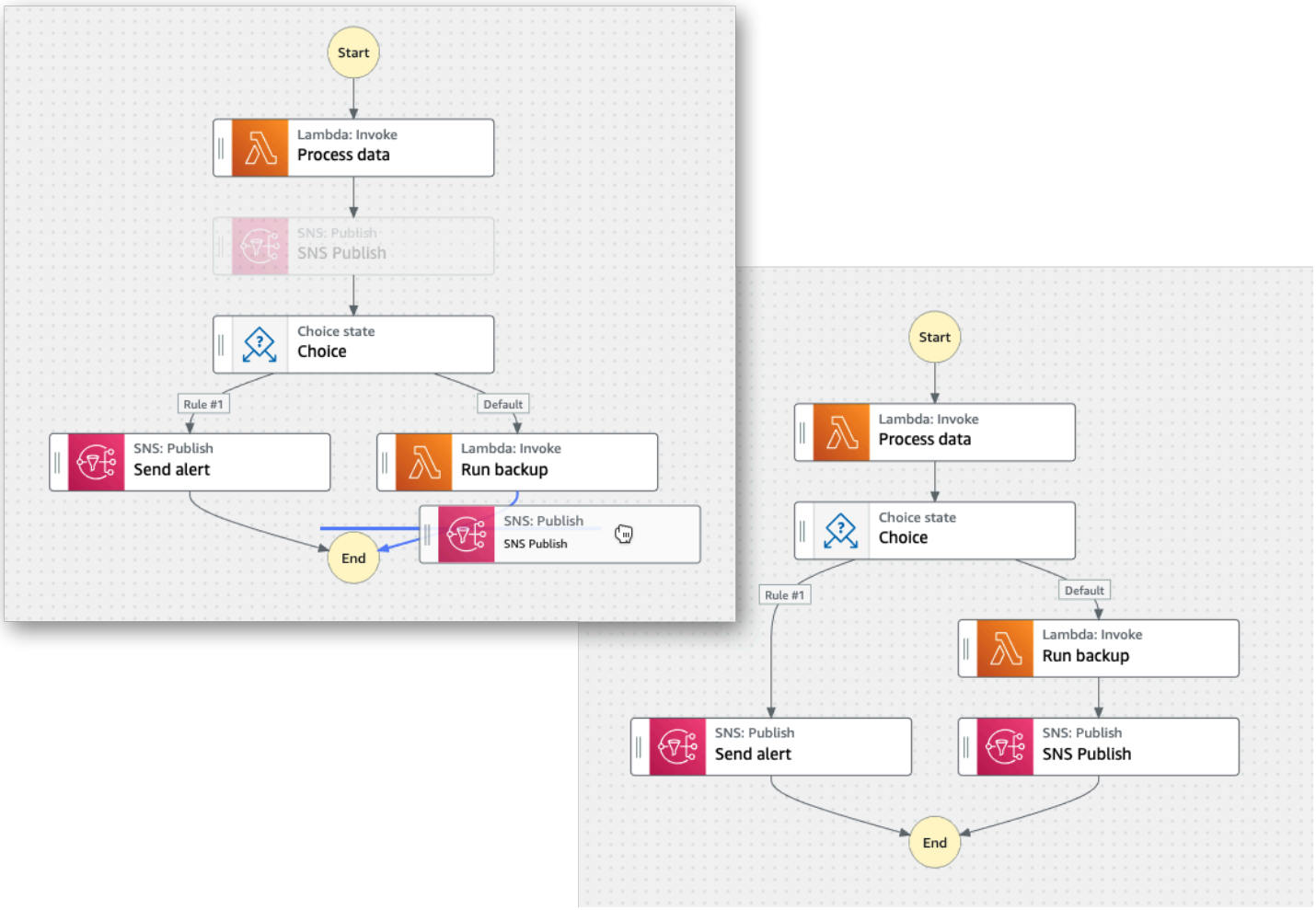
Canvas

워크플로에 추가할 상태를 선택한 후 캔버스로 끌어 워크플로 그래프에 놓습니다. 상태를 끌어서 놓아 워크플로의 다른 위치로 이동할 수도 있습니다. 워크플로가 복잡하면 캔버스 패널에서 모든 워크플로를 보지 못할 수 있습니다. 캔버스 상단에 있는 컨트롤을 사용하여 확대하거나 축소할 수 있습니다. 워크플로 그래프의 다른 부분을 보려면 캔버스에서 워크플로 그래프를 끌면 됩니다.

작업 또는 흐름 탭에서 워크플로 상태를 끌어 워크플로에 놓습니다. 줄에서는 워크플로에서 배치될 위치를 보여줍니다. 새 워크플로 상태가 워크플로에 추가되었으며 해당 코드가 자동으로 생성됩니다.

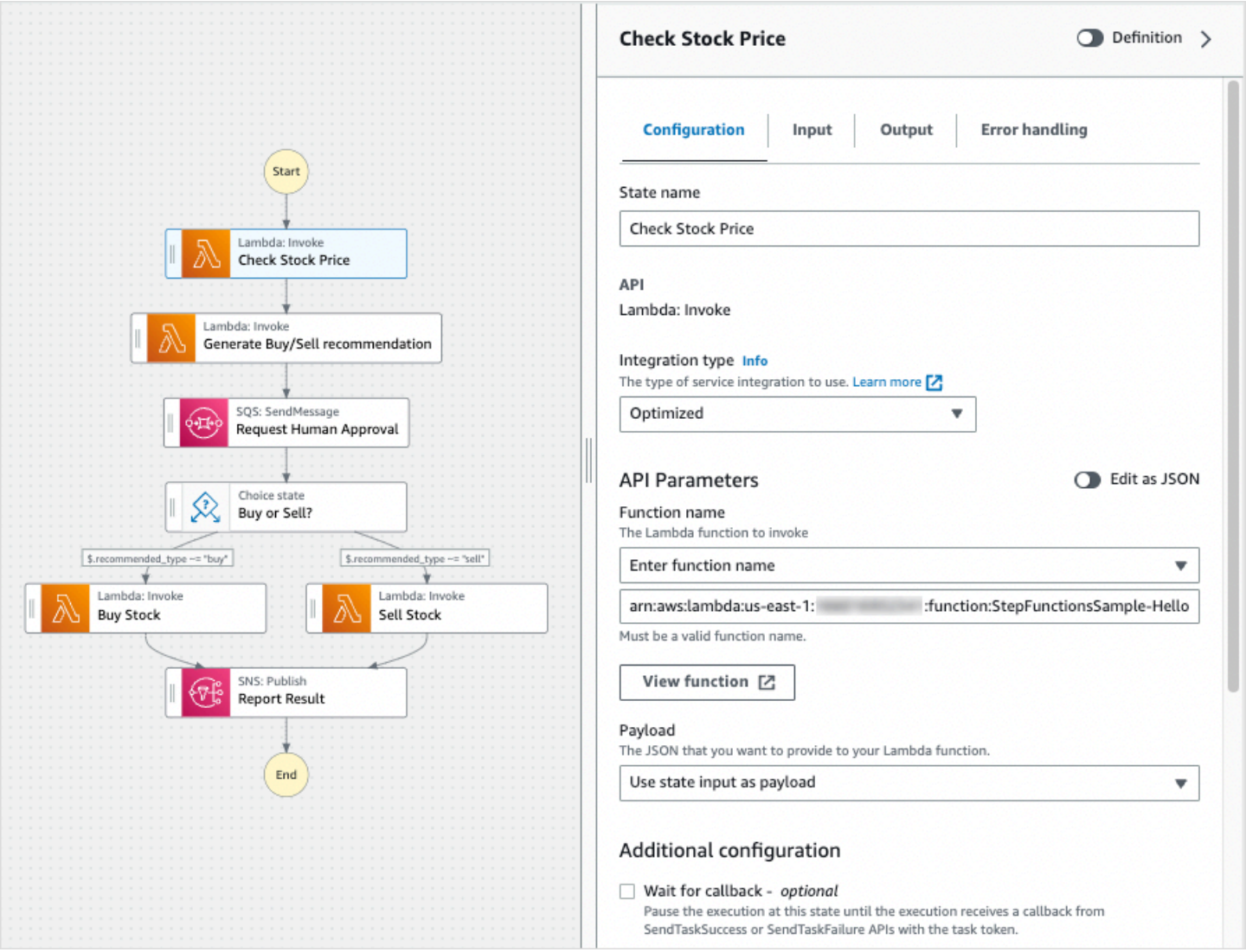


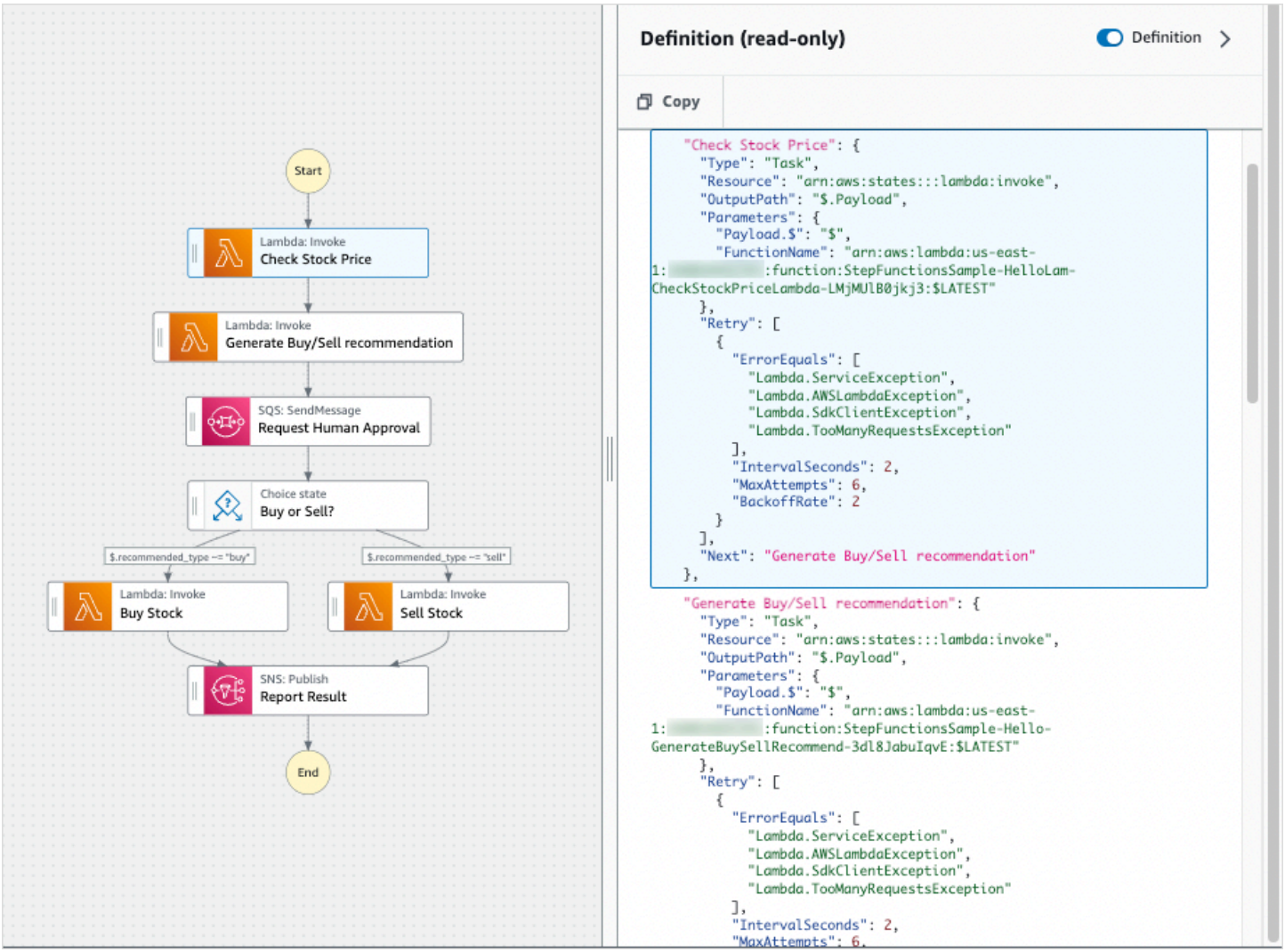
상태 순서를 변경하려면 상태를 워크플로의 다른 위치로 끌면 됩니다.



Inspector

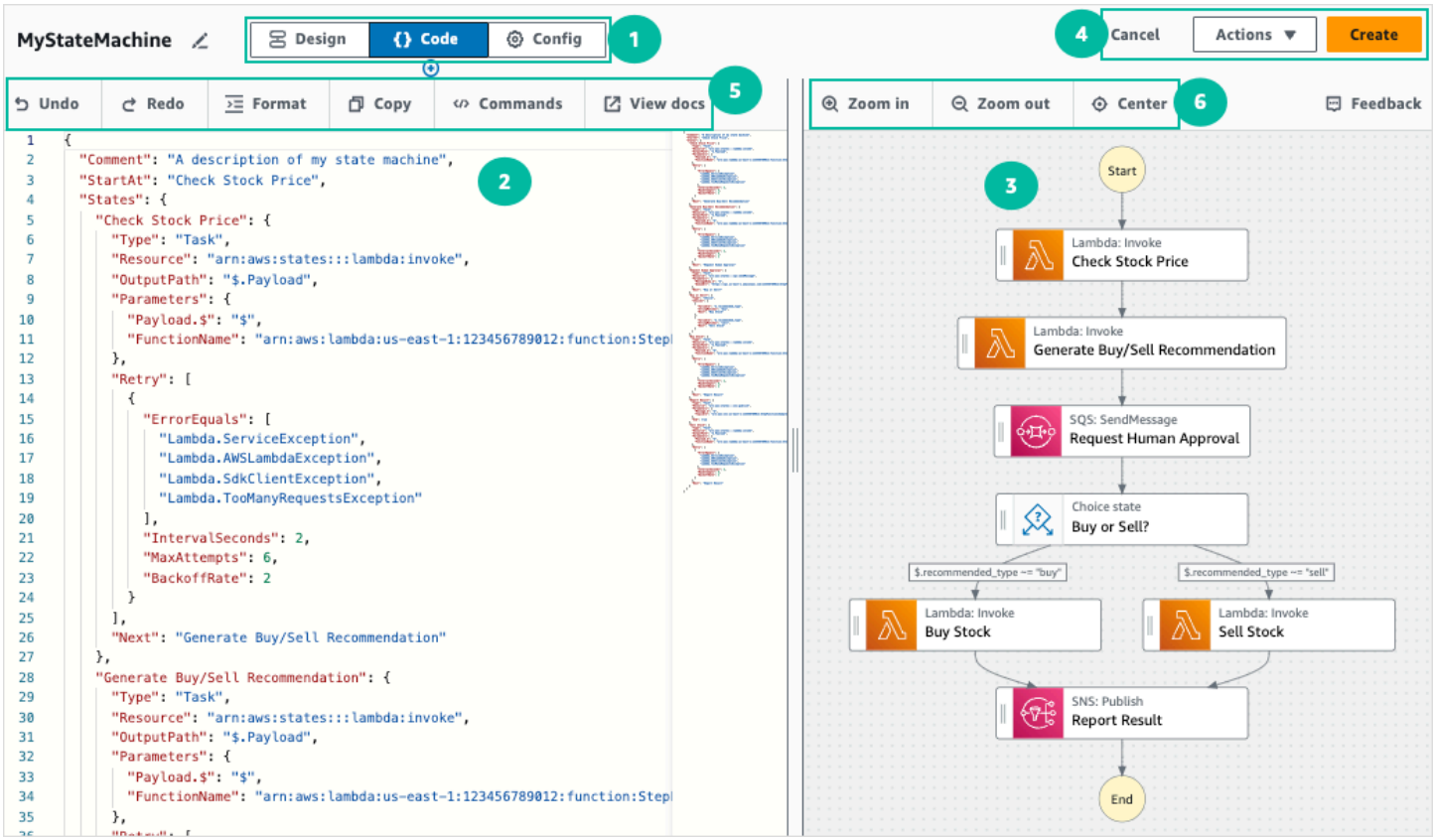
워크플로에 추가하는 상태를 구성할 수 있습니다. 구성하려는 상태를 선택합니다. 그러면 Inspector 패널에 해당 구성 옵션이 표시됩니다. 워크플로 코드에 자동으로 생성된 ASL 정의를 확인하려면 정의 토글을 켭니다. 선택한 상태와 관련된 ASL 정의가 강조 표시됩니다.


Start
Lambda: Invoke
Check Stock Price
Lambda: Invoke
Generate Buy/Sell recommendation
SQS: SendMessage
Request Human Approval
Choice state
Buy or Sell?
$.recommended_type == "buy"
$.recommended_type == "sell"
Lambda: Invoke
Buy Stock
Lambda: Invoke
Sell Stock
SNS: Publish
Report Result
End
Check Stock Price
Definition
Configuration
Input
Output
Error handling
State name
Check Stock Price
API
Lambda: Invoke
Integration type Info
The type of service integration to use. Learn more
Optimized
API Parameters
Edit as JSON
Function name
The Lambda function to invoke
Enter function name
arn:aws:lambda:us-east-1:
:function:StepFunctionsSample-Hello
Must be a valid function name.
View function
Payload
The JSON that you want to provide to your Lambda function.
Use state input as payload
Additional configuration
Wait for callback - optional
Pause the execution at this state until the execution receives a callback from SendTaskSuccess or SendTaskFailure APIs with the task token.



코드 모드

Workflow Studio의 코드 모드는 Step Functions 콘솔 내에서 워크플로의 Amazon States Language(ASL) 정의를 보고, 쓰고, 편집할 수 있는 통합 코드 편집기를 제공합니다. 다음 이미지에서는 코드 모드에서 사용할 수 있는 다양한 구성 요소를 보여줍니다.



1. 모드 버튼 — 모드 버튼을 사용하여 Workflow Studio의 디자인, 코드 또는 구성 모드로 전환합니다. 워크플로 ASL 정의에 있는 JSON이 유효하지 않으면 모드를 전환할 수 없습니다.
2. 코드 편집기에서 Workflow Studio 내에서 워크플로의 ASL 정의 작성하고 편집할 수 있습니다. 코드 편집기는 구문 강조 및 자동 완성과 같은 기능도 제공합니다.
3. 그래프 시각화 창 - 워크플로의 실시간 그래픽 시각화를 보여줍니다.
4. 유틸리티 버튼 - 워크플로 저장 또는 해당 ASL 정의를 JSON 또는 YAML 파일로 내보내기와 같은 작업을 수행하는 데 일련의 버튼입니다.
5. 코드 도구 모음 - 작업 실행 취소나 코드 서식 지정과 같은 일반적인 작업을 수행할 수 있는 일련의 버튼을 포함합니다.
6. 그래프 도구 모음 - 워크플로 그래프 확대 및 축소와 같은 일반적인 작업을 수행할 수 있는 일련의 버튼을 포함합니다.

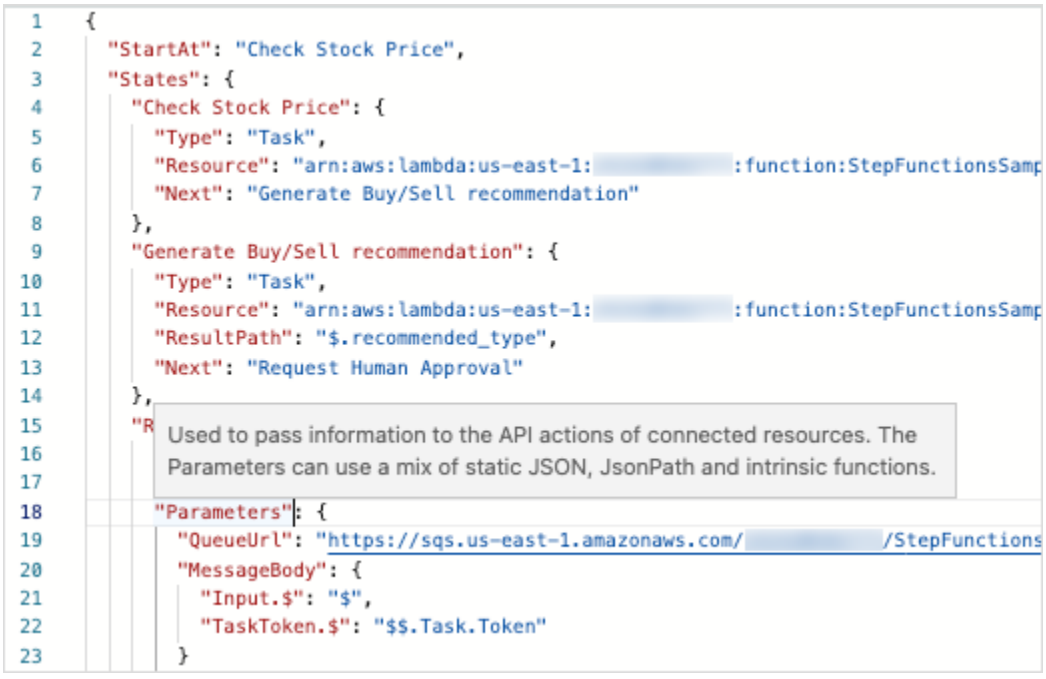
코드 편집기

코드 편집기는 Workflow Studio 내에서 JSON을 사용하여 워크플로 정의를 작성하고 편집할 수 있는 IDE와 유사한 환경을 제공합니다. 코드 편집기에는 구문 강조, 자동 완성 제안, ASL 정의 검증 및 상황에 맞는 도움말 표시와 같은 여러 가지 기능이 포함되어 있습니다. 워크플로 정의를 업데이트하면 그래

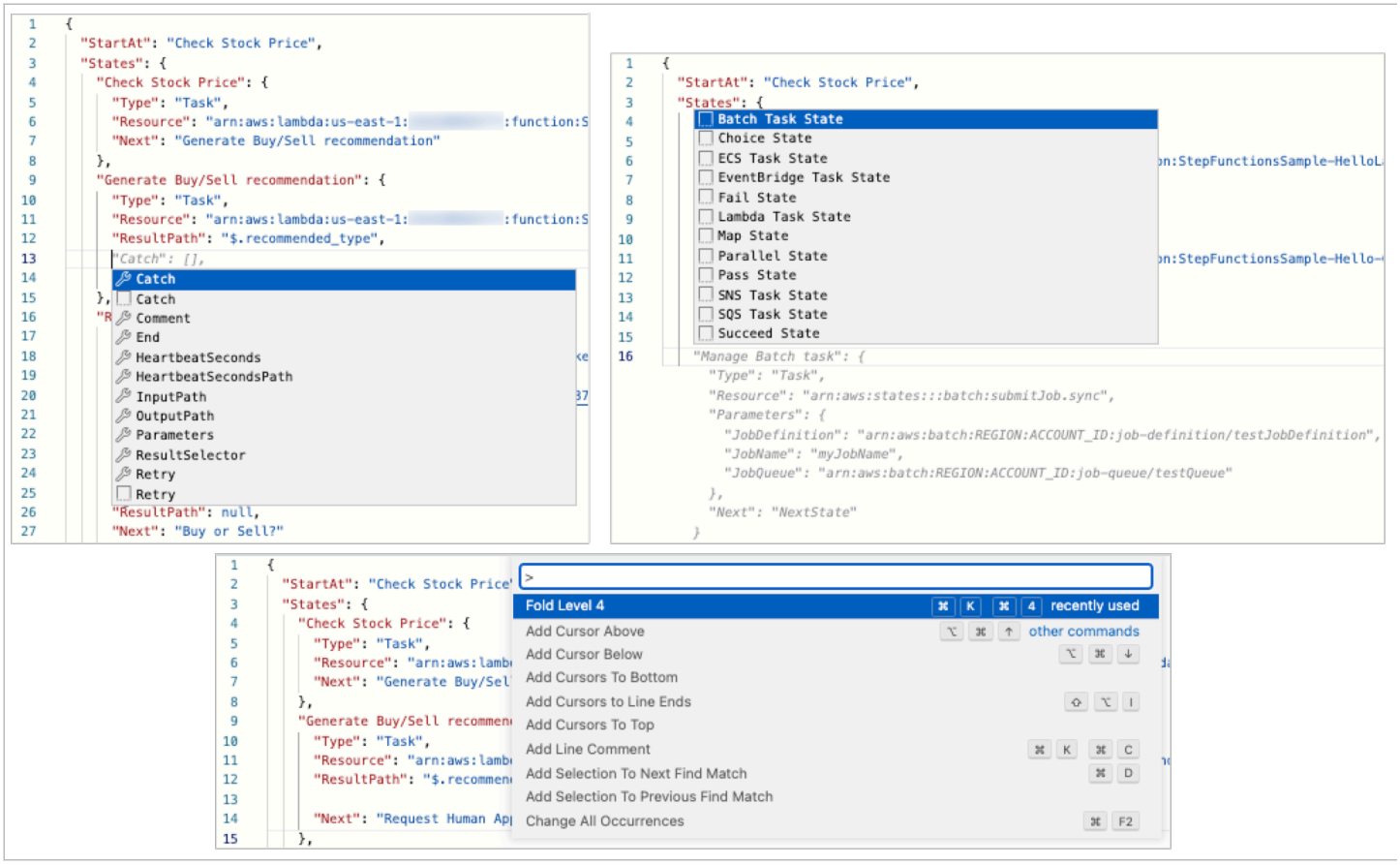
프 시각화 창에서 워크플로 실시간 그래프를 렌더링합니다. 디자인 모드에서도 업데이트된 워크플로 그래프를 확인할 수 있습니다.

디자인 모드 또는 그래프 시각화 창에서 상태를 선택하면 해당 상태의 ASL 정의가 코드 편집기에 강조 표시되어 나타납니다. 디자인 모드나 그래프 시각화 창에서 상태를 재정렬, 삭제 또는 추가하면 워크플로의 ASL 정의가 자동으로 업데이트됩니다.

상황에 맞는 도움말을 도구 설명으로 보려면 워크플로 정의의 아무 필드에나 커서를 올려 놓습니다.



자동 완성 제안에는 워크플로에 포함할 수 있는 필드나 상태의 코드 스니펫이 표시됩니다. 특정 상태에 포함할 수 있는 필드 목록을 보려면 **Ctrl+Space**를 누릅니다. 워크플로의 새 상태에 대한 코드 스니펫을 생성하려면 현재 상태 정의 뒤에서 **Ctrl+Space**를 누릅니다. **F1**을 눌러 사용 가능한 명령 목록도 표시할 수 있습니다.



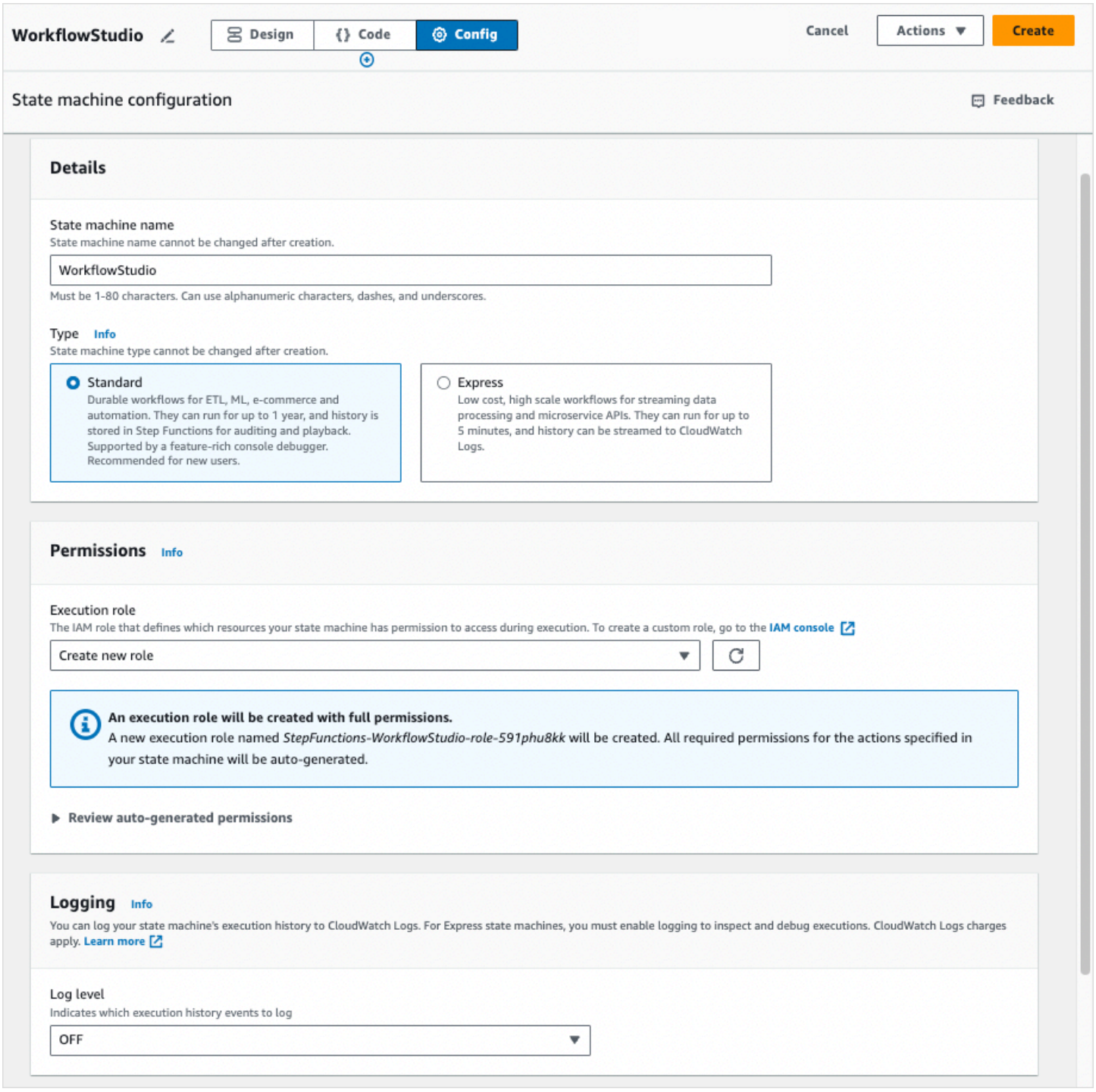
그래프 시각화 창

그래프 시각화를 사용하면 워크플로를 그래픽 형식으로 표시할 수 있습니다. Workflow Studio의 코드 편집기에서 워크플로 정의를 작성하면 그래프 시각화 창에서 워크플로의 실시간 그래프를 렌더링합니다. 그래프 시각화 창에서 상태를 재정렬, 삭제 또는 복제하면 코드 편집기에서 워크플로 정의가 자동으로 업데이트됩니다. 마찬가지로 코드 편집기에서 워크플로 정의를 업데이트하거나 상태를 재정렬, 삭제 또는 추가하면 시각화가 자동으로 업데이트됩니다.

워크플로의 ASL 정의에 있는 JSON이 유효하지 않으면 그래프 시각화 창에서 렌더링을 일시 중지하고 창 하단에 상태 메시지가 표시됩니다.

구성 모드

Workflow Studio의 구성 모드를 사용하면 상태 시스템 구성을 관리할 수 있습니다. 이 모드에서 상태 시스템 이름 및 유형, IAM 권한, 상태 시스템의 로깅 구성과 같은 세부 정보를 지정할 수 있습니다. 이 모드에서 지정할 수 있는 다른 추가 구성으로는 상태 컴퓨터를 만들 때 AWS X-Ray 추적 기능을 활성화하고 버전을 게시하는 방법이 있습니다. 상태 시스템을 만든 후에는 상태 시스템 이름 및 유형을 제외한 모든 상태 시스템 구성 옵션을 편집할 수 있습니다. 다음 이미지에서는 구성 모드에서 지정할 수 있는 일부 구성을 보여줍니다.



상태 시스템 구성 관리

상태 시스템 구성을 관리하려면 다음을 수행합니다.

1. 상태 시스템 이름 상자에 상태 시스템 이름을 입력합니다.

> **Tip**
>
> 또는 기본 상태 컴퓨터 이름 옆에 있는 편집 아이콘을 선택할 수도 있습니다 MyStateMachine. 그런 다음 상태 머신 구성에서 이름을 지정합니다.

> **Important**
>
> 상태 시스템을 만든 후에는 상태 시스템 이름을 편집할 수 없습니다.

2. 유형에서 상태 시스템 유형을 표준 또는 Express로 선택합니다. 상태 시스템 유형은 표준 워크플로와 Express 워크플로 비교 섹션을 참조하세요.

> **Important**
>
> 상태 시스템을 만든 후에는 상태 시스템 유형을 편집할 수 없습니다.

3. 권한에서 상태 시스템 실행 역할로 사용할 IAM 역할을 선택합니다.

   - 새 역할 생성(권장됨): 이 옵션을 선택하면 Step Functions에서 상태 시스템을 만들 때 필요한 최소 권한이 있는 상태 시스템 실행 역할을 자동으로 만듭니다. 자동으로 생성된 이러한 IAM 역할은 상태 머신을 생성하는 시점에 유효합니다. AWS 리전

     > **Tip**
     >
     > Step Functions에서 상태 시스템에 자동으로 생성하는 권한을 검토하려면 자동 생성된 권한 검토를 선택하세요.

     > **Note**
     >
     > Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

   - 기존 역할 선택: 상태 시스템의 고유한 IAM 역할을 만든 다음 기존 역할 선택 아래에 나열된 옵션 중에서 선택합니다. 역할 정책에 상태 시스템에 위임할 권한이 포함되어 있는지 확인합니다.

IAM 정책 생성에 대한 자세한 내용은 IAM 사용 설명서의 IAM 정책 생성을 참조하세요.

- 역할 ARN 입력: 이 상태 시스템에 사용할 기존 IAM 역할의 Amazon 리소스 이름(ARN)을 지정합니다. 예를 들어 `arn:aws:iam::123456789012:role/service-role/StepFunctions-WorkflowStudio-role-777f4027`입니다.

4. 로깅에서 상태 시스템의 로그 수준을 설정합니다. Step Functions는 선택 사항에 따라 실행 내역 이벤트를 로깅합니다. 다음 옵션 중 하나를 선택할 수 있습니다.
   - 전체: 모든 이벤트 유형이 로깅됩니다.
   - 오류: TaskFailed 및 와 ExecutionFailed 같은 모든 오류 이벤트 유형이 기록됩니다.
   - 치명적: 및 와 같은 ExecutionAborted 모든 치명적 오류 이벤트 유형이 기록됩니다. ExecutionFailed
   - 끄기: 이벤트 유형이 로깅되지 않습니다.

   로그 수준에 대한 자세한 내용은 로그 수준 섹션을 참조하세요.
5. 추가 구성에서 다음 선택적 구성을 하나 이상 설정합니다.
   - X-Ray 추적 활성화: 업스트림 서비스에서 추적 ID를 전달하지 않은 경우에도 Step Functions에서 추적을 상태 시스템 실행의 X-Ray에 보내려면 이 확인란을 선택합니다. 자세한 정보는 AWS X-Ray 및 Step Functions을 참조하세요.
   - 생성 시 버전 게시: 버전은 실행할 수 있는 상태 시스템의 번호가 매겨져 있고 변경할 수 없는 스냅샷입니다. 상태 시스템을 만드는 동안 상태 시스템 버전을 게시하려면 이 확인란을 선택합니다. Step Functions는 버전 1을 상태 시스템의 첫 번째 버전으로 게시합니다.

     버전에 대한 자세한 내용은 상태 시스템 버전을 참조하십시오.
   - 새 태그 추가: 상태 시스템에 태그를 추가하려면 이 상자를 선택합니다. 태그를 추가하면 리소스와 관련된 비용을 추적 및 관리하고 IAM 정책에서 보안을 개선하는 데 도움이 될 수 있습니다. 태그에 대한 자세한 내용은 Step Functions에서 태그 지정 단원을 참조하세요.
6. 생성을 선택하세요.
7. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 구성 보기를 선택하여 구성 모드로 돌아갈 수도 있습니다.

키보드 바로 가기

Workflow Studio는 다음 키보드 바로 가기를 지원합니다.

키보드 바로 가기	함수
Shortcuts for the Code mode	
Ctrl+space	Auto-complete suggestions
F1	Display a list of available commands
Common shortcuts for the Design and Code modes	
Ctrl+Z	Undo the last operation
Ctrl+Shift+Z	Redo the last operation
Alt+C	Center the workflow in the canvas
Backspace	Remove all selected states
Delete	Remove all selected states
Ctrl+D	Duplicate selected state

Workflow Studio 사용

Step Functions Workflow Studio를 사용하여 워크플로를 생성, 편집 및 실행하는 방법을 알아봅니다. 워크플로가 준비되면 워크플로를 내보낼 수 있습니다. 또한 Workflow Studio를 사용하여 프로토타입을 빠르게 만들 수 있습니다.

이 주제의 내용

워크플로 만들기

Workflow Studio에서는 스타터 템플릿을 선택하거나 빈 템플릿을 선택하여 워크플로를 처음부터 새로 만들 수 있습니다. 빈 템플릿의 경우 디자인 또는 코드 모드를 사용하여 워크플로를 만들 수 있습니다.

스타터 템플릿은 워크플로우 프로토타입과 정의를 자동으로 생성하고 프로젝트에 필요한 모든 관련 AWS 리소스를 배포하는 ready-to-run 샘플 프로젝트입니다. AWS 계정이러한 스타터 템플릿을 사용하여 그대로 배포 및 실행하거나 워크플로 프로토타입을 사용하여 이를 기반으로 빌드할 수 있습니다. 스타터 템플릿에 대한 자세한 내용은 Step Functions를 위한 샘플 프로젝트 섹션을 참조하세요.

스타터 템플릿을 사용하여 워크플로 만들기

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 템플릿 선택 대화 상자에서 다음 중 하나를 수행하여 샘플 프로젝트를 선택합니다(예: 태스크 타이머 샘플 프로젝트).

   - 키워드로 검색 상자에 **`Task Timer`**를 입력한 다음 반환되는 검색 결과에서 태스크 타이머를 선택합니다.
   - 오른쪽 창의 전체 아래에 나열된 샘플 프로젝트를 탐색한 다음 태스크 타이머를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.
5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다. 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 워크플로의 Amazon States Language(ASL) 정의를 업데이트할 수 있도록 코드 모드로 전환합니다.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

> **Note**
>
> 이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용됩니다. CloudFormation

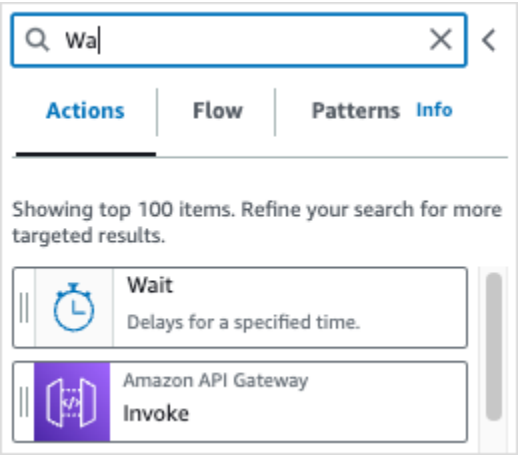
빈 템플릿을 사용하여 워크플로 만들기

1. Step Functions 콘솔을 엽니다.
2. 상태 머신 생성을 선택합니다.
3. 템플릿 선택 대화 상자에서 공백을 선택합니다.
4. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.

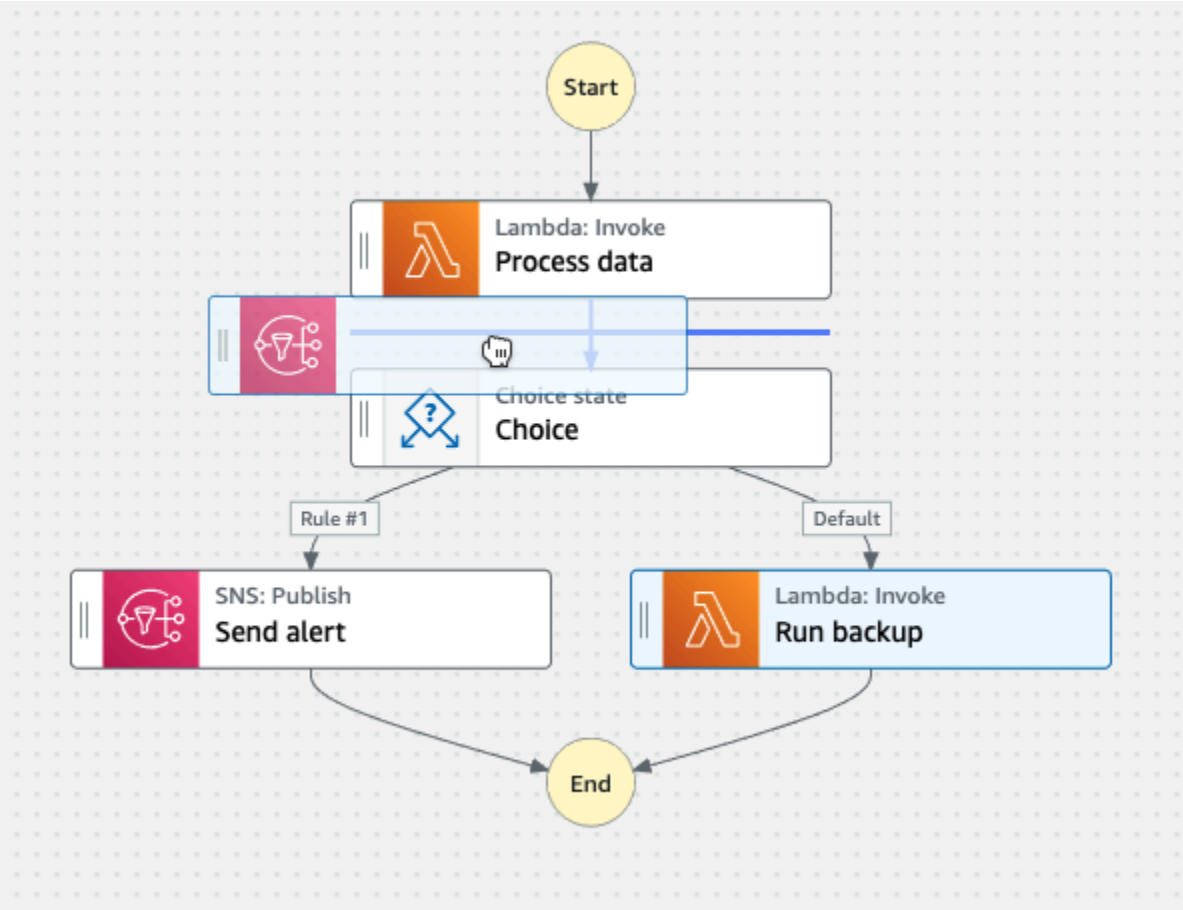
   이제 디자인 모드에서 워크플로를 디자인하거나 코드 모드에서 워크플로 정의를 작성할 수 있습니다.
5. 구성을 선택하여 구성 모드에서 워크플로 구성을 관리합니다. 예를 들어 워크플로 이름을 입력하고 해당 유형을 선택합니다.

워크플로 설계

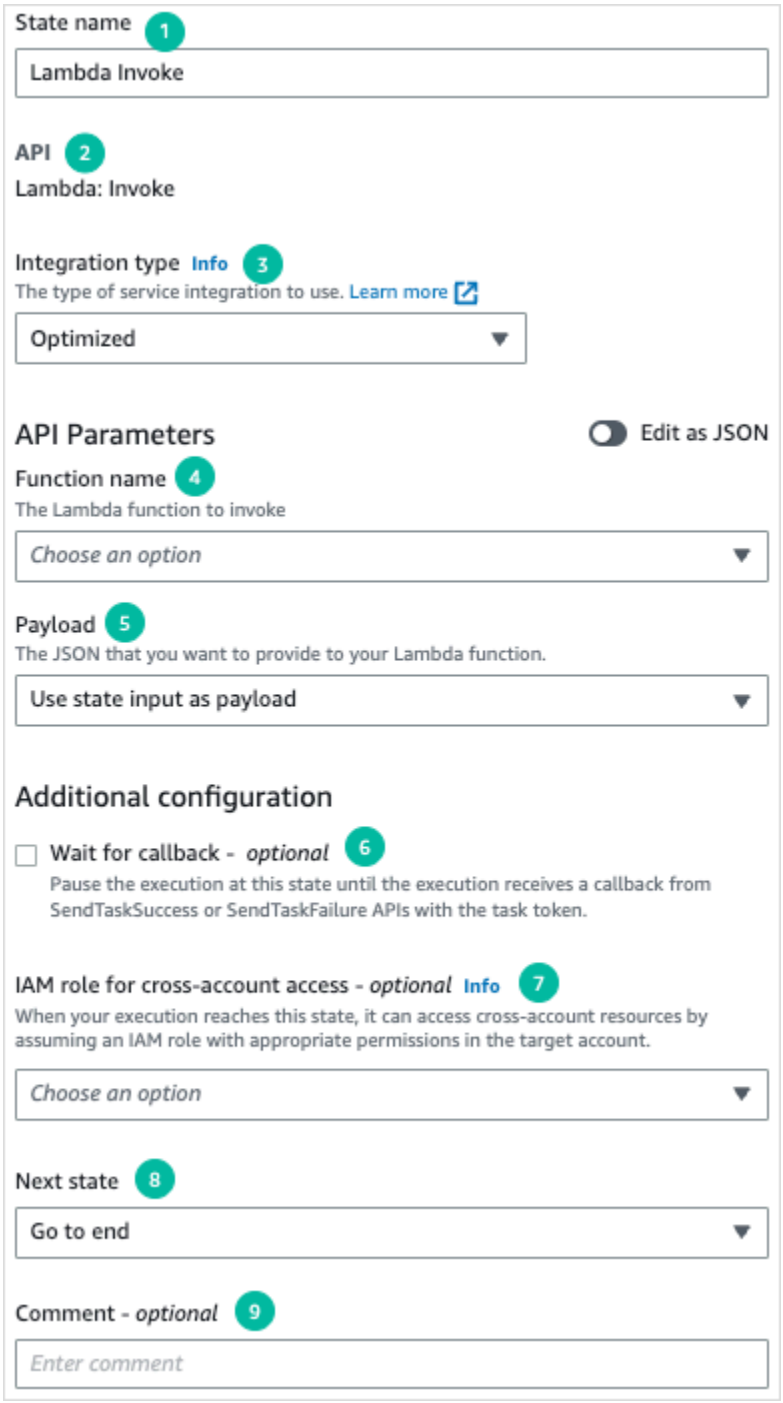
추가하려는 상태의 이름을 알고 있는 경우 상태 브라우저의 상단에 있는 검색 상자를 사용하여 디자인 모드의 작업 및 흐름 탭에서 해당 상태를 찾을 수 있습니다.



그렇지 않으면 상태 브라우저에서 상태를 선택하고 끌어서 캔버스에 놓아 워크플로의 원하는 위치에 배치할 수 있습니다. 워크플로의 다른 위치로 상태를 끌어 워크플로의 상태를 재정렬할 수도 있습니다. 상태를 캔버스로 끌면 워크플로에 놓을 수 있는 모든 위치에 줄이 나타납니다. 상태를 캔버스에 놓으면 해당 코드가 자동으로 생성되고 워크플로 정의에 추가됩니다. 정의를 보려면 Inspector 패널에서 정의 토글을 켭니다. 워크플로 정의를 편집하려면 통합 코드 편집기를 제공하는 코드 모드를 선택합니다.



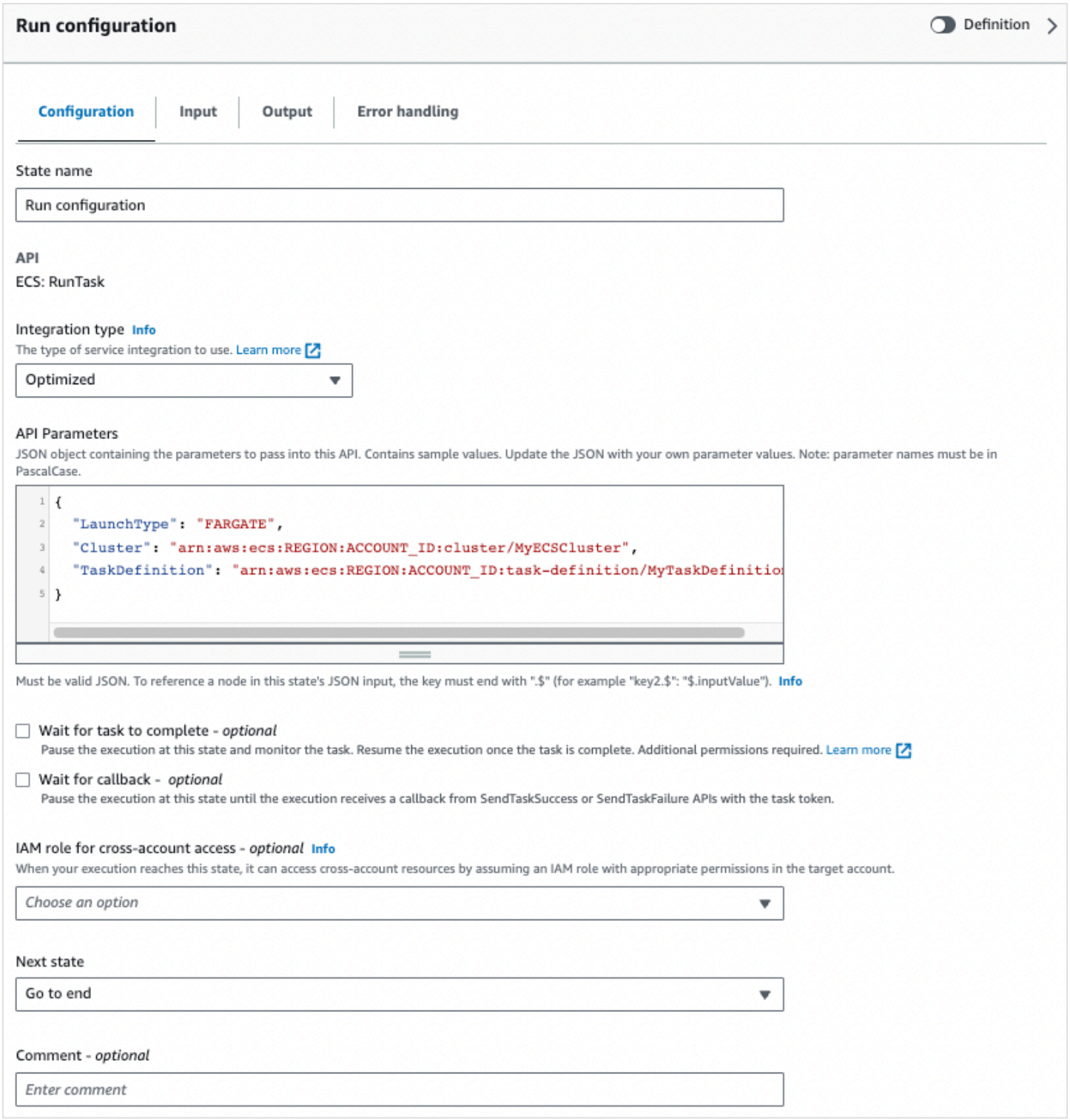
상태를 캔버스에 놓으면 오른쪽의 Inspector 패널에서 상태를 구성할 수 있습니다. 이 패널에는 캔버스에 배치하는 각 상태 또는 API 작업에 대한 구성, 입력, 출력 및 오류 처리 탭이 있습니다. 구성 탭에서 워크플로에 포함할 상태를 구성합니다. 예를 들어 Lambda Invoke API 작업의 구성 탭은 다음 옵션으로 구성되어 있습니다.



1. 상태 이름은 상태를 식별합니다. 고유한 이름을 사용하거나 생성된 기본 이름을 허용할 수 있습니다.
2. API에서는 상태에서 사용하는 API 작업을 보여줍니다.
3. 통합 유형 드롭다운 목록은 Step Functions에서 사용할 수 있는 서비스 통합 유형을 선택하는 옵션을 제공합니다. 선택한 통합 유형은 AWS 서비스 워크플로우에서 특정 API 작업을 호출하는 데 사용됩니다.

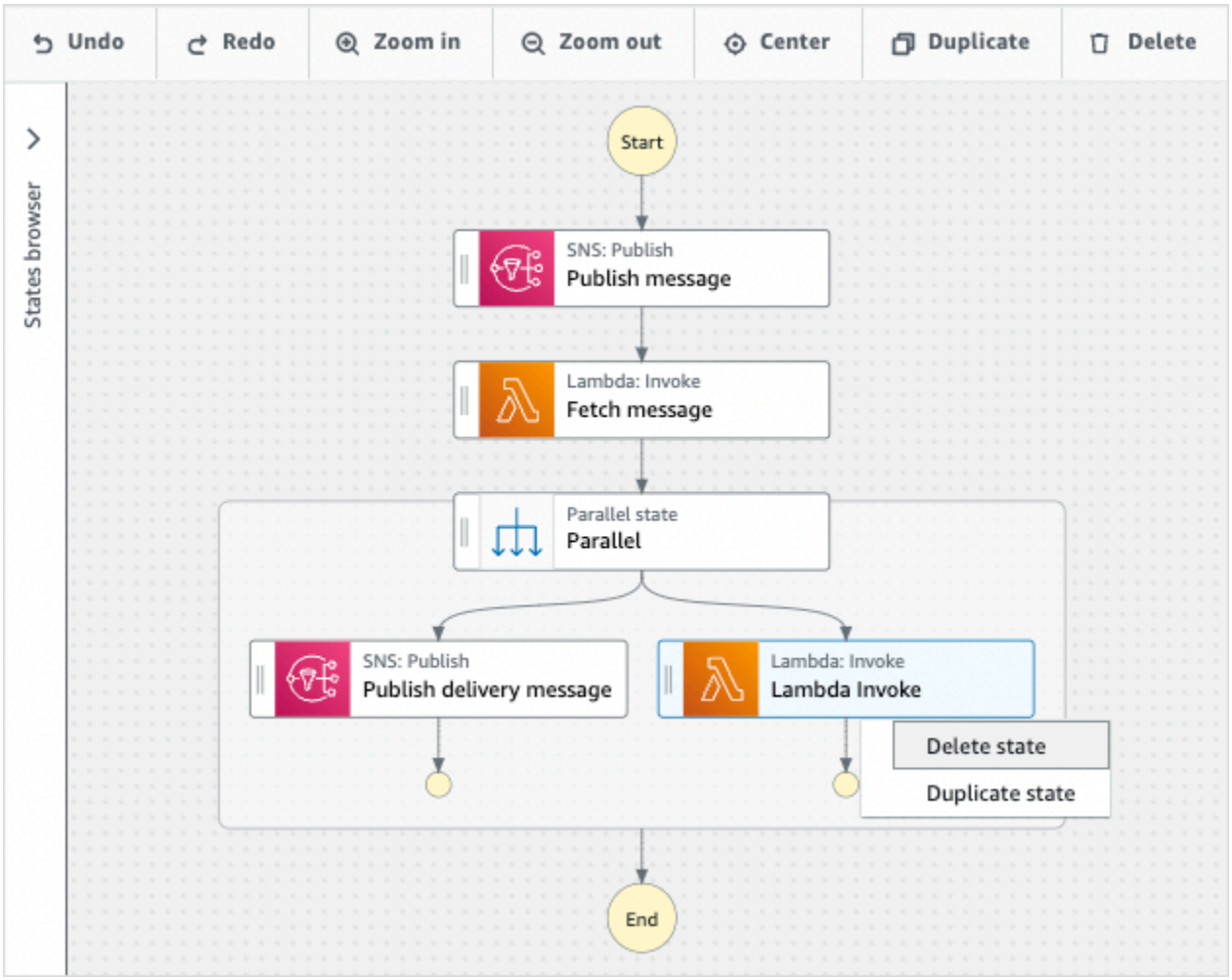
4. 함수 이름은 다음과 같은 옵션을 제공합니다.
   - 함수 이름 입력: 함수 이름이나 해당 ARN을 입력할 수 있습니다.
   - 런타임 시 상태 입력에서 함수 이름 가져오기: 이 옵션을 사용하면 지정한 경로를 기반으로 상태 입력에서 함수 이름을 동적으로 가져올 수 있습니다.
   - 함수 이름 선택: 계정 및 리전에서 사용할 수 있는 함수 중에서 직접 선택할 수 있습니다.
5. 페이로드를 사용하면 다음 옵션 중에서 선택할 수 있습니다.
   - 상태 입력을 페이로드로 사용: 이 옵션을 사용하면 상태의 입력을 Lambda 함수에 제공한 페이로드로 전달할 수 있습니다.
   - 자체 페이로드 입력: 이 옵션을 사용하면 Lambda 함수에 페이로드로 전달할 JSON 객체를 구성할 수 있습니다. 이 JSON에는 정적 값과 상태 입력에서 선택한 값이 모두 포함될 수 있습니다.
   - 페이로드 없음: 페이로드를 Lambda 함수에 전달하지 않으려면 이 옵션을 사용하면 됩니다.
6. (선택 사항) 일부 상태에는 태스크 완료 대기 또는 콜백 대기를 선택할 수 있는 옵션이 있습니다. 가능한 경우 이러한 옵션에서 다음 서비스 통합 패턴 중 하나를 선택합니다.
   - 선택한 옵션 없음: Step Functions에서 요청 및 응답 통합 패턴을 사용합니다. Step Functions는 HTTP 응답을 기다린 후 다음 상태로 진행합니다. Step Functions는 작업이 완료될 때까지 기다리지 않습니다. 사용할 수 있는 옵션이 없으면 상태에서 이 패턴을 사용합니다.
   - 태스크 완료 대기: Step Functions에서 작업 실행(.sync) 통합 패턴을 사용합니다.
   - 콜백 대기: Step Functions에서 작업 토큰을 사용하여 콜백 대기 통합 패턴을 사용합니다.
7. (선택 사항) 워크플로우 AWS 계정 내에서 서로 다르게 구성된 리소스에 액세스할 수 있도록 Step Functions는 계정 간 액세스를 제공합니다. 크로스 계정 액세스를 위한 IAM 역할은 다음과 같은 옵션을 제공합니다.
   - IAM 역할 ARN 제공: 적절한 리소스 액세스 권한이 있는 IAM 역할을 지정합니다. 이러한 리소스는 계정 간 통화를 거는 대상 계정에서 사용할 수 있습니다. 대상 계정인 대상 계정에서 사용할 수 있습니다. AWS 계정
   - 상태 입력에서 런타임 시 IAM 역할 ARN 가져오기: IAM 역할이 있는 상태의 JSON 입력에 있는 기존 키-값 페어에 대한 참조 경로를 지정합니다.
8. 다음 상태를 사용하면 다음에 전환할 상태를 선택할 수 있습니다.
9. (선택 사항) 설명 필드를 사용하여 고유한 메모를 추가할 수 있습니다. 워크플로에는 영향을 주지 않지만 워크플로에 주석을 다는 데 사용할 수 있습니다.

일부 상태에는 보다 일반적인 구성 옵션이 있습니다. 예를 들어 Amazon ECS RunTask 상태 구성에는 자리 표시자 값으로 채워진 `API Parameters` 필드가 있습니다.



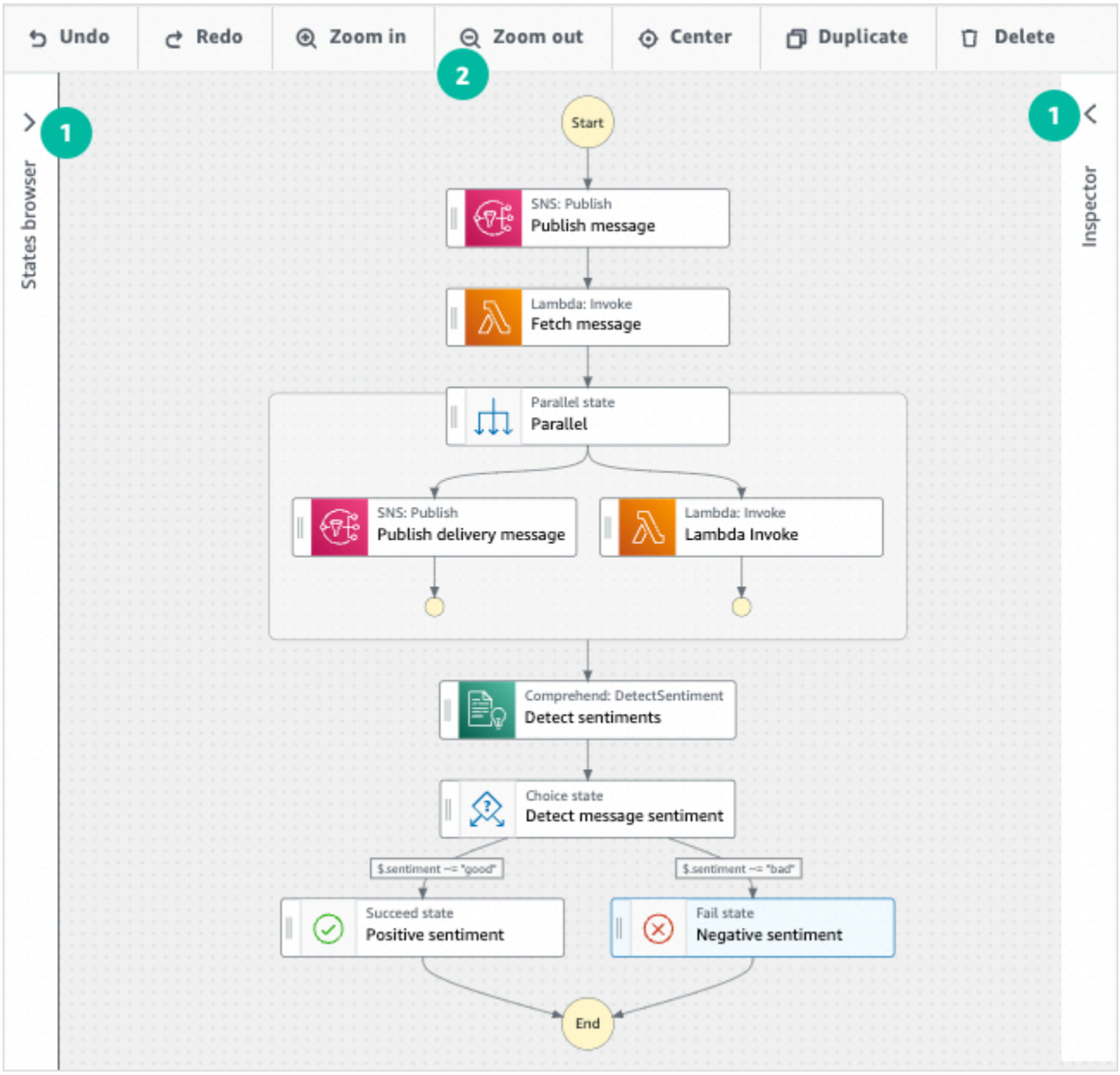
이러한 상태에서는 자리 표시자 값을 필요에 맞는 구성으로 바꿀 수 있습니다.

상태를 삭제하려면 백스페이스를 사용하고 마우스 오른쪽 버튼을 클릭하여 상태 삭제를 선택하거나, 디자인 도구 모음에서 삭제를 선택하면 됩니다.



워크플로가 커지면 캔버스에 맞지 않을 수 있습니다. 다음을 할 수 있습니다.

1. 측면 패널의 컨트롤을 사용하여 패널 크기를 조정하거나 패널을 닫습니다.
2. Canvas의 상단에 있는 디자인 도구 모음 컨트롤을 사용하여 워크플로 그래프를 확대하거나 축소합니다.



워크로드 실행

Workflow Studio를 사용하여 워크플로를 만들거나 편집한 후 Step Functions 콘솔에서 워크플로 실행을 실행하고 볼 수 있습니다.

Workflow Studio에서 워크플로 실행하기

1. 설계, 코드 또는 구성 모드에서 실행을 선택합니다.

   새 탭에 실행 시작 대화 상자가 열립니다.
2. 실행 시작 대화 상자에서 다음을 수행합니다.

1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

   > **Note**
   >
   > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.
3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

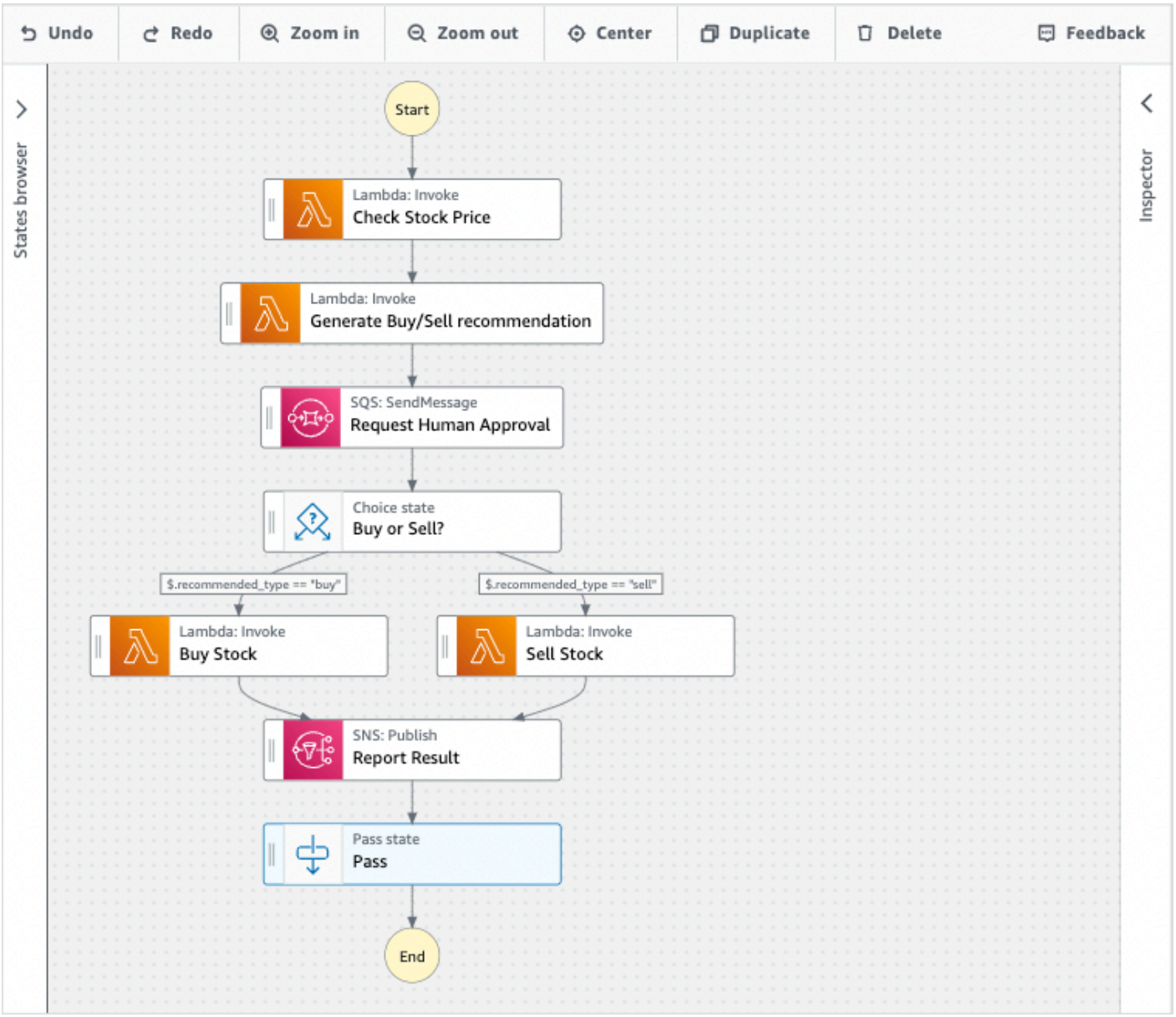
   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

워크플로 편집

Workflow Studio의 디자인 모드에서 기존 워크플로를 시각적으로 편집할 수 있습니다. Workflow Studio의 코드 모드에서 워크플로 정의도 편집할 수 있습니다.

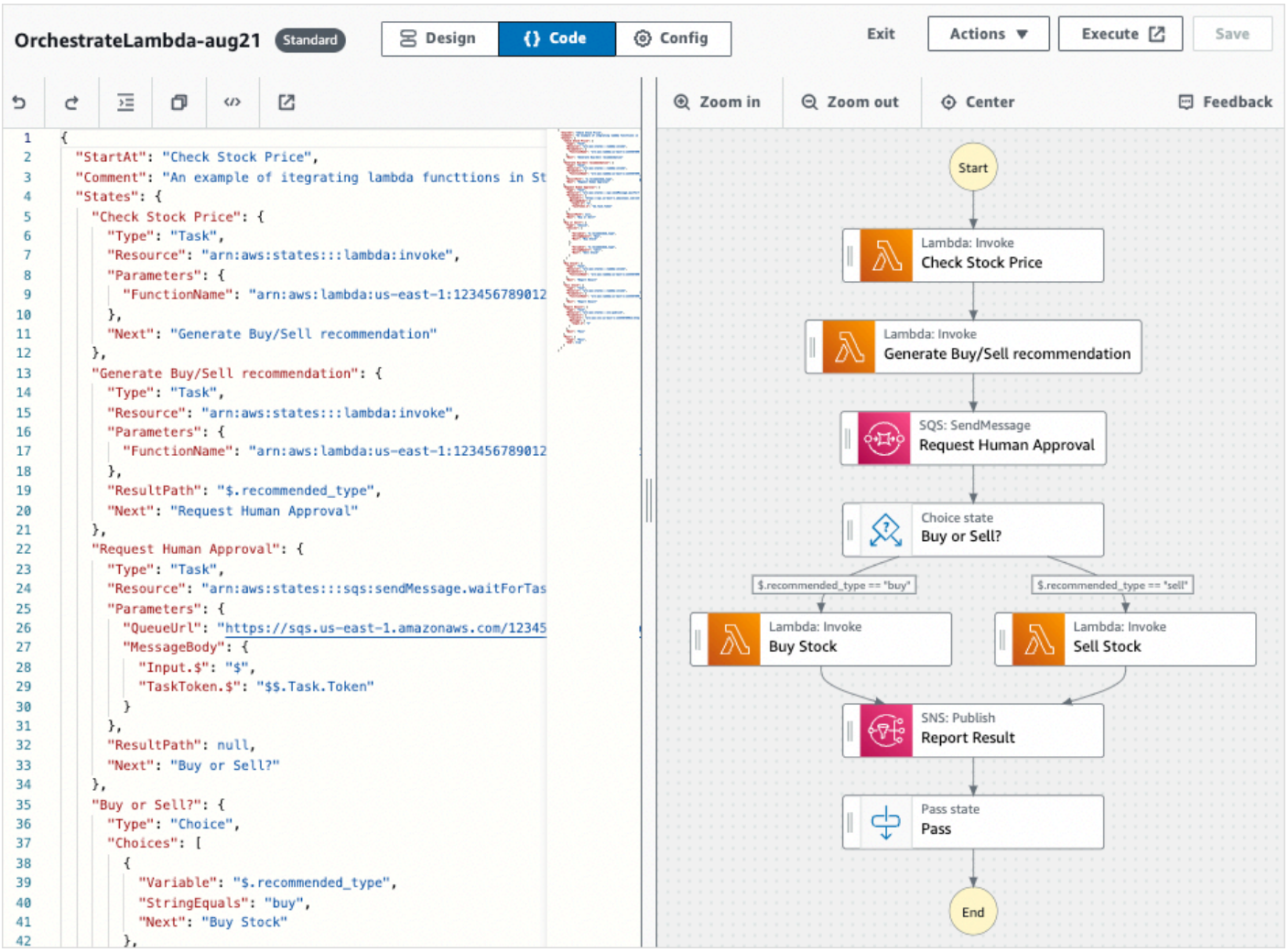
기존 워크플로 편집하기

1. Step Functions 콘솔을 엽니다.
2. 상태 시스템 페이지에서 편집하려는 워크플로를 선택합니다.
3. 상태 시스템 세부 정보 페이지에서 편집을 선택합니다.
4. Workflow Studio의 디자인 모드에서 워크플로가 열립니다. 필요에 따라 워크플로를 편집합니다.



> **Note**
>
> 워크플로에 오류가 있으면 디자인 모드에서 오류를 수정해야 합니다. 워크플로에 오류가 있으면 코드 또는 구성 모드로 전환할 수 없습니다.

5. (선택 사항) Workflow Studio에서 워크플로 정의를 보거나 편집하려면 코드 버튼을 선택합니다.



6. 완료되면 저장을 선택하여 업데이트된 워크플로를 저장합니다.
7. (선택 사항) 업데이트된 워크플로를 실행하려면 실행을 선택합니다. 새 탭에 실행 시작 대화 상자가 열립니다.

워크플로 내보내기

워크플로 Amazon States Language(ASL) 정의와 워크플로 그래프를 내보낼 수 있습니다.

1. Step Functions 콘솔에서 워크플로를 선택합니다.
2. 상태 시스템 세부 정보 페이지에서 편집을 선택합니다.
3. (선택 사항) 워크플로가 Workflow Studio의 디자인 모드에서 열립니다. 디자인 모드에서 워크플로를 편집하거나 코드 모드로 전환합니다.
4. 작업 드롭다운 버튼을 선택한 후 다음 중 하나 또는 둘 다 모두 수행합니다.

- 워크플로 그래프를 SVG 또는 PNG 파일로 내보내려면 그래프 내보내기에서 원하는 형식을 선택합니다.
- 워크플로 정의를 JSON 또는 YAML 파일로 내보내려면 내보내기 정의에서 원하는 형식을 선택합니다.

워크플로 프로토타입 만들기

Workflow Studio를 사용하여 자리 표시자 리소스가 포함된 새 워크플로의 프로토타입을 만들 수 있습니다. Application Composer에서 Workflow Studio를 사용하여 워크플로를 구축할 수도 있습니다. 프로토타입 만들기

1. Step Functions 콘솔에 로그인합니다.
2. 상태 머신 생성을 선택합니다.
3. 템플릿 선택 대화 상자에서 공백을 선택합니다.
4. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
5. Workflow Studio의 디자인 모드가 열립니다. Workflow Studio에서 워크플로를 설계합니다. 자리 표시자 리소스 포함하기

   a. 자리 표시자 리소스를 포함하려는 상태를 선택한 다음 구성에서 다음을 선택합니다.

   - Lambda Invoke 상태의 경우 함수 이름을 선택한 다음 함수 이름 입력을 선택합니다. 함수 이름에 사용자 지정 이름을 입력할 수도 있습니다.
   - Amazon SQS Send Message 상태의 경우 대기열 URL을 선택한 다음 대기열 URL 입력을 선택합니다. 자리 표시자 대기열 URL을 입력합니다.
   - Amazon SNS Publish 상태의 경우 주제에서 주제 ARN을 선택합니다.
   - 작업에 나열된 다른 모든 상태의 경우 기본 구성을 사용할 수 있습니다.

   > **Note**
   >
   > 워크플로에 오류가 있으면 디자인 모드에서 오류를 수정해야 합니다. 워크플로에 오류가 있으면 코드 또는 구성 모드로 전환할 수 없습니다.

   b. (선택 사항) 자동으로 생성된 워크플로의 ASL 정의를 보려면 정의를 선택합니다.

   c. (선택 사항) Workflow Studio에서 워크플로 정의를 업데이트하려면 코드 버튼을 선택합니다.

> **Note**
>
> 워크플로 정의에 오류가 있으면 코드 모드에서 오류를 수정해야 합니다. 워크플로 정의에 오류가 있으면 디자인 또는 구성 모드로 전환할 수 없습니다.

6. (선택 사항) 상태 시스템 이름을 편집하려면 기본 상태 시스템 이름 옆에 있는 편집 아이콘을 선택하고 상태 시스템 이름 상자에 이름을 지정합니다. MyStateMachine

   구성 모드로 전환하여 기본 상태 시스템 이름을 편집할 수도 있습니다.

7. 상태 시스템 유형 및 실행 역할과 같은 워크플로 설정을 지정합니다.

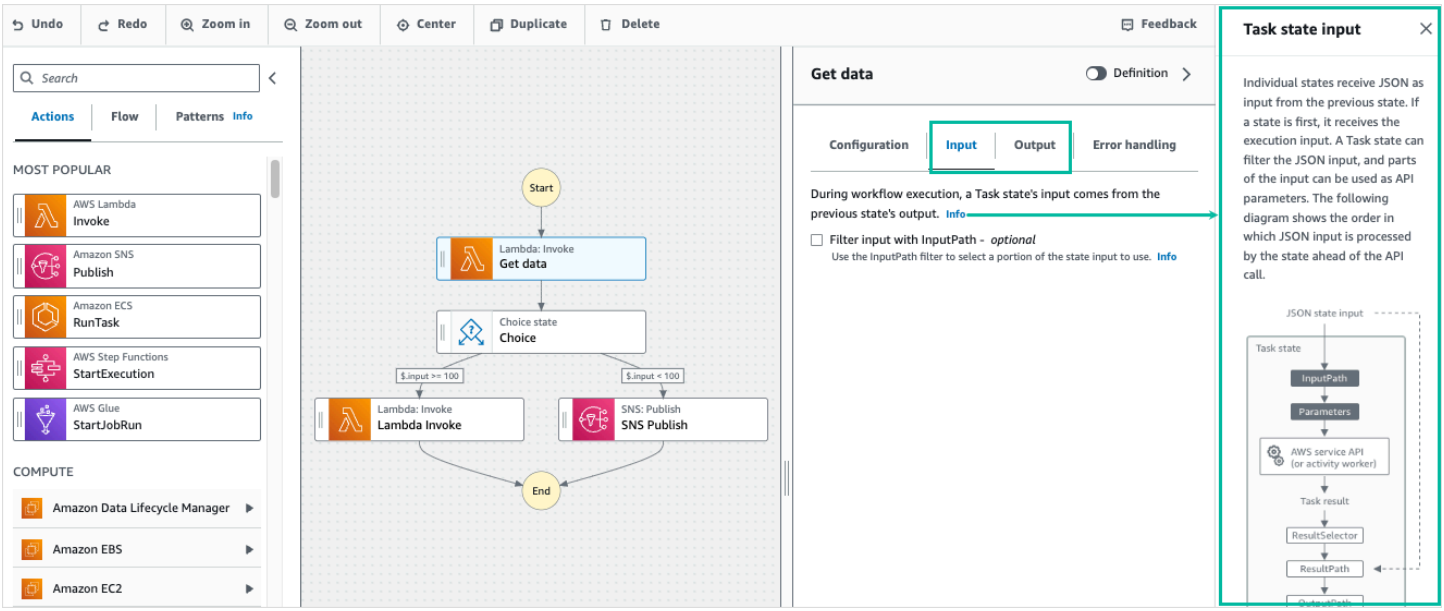
8. 생성을 선택하세요.

이제 프로토타입을 만드는 데 사용할 수 있는 자리 표시자 리소스가 포함된 새 워크플로를 만들었습니다. 워크플로 정의와 워크플로 그래프를 내보낼 수 있습니다.

- 워크플로 정의를 JSON 또는 YAML 파일로 내보내려면 디자인 또는 코드 모드에서 작업 드롭다운 버튼을 선택합니다. 그런 다음 내보내기 정의의에서 내보내려는 형식을 선택합니다. 이 내보낸 정의를 AWS Toolkit for Visual Studio Code을 사용한 로컬 개발의 시작점으로 사용할 수 있습니다.
- 워크플로 그래프를 SVG 또는 PNG 파일로 내보내려면 디자인 또는 코드 모드에서 작업 드롭다운 버튼을 선택합니다. 그런 다음 내보내기 정의에서 원하는 형식을 선택합니다.

상태의 입력 및 출력 구성

각 상태는 수신한 입력을 기반으로 결정을 내리거나 작업을 수행합니다. 대부분의 경우 출력을 다른 상태로 전달합니다. Workflow Studio의 Inspector 패널에 있는 입력 및 출력 탭에서 상태가 입력 및 출력 데이터를 필터링하고 조작하는 방법을 구성할 수 있습니다. 입력과 출력을 구성할 때 정보 링크를 사용하여 상황별 도움말에 액세스할 수 있습니다.



Step Functions에서 입력과 출력을 처리하는 방법에 대한 자세한 내용은 Step Functions에서 입력 및 출력 처리 섹션을 참조하세요.

상태에 대한 입력 구성

각 상태는 이전 상태의 입력을 JSON으로 수신합니다. 입력을 필터링하려면 Inspector 패널의 입력 탭 아래에 있는 `InputPath` 필터를 사용하면 됩니다. `InputPath`는 `$`로 시작하는 문자열로, 특정 JSON 노드를 식별합니다. 이러한 경로를 참조 경로라고 하며 JsonPath 구문을 따릅니다.

입력 필터링하기

- [입력 필터링 대상] 을 선택합니다 InputPath.
- 필터에 유효한 JsonPath`InputPath`필터를 입력합니다. 예를 들어 **`$.data`**입니다.

InputPath 필터가 워크플로에 추가됩니다.

Example 예 1: 워크플로 스튜디오에서 InputPath 필터 사용

상태에 대한 입력에 다음 JSON 데이터가 포함되어 있다고 가정해보겠습니다.

```
{
  "comment": "Example for InputPath",
  "dataset1": {
    "val1": 1,
    "val2": 2,
    "val3": 3
  },
  "dataset2": {
    "val1": "a",
    "val2": "b",
    "val3": "c"
  }
}
```

InputPath필터를 적용하려면 입력 필터링 대상을 선택한 다음 적절한 참조 경로를 입력합니다. InputPath **$.dataset2.val1**을 입력하면 다음 JSON이 상태에 대한 입력으로 전달됩니다.

```
{"a"}
```

참조 경로에도 선택할 수 있는 값이 있습니다. 참조하는 데이터가 `{ "a": [1, 2, 3, 4] }`이고 참조 경로 `$.a[0:2]`를 InputPath 필터로 적용한 경우 결과는 다음과 같습니다.

```
[ 1, 2 ]
```

Parallel, 맵 및 Pass 흐름 상태의 입력 탭 아래에는 Parameters라는 추가 입력 필터링 옵션이 있습니다. 이 필터는 필터 이후에 적용되며 하나 이상의 키-값 쌍으로 구성된 사용자 지정 JSON 객체를 구성하는 데 사용할 수 있습니다. InputPath 각 페어의 값은 정적 값이거나 입력에서 선택되거나 경로가 있는 컨텍스트 객체에서 선택될 수 있습니다.

> **Note**
>
> 파라미터에서 참조 경로를 사용하여 입력의 JSON 노드를 가리키도록 지정하려면 파라미터 이름이 .$로 끝나야 합니다.

Example 예제 2: Parallel 상태를 위한 사용자 지정 JSON 입력 만들기

다음 JSON 데이터가 Parallel 상태에 대한 입력이라고 가정해보겠습니다.

```
{
  "comment": "Example for Parameters",
  "product": {
    "details": {
      "color": "blue",
      "size": "small",
      "material": "cotton"
    },
    "availability": "in stock",
    "sku": "2317",
    "cost": "$23"
  }
}
```

이 입력의 일부를 선택하고 정적 값이 있는 추가 키-값 페어를 전달하려면 Parallel 상태의 입력 탭 아래에 있는 파라미터 필드에 다음을 지정하면 됩니다.

```
{
 "comment": "Selecting what I care about.",
 "MyDetails": {
    "size.$": "$.product.details.size",
    "exists.$": "$.product.availability",
    "StaticValue": "foo"
    }
 }
```

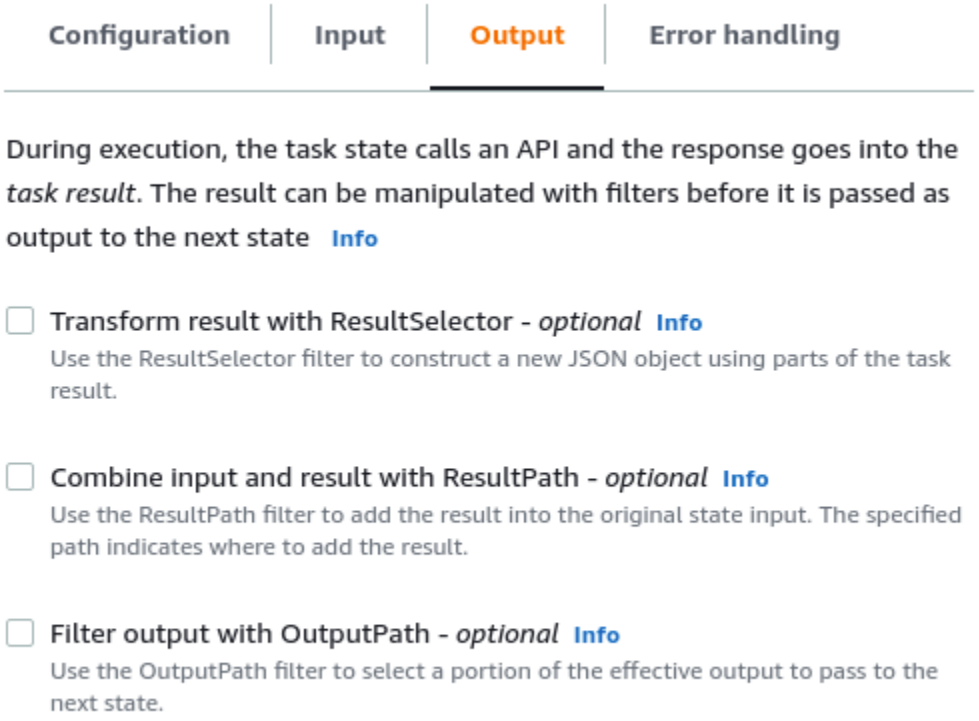
결과는 다음과 같은 JSON 데이터가 됩니다.

```
{
  "comment": "Selecting what I care about.",
  "MyDetails": {
    "size": "small",
    "exists": "in stock",
    "StaticValue": "foo"
  }
}
```

상태 출력 구성

각 상태는 다음 상태로 전달되기 전에 필터링될 수 있는 JSON 출력을 생성합니다. 여러 가지 필터를 사용할 수 있으며 각 필터는 서로 다른 방식으로 출력에 영향을 미칩니다. 각 상태에 사용할 수 있는 출력 필터는 Inspector 패널의 출력 탭 아래에 나열되어 있습니다. 태스크 상태 상태의 경우 선택한 모든 출력 필터는 다음 순서로 처리됩니다.

1. `ResultSelector`:상태 결과를 조작하려면 이 필터를 사용합니다. 결과 일부를 사용하여 새 JSON 객체를 구성할 수 있습니다.
2. `ResultPath`: 상태 입력과 작업 결과의 조합을 선택하여 출력에 전달하려면 이 필터를 사용합니다.
3. `OutputPath`: JSON 출력을 필터링하여 결과에서 다음 상태로 전달될 정보를 선택하려면 이 필터를 사용합니다.



사용 ResultSelector

`ResultSelector`는 다음 상태에 대한 선택적 출력 필터입니다.

- 태스크 상태 상태는 모든 상태가 상태 브라우저의 작업 탭에 나열되는 상태입니다.
- 맵 상태는 상태 브라우저의 흐름 탭에 있습니다.

- Parallel 상태는 상태 브라우저의 흐름 탭에 있습니다.

`ResultSelector`는 키-값 페어 하나 이상으로 구성된 사용자 지정 JSON 객체를 구성하는 데 사용될 수 있습니다. 각 페어의 값은 정적 값이거나 경로가 있는 상태 결과에서 선택될 수 있습니다.

> **Note**
>
> 파라미터에서 경로를 사용하여 결과에서 JSON 노드를 참조하도록 지정하려면 파라미터 이름이 `.$`로 끝나야 합니다.

Example ResultSelector 필터 사용 예제

이 예시에서는 Amazon EMR 상태에 대한 Amazon CreateCluster EMR API 호출의 응답을 조작하는 `ResultSelector` 데 사용합니다. `CreateCluster` 다음은 Amazon EMR `CreateCluster` API 직접 호출의 결과입니다.

```
{
  "resourceType": "elasticmapreduce",
  "resource": "createCluster.sync",
  "output": {
    "SdkHttpMetadata": {
      "HttpHeaders": {
        "Content-Length": "1112",
        "Content-Type": "application/x-amz-JSON-1.1",
        "Date": "Mon, 25 Nov 2019 19:41:29 GMT",
        "x-amzn-RequestId": "1234-5678-9012"
      },
      "HttpStatusCode": 200
    },
    "SdkResponseMetadata": {
      "RequestId": "1234-5678-9012"
    },
    "ClusterId": "AKIAIOSFODNN7EXAMPLE"
  }
}
```

이 정보의 일부를 선택하고 추가 키-값 쌍을 정적 값으로 전달하려면 상태의 출력 탭 아래 ResultSelector필드에 다음을 지정하십시오.

```
{
```

```
 "result": "found",
 "ClusterId.$": "$.output.ClusterId",
 "ResourceType.$": "$.resourceType"
 }
```

ResultSelector를 사용하면 다음 결과가 생성됩니다.

```
{
 "result": "found",
 "ClusterId": "AKIAIOSFODNN7EXAMPLE",
 "ResourceType": "elasticmapreduce"
}
```

사용: ResultPath

상태 출력은 해당 입력의 복사본, 생성된 결과 또는 해당 입력과 결과의 조합일 수 있습니다. ResultPath를 사용하여 상태 출력으로 이들의 어떤 조합을 전달할지 관리합니다. 더 많은 ResultPath 사용 사례는 ResultPath 섹션을 참조하세요.

ResultPath는 다음 상태에 대한 선택적 출력 필터입니다.

- 태스크 상태 상태는 모든 상태가 상태 브라우저의 작업 탭에 나열되는 상태입니다.
- 맵 상태는 상태 브라우저의 흐름 탭에 있습니다.
- Parallel 상태는 상태 브라우저의 흐름 탭에 있습니다.
- Pass 상태는 상태 브라우저의 흐름 탭에 있습니다.

ResultPath는 결과를 원래 상태 입력에 추가하는 데 사용될 수 있습니다. 지정된 경로는 결과를 추가할 위치를 나타냅니다.

Example ResultPath 필터 사용 예제

Task 상태에 대한 입력이 다음과 같다고 가정해보겠습니다.

```
{
  "details": "Default example",
  "who": "AWS Step Functions"
}
```

Task 상태 결과는 다음과 같습니다.

```
Hello, AWS Step Functions
```

ResultPath를 적용하고 결과를 추가하려는 위치를 나타내는 참조 경로(예: $.taskresult)를 입력하여 이 결과를 상태 입력에 추가할 수 있습니다.

이 ResultPath를 사용하여 상태 출력으로 전달되는 JSON은 다음과 같습니다.

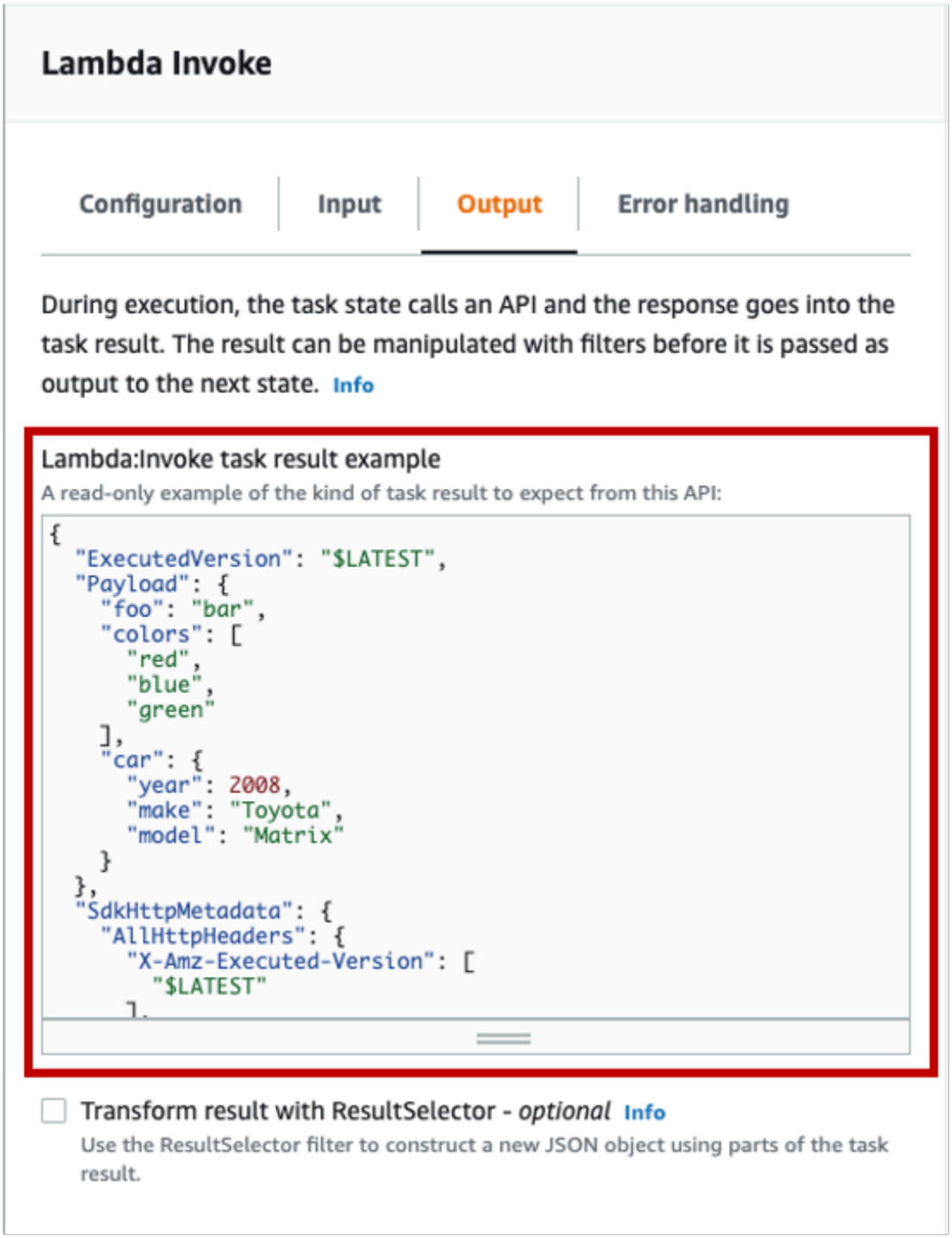
```
{
  "details": "Default example",
  "who": "AWS Step Functions",
  "taskresult": "Hello, AWS Step Functions!"
}
```

사용 OutputPath

OutputPath 필터를 사용하면 원치 않는 정보를 필터링하고 관심 있는 JSON 부분만 전달할 수 있습니다. OutputPath는 $로 시작하는 문자열로, JSON 텍스트 내의 노드를 식별합니다.

Example OutputPath 필터 사용 예제

Lambda Invoke API 직접 호출은 Lambda 함수 결과인 페이로드와 함께 메타데이터를 반환합니다. 이 API 직접 호출의 응답 예제는 상태의 출력 탭 아래에 있습니다.



`OutputPath`를 사용하여 추가 메타데이터를 필터링할 수 있습니다. 기본적으로 워크플로 스튜디오를 통해 생성된 Lambda Invoke 상태에 대한 OutputPath필터 값은 입니다. `$.Payload` 이 기본값은 추가 메타데이터를 제거하고 Lambda 함수를 직접 실행하는 것과 동일한 출력을 반환합니다.

Lambda Invoke 작업 결과 예제와 출력 필터의 `$.Payload` 값은 다음 JSON 데이터를 출력으로 전달합니다.

```
{
 "foo": "bar",
 "colors": [
      "red",
      "blue",
      "green"
 ],
 "car": {
      "year": 2008,
      "make": "Toyota",
      "model": "Matrix"
 }
}
```

> **Note**
>
> `OutputPath` 필터는 마지막으로 적용되는 출력 필터이므로 `ResultSelector` 또는 `ResultPath`와 같은 추가 출력 필터를 사용하는 경우 `OutputPath` 필터의 기본값 `$.Payload`을 적절하게 수정해야 합니다.

Workflow Studio의 실행 역할

모든 Step Functions 상태 머신에는 리소스에 대한 AWS 서비스 작업을 수행하거나 타사 API를 호출할 수 있는 권한을 상태 시스템에 부여하는 AWS Identity and Access Management (IAM) 역할이 필요합니다. 이 역할은 실행 역할이라고 합니다. 이 역할에는 각 작업에 대한 IAM 정책이 포함되어야 합니다. 상태 머신이 AWS Lambda 함수를 호출하거나, AWS Batch 작업을 실행하거나, Stripe API를 호출하도록 허용하는 정책을 예로 들 수 있습니다. Step Functions는 다음의 경우에 실행 역할을 제공해야 합니다.

- 콘솔, AWS SDK 또는 API를 AWS CLI 사용하여 상태 머신을 생성합니다. CreateStateMachine
- 콘솔, AWS SDK 또는 API를 AWS CLI 사용하여 상태를 테스트합니다. TestState

Workflow Studio에는 워크플로의 실행 역할을 쉽게 관리할 수 있는 기능이 있습니다.

주제

- 자동 생성된 역할 정보
- 역할 자동 생성

- 역할 생성 문제 해결
- Workflow Studio에서 HTTP 태스크를 테스트하기 위한 역할
- Workflow Studio에서 최적화 서비스 통합을 테스트하기 위한 역할
- 워크플로 스튜디오에서 AWS SDK 서비스 통합을 테스트하기 위한 역할
- Workflow Studio에서 흐름 상태를 테스트하기 위한 역할

자동 생성된 역할 정보

Step Functions 콘솔에서 상태 머신을 만들면 Workflow Studio에서는 필요한 IAM 정책이 포함된 실행 역할을 자동으로 생성할 수 있습니다. Workflow Studio는 상태 머신 정의를 분석하고 워크플로를 실행하는 데 필요한 최소 권한으로 정책을 생성합니다.

Workflow Studio는 다음에 대한 IAM 정책을 생성할 수 있습니다.

- 타사 API를 호출하는 HTTP 태스크.
- 최적화된 통합 (예: Lambda Invoke, DynamoDB GetItem또는) AWS 서비스 을 사용하여 다른 사용자를 호출하는 작업 상태입니다. AWS Glue StartJobRun
- 중첩된 워크플로를 실행하는 태스크 상태.
- 하위 워크플로 실행 시작, Amazon S3 버킷 나열, S3 객체 읽기 또는 쓰기에 대한 정책 등 분산 맵 상태
- X-Ray 추적. Workflow Studio에서 자동 생성되는 모든 역할에는 상태 머신에 권한을 부여하여 X-Ray에 추적을 전송할 수 있는 정책이 포함되어 있습니다.
- 상태 머신에서 로깅이 활성화된 경우의 CloudWatch Logs를 사용하여 로깅.

Workflow Studio는 AWS SDK 통합을 AWS 서비스 사용하여 다른 사람을 호출하는 작업 상태에 대한 IAM 정책을 생성할 수 없습니다.

역할 자동 생성

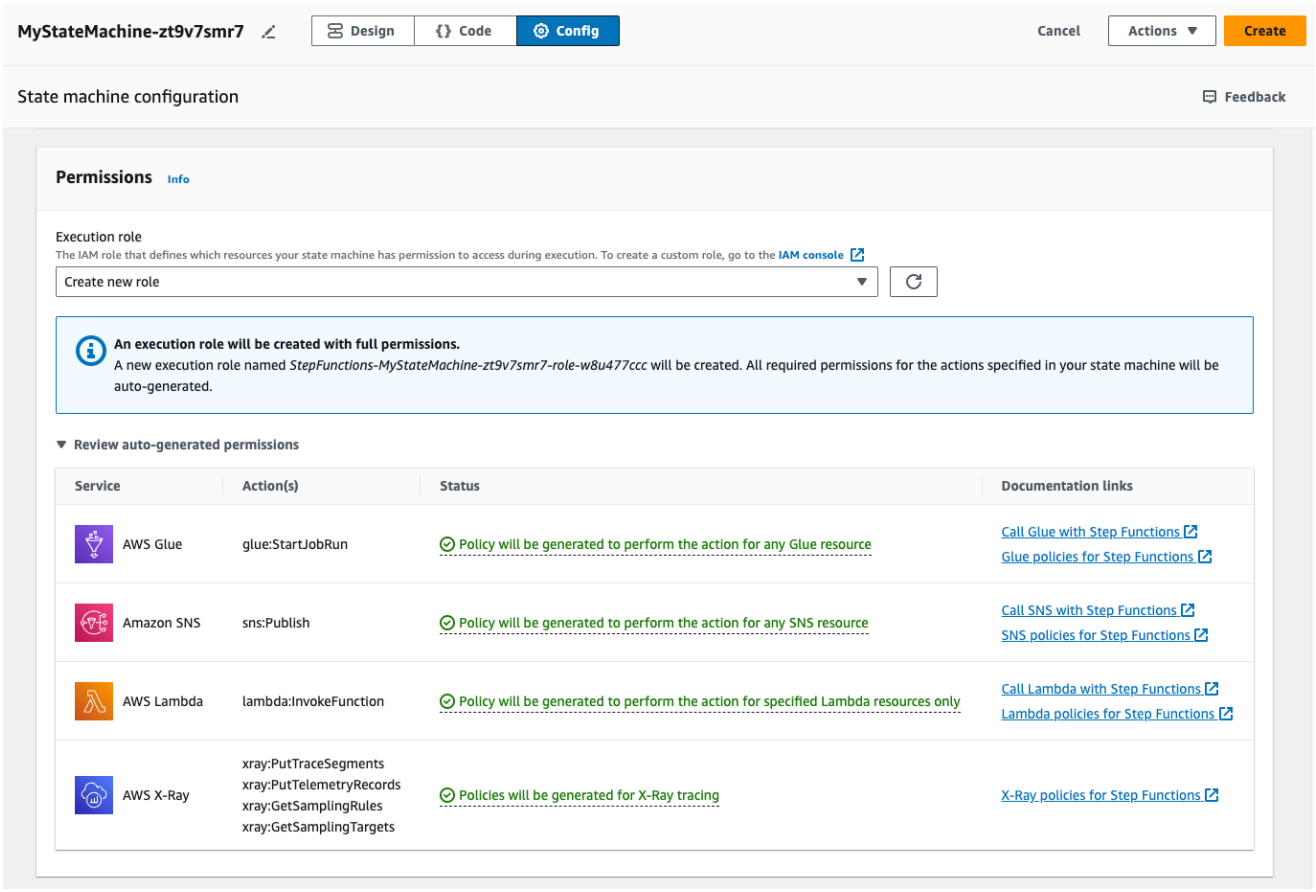
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.

   기존의 상태 머신을 업데이트할 수도 있습니다. 상태 머신을 업데이트하는 중이면 4단계를 참조하세요.
2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.

4. 구성 탭을 선택합니다.
5. 권한 섹션이 나올 때까지 스크롤하고 다음을 수행합니다.

   a. 실행 역할에서 기본 선택인 새 역할 생성을 그대로 유지해야 합니다.

   Workflow Studio는 상태 머신 정의의 모든 유효한 상태에 필요한 모든 IAM 정책을 자동으로 생성합니다. 여기에는 모든 권한을 가진 실행 역할이 생성됩니다라는 메시지가 담긴 배너가 표시됩니다.



> **Tip**
>
> Workflow Studio에서 상태 머신에 자동으로 생성하는 권한을 검토하려면 자동 생성된 권한 검토를 선택하세요.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

Workflow Studio에서 필요한 IAM 정책을 모두 생성할 수 없는 경우 특정 작업에 대한 권한은 자동 생성할 수 없습니다라는 메시지가 포함된 배너가 표시됩니다. IAM 역할은 일부 권한만 있는 상태로 생성됩니다. 누락된 권한을 추가하는 방법을 자세히 알아보려면 역할 생성 문제 해결 섹션을 참조하세요.

b. 상태 머신을 생성 중이면 생성을 선택합니다. 그렇지 않은 경우 저장을 선택합니다.

c. 나타나는 대화 상자에서 확인을 선택합니다.

Workflow Studio는 상태 머신을 저장하고 새 실행 역할을 생성합니다.

역할 생성 문제 해결

다음의 경우에는 Workflow Studio에서 필요한 모든 권한이 포함된 실행 역할을 자동으로 생성할 수 없습니다.

- 상태 머신에서 오류가 발생했습니다. Workflow Studio의 모든 검증 오류를 해결해야 합니다. 또한 저장 과정에서 발생하는 모든 서버 측 오류를 해결해야 합니다.
- 상태 머신에는 AWS SDK 통합을 사용하는 작업이 포함되어 있습니다. 이 경우 Workflow Studio는 IAM 정책을 자동 생성할 수 없습니다. Workflow Studio에는 특정 작업에 대한 권한은 자동 생성할 수 없습니다라는 메시지가 포함된 배너가 표시됩니다. IAM 역할은 일부 권한만 있는 상태로 생성됩니다. 실행 역할이 누락된 정책에 대한 자세한 내용을 보려면 자동 생성 권한 검토 테이블에서 상태에 있는 콘텐츠를 선택합니다. Workflow Studio는 여전히 실행 역할을 생성할 수 있지만 이 역할에 모든 작업에 대한 IAM 정책이 포함되지는 않습니다. 자체 정책을 작성하고 생성된 후 역할에 추가하려면 설명서 링크 아래에 있는 링크를 확인하세요. 이러한 링크는 상태 머신을 저장한 후에도 사용할 수 있습니다.

Workflow Studio에서 HTTP 태스크를 테스트하기 위한 역할

HTTP 태스크 상태를 테스트하려면 실행 역할이 필요합니다. 충분한 권한을 지닌 역할이 없는 경우 다음 옵션 중 하나를 사용하여 역할을 생성합니다.

- Workflow Studio를 사용하여 역할 자동 생성(권장) - 안전한 옵션입니다. 테스트 상태 대화 상자를 닫고 역할 자동 생성 섹션의 지침을 따릅니다. 이렇게 하려면 먼저 상태 머신을 만들거나 업데이트한 다음 Workflow Studio로 돌아가 상태를 테스트해야 합니다.
- 관리자 액세스 권한이 있는 역할 사용 — 의 모든 서비스와 리소스에 대한 전체 액세스 권한이 있는 역할을 생성할 권한이 있는 경우 해당 역할을 사용하여 워크플로의 모든 유형의 상태를 테스트할 수 있습니다. AWS이렇게 하려면 IAM 콘솔 https://console.aws.amazon.com/iam/ 에서 Step Functions 서비스 역할을 생성하고 AdministratorAccess 정책을 추가할 수 있습니다.

Workflow Studio에서 최적화 서비스 통합을 테스트하기 위한 역할

최적화 서비스 통합을 호출하는 태스크 상태에는 실행 역할이 필요합니다. 충분한 권한을 지닌 역할이 없는 경우 다음 옵션 중 하나를 사용하여 역할을 생성합니다.

- Workflow Studio의 설명서 링크를 사용하여 자체 IAM 정책을 직접 작성(권장) - 안전한 옵션입니다. 테스트 상태 대화 상자를 닫고 역할 자동 생성 섹션의 지침을 따릅니다. 이렇게 하려면 먼저 상태 머신을 만들거나 업데이트한 다음 Workflow Studio로 돌아가 상태를 테스트해야 합니다.
- 관리자 액세스 권한이 있는 역할 사용 — 의 모든 서비스와 리소스에 대한 전체 액세스 권한을 가진 역할을 만들 권한이 있는 경우 해당 역할을 사용하여 워크플로의 모든 유형의 상태를 테스트할 수 있습니다. AWS이렇게 하려면 IAM 콘솔 https://console.aws.amazon.com/iam/ 에서 Step Functions 서비스 역할을 생성하고 AdministratorAccess 정책을 추가할 수 있습니다.

워크플로 스튜디오에서 AWS SDK 서비스 통합을 테스트하기 위한 역할

AWS SDK 통합을 호출하는 태스크 상태에는 실행 역할이 필요합니다. 충분한 권한을 지닌 역할이 없는 경우 다음 옵션 중 하나를 사용하여 역할을 생성합니다.

- Workflow Studio의 설명서 링크를 사용하여 자체 IAM 정책을 직접 작성(권장) - 안전한 옵션입니다. 테스트 상태 대화 상자를 닫고 역할 자동 생성 섹션의 지침을 따릅니다. 이렇게 하려면 먼저 상태 머신을 만들거나 업데이트한 다음 Workflow Studio로 돌아가 상태를 테스트해야 합니다. 다음을 따릅니다.

  1. 테스트 상태 대화 상자를 닫습니다.

2. 구성 탭을 선택하여 구성 모드를 확인합니다.
3. 권한 섹션이 나올 때까지 스크롤합니다.
4. Workflow Studio에는 특정 작업에 대한 권한은 자동 생성할 수 없습니다라는 메시지가 포함된 배너가 표시됩니다. IAM 역할은 일부 권한만 있는 상태로 생성됩니다. 자동 생성 권한 검토를 선택합니다.
5. 자동 생성 권한 검토 테이블에는 테스트하려는 태스크 상태에 해당하는 작업을 보여 주는 행이 표시됩니다. 사용자 지정 역할에 자체 IAM 정책을 작성하려면 설명서 링크 아래에 있는 링크를 확인하세요.

- 관리자 액세스 권한이 있는 역할 사용 - 의 모든 서비스와 리소스에 대한 전체 액세스 권한이 있는 역할을 만들 권한이 있는 경우 해당 역할을 사용하여 워크플로의 모든 유형의 상태를 테스트할 수 있습니다. AWS이렇게 하려면 IAM 콘솔 https://console.aws.amazon.com/iam/ 에서 Step Functions 서비스 역할을 생성하고 AdministratorAccess 정책을 추가할 수 있습니다.

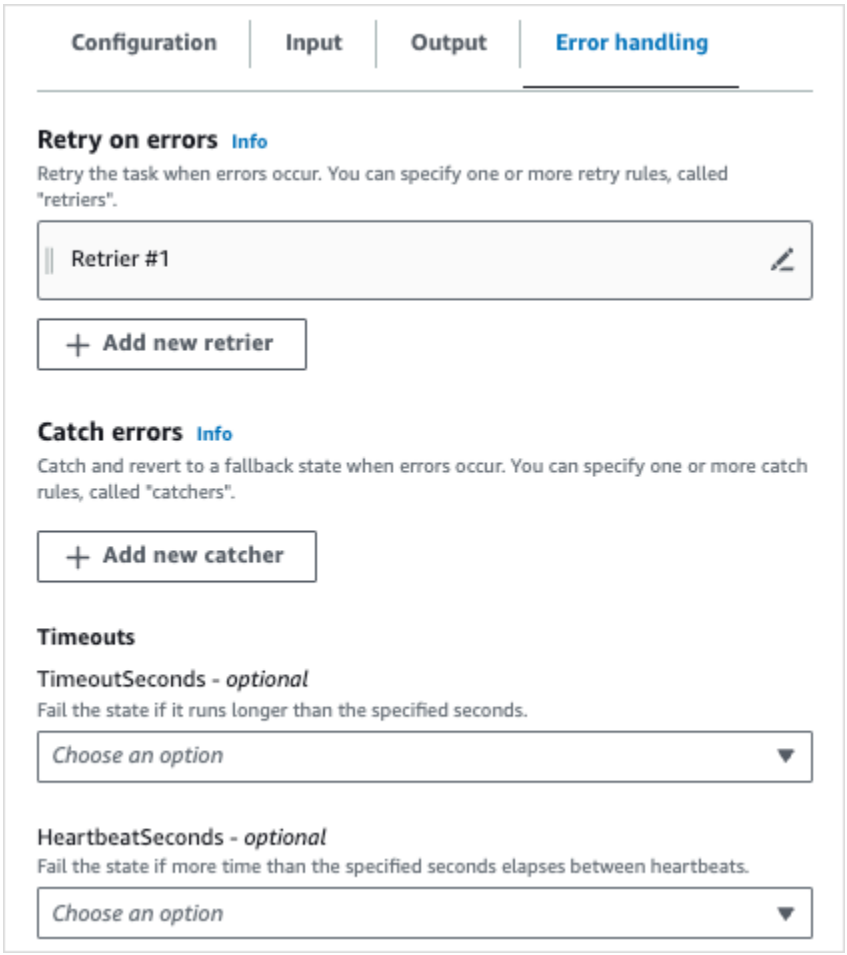
Workflow Studio에서 흐름 상태를 테스트하기 위한 역할

Workflow Studio에서 흐름 상태를 테스트하려면 실행 역할이 필요합니다. 흐름 상태는 실행 흐름으로 연결되는 상태이며 Choice, Parallel, 맵, Pass, Wait, Succeed, Fail 등이 그러한 예입니다. TestStateAPI는 맵 또는 병렬 상태에서는 작동하지 않습니다. 다음 옵션 중 하나를 사용하여 흐름 상태 테스트용 역할을 생성합니다.

- 원하는 역할 사용 AWS 계정 (권장) — 흐름 상태는 AWS 작업이나 리소스를 호출하지 않으므로 특정 IAM 정책이 필요하지 않습니다. 따라서 자신의 역할이라면 어떤 IAM 역할이든 사용할 수 있습니다 AWS 계정.
  1. 테스트 상태 대화 상자의 실행 역할 드롭다운 목록에서 원하는 역할을 선택합니다.
  2. 드롭다운 목록에 역할이 나타나지 않으면 다음 내용을 따릅니다.
     a. IAM 콘솔(https://console.aws.amazon.com/iam/)에서 역할을 선택합니다.
     b. 목록에서 역할을 선택하고 역할 세부 정보 페이지에서 해당 ARN을 복사합니다. 테스트 상태 대화 상자에 이 ARN을 입력해야 합니다.
     c. 테스트 상태 대화 상자의 실행 역할 드롭다운 목록에서 역할 ARN 입력을 선택합니다.
     d. 역할 ARN에 ARN을 붙여넣습니다.
- 관리자 액세스 권한이 있는 역할 사용 - 모든 서비스와 리소스에 대한 전체 액세스 권한을 가진 역할을 만들 권한이 있는 경우 해당 역할을 사용하여 워크플로의 모든 유형의 상태를 테스트할 수 있습니다. AWS이렇게 하려면 IAM 콘솔 https://console.aws.amazon.com/iam/ 에서 Step Functions 서비스 역할을 생성하고 AdministratorAccess 정책을 추가할 수 있습니다.

오류 처리

기본적으로 상태에서 오류를 보고하면 Step Functions로 인해 워크플로 실행 전체가 실패합니다. 작업 및 일부 흐름 상태의 경우 Step Functions에서 오류를 처리하는 방법을 구성할 수 있습니다. 오류 처리를 구성했더라도 여전히 일부 오류로 인해 워크플로 실행이 실패할 수 있습니다. 자세한 정보는 Step Functions에서 오류 처리을 참조하세요. Workflow Studio의 Inspector 패널에 있는 오류 처리 탭에서 오류 처리를 구성합니다.



오류 발생 시 재시도

작업 상태 및 Parallel 흐름 상태에 규칙을 하나 이상 추가하여 오류가 발생할 때 작업을 재시도할 수 있습니다. 이러한 규칙을 Retier라고 합니다. Retrier를 추가하려면 Retrier #1 상자에 있는 편집 아이콘을 선택한 다음 해당 옵션을 구성합니다.

- (선택 사항) 설명명 필드에 메모를 추가합니다. 워크플로에는 영향을 주지 않지만 워크플로에 주석을 다는 데 사용할 수 있습니다.

- 오류 필드에 커서를 올려 놓고 Retrier를 트리거할 오류를 선택하거나 사용자 지정 오류 이름을 입력합니다. 오류를 여러 개 선택하거나 추가할 수 있습니다.
- (선택 사항) 간격을 설정합니다. Step Functions가 처음 재시도하기 전까지의 시간(초)입니다. 추가 재시도는 최대 시도 횟수 및 백오프 비율로 구성할 수 있는 간격을 따릅니다.
- (선택 사항) 최대 시도 횟수를 설정합니다. 이는 Step Functions로 인해 실행이 실패하기 전까지의 최대 재시도 횟수입니다.
- (선택 사항) 백오프 비율을 설정합니다. 이 비율은 시도할 때마다 재시도 간격이 증가하는 정도를 결정하는 승수입니다.

> **Note**
>
> 모든 상태에서 모든 오류 처리 옵션을 사용할 수 있는 것은 아닙니다. Lambda Invoke에는 기본적으로 Retrier 하나가 구성되어 있습니다.

오류 포착

작업 상태와 Parallel 및 맵 흐름 상태에 규칙을 하나 이상 추가하여 오류를 포착할 수 있습니다. 이러한 규칙을 Catcher라고 합니다. Catcher를 추가하려면 새 Catcher 추가를 선택한 다음 해당 옵션을 구성합니다.

- (선택 사항) 설명명 필드에 메모를 추가합니다. 워크플로에는 영향을 주지 않지만 워크플로에 주석을 다는 데 사용할 수 있습니다.
- 오류 필드에 커서를 올려 놓고 Catcher를 트리거할 오류를 선택하거나 사용자 지정 오류 이름을 입력합니다. 오류를 여러 개 선택하거나 추가할 수 있습니다.
- 폴백 상태 필드에서 폴백 상태를 선택합니다. 이 상태는 오류가 포착되면 워크플로가 다음으로 이동하는 상태입니다.
- (선택 사항) ResultPath필드에 `ResultPath` 필터를 추가하여 원래 상태 입력에 오류를 추가합니다. 는 `ResultPath`유효해야 합니다 JsonPath. 이는 폴백 상태로 전송됩니다.

시간 초과

작업 상태의 제한 시간을 구성하여 실패하기 전에 상태를 실행할 수 있는 최대 시간(초)을 설정할 수 있습니다. 제한 시간을 사용하여 실행 멈춤을 방지합니다. 제한 시간을 구성하려면 실행이 실패하기 전

에 상태가 기다려야 하는 시간(초)을 입력합니다. 제한 시간에 대한 자세한 내용은 태스크 상태 상태의 `TimeoutSeconds` 섹션을 참조하세요.

HeartbeatSeconds

작업에서 보내는 하트비트 또는 정기 알림을 구성할 수 있습니다. 하트비트 간격을 설정했지만 상태에서 구성된 간격으로 하트비트 알림을 전송하지 않으면 작업이 실패로 표시됩니다. 하트비트를 구성하려면 0이 아닌 양의 정수로 시간(초)을 설정합니다. 자세한 내용은 태스크 상태의 `HeartBeatSeconds` 섹션을 참조하세요.

자습서: AWS Step Functions Workflow Studio 사용 방법 알아보기

이 자습서에서는 AWS Step Functions용 Workflow Studio를 사용하는 기본 사항을 알아봅니다. Workflow Studio의 디자인 모드에서 `Pass`, `Choice`, `Fail`, `Wait` 및 `Parallel` 등 여러 상태가 포함된 상태 시스템을 만듭니다. 끌어서 놓기 기능을 사용하여 이러한 상태를 검색, 선택 및 구성합니다. 그런 다음 자동으로 생성된 워크플로의 Amazon States Language(ASL) 정의를 봅니다. 또한 Workflow Studio의 코드 모드를 사용하여 워크플로 정의를 편집합니다. 그런 다음 Workflow Studio를 종료하고 상태 시스템을 실행하며 실행 세부 정보를 검토합니다.

이 자습서에서는 상태 시스템을 업데이트하고 실행 출력의 변경 사항을 보는 방법도 알아봅니다. 마지막으로 정리 단계를 수행하고 상태 시스템을 삭제합니다.

이 자습서를 완료하면 Workflow Studio의 디자인 모드와 코드 모드를 모두 사용하여 워크플로를 만들고 구성하는 방법을 알게 됩니다. 또한 상태 시스템을 업데이트, 실행 및 삭제하는 방법도 알게 됩니다.

> **Note**
>
> 시작하기 전에 이 자습서의 사전 조건을 완료해야 합니다.

주제

1단계: Worflow Studio로 이동

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.

2단계: 상태 시스템 만들기

Workflow Studio에서 상태 시스템은 워크플로의 그래픽 표현입니다. Workflow Studio를 사용하면 워크플로의 개별 단계를 정의, 구성 및 검사할 수 있습니다. 다음 단계에서는 Workflow Studio의 디자인 모드를 사용하여 상태 시스템을 만듭니다.

상태 시스템을 생성하려면

1. Workflow Studio의 디자인 모드에 있는지 확인합니다.
2. 왼쪽에 있는 상태 브라우저에서 흐름 탭을 선택합니다. 그런 다음 Pass 상태를 끌어 첫 번째 상태를 여기에 놓기 레이블이 지정된 빈 상태에 놓습니다.
3. 흐름 탭에서 Choice 상태를 끌어 Pass 상태 아래에 놓습니다.
4. 상태 이름에서 기본 이름인 Choice를 바꿉니다. 이 자습서에서는 **`IsHelloWorldExample`** 이름을 사용합니다.
5. 다른 패스 상태를 드래그하여 상태의 한 브랜치에 드롭합니다. IsHelloWorldExample 그런 다음 Fail 상태를 드래그하여 해당 상태의 다른 분기 아래에 놓습니다. IsHelloWorldExample
6. Pass (1) 상태를 선택하고 이름을 **`Yes`**로 바꿉니다. Fail 상태 이름을 **`No`**로 바꿉니다.
7. boolean 변수를 사용하여 IsHelloWorldExample상태의 분기 로직을 지정합니다. `IsHelloWorldExample`

   `IsHelloWorldExample`이 `False`이면 워크플로는 No 상태로 전환됩니다. 그렇지 않으면 워크플로는 Yes 상태에서 실행 흐름을 계속합니다.

   분기 로직을 정의하려면 다음을 수행합니다.

   a. 에서 IsHelloWorldExample상태를 선택한 다음 선택 규칙에서 규칙 #1 상자의 편집 아이콘을 선택하여 첫 번째 선택 규칙을 정의합니다. Canvas

b. 조건 추가를 선택합니다.

c. 규칙 #1 조건 대화 상자의 변수에 **$.IsHelloWorldExample**을 입력합니다.

d. 연산자에서 같음을 선택합니다.

e. 값에서 부울 상수를 선택한 다음 드롭다운 목록에서 true를 선택합니다.

f. 조건 저장을 선택합니다.

g. 다음 상태: 드롭다운 목록에 Yes가 선택되어 있는지 확인합니다.

h. 새 선택 규칙 추가를 선택한 다음 조건 추가를 선택합니다.

i. 규칙 #2 상자에서 7.c~7.f 하위 단계를 반복하여 `IsHelloWorldExample` 변수 값이 false일 때 두 번째 선택 규칙을 정의합니다. 7.e단계에서는 true 대신 false를 선택합니다.

j. 규칙 #2 상자의 다음 상태: 드롭다운 목록에서 No를 선택합니다.

k. 기본 규칙 상자에서 편집 아이콘을 선택하여 기본 선택 규칙을 정의한 다음 드롭다운 목록에서 Yes를 선택합니다.

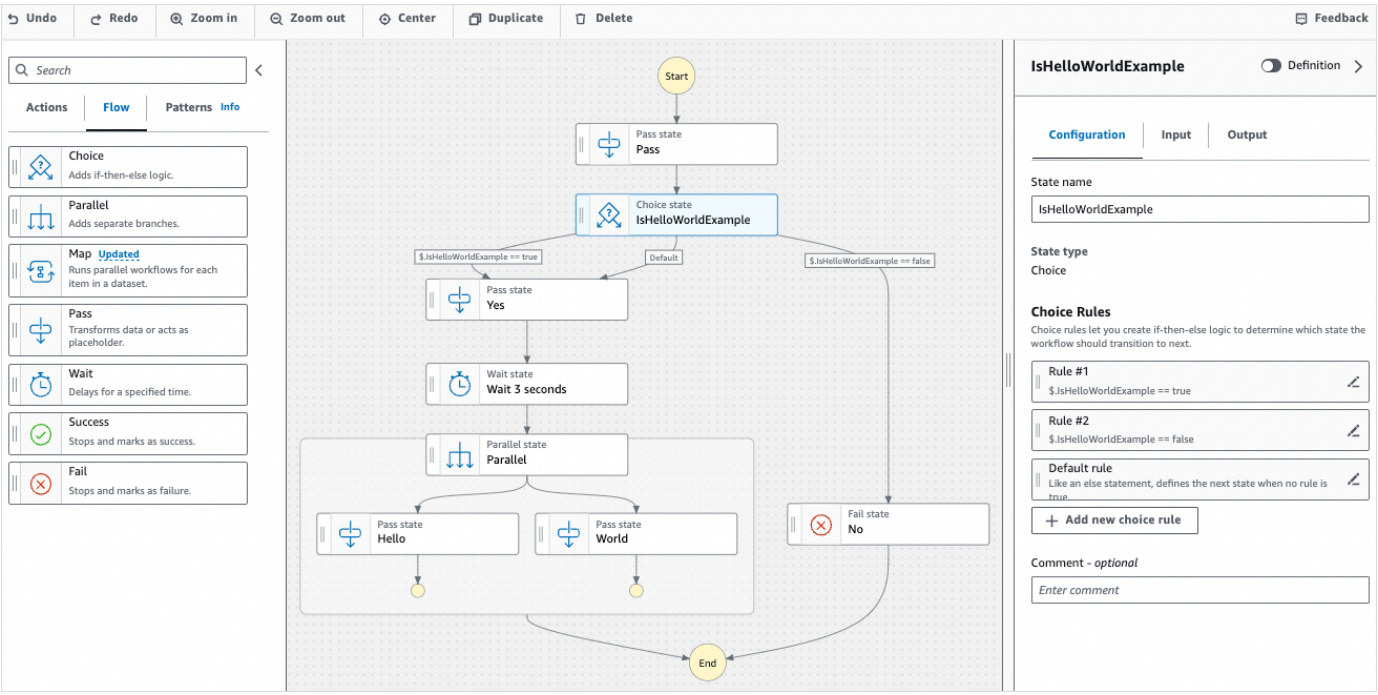
8. Yes 상태 뒤에 Wait 상태를 추가하고 이름을 **Wait 3 sec**로 지정합니다. 그런 후 다음 단계를 수행하여 대기 시간을 3초로 구성합니다.

   a. 옵션에서 기본 선택 항목인 고정 간격 대기를 그대로 둡니다.

   b. 초 아래에서 초 입력이 선택되어 있는지 확인한 다음 상자에 **3**을 입력합니다.

9. Wait 3 sec 상태 뒤에 Parallel 상태를 추가합니다. 두 브랜치에 Pass 상태 2개를 추가합니다. 첫 번째 Pass 상태 이름을 **Hello**로 지정합니다. 두 번째 Pass 상태 이름을 **World**로 지정합니다.

   완료된 워크플로는 다음과 같습니다.



3단계: 자동 생성된 Amazon States Language 정의 검토

흐름 탭에서 상태를 끌어 캔버스에 놓으면 Workflow Studio에서 워크플로의 Amazon States Language(ASL) 정의를 실시간으로 자동 작성합니다. Inspector 패널에서 정의 토글 버튼을 선택하여 이 정의를 보거나 필요에 따라 코드 모드로 전환하여 이 정의를 편집합니다. 워크플로 정의 편집 방법은 이 자습서의 4단계를 참조하세요.

- (선택 사항) Inspector 패널에서 정의를 선택하고 상태 시스템의 워크플로를 봅니다.

  다음 예제 코드에서는 자동으로 생성된 `IsHelloWorldExample` 상태 시스템의 Amazon States Language 정의를 보여줍니다. Workflow Studio에 추가한 `Choice` 상태는 2단계에서 정의한 분기 로직을 기반으로 실행 흐름을 결정하는 데 사용됩니다.

```
{
  "Comment": "A Hello World example of the Amazon States Language using Pass
 states",
  "StartAt": "Pass",
  "States": {
    "Pass": {
      "Type": "Pass",
      "Next": "IsHelloWorldExample",
      "Comment": "A Pass state passes its input to its output, without performing
 work. Pass states are useful when constructing and debugging state machines."
```

```
    },
    "IsHelloWorldExample": {
      "Type": "Choice",
      "Comment": "A Choice state adds branching logic to a state machine. Choice
rules can implement 16 different comparison operators, and can be combined using
And, Or, and Not\"",
      "Choices": [
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": false,
          "Next": "No"
        },
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": true,
          "Next": "Yes"
        }
      ],
      "Default": "Yes"
    },
    "No": {
      "Type": "Fail",
      "Cause": "Not Hello World"
    },
    "Yes": {
      "Type": "Pass",
      "Next": "Wait 3 sec"
    },
    "Wait 3 sec": {
      "Type": "Wait",
      "Seconds": 3,
      "Next": "Parallel"
    },
    "Parallel": {
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
          "StartAt": "Hello",
          "States": {
            "Hello": {
              "Type": "Pass",
              "End": true
            }
```

```
          }
        },
        {
          "StartAt": "World",
          "States": {
            "World": {
              "Type": "Pass",
              "End": true
            }
          }
        }
      ]
    }
  }
}
```

4단계: 코드 모드에서 워크플로 정의 편집

Workflow Studio의 코드 모드는 워크플로의 ASL 정의를 보고 편집할 수 있는 통합 코드 편집기를 제공합니다.

1. 코드를 선택하여 코드 모드로 전환합니다.
2. Parallel 상태가 정의되면 커서를 놓고 **Enter** 키를 누릅니다.
3. **Ctrl+space**를 누르면 Parallel 상태 뒤에 추가할 수 있는 상태의 목록이 표시됩니다.
4. 옵션 목록에서 Pass 상태를 선택합니다. 코드 편집기는 Pass 상태의 보일러플레이트 코드를 추가합니다.
5. 이 상태를 추가하면 워크플로 정의에서 오류가 발생합니다. Parallel 상태 정의에서 `"End": true`를 **`"Next": "PassState"`**로 바꿉니다.
6. 추가한 Pass 상태 정의에서 다음과 같이 변경합니다.
   a. 결과 노드를 제거합니다.
   b. `"ResultPath": "$.result"`, 및 `"Next": "NextState"`를 제거합니다.
   c. `"Type": "Pass"`, 뒤에 **`"End": true`**를 입력합니다.
   d. Pass 상태 정의 뒤에 **`,`**를 추가합니다.

이제 워크플로 정의가 다음 정의와 비슷하게 표시됩니다.

```
{
  "Comment": "A description of my state machine",
  "StartAt": "Pass",
  "States": {
    "Pass": {
      "Type": "Pass",
      "Next": "IsHelloWorldExample"
    },
    "IsHelloWorldExample": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": true,
          "Next": "Yes"
        },
        {
          "Variable": "$.IsHelloWorldExample",
          "BooleanEquals": false,
          "Next": "No"
        }
      ],
      "Default": "Yes"
    },
    "Yes": {
      "Type": "Pass",
      "Next": "Wait 3 seconds"
    },
    "Wait 3 seconds": {
      "Type": "Wait",
      "Seconds": 3,
      "Next": "Parallel"
    },
    "Parallel": {
      "Type": "Parallel",
      "Branches": [
        {
          "StartAt": "Hello",
          "States": {
            "Hello": {
              "Type": "Pass",
              "End": true
            }
```

```
          }
        },
        {
          "StartAt": "World",
          "States": {
            "World": {
              "Type": "Pass",
              "End": true
            }
          }
        }
      ],
      "Next": "PassState"
    },
    "PassState": {
      "Type": "Pass",
      "End": true
    },
    "No": {
      "Type": "Fail"
    }
  }
}
```

5단계: 상태 시스템 저장

1. Config more 를 선택하거나 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택합니다. MyStateMachine 상태 머신 구성에서 이름을 지정합니다. 예를 들면 **HelloWorld**를 입력합니다.
2. (선택 사항) 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다. 이 자습서의 경우 상태 머신 구성의 모든 기본 선택 항목을 그대로 둡니다.
3. 생성을 선택하세요.
4. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 구성 보기를 선택하여 구성 모드로 돌아갈 수도 있습니다.

구성 모드에 대한 자세한 내용은 Workflow Studio의 구성 모드를 참조하세요.

6단계: 상태 시스템 실행

상태 시스템 실행은 워크플로를 실행하여 작업을 수행하는 인스턴스입니다.

1. 상태 머신 페이지에서 스테이트 머신을 선택합니다. HelloWorld
2. HelloWorld페이지에서 실행 시작을 선택합니다.
3. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

   > **Note**
   >
   > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

4. 입력 상자에 실행을 위한 입력 값을 JSON 형식으로 입력합니다. `IsHelloWorldExample` 변수는 입력에 따라 실행될 상태 시스템 흐름을 결정합니다. 지금부터 다음 입력 값을 사용합니다.

   ```
   {
      "IsHelloWorldExample": true
   }
   ```

   > **Note**
   >
   > 실행 입력 지정은 선택 사항이지만 이 자습서에서는 위의 예제 입력과 유사한 실행 입력을 지정해야 합니다. 이 입력 값은 상태 시스템을 실행할 때의 `Choice` 상태에서 참조됩니다.

5. 실행 시작을 선택합니다.
6. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

   이 자습서에서는 입력 값 `"IsHelloWorldExample": true`을 입력하면 다음 출력이 표시됩니다.

   ```
   {
   ```

```
   "IsHelloWorldExample": true
},
{
   "IsHelloWorldExample": true
}
```

7단계: 상태 시스템 업데이트

상태 시스템을 업데이트하면 업데이트가 최종적으로 일관되게 유지됩니다. 잠시 후 새로 시작된 모든 실행에 업데이트된 상태 시스템 정의가 반영됩니다. 현재 실행 중인 모든 실행은 이전 정의에 따라 완료될 때까지 실행됩니다.

이 단계에서는 Workflow Studio의 디자인 모드 모드에서 상태 시스템을 업데이트합니다. World라는 Pass 상태에 `Result` 필드를 추가합니다.

1. 실행 ID가 제목인 페이지에서 상태 머신 편집을 선택합니다.
2. 디자인 모드에 있는지 확인합니다.
3. 캔버스에서 World라는 Pass 상태를 선택한 다음 출력을 선택합니다.
4. 결과 상자에 **`"World has been updated!"`**를 입력합니다.
5. 저장을 선택합니다.
6. (선택 사항) 정의 영역에서 워크플로의 업데이트된 Amazon States Language 정의를 봅니다.

```
{
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
          "StartAt": "Hello",
          "States": {
            "Hello": {
              "Type": "Pass",
              "End": true
            }
          }
        },
        {
          "StartAt": "World",
          "States": {
            "World": {
```

```
              "Type": "Pass",
              "Result": "World has been updated!",
              "End": true
            }
          }
        }
      ],
      "Next": "PassState"
    }
```

7. 실행을 선택합니다.
8. 새 탭에 열리는 실행 시작 대화 상자에 다음 실행 입력을 제공합니다.

```
{
   "IsHelloWorldExample": true
}
```

9. 실행 시작을 선택합니다.
10. (선택 사항) 그래프 보기에서 World 단계를 선택한 다음 출력을 선택합니다. 출력은 World has been updated!입니다.

8단계: 정리

상태 시스템 삭제하기

1. 탐색 창에서 상태 시스템을 선택합니다.
2. 상태 머신 페이지에서 을 선택한 다음 HelloWorld삭제를 선택합니다.
3. 상태 머신 삭제 대화 상자에 **delete**를 입력하여 삭제를 확인합니다.
4. 삭제를 선택합니다.

   성공적으로 삭제되면 화면 상단에 녹색 상태 표시줄이 나타납니다. 녹색 상태 표시줄은 상태 시스템이 삭제 대상으로 표시되었음을 알려줍니다. 진행 중인 모든 실행이 중지되면 상태 시스템이 삭제됩니다.

실행 역할 삭제하기

1. IAM의 역할 페이지를 엽니다.

2. Step Functions에서 자동으로 만든 IAM 역할을 선택합니다. 예: StepFunctions- HelloWorld -역할 예제
3. 역할 삭제(Delete role)를 선택합니다.
4. 예, 삭제(Yes, delete)를 선택합니다.

Step Functions 자습서

이 섹션의 자습서를 통해 AWS Step Functions 사용의 다양한 측면을 이해할 수 있습니다.

이 자습서를 완료하려면 AWS 계정이 필요합니다. 계정이 없는 경우 https://aws.amazon.com/ 으로 이동하여 AWS 계정 만들기를 선택하십시오. AWS

주제

- Lambda를 사용하는 Step Functions 상태 시스템 만들기
- Step Functions 상태 시스템을 사용하여 오류 조건 처리
- Inline Map 상태를 사용하여 작업 반복
- Distributed Map을 사용하여 대규모 CSV 데이터 복사
- Lambda 함수를 사용하여 전체 데이터 배치 처리
- Lambda 함수를 사용하여 개별 데이터 항목 처리
- Amazon S3 Events에 대한 응답으로 상태 시스템 실행 시작
- API Gateway를 사용하여 Step Functions API 만들기
- AWS SAM을 사용하여 Step Functions 상태 시스템 만들기
- Step Functions를 사용하여 Activity 상태 시스템 만들기
- Lambda를 사용하여 루프를 반복하세요
- 장기 실행 워크플로 실행을 새 실행으로 계속하기
- 예제 인간 승인 프로젝트 배포
- Step Functions에서 X-Ray 트레이스 보기
- AWS SDK 서비스 통합을 사용하여 Amazon S3 버킷 정보 수집

Lambda를 사용하는 Step Functions 상태 시스템 만들기

이 자습서에서는 함수를 호출하는 AWS Step Functions 데 사용하는 1단계 워크플로를 생성합니다. AWS Lambda

> **Note**
>
> Step Functions는 스테이트 머신과 태스크를 기반으로 합니다. Step Functions에서는 상태 머신을 워크플로라고 하며, 워크플로우는 일련의 이벤트 기반 단계입니다. 워크플로의 각 단계

를 상태라고 합니다. 예를 들어 작업 상태는 다른 AWS 서비스 서비스나 API 호출과 같이 다른 AWS 서비스가 수행하는 작업 단위를 나타냅니다.
자세한 내용은 다음을 참조하세요.

- 무엇입니까 AWS Step Functions?
- 다른 서비스에 전화해 보세요. AWS

Lambda 함수는 서버리스이고 작성하기 쉬우므로 Lambda는 `Task` 상태에 매우 적합합니다. AWS Management Console 또는 선호하는 편집기에서 코드를 작성할 수 있습니다. AWS 함수에 컴퓨팅 환경을 제공하고 함수를 실행하는 세부 정보를 처리합니다.

이 주제에서 수행할 작업

- 1단계: Lambda 함수 생성
- 2단계: Lambda 함수 테스트
- 3단계: 상태 시스템 만들기
- 4단계: 상태 시스템 실행

1단계: Lambda 함수 생성

Lambda 함수는 이벤트 데이터를 수신하고 인사말 메시지를 반환합니다.

Important

Lambda 함수가 상태 머신과 AWS 동일한 계정 AWS 및 지역에 속하는지 확인하십시오.

1. Lambda 콘솔을 열고 함수 생성을 선택합니다.
2. 함수 생성 페이지에서 처음부터 새로 작성을 선택합니다.
3. [함수 이름]에 `HelloFunction`을 입력합니다.
4. 다른 모든 옵션에는 기본 선택 항목을 그대로 둔 다음 함수 생성을 선택합니다.
5. Lambda 함수를 만든 후 페이지 오른쪽 상단 모서리에 표시된 함수의 Amazon 리소스 이름(ARN)을 복사합니다. ARN을 복사하려면 을 클릭합니다. ARN 예제는 다음과 같습니다.

```
arn:aws:lambda:us-east-1:123456789012:function:HelloFunction
```

6. Lambda 함수의 다음 코드를 페이지의 코드 소스 섹션에 복사합니다. ***HelloFunction***

```
export const handler = async(event, context, callback) => {
    callback(null, "Hello from " + event.who + "!");
};
```

이 코드는 입력 데이터의 who 필드를 사용하여 인삿말을 수집합니다. 이때 입력 데이터는 함수로 전달되는 event 객체가 전달한 것입니다. 이 함수의 입력 데이터는 나중에 새로 실행을 시작할 때 추가합니다. callback 메서드는 함수로부터 수집된 인삿말을 반환합니다.

7. 배포를 선택합니다.

2단계: Lambda 함수 테스트

Lambda 함수를 테스트하여 작동 상태를 확인합니다.

1. 테스트를 선택합니다.
2. 이벤트 이름에 HelloEvent를 입력합니다.
3. 이벤트 JSON 데이터를 다음으로 바꿉니다.

```
{
    "who": "AWS Step Functions"
}
```

"who" 항목은 Lambda 함수의 event.who 필드에 해당하며 인사말을 완성합니다. 상태 시스템을 실행할 때 같은 입력 데이터를 입력합니다.

4. 저장를 선택한 다음 테스트를 선택합니다.
5. 테스트 결과를 확인하려면 실행 결과(Execution result)에서 세부 정보(Details)를 확장합니다.

3단계: 상태 시스템 만들기

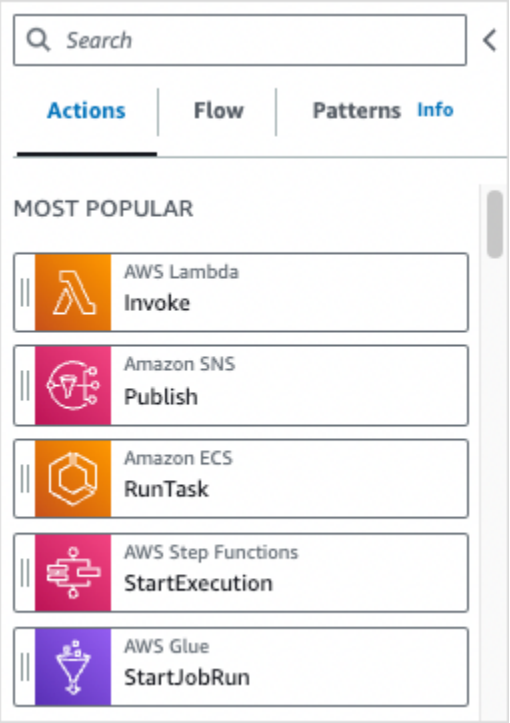
Step Functions 콘솔을 사용하여 1단계에서 만든 Lambda 함수를 간접적으로 호출하는 상태 시스템을 만듭니다.

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.

> **Important**
>
> 스테이트 머신이 이전에 생성한 Lambda 함수와 동일한 AWS 계정 및 리전에 속하는지 확인하십시오.

2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
4. 왼쪽의 상태 브라우저에서 작업 탭을 선택했는지 확인합니다. 뒤이어 다음과 같이 하세요.

   - AWS Lambda Invoke API를 끌어 첫 번째 상태를 여기에 놓기 레이블이 지정된 빈 상태에 놓습니다.



5. 오른쪽의 Inspector 패널에서 Lambda 함수를 구성합니다.

   a. API 파라미터 섹션의 함수 이름 드롭다운 목록에서 이전에 만든 Lambda 함수를 선택합니다.

   b. 페이로드 드롭다운 목록에 기본 선택 항목을 그대로 둡니다.

6. (선택 사항) 정의를 선택하여 상태 시스템 Amazon States Language(ASL) 정의를 봅니다. 이 정의는 작업 탭과 Inspector 패널에서의 선택 항목에 따라 자동으로 생성됩니다.
7. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 시스템 이름 옆에 있는 편집 아이콘을 선택하십시오. MyStateMachine 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

예를 들어 **LambdaStateMachine** 이름을 입력합니다.

> **Note**
>
> 상태 머신, 실행 및 활동 태스크의 이름은 80자를 초과하면 안 됩니다. 이러한 이름은 계정 및 AWS 지역별로 고유해야 하며 다음 내용을 포함하지 않아야 합니다.
>
> - 공백
> - 와일드카드 문자 (? *)
> - 괄호 문자(< > { } [ ])
> - 특수 문자 (" # % \ ^ | ~ ` $ & , ; : /)
> - 제어 문자(\\u0000 - \\u001f 또는 \\u007f - \\u009f).
>
> 상태 시스템이 Express 유형이면 상태 시스템 실행 여러 개에 같은 이름을 제공할 수 있습니다. Step Functions는 여러 실행 이름이 같더라도 Express 상태 시스템 실행마다 고유한 실행 ARN을 생성합니다.
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

8. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 상태 머신 설정의 모든 기본 선택 항목을 그대로 둡니다.

9. 생성을 선택합니다.
10. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

    역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

4단계: 상태 시스템 실행

상태 시스템을 만든 후에 실행할 수 있습니다.

1. 상태 머신 페이지에서 원하는 항목을 선택합니다. LambdaStateMachine
2. 실행 시작을 선택합니다.

   실행 시작 대화 상자가 표시됩니다.
3. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

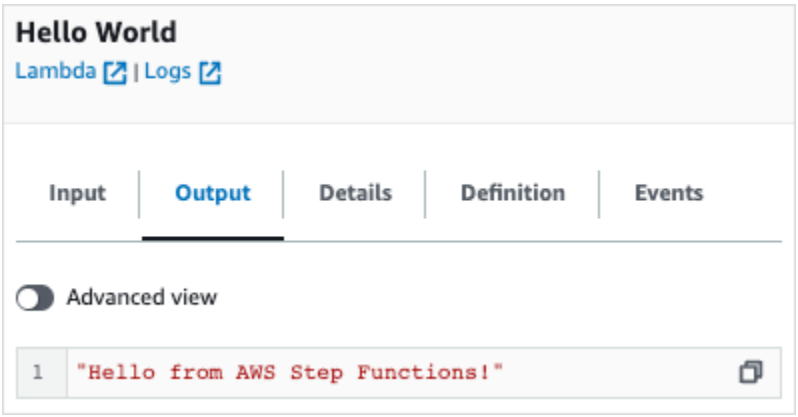
   > **Note**
   >
   > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.
4. 입력 영역에서 예제 실행 데이터를 다음으로 바꿉니다.

   ```
   {
       "who" : "AWS Step Functions"
   }
   ```

   "who"는 인사할 사람 이름을 가져오기 위해 Lambda 함수에서 사용하는 키 이름입니다.
5. 실행 시작을 선택합니다.

   상태 시스템 실행이 시작되고 실행 중인 실행을 보여주는 새로운 페이지가 표시됩니다.
6. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.



> **Note**
>
> 상태 시스템에서 Lambda를 간접적으로 호출하는 동안 페이로드를 전달할 수도 있습니다. `Parameters` 필드에 페이로드를 전달하여 Lambda를 간접적으로 호출하는 방법에 대한 자세한 내용과 예제는 Step Functions를 사용하여 Lambda 간접 호출 섹션을 참조하세요.

Step Functions 상태 시스템을 사용하여 오류 조건 처리

이 자습서에서는 폴백 상태 필드를 사용하여 AWS Step Functions 스테이트 머신을 생성합니다. `Catch`필드는 AWS Lambda 함수를 사용하여 오류 메시지 유형에 따른 조건부 로직으로 응답합니다. 이 기법을 함수 오류 처리라고 합니다.

자세한 내용은 AWS Lambda 개발자 안내서의 Node.js에서AWS Lambda 함수 오류를 참조하세요.

> **Note**
>
> 시간 초과 시 Retry하거나 오류가 발생하거나 시간이 초과될 때 `Catch`를 사용하여 특정 상태로 전환되는 상태 시스템도 만들 수 있습니다. 오류 처리 기술의 예는 Retry 및 Catch 사용 예제를 참조하십시오.

이 주제에서 수행할 작업

- 1단계: 실패하는 Lambda 함수 만들기
- 2단계: Lambda 함수 테스트
- 3단계: Catch 필드를 사용하여 상태 시스템 만들기

- 4단계: 상태 시스템 실행

1단계: 실패하는 Lambda 함수 만들기

Lambda 함수를 사용하면 오류 조건을 시뮬레이션할 수 있습니다.

> **Important**
>
> Lambda 함수가 상태 머신과 AWS 동일한 계정 AWS 및 지역에 속하는지 확인하십시오.

1. https://console.aws.amazon.com/lambda/ 에서 AWS Lambda 콘솔을 엽니다.
2. 함수 생성을 선택합니다.
3. 블루프린트 사용을 선택하고 검색 상자에 `step-functions`를 입력한 다음 사용자 지정 오류 발생 청사진을 선택합니다.
4. [함수 이름]에 `FailFunction`을 입력합니다.
5. 역할의 경우 기본 선택 항목(기본 Lambda 권한으로 새 역할 생성)을 그대로 둡니다.
6. Lambda 함수 코드 창에 다음 코드가 표시됩니다.

```
exports.handler = async (event, context) => {
    function CustomError(message) {
        this.name = 'CustomError';
        this.message = message;
    }
    CustomError.prototype = new Error();

    throw new CustomError('This is a custom error!');
};
```

   `context` 객체가 오류 메시지 `This is a custom error!`를 반환합니다.
7. 함수 생성을 선택합니다.
8. Lambda 함수를 만든 후 페이지 오른쪽 상단 모서리에 표시된 함수의 Amazon 리소스 이름(ARN)을 복사합니다. ARN을 복사하려면 을 클릭합니다. ARN 예제는 다음과 같습니다.

```
arn:aws:lambda:us-east-1:123456789012:function:FailFunction
```

9. 배포를 선택합니다.

2단계: Lambda 함수 테스트

Lambda 함수를 테스트하여 작동 상태를 확인합니다.

1. FailFunction페이지에서 테스트 탭을 선택한 다음 테스트를 선택합니다. 테스트 이벤트를 만들 필요는 없습니다.
2. 테스트 결과(시뮬레이션된 오류)를 확인하려면 실행 결과에서 세부 정보를 확장합니다.

3단계: Catch 필드를 사용하여 상태 시스템 만들기

Step Functions 콘솔을 사용하여 `Catch` 필드와 함께 태스크 상태 상태를 사용하는 상태 시스템을 만듭니다. Task 상태의 Lambda 함수에 참조를 추가합니다. 상태 시스템에서 Lambda 함수를 간접적으로 호출하지만 실행 중에 실패합니다. Step Functions는 재시도 간에 지수 백오프를 사용하여 함수를 두 번 재시도합니다.

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
4. 코드를 선택하여 코드 편집기를 엽니다. 코드 편집기에서 워크플로의 Amazon States Language(ASL) 정의를 작성하고 편집합니다.
5. 다음 코드를 붙여넣되 `Resource` 필드에서 이전에 만든 Lambda 함수의 ARN을 바꿉니다.

```
{
   "Comment": "A Catch example of the Amazon States Language using an AWS Lambda
 function",
   "StartAt": "CreateAccount",
   "States": {
      "CreateAccount": {
         "Type": "Task",
         "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FailFunction",
         "Catch": [ {
            "ErrorEquals": ["CustomError"],
            "Next": "CustomErrorFallback"
```

```
         }, {
            "ErrorEquals": ["States.TaskFailed"],
            "Next": "ReservedTypeFallback"
         }, {
            "ErrorEquals": ["States.ALL"],
            "Next": "CatchAllFallback"
         } ],
         "End": true
      },
      "CustomErrorFallback": {
         "Type": "Pass",
         "Result": "This is a fallback from a custom Lambda function exception",
         "End": true
      },
      "ReservedTypeFallback": {
         "Type": "Pass",
         "Result": "This is a fallback from a reserved error code",
         "End": true
      },
      "CatchAllFallback": {
         "Type": "Pass",
         "Result": "This is a fallback from any error code",
         "End": true
      }
   }
}
```

이 예제는 Amazon States Language를 사용하는 상태 시스템에 대한 설명입니다. 이 예에서는 `CreateAccount`라는 하나의 `Task` 상태를 정의합니다. 자세한 내용은 상태 머신 구조를 참조하십시오.

`Retry` 필드의 구문에 대한 자세한 내용은 Retry 및 Catch를 사용하는 상태 시스템 예제 단원을 참조하십시오.

> **Note**
>
> Lambda에서 처리되지 않은 오류는 오류 출력에서 `Lambda.Unknown`으로 보고됩니다. 여기에는 out-of-memory 오류와 함수 타임아웃이 포함됩니다. `Lambda.Unknown`, `States.ALL` 또는 `States.TaskFailed`를 일치시켜 이러한 오류를 처리할 수 있습니다. Lambda에서 최대 간접 호출 수에 도달하면 오류는

`Lambda.TooManyRequestsException`입니다. Lambda 함수 오류에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 오류 처리 및 자동 재시도를 참조하세요.

6. (선택 사항) 그래프 시각화 창에서 워크플로의 실시간 그래픽 시각화를 확인합니다.
7. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택합니다. MyStateMachine 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 자습서에서는 **`Catchfailure`**을 입력합니다.
8. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 상태 머신 설정의 모든 기본 선택 항목을 그대로 둡니다.
9. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

4단계: 상태 시스템 실행

상태 시스템을 만든 후에 실행할 수 있습니다.

1. 상태 시스템 페이지에서 Catchfailure를 선택합니다.
2. Catchfailure 페이지에서 실행 시작을 선택합니다. 실행 시작 대화 상자가 표시됩니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

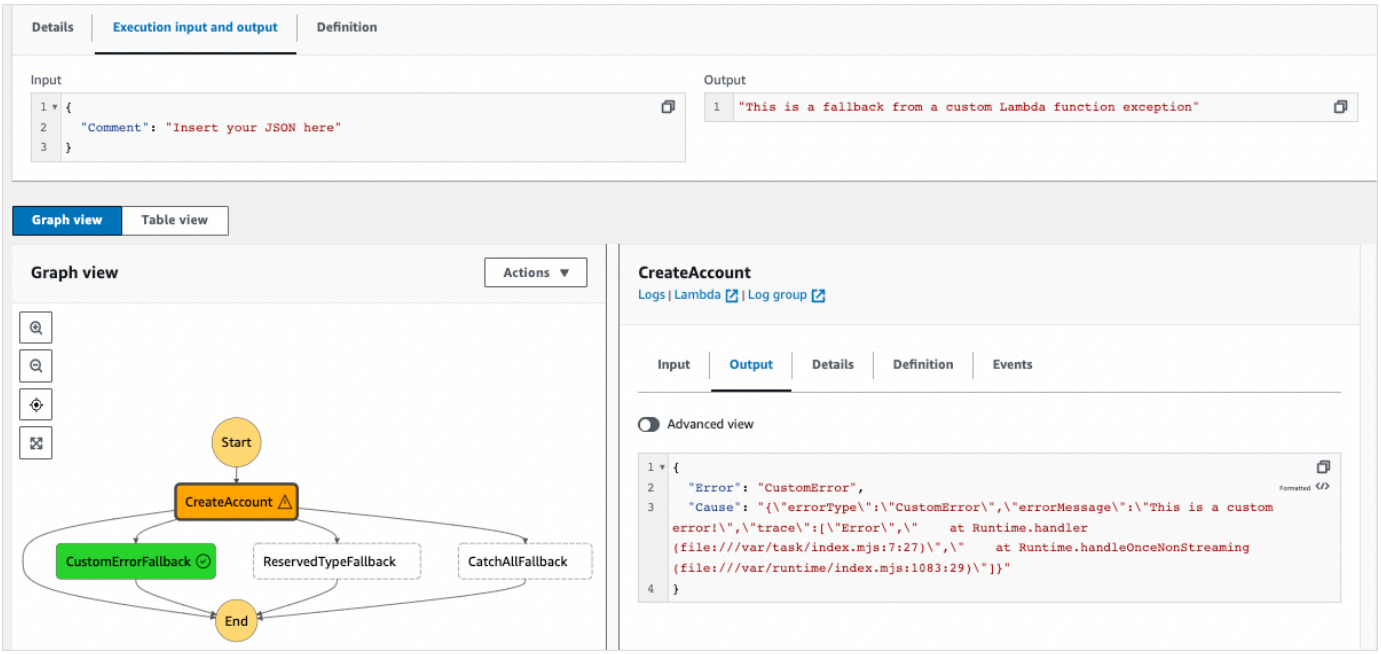
   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.
3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예를 들어 사용자 지정 오류 메시지를 보려면 그래프 보기에서 CreateAccount단계를 선택한 다음 출력 탭을 선택합니다.



> **Note**
>
> `ResultPath`를 사용하여 오류와 함께 상태 입력을 보존할 수 있습니다. ResultPath a에 오류와 입력을 모두 포함하는 데 사용합니다. `Catch` 섹션을 참조하세요.

Inline Map 상태를 사용하여 작업 반복

이 자습서는 인라인 모드에서 `Map` 상태를 시작하는 데 도움이 됩니다. 워크플로의 Inline Map 상태를 사용하여 작업을 반복 수행할 수 있습니다. 인라인 모드에 대한 자세한 내용은 인라인 Inline 모드의 Map 상태를 참조하세요.

이 자습서에서는 Inline Map 상태를 사용하여 버전 4 범용 고유 식별자(v4 UUID)를 반복 생성합니다. 먼저 Workflow Studio에서 두 가지 Pass 상태와 Inline Map 상태가 포함된 워크플로를 만듭니다. 그런 다음 `Map` 상태에 대한 입력 JSON 배열을 포함하여 입력과 출력을 구성합니다. `Map` 상태는 입력 배열의 항목마다 생성된 v4 UUID가 포함된 출력 배열을 반환합니다.

내용

- 1단계: 워크플로 프로토타입 만들기
- 2단계: 입력 및 출력 구성
- 3단계: 자동 생성된 Amazon States Language 정의 검토 및 워크플로 저장
- 4단계: 상태 시스템 실행

1단계: 워크플로 프로토타입 만들기

이 단계에서는 Workflow Studio를 사용하여 워크플로의 프로토타입을 만듭니다. Workflow Studio는 Step Functions 콘솔에서 사용할 수 있는 시각적 워크플로 디자이너입니다. 흐름 탭에서 필요한 상태를 선택하고 Workflow Studio의 끌어서 놓기 기능을 사용하여 워크플로 프로토타입을 만듭니다.

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
4. 흐름 탭에서 Pass 상태를 끌어 첫 번째 상태를 여기에 놓기 레이블이 지정된 빈 상태에 놓습니다.
5. Map 상태를 끌어 Pass 상태 아래에 놓습니다. Map 상태 이름을 **`Map demo`**로 변경합니다.

6. 두 번째 Pass 상태를 끌어 Map demo 상태 안에 놓습니다.
7. 두 번째 Pass 상태 이름을 **`Generate UUID`**로 변경합니다.

2단계: 입력 및 출력 구성

이 단계에서는 워크플로 프로토타입의 모든 상태에 대한 입력과 출력을 구성합니다. 먼저 첫 번째 Pass 상태를 사용하여 일부 고정 데이터를 워크플로에 삽입합니다. 이 Pass 상태는 이 데이터를 Map demo 상태에 대한 입력으로 전달합니다. 이 입력 내에서 Map demo 상태에서 반복해야 하는 입력 배열이 포함된 노드를 지정합니다. 그런 다음 Map demo 상태에서 v4 UUID를 생성하기 위해 반복해야 하는 단계를 정의합니다. 마지막으로 반복마다 반환되도록 출력을 구성합니다.

1. 워크플로 프로토타입에서 첫 번째 Pass 상태를 선택합니다. 출력 탭의 결과 아래에 다음을 입력합니다.

```
{
  "foo": "bar",
  "colors": [
    "red",
    "green",
    "blue",
    "yellow",
    "white"
  ]
}
```

2. Map demo 상태를 선택하고 구성 탭에서 다음을 수행합니다.
   a. 항목 배열에 대한 경로 제공을 선택합니다.
   b. 다음 참조 경로를 지정하여 입력 배열이 포함된 노드를 선택합니다.

```
$.colors
```

3. Generate UUID 상태를 선택하고 입력 탭에서 다음을 수행합니다.
   a. 파라미터로 입력 변환을 선택합니다.
   b. 다음 JSON 입력을 입력하여 각 입력 배열 항목에 대한 v4 UUID를 생성합니다. `States.UUID` 내장 함수를 사용하여 UUID를 생성합니다.

```
{
```

```
  "uuid.$": "States.UUID()"
}
```

4. Generate UUID 상태에서 출력 탭을 선택하고 다음을 수행합니다.

   a. 출력 필터링 기능을 선택합니다 OutputPath.

   b. 다음 참조 경로를 입력하여 출력 배열 항목이 포함된 JSON 노드를 선택합니다.

```
$.uuid
```

3단계: 자동 생성된 Amazon States Language 정의 검토 및 워크플로 저장

흐름 패널에서 상태를 끌어 캔버스에 놓으면 Workflow Studio에서 워크플로의 Amazon States Language(ASL) 정의를 실시간으로 자동 작성합니다. 필요에 따라 이 정의를 편집할 수 있습니다.

1. (선택 사항) Inspector 패널에서 정의를 선택하여 자동으로 생성된 워크플로의 Amazon States Language 정의를 봅니다.

> **Tip**
>
> Workflow Studio의 코드 편집기에서 ASL 정의를 볼 수도 있습니다. 코드 편집기에서 워크플로의 ASL 정의를 편집할 수도 있습니다.

   다음 예제에서는 워크플로에 자동으로 생성된 Amazon States Language 정의를 보여줍니다.

```
{
    "Comment": "Using Map state in Inline mode",
    "StartAt": "Pass",
    "States": {
      "Pass": {
        "Type": "Pass",
        "Next": "Map demo",
        "Result": {
          "foo": "bar",
          "colors": [
            "red",
            "green",
            "blue",
            "yellow",
```

```
            "white"
          ]
        }
      },
      "Map demo": {
        "Type": "Map",
        "ItemsPath": "$.colors",
        "ItemProcessor": {
          "ProcessorConfig": {
            "Mode": "INLINE"
          },
          "StartAt": "Generate UUID",
          "States": {
            "Generate UUID": {
              "Type": "Pass",
              "End": true,
              "Parameters": {
                "uuid.$": "States.UUID()"
              },
              "OutputPath": "$.uuid"
            }
          }
        },
        "End": true
      }
    }
  }
```

2. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택합니다 MyStateMachine. 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 튜토리얼에서는 이름 **InlineMapDemo**를 입력합니다.

3. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 상태 머신 구성의 모든 기본 선택 항목을 그대로 둡니다.

4. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

4단계: 상태 시스템 실행

상태 시스템 실행은 워크플로를 실행하여 작업을 수행하는 인스턴스입니다.

1. InlineMapDemo페이지에서 실행 시작을 선택합니다.
2. 실행 시작 대화 상자에서 다음을 수행합니다.
   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

      > **Note**
      >
      > Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.
   3. 실행 시작을 선택합니다.
   4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

      실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

실행 입력 및 출력을 side-by-side 보려면 실행 입력 및 출력을 선택합니다. 출력에서 Map 상태가 반환한 출력 배열을 봅니다. 다음은 출력 배열의 예제입니다.

```
[
  "a85cbc7b-4e65-4ac2-97af-80ed504adc1d",
  "b05bca11-d481-414e-aa9a-88285ec6590d",
  "f42d59f7-bd32-480f-b270-caddb518ce2a",
  "15f18616-517d-4b69-b7c3-bf22222d2efd",
  "690bcfee-6d58-408c-a6b4-1995ccafdbd2"
]
```

Distributed Map을 사용하여 대규모 CSV 데이터 복사

이 자습서는 분산 모드에서 Map 상태 사용을 시작하는 데 도움이 됩니다. Distributed로 설정된 Map 상태를 Distributed Map 상태라고 합니다. 워크플로의 Distributed Map 상태를 사용하여 대규모 Amazon S3 데이터 소스를 반복할 수 있습니다. Map 상태는 각 반복을 하위 워크플로 실행으로 실행하므로 높은 동시성이 가능합니다. 분산 모드에 대한 자세한 내용은 분산 모드의 Map 상태를 참조하세요.

이 자습서에서는 Distributed Map 상태를 사용하여 Amazon S3 버킷의 CSV 파일을 반복합니다. 그런 다음 하위 워크플로 실행의 ARN과 함께 해당 콘텐츠를 다른 Amazon S3 버킷에 반환합니다. 먼저 Workflow Studio에서 워크플로 프로토타입을 만듭니다. 다음으로 Map 상태 처리 모드를 Distributed로 설정하고 CSV 파일을 데이터 세트로 지정한 다음 해당 위치를 Map 상태에 제공합니다. 또한 Distributed Map 상태가 Express로 시작하는 하위 워크플로 실행의 워크플로 유형을 지정합니다.

이 자습서에서 사용하는 예제 워크플로의 경우 이러한 설정 외에도 하위 워크플로의 최대 동시 실행 수, Map 결과를 내보내는 위치와 같은 다른 구성도 지정합니다.

내용

- 7단계: 상태 시스템 실행

필수 조건

- Amazon S3 버킷에 CSV 파일 업로드 CSV 파일 내 헤더 행을 정의해야 합니다. CSV 파일에 적용되는 크기 제한 및 헤더 행을 지정하는 방법에 대한 자세한 내용은 Amazon S3 버킷에 있는 CSV 파일 섹션을 참조하세요.
- Amazon S3 버킷과 Map 상태 결과를 내보낼 버킷 내에 폴더를 만듭니다.

> **Important**
>
> Amazon S3 버킷이 상태 AWS 계정 머신과 AWS 리전 동일한 위치에 있는지 확인하십시오.

1단계: 워크플로 프로토타입 만들기

이 단계에서는 Workflow Studio를 사용하여 워크플로의 프로토타입을 만듭니다. Workflow Studio는 Step Functions 콘솔에서 사용할 수 있는 시각적 워크플로 디자이너입니다. 흐름 및 작업 탭에서 각각 필요한 상태와 API 작업을 선택합니다. Workflow Studio의 끌어서 놓기 기능을 사용하여 워크플로 프로토타입을 만듭니다.

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
4. 흐름 탭에서 Map 상태를 끌어 첫 번째 상태를 여기에 놓기 레이블이 지정된 빈 상태에 놓습니다.
5. 구성 탭의 상태 이름에 **`Process data`**를 입력합니다.
6. 작업 탭에서 AWS Lambda Invoke API 작업을 끌어 Process data 상태에 놓습니다.
7. AWS Lambda Invoke 상태 이름을 **`Process CSV data`**로 바꿉니다.

2단계: Map 상태에 필요한 필드 구성

이 단계에서는 Distributed Map 상태에 필요한 다음 필드를 구성합니다.

- ItemReader— Map 상태에서 입력을 읽을 수 있는 데이터 세트와 데이터 세트 위치를 지정합니다.

- ItemProcessor - 다음 값을 지정합니다.
  - `ProcessorConfig` - `Mode` 및 `ExecutionType`을 각각 `DISTRIBUTED` 및 `EXPRESS`로 설정합니다. 이렇게 하면 Distributed Map 상태가 시작하는 하위 워크플로 실행의 `Map` 상태 처리 모드와 워크플로 유형이 설정됩니다.
  - `StartAt` - Map 워크플로의 첫 번째 상태입니다.
  - `States` - 각 하위 워크플로 실행에서 반복되는 일련의 단계인 Map 워크플로를 정의합니다.
- ResultWriter— Step Functions가 분산 맵 상태 결과를 기록하는 Amazon S3 위치를 지정합니다.

> **Important**
>
> 맵 실행 결과를 내보내는 데 사용하는 Amazon S3 버킷이 상태 AWS 계정 머신과 AWS 리전 동일한지 확인하십시오. 그렇지 않으면 상태 시스템 실행이 실패하고 `States.ResultWriterFailed` 오류가 표시됩니다.

필수 필드 구성하기:

1. Process data 상태를 선택하고 구성 탭에서 다음을 수행합니다.
   a. 처리 모드에 분산을 선택합니다.
   b. 항목 소스에 Amazon S3를 선택한 다음 S3 항목 소스 드롭다운 목록에서 S3의 CSV 파일을 선택합니다.
   c. CSV 파일의 Amazon S3 위치를 지정하려면 다음을 수행합니다.
      i. S3 객체의 드롭다운 목록에서 버킷 및 키 입력을 선택합니다.
      ii. 버킷에 CSV 파일이 있는 Amazon S3 버킷의 이름을 입력합니다. 예를 들어 **`sourceBucket`**입니다.
      iii. 키에 CSV 파일을 저장한 Amazon S3 객체의 이름을 입력합니다. 이 필드에 CSV 파일 이름도 지정해야 합니다. 예를 들어 **`csvDataset/ratings.csv`**입니다.
   d. CSV 파일의 경우 열 헤더 위치도 지정해야 합니다. 이렇게 하려면 추가 구성을 선택한 다음 CSV 파일의 첫 번째 행이 헤더이면 CSV 헤더 위치에 기본 선택 항목인 첫 번째 행을 그대로 둡니다. 그렇지 않으면 지정을 선택하여 상태 시스템 정의 내에 헤더를 지정합니다. 자세한 정보는 `ReaderConfig`을 참조하세요.
   e. 하위 실행 유형에 Express를 선택합니다.
2. 내보내기 위치에서 맵 실행 결과를 특정 Amazon S3 위치로 내보내도록 Map 상태의 출력을 Amazon S3로 내보내기를 선택합니다.

3. 다음을 따릅니다.

   a. S3 버킷의 드롭다운 목록에서 버킷 이름 및 접두사 입력을 선택합니다.

   b. 버킷에 결과를 내보낼 Amazon S3 버킷의 이름을 입력합니다. 예를 들어 **mapOutputs**입니다.

   c. 접두사에 결과를 저장할 폴더 이름을 입력합니다. 예를 들어 **resultData**입니다.

3단계: 추가 옵션 구성

Distributed Map 상태에 필요한 설정 외에도 다른 옵션도 지정할 수 있습니다. 여기에는 하위 워크플로의 최대 동시 실행 수와 Map 상태 결과를 내보낼 위치가 포함될 수 있습니다.

1. Process data 상태를 선택합니다. 그런 다음 항목 소스에서 추가 구성을 선택합니다.
2. 다음을 따릅니다.

   a. 각 하위 워크플로 실행에 대한 사용자 지정 JSON 입력을 ItemSelector 지정하려면 항목 수정을 선택합니다.

   b. 다음 JSON 입력을 입력합니다.

   ```
   {
     "index.$": "$$.Map.Item.Index",
     "value.$": "$$.Map.Item.Value"
   }
   ```

   사용자 지정 입력을 만드는 방법은 ItemSelector 섹션을 참조하세요.

3. 런타임 설정의 동시성 한도 설정에서 Distributed Map 상태가 시작할 수 있는 하위 워크플로의 동시 실행 수를 지정합니다. 예를 들면 **100**를 입력합니다.
4. 브라우저에서 새 창이나 탭을 열고 4단계: Lambda 함수 구성의 설명대로 이 워크플로에서 사용할 Lambda 함수의 구성을 완료합니다.

4단계: Lambda 함수 구성

> **Important**
>
> Lambda 함수가 상태 머신과 AWS 리전 동일한 위치에 있는지 확인하십시오.

1. Lambda 콘솔을 열고 함수 생성을 선택합니다.
2. 함수 생성 페이지에서 처음부터 새로 작성을 선택합니다.
3. 기본 정보 섹션에서 Lambda 함수를 구성합니다.

   a. [함수 이름]에 **distributedMapLambda**을 입력합니다.

   b. 런타임에는 Node.js 16.x를 선택합니다.

   c. 모든 기본 선택 항목을 그대로 두고 함수 생성을 선택합니다.

   d. Lambda 함수를 만든 후 페이지 오른쪽 상단에 표시된 함수의 Amazon 리소스 이름(ARN)을 복사합니다. 워크플로 프로토타입에 이를 제공해야 합니다. ARN을 복사하려면  을 클릭합니다. ARN 예제는 다음과 같습니다.

   ```
   arn:aws:lambda:us-east-2:123456789012:function:distributedMapLambda
   ```

4. Lambda 함수의 다음 코드를 복사하여 페이지의 코드 소스 섹션에 붙여넣습니다. distributedMapLambda

   ```
   exports.handler = async function(event, context) {
       console.log("Received Input:\n", event);

       return {
           'statusCode' : 200,
           'inputReceived' : event //returns the input that it received
       }
   };
   ```

5. 배포를 선택합니다. 함수가 배포되면 테스트를 선택하여 Lambda 함수 출력을 확인합니다.

5단계: 워크플로 프로토타입 업데이트

Step Functions 콘솔에서 워크플로를 업데이트하여 Lambda 함수의 ARN을 추가합니다.

1. 워크플로 프로토타입을 만든 탭이나 창으로 돌아갑니다.
2. CSV 데이터 처리 단계를 선택하고 구성 탭에서 다음을 수행합니다.

   a. 통합 유형에 최적화를 선택합니다.

   b. 함수 이름에 Lambda 함수 이름을 입력합니다. 표시되는 드롭다운 목록에서 함수를 선택하거나 함수 이름 입력을 선택하고 Lambda 함수 ARN을 제공합니다.

6단계: 자동 생성된 Amazon States Language 정의 검토 및 워크플로 저장

작업 및 흐름 탭에서 상태를 끌어 캔버스에 놓으면 Workflow Studio에서 워크플로의 Amazon States Language 정의를 실시간으로 자동 작성합니다. 필요에 따라 이 정의를 편집할 수 있습니다.

1. (선택 사항) Inspector 패널에서 정의를 선택하고 상태 시스템 정의를 봅니다.

> **Tip**
>
> Workflow Studio의 코드 편집기에서 ASL 정의를 볼 수도 있습니다. 코드 편집기에서 워크플로의 ASL 정의를 편집할 수도 있습니다.

다음 예제 코드에서는 워크플로에 자동으로 생성된 Amazon States Language 정의를 보여줍니다.

```
{
  "Comment": "Using Map state in Distributed mode",
  "StartAt": "Process data",
  "States": {
    "Process data": {
      "Type": "Map",
      "MaxConcurrency": 100,
      "ItemReader": {
        "ReaderConfig": {
          "InputType": "CSV",
          "CSVHeaderLocation": "FIRST_ROW"
        },
        "Resource": "arn:aws:states:::s3:getObject",
        "Parameters": {
          "Bucket": "sourceBucket",
          "Key": "csvDataset/ratings.csv"
        }
      },
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "DISTRIBUTED",
          "ExecutionType": "EXPRESS"
        },
        "StartAt": "Process CSV data",
        "States": {
          "Process CSV data": {
            "Type": "Task",
```

```
            "Resource": "arn:aws:states:::lambda:invoke",
            "OutputPath": "$.Payload",
            "Parameters": {
              "Payload.$": "$",
              "FunctionName": "arn:aws:lambda:us-
east-2:123456789012:function:distributedMapLambda"
            },
            "End": true
          }
        }
      },
      "Label": "Processdata",
      "End": true,
      "ResultWriter": {
        "Resource": "arn:aws:states:::s3:putObject",
        "Parameters": {
          "Bucket": "mapOutputs",
          "Prefix": "resultData"
        }
      },
      "ItemSelector": {
        "index.$": "$$.Map.Item.Index",
        "value.$": "$$.Map.Item.Value"
      }
    }
  }
}
```

2. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 시스템 이름 옆에 있는 편집 아이콘을 선택하십시오. MyStateMachine 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 튜토리얼에서는 이름 **DistributedMapDemo**를 입력합니다.

3. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 상태 머신 구성의 모든 기본 선택 항목을 그대로 둡니다.

4. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

7단계: 상태 시스템 실행

실행은 워크플로를 실행하여 작업을 수행하는 상태 시스템의 인스턴스입니다.

1. DistributedMapDemo페이지에서 실행 시작을 선택합니다.
2. 실행 시작 대화 상자에서 다음을 수행합니다.
   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

      > **Note**
      >
      > Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.
   3. 실행 시작을 선택합니다.
   4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

      실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예를 들어 Map 상태를 선택한 다음 맵 실행을 선택하여 맵 실행 세부 정보 페이지를 엽니다. 이 페이지에서 Distributed Map 상태의 모든 실행 세부 정보와 이 상태가 시작된 하위 워크플로 실행을 볼 수 있습니다. 이 페이지에 대한 자세한 내용은 맵 실행 검사 섹션을 참조하세요.

Lambda 함수를 사용하여 전체 데이터 배치 처리

이 자습서에서는 Distributed Map 상태의 ItemBatcher 필드를 사용하여 Lambda 함수 내 전체 항목 배치를 처리합니다. 배치마다 항목이 최대 3개까지 포함됩니다. Distributed Map 상태에서는 하위 워크플로 실행 4개를 시작합니다. 여기서 각 실행은 항목 3개를 처리하고 실행 하나는 단일 항목을 처리합니다. 각 하위 워크플로 실행은 배치에 있는 개별 항목을 반복하는 Lambda 함수를 간접적으로 호출합니다.

정수 배열에서 곱셈을 수행하는 상태 시스템을 만듭니다. 입력으로 제공하는 정수 배열은 `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`이고 곱셈 인수는 7입니다. 그러면 이러한 정수에 7의 인수를 곱한 후 생성된 결과 배열은 `[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]`입니다.

주제

- 1단계: 상태 시스템 만들기
- 2단계: Lambda 함수 만들기
- 3단계: 상태 시스템 실행

1단계: 상태 시스템 만들기

이 단계에서는 2단계에서 만들 Lambda 함수에 전체 데이터 배치를 전달하는 상태 시스템의 워크플로 프로토타입을 만듭니다.

- 다음 정의를 사용하여 Step Functions 콘솔로 상태 시스템을 만듭니다. 상태 시스템을 만드는 방법은 Distributed Map 상태 사용 시작하기 자습서의 1단계: 워크플로 프로토타입 만들기를 참조하세요.

  이 상태 시스템에서 정수 10개 배열을 입력으로 수락하고 이 배열을 3의 배치로 Lambda 함수에 전달하는 Distributed Map 상태를 정의합니다. Lambda 함수는 배치에 있는 개별 항목을 반복하고 `multiplied` 출력 배열을 반환합니다. 출력 배열에는 입력 배열에 전달된 항목에 수행된 곱셈의 결과가 포함됩니다.

> **Important**
>
> 다음 코드에 있는 Lambda 함수의 Amazon 리소스 이름(ARN)을 2단계에서 만들 함수의 ARN으로 교체해야 합니다.

```
{
  "StartAt": "Pass",
  "States": {
    "Pass": {
      "Type": "Pass",
      "Next": "Map",
      "Result": {
        "MyMultiplicationFactor": 7,
        "MyItems": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
      }
    },
    "Map": {
      "Type": "Map",
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "DISTRIBUTED",
          "ExecutionType": "STANDARD"
        },
        "StartAt": "Lambda Invoke",
        "States": {
          "Lambda Invoke": {
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "OutputPath": "$.Payload",
            "Parameters": {
              "Payload.$": "$",
              "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:functionName"
            },
            "Retry": [
              {
                "ErrorEquals": [
                  "Lambda.ServiceException",
                  "Lambda.AWSLambdaException",
                  "Lambda.SdkClientException",
```

```
                  "Lambda.TooManyRequestsException"
                ],
                "IntervalSeconds": 2,
                "MaxAttempts": 6,
                "BackoffRate": 2
              }
            ],
            "End": true
          }
        }
      },
      "End": true,
      "Label": "Map",
      "MaxConcurrency": 1000,
      "ItemBatcher": {
        "MaxItemsPerBatch": 3,
        "BatchInput": {
          "MyMultiplicationFactor.$": "$.MyMultiplicationFactor"
        }
      },
      "ItemsPath": "$.MyItems"
    }
  }
}
```

2단계: Lambda 함수 만들기

이 단계에서는 배치에서 전달된 모든 항목을 처리하는 Lambda 함수를 만듭니다.

> **Important**
>
> Lambda 함수가 상태 머신과 AWS 리전 동일한 위치에 있는지 확인하십시오.

Lambda 함수를 생성하려면

1. Lambda 콘솔을 사용하여 **ProcessEntireBatch**라는 Python 3.9 Lambda 함수를 만듭니다. Lambda 함수를 만드는 방법은 Distributed Map 상태 사용 시작하기 자습서의 4단계: Lambda 함수 구성을 참조하세요.
2. Lambda 함수의 다음 코드를 복사하여 Lambda 함수의 코드 소스 섹션에 붙여넣습니다.

```
import json

def lambda_handler(event, context):
    multiplication_factor = event['BatchInput']['MyMultiplicationFactor']
    items = event['Items']

    results = [multiplication_factor * item for item in items]

    return {
        'statusCode': 200,
        'multiplied': results
    }
```

3. Lambda 함수를 만든 후 페이지 오른쪽 상단에 표시된 함수의 ARN을 복사합니다. ARN을 복사하려면

    를 클릭합니다. 다음은 예제 ARN입니다. 여기서 *function-name*은 Lambda 함수 이름입니다(이 경우 `ProcessEntireBatch`).

   ```
   arn:aws:lambda:us-east-1:123456789012:function:function-name
   ```

   1단계에서 만든 상태 시스템에 함수 ARN을 제공해야 합니다.

4. 배포를 선택하여 변경 사항을 배포합니다.

3단계: 상태 시스템 실행

상태 시스템을 실행하면 Distributed Map 상태에서 하위 워크플로 실행 4개를 시작합니다. 여기서 각 실행은 항목 3개를 처리하고 실행 하나는 단일 항목을 처리합니다.

다음 예제에서는 하위 워크플로 실행 중 하나가 `ProcessEntireBatch` 함수에 전달한 데이터를 보여줍니다.

```
{
  "BatchInput": {
    "MyMultiplicationFactor": 7
  },
  "Items": [1, 2, 3]
}
```

이 입력이 제공되면 다음 예제에서는 Lambda 함수에서 반환한 `multiplied` 출력 배열을 보여줍니다.

```
{
  "statusCode": 200,
  "multiplied": [7, 14, 21]
}
```

상태 시스템에서 하위 워크플로 실행 4개에 대한 `multiplied` 배열 4개가 포함된 다음 출력을 반환합니다. 이러한 배열에는 개별 입력 항목의 곱셈 결과가 포함됩니다.

```
[
  {
    "statusCode": 200,
    "multiplied": [7, 14, 21]
  },
  {
    "statusCode": 200,
    "multiplied": [28, 35, 42]
  },
  {
    "statusCode": 200,
    "multiplied": [49, 56, 63]
  },
  {
    "statusCode": 200,
    "multiplied": [70]
  }
]
```

반환된 모든 배열 항목을 단일 출력 배열로 결합하려면 ResultSelector 필드를 사용하면 됩니다. Distributed Map 상태 내에서 이 필드를 정의하여 모든 `multiplied` 배열을 찾고 이러한 배열 내의 모든 항목을 추출한 다음 단일 출력 배열로 결합합니다.

`ResultSelector` 필드를 사용하려면 다음 예제와 같이 상태 시스템 정의를 업데이트합니다.

```
{
  "StartAt": "Pass",
  "States": {
    ...
    ...
    "Map": {
      "Type": "Map",
```

```
      ...
      ...
      "ItemsPath": "$.MyItems",
      "ResultSelector": {
        "multiplied.$": "$..multiplied[*]"
      }
    }
  }
}
```

업데이트된 상태 시스템에서 다음 예제와 같이 통합된 출력 배열을 반환합니다.

```
{
  "multiplied": [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
}
```

Lambda 함수를 사용하여 개별 데이터 항목 처리

이 자습서에서는 Distributed Map 상태의 ItemBatcher 필드를 사용하여 Lambda 함수를 통해 배치로 있는 개별 항목을 반복합니다. Distributed Map 상태는 하위 워크플로 실행 4개를 시작합니다. 이러한 각 하위 워크플로는 Inline Map 상태를 실행합니다. 반복마다 Inline Map 상태는 Lambda 함수를 간접적으로 호출하고 배치의 단일 항목을 함수로 전달합니다. 그러면 Lambda 함수에서 항목을 처리하고 결과를 반환합니다.

정수 배열에서 곱셈을 수행하는 상태 시스템을 만듭니다. 입력으로 제공하는 정수 배열은 `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`이고 곱셈 인수는 7입니다. 그러면 이러한 정수에 7의 인수를 곱한 후 생성된 결과 배열은 `[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]`입니다.

주제

- 1단계: 상태 시스템 만들기
- 2단계: Lambda 함수 만들기
- 3단계: 상태 시스템 실행

1단계: 상태 시스템 만들기

이 단계에서는 배치의 단일 항목을 2단계에서 만들 Lambda 함수의 각 간접 호출에 전달하는 상태 시스템의 워크플로 프로토타입을 만듭니다.

- 다음 정의를 사용하여 Step Functions 콘솔로 상태 시스템을 만듭니다. 상태 시스템을 만드는 방법은 Distributed Map 상태 사용 시작하기 자습서의 1단계: 워크플로 프로토타입 만들기를 참조하세요.

  이 상태 시스템에서 정수 10개 배열을 입력으로 수락하고 이러한 배열 항목을 배치로 하위 워크플로 실행에 전달하는 Distributed Map 상태를 정의합니다. 각 하위 워크플로 실행은 항목 3개 배치를 입력으로 수신하고 Inline Map 상태를 실행합니다. Inline Map 상태가 반복될 때마다 Lambda 함수가 간접적으로 호출되고 배치의 항목이 함수로 전달됩니다. 그런 다음 이 함수는 항목에 승수 7을 곱하고 결과를 반환합니다.

  각 하위 워크플로 실행 출력은 전달된 각 항목에 대한 곱셈 결과가 포함된 JSON 배열입니다.

  > **Important**
  >
  > 다음 코드에 있는 Lambda 함수의 Amazon 리소스 이름(ARN)을 2단계에서 만들 함수의 ARN으로 교체해야 합니다.

```
{
  "StartAt": "Pass",
  "States": {
    "Pass": {
      "Type": "Pass",
      "Next": "Map",
      "Result": {
        "MyMultiplicationFactor": 7,
        "MyItems": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
      }
    },
    "Map": {
      "Type": "Map",
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "DISTRIBUTED",
          "ExecutionType": "STANDARD"
        },
        "StartAt": "InnerMap",
        "States": {
          "InnerMap": {
            "Type": "Map",
            "ItemProcessor": {
```

```
              "ProcessorConfig": {
                "Mode": "INLINE"
              },
              "StartAt": "Lambda Invoke",
              "States": {
                "Lambda Invoke": {
                  "Type": "Task",
                  "Resource": "arn:aws:states:::lambda:invoke",
                  "OutputPath": "$.Payload",
                  "Parameters": {
                    "Payload.$": "$",
                    "FunctionName": "arn:aws:lambda:us-
east-1:123456789012:function:functionName"
                  },
                  "Retry": [
                    {
                      "ErrorEquals": [
                        "Lambda.ServiceException",
                        "Lambda.AWSLambdaException",
                        "Lambda.SdkClientException",
                        "Lambda.TooManyRequestsException"
                      ],
                      "IntervalSeconds": 2,
                      "MaxAttempts": 6,
                      "BackoffRate": 2
                    }
                  ],
                  "End": true
                }
              }
            },
            "End": true,
            "ItemsPath": "$.Items",
            "ItemSelector": {
              "MyMultiplicationFactor.$": "$.BatchInput.MyMultiplicationFactor",
              "MyItem.$": "$$.Map.Item.Value"
            }
          }
        }
      },
      "End": true,
      "Label": "Map",
      "MaxConcurrency": 1000,
      "ItemsPath": "$.MyItems",
```

```
      "ItemBatcher": {
        "MaxItemsPerBatch": 3,
        "BatchInput": {
          "MyMultiplicationFactor.$": "$.MyMultiplicationFactor"
        }
      }
    }
  }
}
```

2단계: Lambda 함수 만들기

이 단계에서는 배치에서 전달된 각 항목을 처리하는 Lambda 함수를 만듭니다.

> **Important**
>
> Lambda 함수가 상태 머신과 AWS 리전 동일한 위치에 있는지 확인하십시오.

Lambda 함수를 생성하려면

1. Lambda 콘솔을 사용하여 **ProcessSingleItem**라는 Python 3.9 Lambda 함수를 만듭니다. Lambda 함수를 만드는 방법은 Distributed Map 상태 사용 시작하기 자습서의 4단계: Lambda 함수 구성을 참조하세요.
2. Lambda 함수의 다음 코드를 복사하여 Lambda 함수의 코드 소스 섹션에 붙여넣습니다.

```
import json

def lambda_handler(event, context):

    multiplication_factor = event['MyMultiplicationFactor']
    item = event['MyItem']

    result = multiplication_factor * item

    return {
        'statusCode': 200,
        'multiplied': result
    }
```

3. Lambda 함수를 만든 후 페이지 오른쪽 상단에 표시된 함수의 ARN을 복사합니다. ARN을 복사하려면

   를 클릭합니다. 다음은 예제 ARN입니다. 여기서 *function-name*은 Lambda 함수 이름입니다(이 경우 `ProcessSingleItem`).

   ```
   arn:aws:lambda:us-east-1:123456789012:function:function-name
   ```

   1단계에서 만든 상태 시스템에 함수 ARN을 제공해야 합니다.
4. 배포를 선택하여 변경 사항을 배포합니다.

3단계: 상태 시스템 실행

상태 시스템을 실행하면 Distributed Map 상태에서 하위 워크플로 실행 4개를 시작합니다. 여기서 각 실행은 항목 3개를 처리하고 실행 하나는 단일 항목을 처리합니다.

다음 예제에서는 하위 워크플로 실행 내의 `ProcessSingleItem` 함수 간접 호출 중 하나에 전달된 데이터를 보여줍니다.

```
{
  "MyMultiplicationFactor": 7,
  "MyItem": 1
}
```

이 입력이 제공되면 다음 예제에서는 Lambda 함수에서 반환한 출력을 보여줍니다.

```
{
  "statusCode": 200,
  "multiplied": 7
}
```

다음 예제에서는 하위 워크플로 실행 중 하나의 출력 JSON 배열을 보여줍니다.

```
[
  {
    "statusCode": 200,
    "multiplied": 7
  },
```

```
  {
    "statusCode": 200,
    "multiplied": 14
  },
  {
    "statusCode": 200,
    "multiplied": 21
  }
]
```

상태 시스템은 하위 워크플로 실행 4개에 대한 배열 4개가 포함된 다음 출력을 반환합니다. 이러한 배열에는 개별 입력 항목의 곱셈 결과가 포함됩니다.

마지막으로, 상태 시스템 출력은 하위 워크플로 실행 4개에 대해 반환된 모든 곱셈 결과를 합친 `multiplied` 배열입니다.

```
[
  [
    {
      "statusCode": 200,
      "multiplied": 7
    },
    {
      "statusCode": 200,
      "multiplied": 14
    },
    {
      "statusCode": 200,
      "multiplied": 21
    }
  ],
  [
    {
      "statusCode": 200,
      "multiplied": 28
    },
    {
      "statusCode": 200,
      "multiplied": 35
    },
    {
      "statusCode": 200,
      "multiplied": 42
```

```
    }
  ],
  [
    {
      "statusCode": 200,
      "multiplied": 49
    },
    {
      "statusCode": 200,
      "multiplied": 56
    },
    {
      "statusCode": 200,
      "multiplied": 63
    }
  ],
  [
    {
      "statusCode": 200,
      "multiplied": 70
    }
  ]
]
```

하위 워크플로 실행에서 반환한 모든 곱셈 결과를 단일 출력 배열로 결합하려면 ResultSelector 필드를 사용하면 됩니다. Distributed Map 상태 내에서 이 필드를 정의하여 모든 결과를 찾고 개별 결과를 추출한 다음 `multiplied` 단일 출력 배열로 결합합니다.

`ResultSelector` 필드를 사용하려면 다음 예제와 같이 상태 시스템 정의를 업데이트합니다.

```
{
  "StartAt": "Pass",
  "States": {
    ...
    ...
    "Map": {
      "Type": "Map",
      ...
      ...
      "ItemBatcher": {
        "MaxItemsPerBatch": 3,
        "BatchInput": {
          "MyMultiplicationFactor.$": "$.MyMultiplicationFactor"
```

```
        }
      },
      "ItemsPath": "$.MyItems",
      "ResultSelector": {
        "multiplied.$": "$..multiplied"
      }
    }
  }
}
```

업데이트된 상태 시스템에서 다음 예제와 같이 통합된 출력 배열을 반환합니다.

```
{
  "multiplied": [7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
}
```

Amazon S3 Events에 대한 응답으로 상태 시스템 실행 시작

Amazon EventBridge 규칙에 대한 응답으로 AWS Step Functions 상태 시스템을 실행할 수 있습니다.

이 자습서에서는 상태 시스템을 Amazon EventBridge 규칙의 대상으로 구성하는 방법을 보여줍니다. 이 규칙은 파일이 Amazon Simple Storage Service(S3) 버킷에 추가될 때 상태 시스템 실행을 시작합니다.

실용적 애플리케이션의 경우 섬네일 만들기나 이미지 및 비디오 파일에서 Amazon Rekognition 분석 실행과 같이 버킷에 추가하는 파일에서 작업을 수행하는 상태 시스템을 시작할 수 있습니다.

이 자습서에서는 파일을 Amazon S3 버킷에 업로드하여 `Helloworld` 상태 시스템 실행을 시작합니다. 그런 다음 해당 실행의 예제 입력을 검토하여 EventBridge로 전달된 Amazon S3 이벤트 알림의 입력에 포함된 정보를 식별합니다.

주제

- 사전 조건: 상태 시스템 생성
- 1단계: Amazon S3에 버킷 만들기
- 2단계: EventBridge로 Amazon S3 이벤트 알림 활성화
- 3단계: Amazon EventBridge 규칙 만들기
- 4단계: 규칙 테스트
- 실행 입력의 예

사전 조건: 상태 시스템 생성

상태 시스템을 Amazon EventBridge 대상으로 구성하려면 먼저 상태 시스템을 만들어야 합니다.

- 기본 상태 시스템을 만들려면 Lambda 함수를 사용하는 상태 시스템 생성 자습서를 사용합니다.
- `Helloworld` 상태 시스템이 이미 있으면 다음 단계로 진행합니다.

1단계: Amazon S3에 버킷 만들기

`Helloworld` 상태 시스템이 있으므로 이제 파일을 저장하는 Amazon S3 버킷을 만들어야 합니다. 이 자습서의 3단계에서는 파일이 이 버킷에 업로드될 때 EventBridge에서 상태 시스템 실행을 트리거하도록 규칙을 설정합니다.

1. Amazon S3 콘솔로 이동한 다음 버킷 생성을 선택하여 파일을 저장하고 Amazon S3 이벤트 규칙을 트리거할 버킷을 만듭니다.
2. 버킷 이름을 입력합니다(예: *username*`-sfn-tutorial`).

   > **Note**
   >
   > 버킷 이름은 Amazon S3의 모든 AWS 리전에 있는 모든 기존 버킷 이름에서 고유해야 합니다. 자신의 *### ##*을 사용하여 이 이름을 고유하게 합니다. 동일한 AWS 리전의 모든 리소스를 생성해야 합니다.

3. 페이지에 있는 모든 기본 선택 항목을 그대로 두고 버킷 생성을 선택합니다.

2단계: EventBridge로 Amazon S3 이벤트 알림 활성화

Amazon S3 버킷을 만든 후 S3 버킷에서 파일 업로드와 같은 특정 이벤트가 발생할 때마다 이벤트를 EventBridge로 보내도록 버킷을 구성합니다.

1. Amazon S3 콘솔로 이동합니다.
2. 버킷(Buckets) 목록에서 이벤트를 사용 설정하려는 버킷의 이름을 선택합니다.
3. [속성(Properties)]을 선택합니다.
4. 페이지를 아래로 스크롤하여 이벤트 알림 섹션을 본 다음 Amazon EventBridge 하위 섹션에서 편집을 선택합니다.
5. 이 버킷의 모든 이벤트에 대해 Amazon EventBridge에 알림 보내기에서 켜기를 선택합니다.

6. Save changes(변경 사항 저장)를 선택합니다.

> **Note**
>
> EventBridge를 사용 설정한 후에는 변경 사항이 적용되는 데 5분 정도 걸립니다.

3단계: Amazon EventBridge 규칙 만들기

상태 시스템이 있고 Amazon S3 버킷을 만들고 이벤트 알림을 EventBridge에 보내도록 구성했으면 EventBridge 규칙을 만듭니다.

> **Note**
>
> Amazon S3 버킷과 동일한 AWS 리전에 EventBridge 규칙을 구성해야 합니다.

규칙을 만들려면

1. Amazon EventBridge 콘솔로 이동하고 규칙 생성을 선택합니다.

> **Tip**
>
> 또는 EventBridge 콘솔의 탐색 창에서 버스에서 규칙을 선택한 다음 규칙 생성을 선택합니다.

2. 규칙의 이름(예: *S3Step Functions*)을 입력하고 선택적으로 규칙에 대한 설명을 입력합니다.
3. 이벤트 버스 및 규칙 유형에서는 기본 선택을 유지합니다.
4. 다음(Next)을 선택합니다. 그러면 이벤트 패턴 작성 페이지가 열립니다.
5. 이벤트 패턴 섹션까지 아래로 스크롤하고 다음을 수행합니다.
   a. 이벤트 소스의 경우 기본 선택 항목 AWS 이벤트 또는 EventBridge 파트너 이벤트를 그대로 둡니다.
   b. AWS 서비스에 Simple Storage Service(S3)를 선택합니다.
   c. 이벤트 유형에 Amazon S3 이벤트 알림을 선택합니다.
   d. 특정 이벤트를 선택한 다음 생성된 객체를 선택합니다.

e. 이름 기준 특정 버킷을 선택하고 1단계에서 만든 버킷 이름(*username*-sfn-tutorial)을 입력합니다.

f. 다음(Next)을 선택합니다. 그러면 대상 선택 페이지가 열립니다.

대상을 생성하려면

1. 대상 1에서 기본 선택 항목 AWS 서비스를 그대로 둡니다.
2. 대상 선택 드롭다운 목록에서 Step Functions 상태 시스템을 선택합니다.
3. 상태 시스템 목록에서 이전에 만든 상태 시스템(예: Helloworld)을 선택합니다.
4. 페이지에 있는 모든 기본 선택 항목을 그대로 두고 다음을 선택합니다. 그러면 태그 구성 페이지가 열립니다.
5. 다음을 다시 선택합니다. 그러면 검토 및 생성 페이지가 열립니다.
6. 규칙의 세부 정보를 검토하고 규칙 생성(Create rule)을 선택합니다.

   규칙이 생성되고 규칙 페이지에 모든 Amazon EventBridge 규칙이 나열됩니다.

4단계: 규칙 테스트

모두 구현되었으므로 Amazon S3 버킷에 파일 추가를 테스트한 다음 그에 따른 상태 시스템 실행의 입력을 살펴봅니다.

1. 파일을 Amazon S3에서 버킷에 추가합니다.

   Amazon S3 콘솔로 이동하고 파일을 저장할 생성된 버킷을 선택한 다음(*username*-sfn-tutorial) 업로드를 선택합니다.
2. 파일(예: *test.png*)을 추가한 다음 업로드를 선택합니다.

   이렇게 하면 상태 시스템의 실행이 시작되고 AWS CloudTrail에서 정보가 입력으로 전달됩니다.
3. 상태 시스템의 실행을 점검합니다.

   Step Functions 콘솔로 이동하고 Amazon EventBridge 규칙(Helloworld)에 사용된 상태 시스템을 선택합니다.
4. 해당 상태 시스템의 최근 실행을 선택하고 실행 입력 섹션을 확장합니다.

   이 입력에는 버킷 이름, 객체 이름 등의 정보가 포함되어야 합니다. 실제 사용 사례에서 상태 시스템은 이 입력을 사용하여 해당 객체에 대한 작업을 수행할 수 있습니다.

실행 입력의 예

다음 예제에서는 상태 시스템 실행에 대한 일반적인 입력을 보여줍니다.

```
{
    "version": "0",
    "id": "6c540ad4-0671-9974-6511-756fbd7771c3",
    "detail-type": "Object Created",
    "source": "aws.s3",
    "account": "123456789012",
    "time": "2023-06-23T23:45:48Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:s3:::username-sfn-tutorial"
    ],
    "detail": {
      "version": "0",
      "bucket": {
        "name": "username-sfn-tutorial"
      },
      "object": {
        "key": "test.png",
        "size": 800704,
        "etag": "f31d8546bb67845b4d3048cde533b937",
        "sequencer": "00621049BA9A8C712B"
      },
      "request-id": "79104EXAMPLEB723",
      "requester": "123456789012",
      "source-ip-address": "200.0.100.11",
      "reason": "PutObject"
    }
  }
```

API Gateway를 사용하여 Step Functions API 만들기

Amazon API Gateway를 사용하여 API를 API 게이트웨이 API의 메서드와 연결할 수 있습니다. AWS Step Functions HTTPS 요청이 API 메서드로 전송되면 API Gateway에서 Step Functions API 작업을 간접적으로 호출합니다.

이 자습서에서는 하나의 리소스 및 POST 메서드를 사용하여 `StartExecution` API 작업과 통신하는 API를 생성하는 방법을 보여줍니다. AWS Identity and Access Management (IAM) 콘솔을 사용하여

API Gateway에 대한 역할을 생성합니다. 그런 다음 API Gateway 콘솔을 사용하여 API Gateway API를 만들고 리소스와 메서드를 만들고 메서드를 `StartExecution` API 작업에 매핑합니다. 마지막으로, API를 배포 및 테스트합니다.

> **Note**
>
> `StartExecution`을 직접적으로 호출하여 Amazon API Gateway에서 Step Functions 실행을 시작할 수 있지만 결과를 가져오려면 `DescribeExecution`을 직접적으로 호출해야 합니다.

주제

1단계: API Gateway에 대한 IAM 역할 만들기

API Gateway API를 만들기 전에 Step Functions API 작업을 직접적으로 호출할 수 있는 권한을 API Gateway에 부여해야 합니다.

API Gateway에 대한 권한 설정하기

1. IAM 콘솔에 로그인하고 역할, 역할 생성을 선택합니다.
2. 신뢰할 수 있는 엔터티 선택(Select trusted entity) 페이지에서 다음을 수행합니다.
   a. 신뢰할 수 있는 엔터티 유형의 경우 AWS 서비스의 기본 선택 항목을 그대로 둡니다.
   b. 사용 사례의 드롭다운 목록에서 API Gateway를 선택합니다.
3. API Gateway를 선택한 다음 다음을 선택합니다.
4. 권한 추가 페이지에서 다음을 선택합니다.
5. (선택 사항) 이름 지정, 검토 및 생성 페이지에서 역할 이름과 같은 세부 정보를 입력합니다. 예를 들면 **`APIGatewayToStepFunctions`**를 입력합니다.
6. 역할 생성을 선택합니다.

   역할 목록에 IAM 역할이 표시됩니다.
7. 다음 예제와 같이 역할 이름을 선택하고 역할 ARN을 적어 둡니다.

```
arn:aws:iam::123456789012:role/APIGatewayToStepFunctions
```

IAM 역할에 정책 연결하기

1. 역할 페이지에서 역할(APIGatewayToStepFunctions)을 찾아 선택합니다.
2. 권한 탭에서 권한 추가, 정책 연결을 차례로 선택합니다.
3. 정책 연결 페이지에서 AWSStepFunctionsFullAccess를 검색하고 정책을 선택한 다음 권한 추가를 선택합니다.

2단계: API Gateway API 만들기

IAM 역할을 만든 후에는 사용자 지정 API Gateway API를 만들 수 있습니다.

API를 생성하려면

1. Amazon API Gateway 콘솔을 열고 API 생성을 선택합니다.
2. API 유형 선택 페이지의 REST API 창에서 빌드를 선택합니다.
3. REST API 생성 페이지에서 새 API를 선택한 다음 ***StartExecutionAPI #### API#*** 입력합니다.
4. API 엔드포인트 유형을 리전별로 유지한 다음 API 생성을 선택합니다.

리소스를 생성하려면

1. ***StartExecutionAPI#*** 리소스 페이지에서 리소스 생성을 선택합니다.
2. 리소스 생성 페이지에서 리소스 이름에 **execution**을 입력한 다음 리소스 생성을 선택합니다.

POST 메서드 만들기

1. /execution 리소스를 선택한 다음 메서드 생성을 선택합니다.
2. 메서드 유형에서 POST를 선택합니다.
3. 통합 유형에서 AWS 서비스를 선택합니다.
4. AWS 리전의 목록에서 리전을 선택합니다.

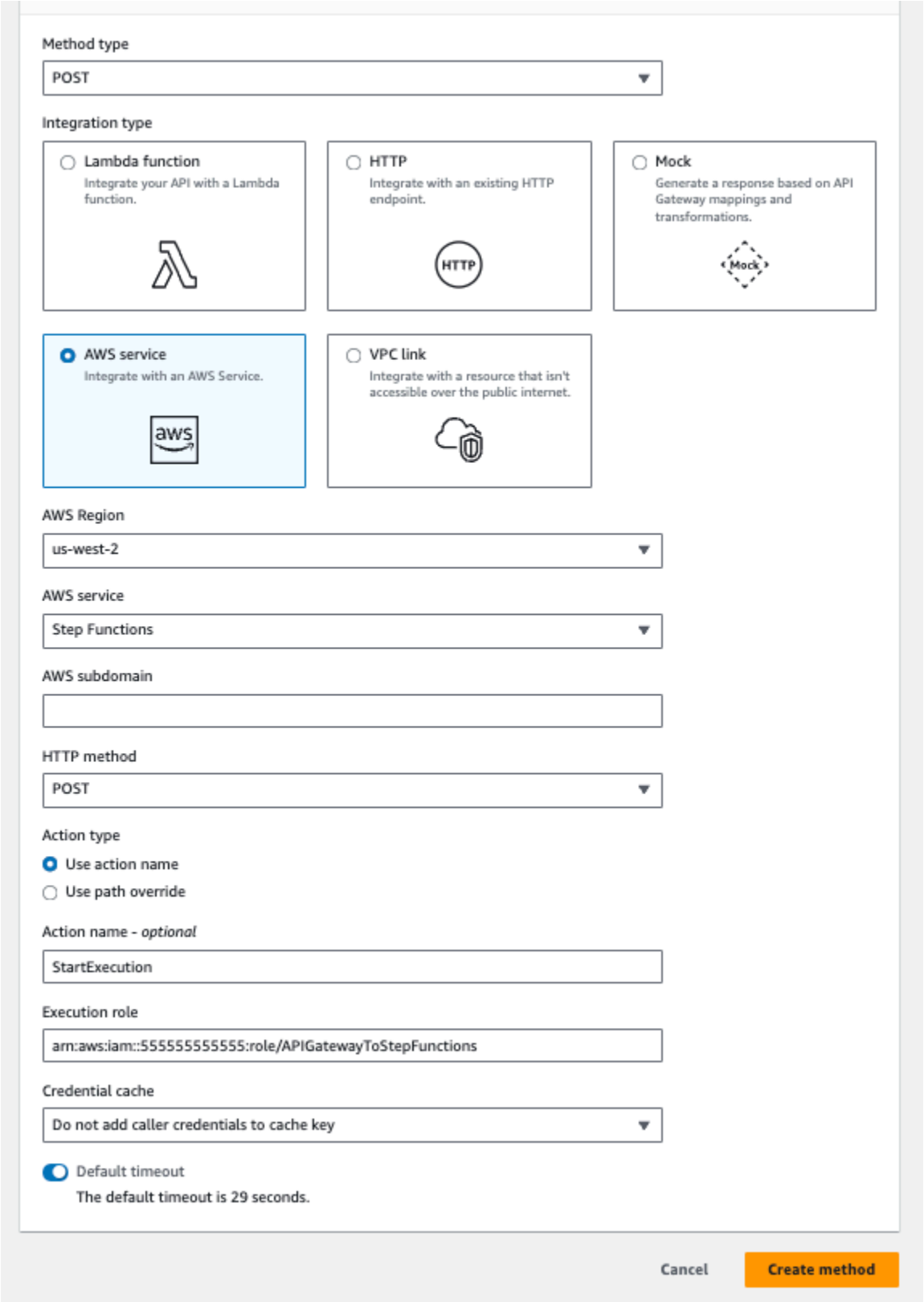
> **Note**
>
> 현재 Step Functions를 지원하는 리전은 지원되는 리전을 참조하세요.

5. AWS 서비스의 목록에서 Step Functions를 선택합니다.
6. AWS 하위 도메인은 비워 둡니다.
7. HTTP 메서드 목록에서 POST를 선택합니다.

> **Note**
>
> 모든 Step Functions API 작업에서 HTTP POST 메서드를 사용합니다.

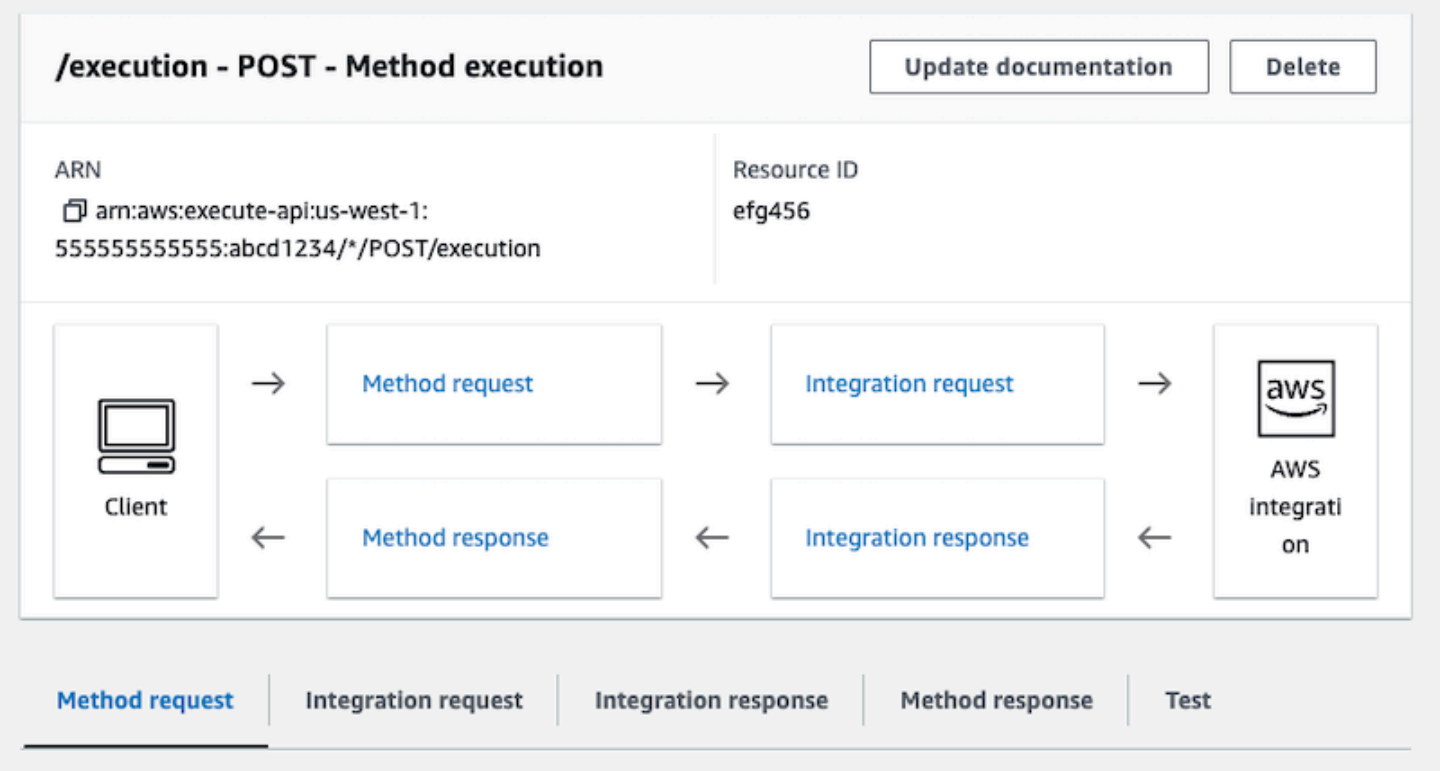
8. 작업 유형에서 작업 이름 사용을 선택합니다.
9. 함수 이름에 **`StartExecution`**를 입력합니다.
10. 다음 예제와 같이 실행 역할에 앞에서 만든 IAM 역할의 역할 ARN을 입력합니다.

```
arn:aws:iam::123456789012:role/APIGatewayToStepFunctions
```



11. 보안 인증 캐시 및 기본 제한 시간에 대한 기본 옵션을 유지한 다음 저장을 선택합니다.

API Gateway 및 Step Functions 간의 시각적 매핑은 /execution - POST - 메서드 실행 페이지에 표시됩니다.



3단계: API Gateway API 테스트 및 배포

API를 생성한 후 테스트하고 배포합니다.

API Gateway 및 Step Functions 간의 통신을 테스트하기

1. /execution - POST - 메서드 실행 페이지에서 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
2. /execution - POST - 메서드 테스트 탭에서 기존 상태 시스템의 ARN을 사용하여 다음 요청 파라미터를 요청 본문 섹션에 복사하거나 Lambda 함수를 사용하는 새 상태 시스템을 만든 다음 테스트를 선택합니다.

```
{
   "input": "{}",
   "name": "MyExecution",
   "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld"
}
```

자세한 내용은 AWS Step Functions API 참조의 `StartExecution` 요청 구문을 참조하세요.

> **Note**
>
> API Gateway 직접 호출 본문에 상태 머신의 ARN을 포함하지 않으려면 다음 예제와 같이 통합 요청 탭에서 매핑 템플릿을 구성하면 됩니다.
>
> ```
> {
>     "input": "$util.escapeJavaScript($input.json('$'))",
>     "stateMachineArn": "$util.escapeJavaScript($stageVariables.arn)"
> }
> ```
>
> 이러한 방법을 사용하면 배포 상태(예: `dev`, `test` 및 `prod`)에 따라 다양한 상태 시스템의 ARN을 지정할 수 있습니다. 매핑 템플릿에서 단계 변수를 지정하는 방법에 대한 자세한 내용은 API Gateway 개발자 안내서의 `$stageVariables`를 참조하세요.

3. 실행이 시작되고 실행 ARN 및 해당 epoch 날짜가 응답 본문 아래에 표시됩니다.

```
{
   "executionArn": "arn:aws:states:us-
east-1:123456789012:execution:HelloWorld:MyExecution",
   "startDate": 1486768956.878
}
```

> **Note**
>
> AWS Step Functions 콘솔에서 상태 시스템을 선택하여 실행을 볼 수 있습니다.

API를 배포하려면

1. ***StartExecutionAPI# ### ##### API*** 배포를 선택합니다.
2. 스테이지에서 새 스테이지를 선택합니다.
3. 단계 이름에 **alpha**를 입력합니다.
4. (선택 사항) 설명에 설명을 입력합니다.
5. 배포를 선택합니다.

배포를 테스트하려면

1. ***StartExecutionAPI#*** 스테이지 페이지에서 알파,/, /실행, POST를 확장한 다음 POST 메서드를 선택합니다.
2. 메서드 재정의에서 복사 아이콘을 선택하여 API의 간접 호출 URL을 복사합니다. 전체 URL은 다음 예제와 비슷해야 합니다.

```
https://a1b2c3d4e5.execute-api.us-east-1.amazonaws.com/alpha/execution
```

3. 다음 예제와 같이 명령줄에서 상태 머신의 ARN을 사용하여 `curl` 명령을 실행하고 배포 URL을 호출합니다.

```
curl -X POST -d '{"input": "{}","name": "MyExecution","stateMachineArn":
 "arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld"}' https://
a1b2c3d4e5.execute-api.us-east-1.amazonaws.com/alpha/execution
```

다음 예제와 같이 실행 ARN 및 해당 epoch 날짜가 반환됩니다.

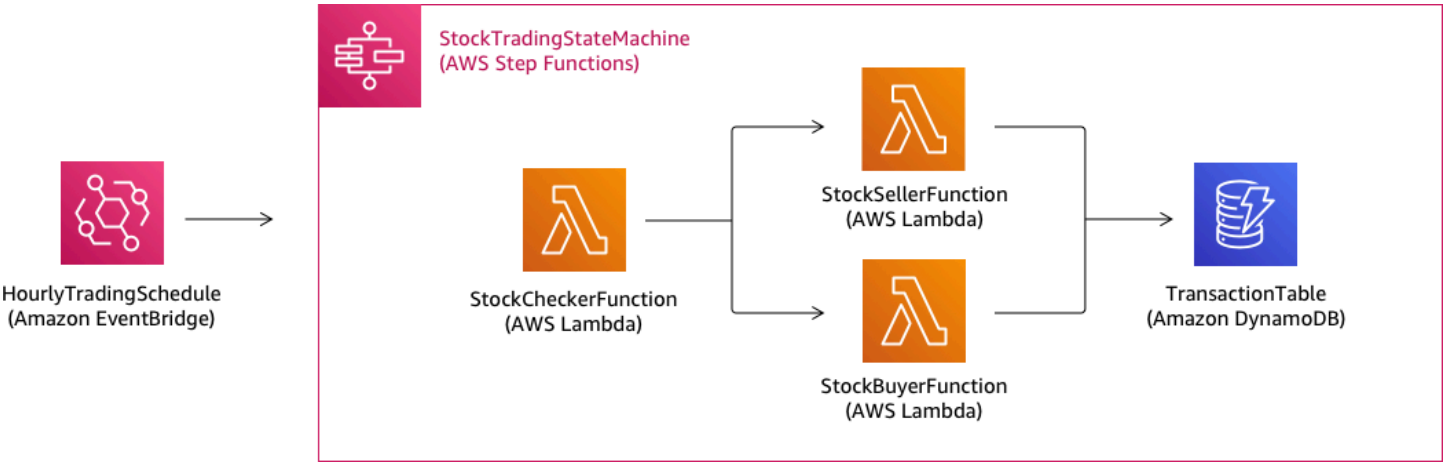
```
{"executionArn":"arn:aws:states:us-
east-1:123456789012:execution:HelloWorld:MyExecution","startDate":1.486772644911E9}
```

> **Note**
>
> “인증 토큰 누락” 오류가 발생하면 간접 호출 URL이 /execution으로 끝나는지 확인합니다.

AWS SAM을 사용하여 Step Functions 상태 시스템 만들기

이 가이드에서는 AWS Step Functions 상태 머신이 포함된 샘플 AWS SAM 애플리케이션을 다운로드, 빌드 및 배포합니다. 이 애플리케이션은 사전 정의된 일정에 따라 실행되는 모의 주식 거래 워크플로를 생성합니다(요금이 부과되지 않도록 일정이 기본적으로 비활성화되어 있음).

다음 다이어그램은 이 애플리케이션의 구성 요소를 보여줍니다.



다음은 샘플 애플리케이션을 생성하기 위해 실행하는 명령의 미리 보기입니다. 이러한 각 명령에 대한 자세한 내용은 이 페이지의 뒷부분에 있는 섹션을 참조하십시오.

```
# Step 1 - Download a sample application. For this tutorial you
#   will follow the prompts to select an AWS Quick Start Template
#   called 'Multi-step workflow'
sam init

# Step 2 - Build your application
cd project-directory
sam build

# Step 3 - Deploy your application
sam deploy --guided
```

필수 조건

이 가이드에서는 OS에 대해 AWS SAM CLI 설치 단계를 완료했다고 가정합니다. 다음 작업을 수행했다고 가정합니다.

1. AWS 계정을 만들었습니다.
2. IAM 권한을 구성했습니다.
3. Homebrew를 설치했습니다. 참고: Homebrew는 Linux 및 macOS의 유일한 사전 조건입니다.
4. AWS SAM CLI를 설치했습니다. 참고: 버전 0.52.0 이상이 설치되어 있는지 확인하십시오. `sam --version` 명령을 실행하여 설치되어 있는 버전을 확인할 수 있습니다.

1단계: 샘플 AWS SAM 애플리케이션 다운로드

실행할 명령:

```
sam init
```

화면에 표시되는 프롬프트에 따라 다음을 선택합니다.

1. 템플릿: AWS 빠른 시작 템플릿
2. 언어: Python, Ruby, NodeJS, Go, Java 또는 .NET
3. 프로젝트 이름: (선택한 이름 - 기본값은 sam-app임)
4. 빠른 시작 애플리케이션: 다단계 워크플로

AWS SAM에서 수행 중인 작업:

이 명령은 '프로젝트 이름' 프롬프트에 제공한 이름으로 디렉터리를 생성합니다(기본값은 sam-app). 구체적인 디렉터리 내용은 선택한 언어에 따라 다릅니다.

Python 런타임 중 하나를 선택할 때 디렉터리 내용은 다음과 같습니다.

```
### README.md
### functions
#   ### __init__.py
#   ### stock_buyer
#   #   ### __init__.py
#   #   ### app.py
#   #   ### requirements.txt
#   ### stock_checker
#   #   ### __init__.py
#   #   ### app.py
#   #   ### requirements.txt
#   ### stock_seller
#       ### __init__.py
#       ### app.py
#       ### requirements.txt
### statemachine
#   ### stock_trader.asl.json
### template.yaml
### tests
    ### unit
```

```
        ### __init__.py
        ### test_buyer.py
        ### test_checker.py
        ### test_seller.py
```

살펴볼 수 있는 두 가지 특히 흥미로운 파일이 있습니다.

- `template.yaml`: 애플리케이션의 AWS 리소스를 정의하는 AWS SAM 템플릿을 포함합니다.
- `statemachine/stockTrader.asl.json`: Amazon States Language에 작성된 애플리케이션의 상태 머신 정의를 포함합니다.

`template.yaml` 파일에서 상태 머신 정의 파일을 가리키는 다음 항목을 볼 수 있습니다.

```
    Properties:
      DefinitionUri: statemachine/stock_trader.asl.json
```

상태 시스템 정의를 AWS SAM 템플릿에 임베딩하는 대신 별도의 파일로 보관하는 것이 유용할 수 있습니다. 예를 들어 템플릿에 정의를 포함하지 않으면 상태 시스템 정의에 대한 변경 사항을 더 쉽게 추적할 수 있습니다. Workflow Studio를 사용하여 상태 시스템 정의를 생성 및 유지하고 정의를 템플릿에 병합하지 않고도 콘솔에서 Amazon States Language 사양 파일로 직접 내보낼 수 있습니다.

샘플 애플리케이션에 대한 자세한 내용은 프로젝트 디렉터리의 `README.md` 파일을 참조하십시오.

2단계: 애플리케이션 빌드

실행할 명령:

먼저 프로젝트 디렉터리(예: 샘플 애플리케이션의 `template.yaml` 파일이 있는 디렉터리, 기본값은 `sam-app`임)로 변경한 후 다음 명령을 실행합니다.

```
sam build
```

출력 예제:

```
Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml
```

```
Commands you can use next
=========================
[*] Invoke Function: sam local invoke
[*] Deploy: sam deploy --guided
```

AWS SAM에서 수행 중인 작업:

AWS SAM CLI는 종속성을 빌드하기 위해 여러 Lambda 런타임에 대한 추상화를 제공하며 모든 항목이 패키징되고 배포될 수 있도록 모든 빌드 아티팩트를 스테이징 폴더에 복사합니다. 이 `sam build` 명령은 애플리케이션에 있는 모든 종속성을 빌드하고 빌드 아티팩트를 `.aws-sam/build` 아래의 폴더에 복사합니다.

3단계: AWS 클라우드에 애플리케이션 배포

실행할 명령:

```
sam deploy --guided
```

화면에 표시되는 프롬프트를 따릅니다. 대화형 환경에서 제공되는 기본 옵션을 수락하려면 `Enter`로 응답하면 됩니다.

AWS SAM에서 수행 중인 작업:

이 명령은 애플리케이션을 AWS 클라우드에 배포합니다. `sam build` 명령으로 빌드한 배포 아티팩트를 가져와서 패키징하고 AWS SAM CLI에서 만든 Amazon S3 버킷에 업로드하고 AWS CloudFormation을 사용하여 애플리케이션을 배포합니다. 배포 명령의 출력에서 AWS CloudFormation 스택에 대한 변경 사항을 볼 수 있습니다.

다음 단계를 수행하면 예제 Step Functions 상태 시스템이 성공적으로 배포되었는지 확인할 수 있습니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/states/에서 Step Functions 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 상태 머신을 선택합니다.
3. 목록에서 새 상태 머신을 찾아 선택합니다. 여기에는 StockTradingStateMachine-*<unique-hash>*라는 이름이 지정됩니다.
4. 정의 탭을 선택합니다.

이제 상태 머신의 시각적 표현을 볼 수 있습니다. 시각적 표현이 프로젝트 디렉터리의 `statemachine/stockTrader.asl.json` 파일에 있는 상태 머신 정의와 일치하는지 확인할 수 있습니다.

문제 해결

SAM CLI 오류: "해당 옵션 없음: --guided"

`sam deploy`를 실행하면 다음 오류가 표시됩니다.

```
Error: no such option: --guided
```

즉, `--guided` 파라미터를 지원하지 않는 이전 버전의 AWS SAM CLI를 사용하고 있습니다. 이 문제를 해결하려면 AWS SAM CLI 버전을 0.33.0 이상으로 업데이트하거나 `sam deploy` 명령에서 `--guided` 파라미터를 생략하면 됩니다.

SAM CLI 오류: "관리형 리소스를 생성하지 못했습니다. 자격 증명을 찾을 수 없습니다."

`sam deploy`를 실행하면 다음 오류가 표시됩니다.

```
Error: Failed to create managed resources: Unable to locate credentials
```

즉, AWS SAM CLI에서 AWS 서비스를 직접적으로 호출할 수 있도록 AWS 보안 인증 정보를 설정하지 않았습니다. 이 문제를 해결하려면 AWS 보안 인증 정보를 설정해야 합니다. 자세한 내용은 AWS Serverless Application Model 개발자 안내서의 AWS 보안 인증 정보 설정 섹션을 참조하세요.

정리

이 자습서를 실행하여 만든 AWS 리소스가 더 이상 필요하지 않은 경우 배포한 AWS CloudFormation 스택을 삭제하여 리소스를 제거할 수 있습니다.

AWS Management Console을 사용하여 이 자습서에서 생성한 AWS CloudFormation 스택을 삭제하려면 다음 단계를 수행합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/cloudformation에서 AWS CloudFormation 콘솔을 엽니다.

2. 왼쪽 탐색 창에서 스택을 선택합니다.
3. 스택 목록에서 sam-app(또는 생성한 스택의 이름)을 선택합니다.
4. 삭제를 선택합니다.

완료되면 스택의 상태가 DELETE_COMPLETE로 변경됩니다.

또는 다음 AWS CLI 명령을 실행하여 AWS CloudFormation 스택을 삭제할 수 있습니다.

```
aws cloudformation delete-stack --stack-name sam-app --region region
```

삭제된 스택 확인

AWS CloudFormation 스택을 삭제하는 두 가지 방법 모두에서 https://console.aws.amazon.com/cloudformation로 이동하여 왼쪽 탐색 창에서 스택을 선택한 다음 검색 텍스트 상자 오른쪽에 있는 드롭다운에서 삭제됨을 선택하면 스택이 삭제되었는지 확인할 수 있습니다. 삭제된 스택 목록에 스택 이름 sam-app(또는 생성한 스택의 이름)이 표시되어야 합니다.

Step Functions를 사용하여 Activity 상태 시스템 만들기

이 자습서에서는 Java 및 AWS Step Functions을 사용하여 활동 기반 상태 머신을 생성하는 방법을 소개합니다. 활동을 사용하면 상태 시스템의 다른 위치에서 실행되는 작업자 코드를 제어할 수 있습니다. 개요는 Step Functions 작동 방식의 활동 단원을 참조하십시오.

이 튜토리얼을 완료하려면 다음이 필요합니다.

- Java용 SDK. 이 자습서의 예제 활동은 를 사용하여 AWS SDK for Java 통신하는 Java 응용 프로그램입니다 AWS.
- AWS 환경 또는 표준 AWS 구성 파일의 자격 증명 자세한 내용은 AWS SDK for Java 개발자 안내서의 AWS 자격 증명 설정을 참조하십시오.

주제

- 1단계: 활동 생성
- 2단계: 상태 시스템 만들기
- 3단계: 작업자 구현
- 4단계: 상태 시스템 실행
- 5단계: 작업자 실행 및 중지

1단계: 활동 생성

Step Functions가 만들려는 작업자(프로그램)의 활동을 인식하게 해야 합니다. Step Functions는 활동 ID를 설정하는 Amazon 리소스 이름(ARN)으로 응답합니다. 이 자격 증명을 사용하면 상태 머신 및 작업자 사이에서 전달되는 정보를 조정할 수 있습니다.

> **Important**
>
> 액티비티 태스크가 스테이트 머신과 동일한 AWS 계정에 속하는지 확인하세요.

1. Step Functions 콘솔 왼쪽에 있는 탐색 창에서 활동을 선택합니다.
2. 활동 생성을 선택합니다.
3. 활동 이름(예: *get-greeting*)을 입력한 다음 활동 생성을 선택합니다.
4. 활동 작업이 생성되면 다음 예제와 같이 해당 ARN을 적어 둡니다.

```
arn:aws:states:us-east-1:123456789012:activity:get-greeting
```

2단계: 상태 시스템 만들기

활동 호출 시기 및 작업자가 주요 작업을 수행하고 결과를 수집하고 이를 반환해야 하는 시기를 결정하는 상태 머신을 생성합니다. 상태 시스템을 만들려면 Workflow Studio의 코드 편집기를 사용합니다.

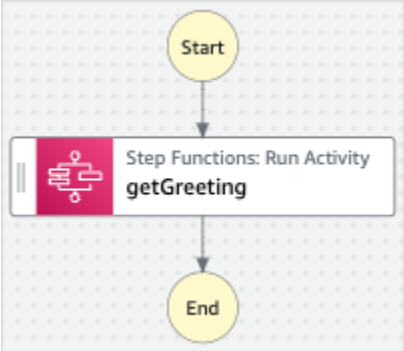
1. Step Functions 콘솔 왼쪽에 있는 탐색 창에서 상태 시스템을 선택합니다.
2. 상태 시스템 페이지에서 상태 시스템 생성을 선택합니다.
3. 템플릿 선택 대화 상자에서 공백을 선택합니다.
4. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
5. 이 자습서에서는 상태 시스템의 Amazon States Language(ASL) 정의를 코드 편집기에서 작성합니다. 이렇게 하려면 코드를 선택합니다.
6. 기존 보일러플레이트 코드를 제거하고 다음 코드를 붙여넣습니다. 이 코드의 예제 ARN을 `Resource` 필드에서 이전에 만든 활동 작업의 ARN으로 바꿔야 합니다.

```
{
  "Comment": "An example using a Task state.",
  "StartAt": "getGreeting",
```

```
  "Version": "1.0",
  "TimeoutSeconds": 300,
  "States":
  {
    "getGreeting": {
      "Type": "Task",
      "Resource": "arn:aws:states:us-east-1:123456789012:activity:get-greeting",
      "End": true
    }
  }
}
```

이 예제는 Amazon States Language(ASL)를 사용하는 상태 시스템에 대한 설명입니다. 이 예에서는 `getGreeting`라는 하나의 `Task` 상태를 정의합니다. 자세한 내용은 상태 머신 구조를 참조하십시오.

7. 그래프 시각화 창에서 추가한 ASL 정의의 워크플로 그래프가 다음 그래프와 비슷한지 확인합니다.



8. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택하십시오 MyStateMachine. 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 튜토리얼에서는 이름 **`ActivityStateMachine`**를 입력합니다.

9. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 상태 머신 설정의 모든 기본 선택 항목을 그대로 둡니다.

   상태 시스템에 대한 올바른 권한을 사용하여 이전에 IAM 역할을 만들었고 이를 사용하려면 권한에서 기존 역할 선택을 선택한 다음 목록에서 역할을 선택합니다. 또는 역할 ARN 입력을 선택한 다음 IAM 역할에 대한 ARN을 제공합니다.

10. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

3단계: 작업자 구현

작업자를 생성합니다. 작업자는 다음을 수행하는 프로그램입니다.

- `GetActivityTask` API 작업을 사용하여 활동에 대한 Step Functions 폴링
- 코드를 사용하여 활동 작업 수행(예: 다음 코드의 `getGreeting()` 메서드).
- `SendTaskSuccess`, `SendTaskFailure` 및 `SendTaskHeartbeat` API 작업을 사용하여 결과 반환.

> **Note**
>
> 활동 작업자의 보다 완벽한 예제는 Ruby 활동 작업자 예제를 참조하십시오. 이 예제는 사용자가 활동 작업자에 대한 참조로 사용할 수 있는 모범 사례를 기반으로 한 구현을 제공합니다. 이 코드는 폴러 및 활동 작업자용 스레드 수를 구성할 수 있는 소비자-생산자 패턴을 구현합니다.

작업자를 구현하려면

1. `GreeterActivities.java`이라는 이름의 파일을 만듭니다.
2. 파일에 다음 코드를 추가합니다.

```
import com.amazonaws.ClientConfiguration;
import com.amazonaws.auth.EnvironmentVariableCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.stepfunctions.AWSStepFunctions;
import com.amazonaws.services.stepfunctions.AWSStepFunctionsClientBuilder;
import com.amazonaws.services.stepfunctions.model.GetActivityTaskRequest;
import com.amazonaws.services.stepfunctions.model.GetActivityTaskResult;
```

```
import com.amazonaws.services.stepfunctions.model.SendTaskFailureRequest;
import com.amazonaws.services.stepfunctions.model.SendTaskSuccessRequest;
import com.amazonaws.util.json.Jackson;
import com.fasterxml.jackson.databind.JsonNode;
import java.util.concurrent.TimeUnit;


public class GreeterActivities {

    public String getGreeting(String who) throws Exception {
        return "{\"Hello\": \"" + who + "\"}";
    }

    public static void main(final String[] args) throws Exception {
        GreeterActivities greeterActivities = new GreeterActivities();
        ClientConfiguration clientConfiguration = new ClientConfiguration();
        clientConfiguration.setSocketTimeout((int)TimeUnit.SECONDS.toMillis(70));

        AWSStepFunctions client = AWSStepFunctionsClientBuilder.standard()
                .withRegion(Regions.US_EAST_1)
                .withCredentials(new EnvironmentVariableCredentialsProvider())
                .withClientConfiguration(clientConfiguration)
                .build();

        while (true) {
            GetActivityTaskResult getActivityTaskResult =
                    client.getActivityTask(
                            new
 GetActivityTaskRequest().withActivityArn(ACTIVITY_ARN));

            if (getActivityTaskResult.getTaskToken() != null) {
                try {
                    JsonNode json =
 Jackson.jsonNodeOf(getActivityTaskResult.getInput());
                    String greetingResult =

 greeterActivities.getGreeting(json.get("who").textValue());
                    client.sendTaskSuccess(
                            new SendTaskSuccessRequest().withOutput(

 greetingResult).withTaskToken(getActivityTaskResult.getTaskToken()));
                } catch (Exception e) {
                    client.sendTaskFailure(new
 SendTaskFailureRequest().withTaskToken(
```

```
                            getActivityTaskResult.getTaskToken()));
                }
            } else {
                Thread.sleep(1000);
            }
        }
    }
}
```

> **Note**
>
> 이 예제의 `EnvironmentVariableCredentialsProvider` 클래스는 `AWS_ACCESS_KEY_ID`(또는 `AWS_ACCESS_KEY`) 및 `AWS_SECRET_KEY`(또는 `AWS_SECRET_ACCESS_KEY`) 환경 변수가 설정되어 있다고 가정합니다. AWSCredentialsProvider팩토리에 필요한 자격 증명을 제공하는 방법에 대한 자세한 내용은 AWS SDK for Java 개발자 안내서의AWS SDK for Java API 참조 및 개발용 AWS 자격 증명 및 지역 설정을 참조하십시오.
>
> 기본적으로 AWS SDK는 모든 작업에 대해 서버로부터 데이터를 수신할 때까지 최대 50초 동안 대기합니다. `GetActivityTask` 작업은 사용 가능한 다음 작업에 대해 최대 60초 동안 대기하는 긴 폴링 작업입니다. `SocketTimeoutException`오류 수신을 SocketTimeout 방지하려면 70초로 설정합니다.

3. `GetActivityTaskRequest().withActivityArn()` 생성자의 파라미터 목록에서 `ACTIVITY_ARN` 값을 앞에서 생성한 활동 작업의 ARN으로 바꿉니다.

4단계: 상태 시스템 실행

상태 시스템 실행을 시작하면 작업자가 활동에 대해 Step Functions를 폴링하고 작업을 수행(제공한 입력 사용)하고 결과를 반환합니다.

1. ***ActivityStateMachine***페이지에서 실행 시작을 선택합니다.

   실행 시작 대화 상자가 표시됩니다.

2. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

b. 입력 상자에 다음 JSON 입력을 입력하여 워크플로를 실행합니다.

```
{
  "who": "AWS Step Functions"
}
```

c. 실행 시작을 선택합니다.

d. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

5단계: 작업자 실행 및 중지

작업자가 활동을 위해 상태 머신을 폴링하도록 하려면 작업자를 실행해야 합니다.

1. 명령줄에서 `GreeterActivities.java`를 생성한 디렉터리로 이동합니다.
2. AWS SDK를 사용하려면 `lib` 및 `third-party` 디렉터리의 전체 경로를 빌드 파일의 종속 항목과 Java에 추가하세요. `CLASSPATH` 자세한 내용은 AWS SDK for Java 개발자 안내서의 SDK 다운로드 및 추출을 참조하세요.
3. 파일을 컴파일합니다.

```
$ javac GreeterActivities.java
```

4. 파일을 실행합니다.

```
$ java GreeterActivities
```

5. Step Functions 콘솔에서 실행 세부 정보 페이지로 이동합니다.
6. 실행이 완료되면 실행 결과를 검사합니다.
7. 작업자를 중지합니다.

Lambda를 사용하여 루프를 반복하세요

이 자습서에서는 상태 머신 및 AWS Lambda 함수를 사용하여 특정 횟수만큼 루프를 반복하는 설계 패턴을 구현합니다.

상태 머신에서 루프 횟수를 추적해야 하는 경우 언제든 이 디자인 패턴을 사용하십시오. 이것을 실행하면 대형 작업 또는 장시간 실행을 작은 덩어리로 나누거나 정해진 수의 이벤트 후에 실행을 종료할 수 있습니다. 비슷한 구현을 사용하여 장기 실행 실행을 주기적으로 종료하고 다시 시작하여 AWS Step Functions AWS Lambda, 또는 기타 서비스에 대한 서비스 할당량을 초과하지 않도록 할 수 있습니다. AWS

시작하기 전에 Lambda를 사용하는 Step Functions 상태 시스템 만들기 자습서를 살펴보고 Lambda와 Step Functions를 함께 사용하는 방법을 숙지해야 합니다.

1단계: 계산을 반복하는 Lambda 함수 만들기

Lambda 함수를 사용하면 상태 시스템에서 루프의 반복 수를 추적할 수 있습니다. 다음 Lambda 함수는 `count`, `index` 및 `step`에 대한 입력값을 수신합니다. 그런 다음 이들 값을 업데이트된 `index` 및 `continue`라는 부울 값과 함께 반환합니다. Lambda 함수는 `index`이 `count` 미만인 경우에 `continue`를 `true`로 설정합니다.

그러면 상태 머신이 `continue`가 `true`일 경우 일부 애플리케이션 로직을 실행하고 `false`일 경우 종료되는 `Choice` 상태를 구현합니다.

Lambda 함수를 생성하려면

1. Lambda 콘솔에 로그인한 다음 함수 생성을 선택합니다.
2. 함수 생성 페이지에서 처음부터 새로 작성을 선택합니다.
3. 기본 정보 섹션에서 다음과 같이 Lambda 함수를 구성합니다.

   a. [함수 이름]에 `Iterator`을 입력합니다.

   b. 런타임에서 Node.js 를 선택합니다.

c. 기본 실행 역할 변경에서 기본 Lambda 권한을 사용하여 새 역할 생성을 선택합니다.

d. 함수 생성을 선택합니다.

4. Lambda 함수의 다음 코드를 코드 소스에 복사합니다.

```
export const handler = function (event, context, callback) {
  let index = event.iterator.index
  let step = event.iterator.step
  let count = event.iterator.count

  index = index + step

  callback(null, {
    index,
    step,
    count,
    continue: index < count
  })
}
```

이 코드는 count, index 및 step에 대한 입력값을 수신합니다. 이 코드는 index를 step 값만큼 증가시키고 이들 값과 부울 continue을 반환합니다. index가 count보다 작으면 continue 값이 true입니다.

5. 배포를 선택합니다.

2단계: Lambda 함수 테스트

숫자 값으로 Lambda 함수를 실행하여 작동하는지 확인합니다. 반복을 모방하는 Lambda 함수에 입력 값을 제공할 수 있습니다.

Lambda 함수 테스트하기

1. 테스트를 선택합니다.
2. 테스트 이벤트 구성 대화 상자에서 이벤트 이름 상자에 TestIterator를 입력합니다.
3. 예제 데이터를 다음으로 바꿉니다.

```
{
  "Comment": "Test my Iterator function",
  "iterator": {
```

```
    "count": 10,
    "index": 5,
    "step": 1
  }
}
```

이러한 값은 반복 도중에 상태 머신에서 얻게 되는 내용처럼 보입니다. Lambda 함수는 인덱스를 증가시키고 인덱스가 다음보다 작을 때 값을 `true` 반환합니다`continue`. `count` 이 테스트에서는 인덱스가 이미 5까지 증가한 상태입니다. 테스트는 로 증가하고 `index` 로 설정됩니다. 6 `continue` `true`

4. 생성을 선택합니다.
5. Lambda 함수를 테스트하려면 [Test] 를 선택합니다.

   테스트 결과는 실행 결과 탭에 표시됩니다.
6. 실행 결과 탭을 선택하여 출력을 확인합니다.

```
{
  "index": 6,
  "step": 1,
  "count": 10,
  "continue": true
}
```

> **Note**
>
> `index`로 9 설정하고 다시 테스트하면 `index` 증분은 로`10`, 그리고 `continue` 계속 증가할 것입니다. `false`

3단계: 상태 머신 생성

> **Lambda 콘솔을 종료하기 전에...**
>
> Lambda 함수 ARN을 복사합니다. 메모에 붙여넣습니다. 다음 단계에서 이 정보를 사용할 것입니다.

다음으로 다음과 같은 상태를 가진 스테이트 머신을 생성합니다.

- `ConfigureCount`— `countindex`, 및 의 기본값을 설정합니다`step`.
- `Iterator`— 이전에 생성한 Lambda 함수를 참조하여 에 구성된 값을 전달합니다. `ConfigureCount`
- `IsCountReached`— 함수에서 반환된 값을 기반으로 루프를 계속하거나 `Done` 상태로 진행하는 선택 상태입니다. `Iterator`
- `ExampleWork`— 수행해야 하는 작업을 위한 스텁입니다. 이 예시에서는 워크플로에 `Pass` 상태가 있지만 실제 솔루션에서는 a를 사용하는 경우가 많습니다. `Task`
- `Done`— 워크플로의 종료 상태.

콘솔에서 상태 머신을 만들려면:

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.

   > **Important**
   >
   > 상태 머신은 Lambda AWS 함수와 동일한 계정 및 지역에 있어야 합니다.

2. 빈 템플릿을 선택합니다.
3. 코드 창에서 상태 머신을 정의하는 다음 JSON을 붙여넣습니다.

   Amazon States Language에 대한 자세한 내용은 상태 시스템 구조를 참조하세요.

```
{
    "Comment": "Iterator State Machine Example",
    "StartAt": "ConfigureCount",
    "States": {

        "ConfigureCount": {
            "Type": "Pass",
            "Result": {
                "count": 10,
                "index": 0,
                "step": 1
            },
            "ResultPath": "$.iterator",
            "Next": "Iterator"
        },
        "Iterator": {
```

```
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Iterate",
            "ResultPath": "$.iterator",
            "Next": "IsCountReached"
        },
        "IsCountReached": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.iterator.continue",
                    "BooleanEquals": true,
                    "Next": "ExampleWork"
                }
            ],
            "Default": "Done"
        },
        "ExampleWork": {
            "Comment": "Your application logic, to run a specific number of times",
            "Type": "Pass",
            "Result": {
              "success": true
            },
            "ResultPath": "$.result",
            "Next": "Iterator"
        },
        "Done": {
            "Type": "Pass",
            "End": true

        }
    }
}
```

4. `Iterator` `Resource`필드를 이전에 생성한 `Iterator` Lambda 함수의 ARN으로 교체하십시오.
5. Config를 선택하고 상태 머신의 이름 (예:) 을 입력합니다. *`IterateCount`*

> **Note**
>
> 상태 머신, 실행 및 활동 태스크의 이름은 80자를 초과하면 안 됩니다. 이러한 이름은 계정 및 AWS 지역별로 고유해야 하며 다음 내용을 포함하지 않아야 합니다.
>
> - 공백

- 와일드카드 문자 (? *)
- 괄호 문자(< > { } [ ])
- 특수 문자 (" # % \ ^ | ~ ` $ & , ; : /)
- 제어 문자(\\u0000 - \\u001f 또는 \\u007f - \\u009f).

상태 시스템이 Express 유형이면 상태 시스템 실행 여러 개에 같은 이름을 제공할 수 있습니다. Step Functions는 여러 실행 이름이 같더라도 Express 상태 시스템 실행마다 고유한 실행 ARN을 생성합니다.
Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

6. 유형에는 기본값인 표준을 그대로 사용합니다. 권한에서 새 역할 생성을 선택합니다.
7. [Create] 를 선택한 다음 역할 생성을 확인합니다.

4단계: 새로운 실행 시작

상태 머신을 생성했으면 실행을 시작할 수 있습니다.

1. IterateCount페이지에서 실행 시작을 선택합니다.
2. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

3. 실행 시작을 선택합니다.

   상태 머신의 신규 실행이 시작되면서 작동 중인 실행이 나타납니다.
   이터레이터 상태가 파란색으로 표시되어 진행 상태를 나타내는 스테이트 머신 그래프 뷰입니다.

실행은 단계적으로 증가하며 Lambda 함수를 사용하여 횟수를 추적합니다. 각 반복에서는 상태 머신의 `ExampleWork` 상태에서 참조한 예시 작업을 수행합니다.

카운트가 상태 머신의 `ConfigureCount` 상태에서 지정된 횟수에 도달하면 실행이 반복을 중지하고 종료합니다.
상태 머신 그래프 보기로, 이터레이터 상태와 완료 상태가 녹색으로 표시되어 둘 다 성공했음을 나타냅니다.

장기 실행 워크플로 실행을 새 실행으로 계속하기

AWS Step Functions은 기간과 단계 수가 한정적인 워크플로우를 실행하기 위해 설계되었습니다. 실행의 최대 기간은 1년이며 최대 25,000개의 이벤트가 있습니다(할당량 참조).

장기 실행의 경우 실행 이벤트 내역의 하드 할당량 25,000개 항목에 도달하지 않으려면 상태 시스템의 `Task` 상태에서 직접 새 워크플로 실행을 시작하는 것이 좋습니다. 이렇게 하면 워크플로를 더 작은 상태 시스템으로 나누고 새 실행에서 진행 중인 작업을 계속할 수 있습니다. 이러한 워크플로 실행을 시작하려면 `Task` 상태에서 `StartExecution` API 작업을 직접적으로 호출하고 필요한 파라미터를 전달합니다.

또는 Lambda 함수를 사용하여 상태 시스템의 새 실행을 시작해 진행 중인 작업을 여러 워크플로 실행으로 분할하는 패턴을 구현할 수도 있습니다.

이 자습서에서는 서비스 할당량을 초과하지 않고 워크플로 실행을 계속할 수 있는 두 가지 방식을 모두 보여줍니다.

주제

- Step Functions API 작업을 사용하여 새 실행 계속하기(권장)
- Lambda 함수를 사용하여 새 실행 계속하기

Step Functions API 작업을 사용하여 새 실행 계속하기(권장)

Step Functions는 자체 API를 통합 서비스로 직접적으로 호출하여 워크플로 실행을 시작할 수 있습니다. 장기 실행의 서비스 할당량이 초과되지 않도록 하려면 이 방식을 사용하는 것이 좋습니다.

1단계: 장기 실행 상태 시스템 만들기

다른 상태 시스템의 `Task` 상태에서 시작하려는 장기 실행 상태 시스템을 만듭니다. 이 자습서에서는 Lambda 함수를 사용하는 상태 시스템을 사용합니다.

> **Note**
>
> 나중에 사용할 수 있도록 이 상태 시스템의 이름과 Amazon 리소스 이름을 텍스트 파일에 복사해야 합니다.

2단계: Step Functions API 작업을 직접적으로 호출할 상태 시스템 만들기

`Task` 상태에서 워크플로 실행 시작하기

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
4. Actions 탭에서 StartExecutionAPI 작업을 드래그하여 여기로 Drag first state라는 레이블이 붙은 빈 상태에 놓습니다.
5. StartExecution상태를 선택하고 구성 탭에서 디자인 모드 다음을 수행하십시오.

   a. 상태 이름을 **`Start nested execution`**로 변경합니다.

   b. 통합 유형의 드롭다운 목록에서 AWS SDK - new를 선택합니다.

   c. API 파라미터에서 다음을 수행합니다.

      i. `StateMachineArn`의 경우 샘플 Amazon 리소스 이름을 상태 시스템 ARN으로 바꿉니다. 예를 들어 Lambda를 사용하는 상태 시스템의 ARN을 입력합니다.

      ii. `Input` 노드의 경우 기존 자리 표시자 텍스트를 다음 값으로 바꿉니다.

      ```
      "Comment": "Starting workflow execution using a Step Functions API action"
      ```

      iii. API 파라미터에 입력한 내용이 다음과 비슷한지 확인합니다.

      ```
      {
        "StateMachineArn": "arn:aws:states:us-
      east-2:123456789012:stateMachine:LambdaStateMachine",
        "Input": {
          "Comment": "Starting workflow execution using a Step Functions API
       action",
          "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
        }
      ```

6. (선택 사항) Inspector 패널에서 정의를 선택하여 자동으로 생성된 워크플로의 Amazon States Language(ASL) 정의를 봅니다.

   > **Tip**
   >
   > Workflow Studio의 코드 편집기에서 ASL 정의를 볼 수도 있습니다. 코드 편집기에서 워크플로의 ASL 정의를 편집할 수도 있습니다.

7. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택합니다 MyStateMachine. 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 튜토리얼에서는 이름 **`ParentStateMachine`**를 입력합니다.

8. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 상태 머신 설정의 모든 기본 선택 항목을 그대로 둡니다.

   상태 시스템에 대한 올바른 권한을 사용하여 이전에 IAM 역할을 만들었고 이를 사용하려면 권한에서 기존 역할 선택을 선택한 다음 목록에서 역할을 선택합니다. 또는 역할 ARN 입력을 선택한 다음 IAM 역할에 대한 ARN을 제공합니다.

9. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

   > **Note**
   >
   > Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

3단계: IAM 정책 업데이트

상태 시스템에 Lambda 함수를 사용하는 상태 시스템의 실행을 시작할 수 있는 권한이 있는지 확인하려면 인라인 정책을 상태 시스템의 IAM 역할에 연결해야 합니다. 자세한 내용은 IAM 사용 설명서의 인라인 정책 임베딩을 참조하세요.

1. ParentStateMachine페이지에서 IAM 역할 ARN을 선택하여 상태 머신의 IAM 역할 페이지로 이동합니다.
2. 다른 상태 시스템의 실행을 시작할 수 있도록 하려면 의 ParentStateMachineIAM 역할에 적절한 권한을 할당하십시오. 권한을 할당하려면 다음을 수행합니다.

   a. IAM 역할 페이지에서 권한 추가를 선택한 다음 인라인 정책 생성을 선택합니다.

   b. 정책 생성 페이지에서 JSON 탭을 선택합니다.

   c. 기존 정책을 다음 정책으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [
              "states:StartExecution"
          ],
          "Resource": [
              "arn:aws:states:us-
east-2:123456789012:stateMachine:LambdaStateMachine"
          ]
      }
  ]
}
```

   d. 정책 검토를 선택합니다.

   e. 정책 이름을 지정한 다음 정책 생성을 선택합니다.

4단계: 상태 시스템 실행

상태 시스템 실행은 워크플로를 실행하여 작업을 수행하는 인스턴스입니다.

1. ParentStateMachine페이지에서 실행 시작을 선택합니다.

   실행 시작 대화 상자가 표시됩니다.

2. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

b. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

c. 실행 시작을 선택합니다.

d. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

3. LambdaStateMachine페이지를 열고 에 의해 트리거된 새 실행을 확인하십시오. ParentStateMachine

Lambda 함수를 사용하여 새 실행 계속하기

현재 실행을 종료하기 전에 Lambda 함수를 사용하는 상태 시스템을 만들어 새로운 실행을 시작할 수 있습니다. 이 방식을 사용하여 새 실행에서 진행 중인 작업을 계속하면 대규모 작업을 더 작은 워크플로로 분리할 수 있는 상태 시스템을 사용하거나 무한정 실행되는 상태 시스템을 사용할 수 있습니다.

이 자습서는 워크플로를 수정하는 외부 Lambda 함수 사용 개념을 기반으로 합니다. 이 개념은 Lambda를 사용하여 루프를 반복하세요 자습서에서 설명되었습니다. 같은 Lambda 함수(`Iterator`)를 사용하여 특정 횟수 동안 루프를 반복합니다. 또한 워크플로의 새 실행을 시작하고 새 실행을 시작할 때마다 카운트를 감소시키는 또 다른 Lambda 함수를 만듭니다. 입력에서 실행 수를 설정하면 이 상태 머신은 지정된 횟수만큼 실행을 종료했다가 다시 시작합니다.

여기서 생성하는 상태 머신은 다음 상태를 구현합니다.

State	용도
`ConfigureCount`	작업 반복을 단계적으로 진행하기 위해 `Iterator` Lambda 함수에서 사용하는 `count`, `index` 및 `step` 값을 구성하는 `Pass` 상태입니다.
`Iterator`	`Task` Lambda 함수를 참조하는 `Iterator` 상태입니다.
`IsCountReached`	`Iterator` 함수의 부울 값을 사용하여 상태 시스템이 예제 작업을 계속할지 또는 `ShouldRestart` 상태로 이동할지를 결정하는 `Choice` 상태입니다.
`ExampleWork`	실제 구현에서 작업을 수행하는 `Task` 상태를 나타내는 `Pass` 상태입니다.
`ShouldRestart`	`executionCount` 값을 사용하여 실행 하나를 종료하고 다른 실행을 시작할지 또는 종료만 할지를 결정하는 `Choice` 상태입니다.
`Restart`	Lambda 함수를 사용하여 상태 시스템의 새 실행을 시작하는 `Task` 상태입니다. `Iterator` 함수와 마찬가지로 이 함수도 카운트를 감소시킵니다. `Restart` 상태는 감소된 카운트 값을 새 실행의 입력으로 전달합니다.

필수 조건

시작하기 전에 Lambda를 사용하는 Step Functions 상태 시스템 만들기 자습서를 살펴보고 Lambda와 Step Functions를 함께 사용하는 방법에 익숙해야 합니다.

주제

- 1단계: 계산을 반복하는 Lambda 함수 만들기
- 2단계: 새로운 Step Functions 실행을 시작하는 재시작 Lambda 함수 만들기
- 3단계: 상태 시스템 만들기
- 4단계: IAM 정책 업데이트
- 5단계: 상태 시스템 실행

1단계: 계산을 반복하는 Lambda 함수 만들기

> **Note**
>
> Lambda를 사용하여 루프를 반복하세요 자습서를 완료했으면 이 단계를 건너뛰고 해당 Lambda 함수를 사용할 수 있습니다.

이 섹션과 Lambda를 사용하여 루프를 반복하세요 자습서에서는 Lambda 함수를 사용하여 카운트(예: 상태 시스템에서 루프 반복 횟수)를 추적하는 방법을 보여줍니다.

다음 Lambda 함수는 `count`, `index` 및 `step`에 대한 입력값을 수신합니다. 업데이트된 `index` 및 불 방식인 `continue`로 이런 값을 반환합니다. Lambda 함수는 `index`이 `count` 미만인 경우에 `continue`를 `true`로 설정합니다.

그러면 상태 머신은 `continue`가 `true`이면 일부 애플리케이션 로직을 실행하고 `continue`가 `false`이면 `ShouldRestart`로 진행하는 `Choice` 상태를 구현합니다.

반복 Lambda 함수 만들기

1. Lambda 콘솔을 열고 함수 생성을 선택합니다.
2. 함수 생성 페이지에서 처음부터 새로 작성을 선택합니다.
3. 기본 정보 섹션에서 다음과 같이 Lambda 함수를 구성합니다.

   a. [함수 이름]에 `Iterator`을 입력합니다.

   b. 런타임에는 Node.js 16.x를 선택합니다.

   c. 페이지에 있는 모든 기본 선택 항목을 그대로 두고 함수 생성을 선택합니다.

   Lambda 함수가 생성되면 페이지 오른쪽 상단 모서리에 있는 Amazon 리소스 이름(ARN)을 기록해 둡니다. 예를 들면 다음과 같습니다.

   ```
   arn:aws:lambda:us-east-1:123456789012:function:Iterator
   ```

4. Lambda 함수의 다음 코드를 Lambda 콘솔에 있는 ***#####*** 페이지의 코드 소스 섹션에 복사합니다.

```
exports.handler = function iterator (event, context, callback) {
  let index = event.iterator.index;
  let step = event.iterator.step;
  let count = event.iterator.count;
```

```
  index = index + step;

  callback(null, {
    index,
    step,
    count,
    continue: index < count
  })
}
```

이 코드는 count, index 및 step에 대한 입력값을 수신합니다. 이 코드는 index를 step 값만큼 증가시키고 이들 값과 continue 부울 값을 반환합니다. index가 count보다 작으면 continue 값이 true입니다.

5. 배포를 선택하여 코드를 배포합니다.

반복 Lambda 함수 테스트

Iterate 함수가 작동하는지 확인하려면 숫자 값으로 이 함수를 실행합니다. Lambda 함수에 입력값을 제공하여 반복을 모방하면 특정 입력값으로 어떤 출력을 얻는지 확인할 수 있습니다.

Lambda 함수 테스트하기

1. 테스트 이벤트 구성 대화 상자에서 새로운 테스트 이벤트 생성을 선택한 다음 이벤트 이름에 TestIterator를 입력합니다.
2. 예제 데이터를 다음으로 바꿉니다.

```
{
  "Comment": "Test my Iterator function",
  "iterator": {
    "count": 10,
    "index": 5,
    "step": 1
  }
}
```

이러한 값은 반복 도중에 상태 머신에서 얻게 되는 내용처럼 보입니다. Lambda 함수는 인덱스를 증가시키고 continue를 true로 반환합니다. 인덱스가 count보다 작지 않으면 이 함수는 continue를 false로 반환합니다. 이 테스트에서는 인덱스가 이미 5까지 증가한 상태입니다. 결과는 index를 6으로 증가시키고 continue를 true로 설정해야 합니다.

3. 생성을 선택하세요.
4. Lambda 콘솔의 ***Iterator*** 페이지에 나열되어 TestIterator있는지 확인한 다음 Test를 선택합니다.

   테스트의 결과는 창 위쪽에 표시됩니다. [Details]를 선택하고 결과를 검토합니다.

   ```
   {
     "index": 6,
     "step": 1,
     "count": 10,
     "continue": true
   }
   ```

   > **Note**
   >
   > 이 테스트에서 `index`를 `9`로 설정할 경우 `index`는 `10`으로 증가하고 `continue`는 `false`가 됩니다.

2단계: 새로운 Step Functions 실행을 시작하는 재시작 Lambda 함수 만들기

1. Lambda 콘솔을 열고 함수 생성을 선택합니다.
2. 함수 생성 페이지에서 처음부터 새로 작성을 선택합니다.
3. 기본 정보 섹션에서 다음과 같이 Lambda 함수를 구성합니다.

   a. [함수 이름]에 `Restart`을 입력합니다.

   b. 런타임에는 Node.js 16.x를 선택합니다.
4. 페이지에 있는 모든 기본 선택 항목을 그대로 두고 함수 생성을 선택합니다.

   Lambda 함수가 생성되면 페이지 오른쪽 상단 모서리에 있는 Amazon 리소스 이름(ARN)을 기록해 둡니다. 예를 들면 다음과 같습니다.

   ```
   arn:aws:lambda:us-east-1:123456789012:function:Iterator
   ```
5. Lambda 함수의 다음 코드를 Lambda 콘솔에 있는 ***###*** 페이지의 코드 소스 섹션에 복사합니다.

   다음 코드는 실행 수 카운트를 감소시키고 감소된 값을 포함하여 상태 머신의 새 실행을 시작합니다.

```
var aws = require('aws-sdk');
var sfn = new aws.StepFunctions();

exports.restart = function(event, context, callback) {

  let StateMachineArn = event.restart.StateMachineArn;
  event.restart.executionCount -= 1;
  event = JSON.stringify(event);

  let params = {
      input: event,
      stateMachineArn: StateMachineArn
  };

  sfn.startExecution(params, function(err, data) {
      if (err) callback(err);
      else callback(null,event);
  });

}
```

6. 배포를 선택하여 코드를 배포합니다.

3단계: 상태 시스템 만들기

Lambda 함수 2개를 만들었습니다. 이제 상태 시스템을 만듭니다. 이 상태 머신에서 `ShouldRestart` 및 `Restart` 상태는 작업을 여러 실행으로 분리하는 방법입니다.

Example ShouldRestart 선택 상태

다음 발췌에서는 `ShouldRestart`[Choice](#) 상태를 보여줍니다. 이 상태는 실행을 다시 시작할지 여부를 결정합니다.

```
"ShouldRestart": {
"Type": "Choice",
"Choices": [
  {
    "Variable": "$.restart.executionCount",
    "NumericGreaterThan": 1,
    "Next": "Restart"
  }
```

```
],
```

`$.restart.executionCount` 값은 초기 실행이 입력에 포함됩니다. 이 값은 `Restart` 함수를 호출할 때마다 하나씩 감소된 다음, 각 후속 실행에 대한 입력으로 배치됩니다.

Example 다시 시작 작업 상태

다음 발췌에서는 `RestartTask` 상태를 보여줍니다. 이 상태는 앞에서 만든 Lambda 함수를 사용하여 실행을 다시 시작하고 카운트를 감소시켜서 시작할 남은 실행 수를 추적합니다.

```
"Restart": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Restart",
  "Next": "Done"
},
```

상태 시스템을 생성하려면

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.

   > **Important**
   >
   > 스테이트 머신이 1단계 및 2단계에서 앞서 생성한 Lambda 함수와 동일한 AWS 계정 및 리전에 속하는지 확인하십시오.

2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
4. 이 자습서에서는 코드 편집기에서 상태 시스템의 Amazon States Language(ASL) 정의를 작성합니다. 이렇게 하려면 코드를 선택합니다.
5. 기존 보일러플레이트 코드를 제거하고 다음 코드를 붙여넣습니다. 이 코드의 ARN을 만든 Lambda 함수의 ARN으로 교체해야 합니다.

```
{
    "Comment": "Continue-as-new State Machine Example",
    "StartAt": "ConfigureCount",
    "States": {
        "ConfigureCount": {
            "Type": "Pass",
            "Result": {
                "count": 100,
```

```
                "index": -1,
                "step": 1
            },
            "ResultPath": "$.iterator",
            "Next": "Iterator"
        },
        "Iterator": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Iterator",
            "ResultPath": "$.iterator",
            "Next": "IsCountReached"
        },
        "IsCountReached": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.iterator.continue",
                    "BooleanEquals": true,
                    "Next": "ExampleWork"
                }
            ],
            "Default": "ShouldRestart"
        },
        "ExampleWork": {
            "Comment": "Your application logic, to run a specific number of times",
            "Type": "Pass",
            "Result": {
              "success": true
            },
            "ResultPath": "$.result",
            "Next": "Iterator"
        },
        "ShouldRestart": {
          "Type": "Choice",
          "Choices": [
            {
              "Variable": "$.restart.executionCount",
              "NumericGreaterThan": 0,
              "Next": "Restart"
            }
          ],
          "Default": "Done"
        },
        "Restart": {
```

```
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:Restart",
            "Next": "Done"
        },
        "Done": {
            "Type": "Pass",
            "End": true
        }
    }
}
```

6. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택하십시오. MyStateMachine 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 튜토리얼에서는 이름 **`ContinueAsNew`**를 입력합니다.

7. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 상태 머신 설정의 모든 기본 선택 항목을 그대로 둡니다.

   상태 시스템에 대한 올바른 권한을 사용하여 이전에 IAM 역할을 만들었고 이를 사용하려면 권한에서 기존 역할 선택을 선택한 다음 목록에서 역할을 선택합니다. 또는 역할 ARN 입력을 선택한 다음 IAM 역할에 대한 ARN을 제공합니다.

8. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

   > **Note**
   >
   > Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

9. 이 상태 시스템의 Amazon 리소스 이름(ARN)을 텍스트 파일에 저장합니다. Lambda 함수에 새로운 Step Functions 실행을 시작할 수 있는 권한을 제공하면서 ARN을 제공해야 합니다.

4단계: IAM 정책 업데이트

Lambda 함수에 새로운 Step Functions 실행을 시작할 수 있는 권한이 있는지 확인하려면 인라인 정책을 `Restart` Lambda 함수에 사용하는 IAM 역할에 연결합니다. 자세한 내용은 IAM 사용 설명서의 인라인 정책 임베딩을 참조하세요.

> **Note**
>
> `ContinueAsNew` 상태 머신의 ARN을 참조하도록 이전 예제의 `Resource` 줄을 업데이트할 수 있습니다. 이렇게 하면 해당 특정 상태 머신의 실행만 시작할 수 있도록 정책이 제한됩니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
     {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": [
             "states:StartExecution"
         ],
         "Resource": "arn:aws:states:us-east-2:123456789012stateMachine:ContinueAsNew"
     }
 ]
}
```

5단계: 상태 시스템 실행

실행을 시작하려면 상태 머신의 ARN 및 새 실행을 시작할 횟수의 `executionCount`이 포함된 입력을 제공합니다.

1. ContinueAsNew페이지에서 실행 시작을 선택합니다.

   실행 시작 대화 상자가 표시됩니다.

2. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

b. 입력 상자에 다음 JSON 입력을 입력하여 워크플로를 실행합니다.

```
{
  "restart": {
    "StateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:ContinueAsNew",
    "executionCount": 4
  }
}
```

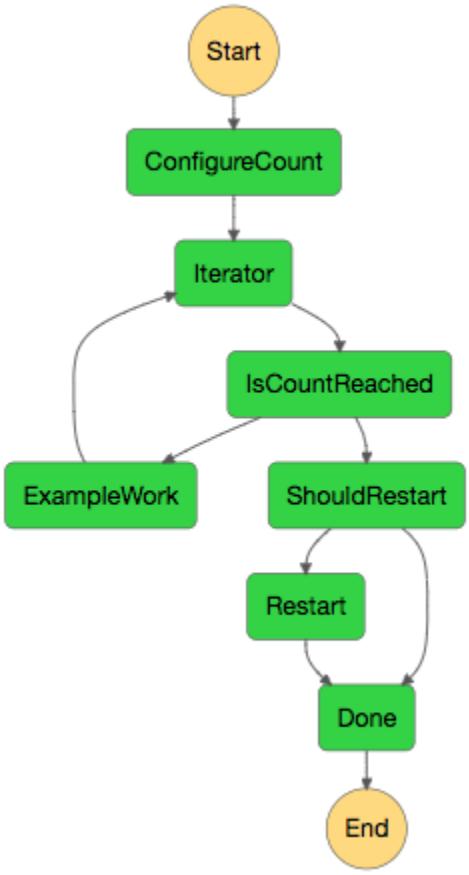
c. `ContinueAsNew` 상태 머신의 ARN으로 `StateMachineArn` 필드를 업데이트합니다.

d. 실행 시작을 선택합니다.

e. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

그래프 보기에는 실행 4개 중 첫 번째가 표시됩니다. 이 실행은 완료되기 전에 `Restart` 상태를 전달하고 새 실행을 시작합니다.



이 실행이 완료되면 다음에 시작되는 실행을 볼 수 있습니다. 상단의 ContinueAsNew링크를 선택하면 실행 목록을 볼 수 있습니다. 최근에 종료된 실행과 `Restart` Lambda 함수가 시작된 진행 중인 실행이 모두 표시됩니다.

모든 실행이 완료되면 네 개의 성공적인 실행이 목록에 표시됩니다. 첫 번째로 시작된 실행에는 선택한 이름이 표시되고, 후속 실행에는 생성된 이름이 표시됩니다.

예제 인간 승인 프로젝트 배포

이 자습서에서는 AWS Step Functions 실행이 작업 중에 일시 중지하고 사용자가 이메일에 응답할 때까지 대기할 수 있는 인간 승인 프로젝트를 배포하는 방법을 알아봅니다. 사용자가 작업을 진행하도록 승인하면 워크플로우는 다음 상태로 진행합니다.

이 자습서에 포함된 AWS CloudFormation 스택을 배포하면 다음을 포함하여 필요한 모든 리소스가 생성됩니다.

- Amazon API Gateway 리소스
- 모든 함수 AWS Lambda
- AWS Step Functions 스테이트 머신
- Amazon Simple Notification Service 이메일 주제
- 관련 AWS Identity and Access Management 역할 및 권한

> **Note**
>
> AWS CloudFormation 스택을 생성할 때 액세스할 수 있는 유효한 이메일 주소를 제공해야 합니다.

자세한 내용은 AWS CloudFormation 사용 설명서의 CloudFormation 템플릿 및 *AWS::StepFunctions::StateMachine* 리소스 사용을 참조하십시오.

주제

1단계: AWS CloudFormation 템플릿 만들기

1. AWS CloudFormation 템플릿 소스 코드 단원에서 예제 코드를 복사합니다.

2. AWS CloudFormation 템플릿의 소스를 로컬 컴퓨터의 파일에 붙여넣습니다.

   이 예제의 경우 파일은 `human-approval.yaml`입니다.

2단계: 스택 만들기

1. AWS CloudFormation 콘솔에 로그인합니다.
2. 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.
3. Create stack(스택 생성) 페이지에서 다음을 수행합니다.

   a. 사전 조건 - 템플릿 준비 섹션에서 준비된 템플릿을 선택합니다.

   b. 템플릿 지정 섹션에서 템플릿 파일 업로드를 선택한 다음 파일 선택을 선택하여 이전에 만든 템플릿 소스 코드가 포함된 `human-approval.yaml` 파일을 업로드합니다.

4. Next(다음)를 선택합니다.
5. 스택 세부 정보 지정 페이지에서 다음 작업을 수행합니다.

   a. 스택 이름에 스택 이름을 입력합니다.

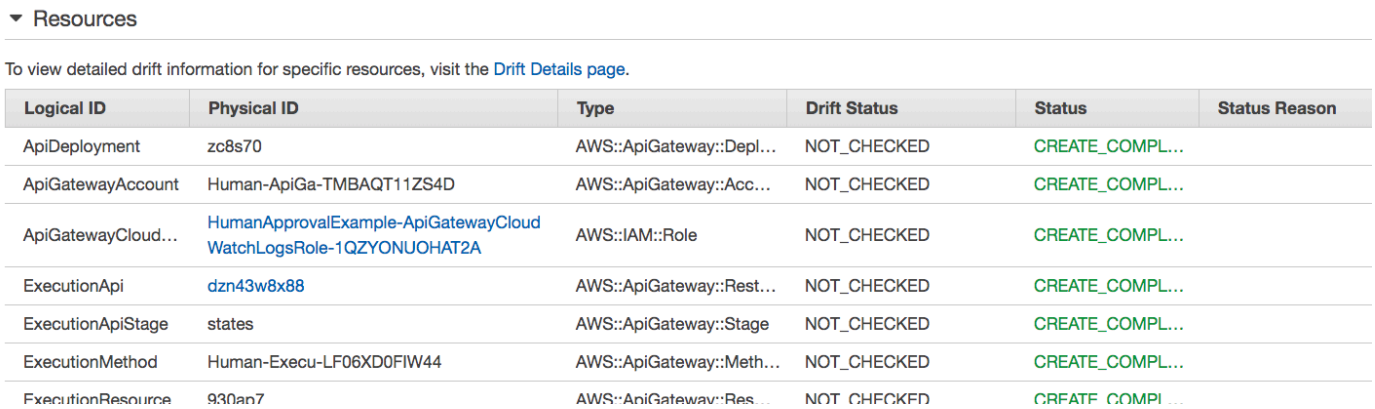
   b. 파라미터에 유효한 이메일 주소를 입력합니다. 이 이메일 주소를 사용하여 Amazon SNS 주제를 구독합니다.

6. 다음을 선택한 다음 다음을 다시 선택합니다.

7. 검토 페이지에서 IAM 리소스를 생성할 AWS CloudFormation 수 있음을 확인합니다를 선택한 다음 [Create] 를 선택합니다.

   AWS CloudFormation 스택 생성을 시작하고 CREATE_IN_PROGRESS 상태를 표시합니다. 프로세스가 완료되면 CREATE_COMPLETE 상태가 표시됩니다 AWS CloudFormation .

8. (선택 사항) 스택에 리소스를 표시하려면 스택을 선택하고 [Resources] 탭을 선택합니다.



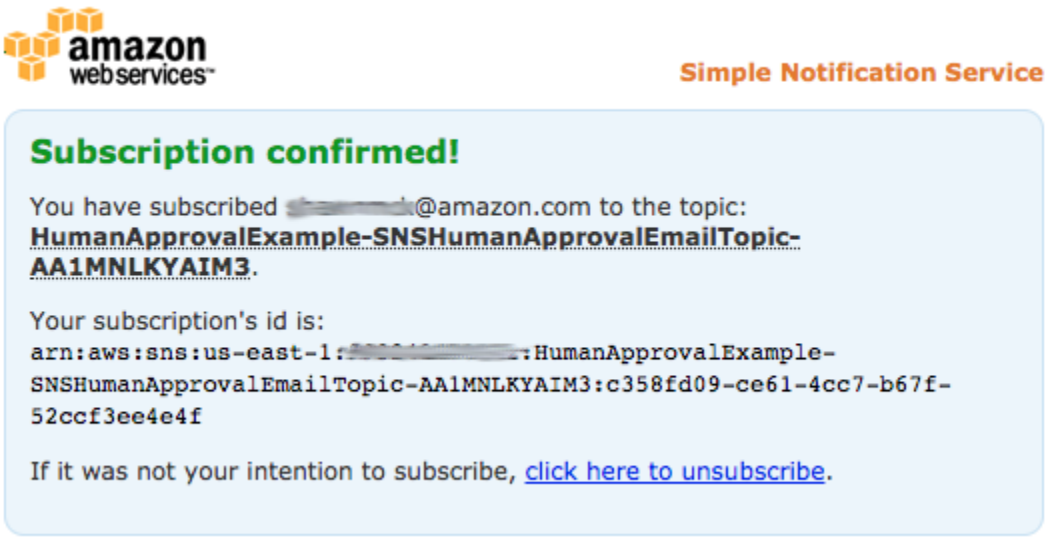
3단계: Amazon SNS 구독 승인

Amazon SNS 주제가 생성되면 구독 확인을 요청하는 이메일이 전송됩니다.

1. 스택을 생성할 때 제공한 이메일 계정을 엽니다. AWS CloudFormation
2. no-reply@sns.amazonaws.com에서 보낸 AWS Notification - Subscription Confirmation 메시지를 엽니다.

   이메일에는 Amazon SNS 주제에 대한 Amazon 리소스 이름과 확인 링크가 있습니다.

3. Confirm subscription(구독 확인) 링크를 선택합니다.



4단계: 상태 시스템 실행

1. HumanApprovalLambdaStateMachine페이지에서 실행 시작을 선택합니다.

   실행 시작 대화 상자가 표시됩니다.

2. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

   > **Note**
   >
   > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   b. 입력 상자에 다음 JSON 입력을 입력하여 워크플로를 실행합니다.

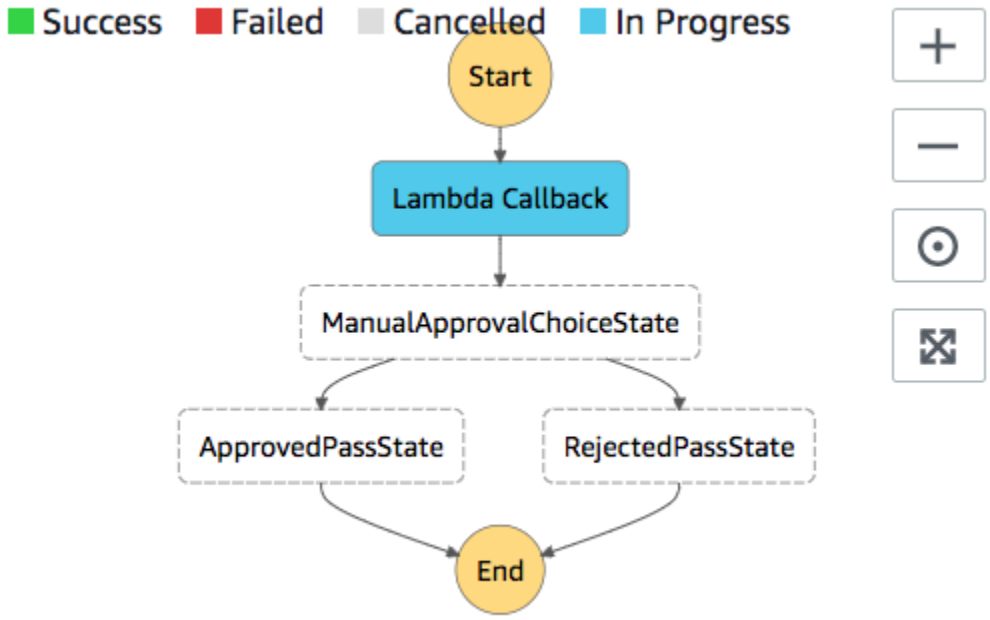
   ```
   {
       "Comment": "Testing the human approval tutorial."
   }
   ```

   c. 실행 시작을 선택합니다.

Lambda Callback 태스크에서 ApprovalTest상태 머신 실행이 시작되고 일시 중지됩니다.

d. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

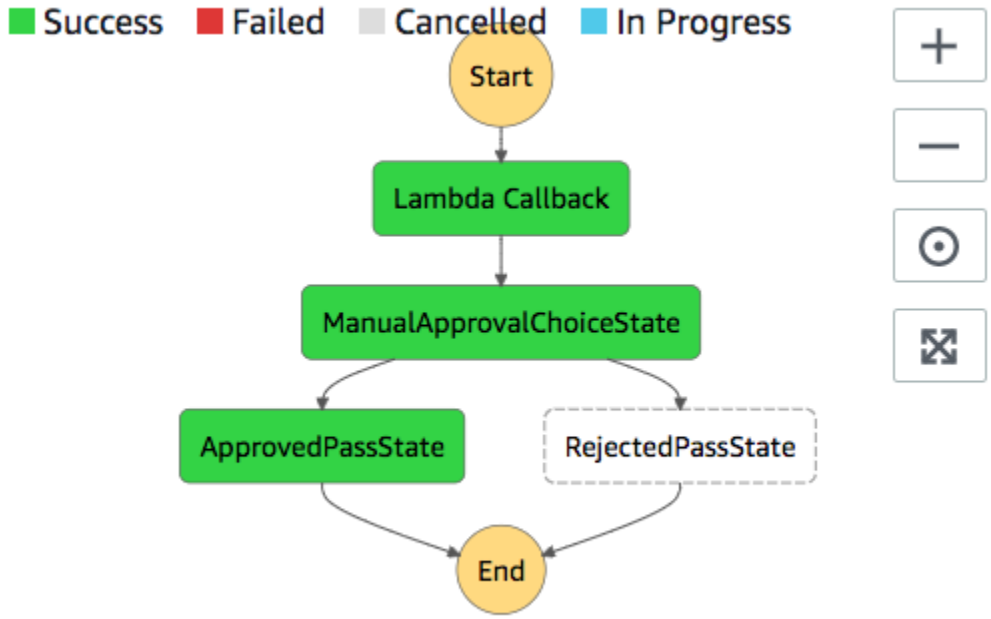


3. 앞서 Amazon SNS 주제에 사용한 이메일 계정에서 승인 필요 양식이라는 제목의 메시지를 엽니다 AWS Step Functions.

   이 메시지에는 Approve(승인) 및 Reject(거부)에 대한 별도의 URL이 포함됩니다.

4. Approve(승인) URL을 선택합니다.

   선택을 기반으로 워크플로우가 계속됩니다.



AWS CloudFormation 템플릿 소스 코드

이 AWS CloudFormation 템플릿을 사용하여 사람의 승인 프로세스 워크플로의 예를 배포할 수 있습니다.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "AWS Step Functions Human based task example. It sends an email with an
 HTTP URL for approval."
Parameters:
  Email:
    Type: String
    AllowedPattern: "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$"
    ConstraintDescription: Must be a valid email address.
Resources:
  # Begin API Gateway Resources
  ExecutionApi:
    Type: "AWS::ApiGateway::RestApi"
    Properties:
      Name: "Human approval endpoint"
      Description: "HTTP Endpoint backed by API Gateway and Lambda"
      FailOnWarnings: true

  ExecutionResource:
```

```
    Type: 'AWS::ApiGateway::Resource'
    Properties:
      RestApiId: !Ref ExecutionApi
      ParentId: !GetAtt "ExecutionApi.RootResourceId"
      PathPart: execution

  ExecutionMethod:
    Type: "AWS::ApiGateway::Method"
    Properties:
      AuthorizationType: NONE
      HttpMethod: GET
      Integration:
        Type: AWS
        IntegrationHttpMethod: POST
        Uri: !Sub "arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/
${LambdaApprovalFunction.Arn}/invocations"
        IntegrationResponses:
          - StatusCode: 302
            ResponseParameters:
              method.response.header.Location:
 "integration.response.body.headers.Location"
        RequestTemplates:
          application/json: |
            {
              "body" : $input.json('$'),
              "headers": {
                #foreach($header in $input.params().header.keySet())
                "$header":
 "$util.escapeJavaScript($input.params().header.get($header))"
 #if($foreach.hasNext),#end

                #end
              },
              "method": "$context.httpMethod",
              "params": {
                #foreach($param in $input.params().path.keySet())
                "$param": "$util.escapeJavaScript($input.params().path.get($param))"
 #if($foreach.hasNext),#end

                #end
              },
              "query": {
                #foreach($queryParam in $input.params().querystring.keySet())
```

```
                "$queryParam":
"$util.escapeJavaScript($input.params().querystring.get($queryParam))"
#if($foreach.hasNext),#end

                #end
              }
            }
      ResourceId: !Ref ExecutionResource
      RestApiId: !Ref ExecutionApi
      MethodResponses:
        - StatusCode: 302
          ResponseParameters:
            method.response.header.Location: true

  ApiGatewayAccount:
    Type: 'AWS::ApiGateway::Account'
    Properties:
      CloudWatchRoleArn: !GetAtt "ApiGatewayCloudWatchLogsRole.Arn"

  ApiGatewayCloudWatchLogsRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - apigateway.amazonaws.com
            Action:
              - 'sts:AssumeRole'
      Policies:
        - PolicyName: ApiGatewayLogsPolicy
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              - Effect: Allow
                Action:
                  - "logs:*"
                Resource: !Sub "arn:${AWS::Partition}:logs:*:*:*"

  ExecutionApiStage:
    DependsOn:
      - ApiGatewayAccount
```

```
    Type: 'AWS::ApiGateway::Stage'
    Properties:
      DeploymentId: !Ref ApiDeployment
      MethodSettings:
        - DataTraceEnabled: true
          HttpMethod: '*'
          LoggingLevel: INFO
          ResourcePath: /*
      RestApiId: !Ref ExecutionApi
      StageName: states

  ApiDeployment:
    Type: "AWS::ApiGateway::Deployment"
    DependsOn:
      - ExecutionMethod
    Properties:
      RestApiId: !Ref ExecutionApi
      StageName: DummyStage
  # End API Gateway Resources

  # Begin
  # Lambda that will be invoked by API Gateway
  LambdaApprovalFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      Code:
        ZipFile:
          Fn::Sub: |
            const { SFN: StepFunctions } = require("@aws-sdk/client-sfn");
            var redirectToStepFunctions = function(lambdaArn, statemachineName,
executionName, callback) {
              const lambdaArnTokens = lambdaArn.split(":");
              const partition = lambdaArnTokens[1];
              const region = lambdaArnTokens[3];
              const accountId = lambdaArnTokens[4];

              console.log("partition=" + partition);
              console.log("region=" + region);
              console.log("accountId=" + accountId);

              const executionArn = "arn:" + partition + ":states:" + region + ":" +
accountId + ":execution:" + statemachineName + ":" + executionName;
              console.log("executionArn=" + executionArn);
```

```
              const url = "https://console.aws.amazon.com/states/home?region=" + region
+ "#/executions/details/" + executionArn;
              callback(null, {
                  statusCode: 302,
                  headers: {
                    Location: url
                  }
              });
            };

            exports.handler = (event, context, callback) => {
              console.log('Event= ' + JSON.stringify(event));
              const action = event.query.action;
              const taskToken = event.query.taskToken;
              const statemachineName = event.query.sm;
              const executionName = event.query.ex;

              const stepfunctions = new StepFunctions();

              var message = "";

              if (action === "approve") {
                message = { "Status": "Approved! Task approved by ${Email}" };
              } else if (action === "reject") {
                message = { "Status": "Rejected! Task rejected by ${Email}" };
              } else {
                console.error("Unrecognized action. Expected: approve, reject.");
                callback({"Status": "Failed to process the request. Unrecognized
Action."});
              }

              stepfunctions.sendTaskSuccess({
                output: JSON.stringify(message),
                taskToken: event.query.taskToken
              })
              .then(function(data) {
                redirectToStepFunctions(context.invokedFunctionArn, statemachineName,
executionName, callback);
              }).catch(function(err) {
                console.error(err, err.stack);
                callback(err);
              });
            }
      Description: Lambda function that callback to AWS Step Functions
```

```
      FunctionName: LambdaApprovalFunction
      Handler: index.handler
      Role: !GetAtt "LambdaApiGatewayIAMRole.Arn"
      Runtime: nodejs18.x

  LambdaApiGatewayInvoke:
    Type: "AWS::Lambda::Permission"
    Properties:
      Action: "lambda:InvokeFunction"
      FunctionName: !GetAtt "LambdaApprovalFunction.Arn"
      Principal: "apigateway.amazonaws.com"
      SourceArn: !Sub "arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:
${ExecutionApi}/*"

  LambdaApiGatewayIAMRole:
    Type: "AWS::IAM::Role"
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Action:
              - "sts:AssumeRole"
            Effect: "Allow"
            Principal:
              Service:
                - "lambda.amazonaws.com"
      Policies:
        - PolicyName: CloudWatchLogsPolicy
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - "logs:*"
                Resource: !Sub "arn:${AWS::Partition}:logs:*:*:*"
        - PolicyName: StepFunctionsPolicy
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - "states:SendTaskFailure"
                  - "states:SendTaskSuccess"
                Resource: "*"
  # End Lambda that will be invoked by API Gateway
```

```
  # Begin state machine that publishes to Lambda and sends an email with the link for
 approval
  HumanApprovalLambdaStateMachine:
    Type: AWS::StepFunctions::StateMachine
    Properties:
      RoleArn: !GetAtt LambdaStateMachineExecutionRole.Arn
      DefinitionString:
        Fn::Sub: |
          {
              "StartAt": "Lambda Callback",
              "TimeoutSeconds": 3600,
              "States": {
                  "Lambda Callback": {
                      "Type": "Task",
                      "Resource": "arn:
${AWS::Partition}:states:::lambda:invoke.waitForTaskToken",
                      "Parameters": {
                        "FunctionName": "${LambdaHumanApprovalSendEmailFunction.Arn}",
                        "Payload": {
                          "ExecutionContext.$": "$$",
                          "APIGatewayEndpoint": "https://${ExecutionApi}.execute-api.
${AWS::Region}.amazonaws.com/states"
                        }
                      },
                      "Next": "ManualApprovalChoiceState"
                  },
                  "ManualApprovalChoiceState": {
                    "Type": "Choice",
                    "Choices": [
                      {
                        "Variable": "$.Status",
                        "StringEquals": "Approved! Task approved by ${Email}",
                        "Next": "ApprovedPassState"
                      },
                      {
                        "Variable": "$.Status",
                        "StringEquals": "Rejected! Task rejected by ${Email}",
                        "Next": "RejectedPassState"
                      }
                    ]
                  },
                  "ApprovedPassState": {
                    "Type": "Pass",
                    "End": true
```

```
                },
                "RejectedPassState": {
                  "Type": "Pass",
                  "End": true
                }
            }
        }

  SNSHumanApprovalEmailTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        -
          Endpoint: !Sub ${Email}
          Protocol: email

  LambdaHumanApprovalSendEmailFunction:
    Type: "AWS::Lambda::Function"
    Properties:
      Handler: "index.lambda_handler"
      Role: !GetAtt LambdaSendEmailExecutionRole.Arn
      Runtime: "nodejs18.x"
      Timeout: "25"
      Code:
        ZipFile:
          Fn::Sub: |
            console.log('Loading function');
            const { SNS } = require("@aws-sdk/client-sns");
            exports.lambda_handler = (event, context, callback) => {
                console.log('event= ' + JSON.stringify(event));
                console.log('context= ' + JSON.stringify(context));

                const executionContext = event.ExecutionContext;
                console.log('executionContext= ' + executionContext);

                const executionName = executionContext.Execution.Name;
                console.log('executionName= ' + executionName);

                const statemachineName = executionContext.StateMachine.Name;
                console.log('statemachineName= ' + statemachineName);

                const taskToken = executionContext.Task.Token;
                console.log('taskToken= ' + taskToken);
```

```
                const apigwEndpint = event.APIGatewayEndpoint;
                console.log('apigwEndpint = ' + apigwEndpint)

                const approveEndpoint = apigwEndpint + "/execution?
action=approve&ex=" + executionName + "&sm=" + statemachineName + "&taskToken=" +
 encodeURIComponent(taskToken);
                console.log('approveEndpoint= ' + approveEndpoint);

                const rejectEndpoint = apigwEndpint + "/execution?
action=reject&ex=" + executionName + "&sm=" + statemachineName + "&taskToken=" +
 encodeURIComponent(taskToken);
                console.log('rejectEndpoint= ' + rejectEndpoint);

                const emailSnsTopic = "${SNSHumanApprovalEmailTopic}";
                console.log('emailSnsTopic= ' + emailSnsTopic);

                var emailMessage = 'Welcome! \n\n';
                emailMessage += 'This is an email requiring an approval for a step
 functions execution. \n\n'
                emailMessage += 'Please check the following information and click
 "Approve" link if you want to approve. \n\n'
                emailMessage += 'Execution Name -> ' + executionName + '\n\n'
                emailMessage += 'Approve ' + approveEndpoint + '\n\n'
                emailMessage += 'Reject ' + rejectEndpoint + '\n\n'
                emailMessage += 'Thanks for using Step functions!'

                const sns = new SNS();
                var params = {
                  Message: emailMessage,
                  Subject: "Required approval from AWS Step Functions",
                  TopicArn: emailSnsTopic
                };

                sns.publish(params)
                  .then(function(data) {
                    console.log("MessageID is " + data.MessageId);
                    callback(null);
                  }).catch(
                    function(err) {
                    console.error(err, err.stack);
                    callback(err);
                  });
            }
```

```
  LambdaStateMachineExecutionRole:
    Type: "AWS::IAM::Role"
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: states.amazonaws.com
            Action: "sts:AssumeRole"
      Policies:
        - PolicyName: InvokeCallbackLambda
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - "lambda:InvokeFunction"
                Resource:
                  - !Sub "${LambdaHumanApprovalSendEmailFunction.Arn}"

  LambdaSendEmailExecutionRole:
    Type: "AWS::IAM::Role"
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: "sts:AssumeRole"
      Policies:
        - PolicyName: CloudWatchLogsPolicy
          PolicyDocument:
            Statement:
              - Effect: Allow
                Action:
                  - "logs:CreateLogGroup"
                  - "logs:CreateLogStream"
                  - "logs:PutLogEvents"
                Resource: !Sub "arn:${AWS::Partition}:logs:*:*:*"
        - PolicyName: SNSSendEmailPolicy
          PolicyDocument:
            Statement:
              - Effect: Allow
```

```
                Action:
                  - "SNS:Publish"
                Resource:
                  - !Sub "${SNSHumanApprovalEmailTopic}"

# End state machine that publishes to Lambda and sends an email with the link for
 approval
Outputs:
  ApiGatewayInvokeURL:
    Value: !Sub "https://${ExecutionApi}.execute-api.${AWS::Region}.amazonaws.com/
states"
  StateMachineHumanApprovalArn:
    Value: !Ref HumanApprovalLambdaStateMachine
```

Step Functions에서 X-Ray 트레이스 보기

이 자습서에서는 X-Ray를 사용하여 상태 시스템을 실행할 때 발생하는 오류를 추적하는 방법을 알아봅니다. AWS X-Ray를 사용하여 상태 시스템 구성 요소를 시각화하고 성능 병목 현상을 식별하며 오류가 발생한 요청 문제를 해결할 수 있습니다. 이 자습서에서는 오류를 무작위로 생성하는 여러 Lambda 함수를 만든 다음 X-Ray를 사용하여 추적 및 분석할 수 있습니다.

Lambda를 사용하는 Step Functions 상태 시스템 만들기 자습서는 Lambda 함수를 직접적으로 호출하는 상태 시스템을 만드는 과정을 안내합니다. 해당 자습서를 마쳤으면 2단계로 건너뛰고 이전에 만든 AWS Identity and Access Management (IAM) 역할을 사용합니다.

주제

- 1단계: Lambda에 대한 IAM 역할 만들기
- 2단계: Lambda 함수 생성
- 3단계: Lambda 함수를 2개 더 만들기
- 4단계: 상태 시스템 만들기
- 5단계: 상태 시스템 실행

1단계: Lambda에 대한 IAM 역할 만들기

둘 다 AWS Lambda 코드를 실행하고 AWS 리소스 (예: Amazon S3 버킷에 저장된 데이터) 에 액세스할 AWS Step Functions 수 있습니다. 보안 유지를 위해 Lambda 및 Step Functions에 이러한 리소스에 대한 액세스 권한을 부여해야 합니다.

Lambda는 Lambda 함수를 생성할 때 (IAM) 역할을 AWS Identity and Access Management 할당하도록 요구합니다. Step Functions에서 상태 머신을 생성할 때 IAM 역할을 할당해야 하는 것과 같습니다.

IAM 콘솔을 사용하여 서비스 연결 역할을 만들 수 있습니다.

역할을 만들려면(콘솔 사용)

1. https://console.aws.amazon.com/iam/ 에서 IAM 콘솔에 AWS Management Console 로그인하고 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택합니다. 그런 다음 역할 생성을 선택합니다.
3. AWS 서비스 역할 유형을 선택한 다음 Lambda를 선택합니다.
4. Lambda 사용 사례를 선택합니다. 사용 사례는 서비스에 필요한 신뢰 정책을 포함하기 위해 서비스에서 정합니다. 그런 다음 다음: 권한을 선택합니다.
5. 역할(예: `AWSLambdaBasicExecutionRole`)에 연결할 하나 이상의 권한 정책을 선택합니다. AWS Lambda 권한 모델을 참조하십시오.

   역할에 부여하려는 권한을 할당하는 정책 옆의 확인란을 선택한 후 다음: 검토를 선택합니다.
6. 역할 이름을 입력합니다.
7. (선택 사항) [Role description]에서 새로운 서비스 연결 역할에 대한 설명을 편집합니다.
8. 역할을 검토한 다음 역할 생성을 선택합니다.

2단계: Lambda 함수 생성

Lambda 함수는 무작위로 오류나 제한 시간을 발생시켜 X-Ray에서 볼 수 있는 예제 데이터를 생성합니다.

> **Important**
>
> Lambda 함수가 상태 머신과 AWS 동일한 계정 AWS 및 지역에 속하는지 확인하십시오.

1. Lambda 콘솔을 열고 함수 생성을 선택합니다.
2. [Create function] 섹션에서 [Author from scratch]를 선택합니다.
3. 기본 정보 섹션에서 Lambda 함수를 구성합니다.

   a. [함수 이름]에 `TestFunction1`을 입력합니다.

   b. 런타임에서 Node.js 18.x를 선택합니다.

c. [역할]에서 [기존 역할 선택]을 선택합니다.

d. 기존 역할에 이전에 만든 Lambda 역할을 선택합니다.

> **Note**
>
> 만든 IAM 역할이 목록에 표시되지 않으면 역할이 Lambda에 전파될 때까지 몇 분 더 기다려야 할 수 있습니다.

e. 함수 생성을 선택합니다.

Lambda 함수가 생성되면 페이지 오른쪽 상단 모서리에 있는 Amazon 리소스 이름(ARN)을 기록해 둡니다. 예:

```
arn:aws:lambda:us-east-1:123456789012:function:TestFunction1
```

4. ***Lambda ### ## ### 1 #### ## ## ### #####. TestFunction***

```
function getRandomSeconds(max) {
    return Math.floor(Math.random() * Math.floor(max)) * 1000;
}
function sleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}
export const handler = async (event) => {
    if(getRandomSeconds(4) === 0) {
        throw new Error("Something went wrong!");
    }
    let wait_time = getRandomSeconds(5);
    await sleep(wait_time);
    return { 'response': true }
};
```

이 코드는 시간이 임의로 설정된 실패를 만듭니다. 이 실패는 상태 시스템에서 X-Ray 트레이스를 사용하여 보고 분석할 수 있는 예제 오류를 생성하는 데 사용됩니다.

5. 저장을 선택합니다.

3단계: Lambda 함수를 2개 더 만들기

Lambda 함수를 2개 더 만듭니다.

1. 2단계를 반복하여 Lambda 함수를 2개 더 만듭니다. 다음 함수의 경우 함수 이름에 `TestFunction2`를 입력합니다. 마지막 함수의 경우 함수 이름에 `TestFunction3`을 입력합니다.
2. Lambda 콘솔에서 이제 Lambda 함수 3개(`TestFunction1`, `TestFunction2` 및 `TestFunction3`)가 있는지 확인합니다.

4단계: 상태 시스템 만들기

이 단계에서는 Step Functions 콘솔을 사용하여 `Task` 상태 3개가 있는 상태 시스템을 만듭니다. 각 `Task` 상태는 Lambda 함수 3개 중 하나를 참조합니다.

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.

   > **Important**
   >
   > 스테이트 머신이 2단계 및 3단계 앞부분에서 생성한 Lambda 함수와 동일한 AWS 계정 및 리전에 속하는지 확인하십시오.

2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.
4. 이 자습서에서는 코드 편집기에서 상태 시스템의 Amazon States Language(ASL) 정의를 작성합니다. 이렇게 하려면 코드를 선택합니다.
5. 기존 보일러플레이트 코드를 제거하고 다음 코드를 붙여넣습니다. Task 상태 정의에서 예제 ARN을 생성한 Lambda 함수의 ARN으로 바꿔야 합니다.

```
{
  "StartAt": "CallTestFunction1",
  "States": {
    "CallTestFunction1": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:test-function1",
      "Catch": [
        {
          "ErrorEquals": [
            "States.TaskFailed"
          ],
          "Next": "AfterTaskFailed"
        }
      ],
```

```
      "Next": "CallTestFunction2"
    },
    "CallTestFunction2": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:test-function2",
      "Catch": [
        {
          "ErrorEquals": [
            "States.TaskFailed"
          ],
          "Next": "AfterTaskFailed"
        }
      ],
      "Next": "CallTestFunction3"
    },
    "CallTestFunction3": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:test-function3",
      "TimeoutSeconds": 5,
      "Catch": [
        {
          "ErrorEquals": [
            "States.Timeout"
          ],
          "Next": "AfterTimeout"
        },
        {
          "ErrorEquals": [
            "States.TaskFailed"
          ],
          "Next": "AfterTaskFailed"
        }
      ],
      "Next": "Succeed"
    },
    "Succeed": {
      "Type": "Succeed"
    },
    "AfterTimeout": {
      "Type": "Fail"
    },
    "AfterTaskFailed": {
      "Type": "Fail"
    }
```

```
  }
}
```

이 예제는 Amazon States Language를 사용하는 상태 시스템에 대한 설명입니다. `Task` 상태 3개(`CallTestFunction1`, `CallTestFunction2` 및 `CallTestFunction3`)를 정의합니다. 각 상태는 Lambda 함수 3개 중 하나를 직접적으로 호출합니다. 자세한 내용은 상태 머신 구조를 참조하십시오.

6. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택하십시오. MyStateMachine 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 튜토리얼에서는 이름 **`TraceFunctions`**를 입력합니다.

7. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   이 자습서의 경우 추가 구성에서 X-Ray 추적 활성화를 선택합니다. 상태 머신 설정의 다른 모든 기본 선택 항목을 그대로 둡니다.

   상태 시스템에 대한 올바른 권한을 사용하여 이전에 IAM 역할을 만들었고 이를 사용하려면 권한에서 기존 역할 선택을 선택한 다음 목록에서 역할을 선택합니다. 또는 역할 ARN 입력을 선택한 다음 IAM 역할에 대한 ARN을 제공합니다.

8. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

5단계: 상태 시스템 실행

상태 시스템 실행은 워크플로를 실행하여 작업을 수행하는 인스턴스입니다.

1. ***`TraceFunctions`***페이지에서 실행 시작을 선택합니다.

   [New execution] 페이지가 표시됩니다.

2. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

   > **Note**
   >
   > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

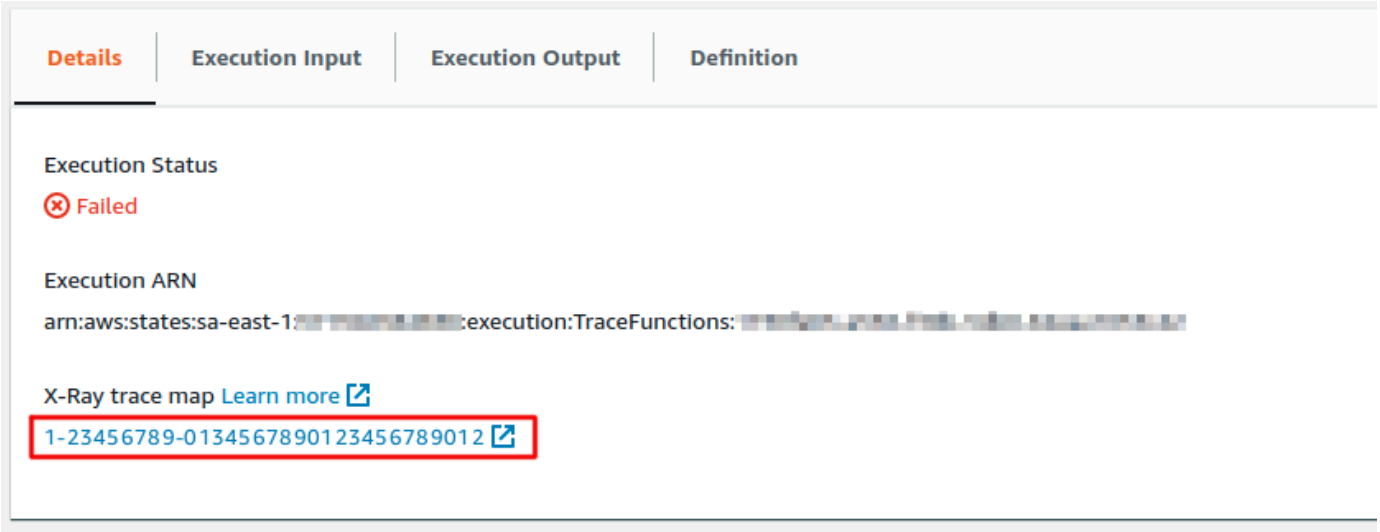
   b. 실행 시작을 선택합니다.

   c. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

      실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

      여러 번(최소 3번 이상) 실행합니다.

3. 실행이 완료되면 X-Ray 트레이스 맵 링크를 따릅니다. 실행이 아직 실행 중인 동안에도 트레이스를 볼 수 있지만 X-Ray 트레이스 맵을 보기 전에 실행 결과를 확인하는 것이 좋습니다.



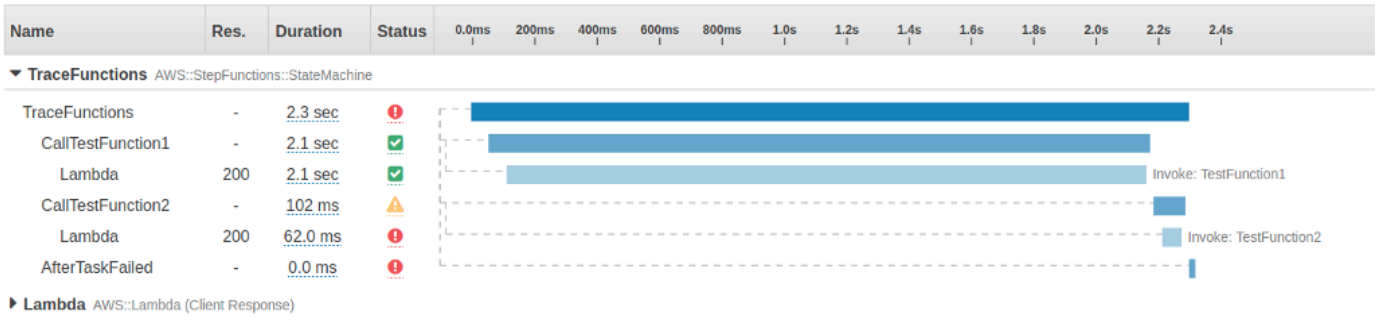
4. 서비스 맵을 보고 오류가 발생한 위치, 지연 시간이 긴 연결 또는 실패한 요청의 트레이스를 식별할 수 있습니다. 이 예시에서는 각 함수에 수신되는 트래픽 양을 확인할 수 있습니다. `TestFunction2`는 `TestFunction3`보다 자주 많이 직접적으로 호출되었고 `TestFunction1`은 `TestFunction2`보다 두 배 넘게 더 자주 직접적으로 호출되었습니다.

   이 서비스는 오류 및 장애 호출 성공 비율을 토대로 각 노드의 색상을 다르게 표시해 노드의 상태를 보여줍니다:

   - 녹색은 성공적인 호출을 의미합니다
   - 빨간색은 서버 장애를 의미합니다(500 시리즈 오류)
   - 노란색은 클라이언트 오류를 의미합니다(400 시리즈 오류)
   - 보라색은 병목 오류를 의미합니다(429 요청 과다)

   서비스 노드를 선택하여 해당 노드에 대한 요청을 보거나 두 노드 간의 엣지를 선택하여 해당 연결을 이동하는 요청을 볼 수 있습니다.

5. X-Ray 트레이스 맵을 보고 각 실행에서 발생하는 내용을 확인합니다. 타임라인 보기에 세그먼트 및 하위 세그먼트 계층 구조가 표시됩니다. 목록의 첫 번째 항목이 세그먼트인데, 단일 요청의 서비스에 의해 기록된 모든 데이터를 나타냅니다. 세그먼트 아래에는 하위 세그먼트가 있습니다. 이 예제에서는 Lambda 함수로 기록된 하위 세그먼트를 보여줍니다.



X-Ray 트레이스를 이해하고 Step Functions에서 X-Ray를 사용하는 방법에 대한 자세한 내용은 AWS X-Ray 및 Step Functions 섹션을 참조하세요.

AWS SDK 서비스 통합을 사용하여 Amazon S3 버킷 정보 수집

이 자습서에서는 Amazon Simple Storage Service와 AWS SDK 통합을 수행하는 방법을 보여줍니다. 이 자습서에서 만든 상태 시스템은 Amazon S3 버킷에 대한 정보를 수집한 다음 현재 리전의 각 버킷에 대한 버전 정보와 함께 버킷을 나열합니다.

주제

- 1단계: 상태 시스템 만들기
- 2단계: 필요한 IAM 역할 권한 추가
- 3단계: 표준 상태 시스템 실행
- 4단계: Express 상태 시스템 실행

1단계: 상태 시스템 만들기

Step Functions 콘솔을 사용하여 현재 계정과 리전의 모든 Amazon S3 버킷을 나열하는 `Task` 상태가 포함된 상태 시스템을 만듭니다. 그런 다음 `HeadBucket` API를 간접적으로 호출하는 다른 `Task` 상태를 추가하여 반환된 버킷이 현재 리전에서 액세스할 수 있는지 확인합니다. 버킷이 액세스할 수 없으면 `HeadBucket` API 직접 호출에서 `S3.S3Exception` 오류를 반환합니다. 이 예외를 포착하기 위한 `Catch` 블록과 폴백 상태로 `Pass` 상태가 포함됩니다.

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 템플릿 선택 대화 상자에서 공백을 선택합니다.
3. 선택을 선택하세요. 디자인 모드에서 Workflow Studio가 열립니다.

4. 이 자습서에서는 코드 편집기에서 상태 시스템의 Amazon States Language(ASL) 정의를 작성합니다. 이렇게 하려면 코드를 선택합니다.

5. 기존 보일러플레이트 코드를 제거하고 다음 상태 시스템 정의를 붙여넣습니다.

```
{
  "Comment": "A description of my state machine",
  "StartAt": "ListBuckets",
  "States": {
    "ListBuckets": {
      "Type": "Task",
      "Parameters": {},
      "Resource": "arn:aws:states:::aws-sdk:s3:listBuckets",
      "Next": "Map"
    },
    "Map": {
      "Type": "Map",
      "ItemsPath": "$.Buckets",
      "ItemProcessor": {
        "ProcessorConfig": {
          "Mode": "INLINE"
        },
        "StartAt": "HeadBucket",
        "States": {
          "HeadBucket": {
            "Type": "Task",
            "ResultPath": null,
            "Parameters": {
              "Bucket.$": "$.Name"
            },
            "Resource": "arn:aws:states:::aws-sdk:s3:headBucket",
            "Catch": [
              {
                "ErrorEquals": [
                  "S3.S3Exception"
                ],
                "ResultPath": null,
                "Next": "Pass"
              }
            ],
            "Next": "GetBucketVersioning"
          },
          "GetBucketVersioning": {
            "Type": "Task",
```

```
            "End": true,
            "Parameters": {
              "Bucket.$": "$.Name"
            },
            "ResultPath": "$.BucketVersioningInfo",
            "Resource": "arn:aws:states:::aws-sdk:s3:getBucketVersioning"
          },
          "Pass": {
            "Type": "Pass",
            "End": true,
            "Result": {
              "Status": "Unknown"
            },
            "ResultPath": "$.BucketVersioningInfo"
          }
        }
      },
      "End": true
    }
  }
}
```

6. 상태 시스템 이름을 지정합니다. 이렇게 하려면 기본 상태 머신 이름 옆에 있는 편집 아이콘을 선택합니다. MyStateMachine 그런 다음 상태 머신 구성에서 상태 머신 이름 상자에 이름을 지정합니다.

   이 튜토리얼에서는 이름 **Gather-S3-Bucket-Info-Standard**를 입력합니다.

7. (선택 사항) 상태 머신 구성에서 상태 시스템 유형 및 실행 역할과 같은 기타 워크플로 설정을 지정합니다.

   상태 머신 설정의 모든 기본 선택 항목을 그대로 둡니다.

   상태 시스템에 대한 올바른 권한을 사용하여 이전에 IAM 역할을 만들었고 이를 사용하려면 권한에서 기존 역할 선택을 선택한 다음 목록에서 역할을 선택합니다. 또는 역할 ARN 입력을 선택한 다음 IAM 역할에 대한 ARN을 제공합니다.

8. 역할 생성 확인 대화 상자에서 확인을 선택하여 계속합니다.

   역할 설정 보기를 선택하여 상태 머신 구성으로 돌아갈 수도 있습니다.

> **Note**
>
> Step Functions에서 만드는 IAM 역할을 삭제하면 나중에 Step Functions에서 이 역할을 다시 만들 수 없습니다. 마찬가지로, 역할을 수정하면(예: IAM 정책의 주요에서 Step Functions 제거) 나중에 Step Functions에서 해당 원본 설정을 복원할 수 없습니다.

2단계에서는 누락된 권한을 상태 시스템 역할에 추가합니다.

2단계: 필요한 IAM 역할 권한 추가

현재 리전의 Amazon S3 버킷에 대한 정보를 수집하려면 상태 시스템에 Amazon S3 버킷에 액세스하는 데 필요한 권한을 제공해야 합니다.

1. 상태 시스템 페이지에서 IAM 역할 ARN을 선택하여 상태 시스템 역할에 대한 역할 페이지를 엽니다.
2. 권한 추가를 선택하고 인라인 정책 생성을 선택합니다.
3. JSON 탭을 선택한 후 다음 권한을 JSON 편집기에 붙여넣습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketVersioning"
            ],
            "Resource": "*"
        }
    ]
}
```

4. 정책 검토를 선택합니다.
5. 정책 검토의 이름에 **s3-bucket-permissions**를 입력합니다.
6. 정책 생성(Create policy)을 선택합니다.

3단계: 표준 상태 시스템 실행

1. Gather-S3-Bucket-Info-Standard 페이지에서 실행 시작을 선택합니다.
2. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

   > **Note**
   >
   > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   b. 실행 시작을 선택합니다.

   c. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

4단계: Express 상태 시스템 실행

1. 1단계에서 제공한 상태 시스템 정의를 사용하여 Express 상태 시스템을 만듭니다. 2단계의 설명대로 필요한 IAM 역할 권한도 포함해야 합니다.

   > **Tip**
   >
   > 앞에서 만든 표준 시스템과 구별하려면 Express 상태 시스템 이름을 **`Gather-S3-Bucket-Info-Express`**로 지정합니다.

2. Gather-S3-Bucket-Info-Standard 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

a. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

b. 실행 시작을 선택합니다.

c. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

개발자 도구

다음 리소스에는 서버리스 워크플로 빌드 및 상태 시스템 사용에 대한 추가 정보가 포함되어 있습니다.

- AWS CDK
- AWS VS Code용 툴킷

다음 주제에는 상태 시스템을 생성, 테스트 및 디버깅하는 방법을 알려주는 정보가 포함되어 있습니다.

주제

- 개발 옵션
- AWS Step Functions 그리고 AWS SAM
- Application Composer에서 Workflow Studio 사용
- AWS CloudFormation를 사용하여 Step Functions용 Lambda 상태 시스템 만들기
- AWS CDK를 사용하여 Step Functions용 Lambda 상태 시스템 만들기
- 를 사용하여 동기식 익스프레스 스테이트 머신을 사용하여 API Gateway REST API 만들기 AWS CDK
- AWS Step Functions Python용 데이터 사이언스 SDK
- Terraform을 사용하여 상태 시스템 배포

개발 옵션

콘솔, SDK 또는 로컬 버전의 Step Functions를 테스트 및 개발용으로 사용하는 등 여러 가지 방법으로 AWS Step Functions 상태 머신을 구현할 수 있습니다.

주제

- Step Functions 콘솔
- AWS SDK
- 표준 및 Express 워크플로
- HTTPS 서비스 API
- 개발 환경

- 엔드포인트
- AWS CLI
- Step Functions Local
- AWS Toolkit for Visual Studio Code
- AWS Serverless Application Model 및 Step Functions
- TerraForm 및 Step Functions
- 정의 형식 지원

Step Functions 콘솔

Step Functions 콘솔을 사용하여 상태 시스템을 정의할 수 있습니다. 를 사용하여 작업에 코드를 제공함으로써 로컬 개발 환경을 사용하지 않고도 클라우드에서 복잡한 상태 머신을 작성할 수 있습니다. AWS Lambda 작성하면 Step Functions 콘솔을 사용하여 Amazon States Language로 상태 시스템을 정의할 수 있습니다.

Lambda 상태 시스템 생성 자습서에서는 이 기술을 사용하여 간단한 상태 시스템을 생성 및 실행하고 결과를 확인합니다.

데이터 흐름 시뮬레이터

Step Functions 콘솔에서 워크플로를 설계, 구현 및 디버깅할 수 있습니다. `JsonPath` 입력 및 출력 처리를 사용하여 워크플로의 데이터 흐름을 제어할 수도 있습니다. Step Functions 콘솔에서 데이터 흐름 시뮬레이터를 사용하여 상태 간에 정보가 이동하는 방식을 알아보고 데이터를 필터링하고 조작하는 방법을 이해합니다. 이 도구는 Step Functions에서 데이터를 처리하는 데 사용하는 각 필드(예: `InputPath`, `Parameters`, `ResultSelector`, `OutputPath` 및 `ResultPath`)를 시뮬레이션합니다.

자세한 정보는 데이터 흐름 시뮬레이터을 참조하세요.

AWS SDK

Step AWS Functions는 자바, .NET, 루비, PHP, 파이썬 (보토 3), Go JavaScript, C++용 SDK에서 지원됩니다. 이러한 SDK는 여러 프로그래밍 언어에서 Step Functions HTTPS API 작업을 사용할 수 있는 편리한 방법을 제공합니다.

이러한 SDK 라이브러리에 제공된 API 작업을 사용하여 상태 머신, 활동 또는 상태 머신 시작자를 개발할 수 있습니다. 이러한 라이브러리를 사용하여 가시성 작업에 액세스하여 고유한 Step Functions 모니터링 및 보고 도구를 개발할 수도 있습니다.

Step Functions를 다른 AWS 서비스와 함께 사용하려면 현재 Amazon Web Services용 AWS SDK 및 도구에 대한 참조 설명서를 참조하십시오.

> **Note**
>
> Step Functions는 HTTPS 엔드포인트만 지원합니다.

표준 및 Express 워크플로

새 상태 머신을 만들 때는 Type을 [표준] 또는 [Express] 중 하나로 선택해야 합니다. 두 경우 모두 Amazon States Language를 사용하여 상태 시스템을 정의합니다. 상태 머신 실행은 선택한 유형에 따라 다르게 작동합니다. 상태 시스템을 만든 후에는 선택한 유형을 변경할 수 없습니다.

자세한 정보는 CloudWatch Logs를 사용하여 로깅을 참조하세요.

HTTPS 서비스 API

Step Functions는 HTTPS 요청을 통해 액세스할 수 있는 서비스 작업을 제공합니다. 이러한 작업을 사용하여 Step Functions와 직접 통신하고 HTTPS를 통해 Step Functions와 통신할 수 있는 언어로 고유한 라이브러리를 개발할 수 있습니다.

서비스 API 작업을 사용하여 상태 머신, 작업자 또는 상태 머신 시작자를 개발할 수 있습니다. API 작업을 통해 가시성 작업에 액세스함으로써 사용자 고유의 모니터링 및 보고 도구를 개발할 수도 있습니다.

API 작업에 대한 자세한 내용은 AWS Step Functions API 참조를 확인하세요.

개발 환경

사용하려는 프로그래밍 언어와 호환되는 개발 환경을 설정해야 합니다.

예를 들어 Java를 사용하여 Step Functions용으로 개발하려면 각 개발 워크스테이션에 와 같은 Java 개발 환경을 설치해야 합니다. AWS SDK for Java Eclipse IDE for Java Developers를 사용하는 경우 AWS Toolkit for Eclipse도 설치해야 합니다. 이 Eclipse 플러그인은 AWS기반 개발에 유용한 기능을 추가합니다.

런타임 환경이 필요한 프로그래밍 언어의 경우 해당 프로세스를 실행할 컴퓨터마다 환경을 설정해야 합니다.

엔드포인트

지연 시간을 줄이고 요구 사항을 충족하는 위치에 데이터를 저장하기 위해 Step Functions는 다양한 AWS 지역에 엔드포인트를 제공합니다.

Step Functions의 각 엔드포인트는 완전히 독립적입니다. 즉, 상태 머신이나 활동이 생성된 리전 내에서만 존재합니다. 한 리전에서 생성한 상태 머신 및 활동은 다른 리전에서 생성한 상태 머신 및 활동과 데이터나 속성을 공유하지 않습니다. 예를 들어 서로 다른 두 리전에 `STATES-Flows-1` 상태 시스템을 등록할 수 있습니다. 한 리전의 `STATES-Flows-1` 상태 시스템은 데이터나 속성을 다른 리전의 `STATES-Flow-1` 상태 시스템과 공유하지 않습니다.

Step Functions 엔드포인트 목록은 AWS 일반 참조의 AWS Step Functions 리전 및 엔드포인트를 참조하세요.

AWS CLI

AWS Command Line Interface (AWS CLI) 에서 많은 Step Functions 기능에 액세스할 수 있습니다. AWS CLI 는 Step Functions 콘솔을 사용하는 대신 사용할 수 있으며, 경우에 따라 Step Functions API 작업을 사용하여 프로그래밍하는 방법도 있습니다. 예를 들어 AWS CLI 를 사용하여 상태 시스템을 만든 후 기존 상태 시스템을 나열할 수 있습니다.

AWS CLI 에서 Step Functions 명령을 사용하여 실행 시작 및 관리, 활동 폴링 및 작업 하트비트 기록 등의 작업을 수행할 수 있습니다. 전체 Step Functions 명령 목록, 사용 가능한 인수 설명 및 사용법을 보여주는 예제는 AWS CLI 명령 참조를 확인하세요.

AWS CLI 명령은 Amazon States 언어를 거의 따르므로 를 사용하여 Step Functions API 작업에 대해 알아볼 수 있습니다. AWS CLI 기존 API 지식을 사용하여 코드 프로토타입을 제작하거나 명령줄에서 Step Functions 작업을 수행할 수도 있습니다.

Step Functions Local

테스트 및 개발용으로 로컬 시스템에 Step Functions를 설치하고 실행할 수 있습니다. Step Functions Local을 사용하면 모든 시스템에서 실행을 시작할 수 있습니다.

로컬 버전의 Step Functions는 로컬에서 실행 중일 때와 로컬에서 AWS 실행할 때 모두 AWS Lambda 함수를 호출할 수 있습니다. 지원되는 다른 AWS 서비스를 조정할 수도 있습니다. 자세한 정보는 상태 시스템을 로컬로 테스트을 참조하세요.

> **Note**
>
> Step Functions Local은 더미 계정을 사용하여 작업합니다.

AWS Toolkit for Visual Studio Code

VS 코드를 사용하여 원격 상태 시스템과 상호 작용하고 상태 시스템을 로컬로 개발할 수 있습니다. 상태 시스템을 생성 및 업데이트하고 기존 상태 시스템을 나열하며 상태 시스템을 실행 또는 다운로드할 수 있습니다. 또한 VS Code를 통해 템플릿에서 새 상태 머신을 생성하고 상태 머신의 화면을 보고 코드 조각, 코드 완성 및 코드 유효성 검사를 할 수 있습니다.

자세한 내용은 AWS Toolkit for Visual Studio Code 사용 설명서를 참조하십시오.

AWS Serverless Application Model 및 Step Functions

Step Functions가 와 통합되어 워크플로를 Lambda 함수, API 및 이벤트와 통합하여 서버리스 애플리케이션을 생성할 수 있습니다. AWS Serverless Application Model

통합 환경의 AWS Toolkit for Visual Studio Code 일부로 AWS SAM CLI를 와 함께 사용할 수도 있습니다.

자세한 내용은 AWS Step Functions 및 AWS SAM 섹션을 참조하세요.

TerraForm 및 Step Functions

Terraform HashiCorp by는 코드형 인프라 (IaC) 를 사용하여 애플리케이션을 구축하기 위한 프레임워크입니다. Terraform을 사용하면 상태 시스템을 만들고 인프라 배포 미리 보기 및 재사용 가능한 템플릿 만들기와 같은 기능을 사용할 수 있습니다. Terraform 템플릿을 사용하면 코드를 더 작은 청크로 나눠 유지하고 재사용할 수 있습니다.

자세한 정보는 Terraform을 사용하여 상태 시스템 배포을 참조하세요.

정의 형식 지원

Step Functions는 상태 시스템 정의를 여러 가지 형식으로 제공할 수 있는 다양한 도구를 제공합니다. 상태 시스템의 세부 정보를 지정하는 Amazon States Language(ASL) 정의는 문자열로 제공되거나 JSON 또는 YAML을 사용하는 직렬화된 객체로 제공될 수 있습니다.

> **Note**
>
> YAML은 한 줄 주석을 허용합니다. 템플릿의 상태 시스템 정의 부분에 제공된 YAML 주석은 생성된 리소스 정의로 전달되지 않습니다. 대신 상태 시스템 정의 내에서 `Comment` 속성을 사용할 수 있습니다. 자세한 내용은 상태 시스템 구조 페이지를 참조하세요.

다음 표에서는 ASL 기반 정의를 지원하는 도구를 보여줍니다.

도구별 정의 형식 지원

	JSON	YAML	문자열로 변환된 Amazon States Language		
Step Functions 콘솔	✓				
HTTPS 서비스 API			✓		
AWS CLI			✓		
Step Functions Local			✓		
AWS Toolkit for Visual Studio Code	✓	✓			
AWS SAM	✓	✓			
AWS CloudFormation	✓	✓	✓		

> **Note**
>
> AWS CloudFormation AWS SAM 또한 상태 머신 정의를 Amazon S3에 JSON 또는 YAML 형식으로 업로드하고 템플릿에 정의의 Amazon S3 위치를 제공할 수 있습니다. 이렇게 하면 상태 시스템 정의가 복잡할 때 템플릿 가독성이 향상될 수 있습니다. 자세한 내용은 AWS::StepFunctions::StateMachine S3Location 페이지를 참조하십시오.

다음 예제 AWS CloudFormation 템플릿은 다양한 입력 형식을 사용하여 동일한 스테이트 머신 정의를 제공하는 방법을 보여줍니다.

JSON with Definition

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS Step Functions sample template.",
  "Resources": {
    "MyStateMachine": {
      "Type": "AWS::StepFunctions::StateMachine",
      "Properties": {
        "RoleArn": {
          "Fn::GetAtt": [ "StateMachineRole", "Arn" ]
        },
        "TracingConfiguration": {
          "Enabled": true
        },
        "Definition": {
          "StartAt": "HelloWorld",
          "States": {
            "HelloWorld": {
              "Type": "Pass",
              "End": true
            }
          }
        }
      }
    },
    "StateMachineRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
```

```
          "Statement": [
            {
              "Action": [
                "sts:AssumeRole"
              ],
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "states.amazonaws.com"
                ]
              }
            }
          ]
        },
        "ManagedPolicyArns": [],
        "Policies": [
          {
            "PolicyName": "StateMachineRolePolicy",
            "PolicyDocument": {
              "Statement": [
                {
                  "Action": [
                    "lambda:InvokeFunction"
                  ],
                  "Resource": "*",
                  "Effect": "Allow"
                }
              ]
            }
          }
        ]
      }
    }
  },
  "Outputs": {
    "StateMachineArn": {
      "Value": {
        "Ref": "MyStateMachine"
      }
    }
  }
}
```

JSON with DefinitionString

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS Step Functions sample template.",
  "Resources": {
    "MyStateMachine": {
      "Type": "AWS::StepFunctions::StateMachine",
      "Properties": {
        "RoleArn": {
          "Fn::GetAtt": [ "StateMachineRole", "Arn" ]
        },
        "TracingConfiguration": {
          "Enabled": true
        },
        "DefinitionString": "{\n  \"StartAt\": \"HelloWorld\",\n  \"States\": {\n
 \"HelloWorld\": {\n      \"Type\": \"Pass\",\n      \"End\": true\n    }\n  }\n}"
      }
    },
    "StateMachineRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Action": [
                "sts:AssumeRole"
              ],
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "states.amazonaws.com"
                ]
              }
            }
          ]
        },
        "ManagedPolicyArns": [],
        "Policies": [
          {
            "PolicyName": "StateMachineRolePolicy",
            "PolicyDocument": {
              "Statement": [
```

```
                {
                  "Action": [
                    "lambda:InvokeFunction"
                  ],
                  "Resource": "*",
                  "Effect": "Allow"
                }
              ]
            }
          }
        ]
      }
    }
  },
  "Outputs": {
    "StateMachineArn": {
      "Value": {
        "Ref": "MyStateMachine"
      }
    }
  }
}
```

YAML with Definition

```
AWSTemplateFormatVersion: 2010-09-09
Description: AWS Step Functions sample template.
Resources:
  MyStateMachine:
    Type: 'AWS::StepFunctions::StateMachine'
    Properties:
      RoleArn: !GetAtt
        - StateMachineRole
        - Arn
      TracingConfiguration:
        Enabled: true
      Definition:
        # This is a YAML comment. This will not be preserved in the state machine
 resource's definition.
        Comment: This is an ASL comment. This will be preserved in the state machine
 resource's definition.
        StartAt: HelloWorld
        States:
```

```
          HelloWorld:
            Type: Pass
            End: true
  StateMachineRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Action:
              - 'sts:AssumeRole'
            Effect: Allow
            Principal:
              Service:
                - states.amazonaws.com
      ManagedPolicyArns: []
      Policies:
        - PolicyName: StateMachineRolePolicy
          PolicyDocument:
            Statement:
              - Action:
                  - 'lambda:InvokeFunction'
                Resource: "*"
                Effect: Allow

Outputs:
  StateMachineArn:
    Value:
      Ref: MyStateMachine
```

YAML with DefinitionString

```
AWSTemplateFormatVersion: 2010-09-09
Description: AWS Step Functions sample template.
Resources:
  MyStateMachine:
    Type: 'AWS::StepFunctions::StateMachine'
    Properties:
      RoleArn: !GetAtt
        - StateMachineRole
        - Arn
      TracingConfiguration:
        Enabled: true
```

```
      DefinitionString: |
        {
            "StartAt": "HelloWorld",
            "States": {
                "HelloWorld": {
                    "Type": "Pass",
                    "End": true
                }
            }
        }
  StateMachineRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Action:
              - 'sts:AssumeRole'
            Effect: Allow
            Principal:
              Service:
                - states.amazonaws.com
      ManagedPolicyArns: []
      Policies:
        - PolicyName: StateMachineRolePolicy
          PolicyDocument:
            Statement:
              - Action:
                  - 'lambda:InvokeFunction'
                Resource: "*"
                Effect: Allow

Outputs:
  StateMachineArn:
    Value:
      Ref: MyStateMachine
```

AWS Step Functions 그리고 AWS SAM

통합 환경의 AWS Toolkit for Visual Studio Code 일부로 AWS SAM CLI와 함께 사용하여 로컬에서 상태 머신을 생성할 수 있습니다. AWS SAM을 사용하여 서버리스 애플리케이션을 빌드한 다음 VS

Code IDE에서 상태 시스템을 빌드할 수 있습니다. 그런 다음 리소스를 검증, 패키징 및 배포할 수 있습니다. 선택적으로 에 게시할 수도 있습니다. AWS Serverless Application Repository

> **Tip**
>
> 를 사용하여 Step Functions 워크플로를 시작하는 샘플 서버리스 애플리케이션을 AWS SAM 배포하려면 모듈 11 - AWS Step Functions 워크샵과 함께 AWS SAM배포를 참조하십시오. AWS 계정

주제

Step Functions를 와 함께 사용하는 이유는 AWS SAM무엇입니까?

Step AWS SAM Functions를 함께 사용하면 다음을 수행할 수 있습니다.

- AWS SAM 샘플 템플릿을 사용하여 시작하세요.
- 상태 머신을 서버리스 애플리케이션에 빌드합니다.
- 변수 대체를 사용하여 배포 시 ARN을 상태 시스템으로 대체합니다.

  AWS CloudFormation은 CloudFormation 템플릿에 제공하는 값에 워크플로 정의의 동적 참조를 추가할 수 있도록 `DefinitionSubstitutions`를 지원합니다. `${dollar_sign_brace}` 표기법으로 워크플로 정의에 대체 항목을 추가하여 동적 참조를 추가할 수 있습니다. 또한 CloudFormation 템플릿의 StateMachine 리소스 `DefinitionSubstitutions` 속성에 이러한 동적 참조를 정의해야 합니다. CloudFormation 스택 생성 프로세스 중에 이러한 대체 값이 실제 값으로 대체됩니다. 자세한 정보는 DefinitionSubstitutions 템플릿에서 AWS SAM을 참조하세요.
- AWS SAM 정책 템플릿을 사용하여 상태 머신의 역할을 지정합니다.
- API Gateway, EventBridge 이벤트를 사용하거나 템플릿 내 일정에 AWS SAM 따라 스테이트 머신 실행을 시작합니다.

AWS SAM 사양과 Step Functions 통합

AWS SAM 정책 템플릿을 사용하여 상태 시스템에 권한을 추가할 수 있습니다. 이러한 권한을 통해 Lambda 함수 및 기타 AWS 리소스를 오케스트레이션하여 복잡하고 강력한 워크플로를 형성할 수 있습니다.

Step Functions와 SAM CLI의 통합

Step Functions는 AWS SAM CLI와 통합되어 있습니다. 이를 사용하여 상태 머신를 서버리스 애플리케이션으로 신속하게 개발합니다.

AWS SAM을 사용하여 Step Functions 상태 시스템 만들기튜토리얼을 통해 스테이트 머신을 생성하는 AWS SAM 데 사용하는 방법을 알아보십시오.

지원되는 AWS SAM CLI 함수는 다음과 같습니다.

CLI 명령	설명
sam 이니트	템플릿을 사용하여 서버리스 애플리케이션을 초기화합니다. AWS SAM Step Functions에 대해 SAM 템플릿과 함께 사용할 수 있습니다.
sam validate	템플릿을 검증합니다. AWS SAM
sam package	AWS SAM 애플리케이션을 패키징합니다. 코드와 종속성의 ZIP 파일을 만든 다음 Amazon S3에 업로드합니다. 그런 다음 로컬 아티팩트에 대한 참조를 명령이 아티팩트를 업로드한 Amazon S3 위치로 변경하여 AWS SAM 템플릿의 사본을 반환합니다.
sam deploy	AWS SAM 애플리케이션을 배포합니다.
sam publish	에 AWS SAM 애플리케이션을 게시합니다. AWS Serverless Application Repository이 명령은 패키지된 AWS SAM 템플릿을 가져와 응용 프로그램을 지정된 지역에 게시합니다.

> **Note**
>
> AWS SAM 로컬을 사용하는 경우 Lambda 및 API Gateway를 로컬로 에뮬레이션할 수 있습니다. 그러나 를 사용하여 AWS SAM로컬에서 Step Functions를 에뮬레이션할 수는 없습니다.

DefinitionSubstitutions 템플릿에서 AWS SAM

AWS SAM으로 CloudFormation 템플릿을 사용하여 상태 머신을 정의할 수 있습니다. AWS SAM을 사용하여 템플릿 또는 별도의 파일에서 상태 머신을 인라인으로 정의할 수 있습니다. 다음 AWS SAM 템플릿에는 주식 거래 워크플로를 시뮬레이션하는 상태 머신이 포함되어 있습니다. 이 상태 머신은 주식 가격을 확인하고 주식 매수 또는 매도 여부를 결정하는 3가지 Lambda 함수를 호출합니다. 그러면 이 거래가 Amazon DynamoDB 테이블에 기록됩니다. 다음 템플릿의 Lambda 함수 및 DynamoDB 테이블의 ARN은 `DefinitionSubstitutions`를 사용하여 지정됩니다.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: |
  step-functions-stock-trader
  Sample SAM Template for step-functions-stock-trader
Resources:
  StockTradingStateMachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      DefinitionSubstitutions:
        StockCheckerFunctionArn: !GetAtt StockCheckerFunction.Arn
        StockSellerFunctionArn: !GetAtt StockSellerFunction.Arn
        StockBuyerFunctionArn: !GetAtt StockBuyerFunction.Arn
        DDBPutItem: !Sub arn:${AWS::Partition}:states:::dynamodb:putItem
        DDBTable: !Ref TransactionTable
      Policies:
        - DynamoDBWritePolicy:
            TableName: !Ref TransactionTable
        - LambdaInvokePolicy:
            FunctionName: !Ref StockCheckerFunction
        - LambdaInvokePolicy:
            FunctionName: !Ref StockBuyerFunction
        - LambdaInvokePolicy:
            FunctionName: !Ref StockSellerFunction
      DefinitionUri: statemachine/stock_trader.asl.json
  StockCheckerFunction:
```

```
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: functions/stock-checker/
      Handler: app.lambdaHandler
      Runtime: nodejs18.x
      Architectures:
        - x86_64
  StockSellerFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: functions/stock-seller/
      Handler: app.lambdaHandler
      Runtime: nodejs18.x
      Architectures:
        - x86_64
  StockBuyerFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: functions/stock-buyer/
      Handler: app.lambdaHandler
      Runtime: nodejs18.x
      Architectures:
        - x86_64
  TransactionTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
```

다음 코드는 AWS SAM을 사용하여 Step Functions 상태 시스템 만들기 자습서에서 사용되는 stock_trader.asl.json 파일의 상태 머신 정의입니다. 이 상태 머신 정의에는 ${dollar_sign_brace} 표기법으로 표시된 여러 개의 DefinitionSubstitutions가 포함되어 있습니다. 예를 들어 Check Stock Value 작업에 정적 Lambda 함수 ARN을 지정하는 대신 대체 ${StockCheckerFunctionArn} 함수를 사용합니다. 이 대체 값은 템플릿의 DefinitionSubstitutions 속성에 정의되어 있습니다. DefinitionSubstitutions는 상태 머신 리소스의 키-값 페어의 맵입니다. 에서 DefinitionSubstitutions $ {StockCheckerFunctionArn} 는 CloudFormation 내장 함수를 사용하여 StockCheckerFunction 리소스의 ARN에 매핑됩니다. !GetAtt AWS SAM 템플릿을 배포하면 템플릿의 DefinitionSubstitutions가 실제 값으로 대체됩니다.

```
{
    "Comment": "A state machine that does mock stock trading.",
```

```
    "StartAt": "Check Stock Value",
    "States": {
        "Check Stock Value": {
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "OutputPath": "$.Payload",
            "Parameters": {
                "Payload.$": "$",
                "FunctionName": "${StockCheckerFunctionArn}"
            },
            "Next": "Buy or Sell?"
        },
        "Buy or Sell?": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.stock_price",
                    "NumericLessThanEquals": 50,
                    "Next": "Buy Stock"
                }
            ],
            "Default": "Sell Stock"
        },
        "Buy Stock": {
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "OutputPath": "$.Payload",
            "Parameters": {
                "Payload.$": "$",
                "FunctionName": "${StockBuyerFunctionArn}"
            },
            "Retry": [
                {
                    "ErrorEquals": [
                        "Lambda.ServiceException",
                        "Lambda.AWSLambdaException",
                        "Lambda.SdkClientException",
                        "Lambda.TooManyRequestsException"
                    ],
                    "IntervalSeconds": 1,
                    "MaxAttempts": 3,
                    "BackoffRate": 2
                }
            ],
```

```
            "Next": "Record Transaction"
        },
        "Sell Stock": {
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "OutputPath": "$.Payload",
            "Parameters": {
                "Payload.$": "$",
                "FunctionName": "${StockSellerFunctionArn}"
            },
            "Next": "Record Transaction"
        },
        "Record Transaction": {
            "Type": "Task",
            "Resource": "arn:aws:states:::dynamodb:putItem",
            "Parameters": {
                "TableName": "${DDBTable}",
                "Item": {
                    "Id": {
                        "S.$": "$.id"
                    },
                    "Type": {
                        "S.$": "$.type"
                    },
                    "Price": {
                        "N.$": "$.price"
                    },
                    "Quantity": {
                        "N.$": "$.qty"
                    },
                    "Timestamp": {
                        "S.$": "$.timestamp"
                    }
                }
            },
            "End": true
        }
    }
}
```

다음 단계

다음 리소스를 통해 Step Functions를 AWS SAM 사용하는 방법에 대해 자세히 알아볼 수 있습니다.

- AWS SAM을 사용하여 Step Functions 상태 시스템 만들기튜토리얼을 완료하여 를 사용하여 스테이트 머신을 생성하십시오 AWS SAM.
- AWS::Serverless::StateMachine리소스를 지정하세요.
- 사용할 AWS SAM 정책 템플릿을 찾습니다.
- Step Functions에서 AWS Toolkit for Visual Studio Code를 사용합니다.
- AWS SAM에서 제공하는 기능을 자세히 알아보려면 AWS SAM CLI 참조를 검토하세요.

또한 Application Composer의 Workflow Studio와 같은 비주얼 빌더를 사용하여 코드형 인프라(IaC)에서 워크플로를 설계하고 구축할 수 있습니다. 자세한 정보는 Application Composer에서 Workflow Studio 사용을 참조하세요.

Application Composer에서 Workflow Studio 사용

AWS Application Composer은 간단한 그래픽 인터페이스를 사용하여 AWS SAM 및 AWS CloudFormation 템플릿을 개발하는 데 도움이 되는 시각적 빌더입니다. Application Composer를 사용하면 시각적 캔버스에서 AWS 서비스를 드래그, 그룹화 및 연결하여 애플리케이션 아키텍처를 설계할 수 있습니다. 그런 다음 Application Composer는 AWS SAM 명령줄 인터페이스(AWS SAM CLI) 또는 CloudFormation을 사용하여 애플리케이션을 배포하는 데 사용할 수 있는 코드형 인프라(IaC) 템플릿을 설계에서 생성합니다. Application Composer에 대한 자세한 내용은 Application Composer란을 참조하십시오.

Application Composer에서 워크플로를 설계하고 구축하는 데 도움이 되는 Workflow Studio를 사용할 수 있습니다. Application Composer의 Workflow Studio는 IaC 도구(예: CloudFormation 템플릿)를 사용하여 구축한 서버리스 애플리케이션에 워크플로를 쉽게 통합할 수 있는 시각적 IaC 환경을 제공합니다. Application Composer에서 Workflow Studio를 사용하면 개별 워크플로 단계를 AWS 리소스에 연결하고 AWS SAM 템플릿에 리소스 구성을 생성합니다. 또한 워크플로를 실행하는 데 필요한 IAM 권한도 추가됩니다. Application Composer에서 Workflow Studio를 사용하면 애플리케이션의 프로토타입을 만들어 바로 사용할 수 있는 애플리케이션으로 전환할 수 있습니다.

Application Composer에서 Workflow Studio를 사용하면 Application Composer 캔버스와 Workflow Studio를 전환할 수 있습니다.

주제

- Application Composer에서 Workflow Studio를 사용하여 서버리스 워크플로 구축
- Workflow Studio에서 CloudFormation 정의 대체를 사용하여 리소스를 동적으로 참조
- 서비스 통합 태스크를 향상된 구성 요소 카드에 연결

- 기존 프로젝트를 가져와 로컬로 동기화합니다.
- AWS Application Composer에서 사용할 수 없는 Workflow Studio 기능

Application Composer에서 Workflow Studio를 사용하여 서버리스 워크플로 구축

1. Application Composer 콘솔을 열고 프로젝트 생성을 선택하여 프로젝트를 생성합니다.
2. 리소스 팔레트의 검색 필드에 **`state machine`**를 입력합니다.
3. Step Functions 상태 머신 리소스를 캔버스 위로 드래그합니다.
4. Workflow Studio에서 편집을 선택하여 상태 머신 리소스를 편집합니다.

   다음 애니메이션은 상태 머신 정의를 편집하기 위해 Workflow Studio로 전환하는 방법을 보여 줍니다.
   Application Composer에서 Workflow Studio를 사용하는 방법을 보여 주는 애니메이션입니다.

   Application Composer에서 만든 상태 머신 리소스를 편집하기 위한 Workflow Studio와의 통합은 `AWS::Serverless::StateMachine` 리소스에만 사용할 수 있습니다. `AWS::StepFunctions::StateMachine` 리소스를 사용하는 템플릿에는 이 통합을 사용할 수 없습니다.

Workflow Studio에서 CloudFormation 정의 대체를 사용하여 리소스를 동적으로 참조

Workflow Studio에서는 워크플로 정의에 CloudFormation 정의 대체를 사용하여 IaC 템플릿에서 정의한 리소스를 동적으로 참조할 수 있습니다. `${dollar_sign_brace}` 표기법을 사용하여 워크플로 정의에 자리 표시자 대체를 추가할 수 있습니다. 그러면 CloudFormation 스택 생성 프로세스 중에 자리 표시자 대체가 실제 값으로 대체됩니다. 정의 대체에 대한 자세한 내용은 DefinitionSubstitutions 템플릿에서 AWS SAM 섹션을 참조하세요.

다음 애니메이션은 상태 머신 정의의 리소스에 자리 표시자 대체를 추가하는 방법을 보여 줍니다.
Application Composer에서 Workflow Studio를 사용할 때 AWS Lambda 함수, 정의 대체 등의 리소스를 동적으로 참조하는 방법을 보여 주는 애니메이션입니다.

서비스 통합 태스크를 향상된 구성 요소 카드에 연결

최적화 서비스 통합을 호출하는 태스크를 Application Composer 캔버스의 향상된 구성 요소 카드에 연결할 수 있습니다. 이렇게 하면 워크플로 정의의 `${dollar_sign_brace}` 표기법으로 지정된 모든 자리 표시자 대체가 `StateMachine` 리소스의 `DefinitionSubstitution` 속성과 자동으로 매핑됩니다. 또한 상태 머신에 적절한 AWS SAM 정책을 추가합니다.

최적화 서비스 통합 태스크를 표준 구성 요소 카드에 매핑하는 경우 Application Composer 캔버스에 연결선이 표시되지 않습니다.

다음 애니메이션은 최적화된 태스크를 향상된 구성 요소 카드에 연결하고 변경 사항 검사기에서 변경 사항을 보는 방법을 보여 줍니다.
Application Composer에서 Workflow Studio를 사용할 때 최적화 서비스 통합을 호출하는 태스크를 향상된 구성 요소 카드에 연결하는 방법을 보여 주는 애니메이션입니다.

태스크 상태의 AWS SDK 통합을 향상된 구성 요소 카드 또는 표준 구성 요소 카드가 포함된 최적화 서비스 통합과 연결할 수는 없습니다. 이러한 태스크의 경우 Application Composer 캔버스의 리소스 속성 패널에서 대체를 매핑하고 AWS SAM 템플릿에 정책을 추가할 수 있습니다.

> **Tip**
>
> 리소스 속성 패널의 정의 대체에서 상태 머신의 자리 표시자 대체를 매핑할 수도 있습니다. 이렇게 하려면 상태 머신 실행 역할의 태스크 상태 호출에 AWS 서비스에 대해 필요한 권한을 추가해야 합니다. 필요할 수 있는 실행 역할 권한에 대해 알아보려면 Workflow Studio의 실행 역할 섹션을 참조하세요.

다음 애니메이션은 리소스 속성 패널에서 자리 표시자 대체 매핑을 수동으로 업데이트하는 방법을 보여 줍니다.
Application Composer에서 Workflow Studio를 사용할 때 리소스 속성 패널에서 자리 표시자 대체 매핑을 수동으로 업데이트하는 방법을 보여 주는 애니메이션입니다.

기존 프로젝트를 가져와 로컬로 동기화합니다.

Application Composer에서 기존 CloudFormation 및 AWS SAM 프로젝트를 열어 이를 시각화하여 디자인을 더 잘 이해하고 수정할 수 있습니다. Application Composer의 로컬 동기화 기능을 사용하면 템플릿과 코드 파일을 로컬 빌드 컴퓨터에 자동으로 동기화하고 저장할 수 있습니다. 로컬 동기화 모드를 사용하면 기존 개발 흐름을 보완할 수 있습니다. 웹 애플리케이션이 로컬 파일 시스템에서 파일을 읽고

쓰고 저장할 수 있도록 하는 File System Access API를 지원하는 브라우저가 있는지 확인합니다. 구글 크롬이나 Microsoft Edge를 사용하는 것이 좋습니다.

AWS Application Composer에서 사용할 수 없는 Workflow Studio 기능

Application Composer에서 Workflow Studio를 사용할 때 일부 Workflow Studio 기능은 사용할 수 없습니다. 또한 Inspector 패널에서 사용할 수 있는 API 파라미터 섹션에서는 CloudFormation 정의 대체를 지원합니다. `${dollar_sign_brace}` 표기법을 사용하여 코드 모드에서 대체를 추가할 수 있습니다. 이 표기법에 대한 자세한 내용은 DefinitionSubstitutions 템플릿에서 AWS SAM 섹션을 참조하세요.

다음 목록은 Application Composer에서 Workflow Studio를 사용할 때 사용할 수 없는 Workflow Studio 기능을 설명합니다.

- 스타터 템플릿 - 스타터 템플릿은 워크플로 프로토타입과 정의를 자동으로 생성하여 바로 실행 가능한 샘플 프로젝트입니다. 이 템플릿은 프로젝트에서 필요한 모든 관련 AWS 리소스를 AWS 계정에 배포합니다.
- 구성 모드 - 이 모드를 사용하면 상태 머신의 구성을 관리할 수 있습니다. IaC 템플릿에서 상태 머신 구성을 업데이트하거나 Application Composer 캔버스의 리소스 속성 패널을 사용할 수 있습니다. 리소스 속성 패널의 구성 업데이트에 대한 자세한 내용은 서비스 통합 태스크를 향상된 구성 요소 카드에 연결 섹션을 참조하세요.
- TestState API
- Workflow Studio의 작업 드롭다운 버튼에서 워크플로 정의를 가져오거나 내보내는 옵션입니다. 대신 Application Composer 메뉴에서 열기 > 프로젝트 폴더를 선택합니다. Application Composer 캔버스의 변경 사항을 로컬 머신에 직접 자동으로 저장하도록 로컬 동기화 모드를 활성화했는지 확인합니다.
- 실행 버튼을 사용합니다. Application Composer에서 Workflow Studio를 사용하는 경우 Application Composer에서 워크플로용 IaC 코드가 생성됩니다. 따라서 먼저 템플릿을 배포해야 합니다. 그런 다음 콘솔 또는 AWS Command Line Interface (AWS CLI)를 통해 워크플로를 실행합니다.

AWS CloudFormation를 사용하여 Step Functions용 Lambda 상태 시스템 만들기

이 자습서에서는 를 사용하여 기본 AWS Lambda 함수를 만드는 방법을 보여줍니다 AWS CloudFormation. AWS CloudFormation 콘솔과 YAML 템플릿을 사용하여 스택 (IAM 역할, Lambda 함수, 상태 머신) 을 생성합니다. 그런 다음 AWS Step Functions 콘솔을 사용하여 스테이트 머신 실행을 시작합니다.

자세한 내용은 AWS CloudFormation 사용 설명서의 CloudFormation 템플릿 및 *AWS::StepFunctions::StateMachine* 리소스 사용을 참조하십시오.

주제

- 1단계: AWS CloudFormation 템플릿 설정
- 2단계: AWS CloudFormation 템플릿을 사용하여 Lambda 상태 시스템 생성
- 3단계: 상태 시스템 실행 시작

1단계: AWS CloudFormation 템플릿 설정

예제 템플릿을 사용하기 전에 AWS CloudFormation 템플릿의 다른 부분을 선언하는 방법을 이해해야 합니다.

주제

- Lambda용 IAM 역할을 생성하려면,
- Lambda 함수를 생성하는 방법
- 상태 시스템 실행에 대한 IAM 역할 만들기
- Lambda 상태 시스템 만들기

Lambda용 IAM 역할을 생성하려면,

Lambda 함수용 IAM 역할과 관련된 신뢰 정책을 정의합니다. 다음 예제에서는 YAML 또는 JSON을 사용하여 신뢰 정책을 정의합니다.

YAML

```
LambdaExecutionRole:
  Type: "AWS::IAM::Role"
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: "sts:AssumeRole"
```

JSON

```
        "LambdaExecutionRole": {
          "Type": "AWS::IAM::Role",
          "Properties": {
              "AssumeRolePolicyDocument": {
                  "Version": "2012-10-17",
                  "Statement": [
                      {
                          "Effect": "Allow",
                          "Principal": {
                              "Service": "lambda.amazonaws.com"
                          },
                          "Action": "sts:AssumeRole"
                      }
                  ]
              }
          }
```

Lambda 함수를 생성하는 방법

`Hello World` 메시지를 출력하는 Lambda 함수의 다음 속성을 정의합니다.

> **Important**
>
> Lambda 함수가 상태 머신과 AWS 동일한 계정 AWS 및 지역에 속하는지 확인하십시오.

YAML

```
MyLambdaFunction:
  Type: "AWS::Lambda::Function"
  Properties:
    Handler: "index.handler"
    Role: !GetAtt [ LambdaExecutionRole, Arn ]
    Code:
      ZipFile: |
        exports.handler = (event, context, callback) => {
            callback(null, "Hello World!");
        };
    Runtime: "nodejs12.x"
```

```
    Timeout: "25"
```

JSON

```
        "MyLambdaFunction": {
            "Type": "AWS::Lambda::Function",
            "Properties": {
                "Handler": "index.handler",
                "Role": {
                    "Fn::GetAtt": [
                        "LambdaExecutionRole",
                        "Arn"
                    ]
                },
                "Code": {
                    "ZipFile": "exports.handler = (event, context, callback) => {\n
   callback(null, \"Hello World!\");\n};\n"
                },
                "Runtime": "nodejs12.x",
                "Timeout": "25"
            }
        },
```

상태 시스템 실행에 대한 IAM 역할 만들기

상태 시스템 실행에 대한 IAM 역할과 관련된 신뢰 정책을 정의합니다.

YAML

```
StatesExecutionRole:
  Type: "AWS::IAM::Role"
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: "Allow"
          Principal:
            Service:
              - !Sub states.${AWS::Region}.amazonaws.com
          Action: "sts:AssumeRole"
    Path: "/"
    Policies:
```

```
      - PolicyName: StatesExecutionPolicy
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action:
                - "lambda:InvokeFunction"
              Resource: "*"
```

JSON

```
        "StatesExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    {
                                        "Fn::Sub": "states.
${AWS::Region}.amazonaws.com"
                                    }
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "StatesExecutionPolicy",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "lambda:InvokeFunction"
                                    ],
```

```
                                    "Resource": "*"
                                }
                            ]
                        }
                    }
                ]
            }
        },
```

Lambda 상태 시스템 만들기

Lambda 상태 시스템을 정의합니다.

YAML

```
MyStateMachine:
  Type: "AWS::StepFunctions::StateMachine"
  Properties:
    DefinitionString:
      !Sub
        - |-
          {
            "Comment": "A Hello World example using an AWS Lambda function",
            "StartAt": "HelloWorld",
            "States": {
              "HelloWorld": {
                "Type": "Task",
                "Resource": "${lambdaArn}",
                "End": true
              }
            }
          }
        - {lambdaArn: !GetAtt [ MyLambdaFunction, Arn ]}
    RoleArn: !GetAtt [ StatesExecutionRole, Arn ]
```

JSON

```
        "MyStateMachine": {
            "Type": "AWS::StepFunctions::StateMachine",
            "Properties": {
                "DefinitionString": {
                    "Fn::Sub": [
```

```
                        "{\n  \"Comment\": \"A Hello World example using an
 AWS Lambda function\",\n  \"StartAt\": \"HelloWorld\",\n  \"States\": {\n
 \"HelloWorld\": {\n      \"Type\": \"Task\",\n      \"Resource\": \"${lambdaArn}\",
\n      \"End\": true\n    }\n  }\n}",
                        {
                            "lambdaArn": {
                                "Fn::GetAtt": [
                                    "MyLambdaFunction",
                                    "Arn"
                                ]
                            }
                        }
                    ]
                },
                "RoleArn": {
                    "Fn::GetAtt": [
                        "StatesExecutionRole",
                        "Arn"
                    ]
                }
            }
        }
```

2단계: AWS CloudFormation 템플릿을 사용하여 Lambda 상태 시스템 생성

AWS CloudFormation 템플릿의 구성 요소를 이해하면 구성 요소를 한데 모아 템플릿을 사용하여 스택을 생성할 수 있습니다. AWS CloudFormation

Lambda 상태 시스템 만들기

1. 다음 예제 데이터를 YAML 예제용 `MyStateMachine.yaml` 또는 JSON용 `MyStateMachine.json` 파일에 복사합니다.

   YAML

   ```
   AWSTemplateFormatVersion: "2010-09-09"
   Description: "An example template with an IAM role for a Lambda state machine."
   Resources:
     LambdaExecutionRole:
       Type: "AWS::IAM::Role"
       Properties:
         AssumeRolePolicyDocument:
   ```

```
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service: lambda.amazonaws.com
          Action: "sts:AssumeRole"

MyLambdaFunction:
  Type: "AWS::Lambda::Function"
  Properties:
    Handler: "index.handler"
    Role: !GetAtt [ LambdaExecutionRole, Arn ]
    Code:
      ZipFile: |
        exports.handler = (event, context, callback) => {
            callback(null, "Hello World!");
        };
    Runtime: "nodejs12.x"
    Timeout: "25"

StatesExecutionRole:
  Type: "AWS::IAM::Role"
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: "Allow"
          Principal:
            Service:
              - !Sub states.${AWS::Region}.amazonaws.com
          Action: "sts:AssumeRole"
    Path: "/"
    Policies:
      - PolicyName: StatesExecutionPolicy
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action:
                - "lambda:InvokeFunction"
              Resource: "*"

MyStateMachine:
  Type: "AWS::StepFunctions::StateMachine"
```

```
    Properties:
      DefinitionString:
        !Sub
          - |-
            {
              "Comment": "A Hello World example using an AWS Lambda function",
              "StartAt": "HelloWorld",
              "States": {
                "HelloWorld": {
                  "Type": "Task",
                  "Resource": "${lambdaArn}",
                  "End": true
                }
              }
            }
          - {lambdaArn: !GetAtt [ MyLambdaFunction, Arn ]}
      RoleArn: !GetAtt [ StatesExecutionRole, Arn ]
```

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "An example template with an IAM role for a Lambda state
 machine.",
    "Resources": {
        "LambdaExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "lambda.amazonaws.com"
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                }
            }
        },
        "MyLambdaFunction": {
```

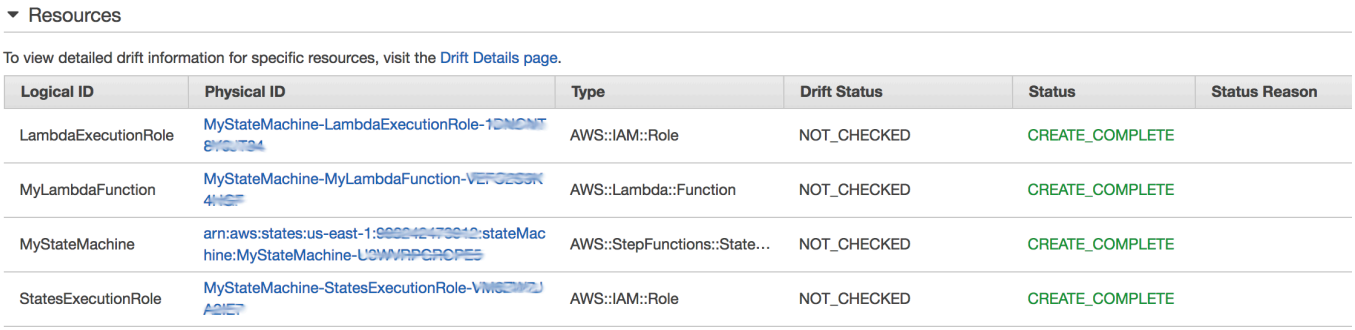
```
            "Type": "AWS::Lambda::Function",
            "Properties": {
                "Handler": "index.handler",
                "Role": {
                    "Fn::GetAtt": [
                        "LambdaExecutionRole",
                        "Arn"
                    ]
                },
                "Code": {
                    "ZipFile": "exports.handler = (event, context, callback) =>
 {\n    callback(null, \"Hello World!\");\n};\n"
                },
                "Runtime": "nodejs12.x",
                "Timeout": "25"
            }
        },
        "StatesExecutionRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    {
                                        "Fn::Sub": "states.
${AWS::Region}.amazonaws.com"
                                    }
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "StatesExecutionPolicy",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
```

```
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "lambda:InvokeFunction"
                                    ],
                                    "Resource": "*"
                                }
                            ]
                        }
                    }
                ]
            }
        },
        "MyStateMachine": {
            "Type": "AWS::StepFunctions::StateMachine",
            "Properties": {
                "DefinitionString": {
                    "Fn::Sub": [
                        "{\n  \"Comment\": \"A Hello World example using
 an AWS Lambda function\",\n  \"StartAt\": \"HelloWorld\",\n  \"States\":
 {\n    \"HelloWorld\": {\n      \"Type\": \"Task\",\n      \"Resource\":
 \"${lambdaArn}\",\n      \"End\": true\n    }\n  }\n}",
                        {
                            "lambdaArn": {
                                "Fn::GetAtt": [
                                    "MyLambdaFunction",
                                    "Arn"
                                ]
                            }
                        }
                    ]
                },
                "RoleArn": {
                    "Fn::GetAtt": [
                        "StatesExecutionRole",
                        "Arn"
                    ]
                }
            }
        }
    }
}
```

2. AWS CloudFormation 콘솔을 열고 스택 생성을 선택합니다.

3. 템플릿 선택 페이지에서 Amazon S3에 템플릿 업로드를 선택합니다. `MyStateMachine` 파일을 선택한 후 다음을 선택합니다.
4. 세부 정보 지정 페이지에서 스택 이름에 `MyStateMachine`을 입력한 후 다음을 선택합니다.
5. 옵션 페이지에서 다음을 선택합니다.
6. 검토 페이지에서 I acknowledge that AWS CloudFormation might create IAM resources를 선택한 다음 생성을 선택합니다.

   AWS CloudFormation `MyStateMachine`스택 생성을 시작하고 CREATE_IN_PROGRESS 상태를 표시합니다. 이 과정이 완료되면 AWS CloudFormation 에는 CREATE_COMPLETE 상태가 표시됩니다.
7. (선택 사항) 스택에 리소스를 표시하려면 스택을 선택하고 [Resources] 탭을 선택합니다.


Resources

To view detailed drift information for specific resources, visit the Drift Details page.

Logical ID	Physical ID	Type	Drift Status	Status	Status Reason
LambdaExecutionRole	MyStateMachine-LambdaExecutionRole-1[illegible]	AWS::IAM::Role	NOT_CHECKED	CREATE_COMPLETE	
MyLambdaFunction	MyStateMachine-MyLambdaFunction-[illegible]	AWS::Lambda::Function	NOT_CHECKED	CREATE_COMPLETE	
MyStateMachine	arn:aws:states:us-east-1:[illegible]:stateMachine:MyStateMachine-[illegible]	AWS::StepFunctions::State...	NOT_CHECKED	CREATE_COMPLETE	
StatesExecutionRole	MyStateMachine-StatesExecutionRole-[illegible]	AWS::IAM::Role	NOT_CHECKED	CREATE_COMPLETE	

3단계: 상태 시스템 실행 시작

Lambda 상태 시스템을 만들었으면 실행을 시작할 수 있습니다.

상태 시스템 실행을 시작하려면

1. Step Functions 콘솔을 열고 사용하여 만든 상태 머신의 이름을 선택합니다 AWS CloudFormation.
2. ***MyStateMachine-ABCDEFGHIJ1K*** 페이지에서 새 실행을 선택합니다.

   [New execution] 페이지가 표시됩니다.
3. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

4. 실행 시작을 선택합니다.

   상태 머신의 새로운 실행이 시작되고 실행 중인 실행을 보여주는 새로운 페이지가 표시됩니다.

5. (선택 사항) 실행 세부 정보에서 실행 상태와 시작됨 및 종료됨 타임스탬프를 검토합니다.

6. 실행 결과를 보려면 Output(출력)을 선택합니다.

AWS CDK를 사용하여 Step Functions용 Lambda 상태 시스템 만들기

이 자습서에서는 AWS Cloud Development Kit (AWS CDK)를 사용하여 AWS Lambda 함수가 포함된 AWS Step Functions 상태 시스템을 만드는 방법을 보여줍니다. AWS CDK는 완전한 프로그래밍 언어를 사용하여 인프라를 정의할 수 있는 코드형 AWS 인프라 (IAC) 프레임워크입니다. 스택이 하나 이상 포함된 CDK 지원 언어 중 하나로 앱을 작성할 수 있습니다. 그런 다음 이를 AWS CloudFormation 템플릿에 합성하여 계정에 배포할 수 있습니다. AWS 이 메서드를 사용하여 Lambda 함수가 포함된 Step Functions 상태 머신을 정의한 다음 AWS Management Console 를 사용하여 스테이트 머신을 실행합니다.

이 자습서를 시작하기 전에 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 AWS CDK 시작하기 - 사전 조건의 설명대로 AWS CDK 개발 환경을 설정해야 합니다. 그런 다음 다음 명령으로 AWS CLI에 AWS CDK를 설치합니다.

```
npm install -g aws-cdk
```

이 자습서에서는 the section called “를 사용하여 Lambda 상태 머신 생성 AWS CloudFormation”와 동일한 결과를 생성합니다. 하지만 이 자습서에서 AWS CDK를 사용하기 위해 IAM 역할을 만들 필요가 없습니다. AWS CDK에서 자동으로 만듭니다. AWS CDK 버전에도 상태 시스템에 단계를 추가하는 방법을 설명하는 Succeed 단계가 포함되어 있습니다.

> **Tip**
>
> wth를 사용하여 Step Functions 워크플로를 AWS CDK 시작하는 샘플 서버리스 애플리케이션을 사용자에게 TypeScript 배포하려면 모듈 10 - AWS 계정 The AWS Step Functions Workshop과 함께 AWS CDK 배포를 참조하십시오.

주제

1단계: AWS CDK 프로젝트 설정

1. 홈 디렉터리 또는 원하는 경우 다른 디렉터리에서 다음 명령을 실행하여 새 AWS CDK 앱을 위한 디렉터리를 만듭니다.

   > **Important**
   >
   > 디렉터리 이름을 `step`으로 지정해야 합니다. AWS CDK 애플리케이션 템플릿은 디렉터리 이름을 사용하여 소스 파일과 클래스의 이름을 생성합니다. 다른 이름을 사용하는 경우 앱이 이 자습서와 일치하지 않습니다.

   TypeScript

   ```
   mkdir step && cd step
   ```

   JavaScript

   ```
   mkdir step && cd step
   ```

Python

```
mkdir step && cd step
```

Java

```
mkdir step && cd step
```

C#

.NET 버전 6.0 이상을 설치했는지 확인합니다. 자세한 내용은 지원되는 버전을 참조하세요.

```
mkdir step && cd step
```

2. cdk init 명령을 사용하여 앱을 초기화합니다. 다음 예제와 같이 원하는 템플릿("app")과 프로그래밍 언어를 지정합니다.

TypeScript

```
cdk init --language typescript
```

JavaScript

```
cdk init --language javascript
```

Python

```
cdk init --language python
```

프로젝트가 초기화되면 프로젝트의 가상 환경을 활성화하고 AWS CDK 기준선 종속성을 설치합니다.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
cdk init --language java
```

C#

```
cdk init --language csharp
```

2단계: AWS CDK를 사용하여 상태 시스템 만들기

먼저 Lambda 함수와 Step Functions 상태 시스템을 정의하는 개별 코드 조각을 살펴보겠습니다. 그런 다음 AWS CDK 앱에 이들을 배치하는 방법을 설명하겠습니다. 마지막으로 이러한 리소스를 합성하고 배포하는 방법을 살펴보겠습니다.

Lambda 함수를 만들려면

다음 AWS CDK 코드는 Lambda 함수를 정의하고 소스 코드를 인라인으로 제공합니다.

TypeScript

```
const helloFunction = new lambda.Function(this, 'MyLambdaFunction', {
    code: lambda.Code.fromInline(`
          exports.handler = (event, context, callback) => {
              callback(null, "Hello World!");
          };
      `),
    runtime: lambda.Runtime.NODEJS_18_X,
    handler: "index.handler",
    timeout: cdk.Duration.seconds(3)
});
```

JavaScript

```
const helloFunction = new lambda.Function(this, 'MyLambdaFunction', {
    code: lambda.Code.fromInline(`
          exports.handler = (event, context, callback) => {
              callback(null, "Hello World!");
          };
      `),
    runtime: lambda.Runtime.NODEJS_18_X,
    handler: "index.handler",
    timeout: cdk.Duration.seconds(3)
});
```

Python

```
hello_function = lambda_.Function(
            self, "MyLambdaFunction",
            code=lambda_.Code.from_inline("""
            exports.handler = (event, context, callback) => {
                callback(null, "Hello World!");
                }"""),
                runtime=lambda_.Runtime.NODEJS_18_X,
                handler="index.handler",
                timeout=Duration.seconds(25))
```

Java

```
final Function helloFunction = Function.Builder.create(this, "MyLambdaFunction")
        .code(Code.fromInline(
                "exports.handler = (event, context, callback) => { callback(null,
 'Hello World!' );}"))
        .runtime(Runtime.NODEJS_18_X)
        .handler("index.handler")
        .timeout(Duration.seconds(25))
        .build();
```

C#

```
var helloFunction = new Function(this, "MyLambdaFunction", new FunctionProps
{
    Code = Code.FromInline(@"`
      exports.handler = (event, context, callback) => {
        callback(null, 'Hello World!');
      }"),
    Runtime = Runtime.NODEJS_12_X,
    Handler = "index.handler",
    Timeout = Duration.Seconds(25)
});
```

이 짧은 예제 코드에서 다음을 확인할 수 있습니다.

- 함수의 논리명(MyLambdaFunction)
- AWS CDK 앱의 소스 코드에 문자열로 임베딩된 함수의 소스 코드
- 기타 함수 속성(예: 사용할 런타임(Node 18.x)), 함수 진입점 및 제한 시간

상태 시스템을 생성하려면

상태 시스템에는 Lambda 함수 작업과 Succeed 상태 등 두 가지 상태가 있습니다. 함수를 사용하려면 함수를 간접적으로 호출하는 Step Functions the section called “작업”를 만들어야 합니다. 이 Task 상태는 상태 시스템의 첫 번째 단계로 사용됩니다. Task 상태의 `next()` 메서드를 통해 성공 상태가 상태 시스템에 추가됩니다. 다음 코드는 먼저 `MyLambdaTask` 함수를 간접적으로 호출한 다음 `next()` 메서드를 사용하여 `GreetedWorld` 성공 상태를 정의합니다.

TypeScript

```
const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
  definition: new tasks.LambdaInvoke(this, "MyLambdaTask", {
    lambdaFunction: helloFunction
  }).next(new sfn.Succeed(this, "GreetedWorld"))
});
```

JavaScript

```
const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
  definition: new tasks.LambdaInvoke(this, "MyLambdaTask", {
    lambdaFunction: helloFunction
  }).next(new sfn.Succeed(this, "GreetedWorld"))
});
```

Python

```
state_machine = sfn.StateMachine(
                                  self, "MyStateMachine",
                                  definition=tasks.LambdaInvoke(
                                  self, "MyLambdaTask",
                                  lambda_function=hello_function)
                                  .next(sfn.Succeed(self, "GreetedWorld")))
```

Java

```
final StateMachine stateMachine = StateMachine.Builder.create(this,
 "MyStateMachine")
        .definition(LambdaInvoke.Builder.create(this, "MyLambdaTask")
            .lambdaFunction(helloFunction)
            .build()
            .next(new Succeed(this, "GreetedWorld")))
```

```
        .build();
```

C#

```
var stateMachine = new StateMachine(this, "MyStateMachine", new StateMachineProps {
    DefinitionBody = DefinitionBody.FromChainable(new LambdaInvoke(this,
 "MyLambdaTask", new LambdaInvokeProps
    {
        LambdaFunction = helloFunction
    })
    .Next(new Succeed(this, "GreetedWorld")))
});
```

AWS CDK 앱을 빌드하고 배포하기

새롭게 만든 AWS CDK 프로젝트에서 스택 정의가 포함된 파일을 다음 예제와 비슷하게 편집합니다. 이전 섹션에서 살펴본 Lambda 함수와 Step Functions 상태 시스템의 정의를 이해합니다.

1. 다음 예제와 같이 스택을 업데이트합니다.

   TypeScript

   다음 코드로 `lib/step-stack.ts`를 업데이트합니다.

   ```
   import * as cdk from 'aws-cdk-lib';
   import * as lambda from 'aws-cdk-lib/aws-lambda';
   import * as sfn from 'aws-cdk-lib/aws-stepfunctions';
   import * as tasks from 'aws-cdk-lib/aws-stepfunctions-tasks';

   export class StepStack extends cdk.Stack {
     constructor(app: cdk.App, id: string) {
       super(app, id);

       const helloFunction = new lambda.Function(this, 'MyLambdaFunction', {
         code: lambda.Code.fromInline(`
             exports.handler = (event, context, callback) => {
                 callback(null, "Hello World!");
             };
         `),
         runtime: lambda.Runtime.NODEJS_18_X,
         handler: "index.handler",
         timeout: cdk.Duration.seconds(3)
   ```

```
    });

    const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
      definition: new tasks.LambdaInvoke(this, "MyLambdaTask", {
        lambdaFunction: helloFunction
      }).next(new sfn.Succeed(this, "GreetedWorld"))
    });
  }
}
```

JavaScript

다음 코드로 `lib/step-stack.js`를 업데이트합니다.

```
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as sfn from 'aws-cdk-lib/aws-stepfunctions';
import * as tasks from 'aws-cdk-lib/aws-stepfunctions-tasks';

export class StepStack extends cdk.Stack {
  constructor(app, id) {
    super(app, id);

    const helloFunction = new lambda.Function(this, 'MyLambdaFunction', {
      code: lambda.Code.fromInline(`
          exports.handler = (event, context, callback) => {
              callback(null, "Hello World!");
          };
      `),
      runtime: lambda.Runtime.NODEJS_18_X,
      handler: "index.handler",
      timeout: cdk.Duration.seconds(3)
    });

    const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
      definition: new tasks.LambdaInvoke(this, "MyLambdaTask", {
        lambdaFunction: helloFunction
      }).next(new sfn.Succeed(this, "GreetedWorld"))
    });
  }
}
```

Python

다음 코드로 `step/step_stack.py`를 업데이트합니다.

```
from aws_cdk import (
    Duration,
    Stack,
    aws_stepfunctions as sfn,
    aws_stepfunctions_tasks as tasks,
    aws_lambda as lambda_
)
class StepStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        hello_function = lambda_.Function(
            self, "MyLambdaFunction",
            code=lambda_.Code.from_inline("""
            exports.handler = (event, context, callback) => {
                callback(null, "Hello World!");
                }"""),
                runtime=lambda_.Runtime.NODEJS_18_X,
                handler="index.handler",
                timeout=Duration.seconds(25))

        state_machine = sfn.StateMachine(
            self, "MyStateMachine",
            definition=tasks.LambdaInvoke(
            self, "MyLambdaTask",
            lambda_function=hello_function)
            .next(sfn.Succeed(self, "GreetedWorld")))
```

Java

다음 코드로 `src/main/java/com.myorg/StepStack.java`를 업데이트합니다.

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
```

```
import software.amazon.awscdk.Duration;
import software.amazon.awscdk.services.lambda.Code;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.stepfunctions.StateMachine;
import software.amazon.awscdk.services.stepfunctions.Succeed;
import software.amazon.awscdk.services.stepfunctions.tasks.LambdaInvoke;

public class StepStack extends Stack {
    public StepStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public StepStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        final Function helloFunction = Function.Builder.create(this,
 "MyLambdaFunction")
                .code(Code.fromInline(
                        "exports.handler = (event, context, callback) =>
 { callback(null, 'Hello World!' );}"))
                .runtime(Runtime.NODEJS_18_X)
                .handler("index.handler")
                .timeout(Duration.seconds(25))
                .build();

        final StateMachine stateMachine = StateMachine.Builder.create(this,
 "MyStateMachine")
                .definition(LambdaInvoke.Builder.create(this, "MyLambdaTask")
                        .lambdaFunction(helloFunction)
                        .build()
                        .next(new Succeed(this, "GreetedWorld")))
                .build();
    }
}
```

C#

다음 코드로 scr/Step/StepStack.cs를 업데이트합니다.

```
using Amazon.CDK;
using Constructs;
```

```
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.StepFunctions;
using Amazon.CDK.AWS.StepFunctions.Tasks;

namespace Step
{
    public class StepStack : Stack
    {
        internal StepStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            var helloFunction = new Function(this, "MyLambdaFunction", new
 FunctionProps
            {
                Code = Code.FromInline(@"exports.handler = (event, context,
 callback) => {
                    callback(null, 'Hello World!');
                }"),
                Runtime = Runtime.NODEJS_18_X,
                Handler = "index.handler",
                Timeout = Duration.Seconds(25)
            });

            var stateMachine = new StateMachine(this, "MyStateMachine", new
 StateMachineProps
            {
                DefinitionBody = DefinitionBody.FromChainable(new
 LambdaInvoke(this, "MyLambdaTask", new LambdaInvokeProps
                {
                    LambdaFunction = helloFunction
                })
                .Next(new Succeed(this, "GreetedWorld")))
            });
        }
    }
}
```

2. 소스 파일을 저장한 다음 앱의 기본 디렉터리에서 `cdk synth` 명령을 실행합니다.

   AWS CDK에서 앱을 실행하고 앱에서 AWS CloudFormation 템플릿을 합성합니다. 그러면 AWS CDK에서 템플릿을 표시합니다.

> **Note**
>
> AWS CDK프로젝트를 만들 때 사용한 TypeScript 경우 `cdk synth` 명령을 실행하면 다음 오류가 반환될 수 있습니다.
>
> ```
> TSError: # Unable to compile TypeScript:
> bin/step.ts:7:33 - error TS2554: Expected 2 arguments, but got 3.
> ```
>
> 다음 예제와 같이 `bin/step.ts` 파일을 수정하여 이 오류를 해결합니다.
>
> ```
> #!/usr/bin/env node
> import 'source-map-support/register';
> import * as cdk from 'aws-cdk-lib';
> import { StepStack } from '../lib/step-stack';
>
> const app = new cdk.App();
> new StepStack(app, 'StepStack');
> app.synth();
> ```

3. Lambda 함수와 Step Functions 상태 시스템을 AWS 계정에 배포하려면 `cdk deploy`를 실행합니다. 생성한 IAM 정책을 승인하라는 메시지가 표시됩니다. AWS CDK

3단계: 상태 시스템 실행 시작

상태 시스템을 만든 후에 실행을 시작할 수 있습니다.

상태 시스템 실행을 시작하려면

1. Step Functions 콘솔을 열고 AWS CDK을 사용하여 만든 상태 시스템의 이름을 선택합니다.
2. 상태 시스템 페이지에서 실행 시작을 선택합니다.

   실행 시작 대화 상자가 표시됩니다.
3. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

4. 실행 시작을 선택합니다.

   상태 시스템 실행이 시작되고 실행 중인 실행을 보여주는 새로운 페이지가 표시됩니다.

5. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

4단계: 정리

상태 시스템을 테스트한 후에는 상태 시스템과 관련 Lambda 함수를 모두 제거하여 AWS 계정에서 리소스를 확보하는 것이 좋습니다. 앱의 기본 디렉터리에서 `cdk destroy` 명령을 실행하여 상태 시스템을 제거합니다.

다음 단계

를 사용하여 AWS 인프라를 개발하는 방법에 대해 자세히 AWS CDK 알아보려면 AWS CDK개발자 안내서를 참조하십시오.

선택한 언어로 AWS CDK 앱을 작성하는 방법에 대한 내용은 다음을 참조하세요.

TypeScript

AWS CDK에서 작업하기 TypeScript

JavaScript

AWS CDK에서 작업하기 JavaScript

Python

Python에서 AWS CDK 사용

Java

Java에서 AWS CDK 사용

C#

C#에서 AWS CDK 사용

이 튜토리얼에서 사용되는 AWS 구조 라이브러리 모듈에 대한 자세한 내용은 다음 AWS CDK API 참조 개요를 참조하세요.

- aws-lambda
- aws-stepfunctions
- aws-stepfunctions-tasks

를 사용하여 동기식 익스프레스 스테이트 머신을 사용하여 API Gateway REST API 만들기 AWS CDK

이 자습서에서는 AWS Cloud Development Kit (AWS CDK)를 사용하여 백엔드 통합으로 동기 Express 상태 시스템이 있는 API Gateway REST API를 만드는 방법을 보여줍니다. 이 자습서에서는 `StepFunctionsRestApi` 구조를 사용하여 상태 시스템을 API Gateway에 연결합니다. `StepFunctionsRestApi` 구조는 필수 권한 및 HTTP “ANY” 메서드와 함께 기본 입력/출력 매핑과 API Gateway REST API를 설정합니다. 완전한 AWS CDK 프로그래밍 언어를 사용하여 인프라를 정의할 수 있는 코드형 AWS 인프라 (IAC) 프레임워크입니다. 하나 이상의 스택이 포함된 CDK 지원 언어 중 하나로 앱을 작성한 다음 AWS CloudFormation 템플릿에 합성하여 계정에 배포합니다. AWS 이를 사용하여 동기식 익스프레스 스테이트 머신과 백엔드로 통합되는 API Gateway REST API를 정의한 다음, 를 사용하여 실행을 시작하겠습니다. AWS Management Console

이 자습서를 시작하기 전에 시작하기 AWS CDK - 사전 요구 사항에 설명된 대로 AWS CDK 개발 환경을 설정한 다음 다음을 실행하여 설치하십시오. AWS CDK

```
npm install -g aws-cdk
```

주제

1단계: AWS CDK 프로젝트 설정

먼저 새 AWS CDK 앱을 위한 디렉터리를 만들고 프로젝트를 초기화합니다.

TypeScript

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language typescript
```

JavaScript

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language javascript
```

Python

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language python
```

프로젝트가 초기화된 후 프로젝트의 가상 환경을 활성화하고 기본 AWS CDK 종속 항목을 설치합니다.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language java
```

C#

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language csharp
```

Go

```
mkdir stepfunctions-rest-api
cd stepfunctions-rest-api
cdk init --language go
```

> **Note**
>
> 디렉터리 이름을 `stepfunctions-rest-api`으로 지정해야 합니다. AWS CDK 응용 프로그램 템플릿은 디렉토리 이름을 사용하여 소스 파일 및 클래스의 이름을 생성합니다. 다른 이름을 사용하는 경우 앱이 이 자습서와 일치하지 않습니다.

이제 Amazon API Gateway용 AWS Step Functions 구성 라이브러리 모듈을 설치합니다.

TypeScript

```
npm install @aws-cdk/aws-stepfunctions @aws-cdk/aws-apigateway
```

JavaScript

```
npm install @aws-cdk/aws-stepfunctions @aws-cdk/aws-apigateway
```

Python

```
python -m pip install aws-cdk.aws-stepfunctions
python -m pip install aws-cdk.aws-apigateway
```

Java

프로젝트의 `pom.xml`을 편집하여 기존 `<dependencies>` 컨테이너 내에 다음 종속성을 추가합니다.

```
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>stepfunctions</artifactId>
            <version>${cdk.version}</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>apigateway</artifactId>
            <version>${cdk.version}</version>
        </dependency>
```

Maven은 다음에 앱을 구축할 때 이러한 종속성을 자동으로 설치합니다. 구축하려면 `mvn compile`을 실행하거나 Java IDE의 Build 명령을 사용합니다.

C#

```
dotnet add src/StepfunctionsRestApi package Amazon.CDK.AWS.Stepfunctions
dotnet add src/StepfunctionsRestApi package Amazon.CDK.AWS.APIGateway
```

도구 > 패키지 관리자 > NuGet 솔루션용 패키지 관리를 통해 제공되는 Visual Studio NuGet GUI를 사용하여 표시된 NuGet 패키지를 설치할 수도 있습니다.

모듈을 설치한 후에는 다음 패키지를 가져와서 AWS CDK 앱에서 모듈을 사용할 수 있습니다.

TypeScript

```
@aws-cdk/aws-stepfunctions
@aws-cdk/aws-apigateway
```

JavaScript

```
@aws-cdk/aws-stepfunctions
@aws-cdk/aws-apigateway
```

Python

```
aws_cdk.aws_stepfunctions
aws_cdk.aws_apigateway
```

Java

```
software.amazon.awscdk.services.apigateway.StepFunctionsRestApi
software.amazon.awscdk.services.stepfunctions.Pass
software.amazon.awscdk.services.stepfunctions.StateMachine
software.amazon.awscdk.services.stepfunctions.StateMachineType
```

C#

```
Amazon.CDK.AWS.StepFunctions
Amazon.CDK.AWS.APIGateway
```

Go

다음을 `stepfunctions-rest-api.go` 내 `import`에 추가합니다.

```
"github.com/aws/aws-cdk-go/awscdk/awsapigateway"
"github.com/aws/aws-cdk-go/awscdk/awsstepfunctions"
```

2단계: AWS CDK 를 사용하여 동기식 익스프레스 스테이트 머신 백엔드 통합이 포함된 API Gateway REST API를 생성합니다.

먼저 동기식 익스프레스 스테이트 머신과 API Gateway REST API를 정의하는 개별 코드를 제시한 다음, 이를 AWS CDK 앱에 통합하는 방법을 설명하겠습니다. 그런 다음 이러한 리소스를 합성하고 배포하는 방법을 알아보겠습니다.

> **Note**
>
> 여기에 표시되는 상태 시스템은 `Pass` 상태가 있는 단순한 상태 시스템입니다.

Express 상태 시스템 만들기

이 AWS CDK 코드는 상태가 있는 간단한 상태 머신을 정의하는 코드입니다`Pass`.

TypeScript

```
const machineDefinition = new stepfunctions.Pass(this, 'PassState', {
    result: {value:"Hello!"},
})
```

```
const stateMachine = new stepfunctions.StateMachine(this, 'MyStateMachine', {
    definition: machineDefinition,
    stateMachineType: stepfunctions.StateMachineType.EXPRESS,
});
```

JavaScript

```
const machineDefinition = new sfn.Pass(this, 'PassState', {
    result: {value:"Hello!"},
})

const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
    definition: machineDefinition,
    stateMachineType: stepfunctions.StateMachineType.EXPRESS,
});
```

Python

```
machine_definition = sfn.Pass(self,"PassState",
                        result = sfn.Result("Hello"))

state_machine = sfn.StateMachine(self, 'MyStateMachine',
        definition = machine_definition,
        state_machine_type = sfn.StateMachineType.EXPRESS)
```

Java

```
Pass machineDefinition = Pass.Builder.create(this, "PassState")
                        .result(Result.fromString("Hello"))
                        .build();

StateMachine stateMachine = StateMachine.Builder.create(this, "MyStateMachine")
                            .definition(machineDefinition)
                            .stateMachineType(StateMachineType.EXPRESS)
                            .build();
```

C#

```
var machineDefinition = new Pass(this, "PassState", new PassProps
{
    Result = Result.FromString("Hello")
```

```
});

var stateMachine = new StateMachine(this, "MyStateMachine", new StateMachineProps
{
    Definition = machineDefinition,
    StateMachineType = StateMachineType.EXPRESS
});
```

Go

```
var machineDefinition = awsstepfunctions.NewPass(stack, jsii.String("PassState"),
 &awsstepfunctions.PassProps
{
    Result: awsstepfunctions.NewResult(jsii.String("Hello")),
})

var stateMachine = awsstepfunctions.NewStateMachine(stack,
 jsii.String("StateMachine"), &awsstepfunctions.StateMachineProps
{
    Definition: machineDefinition,
    StateMachineType: awsstepfunctions.StateMachineType_EXPRESS,
})
```

이 짧은 조각에서 확인할 수 있는 항목은 다음과 같습니다.

- `PassState` 시스템 정의(Pass 상태)
- 상태 시스템의 논리명(`MyStateMachine`)
- 상태 시스템 정의로 사용되는 시스템 정의
- `StepFunctionsRestApi`에서 동기 Express 상태 시스템만 허용하기 때문에 `EXPRESS`로 설정된 상태 시스템 유형

`StepFunctionsRestApi` 구성을 사용하여 API Gateway REST API 만들기

`StepFunctionsRestApi` 구성을 사용하여 필요한 권한과 기본 입력/출력 매핑이 있는 API Gateway REST API를 만듭니다.

TypeScript

```
const api = new apigateway.StepFunctionsRestApi(this,
```

```
  'StepFunctionsRestApi', { stateMachine: stateMachine });
```

JavaScript

```
const api = new apigateway.StepFunctionsRestApi(this,
  'StepFunctionsRestApi', { stateMachine: stateMachine });
```

Python

```
api = apigw.StepFunctionsRestApi(self, "StepFunctionsRestApi",
                             state_machine = state_machine)
```

Java

```
StepFunctionsRestApi api = StepFunctionsRestApi.Builder.create(this,
 "StepFunctionsRestApi")
                          .stateMachine(stateMachine)
                          .build();
```

C#

```
var api = new StepFunctionsRestApi(this, "StepFunctionsRestApi", new
 StepFunctionsRestApiProps
{
    StateMachine = stateMachine
});
```

Go

```
awsapigateway.NewStepFunctionsRestApi(stack, jsii.String("StepFunctionsRestApi"),
 &awsapigateway.StepFunctionsRestApiProps
{
    StateMachine = stateMachine,
})
```

AWS CDK 앱을 빌드하고 배포하기

만든 AWS CDK 프로젝트에서 스택의 정의가 들어 있는 파일을 아래 코드와 같이 편집합니다. 위에서 Step Functions 상태 시스템과 API Gateway의 정의를 이해합니다.

TypeScript

`lib/stepfunctions-rest-api-stack.ts`를 업데이트하여 다음을 확인합니다.

```
import * as cdk from 'aws-cdk-lib';
import * as stepfunctions from 'aws-cdk-lib/aws-stepfunctions'
import * as apigateway from 'aws-cdk-lib/aws-apigateway';


export class StepfunctionsRestApiStack extends cdk.Stack {
    constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const machineDefinition = new stepfunctions.Pass(this, 'PassState', {
        result: {value:"Hello!"},
    });

    const stateMachine = new stepfunctions.StateMachine(this, 'MyStateMachine', {
        definition: machineDefinition,
        stateMachineType: stepfunctions.StateMachineType.EXPRESS,
    });

    const api = new apigateway.StepFunctionsRestApi(this,
        'StepFunctionsRestApi', { stateMachine: stateMachine });
```

JavaScript

`lib/stepfunctions-rest-api-stack.js`를 업데이트하여 다음을 확인합니다.

```
const cdk = require('@aws-cdk/core');
const stepfunctions = require('@aws-cdk/aws-stepfunctions');
const apigateway = require('@aws-cdk/aws-apigateway');


class StepfunctionsRestApiStack extends cdk.Stack {
    constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const machineDefinition = new stepfunctions.Pass(this, "PassState", {
        result: {value:"Hello!"},
    })

    const stateMachine = new sfn.StateMachine(this, 'MyStateMachine', {
        definition: machineDefinition,
```

```
        stateMachineType: stepfunctions.StateMachineType.EXPRESS,
    });

    const api = new apigateway.StepFunctionsRestApi(this,
        'StepFunctionsRestApi', { stateMachine: stateMachine });

    }
}

module.exports = { StepStack }
```

Python

`stepfunctions_rest_api/stepfunctions_rest_api_stack.py`를 업데이트하여 다음을 확인합니다.

```
from aws_cdk import App, Stack
from constructs import Construct
from aws_cdk import aws_stepfunctions as sfn
from aws_cdk import aws_apigateway as apigw

class StepfunctionsRestApiStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)


        machine_definition = sfn.Pass(self,"PassState",
                                result = sfn.Result("Hello"))

        state_machine = sfn.StateMachine(self, 'MyStateMachine',
                definition = machine_definition,
                state_machine_type = sfn.StateMachineType.EXPRESS)

        api = apigw.StepFunctionsRestApi(self,
                    "StepFunctionsRestApi",
                    state_machine = state_machine)
```

Java

`src/main/java/com.myorg/StepfunctionsRestApiStack.java`를 업데이트하여 다음을 확인합니다.

```
package com.myorg;


import software.amazon.awscdk.core.Construct;
import software.amazon.awscdk.core.Stack;
import software.amazon.awscdk.core.StackProps;
import software.amazon.awscdk.services.stepfunctions.Pass;
import software.amazon.awscdk.services.stepfunctions.StateMachine;
import software.amazon.awscdk.services.stepfunctions.StateMachineType;
import software.amazon.awscdk.services.apigateway.StepFunctionsRestApi;

public class StepfunctionsRestApiStack extends Stack {
    public StepfunctionsRestApiStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public StepfunctionsRestApiStack(final Construct scope, final String id, final
 StackProps props) {
        super(scope, id, props);

        Pass machineDefinition = Pass.Builder.create(this, "PassState")
                                .result(Result.fromString("Hello"))
                                .build();

        StateMachine stateMachine = StateMachine.Builder.create(this,
 "MyStateMachine")
                                    .definition(machineDefinition)
                                    .stateMachineType(StateMachineType.EXPRESS)
                                    .build();

        StepFunctionsRestApi api = StepFunctionsRestApi.Builder.create(this,
 "StepFunctionsRestApi")
                                   .stateMachine(stateMachine)
                                   .build();

    }
}
```

C#

src/StepfunctionsRestApi/StepfunctionsRestApiStack.cs를 업데이트하여 다음을 확인합니다.

```
using Amazon.CDK;
using Amazon.CDK.AWS.StepFunctions;
using Amazon.CDK.AWS.APIGateway;

namespace StepfunctionsRestApi
{
    public class StepfunctionsRestApiStack : Stack
    {
        internal StepfunctionsRestApi(Construct scope, string id, IStackProps props
 = null) : base(scope, id, props)
        {
            var machineDefinition = new Pass(this, "PassState", new PassProps
            {
                Result = Result.FromString("Hello")
            });

            var stateMachine = new StateMachine(this, "MyStateMachine", new
 StateMachineProps
            {
                Definition = machineDefinition,
                StateMachineType = StateMachineType.EXPRESS
            });

            var api = new StepFunctionsRestApi(this, "StepFunctionsRestApi", new
 StepFunctionsRestApiProps
            {
                StateMachine = stateMachine
            });

        }
    }
}
```

Go

`stepfunctions-rest-api.go`를 업데이트하여 다음을 확인합니다.

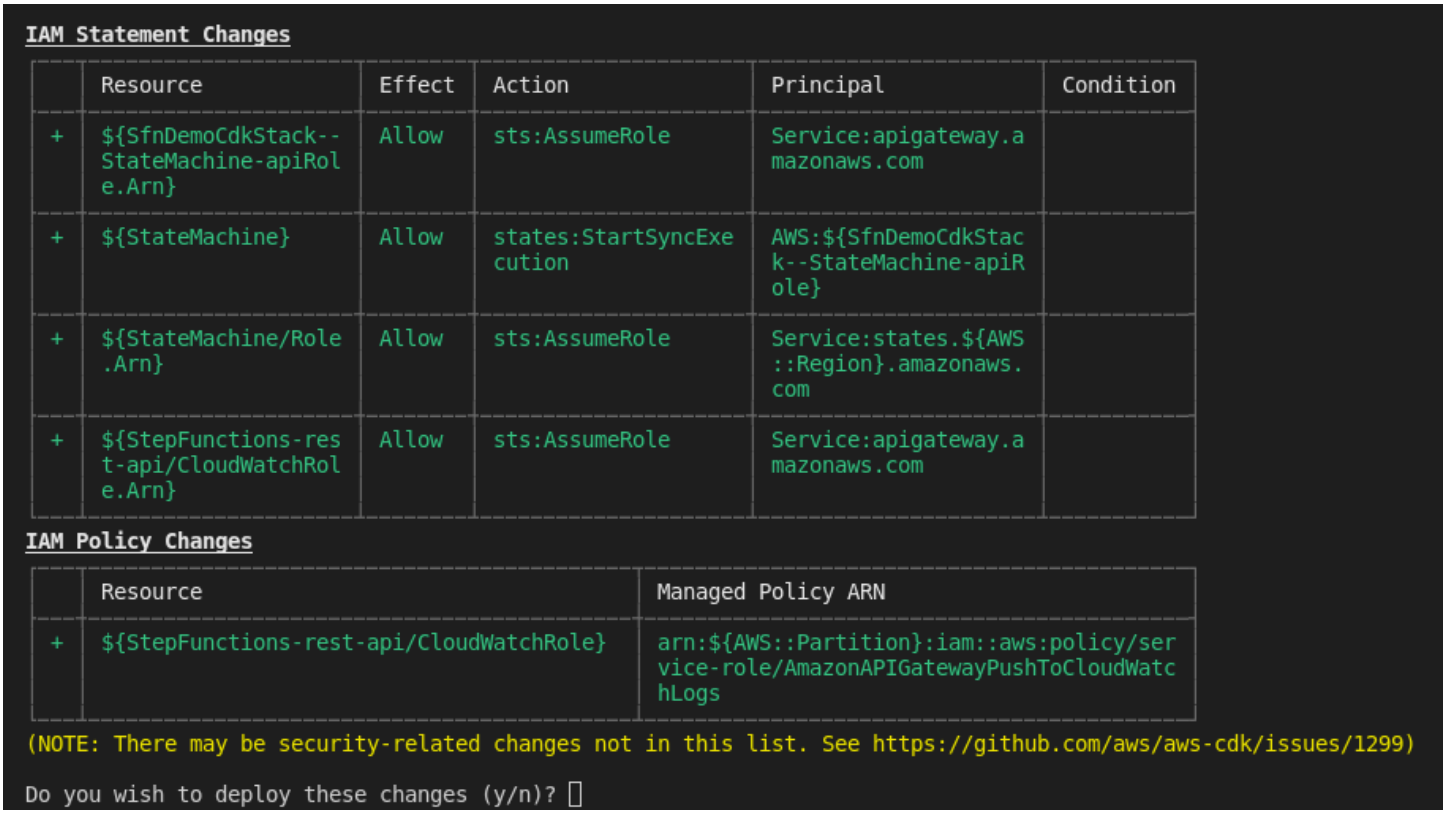
```
package main
import (
    "github.com/aws/aws-cdk-go/awscdk"
    "github.com/aws/aws-cdk-go/awscdk/awsapigateway"
    "github.com/aws/aws-cdk-go/awscdk/awsstepfunctions"
    "github.com/aws/constructs-go/constructs/v3"
```

```
    "github.com/aws/jsii-runtime-go"
)


type StepfunctionsRestApiGoStackProps struct {
    awscdk.StackProps
}

func NewStepfunctionsRestApiGoStack(scope constructs.Construct, id string, props
 *StepfunctionsRestApiGoStackProps) awscdk.Stack {
    var sprops awscdk.StackProps
    if props != nil {
        sprops = props.StackProps
    }
    stack := awscdk.NewStack(scope, &id, &sprops)

    // The code that defines your stack goes here
    var machineDefinition = awsstepfunctions.NewPass(stack,
 jsii.String("PassState"), &awsstepfunctions.PassProps
    {
        Result: awsstepfunctions.NewResult(jsii.String("Hello")),
    })

    var stateMachine = awsstepfunctions.NewStateMachine(stack,
 jsii.String("StateMachine"), &awsstepfunctions.StateMachineProps{
        Definition: machineDefinition,
        StateMachineType: awsstepfunctions.StateMachineType_EXPRESS,
    });

    awsapigateway.NewStepFunctionsRestApi(stack,
 jsii.String("StepFunctionsRestApi"), &awsapigateway.StepFunctionsRestApiProps{
        StateMachine = stateMachine,
    })

    return stack
}

func main() {
    app := awscdk.NewApp(nil)

    NewStepfunctionsRestApiGoStack(app, "StepfunctionsRestApiGoStack",
 &StepfunctionsRestApiGoStackProps{
        awscdk.StackProps{
            Env: env(),
```

```
        },
    })

    app.Synth(nil)
}

// env determines the AWS environment (account+region) in which our stack is to
// be deployed. For more information see: https://docs.aws.amazon.com/cdk/latest/
guide/environments.html
func env() *awscdk.Environment {
    // If unspecified, this stack will be "environment-agnostic".
    // Account/Region-dependent features and context lookups will not work, but a
    // single synthesized template can be deployed anywhere.
    //---------------------------------------------------------------------------
    return nil

    // Uncomment if you know exactly what account and region you want to deploy
    // the stack to. This is the recommendation for production stacks.
    //---------------------------------------------------------------------------
    // return &awscdk.Environment{
    //  Account: jsii.String("123456789012"),
    //  Region:  jsii.String("us-east-1"),
    // }

    // Uncomment to specialize this stack for the AWS Account and Region that are
    // implied by the current CLI configuration. This is recommended for dev
    // stacks.
    //---------------------------------------------------------------------------
    // return &awscdk.Environment{
    //  Account: jsii.String(os.Getenv("CDK_DEFAULT_ACCOUNT")),
    //  Region:  jsii.String(os.Getenv("CDK_DEFAULT_REGION")),
    // }
}
```

소스 파일을 저장한 다음 앱의 기본 디렉터리에서 `cdk synth`를 실행합니다. 는 앱을 AWS CDK 실행하고 앱을 기반으로 AWS CloudFormation 템플릿을 합성한 다음 템플릿을 표시합니다.

Amazon API Gateway와 AWS Step Functions 상태 머신을 AWS 계정에 실제로 배포하려면 다음을 실행하십시오`cdk deploy`. 생성한 IAM 정책을 승인하라는 요청을 받게 됩니다 AWS CDK . 생성되는 정책은 다음과 비슷합니다.



IAM Statement Changes

	Resource	Effect	Action	Principal	Condition
+	${SfnDemoCdkStack--StateMachine-apiRole.Arn}	Allow	sts:AssumeRole	Service:apigateway.amazonaws.com	
+	${StateMachine}	Allow	states:StartSyncExecution	AWS:${SfnDemoCdkStack--StateMachine-apiRole}	
+	${StateMachine/Role.Arn}	Allow	sts:AssumeRole	Service:states.${AWS::Region}.amazonaws.com	
+	${StepFunctions-rest-api/CloudWatchRole.Arn}	Allow	sts:AssumeRole	Service:apigateway.amazonaws.com	

IAM Policy Changes

	Resource	Managed Policy ARN
+	${StepFunctions-rest-api/CloudWatchRole}	arn:${AWS::Partition}:iam::aws:policy/service-role/AmazonAPIGatewayPushToCloudWatchLogs

(NOTE: There may be security-related changes not in this list. See https://github.com/aws/aws-cdk/issues/1299)

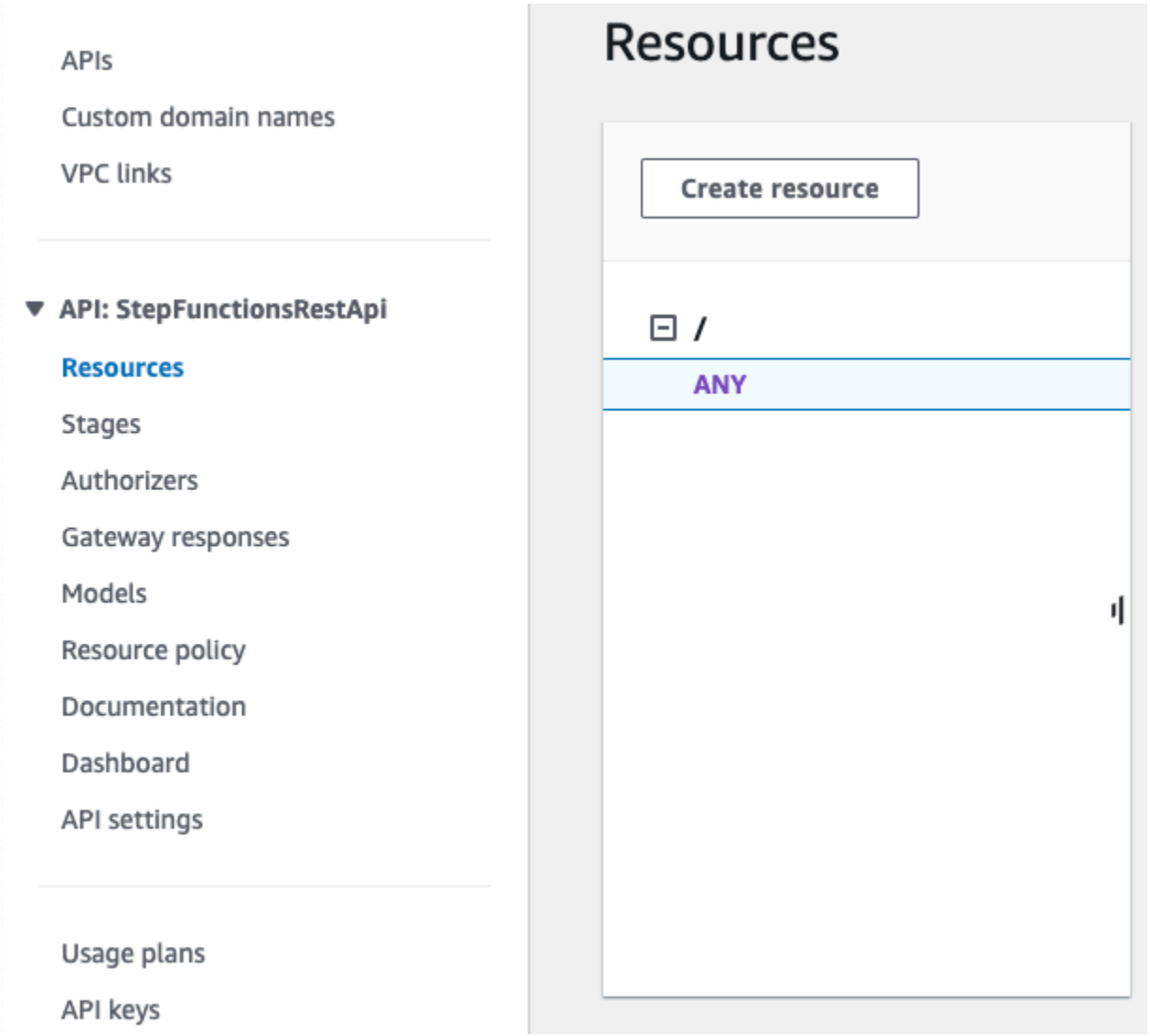
Do you wish to deploy these changes (y/n)?

3단계: API Gateway 테스트

동기 Express 상태 시스템이 백엔드 통합으로 있는 API Gateway REST API를 만든 후 API Gateway를 테스트할 수 있습니다.

API Gateway 콘솔을 사용하여 배포된 API Gateway 테스트하기

1. Amazon API Gateway 콘솔을 열고 로그인합니다.
2. 이름이 `StepFunctionsRestApi`인 REST API를 선택합니다.
3. 리소스 창에서 `ANY` 메서드를 선택합니다.



4. 테스트 탭을 선택합니다. 탭을 표시하려면 오른쪽 화살표 버튼을 선택해야 할 수도 있습니다.
5. Method(메서드)에서 POST를 선택합니다.
6. 요청 본문에서 다음 요청 파라미터를 복사합니다.

```
{
    "key": "Hello"
}
```

7. 테스트를 선택합니다. 다음 정보가 표시됩니다.

   - 요청은 메서드에 대해 호출된 리소스의 경로입니다.
   - 상태는 응답의 HTTP 상태 코드입니다.
   - 지연 시간은 호출자로부터 요청을 수신한 시간과 응답 반환 시간의 시간차입니다.

- 응답 본문은 HTTP 응답 본문입니다.
- 응답 헤더는 HTTP 응답 헤더입니다.
- 로그에는 이 메서드가 API Gateway 콘솔 외부에서 호출될 경우 작성되었을 시뮬레이션된 Amazon CloudWatch Logs 항목이 표시됩니다.

> **Note**
>
> CloudWatch 로그 항목이 시뮬레이션되기는 하지만 메서드 호출의 결과는 실제 결과입니다.

응답 본문 출력은 다음과 같아야 합니다.

```
"Hello"
```

> **Tip**
>
> 다른 메서드와 잘못된 입력으로 API Gateway를 사용해 보고 오류 출력을 확인합니다. 특정 키를 찾도록 상태 시스템을 변경하고 테스트 중에 잘못된 키를 입력하면 상태 머신 실행이 실패하고 응답 본문 출력에 오류 메시지가 생성될 수 있습니다.

cURL을 사용하여 배포된 API를 테스트하기

1. 터미널 창을 엽니다.
2. 다음 cURL 명령을 복사하여 터미널 창에 붙여 넣습니다. `<api-id>`를 해당 API의 API ID로 바꾸고, `<region>`을 해당 API가 배포된 리전으로 바꾸십시오.

```
curl -X POST\
 'https://<api-id>.execute-api.<region>.amazonaws.com/prod' \
 -d '{"key":"Hello"}' \
 -H 'Content-Type: application/json'
```

응답 본문 출력은 다음과 같아야 합니다.

```
"Hello"
```

> **Tip**
>
> 다른 메서드와 잘못된 입력으로 API Gateway를 사용해 보고 오류 출력을 확인합니다. 특정 키를 찾도록 상태 시스템을 변경하고 테스트 중에 잘못된 키를 입력하면 상태 시스템 실행이 실패하고 응답 본문 출력에 오류 메시지가 생성될 수 있습니다.

4단계: 정리

API Gateway 사용을 마치면 AWS CDK를 사용하여 상태 시스템과 API Gateway를 모두 제거할 수 있습니다. 앱의 기본 디렉토리에서 `cdk destroy`를 실행합니다.

AWS Step Functions Python용 데이터 사이언스 SDK

AWS Step Functions 데이터 과학 SDK는 데이터 과학자를 위한 오픈소스 라이브러리입니다. 이 SDK를 사용하면 및 Step Functions를 사용하여 SageMaker 기계 학습 모델을 처리하고 게시하는 워크플로를 만들 수 있습니다. 또한 AWS 서비스를 별도로 프로비저닝하고 통합할 필요 없이 Python으로 AWS 인프라를 대규모로 오케스트레이션하는 다단계 기계 학습 워크플로를 만들 수 있습니다.

AWS Step Functions Data Science SDK는 Step Functions 워크플로를 만들고 간접적으로 호출할 수 있는 Python API를 제공합니다. Python과 Jupyter Notebook에서 이러한 워크플로를 직접 관리하고 실행할 수 있습니다.

AWS Step Functions Data Science SDK를 사용하면 Python에서 바로 사용할 수 있는 워크플로를 만들 수 있을 뿐만 아니라 해당 워크플로를 복사하고 새 옵션을 실험한 다음 개선된 워크플로를 프로덕션에 적용할 수 있습니다.

AWS Step Functions 데이터 과학 SDK에 대한 자세한 내용은 다음을 참조하십시오.

- Github의 프로젝트
- SDK 설명서
- SageMaker 콘솔 및 관련 프로젝트의 Jupyter 노트북 인스턴스에서 사용할 수 있는 다음 예제 노트북은 다음과 같습니다. GitHub
  - `hello_world_workflow.ipynb`
  - `machine_learning_workflow_abalone.ipynb`
  - `training_pipeline_pytorch_mnist.ipynb`

Terraform을 사용하여 상태 시스템 배포

HashiCorp의 Terraform은 코드형 인프라(IaC)를 사용하여 애플리케이션을 빌드하기 위한 프레임워크입니다. Terraform을 사용하면 상태 시스템을 만들고 인프라 배포 미리 보기 및 재사용 가능한 템플릿 만들기와 같은 기능을 사용할 수 있습니다. Terraform 템플릿을 사용하면 코드를 더 작은 청크로 나눠 유지하고 재사용할 수 있습니다.

Terraform에 익숙하면 Terraform에서 상태 시스템을 만들고 배포하기 위한 모델로 이 주제에 설명된 개발 수명 주기를 따를 수 있습니다. Terraform에 익숙하지 않으면 Terraform에 익숙해질 수 있도록 먼저 AWS용 Terraform 소개 워크숍을 완료하는 것이 좋습니다.

> **Tip**
>
> Terraform을 사용하여 빌드한 상태 시스템의 예제를 AWS 계정에 배포하려면 AWS Step Functions 워크숍의 코드형 인프라를 사용하여 상태 시스템 관리 모듈을 참조하세요.

이 주제의 내용

필수 조건

시작하기 전에 다음 사전 조건을 완료해야 합니다.

- 시스템에 Terraform을 설치합니다. Terraform 설치 방법은은 Terraform 설치를 참조하세요.
- 시스템에 Step Functions Local을 설치합니다. Step Functions Local을 사용하려면 Step Functions Local 도커 이미지를 설치하는 것이 좋습니다. 자세한 내용은 상태 시스템을 로컬로 테스트 섹션을 참조하세요.
- AWS SAM CLI를 설치합니다. 자세한 내용은 AWS Serverless Application Model 개발자 가이드의 AWS SAM CLI 설치를 참조하세요.
- AWS Toolkit for Visual Studio Code를 설치하여 상태 시스템의 워크플로 다이어그램을 봅니다. 설치 정보는 AWS Toolkit for Visual Studio Code 사용 설명서의 AWS Toolkit for Visual Studio Code 설치를 참조하세요.

Terraform을 사용한 상태 시스템 개발 수명 주기

다음 절차에서는 Step Functions 콘솔의 Workflow Studio를 사용하여 빌드한 상태 시스템 프로토타입을 Terraform 및 AWS Toolkit for Visual Studio Code를 사용한 로컬 개발의 시작점으로 사용하는 방법을 설명합니다.

Terraform을 사용한 상태 시스템 개발을 설명하고 모범 사례를 자세히 보여주는 전체 예제를 보려면 Step Functions Terraform 프로젝트 작성 모범 사례를 참조하세요.

Terraform을 사용하여 상태 시스템 개발 수명 주기 시작하기

1. 다음 명령을 사용하여 새 Terraform 프로젝트를 부트스트랩합니다.

   ```
   terraform init
   ```

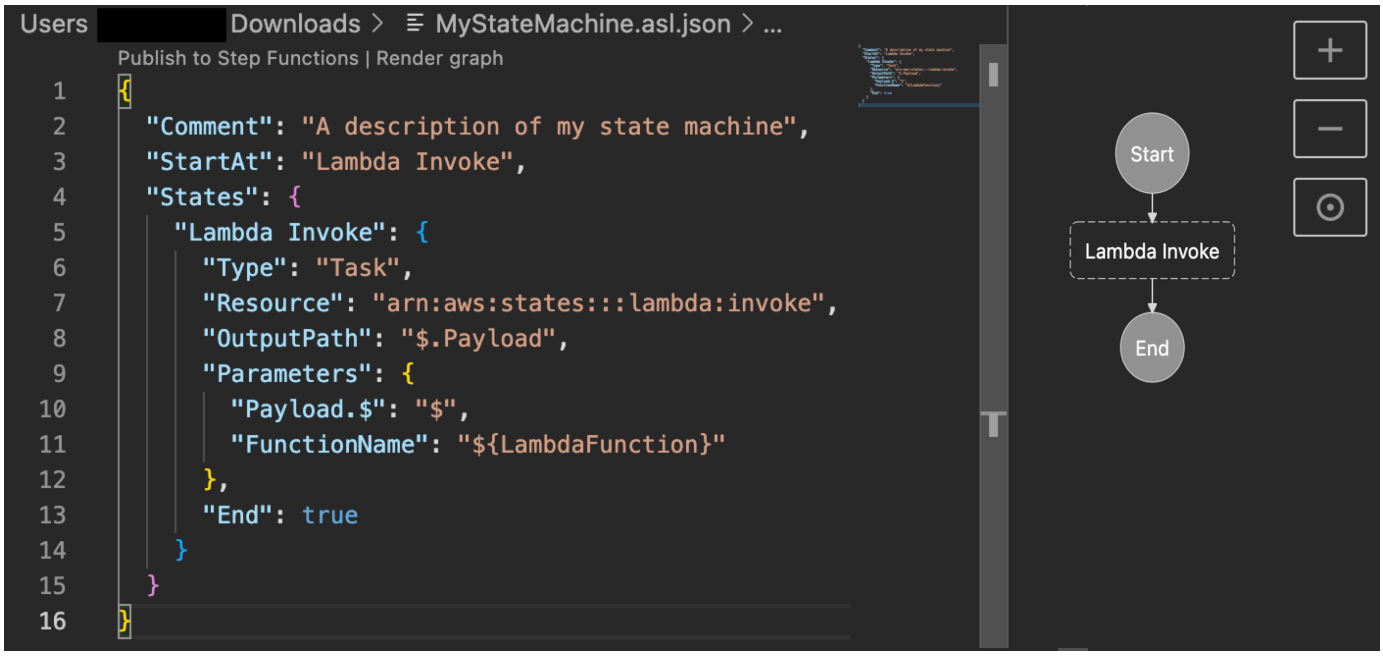
2. Step Functions 콘솔을 열어 상태 시스템의 프로토타입을 만듭니다.
3. Workflow Studio에서 다음을 수행합니다.

   a. 워크플로 프로토타입을 만듭니다.

   b. 워크플로의 Amazon States Language(ASL) 정의를 내보냅니다. 이렇게 하려면 Import/Export 드롭다운 목록을 선택한 다음 JSON 정의 내보내기를 선택합니다.
4. 내보낸 ASL 정의를 프로젝트 디렉토리에 저장합니다.

   내보낸 ASL 정의를 입력 파라미터로 `templatefile` 함수를 사용하는 `aws_sfn_state_machine` Terraform 리소스에 전달합니다. 이 함수는 내보낸 ASL 정의와 모든 변수 대체를 전달하는 정의 필드 내에서 사용됩니다.

   > **Tip**
   >
   > ASL 정의 파일에는 긴 텍스트 블록이 포함될 수 있으므로 인라인 EOF 메서드를 사용하지 않는 것이 좋습니다. 이렇게 하면 파라미터를 쉽게 상태 시스템 정의로 대체할 수 있습니다.
5. (선택 사항) IDE 내에서 ASL 정의를 업데이트하고 AWS Toolkit for Visual Studio Code를 사용하여 변경 사항을 시각화합니다.



정의를 계속 내보내고 프로젝트로 리팩토링하지 않으려면 IDE에서 로컬로 업데이트하고 Git을 사용하여 업데이트를 추적하는 것이 좋습니다.

6. Step Functions Local을 사용하여 워크플로를 테스트합니다.

> **Tip**
>
> 또한 AWS SAM CLI Local을 사용하여 상태 시스템의 Lambda 함수 및 API Gateway API와의 서비스 통합을 로컬에서 테스트할 수 있습니다.

7. 상태 시스템을 배포하기 전에 상태 시스템 및 기타 AWS 리소스를 미리 봅니다. 이를 위해 다음 명령을 실행합니다.

```
terraform plan
```

8. 다음 명령어를 사용하여 로컬 환경에서 또는 CI/CD 파이프라인을 통해 상태 시스템을 배포합니다.

```
terraform apply
```

9. (선택 사항) 다음 명령을 사용하여 리소스를 정리하고 상태 시스템을 삭제합니다.

```
terraform destroy
```

상태 시스템에 대한 IAM 역할 및 정책

Terraform 서비스 통합 정책을 사용하여 상태 시스템에 필요한 IAM 권한(예: Lambda 함수를 간접적으로 호출할 수 있는 권한)을 추가할 수 있습니다. 또한 명시적 역할과 정책을 정의하고 상태 시스템에 연결할 수 있습니다.

다음 IAM 정책 예제에서는 *myFunction* Lambda 함수를 간접적으로 호출할 수 있는 액세스 권한을 상태 시스템에 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:myFunction"
    }
  ]
}
```

또한 Terraform에서 상태 시스템에 대한 IAM 정책을 정의할 때 aws_iam_policy_document 데이터 소스를 사용하는 것이 좋습니다. 이렇게 하면 정책이 잘못되었는지 확인하고 리소스를 변수로 대체할 수 있습니다.

다음 IAM 정책 예제에서는 aws_iam_policy_document 데이터 소스를 사용하고 상태 시스템에 이라는 *myFunction* Lambda 함수를 간접적으로 호출할 수 있는 액세스 권한을 부여합니다.

```
data "aws_iam_policy_document" "state_machine_role_policy" {

  statement {
    effect = "Allow"

    actions = [
      "lambda:InvokeFunction"
    ]

    resources = ["${aws_lambda_function.[[myFunction]].arn}:*"]
  }
```

```
}
```

> **Tip**
>
> Terraform으로 배포된 고급 AWS 아키텍처 패턴을 자세히 보려면 Serverless Land Workflows Collection에서 Terraform 예제를 참조하세요.

테스트 및 디버깅

Step Functions는 상태 머신을 테스트하고 디버깅하는 다양한 방법을 제공합니다. 예를 들어 콘솔에서 상태 머신을 테스트 및 디버깅하거나, TestState API를 사용하여 개별 상태를 테스트하거나, Step Functions 로컬을 사용하여 상태 머신을 로컬에서 테스트할 수 있습니다.

TestState API를 사용하여 단일 상태에 대한 정의를 제공하고 이를 실행합니다. 상태 머신을 생성하거나 기존 상태 머신을 업데이트하지 않고도 단일 상태를 테스트할 수 있습니다.

Step Functions 로컬은 다운로드 가능한 Step Functions 버전으로, 이를 사용하면 자체 개발 환경에서 실행 중인 Step Functions 버전을 사용하여 애플리케이션을 개발하고 테스트할 수 있습니다. Step Functions 로컬을 사용하면 로컬 개발 환경에서 상태 머신을 실행하여 입력 및 출력 데이터 흐름, 지원되는 서비스와의 통합 등을 테스트할 수 있습니다.

주제

- TestState API를 사용하여 상태 테스트
- 상태 시스템을 로컬로 테스트

TestState API를 사용하여 상태 테스트

TestStateAPI는 단일 상태의 정의를 받아들이고 이를 실행합니다. 상태 머신을 생성하거나 기존 상태 머신을 업데이트하지 않고도 상태를 테스트할 수 있습니다.

TestState API를 사용하여 다음을 테스트할 수 있습니다.

- 상태의 입력 및 출력 처리 데이터 흐름.
- 다른 AWS 서비스 요청 및 응답과의 AWS 서비스 통합
- HTTP 태스크 요청 및 응답

상태를 테스트하기 위해 Step Functions 콘솔, AWS Command Line Interface (AWS CLI) 또는 SDK를 사용할 수도 있습니다.

`TestState` API는 IAM 역할을 수임하며, IAM 역할에는 해당 상태에서 액세스하는 리소스에 대한 필수 IAM 권한이 포함되어야 합니다. 필요할 수 있는 상태 권한을 알아보려면 IAM TestState API 사용 권한 섹션을 참조하세요.

주제

TestState API 사용에 대한 고려사항

TestStateAPI를 사용하면 한 번에 한 상태만 테스트할 수 있습니다. 테스트할 수 있는 상태는 다음과 같습니다.

- 활동을 제외한 모든 태스크 유형
- Pass
- Wait
- Choice
- Succeed
- Fail

`TestState` API를 사용하는 동안 다음 사항에 유의합니다.

- TestState API에는 다음에 대한 지원이 포함되어 있지 않습니다.
  - 다음 리소스 유형을 사용하는 태스크 상태 상태:
    - 활동
    - 서비스 통합 패턴(유형: `.sync`, `.waitForTaskToken`)
  - Parallel 상태
  - 맵 상태
- 테스트는 최대 5분 동안 실행할 수 있습니다. 테스트가 이 시간을 초과하면 `States.Timeout` 오류가 발생하여 테스트가 실패합니다.

TestState API의 검사 수준 사용

TestStateAPI를 사용하여 상태를 테스트하려면 해당 상태의 정의를 제공합니다. 그러면 테스트에서 출력이 반환됩니다. 각 상태에서는 테스트 결과에서 보려는 세부 정보의 양을 지정할 수 있습니다. 이 세부 정보를 통해 테스트 중인 상태에 대한 추가 정보를 얻을 수 있습니다. 예를 들어 상태에서 `InputPath`, `ResultPath` 등 입력 및 출력 데이터 처리 필터를 사용한 경우 중간 및 최종 데이터 처리 결과를 볼 수 있습니다.

Step Functions는 다음 레벨을 제공하므로 보려는 세부 정보를 지정할 수 있습니다.

- INFO
- DEBUG
- TRACE

이러한 모든 레벨은 `status` 및 `nextState` 필드도 반환합니다. `status`는 상태 실행 상태를 나타냅니다. 예로 `SUCCEEDED`, `FAILED`, `RETRIABLE`, `CAUGHT_ERROR`, `nextState`는 전환할 다음 상태의 이름을 나타냅니다. 정의에서 다음 상태를 정의하지 않은 경우 이 필드에는 빈 값이 반환됩니다.

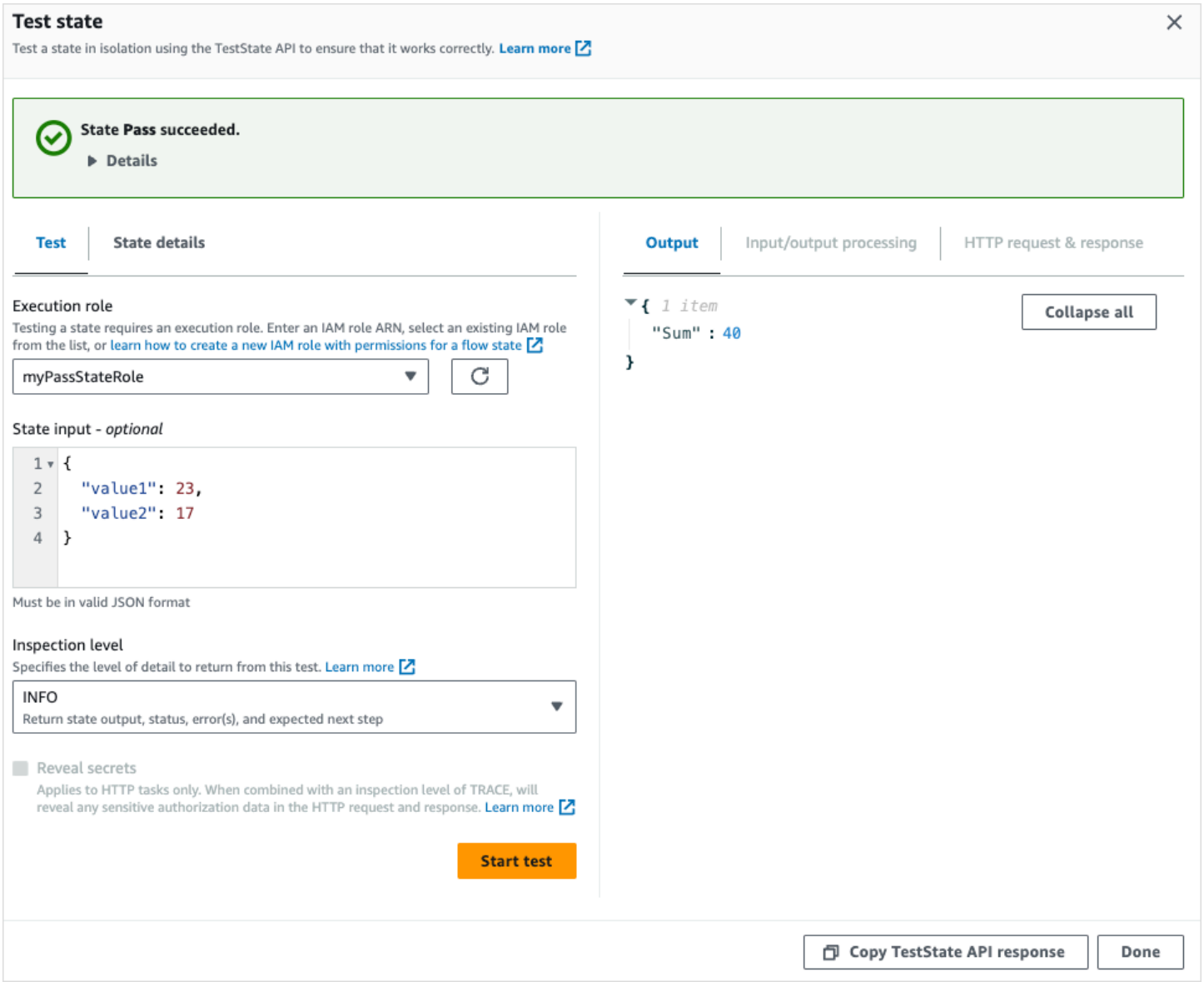
Step Functions 콘솔 및 AWS CLI에서 이러한 검사 레벨을 사용하여 상태를 테스트하는 방법에 대한 자세한 내용은 상태 테스트(콘솔) 및 AWS CLI를 사용하여 상태 테스트 섹션을 참조하세요.

INFO 검사 레벨

테스트가 성공하면 이 레벨에 상태 출력이 표시됩니다. 테스트에 실패한 경우 이 레벨에 오류 출력이 표시됩니다. 레벨을 지정하지 않은 경우 Step Functions는 기본적으로 검사 레벨을 INFO로 설정합니다.

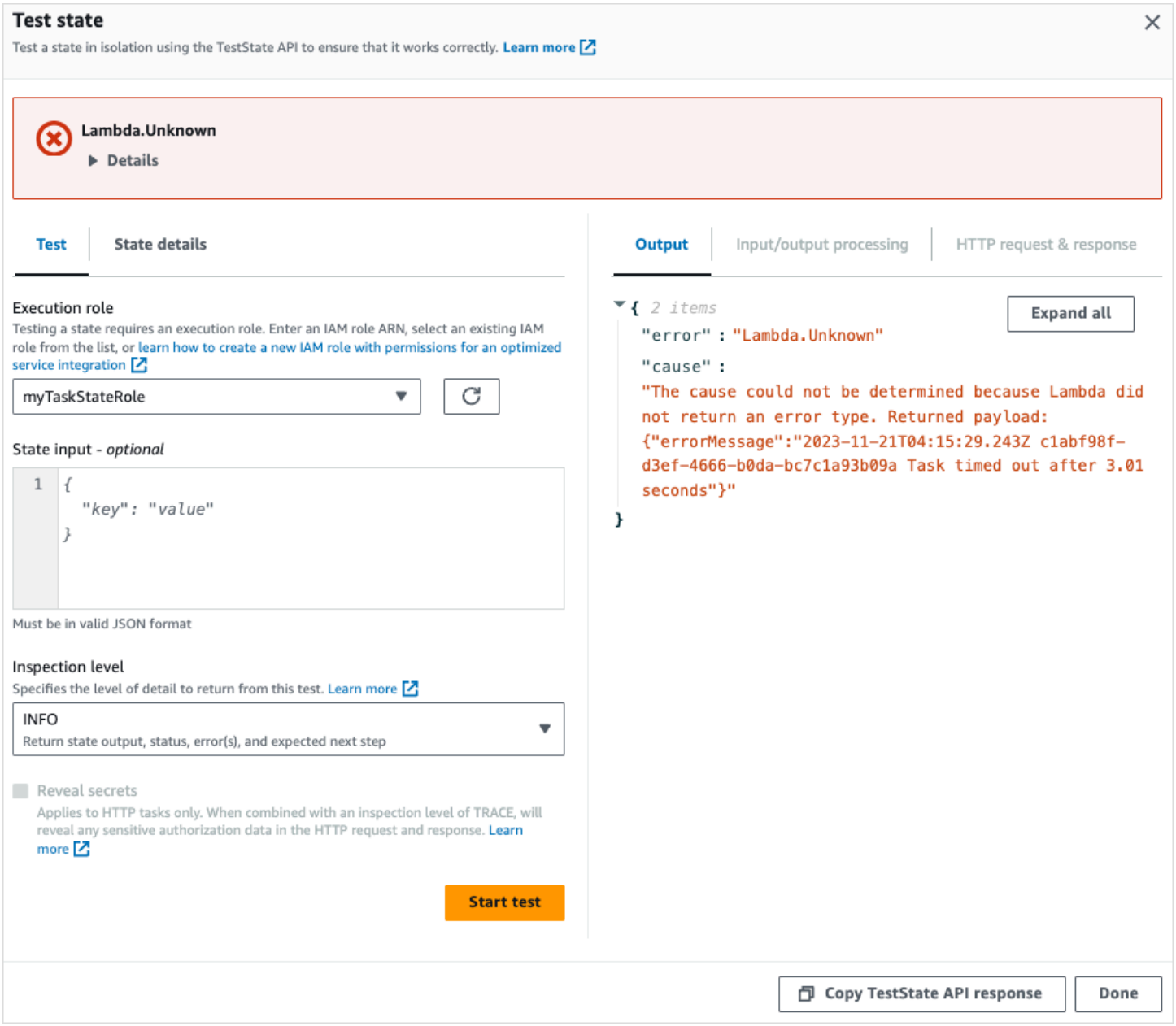
성공한 INFO 레벨 테스트의 예제

다음 이미지는 성공한 통과 상태 테스트를 보여 줍니다. 이 상태의 검사 레벨은 INFO로 설정되고 상태에 대한 출력이 출력 탭에 표시됩니다.



실패한 INFO 레벨 테스트의 예제

다음 이미지는 검사 레벨이 INFO로 설정된 경우 태스크 상태에서 실패한 테스트를 보여 줍니다. 출력 탭에는 오류 이름과 해당 오류의 원인에 대한 자세한 설명이 포함된 오류 출력이 표시됩니다.



DEBUG 검사 레벨

테스트가 성공하면 이 레벨에는 상태 출력과 입력 및 출력 데이터 처리 결과가 표시됩니다.

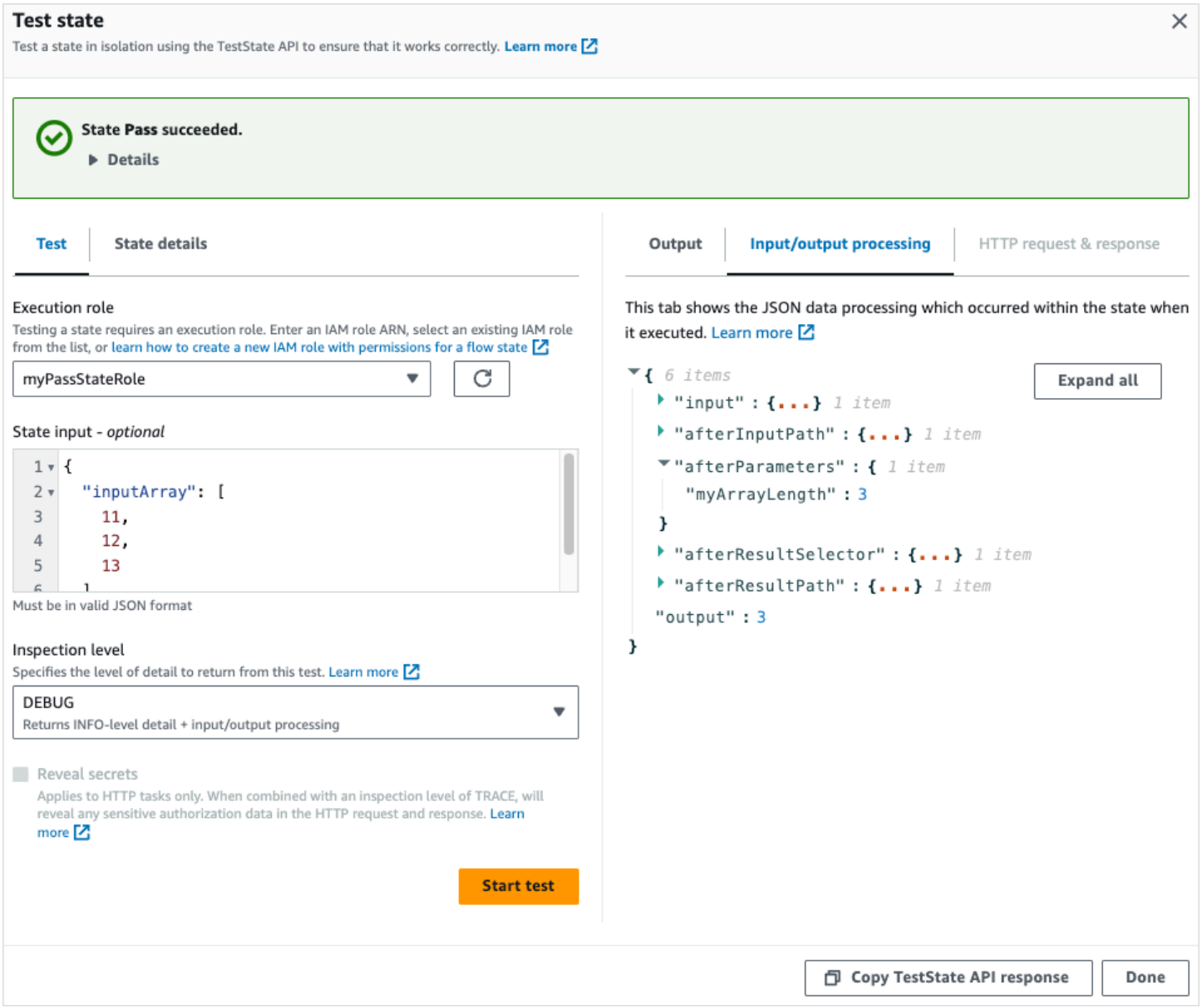
테스트에 실패한 경우 이 레벨에 오류 출력이 표시됩니다. 이 레벨은 실패 지점까지의 중간 데이터 처리 결과를 보여 줍니다. 예를 들어 Lambda 함수를 호출하는 태스크 상태를 테스트했다고 가정해 보겠습니다. InputPath, 파라미터, ResultPath, OutputPath 필터를 태스크 상태에 적용했다고 생각해 봅니다. 간접 호출이 실패했다고 가정해 보겠습니다. 이 경우 DEBUG 레벨에는 필터 적용에 따른 데이터 처리 결과가 다음과 같은 순서로 표시됩니다.

- `input` - 원시 상태 입력
- `afterInputPath` - Step Functions에서 `InputPath` 필터를 적용한 후 입력.
- `afterParameters` - Step Functions에서 `Parameters` 필터를 적용한 후의 유효 입력.

이 레벨에서 사용할 수 있는 진단 정보는 정의했을 수 있는 서비스 통합 또는 입/출력 데이터 처리 흐름과 관련된 문제를 해결하는 데 도움이 될 수 있습니다.

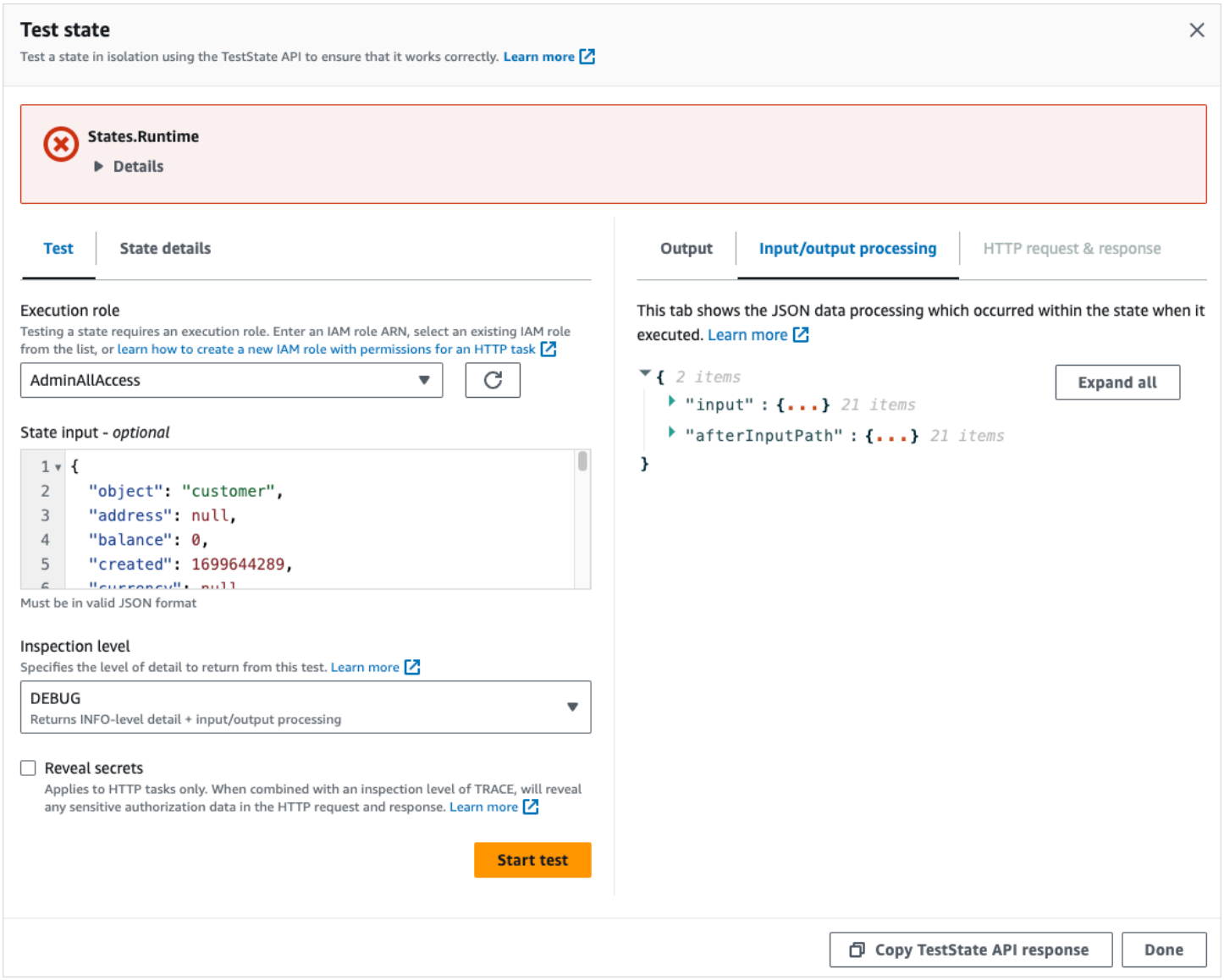
성공한 DEBUG 레벨 테스트의 예제

다음 이미지는 성공한 통과 상태 테스트를 보여 줍니다. 이 상태의 검사 레벨은 DEBUG로 설정되어 있습니다. 다음 이미지의 입력/출력 처리 탭은 이 상태에 제공된 입력에 대한 `Parameters`의 애플리케이션 결과를 보여 줍니다.



실패한 DEBUG 레벨 테스트의 예제

다음 이미지는 검사 레벨이 DEBUG로 설정된 경우 태스크 상태에서 실패한 테스트를 보여 줍니다. 다음 이미지의 입력/출력 처리 탭은 실패 시점까지의 상태에 대한 입력 및 출력 데이터 처리 결과를 보여 줍니다.



추적 검사 레벨

Step Functions는 HTTP 태스크를 테스트하기 위한 TRACE 레벨을 제공합니다. 이 레벨은 Step Functions가 수행하는 HTTP 요청과 타사 API가 반환하는 응답에 대한 정보를 반환합니다. 응답에는 헤더 및 요청 본문과 같은 정보가 포함될 수 있습니다. 또한 이 레벨에서 입력 및 출력 데이터 처리의 상태 출력과 결과를 볼 수 있습니다.

테스트에 실패한 경우 이 레벨에 오류 출력이 표시됩니다.

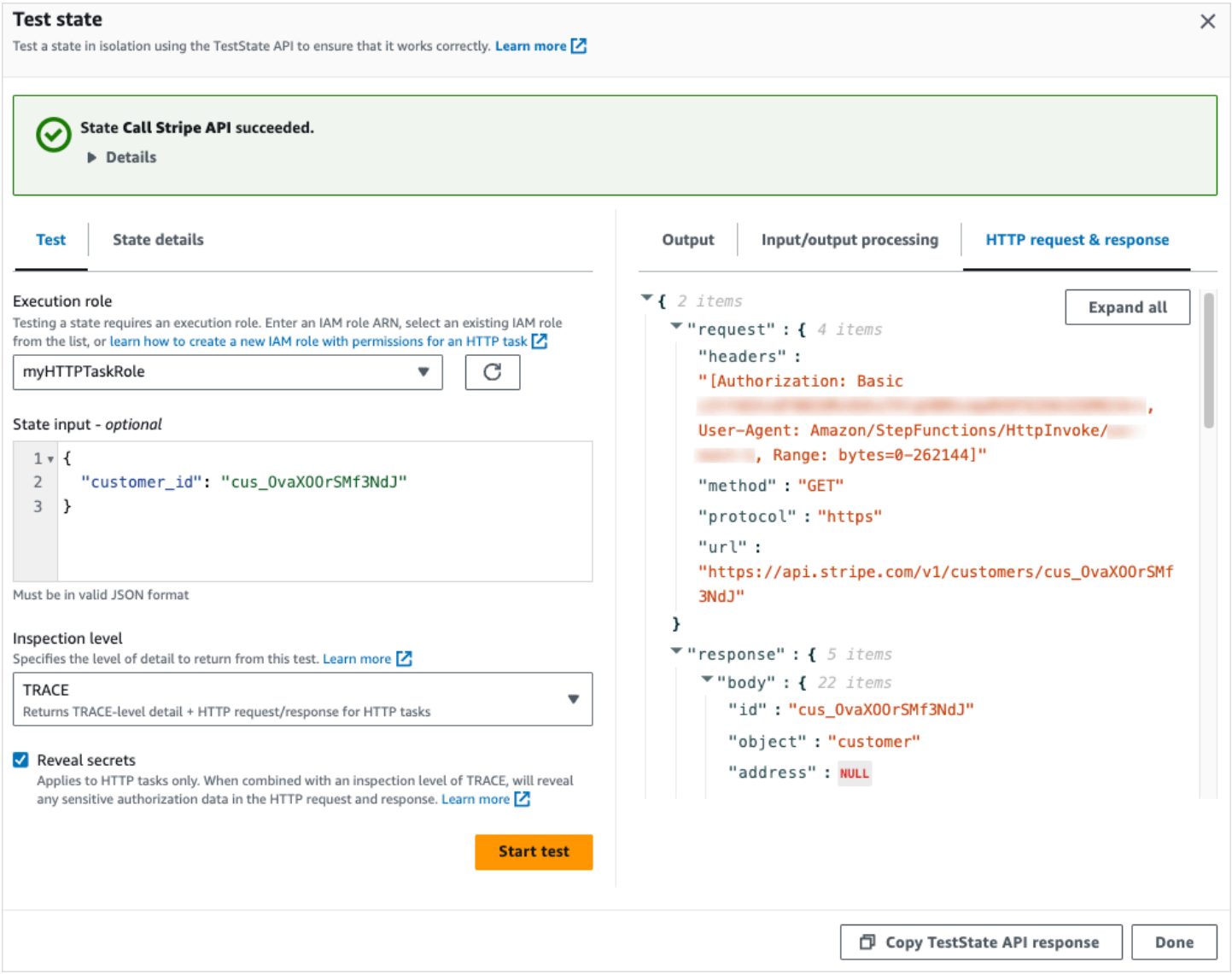
이 레벨은 HTTP 태스크에만 적용됩니다. Step Functions에서 이 레벨을 다른 상태 유형에 사용하면 오류가 발생합니다.

검사 수준을 TRACE로 설정하면 EventBridge 연결에 포함된 비밀도 볼 수 있습니다. 이렇게 하려면 `true` TestStateAPI에서 `revealSecrets` 파라미터를 로 설정해야 합니다. 또한 TestState API를 호출하는 IAM 사용자에게 `states:RevealSecrets` 작업에 대한 권한이 있는지 확인해야 합니다. `states:RevealSecrets` 권한을 설정하는 IAM 정책에 대한 예제는 IAM TestState API 사용 권한 섹션을 참조하세요. 이 권한이 없으면 Step Functions에서 액세스 거부 오류가 발생합니다.

`revealSecrets` 파라미터를 `false`로 설정하면 HTTP 요청 및 응답 데이터의 모든 암호가 생략됩니다.

성공한 TRACE 레벨 테스트의 예제

다음 이미지는 성공한 HTTP 태스크 테스트를 보여 줍니다. 이 상태의 검사 레벨은 TRACE로 설정되어 있습니다. 다음 이미지의 HTTP 요청 및 응답 탭은 타사 API 호출의 결과를 보여 줍니다.



IAM TestState API 사용 권한

`TestState` API를 호출하는 IAM 사용자에게 `states:TestState` 및 `iam:PassRole` 작업을 수행할 권한이 있어야 합니다. 또한 revealSecrets 파라미터를 `true`로 설정하는 경우 IAM 사용자에게 `states:RevealSecrets` 작업을 수행할 권한이 있는지 확인해야 합니다. 이 권한이 없으면 Step Functions에서 액세스 거부 오류가 발생합니다.

또한 실행 역할에 해당 상태에서 액세스 중인 리소스에 필요한 IAM 권한이 포함되어 있는지 확인해야 합니다. 상태에 필요할 수 있는 권한을 알아보려면 실행 역할 관리 섹션을 참조하세요.

다음 IAM 정책 예제는 `states:TestState`, `iam:PassRole`, `states:RevealSecrets` 권한을 설정합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:TestState",
        "states:RevealSecrets",
        "iam:PassRole"
      ],
      "Resource": "*"
    }
  ]
}
```

상태 테스트(콘솔)

콘솔에서 상태를 테스트하고 상태 출력 또는 입출력 데이터 처리 흐름을 확인할 수 있습니다. HTTP 태스크에서 원시 HTTP 요청 및 응답을 테스트할 수 있습니다.

상태를 테스트하려면

1. Step Functions 콘솔을 엽니다.
2. 상태 머신 생성을 선택하여 상태 머신 생성을 시작하거나 기존 상태 머신을 선택합니다.
3. Workflow Studio의 디자인 모드에서 테스트하려는 상태를 선택합니다.
4. Workflow Studio의 Inspector 패널에서 테스트 상태를 선택합니다.

5. 테스트 상태 대화 상자에서 다음을 수행합니다.

   a. 실행 역할에서 상태를 테스트할 실행 역할을 선택합니다. 테스트하려는 상태에 필요한 IAM 권한이 있는지 확인합니다.

   b. (선택 사항) 선택한 상태에서 테스트에 필요한 모든 JSON 입력을 제공합니다.

   c. 검사 레벨에서 보려는 값을 기준으로 다음 옵션 중 하나를 선택합니다.

      - 정보 - 테스트 성공 시 출력 탭에 상태 출력을 표시합니다. 테스트가 실패하면 정보 탭에는 오류 이름과 해당 오류의 원인에 대한 자세한 설명이 포함된 오류 출력이 표시됩니다. 레벨을 지정하지 않은 경우 Step Functions는 기본적으로 검사 레벨을 정보로 설정합니다.
      - 디버그 - 테스트가 성공하면 상태 출력과 입력 및 출력 데이터 처리 결과가 표시됩니다. 테스트가 실패하면 디버그에는 오류 이름과 해당 오류의 원인에 대한 자세한 설명이 포함된 오류 출력이 표시됩니다.
      - 추적 - 원시 HTTP 요청 및 응답을 보여 주며 헤더, 쿼리 파라미터 및 기타 API별 세부 정보를 확인하는 데 유용합니다. 이 옵션은 HTTP 태스크에만 사용할 수 있습니다.

        필요에 따라 비밀 공개를 선택할 수 있습니다. 이 설정을 추적과 함께 사용하면 API 키 등와 같이 EventBridge 연결에 삽입되는 민감한 데이터를 볼 수 있습니다. 콘솔에 액세스하는 데 사용하는 IAM 사용자 ID에는 `states:RevealSecrets` 작업을 수행할 권한이 있어야 합니다. 이 권한이 없으면 Step Functions에서 테스트가 시작될 때 액세스 거부 오류가 발생합니다. `states:RevealSecrets` 권한을 설정하는 IAM 정책에 대한 예제는 IAM TestState API 사용 권한 섹션을 참조하세요.

   d. 테스트 시작을 선택합니다.

AWS CLI를 사용하여 상태 테스트

에서 TestStateAPI를 사용하여 지원되는 상태를 테스트할 수 AWS CLI 있습니다. 이 API를 사용하여 상태에 대한 정의를 수락하고 이를 실행합니다.

각 상태에서는 테스트 결과에서 보려는 세부 정보의 양을 지정할 수 있습니다. 이러한 세부 정보는 입력 및 출력 데이터 처리 결과, HTTP 요청 및 응답 정보를 포함하여 상태 실행에 대한 추가 정보를 제공합니다. 다음 예는 TestState API에 지정할 수 있는 다양한 검사 수준을 보여줍니다. `####` 텍스트를 리소스별 정보로 바꿔야 합니다.

이 섹션에는 Step Functions가 AWS CLI에 제공하는 다양한 검사 레벨을 사용하는 방법을 설명하는 다음 예제가 포함되어 있습니다.

- INFO inspectionLevel 사용

- DEBUG inspectionLevel 사용
- TRACE inspectionLevel 사용
- jq 유틸리티를 사용하여 TestState API가 AWS CLI 반환하는 HTTP 응답을 필터링하고 인쇄합니다.

예제 1: INFO inspectionLevel을 사용하여 선택 상태 테스트

에서 `INFO` InspectionLevel을 사용하여 상태를 테스트하려면 `test-state` 다음 예제와 같이 명령을 실행합니다. AWS CLI

```
aws stepfunctions test-state \
    --definition '{"Type": "Choice", "Choices": [{"Variable": "$.number",
 "NumericEquals": 1, "Next": "Equals 1"}, {"Variable": "$.number", "NumericEquals": 2,
 "Next": "Equals 2"}], "Default": "No Match"}' \
    --role-arn arn:aws:iam::123456789012:role/myRole \
    --input '{"number": 2}'
```

이 예제에서는 선택 상태를 사용하여 사용자가 제공한 숫자 입력을 기반으로 상태의 실행 경로를 결정합니다. 레벨을 설정하지 않은 경우 기본적으로 Step Functions는 `inspectionLevel`을 `INFO`로 설정합니다.

Step Functions는 다음 출력을 반환합니다.

```
{
    "output": "{\"number\": 2}",
    "nextState": "Equals 2",
    "status": "SUCCEEDED"
}
```

예제 2: DEBUG inspectionLevel을 사용하여 패스 상태의 입력 및 출력 데이터 처리 디버깅

에서 `DEBUG` InspectionLevel을 사용하여 상태를 테스트하려면 다음 예제와 같이 `test-state` 명령을 실행합니다. AWS CLI

```
aws stepfunctions test-state \
    --definition '{"Type": "Pass", "InputPath": "$.payload", "Parameters": {"data": 1},
 "ResultPath": "$.result", "OutputPath": "$.result.data", "Next": "Another State"}' \
    --role-arn arn:aws:iam::123456789012:role/myRole \
    --input '{"payload": {"foo": "bar"}}' \
```

```
    --inspection-level DEBUG
```

이 예제에서는 Step Functions가 Pass 상태를 사용하여 입력 및 출력 데이터 처리 필터로 입력 JSON 데이터를 필터링하고 조작하는 방법을 보여 줍니다. 이 예제에서는 InputPath, ####, ResultPath, OutputPath 필터를 사용합니다.

Step Functions는 다음 출력을 반환합니다.

```
{
    "output": "1",
    "inspectionData": {
        "input": "{\"payload\": {\"foo\": \"bar\"}}",
        "afterInputPath": "{\"foo\":\"bar\"}",
        "afterParameters": "{\"data\":1}",
        "afterResultSelector": "{\"data\":1}",
        "afterResultPath": "{\"payload\":{\"foo\":\"bar\"},\"result\":{\"data\":1}}"
    },
    "nextState": "Another State",
    "status": "SUCCEEDED"
}
```

예제 3: TRACE inspectionLevel 및 revealSecrets를 사용하여 타사 API로 전송된 HTTP 요청 검사

에서 TRACE InspectionLevel을 RevealSecrets 매개 변수와 함께 사용하여 HTTP 작업을 테스트하려면 다음 AWS CLI 예제와 같이 `test-state` 명령을 실행합니다.

```
aws stepfunctions test-state \
    --definition '{"Type": "Task", "Resource": "arn:aws:states:::http:invoke",
 "Parameters": {"Method": "GET", "Authentication": {"ConnectionArn":
 "arn:aws:events:us-
east-1:123456789012:connection/MyConnection/0000000-0000-0000-0000-000000000000"},
 "ApiEndpoint": "https://httpbin.org/get", "Headers": {"definitionHeader": "h1"},
 "RequestBody": {"message": "Hello from Step Functions!"}, "QueryParameters":
 {"queryParam": "q1"}}, "End": true}' \
    --role-arn arn:aws:iam::123456789012:role/myRole \
    --inspection-level TRACE \
    --reveal-secrets
```

이 예제에서는 HTTP 태스크가 지정된 타사 API를 호출하는지 테스트합니다(`https://httpbin.org/`). 또한 API 직접 호출에 대한 HTTP 요청 및 응답 데이터도 보여 줍니다.

```
{
    "output": "{\"Headers\":{\"date\":[\"Tue, 21 Nov 2023 00:06:17 GMT\"],
\"access-control-allow-origin\":[\"*\"],\"content-length\":[\"620\"],\"server\":
[\"gunicorn/19.9.0\"],\"access-control-allow-credentials\":[\"true\"],\"content-
type\":[\"application/json\"]},\"ResponseBody\":{\"args\":{\"QueryParam1\":
\"QueryParamValue1\",\"queryParam\":\"q1\"},\"headers\":{\"Authorization
\":\"Basic XXXXXXXX\",\"Content-Type\":\"application/json; charset=UTF-8\",
\"Customheader1\":\"CustomHeaderValue1\",\"Definitionheader\":\"h1\",\"Host\":
\"httpbin.org\",\"Range\":\"bytes=0-262144\",\"Transfer-Encoding\":\"chunked\",
\"User-Agent\":\"Amazon|StepFunctions|HttpInvoke|us-east-1\",\"X-Amzn-Trace-Id\":
\"Root=1-0000000-0000-0000-0000-000000000000\"},\"origin\":\"12.34.567.891\",\"url\":
\"https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1\"},\"StatusCode
\":200,\"StatusText\":\"OK\"}",
    "inspectionData": {
        "input": "{}",
        "afterInputPath": "{}",
        "afterParameters": "{\"Method\":\"GET\",\"Authentication\":{\"ConnectionArn
\":\"arn:aws:events:us-east-1:123456789012:connection/foo/a59c10f0-a315-4c1f-
be6a-559b9a0c6250\"},\"ApiEndpoint\":\"https://httpbin.org/get\",\"Headers\":
{\"definitionHeader\":\"h1\"},\"RequestBody\":{\"message\":\"Hello from Step Functions!
\"},\"QueryParameters\":{\"queryParam\":\"q1\"}}",
        "result": "{\"Headers\":{\"date\":[\"Tue, 21 Nov 2023 00:06:17 GMT\"],
\"access-control-allow-origin\":[\"*\"],\"content-length\":[\"620\"],\"server\":
[\"gunicorn/19.9.0\"],\"access-control-allow-credentials\":[\"true\"],\"content-
type\":[\"application/json\"]},\"ResponseBody\":{\"args\":{\"QueryParam1\":
\"QueryParamValue1\",\"queryParam\":\"q1\"},\"headers\":{\"Authorization
\":\"Basic XXXXXXXX\",\"Content-Type\":\"application/json; charset=UTF-8\",
\"Customheader1\":\"CustomHeaderValue1\",\"Definitionheader\":\"h1\",\"Host\":
\"httpbin.org\",\"Range\":\"bytes=0-262144\",\"Transfer-Encoding\":\"chunked\",
\"User-Agent\":\"Amazon|StepFunctions|HttpInvoke|us-east-1\",\"X-Amzn-Trace-Id\":
\"Root=1-0000000-0000-0000-0000-000000000000\"},\"origin\":\"12.34.567.891\",\"url\":
\"https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1\"},\"StatusCode
\":200,\"StatusText\":\"OK\"}",
        "afterResultSelector": "{\"Headers\":{\"date\":[\"Tue, 21 Nov 2023
 00:06:17 GMT\"],\"access-control-allow-origin\":[\"*\"],\"content-length\":
[\"620\"],\"server\":[\"gunicorn/19.9.0\"],\"access-control-allow-credentials
\":[\"true\"],\"content-type\":[\"application/json\"]},\"ResponseBody\":{\"args
\":{\"QueryParam1\":\"QueryParamValue1\",\"queryParam\":\"q1\"},\"headers\":
{\"Authorization\":\"Basic XXXXXXXX\",\"Content-Type\":\"application/json;
 charset=UTF-8\",\"Customheader1\":\"CustomHeaderValue1\",\"Definitionheader\":\"h1\",
\"Host\":\"httpbin.org\",\"Range\":\"bytes=0-262144\",\"Transfer-Encoding\":\"chunked
\",\"User-Agent\":\"Amazon|StepFunctions|HttpInvoke|us-east-1\",\"X-Amzn-Trace-Id\":
\"Root=1-0000000-0000-0000-0000-000000000000\"},\"origin\":\"12.34.567.891\",\"url\":
```

```
\"https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1\"},\"StatusCode
\":200,\"StatusText\":\"OK\"}",
        "afterResultPath": "{\"Headers\":{\"date\":[\"Tue, 21 Nov 2023 00:06:17
 GMT\"],\"access-control-allow-origin\":[\"*\"],\"content-length\":[\"620\"],
\"server\":[\"gunicorn/19.9.0\"],\"access-control-allow-credentials\":[\"true\"],
\"content-type\":[\"application/json\"]},\"ResponseBody\":{\"args\":{\"QueryParam1\":
\"QueryParamValue1\",\"queryParam\":\"q1\"},\"headers\":{\"Authorization\":
\"Basic XXXXXXXX\",\"Content-Type\":\"application/json; charset=UTF-8\",
\"Customheader1\":\"CustomHeaderValue1\",\"Definitionheader\":\"h1\",\"Host\":
\"httpbin.org\",\"Range\":\"bytes=0-262144\",\"Transfer-Encoding\":\"chunked\",
\"User-Agent\":\"Amazon|StepFunctions|HttpInvoke|us-east-1\",\"X-Amzn-Trace-Id\":
\"Root=1-0000000-0000-0000-0000-000000000000\"},\"origin\":\"12.34.567.891\",\"url\":
\"https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1\"},\"StatusCode
\":200,\"StatusText\":\"OK\"}",
        "request": {
            "protocol": "https",
            "method": "GET",
            "url": "https://httpbin.org/get?
queryParam=q1&QueryParam1=QueryParamValue1",
            "headers": "[definitionHeader: h1, Authorization: Basic XXXXXXXX,
 CustomHeader1: CustomHeaderValue1, User-Agent: Amazon|StepFunctions|HttpInvoke|us-
east-1, Range: bytes=0-262144]",
            "body": "{\"message\":\"Hello from Step Functions!\",\"BodyKey1\":
\"BodyValue1\"}"
        },
        "response": {
            "protocol": "https",
            "statusCode": "200",
            "statusMessage": "OK",
            "headers": "[date: Tue, 21 Nov 2023 00:06:17 GMT, content-type:
 application/json, content-length: 620, server: gunicorn/19.9.0, access-control-allow-
origin: *, access-control-allow-credentials: true]",
            "body": "{\n  \"args\": {\n    \"QueryParam1\": \"QueryParamValue1\", \n
    \"queryParam\": \"q1\"\n  }, \n  \"headers\": {\n    \"Authorization\": \"Basic
 XXXXXXXX\", \n    \"Content-Type\": \"application/json; charset=UTF-8\", \n
 \"Customheader1\": \"CustomHeaderValue1\", \n    \"Definitionheader\": \"h1\", \n
   \"Host\": \"httpbin.org\", \n    \"Range\": \"bytes=0-262144\", \n    \"Transfer-
Encoding\": \"chunked\", \n    \"User-Agent\": \"Amazon|StepFunctions|HttpInvoke|us-
east-1\", \n    \"X-Amzn-Trace-Id\": \"Root=1-0000000-0000-0000-0000-000000000000\"\n
  }, \n  \"origin\": \"12.34.567.891\", \n  \"url\": \"https://httpbin.org/get?
queryParam=q1&QueryParam1=QueryParamValue1\"\n}\n"
        }
    },
    "status": "SUCCEEDED"
```

```
}
```

예제 4: jq 유틸리티를 사용하여 API가 반환하는 응답을 필터링하고 인쇄하기 TestState

TestState API는 응답에서 JSON 데이터를 이스케이프된 문자열로 반환합니다. 다음 AWS CLI 예제는 예제 3을 확장하고 이 `jq` 유틸리티를 사용하여 TestState API가 반환하는 HTTP 응답을 사람이 읽을 수 있는 형식으로 필터링하고 인쇄합니다. 에 대한 자세한 내용 `jq` 및 설치 지침은 jq on을 참조하십시오. GitHub

```
aws stepfunctions test-state \
    --definition '{"Type": "Task", "Resource": "arn:aws:states:::http:invoke",
 "Parameters": {"Method": "GET", "Authentication": {"ConnectionArn":
 "arn:aws:events:us-
east-1:123456789012:connection/MyConnection/0000000-0000-0000-0000-000000000000"},
 "ApiEndpoint": "https://httpbin.org/get", "Headers": {"definitionHeader": "h1"},
 "RequestBody": {"message": "Hello from Step Functions!"}, "QueryParameters":
 {"queryParam": "q1"}}, "End": true}' \
    --role-arn arn:aws:iam::123456789012:role/myRole \
    --inspection-level TRACE \
    --reveal-secrets \
    | jq '.inspectionData.response.body | fromjson'
```

다음 예제에서는 사람이 읽을 수 있는 형식으로 반환된 출력을 보여 줍니다.

```
{
  "args": {
    "QueryParam1": "QueryParamValue1",
    "queryParam": "q1"
  },
  "headers": {
    "Authorization": "Basic XXXXXXXX",
    "Content-Type": "application/json; charset=UTF-8",
    "Customheader1": "CustomHeaderValue1",
    "Definitionheader": "h1",
    "Host": "httpbin.org",
    "Range": "bytes=0-262144",
    "Transfer-Encoding": "chunked",
    "User-Agent": "Amazon|StepFunctions|HttpInvoke|us-east-1",
    "X-Amzn-Trace-Id": "Root=1-0000000-0000-0000-0000-000000000000"
  },
  "origin": "12.34.567.891",
  "url": "https://httpbin.org/get?queryParam=q1&QueryParam1=QueryParamValue1"
```

```
}
```

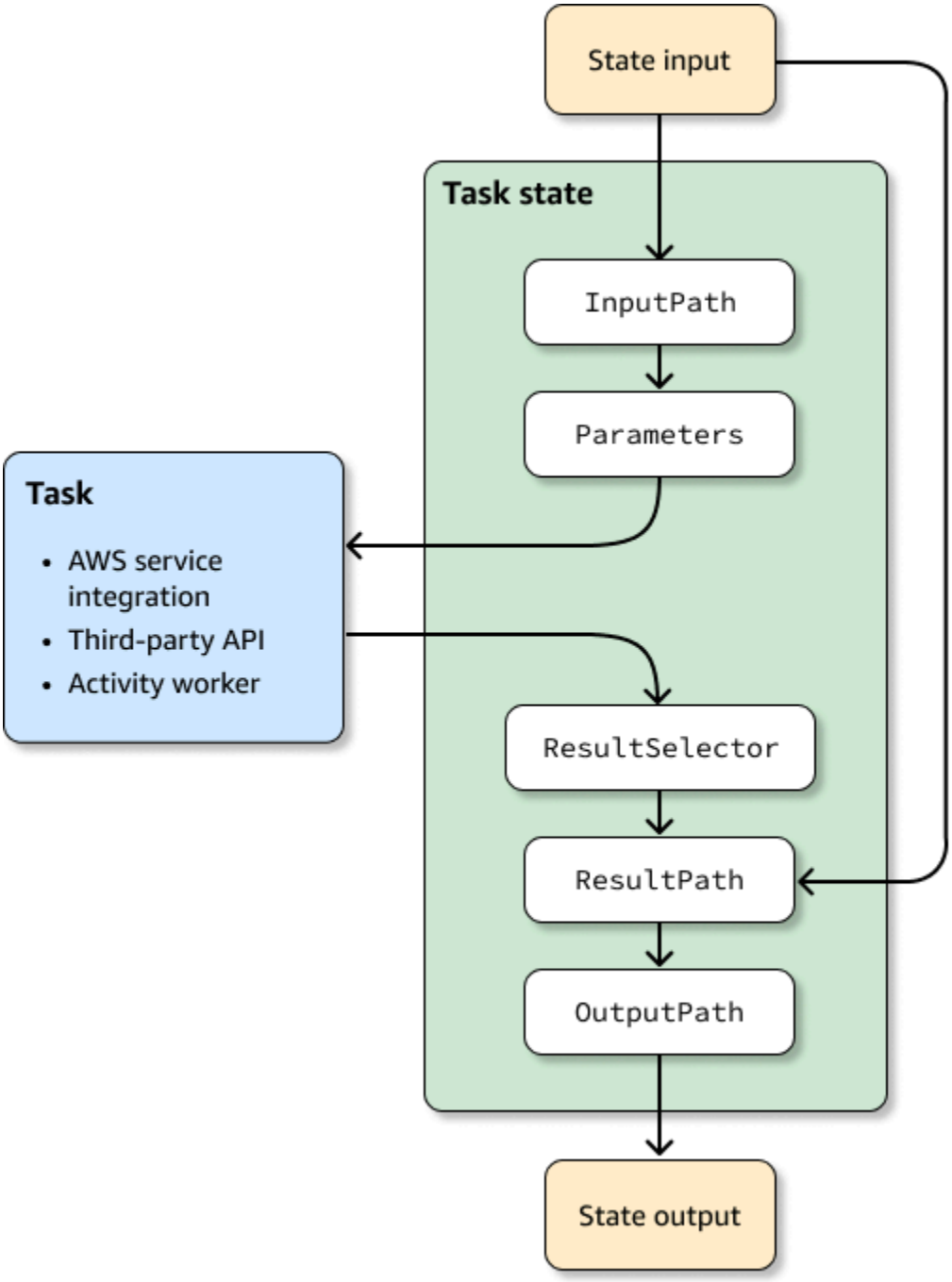
입력 및 출력 데이터 흐름 테스트 및 디버깅

`TestState` API는 워크플로를 통해 전달되는 데이터를 테스트하고 디버깅하는 데 유용합니다. 이 섹션에서는 몇 가지 주요 개념을 제공하고 이를 TestState 위해 를 사용하는 방법을 설명합니다.

주요 개념

Step Functions에서는 상태 머신의 상태를 통과하는 JSON 데이터를 필터링하고 조작하는 프로세스를 입력 및 출력 처리라고 합니다. 이 기능의 작동 방식에 대한 자세한 내용은 Step Functions에서 입력 및 출력 처리 단원을 참조하십시오.

Amazon States Language(ASL)의 모든 상태 유형(태스크, 병렬, 맵, 통과, 대기, 선택, 성공, 실패)은 통과하는 JSON 데이터를 필터링하고 조작하기 위한 일련의 공통 필드를 공유합니다. 이러한 필드로는 InputPath, 파라미터, ResultSelector, ResultPath, OutputPath 등이 있습니다. 각 필드에 대한 지원은 상태마다 다릅니다. 런타임 시 Step Functions는 각 필드를 특정 순서로 적용합니다. 다음 다이어그램은 이러한 필드가 태스크 상태 내의 데이터에 적용되는 순서를 보여 줍니다.



다음 목록은 다이어그램에 표시된 입력 및 출력 처리 필드의 적용 순서를 설명합니다.

1. 상태 입력은 이전 상태에서 현재 상태로 전달된 JSON 데이터입니다.
2. InputPath에서는 원시 상태 입력의 일부를 필터링합니다.
3. 파라미터에서는 태스크에 전달할 값 집합을 구성합니다.
4. 태스크는 작업을 수행하고 결과를 반환합니다.
5. ResultSelector에서는 태스크 결과에서 유지할 값 집합을 선택합니다.

6. ResultPath에서는 결과를 원시 상태 입력과 결합하거나 결과를 원시 상태 입력으로 대체합니다.
7. OutputPath에서는 출력의 일부를 필터링하여 다음 상태로 전달합니다.
8. 상태 출력은 현재 상태에서 다음 상태로 전달되는 JSON 데이터입니다.

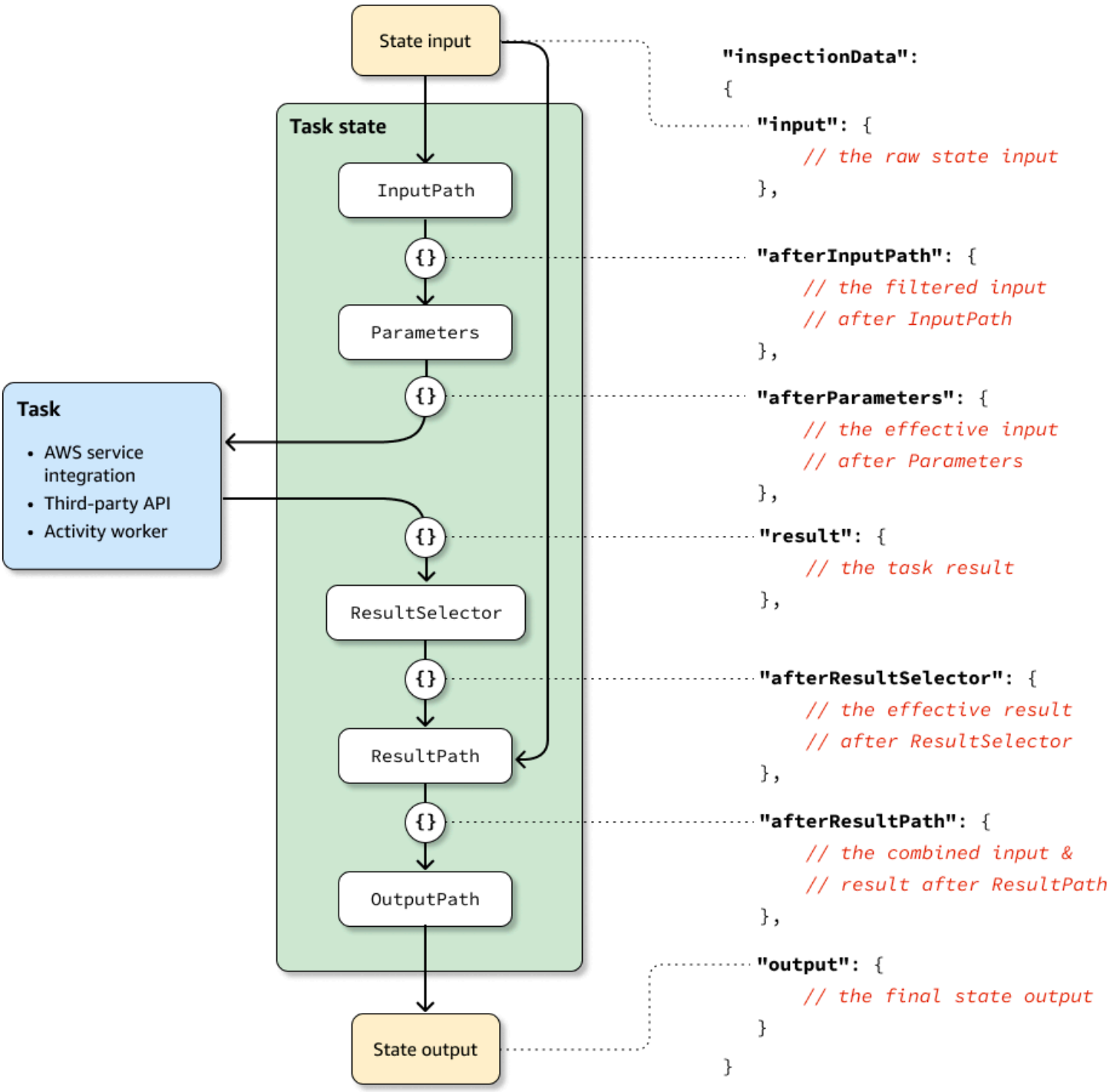
이러한 입력 및 출력 처리 필드는 선택 사항입니다. 상태 정의에서 이러한 필드를 사용하지 않는 경우 태스크는 원시 상태 입력을 사용하고 태스크 결과를 상태 출력으로 반환합니다.

입력 및 출력 처리를 TestState 검사하는 데 사용

`TestState` API를 호출하고 `inspectionLevel` 파라미터를 `DEBUG`로 설정하면 API 응답에 `inspectionData`라는 객체가 포함됩니다. 이 객체에는 데이터가 실행되었을 때 상태 내에서 데이터가 필터링 또는 조작된 방식을 검사하는 데 도움이 되는 필드가 포함되어 있습니다. 다음 예제는 태스크 상태용 `inspectionData` 객체를 보여 줍니다.

```
"inspectionData":   {
  "input": string,
  "afterInputPath": string,
  "afterParameters": string,
  "result": string,
  "afterResultSelector": string,
  "afterResultPath": string,
  "output": string
}
```

이 예제에서 `after` 접두사가 포함된 각 필드는 특정 필드가 적용된 이후의 데이터를 보여 줍니다. 예를 들어 `afterInputPath`는 `InputPath` 필드를 적용하여 원시 상태 입력을 필터링했을 때의 효과를 보여 줍니다. 다음 다이어그램은 각 ASL 정의 필드를 `inspectionData` 객체의 해당 필드에 매핑합니다.



TestState API를 사용하여 입력 및 출력 처리를 디버깅하는 예는 다음을 참조하십시오.

- Step Functions 콘솔에서 DEBUG 검사 레벨을 사용하여 상태 테스트
- 의 디버그 검사 수준을 사용하여 상태를 테스트합니다. AWS CLI

상태 시스템을 로컬로 테스트

AWS Step Functions Local은 다운로드 가능한 Step Functions 버전으로, 이를 사용하면 자체 개발 환경에서 실행 중인 Step Functions 버전을 사용하여 애플리케이션을 개발하고 테스트할 수 있습니다. Step Functions의 로컬 버전은 AWS 모두에서 그리고 로컬로 실행될 때 AWS Lambda 함수를 간접적으로 호출할 수 있습니다. 또한 기타 지원되는 AWS 서비스를 조정할 수도 있습니다.

> **Note**
>
> Step Functions Local은 더미 계정을 사용하여 작업합니다.

Step Functions Local을 실행하는 동안 다음 방법 중 하나를 사용하여 서비스 통합을 간접적으로 호출할 수 있습니다.

- AWS Lambda 및 기타 서비스에 대한 로컬 엔드포인트 구성. 지원되는 엔드포인트는 Step Functions Local의 구성 옵션 설정을 참조하세요.
- Step Functions Local에서 AWS 서비스 직접 호출
- 서비스 통합의 응답 모의. 모의 서비스 통합 사용 방법은 모의 서비스 통합 사용을 참조하세요.

AWS Step Functions Local은 JAR 패키지 또는 Microsoft Windows, Linux, macOS 및 Java나 Docker를 지원하는 기타 플랫폼에서 실행되는 독립형 도커 이미지로 제공됩니다.

> **Warning**
>
> AWS Step Functions의 다운로드 가능 버전은 테스트용으로만 사용될 수 있으며 민감한 정보를 처리하는 데 사용해서는 안 됩니다.

> **Tip**
>
> 워크플로에 모든 내장 함수를 포함할 수 있으려면 Step Functions Local 버전 1.12.0 이상을 사용해야 합니다.

다음 주제에서는 Docker 및 JAR 파일을 사용하여 Step Functions Local을 설정하고 AWS Lambda, AWS Serverless Application Model(AWS SAM) CLI Local 또는 기타 지원되는 서비스와 함께 작동하도록 Step Functions Local을 실행하는 방법을 설명합니다.

주제

Step Functions Local(다운로드 가능 버전) 및 Docker 설정

Step Functions Local 도커 이미지를 사용하면 필요한 모든 종속성과 함께 도커 이미지를 사용하여 Step Functions Local을 빠르게 시작할 수 있습니다. 도커 이미지를 사용하면 Step Functions Local을 컨테이너화된 빌드에 지속적 통합 테스트의 일부로 포함할 수 있습니다.

Step Functions Local에 대한 도커 이미지를 가져오려면 https://hub.docker.com/r/amazon/aws-stepfunctions-local을 참조하거나 다음 Docker `pull` 명령을 입력합니다.

```
docker pull amazon/aws-stepfunctions-local
```

Docker에서 Step Functions의 다운로드 가능 버전을 시작하려면 다음 Docker `run` 명령을 실행합니다.

```
docker run -p 8083:8083 amazon/aws-stepfunctions-local
```

AWS Lambda 또는 기타 지원되는 서비스와 상호 작용하려면 먼저 보안 인증 정보와 기타 구성 옵션을 구성해야 합니다. 자세한 정보는 다음 주제를 참조하십시오.

- Step Functions Local의 구성 옵션 설정
- Docker에 대한 보안 인증 정보 및 구성

Step Functions Local(다운로드 가능 버전) 설정 - Java 버전

AWS Step Functions의 다운로드 가능 버전은 JAR 실행 파일이나 도커 이미지로 제공됩니다. Java 애플리케이션은 Windows, Linux, macOS 및 Java를 지원하는 기타 플랫폼에서 실행됩니다. Java 외에도 AWS Command Line Interface(AWS CLI)를 설치해야 합니다. AWS CLI 설치 및 구성 방법은 AWS Command Line Interface 사용 설명서를 참조하세요.

컴퓨터에 Step Functions를 설정하고 실행하기

1. 다음 링크를 사용하여 Step Functions를 다운로드합니다.

다운로드 링크	체크섬
.tar.gz	.tar.gz.md5
.zip	.zip.md5

2. `.zip` 파일의 압축을 풉니다.
3. 다운로드를 테스트하고 버전 정보를 봅니다.

```
$ java -jar StepFunctionsLocal.jar -v
Step Function Local
Version: 1.0.0
Build: 2019-01-21
```

4. (선택 사항) 사용 가능한 명령 목록을 봅니다.

```
$ java -jar StepFunctionsLocal.jar -h
```

5. 컴퓨터에서 Step Functions를 시작하려면 명령 프롬프트를 열고 `StepFunctionsLocal.jar` 압축을 해제한 디렉터리로 이동한 후 다음 명령을 입력합니다.

```
java -jar StepFunctionsLocal.jar
```

6. 로컬에서 실행되는 Step Functions에 액세스하려면 `--endpoint-url` 파라미터를 사용합니다. 예를 들어 AWS CLI를 사용하여 Step Functions 명령을 다음과 같이 지정합니다.

```
aws stepfunctions --endpoint-url http://localhost:8083 command
```

> **Note**
>
> 기본적으로 Step Functions Local은 로컬 테스트 계정과 보안 인증 정보를 사용하며 AWS 리전은 미국 동부(버지니아 북부)로 설정됩니다. Step Functions Local을 AWS Lambda 또는 기타 지원되는 서비스와 함께 사용하려면 보안 인증 정보와 리전을 구성해야 합니다.
> Step Functions Local에서 Express 워크플로를 사용하면 실행 내역이 로그 파일에 저장됩니다. CloudWatch Logs에는 로깅되지 않습니다. 로그 파일 경로는 로컬 상태 시스템을 만들 때 제공된 CloudWatch Logs 로그 그룹 ARN을 기반으로 합니다. 로그 파일은 Step Functions Local을 실행하는 위치를 기준으로 `/aws/states/log-group-name/${execution_arn}.log`에 저장됩니다. 예를 들어 실행 ARN이 다음과 같은 경우
>
> ```
> arn:aws:states:us-east-1:123456789012:express:test:example-ExpressLogGroup-wJalrXUtnFEMI
> ```
>
> 로그 파일은 다음과 같습니다.
>
> ```
> aws/states/log-group-name/arn:aws:states:us-east-1:123456789012:express:test:example-ExpressLogGroup-wJalrXUtnFEMI.log
> ```

Step Functions Local의 구성 옵션 설정

JAR 파일을 사용하여 AWS Step Functions Local을 시작할 때 AWS Command Line Interface(AWS CLI)를 사용하거나 시스템 환경에 포함시켜서 구성 옵션을 설정할 수 있습니다. Docker의 경우 Step Functions Local을 시작할 때 참조하는 파일에 이러한 옵션을 지정해야 합니다.

구성 옵션

Lambda 엔드포인트 및 배치 엔드포인트와 같은 재정의 엔드포인트를 사용하도록 Step Functions Local 컨테이너를 구성하고 해당 엔드포인트를 직접적으로 호출하면 Step Functions Local에서 지정된 보안 인증 정보를 사용하지 않습니다. 이러한 엔드포인트 재정의 설정은 선택 사항입니다.

옵션	명령줄	환경
계정	-account, --aws-account	AWS_ACCOUNT_ID
리전	-region, --aws-region	AWS_DEFAULT_REGION

옵션	명령줄	환경
대기 시간 비율	-waitTimeScale, --wait-time-scale	WAIT_TIME_SCALE
Lambda 엔드포인트	-lambdaEndpoint, --lambda-endpoint	LAMBDA_ENDPOINT
배치 엔드포인트	-batchEndpoint, --batch-endpoint	BATCH_ENDPOINT
DynamoDB 엔드포인트	-dynamoDBEndpoint, --dynamodb-endpoint	DYNAMODB_ENDPOINT
ECS 엔드포인트	-ecsEndpoint, --ecs-endpoint	ECS_ENDPOINT
Glue 엔드포인트	-glueEndpoint, --glue-endpoint	GLUE_ENDPOINT
SageMaker 엔드포인트	-sageMakerEndpoint, --sagemaker-endpoint	SAGE_MAKER_ENDPOINT
SQS 엔드포인트	-sqsEndpoint, --sqs-endpoint	SQS_ENDPOINT
SNS 엔드포인트	-snsEndpoint, --sns-endpoint	SNS_ENDPOINT
Step Functions 엔드포인트	-stepFunctionsEndpoint, --step-functions-endpoint	STEP_FUNCTIONS_ENDPOINT

Docker에 대한 보안 인증 정보 및 구성

Docker용 Step Functions Local을 구성하려면 `aws-stepfunctions-local-credentials.txt` 파일을 만듭니다.

이 파일에는 보안 인증 정보와 기타 구성 옵션이 포함됩니다. 다음은 `aws-stepfunctions-local-credentials.txt` 파일을 만들 때 템플릿으로 사용될 수 있습니다.

```
AWS_DEFAULT_REGION=AWS_REGION_OF_YOUR_AWS_RESOURCES
AWS_ACCESS_KEY_ID=YOUR_AWS_ACCESS_KEY
AWS_SECRET_ACCESS_KEY=YOUR_AWS_SECRET_KEY
WAIT_TIME_SCALE=VALUE
```

```
LAMBDA_ENDPOINT=VALUE
BATCH_ENDPOINT=VALUE
DYNAMODB_ENDPOINT=VALUE
ECS_ENDPOINT=VALUE
GLUE_ENDPOINT=VALUE
SAGE_MAKER_ENDPOINT=VALUE
SQS_ENDPOINT=VALUE
SNS_ENDPOINT=VALUE
STEP_FUNCTIONS_ENDPOINT=VALUE
```

`aws-stepfunctions-local-credentials.txt`에서 보안 인증 정보와 구성 옵션을 구성하면 다음 명령을 사용하여 Step Functions를 시작합니다.

```
docker run -p 8083:8083 --env-file aws-stepfunctions-local-credentials.txt amazon/aws-stepfunctions-local
```

> **Note**
>
> 호스트에서 사용하는 내부 IP 주소(예: `http://host.docker.internal:8000`)로 확인되는 특수 DNS 이름 `host.docker.internal`을 사용하는 것이 좋습니다. 자세한 내용은 Mac용 Docker Desktop의 네트워킹 기능 및 Windows용 Docker Desktop의 네트워킹 기능에서 Mac 및 Windows용 Docker 설명서를 참조하세요.

컴퓨터에서 Step Functions Local 실행

로컬 버전의 Step Functions를 사용하여 컴퓨터에 상태 시스템을 구성, 개발 및 테스트합니다.

HelloWorld 상태 시스템을 로컬로 실행

AWS Command Line Interface(AWS CLI)를 사용하여 Step Functions Local을 로컬로 실행한 후에 상태 시스템 실행을 시작할 수 있습니다.

1. 상태 시스템 정의를 이스케이프하여 AWS CLI에서 상태 시스템을 만듭니다.

```
aws stepfunctions --endpoint-url http://localhost:8083 create-state-machine --
definition "{\
  \"Comment\": \"A Hello World example of the Amazon States Language using a Pass
 state\",\
  \"StartAt\": \"HelloWorld\",\
```

```
  \"States\": {\
    \"HelloWorld\": {\
      \"Type\": \"Pass\",\
      \"End\": true\
    }\
  }}" --name "HelloWorld" --role-arn "arn:aws:iam::012345678901:role/DummyRole"
```

> **Note**
>
> `role-arn`은 Step Functions Local에 사용되지 않지만 적절한 구문을 통해 포함되도록 해야 합니다. 이전 예제의 Amazon 리소스 이름(ARN)을 사용할 수 있습니다.

상태 시스템을 성공적으로 만들면 Step Functions는 생성 날짜와 상태 시스템 ARN으로 응답합니다.

```
{
    "creationDate": 1548454198.202,
    "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld"
}
```

2. 생성한 상태 머신의 ARN을 사용하여 실행을 시작합니다.

```
aws stepfunctions --endpoint-url http://localhost:8083 start-execution --state-
machine-arn arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld
```

AWS SAM CLI Local을 사용하는 Step Functions Local

로컬 버전의 AWS Lambda과 함께 로컬 버전의 Step Functions를 사용할 수 있습니다. 이를 구성하려면 AWS SAM을 설치하고 구성해야 합니다.

AWS SAM 구성 및 실행에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SAM 설정
- AWS SAM CLI Local을 시작합니다.

로컬 시스템에서 Lambda가 실행되면 Step Functions Local을 시작할 수 있습니다. Step Functions 로컬 JAR 파일을 추출한 디렉터리에서 Step Functions Local을 시작하고 `--lambda-endpoint` 파라미터를 사용하여 로컬 Lambda 엔드포인트를 구성합니다.

```
java -jar StepFunctionsLocal.jar --lambda-endpoint http://127.0.0.1:3001 command
```

AWS Lambda을 사용하여 Step Functions Local을 실행하는 방법에 대한 자세한 내용은 Step Function 및 AWS SAM CLI Local 테스트를 참조하세요.

Step Function 및 AWS SAM CLI Local 테스트

로컬 시스템에서 AWS Step Functions 및 AWS Lambda 모두 실행 중이면 코드를 AWS에 배포하지 않고 상태 시스템과 Lambda 함수를 테스트할 수 있습니다.

자세한 정보는 다음 주제를 참조하세요.

- 상태 시스템을 로컬로 테스트
- AWS SAM 설정

주제

- 1단계: AWS SAM 설정
- 2단계: AWS SAM CLI Local 테스트
- 3단계: AWS SAM CLI Local 시작
- 4단계: Step Functions Local 시작
- 5단계: AWS SAM CLI Local 함수를 참조하는 상태 머신 생성
- 6단계: 로컬 상태 머신의 실행 시작

1단계: AWS SAM 설정

AWS Serverless Application Model(AWS SAM) CLI Local을 사용하려면 AWS Command Line Interface, AWS SAM 및 Docker를 설치해야 합니다.

1. AWS SAM CLI를 설치합니다.

> **Note**
>
> AWS SAM CLI를 설치하기 전에 AWS CLI와 Docker를 설치해야 합니다. CLI AWS SAM 설치를 위한 사전 조건을 참조하세요.

2. AWS SAM 빠른 시작 설명서를 살펴봅니다. 다음 단계에 따라 다음을 수행하십시오.

   1. 애플리케이션 초기화
   2. 로컬로 애플리케이션 테스트

   이렇게 하면 `sam-app` 디렉터리가 생성되고 Python 기반 Hello World Lambda 함수가 포함된 환경이 빌드됩니다.

2단계: AWS SAM CLI Local 테스트

이제 AWS SAM을 설치하고 Hello World Lambda 함수를 만들었으므로 테스트할 수 있습니다. `sam-app` 디렉터리에서 다음 명령을 입력합니다.

```
sam local start-api
```

그러면 Lambda 함수의 로컬 인스턴스가 시작됩니다. 다음과 유사한 출력이 표시되어야 합니다.

```
2019-01-31 16:40:27 Found credentials in shared credentials file: ~/.aws/credentials
2019-01-31 16:40:27 Mounting HelloWorldFunction at http://127.0.0.1:3000/hello [GET]
2019-01-31 16:40:27 You can now browse to the above endpoints to invoke your functions.
 You do not need to restart/reload SAM CLI while working on your functions changes will
 be reflected instantly/automatically. You only need to restart SAM CLI if you update
 your AWS SAM template
2019-01-31 16:40:27  * Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)
```

브라우저를 열고 다음을 입력합니다.

```
http://127.0.0.1:3000/hello
```

다음과 유사한 응답이 출력됩니다.

```
{"message": "hello world", "location": "72.21.198.66"}
```

CTRL+C를 입력하여 Lambda API를 종료합니다.

3단계: AWS SAM CLI Local 시작

이제 함수가 작동하는지 테스트했으므로 AWS SAM CLI Local을 시작합니다. `sam-app` 디렉터리에서 다음 명령을 입력합니다.

```
sam local start-lambda
```

그러면 AWS SAM CLI Local이 시작되고 다음 출력과 유사한 사용할 엔드포인트가 제공됩니다.

```
2019-01-29 15:33:32 Found credentials in shared credentials file: ~/.aws/credentials
2019-01-29 15:33:32 Starting the Local Lambda Service. You can now invoke your Lambda
 Functions defined in your template through the endpoint.
2019-01-29 15:33:32  * Running on http://127.0.0.1:3001/ (Press CTRL+C to quit)
```

4단계: Step Functions Local 시작

JAR 파일

Step Funtions Local의 `.jar` 파일 버전을 사용하는 경우 Step Functions를 시작하고 Lambda 엔드포인트를 지정합니다. `.jar` 파일 압축을 해제한 디렉터리에 다음 명령을 입력합니다.

```
java -jar StepFunctionsLocal.jar --lambda-endpoint http://localhost:3001
```

Step Functions Local이 시작되면 환경을 확인한 다음 `~/.aws/credentials` 파일에 구성된 보안 인증 정보를 확인합니다. 기본적으로 가상의 사용자 ID를 사용하기 시작하며 `region us-east-1`로 나열됩니다.

```
2019-01-29 15:38:06.324: Failed to load credentials from environment because Unable to
 load AWS credentials from environment variables (AWS_ACCESS_KEY_ID (or AWS_ACCESS_KEY)
 and AWS_SECRET_KEY (or AWS_SECRET_ACCESS_KEY))
2019-01-29 15:38:06.326: Loaded credentials from profile: default
2019-01-29 15:38:06.326: Starting server on port 8083 with account 123456789012, region
 us-east-1
```

도커

Step Functions Local의 Docker 버전을 사용하는 경우 다음 명령으로 Step Functions를 시작합니다.

```
docker run -p 8083:8083 amazon/aws-stepfunctions-local
```

Step Functions의 Docker 버전 설치 방법은 Step Functions Local(다운로드 가능 버전) 및 Docker 설정을 참조하세요.

> **Note**
>
> `.jar` 파일에서 Step Functions를 시작하면 명령줄을 통해 또는 환경 변수를 설정하여 엔드포인트를 지정할 수 있습니다. Docker 버전의 경우 텍스트 파일에서 엔드포인트와 자격 증명을 지정해야 합니다. Step Functions Local의 구성 옵션 설정 섹션을 참조하세요.

5단계: AWS SAM CLI Local 함수를 참조하는 상태 머신 생성

Step Functions Local이 실행 중이면 1단계: AWS SAM 설정에서 초기화한 `HelloWorldFunction`을 참조하는 상태 시스템을 만듭니다.

```
aws stepfunctions --endpoint http://localhost:8083 create-state-machine --definition
 "{\
  \"Comment\": \"A Hello World example of the Amazon States Language using an AWS
 Lambda Local function\",\
  \"StartAt\": \"HelloWorld\",\
  \"States\": {\
    \"HelloWorld\": {\
      \"Type\": \"Task\",\
      \"Resource\": \"arn:aws:lambda:us-east-1:123456789012:function:HelloWorldFunction
\",\
      \"End\": true\
    }\
  }\
}\
}}" --name "HelloWorld" --role-arn "arn:aws:iam::012345678901:role/DummyRole"
```

그러면 상태 시스템이 생성되고 실행을 시작하는 데 사용할 수 있는 Amazon 리소스 이름(ARN)이 제공됩니다.

```
{
    "creationDate": 1548805711.403,
    "stateMachineArn": "arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld"
}
```

6단계: 로컬 상태 머신의 실행 시작

상태 시스템을 만들었으면 실행을 시작합니다. 다음 **aws stepfunctions** 명령어를 사용할 때는 엔드포인트와 상태 시스템 ARN을 참조해야 합니다.

```
aws stepfunctions --endpoint http://localhost:8083 start-execution --state-machine
 arn:aws:states:us-east-1:123456789012:stateMachine:HelloWorld --name test
```

이렇게 하면 `HelloWorld` 상태 시스템의 `test` 실행이 시작됩니다.

```
{
    "startDate": 1548810641.52,
    "executionArn": "arn:aws:states:us-east-1:123456789012:execution:HelloWorld:test"
}
```

이제 Step Functions가 로컬로 실행 중이므로 AWS CLI를 사용하여 Step Functions와 상호 작용할 수 있습니다. 예를 들어 이 실행에 대한 정보를 얻으려면 다음 명령을 사용합니다.

```
aws stepfunctions --endpoint http://localhost:8083 describe-execution --execution-arn
 arn:aws:states:us-east-1:123456789012:execution:HelloWorld:test
```

실행에 대해 `describe-execution`을 직접적으로 호출하면 다음 출력과 같이 더욱 완전한 세부 정보가 제공됩니다.

```
{
    "status": "SUCCEEDED",
    "startDate": 1549056334.073,
    "name": "test",
    "executionArn": "arn:aws:states:us-east-1:123456789012:execution:HelloWorld:test",
    "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
    "stopDate": 1549056351.276,
    "output": "{\"statusCode\": 200, \"body\": \"{\\\"message\\\": \\\"hello world\\\",
 \\\"location\\\": \\\"72.21.198.64\\\"}\"}",
    "input": "{}"
}
```

모의 서비스 통합 사용

Step Functions Local에서 모의 서비스 통합을 사용하여 통합 서비스를 실제로 직접적으로 호출하지 않고도 상태 시스템의 실행 경로를 테스트할 수 있습니다. 모의 서비스 통합을 사용하도록 상태 시스템

을 구성하려면 모의 구성 파일을 만듭니다. 이 파일에서는 서비스 통합의 원하는 출력을 모의 응답으로 정의하고 모의 응답을 사용하여 실행 경로를 테스트 사례로 시뮬레이션하는 실행을 정의합니다.

Step Functions Local에 모의 구성 파일을 제공하면 실제 서비스 통합을 직접적으로 호출하지 않고 테스트 사례에 지정된 모의 응답을 사용하는 상태 시스템을 실행하여 서비스 통합 직접 호출을 테스트할 수 있습니다.

> **Note**
>
> 모의 구성 파일에 모의 서비스 통합 응답을 지정하지 않는 경우 Step Functions Local은 Step Functions Local을 설정하는 동안 구성한 엔드포인트를 사용하여 AWS 서비스 통합을 호출합니다. Step Functions Local의 엔드포인트 구성 방법은 Step Functions Local의 구성 옵션 설정을 참조하세요.

주제

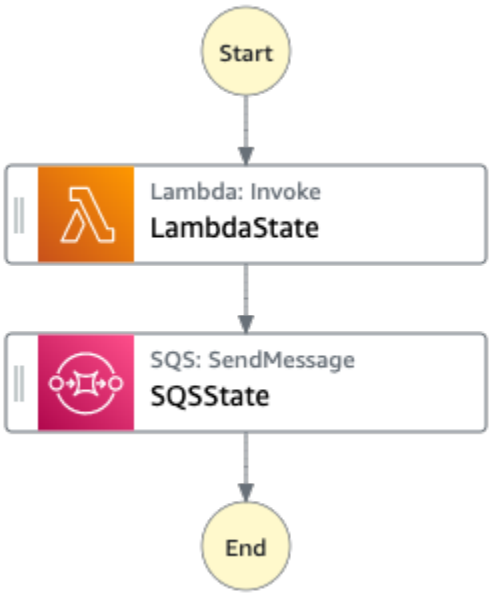
이 주제의 주요 개념

이 주제에서는 다음 목록에 정의된 몇 가지 개념을 사용합니다.

- 모의 서비스 통합 - 실제 서비스를 직접적으로 호출하는 대신 모의 응답을 사용하도록 구성된 Task 상태를 나타냅니다.
- 모의 응답 - Task 상태가 사용하도록 구성될 수 있는 모의 데이터를 나타냅니다.
- 테스트 사례 - 모의 서비스 통합을 사용하도록 구성된 상태 시스템 실행을 나타냅니다.
- 모의 구성 파일 - 모의 서비스 통합, 모의 응답 및 테스트 사례를 정의하는 JSON이 포함된 모의 구성 파일을 나타냅니다.

1단계: 모의 구성 파일에 모의 서비스 통합 지정

Step Functions Local을 사용하여 Step Functions AWS SDK와 최적화된 서비스 통합을 테스트할 수 있습니다. 다음 이미지에서는 상태 시스템 정의 탭에 정의된 상태 시스템을 보여줍니다.



이렇게 하려면 모의 구성 구조 소개에 정의된 섹션이 포함된 모의 구성 파일을 만들어야 합니다.

1. `MockConfigFile.json` 파일을 만들어 모의 서비스 통합으로 테스트를 구성합니다.

   다음 예제에서는 `LambdaState` 및 `SQSState`라는 정의된 상태 2개가 있는 상태 시스템을 참조하는 모의 구성 파일을 보여줍니다.

   Mock configuration file example

   다음은 Lambda 함수를 간접적으로 호출하고 메시지를 Amazon SQS로 전송하여 응답을 모의하는 방법을 보여주는 모의 구성 파일의 예제입니다. 이 예제에서 `LambdaSQSIntegration` 상태 시스템에는 `LambdaState` 및 `SQSState`라는 `Task` 상태를 모의하는 `HappyPath`, `RetryPath` 및 `HybridPath`라는 테스트 사례 3개가 포함되어 있습니다. 이러한 상태는 `MockedLambdaSuccess`, `MockedSQSSuccess` 및 `MockedLambdaRetry` 모의 서비스 응답을 사용합니다. 이러한 모의 서비스 응답은 파일의 `MockedResponses` 섹션에 정의되어 있습니다.

   ```
   {
     "StateMachines":{
       "LambdaSQSIntegration":{
         "TestCases":{
   ```

```
        "HappyPath":{
          "LambdaState":"MockedLambdaSuccess",
          "SQSState":"MockedSQSSuccess"
        },
        "RetryPath":{
          "LambdaState":"MockedLambdaRetry",
          "SQSState":"MockedSQSSuccess"
        },
        "HybridPath":{
          "LambdaState":"MockedLambdaSuccess"
        }
      }
    }
  },
  "MockedResponses":{
    "MockedLambdaSuccess":{
      "0":{
        "Return":{
          "StatusCode":200,
          "Payload":{
            "StatusCode":200,
            "body":"Hello from Lambda!"
          }
        }
      }
    },
    "LambdaMockedResourceNotReady":{
      "0":{
        "Throw":{
          "Error":"Lambda.ResourceNotReadyException",
          "Cause":"Lambda resource is not ready."
        }
      }
    },
    "MockedSQSSuccess":{
      "0":{
        "Return":{
          "MD5OfMessageBody":"3bcb6e8e-7h85-4375-b0bc-1a59812c6e51",
          "MessageId":"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51"
        }
      }
    },
    "MockedLambdaRetry":{
      "0":{
```

```
        "Throw":{
          "Error":"Lambda.ResourceNotReadyException",
          "Cause":"Lambda resource is not ready."
        }
      },
      "1-2":{
        "Throw":{
          "Error":"Lambda.TimeoutException",
          "Cause":"Lambda timed out."
        }
      },
      "3":{
        "Return":{
          "StatusCode":200,
          "Payload":{
            "StatusCode":200,
            "body":"Hello from Lambda!"
          }
        }
      }
    }
  }
}
```

State machine definition

다음은 LambdaSQSIntegration이라는 상태 시스템 정의의 예제입니다. 이 정의에서 LambdaState 및 SQSState라는 서비스 통합 작업 상태 2개를 정의합니다. LambdaState에는 States.ALL에 기반한 재시도 정책이 포함되어 있습니다.

```
{
  "Comment":"This state machine is called: LambdaSQSIntegration",
  "StartAt":"LambdaState",
  "States":{
    "LambdaState":{
      "Type":"Task",
      "Resource":"arn:aws:states:::lambda:invoke",
      "Parameters":{
        "Payload.$":"$",
        "FunctionName":"HelloWorldFunction"
      },
      "Retry":[
        {
```

```
          "ErrorEquals":[
            "States.ALL"
          ],
          "IntervalSeconds":2,
          "MaxAttempts":3,
          "BackoffRate":2
        }
      ],
      "Next":"SQSState"
    },
    "SQSState":{
      "Type":"Task",
      "Resource":"arn:aws:states:::sqs:sendMessage",
      "Parameters":{
        "QueueUrl":"https://sqs.us-east-1.amazonaws.com/123456789012/myQueue",
        "MessageBody.$":"$"
      },
      "End": true
    }
  }
}
```

다음 테스트 사례 중 하나를 사용하여 모의 구성 파일에 참조된 LambdaSQSIntegration 상태 시스템 정의를 실행할 수 있습니다.

- HappyPath - 이 테스트에서는 MockedLambdaSuccess 및 MockedSQSSuccess를 각각 사용하여 LambdaState 및 SQSState의 출력을 모의합니다.
  - LambdaState는 다음 값을 반환합니다.

```
"0":{
  "Return":{
    "StatusCode":200,
    "Payload":{
      "StatusCode":200,
      "body":"Hello from Lambda!"
    }
  }
}
```

  - SQSState는 다음 값을 반환합니다.

```
"0":{
  "Return":{
    "MD5OfMessageBody":"3bcb6e8e-7h85-4375-b0bc-1a59812c6e51",
    "MessageId":"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51"
  }
}
```

- RetryPath - 이 테스트에서는 MockedLambdaRetry 및 MockedSQSSuccess를 각각 사용하여 LambdaState 및 SQSState의 출력을 모의합니다. 또한 LambdaState는 재시도를 4회 수행하도록 구성되어 있습니다. 이러한 시도에 대한 모의 응답은 MockedLambdaRetry 상태에서 정의되고 인덱싱됩니다.
  - 초기 시도는 다음 예제와 같이 원인과 오류 메시지가 포함된 작업 실패로 끝납니다.

```
"0":{
  "Throw": {
    "Error": "Lambda.ResourceNotReadyException",
    "Cause": "Lambda resource is not ready."
  }
}
```

  - 첫 번째 및 두 번째 재시도는 다음 예제와 같이 원인과 오류 메시지가 포함된 작업 실패로 끝납니다.

```
"1-2":{
  "Throw": {
    "Error": "Lambda.TimeoutException",
    "Cause": "Lambda timed out."
  }
}
```

  - 세 번째 재시도는 모의 Lambda 응답의 Payload 섹션의 상태 결과가 포함된 작업 성공으로 끝납니다.

```
"3":{
  "Return": {
    "StatusCode": 200,
    "Payload": {
      "StatusCode": 200,
      "body": "Hello from Lambda!"
    }
  }
```

```
}
```

> **Note**
>
> - 재시도 정책이 있는 상태의 경우 Step Functions Local은 성공 응답을 수신할 때까지 정책에 설정된 재시도 횟수를 모두 활용합니다. 즉, 연속 시도 횟수가 있는 재시도에 대해서는 모의로 표시해야 하며 성공 응답을 반환하기 전의 모든 재시도 횟수를 포함해야 합니다.
> - 특정 재시도(예: 재시도 “3”)에 모의 응답을 지정하지 않으면 상태 시스템 실행이 실패합니다.

- `HybridPath` - 이 테스트는 `LambdaState` 출력을 모의합니다. `LambdaState`가 성공적으로 실행되고 모의 데이터를 응답으로 수신한 후에`SQSState`는 프로덕션에 지정된 리소스에 대한 실제 서비스를 직접적으로 호출합니다.

모의 서비스 통합으로 테스트 실행을 시작하는 방법은 3단계: 모의 서비스 통합 테스트 실행을 참조하세요.

2. 모의 응답 구조가 통합 서비스를 직접적으로 호출할 때 수신한 실제 서비스 응답 구조를 따르는지 확인합니다. 모의 응답의 구조적 요구 사항은 모의 서비스 통합 구성을 참조하세요.

   이전 예제 모의 구성 파일에서는 `MockedLambdaSuccess` 및 `MockedLambdaRetry`에 정의된 모의 응답이 `HelloFromLambda` 직접 호출에서 반환되는 실제 응답 구조를 따릅니다.

   > **Important**
   >
   > AWS 서비스 응답 구조는 서비스마다 다를 수 있습니다. Step Functions Local은 모의 응답 구조가 실제 서비스 응답 구조를 따르는지 검증하지 않습니다. 테스트하기 전에 모의 응답이 실제 응답을 따르는지 확인해야 합니다. 서비스 응답 구조를 검토하려면 Step Functions를 사용하여 실제 서비스를 직접적으로 호출하거나 해당 서비스에 대한 설명서를 보면 됩니다.

2단계: Step Functions Local에 모의 구성 파일 제공

다음 방법 중 하나로 Step Functions Local에 모의 구성 파일을 제공할 수 있습니다.

Docker

> **Note**
>
> Docker 버전의 Step Functions Local을 사용하는 경우 환경 변수만 사용하여 모의 구성 파일을 제공할 수 있습니다. 또한 초기 서버 부팅 시 모의 구성 파일을 Step Functions Local 컨테이너에 마운트해야 합니다.

모의 구성 파일을 Step Functions Local 컨테이너 내 임의의 디렉터리에 마운트합니다. 그런 다음 컨테이너에 있는 모의 구성 파일의 경로가 포함된 SFN_MOCK_CONFIG 환경 변수를 설정합니다. 이 방법을 사용하면 환경 변수에 파일 경로와 이름이 포함되어 있는 한 모의 구성 파일 이름을 원하는 대로 지정할 수 있습니다.

다음 명령에서는 도커 이미지를 시작하는 형식을 보여줍니다.

```
docker run -p 8083:8083
--mount type=bind,readonly,source={absolute path to mock config file},destination=/
home/StepFunctionsLocal/MockConfigFile.json
-e SFN_MOCK_CONFIG="/home/StepFunctionsLocal/MockConfigFile.json" amazon/aws-
stepfunctions-local
```

다음 예제에서는 명령을 사용하여 도커 이미지를 시작합니다.

```
docker run -p 8083:8083
--mount type=bind,readonly,source=/Users/admin/Desktop/workplace/
MockConfigFile.json,destination=/home/StepFunctionsLocal/MockConfigFile.json
-e SFN_MOCK_CONFIG="/home/StepFunctionsLocal/MockConfigFile.json" amazon/aws-
stepfunctions-local
```

JAR File

다음 방법 중 하나를 사용하여 Step Functions Local에 모의 구성 파일을 제공합니다.

- Step FunctionsLocal.jar와 동일한 디렉토리에 모의 구성 파일을 배치합니다. 이 방법을 사용할 때는 모의 구성 파일 MockConfigFile.json 이름을 지정해야 합니다.
- Step Functions Local을 실행하는 세션에서 SFN_MOCK_CONFIG 환경 변수를 모의 구성 파일의 전체 경로로 설정합니다. 이 방법을 사용하면 환경 변수에 파일 경로와 이름이 포함되어 있는 한 모의 구성 파일 이름을 원하는 대로 지정할 수 있습니다. 다음 예제에서 SFN_MOCK_CONFIG 변

수는 /home/workspace 디렉터리에 있는 EnvSpecifiedMockConfig.json 모의 구성 파일을 가리키도록 설정됩니다.

```
export SFN_MOCK_CONFIG="/home/workspace/EnvSpecifiedMockConfig.json"
```

> **Note**
>
> - Step Functions Local에 SFN_MOCK_CONFIG 환경 변수를 제공하지 않으면 기본적으로 Step Functions Local을 시작한 디렉토리에 있는 MockConfigFile.json 모의 구성 파일을 읽으려고 시도합니다.
> - 모의 구성 파일을 Step FunctionsLocal.jar와 동일한 디렉토리에 배치하고 SFN_MOCK_CONFIG 환경 변수를 설정하면 Step Functions Local은 환경 변수로 지정된 파일을 읽습니다.

3단계: 모의 서비스 통합 테스트 실행

모의 구성 파일을 만들어 Step Functions Local에 제공한 후 모의 서비스 통합을 사용하여 모의 구성 파일에 구성된 상태 시스템을 실행합니다. 그런 다음 API 작업을 사용하여 실행 결과를 확인합니다.

1. 앞서 언급한 모의 구성 파일의 정의를 기반으로 상태 시스템을 만듭니다.

```
aws stepfunctions create-state-machine \
    --endpoint http://localhost:8083 \
    --definition "{\"Comment\":\"Thisstatemachineiscalled:LambdaSQSIntegration
\",\"StartAt\":\"LambdaState\",\"States\":{\"LambdaState\":{\"Type\":
\"Task\",\"Resource\":\"arn:aws:states:::lambda:invoke\",\"Parameters
\":{\"Payload.$\":\"$\",\"FunctionName\":\"arn:aws:lambda:us-
east-1:123456789012:function:HelloWorldFunction\"},\"Retry\":[{\"ErrorEquals
\":[\"States.ALL\"],\"IntervalSeconds\":2,\"MaxAttempts\":3,\"BackoffRate
\":2}],\"Next\":\"SQSState\"},\"SQSState\":{\"Type\":\"Task\",\"Resource\":
\"arn:aws:states:::sqs:sendMessage\",\"Parameters\":{\"QueueUrl\":\"https://
sqs.us-east-1.amazonaws.com/123456789012/myQueue\",\"MessageBody.$\":\"$\"},\"End
\":true}}}" \
    --name "LambdaSQSIntegration" --role-arn "arn:aws:iam::123456789012:role/
service-role/LambdaSQSIntegration"
```

2. 모의 서비스 통합을 사용하여 상태 시스템을 실행합니다.

모의 구성 파일을 사용하려면 모의 구성 파일에 구성된 상태 시스템에서 StartExecution API 직접 호출을 수행합니다. 이렇게 하려면 StartExecution에서 사용하는 상태 시스템 ARN에 접미사 #*test_name*을 추가합니다. *test_name*은 같은 모의 구성 파일에서 상태 시스템에 대해 구성된 테스트 사례입니다.

다음 명령은 LambdaSQSIntegration 상태 시스템과 모의 구성을 사용하는 예제입니다. 이 예시에서 LambdaSQSIntegration 상태 시스템은 1단계: 모의 구성 파일에 모의 서비스 통합 지정에 정의된 HappyPath 테스트를 통해 실행됩니다. HappyPath 테스트에는 LambdaState 및 SQSState 상태에서 MockedLambdaSuccess 및 MockedSQSSuccess 모의 서비스 응답을 사용하여 수행하는 모의 서비스 통합 직접 호출을 처리하기 위한 실행 구성이 포함되어 있습니다.

```
aws stepfunctions start-execution \
    --endpoint http://localhost:8083 \
    --name executionWithHappyPathMockedServices \
    --state-machine arn:aws:states:us-
east-1:123456789012:stateMachine:LambdaSQSIntegration#HappyPath
```

3. 상태 시스템 실행 응답을 봅니다.

   모의 서비스 통합 테스트를 사용한 StartExecution 직접 호출에 대한 응답은 보통 실행 ARN과 시작 날짜를 반환하는 StartExecution 직접 호출에 대한 응답과 동일합니다.

   다음은 모의 서비스 통합 테스트를 사용한 StartExecution 직접 호출에 대한 응답의 예제입니다.

```
{
   "startDate":"2022-01-28T15:03:16.981000-05:00",
   "executionArn":"arn:aws:states:us-
east-1:123456789012:execution:LambdaSQSIntegration:executionWithHappyPathMockedServices"
}
```

4. ListExecutions, DescribeExecution 또는 GetExecutionHistory API를 직접 호출하여 실행 결과를 확인합니다.

```
aws stepfunctions get-execution-history \
    --endpoint http://localhost:8083 \
    --execution-arn arn:aws:states:us-
east-1:123456789012:execution:LambdaSQSIntegration:executionWithHappyPathMockedServices
```

다음 예제에서는 2단계에 표시된 예제 응답의 실행 ARN을 사용한 GetExecutionHistory 직접 호출에 대한 응답의 일부를 보여줍니다. 이 예제에서 LambdaState 및 SQSState의 출력은 모의 구성 파일의 MockedLambdaSuccess 및 MockedSQSSuccess에 정의된 모의 데이터입니다. 또한 모의 데이터는 실제 서비스 통합을 직접적으로 호출하여 반환된 데이터를 사용하는 방식과 동일한 방식으로 사용됩니다. 또한 이 예제에서는 LambdaState의 출력이 입력으로 SQSState에 전달됩니다.

```
{
    "events": [
        ...
        {
            "timestamp": "2021-12-02T19:39:48.988000+00:00",
            "type": "TaskStateEntered",
            "id": 2,
            "previousEventId": 0,
            "stateEnteredEventDetails": {
                "name": "LambdaState",
                "input": "{}",
                "inputDetails": {
                    "truncated": false
                }
            }
        },
        ...
        {
            "timestamp": "2021-11-25T23:39:10.587000+00:00",
            "type": "LambdaFunctionSucceeded",
            "id": 5,
            "previousEventId": 4,
            "lambdaFunctionSucceededEventDetails": {
                "output": "{\"statusCode\":200,\"body\":\"\\\"Hello from Lambda!\\
\"\"}",
                "outputDetails": {
                    "truncated": false
                }
            }
        },
        ...
            "timestamp": "2021-12-02T19:39:49.464000+00:00",
            "type": "TaskStateEntered",
            "id": 7,
            "previousEventId": 6,
```

```
            "stateEnteredEventDetails": {
                "name": "SQSState",
                "input": "{\"statusCode\":200,\"body\":\"\\\"Hello from Lambda!\\
\"\"}",
                "inputDetails": {
                    "truncated": false
                }
            }
        },
        ...
        {
            "timestamp": "2021-11-25T23:39:10.652000+00:00",
            "type": "TaskSucceeded",
            "id": 10,
            "previousEventId": 9,
            "taskSucceededEventDetails": {
                "resourceType": "sqs",
                "resource": "sendMessage",
                "output": "{\"MD5OfMessageBody\":\"3bcb6e8e-7h85-4375-
b0bc-1a59812c6e51\",\"MessageId\":\"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51\"}",
                "outputDetails": {
                    "truncated": false
                }
            }
        },
        ...
    ]
}
```

모의 서비스 통합을 위한 구성 파일

모의 서비스 통합을 사용하려면 먼저 모의 구성이 포함된 `MockConfigFile.json`이라는 모의 구성 파일을 만들어야 합니다. 그런 다음 모의 구성 파일과 함께 Step Functions Local을 제공합니다. 이 구성 파일은 테스트 사례를 정의하며 이 사례에는 모의 서비스 통합 응답을 사용하는 모의 상태가 포함됩니다. 다음 섹션에는 모의 상태와 모의 응답이 포함된 모의 구성 구조에 대한 정보가 포함되어 있습니다.

주제

- 모의 구성 구조 소개
- 모의 서비스 통합 구성

모의 구성 구조 소개

모의 구성은 다음 최상위 필드를 포함하는 JSON 객체입니다.

- `StateMachines` - 이 객체의 필드는 모의 서비스 통합을 사용하도록 구성된 상태 시스템을 나타냅니다.
- `MockedResponse` - 이 객체의 필드는 서비스 통합 직접 호출에 대한 모의 응답을 나타냅니다.

다음은 `StateMachine` 정의와 `MockedResponse`가 포함된 모의 구성 파일의 예제입니다.

```
{
  "StateMachines":{
    "LambdaSQSIntegration":{
      "TestCases":{
        "HappyPath":{
          "LambdaState":"MockedLambdaSuccess",
          "SQSState":"MockedSQSSuccess"
        },
        "RetryPath":{
          "LambdaState":"MockedLambdaRetry",
          "SQSState":"MockedSQSSuccess"
        },
        "HybridPath":{
          "LambdaState":"MockedLambdaSuccess"
        }
      }
    }
  },
  "MockedResponses":{
    "MockedLambdaSuccess":{
      "0":{
        "Return":{
          "StatusCode":200,
          "Payload":{
            "StatusCode":200,
            "body":"Hello from Lambda!"
          }
        }
      }
    },
    "LambdaMockedResourceNotReady":{
      "0":{
```

```
        "Throw":{
          "Error":"Lambda.ResourceNotReadyException",
          "Cause":"Lambda resource is not ready."
        }
      }
    },
    "MockedSQSSuccess":{
      "0":{
        "Return":{
          "MD5OfMessageBody":"3bcb6e8e-7h85-4375-b0bc-1a59812c6e51",
          "MessageId":"3bcb6e8e-8b51-4375-b0bc-1a59812c6e51"
        }
      }
    },
    "MockedLambdaRetry":{
      "0":{
        "Throw":{
          "Error":"Lambda.ResourceNotReadyException",
          "Cause":"Lambda resource is not ready."
        }
      },
      "1-2":{
        "Throw":{
          "Error":"Lambda.TimeoutException",
          "Cause":"Lambda timed out."
        }
      },
      "3":{
        "Return":{
          "StatusCode":200,
          "Payload":{
            "StatusCode":200,
            "body":"Hello from Lambda!"
          }
        }
      }
    }
  }
}
```

모의 구성 필드 참조

다음 섹션에서는 모의 구성에서 정의해야 하는 최상위 객체 필드를 설명합니다.

- StateMachines
- MockedResponses

StateMachines

StateMachines 객체는 모의 서비스 통합을 사용할 상태 시스템을 정의합니다. 각 상태 시스템의 구성은 StateMachines의 최상위 필드로 표시됩니다. 필드 이름은 상태 시스템 이름이고 값은 TestCases라는 단일 필드가 포함된 객체입니다. 이 필드는 해당 상태 시스템의 테스트 사례를 나타냅니다.

다음 구문에서는 테스트 사례가 2개 있는 상태 시스템을 보여줍니다.

```
"MyStateMachine": {
  "TestCases": {
    "HappyPath": {
      ...
    },
    "SadPath": {
      ...
    }
  }
```

TestCases

TestCases의 필드는 상태 시스템의 개별 테스트 사례를 나타냅니다. 각 테스트 사례 이름은 상태 시스템마다 고유해야 하며 각 테스트 사례 값은 상태 시스템의 Task 상태에 사용할 모의 응답을 지정하는 객체입니다.

다음 TestCase 예제에서는 두 Task 상태를 MockedResponses 2개에 연결합니다.

```
"HappyPath": {
  "SomeTaskState": "SomeMockedResponse",
  "AnotherTaskState": "AnotherMockedResponse"
}
```

MockedResponses

MockedResponses는 고유한 필드 이름을 가진 모의 응답 객체가 여러 개 포함된 객체입니다. 모의 응답 객체는 모의 Tase 상태를 간접적으로 호출할 때마다 성공한 결과나 오류 출력을 정의합니다. “0”,

“1”, “2” 및 “3”과 같은 개별 정수 문자열이나 “0-1”, “2-3"과 같은 포괄적인 정수 범위를 사용하여 간접 호출 번호를 지정합니다.

Task를 모의할 때 모든 간접 호출에 모의 응답을 지정해야 합니다. 응답에는 값이 모의 Task 간접 호출에 대한 결과나 오류 출력인 Return 또는 Throw라는 단일 필드가 포함되어야 합니다. 모의 응답을 지정하지 않으면 상태 시스템 실행이 실패합니다.

다음은 Throw 및 Return 객체가 있는 MockedResponse의 예제입니다. 이 예제에서는 상태 시스템이 처음 3번 실행되면 "0-2"에 지정된 응답이 반환되고 상태 시스템이 4번 실행되면 "3"에 지정된 응답이 반환됩니다.

```
"SomeMockedResponse": {
  "0-2": {
    "Throw": {
      ...
    }
  },
  "3": {
    "Return": {
      ...
    }
  }
}
```

> **Note**
>
> Map 상태를 사용 중이고 Map 상태에 대해 예측 가능한 응답을 보장하려면 maxConcurrency 값을 1로 설정하세요. 값을 1보다 크게 설정하면 Step Functions Local이 여러 반복을 동시에 실행하므로 반복 전반에 걸친 상태의 전체 실행 순서를 예측할 수 없게 됩니다. 이로 인해 Step Functions Local이 한 실행에서 다음 실행까지의 반복 상태에 서로 다른 모의 응답을 사용할 수도 있습니다.

반환

Return은 MockedResponse 객체 필드로 표시됩니다. 모의 Task 상태의 성공 결과를 지정합니다.

다음은 Lambda 함수에서 Invoke 직접 호출에 대한 모의 응답이 포함된 Return 객체의 예입니다.

```
"Return": {
```

```
  "StatusCode": 200,
  "Payload": {
    "StatusCode": 200,
    "body": "Hello from Lambda!"
  }
}
```

Throw

Throw는 MockedResponse 객체 필드로 표시됩니다. 실패한 Task의 오류 출력을 지정합니다. Throw 값은 문자열 값이 있는 Error 및 Cause 필드가 포함된 객체여야 합니다. 또한 MockConfigFile.json의 Error 필드에 지정하는 문자열 값은 상태 시스템의 Retry 및 Catch 섹션에서 처리된 오류와 일치해야 합니다.

다음은 Lambda 함수에서 Invoke 직접 호출에 대한 모의 응답이 포함된 Throw 객체의 예입니다.

```
"Throw": {
  "Error": "Lambda.TimeoutException",
  "Cause": "Lambda timed out."
}
```

모의 서비스 통합 구성

Step Functions Local을 사용하여 모든 서비스 통합을 모의할 수 있습니다. 하지만 Step Functions Local은 모의를 실제 API와 동일하게 적용하지 않습니다. 모의 Task는 서비스 엔드포인트를 직접적으로 호출하지 않습니다. 모의 응답을 지정하지 않으면 Task는 서비스 엔드포인트를 직접적으로 호출하려고 시도합니다. 또한 Step Functions Local은 .waitForTaskToken을 사용하여 Task를 모의할 때 자동으로 작업 토큰을 생성합니다.

Step Functions 모범 사례

다음의 AWS Step Functions 워크플로우 구현 모범 사례는 구현 성능을 최적화하는 데 활용할 수 있습니다.

주제

제한 시간을 사용하여 실행 멈춤 방지

기본적으로 Amazon States Language는 상태 시스템 정의 제한 시간을 지정하지 않습니다. 명시적 제한 시간 없이 Step Functions는 활동 작업자의 응답만 사용하여 작업이 완료되었는지를 확인합니다. 문제가 발생하고 `Activity` 및 `Task` 상태에 `TimeoutSeconds` 필드가 지정되지 않으면 돌아오지 않는 응답을 기다리기 위해 실행이 멈춥니다.

이를 방지하려면 상태 시스템에 `Task`를 만들 때 적절한 제한 시간을 지정합니다. 예:

```
"ActivityState": {
  "Type": "Task",
  "Resource": "arn:aws:states:us-east-1:123456789012:activity:HelloWorld",
  "TimeoutSeconds": 300,
  "Next": "NextState"
}
```

작업 토큰(.waitForTaskToken)과 함께 콜백을 사용하는 경우 하트비트를 사용하고 `Task` 상태 정의에 `HeartbeatSeconds` 필드를 추가하는 것이 좋습니다. `HeartbeatSeconds`를 작업 제한 시간보다 작게 설정할 수 있으므로 하트비트 오류로 인해 워크플로가 실패하는 경우 작업이 완료되는 데 오랜 시간이 걸리는 것이 아니라 작업 실패가 원인이라는 것을 알 수 있습니다.

```
{
  "StartAt": "Push to SQS",
  "States": {
    "Push to SQS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
      "HeartbeatSeconds": 600,
      "Parameters": {
        "MessageBody": { "myTaskToken.$": "$$.Task.Token" },
        "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/push-based-queue"
      },
      "ResultPath": "$.SQS",
      "End": true
    }
  }
}
```

자세한 내용은 Amazon States Language 문서의 태스크 상태를 참조하세요.

> **Note**
>
> Amazon States Language 정의의 `TimeoutSeconds` 필드를 사용하여 상태 시스템 제한 시간을 설정할 수 있습니다. 자세한 내용은 상태 시스템 구조 섹션을 참조하세요.

대용량 페이로드를 전달하는 대신 Amazon S3 ARN 사용

상태 사이에 대용량 데이터 페이로드를 전달하는 실행은 종료될 수 있습니다. 상태 간에 전달하는 데이터가 256KB를 초과할 수 있는 경우 Amazon Simple Storage Service(S3)를 사용하여 데이터를 저장하고 `Payload` 파라미터에서 버킷의 Amazon 리소스 이름(ARN)을 파싱하여 버킷 이름과 키 값을 가져옵니다. 또는 실행에서 더 작은 용량의 페이로드를 전달하도록 구현을 조정합니다.

다음 예제에서 상태 시스템은 입력을 Amazon S3 버킷의 JSON 파일을 처리하는 AWS Lambda 함수에 전달합니다. 이 상태 시스템을 실행하면 Lambda 함수에서 JSON 파일 콘텐츠를 읽고 파일 콘텐츠를 출력으로 반환합니다.

Lambda 함수 생성

*`pass-large-payload`*라는 다음 Lambda 함수는 특정 Amazon S3 버킷에 저장된 JSON 파일의 콘텐츠를 읽습니다.

> **Note**
>
> 이 Lambda 함수를 만든 후에는 해당 IAM 역할에 Amazon S3 버킷에서 읽을 수 있는 적절한 권한을 제공해야 합니다. 예를 들어 AmazonS3ReadOnlyAccess 권한을 Lambda 함수 역할에 연결합니다.

```
import json
import boto3
import io
import os

s3 = boto3.client('s3')

def lambda_handler(event, context):
    event = event['Input']
    final_json = str()

    s3 = boto3.resource('s3')
    bucket = event['bucket'].split(':')[-1]
    filename = event['key']
    directory = "/tmp/{}".format(filename)

    s3.Bucket(bucket).download_file(filename, directory)

    with open(directory, "r") as jsonfile:

        final_json = json.load(jsonfile)

    os.popen("rm -rf /tmp")

    return final_json
```

상태 시스템 만들기

다음 상태 시스템은 이전에 만든 Lambda 함수를 간접적으로 호출합니다.

```
{
   "StartAt":"Invoke Lambda function",
   "States":{
      "Invoke Lambda function":{
         "Type":"Task",
```

```
         "Resource":"arn:aws:states:::lambda:invoke",
         "Parameters":{
            "FunctionName":"arn:aws:lambda:us-east-2:123456789012:function:pass-large-
payload",
            "Payload":{
               "Input.$":"$"
            }
         },
         "OutputPath": "$.Payload",
         "End":true
      }
   }
}
```

대량의 데이터를 입력에 전달하는 대신 해당 데이터를 Amazon S3 버킷에 저장하고 버킷의 Amazon 리소스 이름(ARN)을 Payload 파라미터에 전달하여 버킷 이름과 키 값을 가져올 수 있습니다. 그러면 Lambda 함수에서 해당 ARN을 사용하여 데이터에 직접 액세스할 수 있습니다. 다음은 상태 시스템 실행을 위한 입력의 예제입니다. 여기서 데이터는 data.json이라는 Amazon S3 버킷의 *large-payload-json*에 저장됩니다.

```
{
  "key": "data.json",
  "bucket": "arn:aws:s3:::large-payload-json"
}
```

내역 할당량 도달 방지

AWS Step Functions에는 실행 이벤트 내역의 하드 할당량 25,000개 항목이 있습니다. 실행이 이벤트 24,999개에 도달하면 다음 이벤트가 발생할 때까지 대기합니다.

- 이벤트 번호 25,000이 ExecutionSucceeded이면 실행이 성공적으로 완료됩니다.
- 이벤트 번호 25,000이 ExecutionSucceeded가 아니면 ExecutionFailed 이벤트가 로깅되고 내역 한도에 도달하여 상태 시스템 실행이 실패합니다.

이 장기 실행 할당량에 도달하지 않게 하려면 다음 해결 방법 중 하나를 시도하면 됩니다.

- 분산 모드에서 Map 상태를 사용합니다. 이 모드에서 Map 상태는 각 반복을 하위 워크플로 실행으로 실행하므로 병렬 하위 워크플로를 동시에 최대 10,000개까지 실행할 수 있습니다. 각 하위 워크플로 실행에는 상위 워크플로와 별개인 자체 실행 내역이 있습니다.

- 실행 중인 Task 상태에서 직접 새 상태 시스템 실행을 시작합니다. 이러한 중첩된 워크플로 실행을 시작하려면 필요한 파라미터와 함께 상위 상태 시스템에서 Step Functions의 StartExecution API 작업을 사용합니다. 중첩된 워크플로 사용에 대한 자세한 내용은 작업 상태에서 워크플로 실행 시작 또는 Step Functions API 작업을 사용하여 새 실행 계속하기 자습서를 참조하세요.

  > **Tip**
  >
  > 중첩된 워크플로 예제를 AWS 계정에 배포하려면 모듈 13 - 중첩된 Express 워크플로를 참조하세요.

- AWS Lambda 함수를 사용하는 패턴을 구현합니다. 그러면 상태 시스템 실행을 새로 시작하여 여러 워크플로 실행에서 진행 중인 작업을 분할할 수 있습니다. 자세한 내용은 Lambda 함수를 사용하여 새 실행 계속하기 자습서를 참조하십시오.

Lambda 서비스 예외 처리

AWS Lambda는 가끔 일시적 서비스 오류가 발생할 수 있습니다. 이 경우 Lambda를 간접적으로 호출하면 ClientExecutionTimeoutException, ServiceException, AWSLambdaException 또는 SdkClientException과 같은 500 오류가 발생합니다. 모범 사례로서, 상태 시스템에서 이러한 예외를 사전에 처리하여 Lambda 함수 간접 호출을 Retry하거나 오류를 Catch하는 것이 좋습니다.

Lambda 오류가 Lambda.*ErrorName*으로 보고됩니다. Lambda 서비스 예외 오류를 다시 시도하려면 다음 Retry 코드를 사용하면 됩니다.

```
"Retry": [ {
   "ErrorEquals": [ "Lambda.ClientExecutionTimeoutException",
 "Lambda.ServiceException", "Lambda.AWSLambdaException", "Lambda.SdkClientException"],
   "IntervalSeconds": 2,
   "MaxAttempts": 6,
   "BackoffRate": 2
} ]
```

> **Note**
>
> Lambda에서 처리되지 않은 오류는 오류 출력에서 Lambda.Unknown으로 보고됩니다. 메모리 부족 오류 및 함수 시간 초과도 여기에 포함됩니다. Lambda.Unknown, States.ALL 또는 States.TaskFailed를 일치시켜 이러한 오류를 처리할 수 있습니다. Lambda에서 최대 간접 호출 수에 도달하면 오류는 Lambda.TooManyRequestsException입니다. Lambda 함수

오류에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 오류 처리 및 자동 재시도를 참조하세요.

자세한 내용은 다음을 참조하세요.

- 오류 후 재시도
- Step Functions 상태 시스템을 사용하여 오류 조건 처리
- Lambda 간접 호출 오류

활동 작업을 폴링할 때 지연 시간 방지

`GetActivityTask` API는 `taskToken`을 정확히 한 번만 제공하도록 설계되어 있습니다. 활동 작업자와 의사 소통 중 `taskToken`이 삭제되면`GetActivityTask` 시간이 초과될 때까지 응답을 기다리는 60초 동안 많은 `GetActivityTask` 요청이 차단될 수 있습니다.

응답을 기다리는 폴링 수가 적은 경우에만 모든 요청을 차단된 요청 뒤쪽의 대기열에 넣고 정지합니다. 그러나 활동 Amazon 리소스 이름(ARN)마다 미해결 폴링이 상당수 있고 요청 일부가 대기 상태에 머물러 있으면 계속 `taskToken`을 가져와 작업을 처리할 수 있는 경우가 많습니다.

프러덕션 시스템의 경우 각 시점에서 활동 ARN당 100개 이상의 공개 폴링을 추천합니다. 하나의 폴링이 차단되고 이 폴링의 일부를 뒤쪽의 대기열에 넣으면 `taskToken`를 받아 작업을 처리할 수 있는 요청이 여전히 많은 반면 `GetActivityTask` 요청은 차단됩니다.

작업에 대한 폴링 시 이러한 종류의 지연 시간 문제를 피하려면:

- 활동 작업자 구현 시 작업에서 별도의 스레드로 poller를 구현하십시오.
- 각 시점에서 활동 ARN당 100개 이상의 공개 폴링을 하십시오.

> **Note**
>
> ARN당 100개의 공개 폴링으로 확장하는 것은 비용이 많이 들 수 있습니다. 예를 들어 ARN당 Lambda 함수 폴링 100개 보유 비용이 폴링 스레드가 100개 있는 단일 Lambda 함수 보유에 비해 100배 이상 높습니다. 지연 시간 단축 및 비용 최적화를 모두 달성하려면 비동기 I/O가 있는 언어를 사용하고 작업자마다 여러 개의 폴링 스레드를 구현합니다. Poller 스레드가 작업 스레드와 분리되어 있는 경우 활동 작업자의 예제는 Ruby 활동 작업자 예제를 참조하십시오.

활동 및 활동 작업자에 대한 자세한 내용은 활동을 참조하십시오.

표준 또는 Express 워크플로 선택

AWS Step Functions은 Express 워크플로를 선택할 수 있는 옵션을 사용하여 표준 워크플로를 기본 워크플로 유형으로 제공합니다.

장기 실행되고 내구성이 뛰어나며 감사 가능한 워크플로가 필요한 경우에는 표준 워크플로를 선택하고 대용량 이벤트 처리 워크로드의 경우에는 Express 워크플로를 선택할 수 있습니다. 상태 머신 실행은 선택한 Type에 따라 다르게 작동합니다. 선택한 Type은 상태 머신이 생성된 후에는 변경할 수 없습니다.

- 표준 워크플로와 Express 워크플로 간의 차이점에 대한 자세한 내용은 표준 워크플로와 Express 워크플로 비교를 참조하세요.
- Step Functions를 사용하여 서버리스 워크플로를 빌드하는 동안 비용을 최적화하는 방법에 대한 자세한 내용은 Express 워크플로를 사용하여 비용 최적화을 참조하세요.

Amazon CloudWatch Logs 리소스 정책 크기 제한

CloudWatch Logs 리소스 정책은 5,120자로 제한됩니다. CloudWatch Logs는 정책이 이 크기 제한에 도달하는 것을 감지하면 `/aws/vendedlogs/`로 시작하는 로그 그룹을 자동으로 활성화합니다.

로깅이 활성화된 상태 시스템을 만들면 Step Functions에서 지정된 로그 그룹으로 CloudWatch Logs 리소스 정책을 업데이트해야 합니다. CloudWatch Logs 리소스 정책 크기 한도에 도달하지 않도록 CloudWatch Logs 로그 그룹 이름에 접두사 `/aws/vendedlogs/`를 추가합니다. Step Functions 콘솔에서 로그 그룹을 만들면 로그 그룹 이름에 접두사 `/aws/vendedlogs/states`가 추가됩니다. 자세한 내용은 특정 AWS 서비스에서 로깅 활성화를 참조하세요.

CloudWatch Logs에 액세스할 수 없는 경우 자세한 내용은 Unable to access the CloudWatch Logs을 참조하세요.

다른 서비스와 AWS Step Functions 함께 사용

다른 API를 AWS 서비스 조정하거나 타사 API를 호출하는 방법에 대해 알아보세요. AWS Step Functions

주제

다른 서비스에 전화해 보세요. AWS

AWS 서비스 통합을 사용하면 워크플로우에서 직접 API 작업을 호출하고 실행을 조정할 수 있습니다. Step Functions의 AWS SDK 통합을 사용하여 상태 머신에서 200개가 넘는 AWS 서비스를 직접 호출할 수 있으므로 9,000개 이상의 API 작업에 액세스할 수 있습니다. 또한 Step Functions의 최적화된 통합을 사용할 수 있으며 각 통합은 워크플로에 대한 특수 기능을 제공하도록 사용자 지정되었습니다. 일부 API 작업은 두 가지 통합 유형 모두에서 사용 가능합니다. 가능하면 최적화된 통합을 사용하는 것이 좋습니다.

Amazon States Language의 `Task` 상태에서 이러한 서비스를 직접 조정합니다. 예를 들어 Step Functions를 사용하여 다른 서비스를 직접적으로 호출할 수 있습니다.

- AWS Lambda 함수를 호출합니다.
- AWS Batch 작업을 실행한 다음 결과에 따라 다른 작업을 수행합니다.
- Amazon DynamoDB에서 항목을 삽입하거나 가져옵니다.
- Amazon Elastic Container Service(Amazon ECS) 작업을 실행하고 작업이 완료될 때까지 기다립니다.
- Amazon Simple Notification Service(SNS) 주제에 게시합니다.
- Amazon Simple Queue Service(Amazon SQS)의 메시지를 전송합니다.
- AWS Glue 또는 Amazon에서 작업을 SageMaker 관리하십시오.

- Amazon EMR 작업 실행을 위한 워크플로를 빌드합니다.
- AWS Step Functions 워크플로 실행 시작.

최적화된 통합

워크플로 상황에 맞는 특수 기능을 제공하도록 Step Functions에서 최적화된 통합을 사용자 지정했습니다. 예를 들어 Lambda Invoke는 이스케이프된 JSON의 API 출력을 JSON 객체로 변환합니다. AWS BatchSubmitJob을 사용하면 작업이 완료될 때까지 실행을 일시 중지할 수 있습니다. 첫 번째 최적화된 통합 집합은 2018년에 출시되었으며 현재 API가 50개가 넘게 있습니다.

AWS SDK 통합

AWS SDK 통합은 SDK를 사용하는 표준 API 호출과 똑같이 작동합니다. AWS 상태 머신 정의에서 직접 200개 이상의 AWS 서비스에 걸쳐 9,000개 이상의 API를 호출할 수 있는 기능을 제공합니다.

통합 패턴 지원

표준 워크플로와 익스프레스 워크플로는 동일한 통합을 지원하지만 동일한 통합 패턴은 지원하지 않습니다.

- 최적화된 통합 패턴 지원은 통합마다 다릅니다.
- 익스프레스 워크플로는 작업 실행 (.sync) 또는 콜백 대기 (. waitForTask토큰).
- 자세한 정보는 표준 워크플로와 Express 워크플로 비교을 참조하세요.

Standard Workflows

지원되는 서비스 통합

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
최적화된 통합	Amazon API Gateway	✓		✓
	Amazon Athena	✓	✓	
	AWS Batch	✓	✓	

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
	Amazon Bedrock	✓	✓	✓
	AWS CodeBuild	✓	✓	
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓	✓	✓
	Amazon EKS	✓	✓	✓
	Amazon EMR	✓	✓	
	Amazon EMR on EKS	✓	✓	
	Amazon EMR Serverless	✓	✓	
	Amazon EventBridge	✓		✓
	AWS Glue	✓	✓	
	AWS Glue DataBrew	✓	✓	
	AWS Lambda	✓		✓
	AWS Elemental MediaConvert	✓	✓	
	Amazon SageMaker	✓	✓	
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
AWS SDK 통합	200개 초과	✓		✓

Express Workflows

지원되는 서비스 통합

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
최적화된 통합	Amazon API Gateway	✓		
	Amazon Athena	✓		
	AWS Batch	✓		
	Amazon Bedrock	✓		
	AWS CodeBuild	✓		
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓		
	Amazon EKS	✓		
	Amazon EMR	✓		
	Amazon EMR on EKS	✓		
	Amazon EMR Serverless	✓		

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
	Amazon EventBridge	✓		
	AWS Glue	✓		
	AWS Glue DataBrew	✓		
	AWS Lambda	✓		
	AWS Elemental MediaConvert	✓		
	Amazon SageMaker	✓		
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		
AWS SDK 통합	200개 초과	✓		

크로스 계정 액세스

Step Functions는 AWS 계정 워크플로우에서 서로 다르게 구성된 리소스에 대한 계정 간 액세스를 제공합니다. Step Functions 서비스 통합을 사용하면 AWS 리소스 기반 정책이나 교차 계정 호출을 AWS 서비스 지원하지 않는 경우에도 모든 계정 간 리소스를 호출할 수 있습니다.

자세한 정보는 워크플로의 다른 AWS 계정 리소스에 액세스을 참조하세요.

AWS SDK 서비스 통합

AWS Step Functions 와 AWS 서비스통합되어 워크플로우에서 각 서비스의 API 작업을 호출할 수 있습니다. Step Functions의 AWS SDK 통합을 사용하여 상태 머신에서 거의 모든 AWS 서비스 API 작업

을 호출할 수 있습니다. 또한 Step Functions의 최적화된 통합을 사용할 수 있으며 각 통합은 워크플로에 대한 특수 기능을 제공하도록 사용자 정의되었습니다.

일부 서비스 또는 SDK는 일시적 또는 영구적으로 AWS SDK 통합으로 제공되지 않을 수 있습니다. 최근에 출시된 서비스는 나중에 업데이트될 때까지 SDK 상호 작용을 사용하지 못할 수 있습니다. 일부 서비스에는 고객별 엔드포인트 지정과 같은 사용자 지정된 구성이 필요하며 이는 최적화된 통합에 더 적합할 수 있습니다. 오디오 또는 비디오 스트리밍용 SDK와 같은 다른 SDK는 워크플로에서 사용하기에 적합하지 않습니다. 마지막으로, 일부 서비스는 Step Functions에서 수행하는 특정 내부 검증을 통과할 때까지 보류될 수 있습니다.

> **Tip**
>
> SDK 통합을 사용하는 워크플로의 예를 사용자 AWS 계정환경에 배포하려면 워크숍의 모듈 9 - AWSAWS SDK 서비스 통합을 참조하십시오. AWS Step Functions

주제

- SDK 서비스 통합 사용 AWS
- 지원되는 AWS SDK 서비스 통합
- 지원되는 서비스에 지원되지 않는 API 작업
- 더 이상 사용되지 않는 AWS SDK 서비스 통합

SDK 서비스 통합 사용 AWS

AWS SDK 통합을 사용하려면 서비스 이름과 API 호출을 지정하고 선택적으로 서비스 통합 패턴을 지정합니다.

> **Note**
>
> - 네이티브 서비스 Step Functions API가 PascalCase CamelCase인 경우에도 의 파라미터는 로 표현됩니다. 예를 들어 Step Functions API 작업 `startSyncExecution`을 사용하고 해당 파라미터를 `StateMachineArn`으로 지정할 수 있습니다.
> - Amazon EC2용 `DescribeLaunchTemplateVersions` API 작업과 같이 열거된 파라미터를 허용하는 API 작업의 경우 파라미터 이름의 복수 버전을 지정합니다. 예를 들어 `DescribeLaunchTemplateVersions` API 작업의 `Filter.N` 파라미터에 `Filters`를 지정합니다.

작업 상태 `Resource` 필드의 Amazon States 언어에서 직접 AWS SDK 서비스를 호출할 수 있습니다. 이렇게 하려면 다음 구문을 사용합니다.

`arn:aws:states:::aws-sdk:`*`serviceName`*`:`*`apiAction`*`.`*`[serviceIntegrationPattern]`*

예를 들어 Amazon EC2의 경우 `arn:aws:states:::aws-sdk:ec2:describeInstances`를 사용할 수 있습니다. 그러면 Amazon EC2 describeInstances API 직접 호출에 정의된 대로 출력이 반환됩니다.

AWS SDK 통합에서 오류가 발생하는 경우 결과 오류 필드는 서비스 이름과 오류 이름을 마침표로 구분하여 구성됩니다. *`ServiceName`*`.`*`ErrorName`* 서비스 이름과 오류 이름 모두 파스칼 표기법으로 표시됩니다. Task 상태의 리소스 필드에서는 서비스 이름이 소문자로도 표시될 수 있습니다. 대상 서비스의 API 참조 설명서에서 잠재적 오류 이름을 찾을 수 있습니다.

예를 들어 `arn:aws:states:::aws-sdk:acmpca:deleteCertificateAuthority` AWS SDK 통합을 사용할 수 있습니다. AWS Private Certificate Authority API 참조는 `DeleteCertificateAuthority` API 작업으로 인해 예를 들어 `ResourceNotFoundException`가 발생할 수 있음을 나타냅니다. 이 오류를 처리하려면 Task 상태의 Retrier 또는 Catcher에 `AcmPca.ResourceNotFoundException` 오류를 지정합니다. Step Functions에서 오류를 처리하는 방법에 대한 자세한 내용은 Step Functions에서 오류 처리 섹션을 참조하세요.

Step Functions는 AWS SDK 연동을 위한 IAM 정책을 자동 생성할 수 없습니다. 상태 시스템을 만든 후에는 IAM 콘솔로 이동하여 역할 정책을 구성해야 합니다. 자세한 정보는 통합 서비스용 IAM 정책을 참조하세요.

SDK 연동 사용 방법에 대한 예시는 AWS SDK 서비스 통합을 사용하여 Amazon S3 버킷 정보 수집 튜토리얼을 참조하십시오. AWS

지원되는 AWS SDK 서비스 통합

다음 표에는 Step Functions에서 지원하는 AWS SDK 서비스 통합이 나와 있습니다. *`Task`* 상태 리소스 열에는 왼쪽의 서비스 이름 열에 지정된 서비스에 대한 통합을 사용할 때 특정 API 작업을 직접적으로 호출하는 구문이 나와 있습니다. 지원 날짜 열에는 서비스 통합이 지원된 날짜에 대한 정보가 있습니다. 또한 오른쪽의 예외 접두사 열에는 각 서비스 통합의 예외 접두사가 나와 있습니다. 이러한 예외 접두사는 Step Functions와의 AWS SDK 서비스 통합을 잘못 수행할 때 생성되는 예외에 존재합니다.

> **Note**
>
> - ***로 표시된 서비스에는 현재 Step Functions에서 지원하지 않는 API 작업이 있습니다. 서비스에 지원되지 않는 작업은 지원되는 서비스에 지원되지 않는 API 작업 표를 참조하세요.

- SDK 연동 지원을 확대하기 위해 출시할 때마다 이루어진 업데이트에 대한 AWS 자세한 내용은 을 참조하십시오. 지원되는 AWS SDK 통합에 대한 변경 로그

Important

API 액션 지원은 분기별로 릴리스됩니다. 새 매개변수와 같이 이미 지원되는 작업에 대한 업데이트는 즉시 제공되지 않을 수 있습니다.

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS AppFabric	`arn:aws:states:::aws-sdk:appfabric: [apiAction]`	2024년 1월 18일	AppFabric
B2B Data Interchange	`arn:aws:states:::aws-sdk:b2bi: [apiAction]`	2024년 1월 18일	B2Bi
AWS Data Exports	`arn:aws:states:::aws-sdk:bcmdataexports: [apiAction]`	2024년 1월 18일	BcmDataExports
Amazon Bedrock	`arn:aws:states:::aws-sdk:bedrock: [apiAction]`	2024년 1월 18일	Bedrock
Amazon Bedrock Agents	`arn:aws:states:::aws-sdk:bedrockagent: [apiAction]`	2024년 1월 18일	BedrockAgent
Amazon Bedrock Runtime Agents	`arn:aws:states:::aws-sdk:bedrockagentruntime: [apiAction]`	2024년 1월 18일	BedrockAgentRuntime

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Bedrock Runtime	`arn:aws:states:::aws-sdk:bedrockruntime:` *`[apiAction]`*	2024년 1월 18일	BedrockRuntime
AWS Clean Rooms	`arn:aws:states:::aws-sdk:cleanroomsml:` *`[apiAction]`*	2024년 1월 18일	CleanRoomsML
Amazon CloudFront KeyValueStore	`arn:aws:states:::aws-sdk:cloudfrontkeyvaluestore:` *`[apiAction]`*	2024년 1월 18일	CloudFrontKeyValueStore
CodeGuru Security	`arn:aws:states:::aws-sdk:codegurusecurity:` *`[apiAction]`*	2024년 1월 18일	CodeGuruSecurity
AWS Cost Optimization Hub	`arn:aws:states:::aws-sdk:costoptimizationhub:` *`[apiAction]`*	2024년 1월 18일	CostOptimizationHub
Amazon DataZone	`arn:aws:states:::aws-sdk:datazone:` *`[apiAction]`*	2024년 1월 18일	DataZone
Amazon EKS Auth	`arn:aws:states:::aws-sdk:eksauth:` *`[apiAction]`*	2024년 1월 18일	EksAuth
AWS Entity Resolution	`arn:aws:states:::aws-sdk:entityresolution:` *`[apiAction]`*	2024년 1월 18일	EntityResolution

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS 프리 티어	`arn:aws:states:::aws-sdk:freetier: [apiAction]`	2024년 1월 18일	FreeTier
Amazon Inspector Scan	`arn:aws:states:::aws-sdk:inspectorscan: [apiAction]`	2024년 1월 18일	InspectorScan
AWS Launch Wizard	`arn:aws:states:::aws-sdk:launchwizard: [apiAction]`	2024년 1월 18일	LaunchWizard
Amazon Managed Blockchain Query	`arn:aws:states:::aws-sdk:managedblockchainquery: [apiAction]`	2024년 1월 18일	ManagedBlockchainQuery
AWS Elemental MediaPackage V2	`arn:aws:states:::aws-sdk:mediapackagev2: [apiAction]`	2024년 1월 18일	MediaPackageV2
AWS HealthImaging	`arn:aws:states:::aws-sdk:medicalimaging: [apiAction]`	2024년 1월 18일	MedicalImaging
Network Manager	`arn:aws:states:::aws-sdk:networkmanager: [apiAction]`	2024년 1월 18일	NetworkManager
AWS Payment Cryptography	`arn:aws:states:::aws-sdk:paymentcryptography: [apiAction]`	2024년 1월 18일	PaymentCryptography

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Payment Cryptography Data	`arn:aws:states:::aws-sdk:paymentcryptographydata: [apiAction]`	2024년 1월 18일	PaymentCryptographyData
AWS Private CA Connector for Active Directory	`arn:aws:states:::aws-sdk:pcaconnectorad: [apiAction]`	2024년 1월 18일	PcaConnectorAd
Amazon Q Business	`arn:aws:states:::aws-sdk:qbusiness: [apiAction]`	2024년 1월 18일	QBusiness
Amazon Q Connect	`arn:aws:states:::aws-sdk:qconnect: [apiAction]`	2024년 1월 18일	QConnect
AWS re:Post	`arn:aws:states:::aws-sdk:repostspace: [apiAction]`	2024년 1월 18일	Repostspace
Amazon Timestream Query	`arn:aws:states:::aws-sdk:timestreamquery: [apiAction]`	2024년 1월 18일	TimestreamQuery
Amazon Timestream Write	`arn:aws:states:::aws-sdk:timestreamwrite: [apiAction]`	2024년 1월 18일	TimestreamWrite
Trusted Advisor	`arn:aws:states:::aws-sdk:trustedadvisor: [apiAction]`	2024년 1월 18일	TrustedAdvisor

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Verified Permissions	`arn:aws:states:::aws-sdk:verifiedpermissions:` *[apiAction]*	2024년 1월 18일	VerifiedPermissions
Amazon WorkSpaces Thin Client	`arn:aws:states:::aws-sdk:workspacesthinclient:` *[apiAction]*	2024년 1월 18일	WorkSpacesThinClient
AWS CloudTrail Data	`arn:aws:states:::aws-sdk:cloudtrailddata:` *[apiAction]*	2023년 6월 16일	CloudTrailData
Amazon CloudWatch Internet Monitor	`arn:aws:states:::aws-sdk:internetmonitor:` *[apiAction]*	2023년 6월 16일	InternetMonitor
Amazon Interactive Video Service RealTime	`arn:aws:states:::aws-sdk:ivsrealtime:` *[apiAction]*	2023년 6월 16일	IvsRealTime
AWS IoT TwinMaker	`arn:aws:states:::aws-sdk:iottwinmaker:` *[apiAction]*	2023년 6월 16일	IoTTwinMaker
Amazon OpenSearch Ingestion	`arn:aws:states:::aws-sdk:osis:` *[apiAction]*	2023년 6월 16일	Osis
AWS Telco Network Builder	`arn:aws:states:::aws-sdk:` **`tnb`**`:`*[apiAction]*	2023년 6월 16일	Tnb

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon VPC Lattice	`arn:aws:states:::aws-sdk:vpclattice:` *[apiAction]*	2023년 6월 16일	VpcLattice
Amazon Chime Media Pipelines	`arn:aws:states:::aws-sdk:chimesdkmediapipelines:` *[apiAction]*	2023년 2월 17일	ChimeSdkMediaPipelines
Amazon Chime Voice	`arn:aws:states:::aws-sdk:chimesdkvoice:` *[apiAction]*	2023년 2월 17일	ChimeSdkVoice
Amazon CodeCatalyst	`arn:aws:states:::aws-sdk:codecatalyst:` *[apiAction]*	2023년 2월 17일	CodeCatalyst
Amazon Connect Cases	`arn:aws:states:::aws-sdk:connectcases:` *[apiAction]*	2023년 2월 17일	ConnectCases
Amazon DocumentDB Elastic Clusters	`arn:aws:states:::aws-sdk:docdbelastic:` *[apiAction]*	2023년 2월 17일	DocDbElastic
Amazon EMR Serverless	`arn:aws:states:::aws-sdk:emrserverless:` *[apiAction]*	2023년 2월 17일	EmrServerless
Amazon IVS Chat	`arn:aws:states:::aws-sdk:ivs:` *[apiAction]*	2023년 2월 17일	Ivs
Amazon Kendra Intelligent Ranking	`arn:aws:states:::aws-sdk:kendraranking:` *[apiAction]*	2023년 2월 17일	KendraRanking

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS HealthOmics	`arn:aws:states:::aws-sdk:omics: [apiAction]`	2023년 2월 17일	Omics
Amazon Redshift Serverless	`arn:aws:states:::aws-sdk:redshiftserverless: [apiAction]`	2023년 2월 17일	RedshiftServerless
Amazon Security Lake	`arn:aws:states:::aws-sdk:securitylake: [apiAction]`	2023년 2월 17일	SecurityLake
AWS Backup Storage	`arn:aws:states:::aws-sdk:backupstorage: [apiAction]`	2023년 2월 17일	BackupStorage
AWS Clean Rooms	`arn:aws:states:::aws-sdk:cleanrooms: [apiAction]`	2023년 2월 17일	CleanRooms
AWS Control Tower	`arn:aws:states:::aws-sdk:controltower: [apiAction]`	2023년 2월 17일	ControlTower
AWS Health	`arn:aws:states:::aws-sdk:health: [apiAction]`	2023년 2월 17일	Health
AWS IoT FleetWise	`arn:aws:states:::aws-sdk:iotfleetwise: [apiAction]`	2023년 2월 17일	IotFleetWise
AWS Mainframe Modernization	`arn:aws:states:::aws-sdk:m2: [apiAction]`	2023년 2월 17일	M2

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Migration Hub Orchestrator	`arn:aws:states:::aws-sdk:migrationhuborchestrator: [apiAction]`	2023년 2월 17일	MigrationHubOrchestrator
AWS Private 5G	`arn:aws:states:::aws-sdk:privatenetworks: [apiAction]`	2023년 2월 17일	PrivateNetworks
AWS 리소스 탐색기	`arn:aws:states:::aws-sdk:resourceexplorer2: [apiAction]`	2023년 2월 17일	ResourceExplorer2
AWS SimSpace Weaver	`arn:aws:states:::aws-sdk:simspaceweaver: [apiAction]`	2023년 2월 17일	SimSpaceWeaver
AWS Support App	`arn:aws:states:::aws-sdk:supportapp: [apiAction]`	2023년 2월 17일	SupportApp
CloudWatch Observability Access Manager	`arn:aws:states:::aws-sdk:oam: [apiAction]`	2023년 2월 17일	Oam
EventBridge Pipes	`arn:aws:states:::aws-sdk:pipes: [apiAction]`	2023년 2월 17일	Pipes
EventBridge Scheduler	`arn:aws:states:::aws-sdk:scheduler: [apiAction]`	2023년 2월 17일	Scheduler
IAM Roles Anywhere	`arn:aws:states:::aws-sdk:rolesanywhere: [apiAction]`	2023년 2월 17일	RolesAnywhere

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Kinesis Video WebRTC Storage	`arn:aws:states:::aws-sdk:kinesisvideowebrtcstorage:` *`[apiAction]`*	2023년 2월 17일	KinesisVideoWebRtcStorage
License Manager Linux Subscriptions	`arn:aws:states:::aws-sdk:licensemanagerlinuxsubscriptions:` *`[apiAction]`*	2023년 2월 17일	LicenseManagerLinuxSubscriptions
License Manager User Subscriptions	`arn:aws:states:::aws-sdk:licensemanagerusersubscriptions:` *`[apiAction]`*	2023년 2월 17일	LicenseManagerUserSubscriptions
OpenSearch Serverless	`arn:aws:states:::aws-sdk:opensearchserverless:` *`[apiAction]`*	2023년 2월 17일	OpenSearchServerless
Route 53 ARC Zonal Shift	`arn:aws:states:::aws-sdk:arczonalshift:` *`[apiAction]`*	2023년 2월 17일	ArcZonalShift
SageMaker Geospatial	`arn:aws:states:::aws-sdk:sagemakergeospatial:` *`[apiAction]`*	2023년 2월 17일	SageMaker Geospatial
SageMaker Metrics	`arn:aws:states:::aws-sdk:sagemakermetrics:` *`[apiAction]`*	2023년 2월 17일	SageMakerMetrics
Systems Manager for SAP	`arn:aws:states:::aws-sdk:ssmsap:` *`[apiAction]`*	2023년 2월 17일	SsmSap

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Account Management	`arn:aws:states:::aws-sdk:account:` *[apiAction]*	2022년 4월 19일	Account
AWS Amplify	`arn:aws:states:::aws-sdk:amplify:` *[apiAction]*	2021년 9월 30일	Amplify
AWS App Mesh	`arn:aws:states:::aws-sdk:appmesh:` *[apiAction]*	2021년 9월 30일	AppMesh
AWS App Runner	`arn:aws:states:::aws-sdk:apprunner:` *[apiAction]*	2021년 9월 30일	AppRunner
AWS AppConfig	`arn:aws:states:::aws-sdk:appconfig:` *[apiAction]*	2021년 9월 30일	AppConfig
AWS AppConfig Data	`arn:aws:states:::aws-sdk:appconfigdata:` *[apiAction]*	2022년 4월 19일	AppConfigData
AWS AppSync	`arn:aws:states:::aws-sdk:appsync:` *[apiAction]*	2021년 9월 30일	AppSync
AWS Application Discovery Service	`arn:aws:states:::aws-sdk:applicationdiscovery:` *[apiAction]* ***	2021년 9월 30일	ApplicationDiscovery
AWS Application Migration Service	`arn:aws:states:::aws-sdk:mgn:` *[apiAction]*	2021년 9월 30일	Mgn

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Audit Manager	`arn:aws:states:::aws-sdk:auditmanager: [apiAction]`	2021년 9월 30일	AuditManager
AWS Auto Scaling Plans	`arn:aws:states:::aws-sdk:autoscalingplans: [apiAction]`	2021년 9월 30일	AutoScalingPlans
AWS Backup	`arn:aws:states:::aws-sdk:backup: [apiAction]`	2021년 9월 30일	Backup
AWS Backup gateway	`arn:aws:states:::aws-sdk:backupgateway: [apiAction]`	2022년 4월 19일	BackupGateway
AWS Batch	`arn:aws:states:::aws-sdk:batch: [apiAction]`	2021년 9월 30일	Batch
AWS Billing Conductor	`arn:aws:states:::aws-sdk:billingconductor: [apiAction]`	2022년 7월 26일	Billingconductor
AWS Budgets	`arn:aws:states:::aws-sdk:budgets: [apiAction]`	2021년 9월 30일	Budgets
AWS Certificate Manager	`arn:aws:states:::aws-sdk:acm: [apiAction]`	2021년 9월 30일	Acm
AWS Private Certificate Authority	`arn:aws:states:::aws-sdk:acmpca: [apiAction]`	2021년 9월 30일	AcmPca

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Cloud Map	`arn:aws:states:::aws-sdk:servicediscovery: [apiAction]`	2021년 9월 30일	ServiceDiscovery
AWS Cloud9	`arn:aws:states:::aws-sdk:cloud9: [apiAction]`	2021년 9월 30일	Cloud9
AWS CloudFormation	`arn:aws:states:::aws-sdk:cloudformation: [apiAction]`	2021년 9월 30일	CloudFormation
AWS CloudHSM	`arn:aws:states:::aws-sdk:cloudhsm: [apiAction]`	2021년 9월 30일	CloudHsm
AWS CloudHSM	`arn:aws:states:::aws-sdk:cloudhsmv2: [apiAction]`	2021년 9월 30일	CloudHsmV2
AWS CloudTrail	`arn:aws:states:::aws-sdk:cloudtrail: [apiAction]`	2021년 9월 30일	CloudTrail
AWS Cloud Control	`arn:aws:states:::aws-sdk:cloudcontrol: [apiAction]`	2022년 4월 19일	CloudControl
AWS CodeBuild	`arn:aws:states:::aws-sdk:codebuild: [apiAction]`	2021년 9월 30일	CodeBuild
AWS CodeCommit	`arn:aws:states:::aws-sdk:codecommit: [apiAction]`	2021년 9월 30일	CodeCommit

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS CodeDeploy	`arn:aws:states:::aws-sdk:codedeploy:` *`[apiAction]`* ***	2021년 9월 30일	CodeDeploy
AWS CodePipeline	`arn:aws:states:::aws-sdk:codepipeline:` *`[apiAction]`*	2021년 9월 30일	CodePipeline
AWS CodeStar	`arn:aws:states:::aws-sdk:codestar:` *`[apiAction]`*	2021년 9월 30일	CodeStar
AWS CodeStar	`arn:aws:states:::aws-sdk:codestarnotifications:` *`[apiAction]`*	2021년 9월 30일	CodestarNotifications
AWS CodeStar	`arn:aws:states:::aws-sdk:codestarconnections:` *`[apiAction]`*	2021년 9월 30일	CodeStarConnections
AWS Compute Optimizer	`arn:aws:states:::aws-sdk:computeoptimizer:` *`[apiAction]`*	2021년 9월 30일	ComputeOptimizer
AWS Config	`arn:aws:states:::aws-sdk:config:` *`[apiAction]`*	2021년 9월 30일	Config
AWS Cost Explorer Service	`arn:aws:states:::aws-sdk:costexplorer:` *`[apiAction]`*	2021년 9월 30일	CostExplorer

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Cost and Usage Report	`arn:aws:states:::aws-sdk:costandusagereport:` *[apiAction]*	2021년 9월 30일	CostAndUsageReport
AWS Data Exchange	`arn:aws:states:::aws-sdk:dataexchange:` *[apiAction]*	2021년 9월 30일	DataExchange
AWS Data Pipeline	`arn:aws:states:::aws-sdk:datapipeline:` *[apiAction]*	2021년 9월 30일	DataPipeline
AWS DataSync	`arn:aws:states:::aws-sdk:datasync:` *[apiAction]*	2021년 9월 30일	DataSync
AWS Database Migration Service	`arn:aws:states:::aws-sdk:databasemigration:` *[apiAction]*	2021년 9월 30일	DatabaseMigration
AWS Device Farm	`arn:aws:states:::aws-sdk:devicefarm:` *[apiAction]*	2021년 9월 30일	DeviceFarm
AWS Direct Connect	`arn:aws:states:::aws-sdk:directconnect:` *[apiAction]* *** —	2021년 9월 30일	DirectConnect
AWS Directory Service	`arn:aws:states:::aws-sdk:directory:` *[apiAction]*	2021년 9월 30일	Directory
AWS EC2 Instance Connect	`arn:aws:states:::aws-sdk:ec2instanceconnect:` *[apiAction]*	2021년 9월 30일	Ec2InstanceConnect

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Elastic Beanstalk	`arn:aws:states:::aws-sdk:elasticbeanstalk:` *[apiAction]*	2021년 9월 30일	ElasticBeanstalk
AWS Elemental MediaLive	`arn:aws:states:::aws-sdk:medialive:` *[apiAction]*	2021년 9월 30일	MediaLive
AWS Elemental MediaPackage	`arn:aws:states:::aws-sdk:mediapackage:` *[apiAction]* *** —	2021년 9월 30일	MediaPackage
AWS Elemental MediaPackage VOD	`arn:aws:states:::aws-sdk:mediapackagevod:` *[apiAction]*	2021년 9월 30일	MediaPackageVod
AWS Elemental MediaStore	`arn:aws:states:::aws-sdk:mediastore:` *[apiAction]*	2021년 9월 30일	MediaStore
AWS Fault Injection Service	`arn:aws:states:::aws-sdk:fis:` *[apiAction]*	2021년 9월 30일	Fis
AWS Firewall Manager	`arn:aws:states:::aws-sdk:fms:` *[apiAction]*	2021년 9월 30일	Fms
AWS Glue	`arn:aws:states:::aws-sdk:glue:` *[apiAction]*	2021년 9월 30일	Glue
AWS Glue DataBrew	`arn:aws:states:::aws-sdk:databrew:` *[apiAction]*	2021년 9월 30일	DataBrew

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS IoT Greengrass	`arn:aws:states:::aws-sdk:greengrass:` *`[apiAction]`*	2021년 9월 30일	Greengrass
AWS Ground Station	`arn:aws:states:::aws-sdk:groundstation:` *`[apiAction]`*	2021년 9월 30일	GroundStation
AWS Identity and Access Management	`arn:aws:states:::aws-sdk:iam:` *`[apiAction]`*	2021년 9월 30일	Iam
AWS IoT	`arn:aws:states:::aws-sdk:iot:` *`[apiAction]`* ***	2021년 9월 30일	Iot
AWS IoT 1-Click	`arn:aws:states:::aws-sdk:iot1clickprojects:` *`[apiAction]`*	2021년 9월 30일	Iot1ClickProjects
AWS IoT Analytics	`arn:aws:states:::aws-sdk:iotanalytics:` *`[apiAction]`*	2021년 9월 30일	IoTAnalytics
AWS IoT Core Device Advisor	`arn:aws:states:::aws-sdk:iotdeviceadvisor:` *`[apiAction]`* ***	2021년 9월 30일	IotDeviceAdvisor
AWS IoT Events	`arn:aws:states:::aws-sdk:iotevents:` *`[apiAction]`*	2021년 9월 30일	IotEvents
AWS IoT Events 데이터	`arn:aws:states:::aws-sdk:ioteventsdata:` *`[apiAction]`*	2021년 9월 30일	IotEventsData

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS IoT Fleet Hub	`arn:aws:states:::aws-sdk:iotfleethub:` *[apiAction]*	2021년 9월 30일	IoTFleetHub
AWS IoT Greengrass Version 2	`arn:aws:states:::aws-sdk:greengrassv2:` *[apiAction]*	2021년 9월 30일	GreengrassV2
AWS IoT 작업 데이터 Plane	`arn:aws:states:::aws-sdk:iotjobsdataplane:` *[apiAction]*	2021년 9월 30일	IotJobsDataPlane
AWS IoT Secure Tunneling	`arn:aws:states:::aws-sdk:iotsecuretunneling:` *[apiAction]*	2021년 9월 30일	IoTSecure Tunneling
AWS IoT SiteWise	`arn:aws:states:::aws-sdk:iotsitewise:` *[apiAction]*	2021년 9월 30일	IoTSiteWise
AWS IoT Wireless	`arn:aws:states:::aws-sdk:iotwireless:` *[apiAction]*	2021년 9월 30일	IotWireless
AWS Key Management Service	`arn:aws:states:::aws-sdk:kms:` *[apiAction]*	2021년 9월 30일	Kms
AWS Lake Formation	`arn:aws:states:::aws-sdk:lakeformation:` *[apiAction]*	2021년 9월 30일	LakeFormation
AWS Lambda	`arn:aws:states:::aws-sdk:lambda:` *[apiAction]* ***	2021년 9월 30일	Lambda

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS License Manager	`arn:aws:states:::aws-sdk:licensemanager: [apiAction]`	2021년 9월 30일	LicenseManager
AWS Marketplace	`arn:aws:states:::aws-sdk:marketplacecatalog: [apiAction]`	2021년 9월 30일	MarketplaceCatalog
AWS Marketplace Commerce Analytics	`arn:aws:states:::aws-sdk:marketplacecommerceanalytics: [apiAction]`	2021년 9월 30일	MarketplaceCommerceAnalytics
AWS Marketplace Entitlement Service	`arn:aws:states:::aws-sdk:marketplaceentitlement: [apiAction]`	2021년 9월 30일	MarketplaceEntitlement
AWS Elemental MediaTailor	`arn:aws:states:::aws-sdk:mediatailor: [apiAction]`	2021년 9월 30일	MediaTailor
AWS Migration Hub	`arn:aws:states:::aws-sdk:migrationhub: [apiAction]`	2021년 9월 30일	MigrationHub
AWS Migration Hub Config	`arn:aws:states:::aws-sdk:migrationhubconfig: [apiAction]`	2021년 9월 30일	MigrationHubConfig
AWS Migration Hub 전략 권장 사항	`arn:aws:states:::aws-sdk:migrationhubstrategy: [apiAction]`	2022년 4월 19일	MigrationHubStrategy

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Mobile	`arn:aws:states:::aws-sdk:mobile:` *[apiAction]*	2021년 9월 30일	
AWS Network Firewall	`arn:aws:states:::aws-sdk:networkfirewall:` *[apiAction]*	2021년 9월 30일	NetworkFirewall
AWS OpsWorks	`arn:aws:states:::aws-sdk:opsworks:` *[apiAction]*	2021년 9월 30일	OpsWorks
AWS OpsWorks CM	`arn:aws:states:::aws-sdk:opsworkscm:` *[apiAction]*	2021년 9월 30일	OpsWorksCm
AWS Organizations	`arn:aws:states:::aws-sdk:organizations:` *[apiAction]*	2021년 9월 30일	Organizations
AWS Outposts	`arn:aws:states:::aws-sdk:outposts:` *[apiAction]*	2021년 9월 30일	Outposts
AWS Panorama	`arn:aws:states:::aws-sdk:panorama:` *[apiAction]*	2022년 4월 19일	Panorama
Amazon Relational Database Service Performance Insights	`arn:aws:states:::aws-sdk:pi:` *[apiAction]*	2021년 9월 30일	Pi
AWS 가격표	`arn:aws:states:::aws-sdk:pricing:` *[apiAction]*	2021년 9월 30일	Pricing

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Relational Database Service	`arn:aws:states:::aws-sdk:rdsdata:` *[apiAction]* ***	2021년 9월 30일	RdsData
AWS Resilience Hub	`arn:aws:states:::aws-sdk:resiliencehub:` *[apiAction]*	2022년 4월 19일	Resiliencehub
AWS Resource Access Manager	`arn:aws:states:::aws-sdk:ram:` *[apiAction]*	2021년 9월 30일	Ram
AWS Resource Groups	`arn:aws:states:::aws-sdk:resourcegroups:` *[apiAction]*	2021년 9월 30일	ResourceGroups
AWS Resource Groups Tagging API	`arn:aws:states:::aws-sdk:resourcegroupstaggingapi:` *[apiAction]*	2021년 9월 30일	ResourceGroupsTaggingApi
AWS RoboMaker	`arn:aws:states:::aws-sdk:robomaker:` *[apiAction]*	2021년 9월 30일	RoboMaker
AWS IAM Identity Center	`arn:aws:states:::aws-sdk:identitystore:` *[apiAction]*	2021년 9월 30일	Identitystore
IAM Identity Center OIDC	`arn:aws:states:::aws-sdk:ssooidc:` *[apiAction]*	2021년 9월 30일	SsoOidc
AWS Secrets Manager	`arn:aws:states:::aws-sdk:secretsmanager:` *[apiAction]*	2021년 9월 30일	SecretsManager

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Security Token Service	`arn:aws:states:::aws-sdk:sts:` *`[apiAction]`* ***	2021년 9월 30일	Sts
AWS Security Hub	`arn:aws:states:::aws-sdk:securityhub:` *`[apiAction]`*	2021년 9월 30일	SecurityHub
AWS Server Migration Service	`arn:aws:states:::aws-sdk:sms:` *`[apiAction]`*	2021년 9월 30일	Sms
AWS Service Catalog	`arn:aws:states:::aws-sdk:servicecatalog:` *`[apiAction]`*	2021년 9월 30일	ServiceCatalog
AWS Service Catalog AppRegistry	`arn:aws:states:::aws-sdk:servicecatalogappregistry:` *`[apiAction]`*	2021년 9월 30일	ServiceCatalogAppRegistry
AWS Shield	`arn:aws:states:::aws-sdk:shield:` *`[apiAction]`* ***	2021년 9월 30일	Shield
AWS Signer	`arn:aws:states:::aws-sdk:signer:` *`[apiAction]`*	2021년 9월 30일	Signer
IAM Identity Center	`arn:aws:states:::aws-sdk:sso:` *`[apiAction]`*	2021년 9월 30일	Sso
IAM Identity Center Admin	`arn:aws:states:::aws-sdk:ssoadmin:` *`[apiAction]`*	2021년 9월 30일	SsoAdmin

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS Step Functions	`arn:aws:states:::aws-sdk:sfn:` *[apiAction]*	2021년 9월 30일	Sfn
AWS Storage Gateway	`arn:aws:states:::aws-sdk:storagegateway:` *[apiAction]*	2021년 9월 30일	StorageGateway
AWS Support	`arn:aws:states:::aws-sdk:support:` *[apiAction]*	2021년 9월 30일	Support
AWS Systems Manager Incident Manager	`arn:aws:states:::aws-sdk:ssmincidents:` *[apiAction]*		SsmIncidents
AWS Transfer Family	`arn:aws:states:::aws-sdk:transfer:` *[apiAction]*	2021년 9월 30일	Transfer
AWS WAF	`arn:aws:states:::aws-sdk:waf:` *[apiAction]*	2021년 9월 30일	Waf
AWS WAF Regional	`arn:aws:states:::aws-sdk:wafregional:` *[apiAction]*	2021년 9월 30일	WafRegional
AWS WAFV2	`arn:aws:states:::aws-sdk:wafv2:` *[apiAction]*	2021년 9월 30일	Wafv2
AWS Well-Architected Tool	`arn:aws:states:::aws-sdk:wellarchitected:` *[apiAction]*	2021년 9월 30일	WellArchitected

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
AWS X-Ray	`arn:aws:states:::aws-sdk:xray: [apiAction]`	2021년 9월 30일	XRay
AWS Marketplace Metering Service	`arn:aws:states:::aws-sdk:marketplacemetering: [apiAction]`	2021년 9월 30일	MarketplaceMetering
AWS Serverless Application Repository	`arn:aws:states:::aws-sdk:serverlessapplicationrepository: [apiAction]`	2021년 9월 30일	ServerlessApplicationRepository
AWS Identity and Access Management Access Analyzer	`arn:aws:states:::aws-sdk:accessanalyzer: [apiAction]`	2021년 9월 30일	AccessAnalyzer
Alexa for Business	`arn:aws:states:::aws-sdk:alexaforbusiness: [apiAction]`	2021년 9월 30일	AlexaForBusiness
Amazon API Gateway	`arn:aws:states:::aws-sdk:apigateway: [apiAction]`	2021년 9월 30일	ApiGateway
Amazon API Gateway	`arn:aws:states:::aws-sdk:apigatewayv2: [apiAction]`	2021년 9월 30일	ApiGatewayV2
Amazon AppIntegrations	`arn:aws:states:::aws-sdk:appintegrations: [apiAction]`	2021년 9월 30일	AppIntegrations

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon AppStream 2.0	`arn:aws:states:::aws-sdk:appstream:` *[apiAction]*	2021년 9월 30일	AppStream
Amazon AppFlow	`arn:aws:states:::aws-sdk:appflow:` *[apiAction]*	2021년 9월 30일	Appflow
Amazon Athena	`arn:aws:states:::aws-sdk:athena:` *[apiAction]*	2021년 9월 30일	Athena
Amazon Augmented AI	`arn:aws:states:::aws-sdk:sagemakera2iruntime:` *[apiAction]*	2021년 9월 30일	SageMaker A2IRuntime
Amazon Braket	`arn:aws:states:::aws-sdk:braket:` *[apiAction]*	2021년 9월 30일	Braket
Amazon Chime	`arn:aws:states:::aws-sdk:chime:` *[apiAction]*	2021년 9월 30일	Chime
Amazon Chime Meetings	`arn:aws:states:::aws-sdk:chimesdkmeetings:` *[apiAction]*	2022년 4월 19일	ChimeSdkMeetings
Amazon Cloud Directory	`arn:aws:states:::aws-sdk:clouddirectory:` *[apiAction]*	2021년 9월 30일	CloudDirectory
Amazon CloudFront	`arn:aws:states:::aws-sdk:cloudfront:` *[apiAction]*	2021년 9월 30일	CloudFront

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon CloudSearch	`arn:aws:states:::aws-sdk:cloudsearch: [apiAction]`	2021년 9월 30일	CloudSearch
Amazon CloudWatch	`arn:aws:states:::aws-sdk:cloudwatch: [apiAction]`	2021년 9월 30일	CloudWatch
Amazon CloudWatch Application Insights	`arn:aws:states:::aws-sdk:applicationinsights: [apiAction]`	2021년 9월 30일	ApplicationInsights
CloudWatch Evidently	`arn:aws:states:::aws-sdk:evidently: [apiAction]`	2022년 4월 19일	Evidently
Amazon CloudWatch Logs	`arn:aws:states:::aws-sdk:cloudwatchlogs: [apiAction]`	2021년 9월 30일	CloudWatchLogs
Amazon CloudWatch RUM	`arn:aws:states:::aws-sdk:rum: [apiAction]`	2022년 4월 19일	Rum
Amazon CloudWatch Synthetics	`arn:aws:states:::aws-sdk:synthetics: [apiAction]`	2021년 9월 30일	Synthetics
Amazon CodeGuru Profiler	`arn:aws:states:::aws-sdk:codeguruprofiler: [apiAction]`	2021년 9월 30일	CodeGuruProfiler
Amazon CodeGuru Reviewer	`arn:aws:states:::aws-sdk:codegurureviewer: [apiAction]`	2021년 9월 30일	CodeGuruReviewer

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Cognito	`arn:aws:states:::aws-sdk:cognitoidentity: [apiAction]`	2021년 9월 30일	CognitoIdentity
Amazon Cognito Identity Provider	`arn:aws:states:::aws-sdk:cognitoidentityprovider: [apiAction]`	2021년 9월 30일	CognitoIdentityProvider
Amazon Cognito Sync	`arn:aws:states:::aws-sdk:cognitosync: [apiAction]`	2021년 9월 30일	CognitoSync
Amazon Comprehend	`arn:aws:states:::aws-sdk:comprehend: [apiAction]`	2021년 9월 30일	Comprehend
Amazon Comprehend Medical	`arn:aws:states:::aws-sdk:comprehendmedical: [apiAction]` ***	2021년 9월 30일	ComprehendMedical
Amazon Connect Contact Lens	`arn:aws:states:::aws-sdk:connectcontactlens: [apiAction]`	2021년 9월 30일	ConnectContactLens
Amazon Connect Participant Service	`arn:aws:states:::aws-sdk:connectparticipant: [apiAction]`	2021년 9월 30일	ConnectParticipant
Amazon Connect	`arn:aws:states:::aws-sdk:connect: [apiAction]`	2021년 9월 30일	Connect
Amazon Connect Voice ID	`arn:aws:states:::aws-sdk:voiceid: [apiAction]`	2022년 4월 19일	VoiceId

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Connect Wisdom	`arn:aws:states:::aws-sdk:wisdom:` *[apiAction]*	2022년 4월 19일	Wisdom
Amazon Data Lifecycle Manager	`arn:aws:states:::aws-sdk:dlm:` *[apiAction]*	2021년 9월 30일	Dlm
Amazon Detective	`arn:aws:states:::aws-sdk:detective:` *[apiAction]*	2021년 9월 30일	Detective
Amazon DevOps Guru	`arn:aws:states:::aws-sdk:devopsguru:` *[apiAction]*	2021년 9월 30일	DevOpsGuru
Amazon DocumentDB (with MongoDB compatibility)	`arn:aws:states:::aws-sdk:docdb:` *[apiAction]*	2021년 9월 30일	DocDb
Amazon DynamoDB	`arn:aws:states:::aws-sdk:dynamodb:` *[apiAction]*	2021년 9월 30일	DynamoDb
Amazon DynamoDB Streams	`arn:aws:states:::aws-sdk:dynamodbstreams:` *[apiAction]*	2021년 9월 30일	DynamoDbStreams
Amazon EC2 Container Registry	`arn:aws:states:::aws-sdk:ecr:` *[apiAction]*	2021년 9월 30일	Ecr
Amazon EC2 Container Service	`arn:aws:states:::aws-sdk:ecs:` *[apiAction]*	2021년 9월 30일	Ecs

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon EC2 Systems Manager	`arn:aws:states:::aws-sdk:ssm:` *[apiAction]*	2021년 9월 30일	Ssm
Amazon EMR	`arn:aws:states:::aws-sdk:emrcontainers:` *[apiAction]* ***	2021년 9월 30일	EmrContainers
Amazon ElastiCache	`arn:aws:states:::aws-sdk:elasticache:` *[apiAction]*	2021년 9월 30일	ElastiCache
Amazon Elastic Inference	`arn:aws:states:::aws-sdk:elasticinference:` *[apiAction]*	2021년 9월 30일	ElasticInference
Amazon Elastic Block Store	`arn:aws:states:::aws-sdk:ebs:` *[apiAction]*	2021년 9월 30일	Ebs
Amazon Elastic Compute Cloud	`arn:aws:states:::aws-sdk:ec2:` *[apiAction]*	2021년 9월 30일	Ec2
Amazon Elastic Container Registry Public	`arn:aws:states:::aws-sdk:ecrpublic:` *[apiAction]*	2021년 9월 30일	EcrPublic
Amazon Elastic File System	`arn:aws:states:::aws-sdk:efs:` *[apiAction]* ***	2021년 9월 30일	Efs
Amazon Elastic Kubernetes Service	`arn:aws:states:::aws-sdk:eks:` *[apiAction]*	2021년 9월 30일	Eks

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon EMR	`arn:aws:states:::aws-sdk:emr:` *[apiAction]*	2021년 9월 30일	Emr
Amazon Elastic Transcoder	`arn:aws:states:::aws-sdk:elastictranscoder:` *[apiAction]* ***	2021년 9월 30일	ElasticTranscoder
Amazon OpenSearch Service	`arn:aws:states:::aws-sdk:elasticsearch:` *[apiAction]*	2021년 9월 30일	Elasticsearch
Amazon OpenSearch Service	`arn:aws:states:::aws-sdk:opensearch:` *[apiAction]*	2022년 4월 19일	OpenSearch
Amazon EventBridge	`arn:aws:states:::aws-sdk:eventbridge:` *[apiAction]*	2021년 9월 30일	EventBridge
Amazon FSx	`arn:aws:states:::aws-sdk:fsx:` *[apiAction]*	2021년 9월 30일	FSx
Amazon Forecast Query	`arn:aws:states:::aws-sdk:forecastquery:` *[apiAction]*	2021년 9월 30일	Forecastquery
Amazon Forecast Service	`arn:aws:states:::aws-sdk:forecast:` *[apiAction]*	2021년 9월 30일	Forecast
Amazon Fraud Detector	`arn:aws:states:::aws-sdk:frauddetector:` *[apiAction]*	2021년 9월 30일	FraudDetector

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon GameLift	`arn:aws:states:::aws-sdk:gamelift:` *[apiAction]*	2021년 9월 30일	Amazon GameLift
Amazon GameSparks	`arn:aws:states:::aws-sdk:gamesparks:` *[apiAction]*	2022년 7월 27일	GameSparks
Amazon S3 Glacier	`arn:aws:states:::aws-sdk:glacier:` *[apiAction]*	2021년 9월 30일	Glacier
Amazon GuardDuty	`arn:aws:states:::aws-sdk:guardduty:` *[apiAction]*	2021년 9월 30일	GuardDuty
AWS HealthLake	`arn:aws:states:::aws-sdk:healthlake:` *[apiAction]*	2021년 9월 30일	HealthLake
Amazon Honeycode	`arn:aws:states:::aws-sdk:honeycode:` *[apiAction]*	2021년 9월 30일	Honeycode
Amazon Inspector	`arn:aws:states:::aws-sdk:inspector:` *[apiAction]*	2021년 9월 30일	Inspector
Amazon Inspector V2	`arn:aws:states:::aws-sdk:inspector2:` *[apiAction]*	2022년 4월 19일	Inspector2
Amazon Interactive Video Service	`arn:aws:states:::aws-sdk:ivs:` *[apiAction]*	2021년 9월 30일	Ivs

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Kendra	`arn:aws:states:::aws-sdk:kendra:` *`[apiAction]`*	2021년 9월 30일	Kendra
Amazon Kinesis	`arn:aws:states:::aws-sdk:kinesis:` *`[apiAction]`* ***	2021년 9월 30일	Kinesis
Amazon Kinesis Analytics	`arn:aws:states:::aws-sdk:kinesisanalytics:` *`[apiAction]`*	2021년 9월 30일	KinesisAnalytics
Amazon Kinesis Analytics V2	`arn:aws:states:::aws-sdk:kinesisanalytics2:` *`[apiAction]`*	2021년 9월 30일	KinesisAnalyticsV2
Amazon Kinesis Firehose	`arn:aws:states:::aws-sdk:firehose:` *`[apiAction]`*	2021년 9월 30일	Firehose
Amazon Kinesis Video Signaling Channels	`arn:aws:states:::aws-sdk:kinesisvideosignaling:` *`[apiAction]`*	2021년 9월 30일	KinesisVideoSignaling
Amazon Kinesis Video Streams	`arn:aws:states:::aws-sdk:kinesisvideo:` *`[apiAction]`*	2021년 9월 30일	KinesisVideo
Amazon Kinesis Video Streams Archived Media	`arn:aws:states:::aws-sdk:kinesisvideoarchivedmedia:` *`[apiAction]`*	2021년 9월 30일	KinesisVideoArchivedMedia

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Kinesis video stream	`arn:aws:states:::aws-sdk:kinesisvideomedia:` *[apiAction]*	2021년 9월 30일	KinesisVideoMedia
Amazon Lex Model Building Service	`arn:aws:states:::aws-sdk:lexmodelbuilding:` *[apiAction]*	2021년 9월 30일	LexModelBuilding
Amazon Lex Model Building Service V2	`arn:aws:states:::aws-sdk:lexmodelsv2:` *[apiAction]*	2021년 9월 30일	LexModelsV2
Amazon Lex	`arn:aws:states:::aws-sdk:lexruntime:` *[apiAction]*	2021년 9월 30일	LexRuntime
Amazon Lex Runtime V2	`arn:aws:states:::aws-sdk:lexruntimev2:` *[apiAction]* ***	2021년 9월 30일	LexRuntimeV2
Amazon Lightsail	`arn:aws:states:::aws-sdk:lightsail:` *[apiAction]*	2021년 9월 30일	Lightsail
Amazon Location Service	`arn:aws:states:::aws-sdk:location:` *[apiAction]*	2021년 9월 30일	Location
Amazon Lookout for Equipment	`arn:aws::states:::aws-sdk:lookoutequipment:` *[apiAction]*	2021년 9월 30일	LookoutEquipment
Amazon Lookout for Metrics	`arn:aws:states:::aws-sdk:lookoutmetrics:` *[apiAction]*	2021년 9월 30일	LookoutMetrics

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Lookout for Vision	`arn:aws:states:::aws-sdk:lookoutvision: [apiAction]`	2021년 9월 30일	LookoutVision
Amazon MQ	`arn:aws:states:::aws-sdk:mq: [apiAction]`	2021년 9월 30일	Mq
Amazon Macie	`arn:aws:states:::aws-sdk:macie: [apiAction]`	2021년 9월 30일	
Amazon Macie 2	`arn:aws:states:::aws-sdk:macie2: [apiAction]`	2021년 9월 30일	Macie2
Amazon Managed Blockchain	`arn:aws:states:::aws-sdk:managedblockchain: [apiAction]`	2021년 9월 30일	ManagedBlockchain
Amazon Managed Grafana	`arn:aws:states:::aws-sdk:grafana: [apiAction]`	2022년 4월 19일	Grafana
Amazon Managed Service for Prometheus	`arn:aws:states:::aws-sdk:amp: [apiAction]`	2021년 9월 30일	Amp
Amazon Managed Streaming for Apache Kafka	`arn:aws:states:::aws-sdk:kafka: [apiAction]`	2021년 9월 30일	Kafka
Amazon MSK Connect	`arn:aws:states:::aws-sdk:kafkaconnect: [apiAction]`	2022년 4월 19일	KafkaConnect

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Managed Workflows for Apache Airflow	`arn:aws:states:::aws-sdk:mwaa: [apiAction]`	2021년 9월 30일	Mwaa
Amazon Mechanical Turk	`arn:aws:states:::aws-sdk:mturk: [apiAction]`	2021년 9월 30일	MTurk
Amazon MemoryDB for Redis	`arn:aws:states:::aws-sdk:memorydb: [apiAction]`	2022년 4월 19일	MemoryDB
Amazon Nimble Studio	`arn:aws:states:::aws-sdk:nimble: [apiAction]`	2021년 9월 30일	Nimble
Amazon Personalize	`arn:aws:states:::aws-sdk:personalize: [apiAction]`	2021년 9월 30일	Personalize
Amazon Personalize Events	`arn:aws:states:::aws-sdk:personalizeevents: [apiAction]`	2021년 9월 30일	PersonalizeEvents
Amazon Personalize Runtime	`arn:aws:states:::aws-sdk:personalizeruntime: [apiAction]`	2021년 9월 30일	PersonalizeRuntime
Amazon Pinpoint	`arn:aws:states:::aws-sdk:pinpoint: [apiAction]`	2021년 9월 30일	Pinpoint
Amazon Pinpoint Email Service	`arn:aws:states:::aws-sdk:pinpointemail: [apiAction]`	2021년 9월 30일	PinpointEmail

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Pinpoint SMS and Voice Service	`arn:aws:states:::aws-sdk:pinpointsmsvoice: [apiAction]`	2021년 9월 30일	PinpointSmsVoice
Amazon Pinpoint SMS and Voice V2 Service	`arn:aws:states:::aws-sdk:pinpointsmsvoicev2: [apiAction]`	2022년 7월 27일	PinpointSmsVoiceV2
Amazon Polly	`arn:aws:states:::aws-sdk:polly: [apiAction]`	2021년 9월 30일	Polly
Amazon QLDB	`arn:aws:states:::aws-sdk:qldb: [apiAction]`	2021년 9월 30일	Qldb
Amazon QLDB Session	`arn:aws:states:::aws-sdk:qldbsession: [apiAction]`	2021년 9월 30일	QldbSession
Amazon QuickSight	`arn:aws:states:::aws-sdk:quicksight: [apiAction]`	2021년 9월 30일	QuickSight
Amazon Redshift	`arn:aws:states:::aws-sdk:redshift: [apiAction]`	2021년 9월 30일	Redshift
Amazon Redshift Data API	`arn:aws:states:::aws-sdk:redshiftdata: [apiAction]`	2021년 9월 30일	RedshiftData
Amazon Rekognition	`arn:aws:states:::aws-sdk:rekognition: [apiAction]`	2021년 9월 30일	Rekognition

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Relational Database Service	`arn:aws:states:::aws-sdk:rds:` *`[apiAction]`*	2021년 9월 30일	Rds
Amazon Route 53	`arn:aws:states:::aws-sdk:route53:` *`[apiAction]`*	2021년 9월 30일	Route53
Amazon Route 53 Recovery Control Config	`arn:aws:states:::aws-sdk:route53recoverycontrolconfig:` *`[apiAction]`*	2021년 9월 30일	Route53RecoveryControlConfig
Amazon Route 53 Domains	`arn:aws:states:::aws-sdk:route53domains:` *`[apiAction]`*	2021년 9월 30일	Route53Domains
Amazon Route 53 Resolver	`arn:aws:states:::aws-sdk:route53resolver:` *`[apiAction]`*	2021년 9월 30일	Route53Resolver
Amazon S3 on Outposts	`arn:aws:states:::aws-sdk:s3outposts:` *`[apiAction]`*	2021년 9월 30일	S3Outposts
Amazon SageMaker Runtime Feature Store Runtime	`arn:aws:states:::aws-sdk:sagemakerfeaturestoreruntime:` *`[apiAction]`*	2021년 9월 30일	SageMakerRuntimeFeatureStoreRuntime
Amazon SageMaker Runtime Runtime	`arn:aws:states:::aws-sdk:sagemakerruntime:` *`[apiAction]`*	2021년 9월 30일	SageMakerRuntimeRuntime

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon SageMaker	`arn:aws:states:::aws-sdk:sagemaker:` *[apiAction]*	2021년 9월 30일	SageMaker Runtime
Amazon SageMaker Edge Manager	`arn:aws:states:::aws-sdk:sagemakeredge:` *[apiAction]*	2021년 9월 30일	SagemakerEdge
Amazon Simple Email Service	`arn:aws:states:::aws-sdk:ses:` *[apiAction]*	2021년 9월 30일	Ses
Amazon Simple Email Service V2	`arn:aws:states:::aws-sdk:sesv2:` *[apiAction]*	2021년 9월 30일	SesV2
Amazon Simple Notification Service	`arn:aws:states:::aws-sdk:sns:` *[apiAction]*	2021년 9월 30일	Sns
Amazon Simple Queue Service	`arn:aws:states:::aws-sdk:sqs:` *[apiAction]*	2021년 9월 30일	Sqs
Amazon Simple Storage Service	`arn:aws:states:::aws-sdk:s3:` *[apiAction]* *** —	2021년 9월 30일	S3
Amazon Simple Workflow Service	`arn:aws:states:::aws-sdk:swf:` *[apiAction]*	2021년 9월 30일	Swf
Amazon Textract	`arn:aws:states:::aws-sdk:textract:` *[apiAction]*	2021년 9월 30일	Textract

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Amazon Transcribe	`arn:aws:states:::aws-sdk:transcribe:` *`[apiAction]`*	2021년 9월 30일	Transcribe
Amazon Translate	`arn:aws:states:::aws-sdk:translate:` *`[apiAction]`*	2021년 9월 30일	Translate
Amazon WorkDocs	`arn:aws:states:::aws-sdk:workdocs:` *`[apiAction]`*	2021년 9월 30일	WorkDocs
Amazon WorkMail	`arn:aws:states:::aws-sdk:workmail:` *`[apiAction]`*	2021년 9월 30일	WorkMail
Amazon WorkMail Message Flow	`arn:aws:states:::aws-sdk:workmailmessageflow:` *`[apiAction]`*	2021년 9월 30일	WorkMailMessageFlow
Amazon WorkSpaces	`arn:aws:states:::aws-sdk:workspaces:` *`[apiAction]`*	2021년 9월 30일	WorkSpaces
Amazon WorkSpaces Web	`arn:aws:states:::aws-sdk:workspacesweb:` *`[apiAction]`*	2022년 4월 19일	WorkSpacesWeb
Amplify	`arn:aws:states:::aws-sdk:amplifybackend:` *`[apiAction]`*	2021년 9월 30일	AmplifyBackend
Amplify UI Builder	`arn:aws:states:::aws-sdk:amplifyuibuilder:` *`[apiAction]`*	2022년 4월 19일	AmplifyUiBuilder

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
Application Auto Scaling	`arn:aws:states:::aws-sdk:applicationautoscaling: [apiAction]`	2021년 9월 30일	ApplicationAutoScaling
Amazon EC2 Auto Scaling	`arn:aws:states:::aws-sdk:autoscaling: [apiAction]`	2021년 9월 30일	Auto Scaling
CodeArtifact	`arn:aws:states:::aws-sdk:codeartifact: [apiAction]`	2021년 9월 30일	Codeartifact
DynamoDB Accelerator	`arn:aws:states:::aws-sdk:dax: [apiAction]`	2021년 9월 30일	Dax
EC2 Image Builder	`arn:aws:states:::aws-sdk:imagebuilder: [apiAction]`	2021년 9월 30일	Imagebuilder
AWS Elastic Disaster Recovery	`arn:aws:states:::aws-sdk:drs: [apiAction]`	2022년 4월 19일	Drs
Elastic Load Balancing	`arn:aws:states:::aws-sdk:elasticloadbalancing: [apiAction]`	2021년 9월 30일	ElasticLoadBalancing
Elastic Load Balancing V2	`arn:aws:states:::aws-sdk:elasticloadbalancingv2: [apiAction]`	2021년 9월 30일	ElasticLoadBalancingV2

서비스 이름	**Task** 상태 리소스	지원 날짜	예외 접두사
MediaConnect	`arn:aws:states:::aws-sdk:mediaconnect:` *[apiAction]*	2021년 9월 30일	MediaConnect
Amazon S3 Control	`arn:aws:states:::aws-sdk:s3control:` *[apiAction]* ***	2021년 9월 30일	S3Control
Recycle Bin for Amazon EBS	`arn:aws:states:::aws-sdk:rbin:` *[apiAction]*	2022년 4월 19일	Rbin
Savings Plans	`arn:aws:states:::aws-sdk:savingsplans:` *[apiAction]*	2021년 9월 30일	Savingsplans
Amazon EventBridge Schema Registry	`arn:aws:states:::aws-sdk:schemas:` *[apiAction]*	2021년 9월 30일	Schemas
Service Quotas	`arn:aws:states:::aws-sdk:servicequotas:` *[apiAction]*	2021년 9월 30일	ServiceQuotas
AWS Snowball	`arn:aws:states:::aws-sdk:snowball:` *[apiAction]*	2021년 9월 30일	Snowball

지원되는 서비스에 지원되지 않는 API 작업

다음 표에는 AWS SDK 서비스 통합에 지원되지 않는 API 작업이 나열되어 있습니다. 오른쪽 열에는 왼쪽 열에 나열된 서비스에 현재 지원되지 않는 API 작업이 포함되어 있습니다.

서비스 이름	지원되지 않는 API 작업
AWS Application Discovery Service	• DescribeExportConfigurations • ExportConfigurations
Amazon Bedrock	• InvokeModelWithResponseStream
Amazon 베드락 런타임용 에이전트	• InvokeAgent
AWS CodeDeploy	• BatchGetDeploymentInstances • GetDeploymentInstance • ListDeploymentInstances • SkipWaitTimeForInstanceTermination
Amazon Comprehend Medical	• DetectEntities
AWS Direct Connect	• AllocateConnectionOnInterconnect • DescribeConnectionLoa • DescribeConnectionsOnInterconnect • DescribeInterconnectLoa
Amazon Elastic File System	• CreateTags
Amazon Elastic Transcoder	• TestRole
Amazon EMR	• DescribeJobFlows
AWS IoT	• AttachPrincipalPolicy • ListPrincipalPolicies • DetachPrincipalPolicy • ListPolicyPrincipals • DetachPrincipalPolicy

서비스 이름	지원되지 않는 API 작업
AWS IoT 코어 디바이스 어드바이저	• `ListTestCases`
Amazon Kinesis	• `SubscribeToShard`
AWS Lambda	• `InvokeAsync` • `InvokeWithResponseStream`
Amazon Lex Runtime V2	• `StartConversation`
AWS Elemental MediaPackage	• `RotateChannelCredentials`
Amazon Relational Database Service	• `ExecuteSql`
Amazon Simple Storage Service(S3)	• `SelectObjectContent`
Amazon S3 콘솔	• `SelectObjectContent`
AWS Shield	• `DeleteSubscription`
AWS Security Token Service	• `AssumeRole` • `AssumeRoleWithSAML` • `AssumeRoleWithWebIdentity`

더 이상 사용되지 않는 AWS SDK 서비스 통합

다음 AWS SDK 서비스 통합은 이제 더 이상 사용되지 않습니다.

- AWS 모바일
- Amazon Macie
- AWS IoT RoboRunner

Step Functions를 위한 최적화된 통합

다음 주제에는 다른 서비스를 조정하기 위한 Amazon States 언어의 지원되는 API, 파라미터 및 요청/응답 구문이 포함됩니다. AWS 이 주제에서는 예제 코드도 제공합니다. `Task` 상태의 `Resource` 필드에 있는 Amazon States Language에서 최적화된 통합 서비스를 직접적으로 호출할 수 있습니다.

세 가지의 서비스 통합 패턴을 사용할 수 있습니다.

- 응답 요청 (기본값) - HTTP 응답을 기다린 후 다음 상태로 이동
- Job 실행 (`.sync`) - 작업이 완료될 때까지 기다립니다.
- Callback 대기 (`.waitForTaskToken`) - 작업 토큰이 반환될 때까지 워크플로를 일시 중지합니다.

표준 워크플로와 익스프레스 워크플로는 동일한 통합을 지원하지만 동일한 통합 패턴은 지원하지 않습니다.

- 최적화된 통합 패턴 지원은 통합마다 다릅니다.
- 익스프레스 워크플로는 작업 실행 (.sync) 또는 콜백 대기 (. waitForTask토큰).
- 자세한 정보는 표준 워크플로와 Express 워크플로 비교을 참조하세요.

Standard Workflows

지원되는 서비스 통합

	Service	요청 및 응답	작업 실행 (`.sync`)	콜백 대기 (`.waitForTaskToken`)
최적화된 통합	Amazon API Gateway	✓		✓
	Amazon Athena	✓	✓	
	AWS Batch	✓	✓	
	Amazon Bedrock	✓	✓	✓
	AWS CodeBuild	✓	✓	

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓	✓	✓
	Amazon EKS	✓	✓	✓
	Amazon EMR	✓	✓	
	Amazon EMR on EKS	✓	✓	
	Amazon EMR Serverless	✓	✓	
	Amazon EventBridge	✓		✓
	AWS Glue	✓	✓	
	AWS Glue DataBrew	✓	✓	
	AWS Lambda	✓		✓
	AWS Elemental MediaConvert	✓	✓	
	Amazon SageMaker	✓	✓	
	Amazon SNS	✓		✓
	Amazon SQS	✓		✓
	AWS Step Functions	✓	✓	✓
AWS SDK 통합	200개 초과	✓		✓

Express Workflows

지원되는 서비스 통합

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
최적화된 통합	Amazon API Gateway	✓		
	Amazon Athena	✓		
	AWS Batch	✓		
	Amazon Bedrock	✓		
	AWS CodeBuild	✓		
	Amazon DynamoDB	✓		
	Amazon ECS/Fargate	✓		
	Amazon EKS	✓		
	Amazon EMR	✓		
	Amazon EMR on EKS	✓		
	Amazon EMR Serverless	✓		
	Amazon EventBridge	✓		
	AWS Glue	✓		
	AWS Glue DataBrew	✓		
	AWS Lambda	✓		
	AWS Elemental MediaConvert	✓		
	Amazon SageMaker	✓		

	Service	요청 및 응답	작업 실행 (.sync)	콜백 대기 (.waitForTaskToken)
	Amazon SNS	✓		
	Amazon SQS	✓		
	AWS Step Functions	✓		
AWS SDK 통합	200개 초과	✓		

Step Functions를 사용하여 API Gateway 직접 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 API 게이트웨이 통합과 API Gateway AWS SDK 통합의 차이점
>
> - `apigateway:invoke`: AWS SDK 서비스 통합에는 이에 상응하는 기능이 없습니다. 대신 최적화된 API Gateway 서비스에서 API Gateway 엔드포인트를 직접 호출합니다.

Amazon API Gateway를 사용하여 HTTP 및 REST API를 생성, 게시, 유지 및 모니터링합니다. API Gateway와 통합하려면 코드를 작성하거나 다른 인프라를 사용하지 않고도 API Gateway HTTP 또는 API Gateway REST 엔드포인트를 직접 호출하는 Step Functions에서 `Task` 상태를 정의합니다. `Task` 상태 정의에는 API 직접 호출에 필요한 모든 정보가 포함됩니다. 다른 인증 방법을 선택할 수도 있습니다.

API Gateway 기능 지원

Step Functions API Gateway 통합은 API Gateway 기능 전부가 아닌 일부만 지원합니다. 지원되는 기능에 대한 자세한 목록은 다음을 참조하세요.

- Step Functions API Gateway REST API 및 API Gateway HTTP API 통합 모두에서 지원됩니다.

  - 권한 부여자: IAM(Signature Version 4 사용), 인증 없음, Lambda 권한 부여자(요청 파라미터 기반 및 사용자 지정 헤더가 있는 토큰 기반)
  - API 유형: 리전별
  - API 관리: API Gateway API 도메인 이름, API 단계, 경로, 쿼리 파라미터, 요청 본문
- Step Functions API Gateway HTTP API 통합에서 지원됩니다. 엣지에 최적화된 API에 대한 옵션을 제공하는 Step Functions API Gateway REST API 통합은 지원되지 않습니다.
- 다음은 Step Functions API Gateway 통합에서 지원되지 않습니다.
  - 권한 부여자: Amazon Cognito, 기본 Open ID Connect/OAuth 2.0, 토큰 기반 Lambda 권한 부여자를 위한 인증 헤더
  - API 유형: 프라이빗
  - API 관리: 사용자 지정 도메인 이름

API Gateway, HTTP 및 REST API에 대한 자세한 내용은 다음을 참조하세요.

- Amazon API Gateway 개념 페이지
- API Gateway 개발자 안내서의 HTTP API와 REST API 중에서 선택

요청 형식

`Task` 상태 정의를 만들면 Step Functions에서 파라미터를 검증하고 직접 호출을 수행하는 데 필요한 URL을 빌드한 다음 API를 직접적으로 호출합니다. 응답에는 HTTP 상태 코드, 헤더 및 응답 본문이 포함됩니다. 요청 형식에는 필수 및 선택적 파라미터가 있습니다.

필수 요청 파라미터

- `ApiEndpoint`
  - 유형: `String`
  - API Gateway URL의 호스트 이름입니다. 형식은 *`<API ID>`*`.execute-api.`*`<region>`*`.amazonaws.com`입니다.

    API ID에는 영숫자 조합(`0123456789abcdefghijklmnopqrstuvwxyz`)만 사용할 수 있습니다.
- `Method`
  - 유형: `Enum`
  - HTTP 메서드로, 다음 중 하나여야 합니다.

- GET
- POST
- PUT
- DELETE
- PATCH
- HEAD
- OPTIONS

선택적 요청 파라미터

- Headers
  - 유형: JSON
  - HTTP 헤더는 같은 키와 연결된 값의 목록을 허용합니다.
- Stage
  - 유형: String
  - API가 API Gateway에 배포되는 단계의 이름입니다. $default 단계를 사용하는 모든 HTTP API의 경우 선택 사항입니다.
- Path
  - 유형: String
  - API 엔드포인트 다음에 추가되는 경로 파라미터입니다.
- QueryParameters
  - 유형: JSON
  - 쿼리 문자열은 같은 키와 연결된 값의 목록만 허용됩니다.
- RequestBody
  - 유형: JSON 또는 String
  - HTTP 요청 본문입니다. 유형은 JSON 객체나 String일 수 있습니다. RequestBody는 PATCH, POST 및 PUT HTTP 메서드에만 지원됩니다.
- AllowNullValues
  - 유형: BOOLEAN — 기본값: false

- 기본 설정을 사용하면 요청 입력 상태의 모든 null 값이 API로 전송되지 않습니다. 다음 예시에서는 스테이트 머신 정의에 로 `AllowNullValues` `true` 설정되어 있지 않는 한 `category` 필드가 요청에 포함되지 않습니다.

```
{
    "NewPet": {
        "type": "turtle",
        "price": 123,
        "category": null
    }
}
```

> **Note**
>
> 기본적으로 요청 입력 상태에 null 값이 있는 필드는 API로 전송되지 않습니다. 상태 머신 정의에서 로 `AllowNullValues` 설정하여 null 값을 API로 강제로 전송할 수 있습니다. `true`

- `AuthType`
  - 유형: `JSON`
  - 인증 방법입니다. 기본 방법은 `NO_AUTH`입니다. 허용 값은 다음과 같습니다.
    - `NO_AUTH`
    - `IAM_ROLE`
    - `RESOURCE_POLICY`

    자세한 내용은 인증 및 권한 부여를 참조하세요.

> **Note**
>
> 보안상의 이유로 현재 다음 HTTP 헤더 키는 허용되지 않습니다.
>
> - `X-Forwarded`, `X-Amz` 또는 `X-Amzn` 접두사가 있는 모든 항목
> - `Authorization`
> - `Connection`
> - `Content-md5`
> - `Expect`

- Host
- Max-Forwards
- Proxy-Authenticate
- Server
- TE
- Transfer-Encoding
- Trailer
- Upgrade
- Via
- Www-Authenticate

다음 코드 예제에서는 Step Functions를 사용하여 API Gateway를 간접적으로 호출하는 방법을 보여줍니다.

```
{
    "Type": "Task",
    "Resource":"arn:aws:states:::apigateway:invoke",
    "Parameters": {
        "ApiEndpoint": "example.execute-api.us-east-1.amazonaws.com",
        "Method": "GET",
        "Headers": {
            "key": ["value1", "value2"]
        },
        "Stage": "prod",
        "Path": "bills",
        "QueryParameters": {
            "billId": ["123456"]
        },
        "RequestBody": {},
        "AuthType": "NO_AUTH"
    }
}
```

인증 및 권한 부여

다음 인증 방법을 사용할 수 있습니다.

- 권한 없음: 권한 부여 방법을 사용하지 않고 API를 직접 호출합니다.
- IAM 역할: 이 메서드를 사용하면 Step Functions에서 상태 시스템 역할을 가정하고 Signature Version 4(SigV4)로 요청에 서명한 다음 API를 직접적으로 호출합니다.
- 리소스 정책: Step Functions는 요청을 인증한 다음 API를 직접적으로 호출합니다. 다음을 지정하는 API에 리소스 정책을 연결해야 합니다.

  1. API Gateway를 간접적으로 호출할 상태 시스템입니다.

     > **Important**
     >
     > 액세스를 제한하려면 상태 시스템을 지정해야 합니다. 그렇지 않으면 API에 대한 리소스 정책 인증을 사용하여 API Gateway 요청을 인증하는 모든 상태 시스템에 액세스 권한이 부여됩니다.

  2. 해당 Step Functions는 API Gateway를 직접적으로 호출하는 서비스입니다(`"Service": "states.amazonaws.com"`).
  3. 액세스할 리소스는 다음과 같습니다.
     - *region*
     - 지정된 리전의 *account-id*
     - *api-id*
     - *stage-name*
     - *HTTP-VERB*(메서드)
     - *resource-path-specifier*

  리소스 정책 예제는 Step Functions의 IAM 정책 및 API Gateway를 참조하세요.

  리소스 형식에 대한 자세한 내용은 API Gateway 개발자 안내서의 API Gateway의 API 실행 권한의 리소스 형식을 참조하세요.

  > **Note**
  >
  > 리소스 정책은 REST API에서만 지원됩니다.

서비스 통합 패턴

API Gateway 통합은 두 가지 서비스 통합 패턴을 지원합니다.

- 요청 및 응답 - 기본 통합 패턴입니다. 이를 사용하면 Step Functions에서 HTTP 응답을 수신한 직후에 다음 단계로 진행할 수 있습니다.
- 작업 토큰을 사용하여 콜백 대기(.waitForTaskToken) - 페이로드와 함께 작업 토큰이 반환될 때까지 기다립니다. .waitForTaskToken패턴을 사용하려면 다음 예와 같이 작업 정의의 Resource 필드 끝에 ForTaskToken .wait를 추가합니다.

```
{
    "Type": "Task",
    "Resource":"arn:aws:states:::apigateway:invoke.waitForTaskToken",
    "Parameters": {
        "ApiEndpoint": "example.execute-api.us-east-1.amazonaws.com",
        "Method": "POST",
        "Headers": {
            "TaskToken.$": "States.Array($$.Task.Token)"
        },
        "Stage": "prod",
        "Path": "bills/add",
        "QueryParameters": {},
        "RequestBody": {
            "billId": "my-new-bill"
        },
        "AuthType": "IAM_ROLE"
    }
}
```

출력 형식

다음 출력 파라미터가 제공됩니다.

명칭	유형	설명
ResponseBody	JSON 또는 String	API 직접 호출의 응답 본문입니다.
Headers	JSON	응답 헤더입니다.
StatusCode	Integer	응답의 HTTP 상태 코드입니다.

명칭	유형	설명
StatusText	String	응답의 상태 텍스트입니다.

다음은 응답 예제입니다.

```
{
    "ResponseBody": {
        "myBills": []
    },
    "Headers": {
        "key": ["value1", "value2"]
    },
    "StatusCode": 200,
    "StatusText": "OK"
}
```

오류 처리

오류가 발생하면 다음과 같이 error 및 cause가 반환됩니다.

- HTTP 상태 코드를 사용할 수 있는 경우 오류는 ApiGateway.*<HTTP Status Code>* 형식으로 반환됩니다.
- HTTP 상태 코드를 사용할 수 없는 경우 오류는 ApiGateway.*<Exception>* 형식으로 반환됩니다.

두 경우 모두 cause는 문자열로 반환됩니다.

다음 예제에서는 오류가 발생한 응답을 보여줍니다.

```
{
    "error": "ApiGateway.403",
    "cause": "{\"message\":\"Missing Authentication Token\"}"
}
```

> **Note**
>
> 2XX 상태 코드는 성공을 나타내며 오류는 반환되지 않습니다. 다른 모든 상태 코드나 예외 발생으로 인해 오류가 발생합니다.

자세한 내용은 다음을 참조하세요.

- API Gateway 개발자 안내서의 Amazon API Gateway 개념
- 아마존 API 게이트웨이에 대한 IAM 정책
- API Gateway 직접 호출하는 방법을 보여주는 샘플 프로젝트

API Gateway 개발자 안내서의 Amazon API Gateway 개념

Step Functions를 사용하여 Athena 직접 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 아테나 통합이 Athena SDK 통합과 어떻게 다른가 AWS
>
> - 작업 실행(.sync) 통합 패턴이 지원됩니다.
> - 요청 및 응답 통합 패턴에 대한 최적화는 없습니다.
> - 작업 토큰을 사용하여 콜백 대기 통합 패턴은 지원되지 않습니다.

Amazon Athena와의 AWS Step Functions 서비스 통합을 통해 Step Functions를 사용하여 쿼리 실행을 시작 및 중지하고 쿼리 결과를 가져올 수 있습니다. Step Functions를 사용하면 임시 또는 예약된 데이터 쿼리를 실행하고 S3 데이터 레이크를 대상으로 결과를 검색할 수 있습니다. Athena는 서버리스 서비스이므로 설정하거나 관리할 인프라가 없으며 실행한 쿼리에 대해서만 비용을 지불하면 됩니다.

Amazon AWS Step Functions Athena와 통합하려면 제공된 Athena 서비스 통합 API를 사용합니다.

서비스 통합 API는 해당 Athena API와 동일합니다. 다음 표에 나와 있는 것처럼 일부 API는 일부 통합 패턴을 지원하지 않습니다.

API	요청 및 응답	작업 실행(.sync)
`StartQueryExecution`	✓	✓
`StopQueryExecution`	✓	
`GetQueryExecution`	✓	

API	요청 및 응답	작업 실행(.sync)
GetQueryResults	✓	

지원되는 Amazon Athena API는 다음과 같습니다.

> **Note**
>
> Step Functions에는 작업의 최대 입력 또는 결과 데이터 크기에 대한 할당량이 있습니다. 이를 통해 다른 서비스에 데이터를 보내거나 다른 서비스로부터 데이터를 받을 때 UTF-8 인코딩 문자열로 데이터 256KB까지 제한됩니다. 상태 시스템 실행과 관련된 할당량 섹션을 참조하십시오.

- StartQueryExecution
  - 요청 구문
  - 지원되는 파라미터:
    - ClientRequestToken
    - ExecutionParameters
    - QueryExecutionContext
    - QueryString
    - ResultConfiguration
    - WorkGroup
  - Response syntax
- StopQueryExecution
  - 요청 구문
  - 지원되는 파라미터:
    - QueryExecutionId
- GetQueryExecution
  - 요청 구문
  - 지원되는 파라미터:
    - QueryExecutionId
  - Response syntax

- GetQueryResults
  - 요청 구문
  - 지원되는 파라미터:
    - MaxResults
    - NextToken
    - QueryExecutionId
  - Response syntax

다음에는 Athena 쿼리를 시작하는 Task 상태가 포함됩니다.

```
"Start an Athena query": {
  "Type": "Task",
  "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
  "Parameters": {
    "QueryString": "SELECT * FROM \"myDatabase\".\"myTable\" limit 1",
    "WorkGroup": "primary",
    "ResultConfiguration": {
       "OutputLocation": "s3://athenaQueryResult"
    }
  },
  "Next": "Get results of the query"
}
```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 통합 서비스용 IAM 정책

Step AWS Batch Functions를 사용한 관리

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 AWS Batch 통합과 AWS BatchAWS SDK 통합의 차이점
>
> - 작업 실행(.sync) 통합 패턴을 사용할 수 있습니다.
>
> 요청 및 응답 또는 작업 토큰을 사용하여 콜백 대기 통합 패턴에 대한 최적화는 없습니다.

지원되는 AWS Batch API:

- SubmitJob
  - 요청 구문
  - 지원되는 파라미터:
    - ArrayProperties
    - ContainerOverrides
    - DependsOn
    - JobDefinition
    - JobName
    - JobQueue
    - Parameters
    - RetryStrategy
    - Timeout
    - Tags
  - 응답 구문

> 의 매개변수는 Step Functions 다음과 같이 표현됩니다. PascalCase
>
> 네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

다음은 AWS Batch 작업을 제출하고 완료될 때까지 대기하는 `Task` 상태를 포함합니다.

```
{
  "StartAt": "BATCH_JOB",
  "States": {
    "BATCH_JOB": {
      "Type": "Task",
      "Resource": "arn:aws:states:::batch:submitJob.sync",
      "Parameters": {
        "JobDefinition": "preprocessing",
        "JobName": "PreprocessingBatchJob",
        "JobQueue": "SecondaryQueue",
        "Parameters.$": "$.batchjob.parameters",
```

```
        "ContainerOverrides": {
          "ResourceRequirements": [
            {
              "Type": "VCPU",
              "Value": "4"
            }
          ]
        }
      },
      "End": true
    }
  }
}
```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 통합 서비스용 IAM 정책

Step Functions 사용을 통한 Amazon Bedrock 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

주제

- Amazon Bedrock 서비스 통합 API
- Amazon Bedrock 통합용 태스크 상태 정의

Amazon Bedrock 서비스 통합 API

AWS Step Functions과 Amazon Bedrock을 통합하려면 다음 API를 사용합니다. 이러한 API는 해당 Amazon Bedrock API와 유사하지만 전달되는 요청 필드에 약간의 차이가 있습니다.

다음 표에는 각 서비스 통합 API와 해당 Amazon Bedrock API 간의 차이점이 나와 있습니다.

Amazon Bedrock 서비스 통합 API 및 해당 Amazon Bedrock API

Amazon Bedrock 서비스 통합 API	해당 Amazon Bedrock API	차이
InvokeModel	InvokeModel	Amazon Bedrock 서비스 통합 API 요청 본문에는 다음 추가 파라미터가 포함됩니다.

Amazon Bedrock 서비스 통합 API	해당 Amazon Bedrock API	차이
요청 본문에 제공된 입력을 사용하여 추론을 실행하기 위해 지정된 Amazon Bedrock 모델을 호출합니다. `InvokeModel` 을 사용하여 텍스트 모델, 이미지 모델 및 임베딩 모델의 추론을 실행합니다.		• Body - 콘텐츠 유형 요청 헤더에 지정된 형식으로 입력 데이터를 지정합니다. Body에는 대상 모델별 파라미터가 포함되어 있습니다. `InvokeModel` API를 사용하는 경우 Body 파라미터를 지정해야 합니다. Step Functions는 Body에 입력된 값의 유효성을 검사하지 않습니다. Amazon Bedrock 최적화 통합을 사용하여 Body를 지정할 때 최대 256KB의 페이로드를 지정할 수 있습니다. 페이로드가 256KB를 초과하는 경우 `Input`을 사용하는 것이 좋습니다. • `Input` - 입력 데이터를 검색할 소스를 지정합니다. 이 옵션 필드는 Step Functions를 사용한 Amazon Bedrock 최적화 통합에만 해당됩니다. 이 필드에서 `S3Uri`를 지정할 수 있습니다. 파라미터의 Body 또는 `Input` 중 하나를 지정할 수 있지만 둘 다 지정할 수는 없습니다. `ContentType` 을 지정하지 않고 `Input`을 지정하면 입

Amazon Bedrock 서비스 통합 API	해당 Amazon Bedrock API	차이
		력 데이터 소스의 콘텐츠 유형이 `ContentType` 의 값이 됩니다. • `Output` - API 응답이 기록되는 대상을 지정합니다. 이 옵션 필드는 Step Functions를 사용한 Amazon Bedrock 최적화 통합에만 해당됩니다. 이 필드에서 `S3Uri`를 지정할 수 있습니다. 이 필드를 지정하면 API 응답 본문이 원래 출력의 Amazon S3 위치에 대한 참조로 대체됩니다. 다음 예제는 Amazon Bedrock 통합을 위한 InvokeModel API 구문을 보여줍니다. `{ "ModelId": String, // required "Accept": String, // default: application/json "ContentType": String, // default: application/json "Input": { // not from Bedrock API "S3Uri": String }, "Output": { // not from Bedrock API "S3Uri": String`

Amazon Bedrock 서비스 통합 API	해당 Amazon Bedrock API	차이
		} }
CreateModelCustomizationJob 기본 모델을 사용자 지정하기 위한 미세 튜닝 작업을 생성합니다.	CreateModelCustomizationJob	None
CreateModelCustomizationJob.sync 기본 모델을 사용자 지정하기 위한 미세 튜닝 작업을 생성합니다.	CreateModelCustomizationJob	None

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Amazon Bedrock 통합용 태스크 상태 정의

다음 태스크 상태 정의는 상태 머신의 Amazon Bedrock과 통합하는 방법을 보여 줍니다. 이 예제는 경로 `result_one`으로 지정된 모델 간접 호출의 전체 결과를 추출하는 태스크 상태를 보여 줍니다. 이는 파운데이션 모델의 추론 파라미터를 기반으로 합니다. 이 예제에서는 Cohere Command 대규모 언어 모델(LLM)을 사용합니다.

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::bedrock:invokeModel",
  "Parameters": {
    "ModelId": "cohere.command-text-v14",
    "Body": {
      "prompt.$": "$.prompt_one",
      "max_tokens": 250
    },
    "ContentType": "application/json",
    "Accept": "*/*"
```

```
  },
  "ResultPath": "$.result_one",
  "ResultSelector": {
    "result_one.$": "$.Body.generations[0].text"
  },
  "End": true
}
```

> **Tip**
>
> 사용자 컴퓨터에 AWS 계정통합되는 상태 머신의 예를 Amazon Bedrock 배포하려면 을 참조하십시오Amazon Bedrock을 사용하여 AI 프롬프트 체이닝 수행.

AWS CodeBuild Step 함수를 사용한 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> **최적화된 CodeBuild 통합과 CodeBuild AWS SDK 통합의 차이점**
>
> - 작업 실행(.sync) 통합 패턴이 지원됩니다.
> - `StopBuild`또는 `StopBuildBatch` 를 호출한 후에는 빌드 또는 빌드의 상태를 CodeBuild 확정하기 위한 내부 작업이 완료될 때까지 빌드 또는 빌드 배치를 즉시 삭제할 수 없습니다. 이 기간 동안 `BatchDeleteBuilds` 또는 `DeleteBuildBatch`를 사용하려고 하면 빌드나 빌드 배치가 삭제되지 않을 수 있습니다. `BatchDeleteBuilds` 및 `DeleteBuildBatch`의 최적화된 서비스 통합에는 중지 후 즉시 삭제하는 사용 사례를 단순화하기 위한 내부 재시도가 포함되어 있습니다.

와의 AWS Step Functions 서비스 통합을 AWS CodeBuild 통해 Step Functions를 사용하여 빌드를 트리거, 중지, 관리하고 빌드 보고서를 공유할 수 있습니다. Step Functions를 사용하면 애플리케이션의 소프트웨어 변경 사항을 검증할 수 있도록 지속적인 통합 파이프라인을 설계하고 실행할 수 있습니다.

다음 표에 나와 있는 것처럼 일부 API는 일부 통합 패턴을 지원하지 않습니다.

API	요청 및 응답	작업 실행(.sync)
StartBuild	✓	✓
StopBuild	✓	
BatchDeleteBuilds	✓	
BatchGetReports	✓	
StartBuildBatch	✓	✓
StopBuildBatch	✓	
RetryBuildBatch	✓	✓
DeleteBuildBatch	✓	

의 파라미터는 Step Functions 다음과 같이 표현됩니다. PascalCase
네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

지원되는 CodeBuild API 및 구문:

- `StartBuild`
  - 요청 구문
  - 지원되는 파라미터:
    - `ProjectName`
    - `ArtifactsOverride`
    - `BuildspecOverride`
    - `CacheOverride`
    - `CertificateOverride`
    - `ComputeTypeOverride`
    - `EncryptionKeyOverride`

    - EnvironmentTypeOverride
    - EnvironmentVariablesOverride
    - GitCloneDepthOverride
    - GitSubmodulesConfigOverride
    - IdempotencyToken
    - ImageOverride
    - ImagePullCredentialsTypeOverride
    - InsecureSslOverride
    - LogsConfigOverride
    - PrivilegedModeOverride
    - QueuedTimeoutInMinutesOverride
    - RegistryCredentialOverride
    - ReportBuildStatusOverride
    - SecondaryArtifactsOverride
    - SecondarySourcesOverride
    - SecondarySourcesVersionOverride
    - ServiceRoleOverride
    - SourceAuthOverride
    - SourceLocationOverride
    - SourceTypeOverride
    - SourceVersion
    - TimeoutInMinutesOverride
  - 응답 구문
- StopBuild
  - 요청 구문
  - 지원되는 파라미터:
    - Id
  - 응답 구문
- BatchDeleteBuilds
  - 요청 구문

- 지원되는 파라미터:
  - Ids
- 응답 구문

- BatchGetReports
  - 요청 구문
  - 지원되는 파라미터:
    - ReportArns
  - 응답 구문
- StartBuildBatch
  - 요청 구문
  - 지원되는 파라미터:
    - ProjectName
    - ArtifactsOverride
    - BuildBatchConfigOverride
    - BuildspecOverride
    - BuildTimeoutInMinutesOverride
    - CacheOverride
    - CertificateOverride
    - ComputeTypeOverride
    - DebugSessionEnabled
    - EncryptionKeyOverride
    - EnvironmentTypeOverride
    - EnvironmentVariablesOverride
    - GitCloneDepthOverride
    - GitSubmodulesConfigOverride
    - IdempotencyToken
    - ImageOverride
    - ImagePullCredentialsTypeOverride
    - InsecureSslOverride
    - LogsConfigOverride

    - PrivilegedModeOverride
    - QueuedTimeoutInMinutesOverride
    - RegistryCredentialOverride
    - ReportBuildBatchStatusOverride
    - SecondaryArtifactsOverride
    - SecondarySourcesOverride
    - SecondarySourcesVersionOverride
    - ServiceRoleOverride
    - SourceAuthOverride
    - SourceLocationOverride
    - SourceTypeOverride
    - SourceVersion
  - 응답 구문
- StopBuildBatch
  - 요청 구문
  - 지원되는 파라미터:
    - Id
  - 응답 구문
- RetryBuildBatch
  - 요청 구문
  - 지원되는 파라미터:
    - Id
    - IdempotencyToken
    - RetryType
  - 응답 구문
- DeleteBuildBatch
  - 요청 구문
  - 지원되는 파라미터:
    - Id
  - 응답 구문

> **Note**
>
> BatchDeleteBuilds에 대한 JSONPath 재귀적 하강(..) 연산자를 사용할 수 있습니다. 이렇게 하면 배열이 반환되며, 다음 예제와 같이 Arn 필드를 StartBuild에서 복수 Ids 파라미터로 변환할 수 있습니다.
>
> ```
> "BatchDeleteBuilds": {
>     "Type": "Task",
>     "Resource": "arn:aws:states:::codebuild:batchDeleteBuilds",
>     "Parameters": {
>         "Ids.$": "$.Build..Arn"
>     },
>     "Next": "MyNextState"
> },
> ```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Step Functions를 사용하여 DynamoDB API 직접 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> **Note**
>
> Step Functions에는 작업의 최대 입력 또는 결과 데이터 크기에 대한 할당량이 있습니다. 이를 통해 다른 서비스에 데이터를 보내거나 다른 서비스로부터 데이터를 받을 때 UTF-8 인코딩 문자열로 데이터 256KB까지 제한됩니다. 상태 시스템 실행과 관련된 할당량 섹션을 참조하십시오.

> **Note**
>
> 최적화된 DynamoDB 통합이 DynamoDB SDK 통합과 다른 점 AWS
>
> - 요청 및 응답 통합 패턴에 대한 최적화는 없습니다.
> - 작업 토큰을 사용하여 콜백 대기 통합 패턴은 지원되지 않습니다.

- 최적화된 통합을 통해 GetItem, PutItem, UpdateItem 및 DeleteItem API 작업만 사용할 수 있습니다. 기타 API 작업 (예 CreateTable: AWS DynamoDB SDK 통합을 사용하여 사용 가능).

지원되는 Amazon DynamoDB API 및 구문은 다음과 같습니다.

- GetItem
  - 요청 구문
  - 지원되는 파라미터:
    - Key
    - TableName
    - AttributesToGet
    - ConsistentRead
    - ExpressionAttributeNames
    - ProjectionExpression
    - ReturnConsumedCapacity
  - 응답 구문
- PutItem
  - 요청 구문
  - 지원되는 파라미터:
    - Item
    - TableName
    - ConditionalOperator
    - ConditionExpression
    - Expected
    - ExpressionAttributeNames
    - ExpressionAttributeValues
    - ReturnConsumedCapacity
    - ReturnItemCollectionMetrics
    - ReturnValues
  - 응답 구문

- DeleteItem
  - 요청 구문
  - 지원되는 파라미터:
    - Key
    - TableName
    - ConditionalOperator
    - ConditionExpression
    - Expected
    - ExpressionAttributeNames
    - ExpressionAttributeValues
    - ReturnConsumedCapacity
    - ReturnItemCollectionMetrics
    - ReturnValues
  - 응답 구문
- UpdateItem
  - 요청 구문
  - 지원되는 파라미터:
    - Key
    - TableName
    - AttributeUpdates
    - ConditionalOperator
    - ConditionExpression
    - Expected
    - ExpressionAttributeNames
    - ExpressionAttributeValues
    - ReturnConsumedCapacity
    - ReturnItemCollectionMetrics
    - ReturnValues
    - UpdateExpression
  - 응답 구문

의 파라미터는 다음과 같이 표현됩니다. Step Functions PascalCase

네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

다음은 DynamoDB에서 보낸 메시지를 검색하는 `Task` 상태입니다.

```
"Read Next Message from DynamoDB": {
  "Type": "Task",
  "Resource": "arn:aws:states:::dynamodb:getItem",
  "Parameters": {
    "TableName": "TransferDataRecords-DDBTable-3I41R5L5EAGT",
    "Key": {
      "MessageId": {"S.$": "$.List[0]"}
    }
  },
  "ResultPath": "$.DynamoDB",
  "Next": "Send Message to SQS"
},
```

작업 중인 예제에서 이 상태를 보려면 데이터 레코드 전송(Lambda, DynamoDB, Amazon SQS) 샘플 프로젝트를 확인하십시오.

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 통합 서비스용 IAM 정책

Step Functions를 사용하여 Amazon ECS 또는 Fargate 작업 관리

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

최적화된 아마존 ECS/Fargate 통합이 아마존 ECS 또는 Fargate SDK 통합과 어떻게 다른지 AWS

- 작업 실행(.sync) 통합 패턴이 지원됩니다.
- `ecs:runTask`에서 HTTP 200 응답을 반환할 수 있지만 다음과 같이 비어 있지 않은 `Failures` 필드가 있습니다.

- 요청 응답: 응답을 반환하고 태스크를 실패하지 않습니다. 이는 최적화가 없는 것과 동일합니다.
- 작업 실행 또는 태스크 토큰: 비어 있지 않은 `Failures` 필드가 발견되면 `AmazonECS.Unknown` 오류가 발생하여 태스크가 실패합니다.

지원되는 Amazon ECS/Fargate API 및 구문은 다음과 같습니다.

Step Functions의 파라미터는 다음과 같이 표현됩니다. PascalCase
네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

- `RunTask`는 지정된 작업 정의를 사용하여 새 작업을 시작합니다.
  - 요청 구문
  - 지원되는 파라미터:
    - `Cluster`
    - `Group`
    - `LaunchType`
    - `NetworkConfiguration`
    - `Overrides`
    - `PlacementConstraints`
    - `PlacementStrategy`
    - `PlatformVersion`
    - `PropagateTags`
    - `TaskDefinition`
    - `EnableExecuteCommand`
  - 응답 구문

Amazon ECS 작업에 데이터 전달

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

overrides를 사용하여 컨테이너에 대한 기본 명령을 재정의하고 입력을 Amazon ECS 작업으로 전달할 수 있습니다. ContainerOverride 섹션을 참조하십시오. 이 예제에서는 Task 입력에서 Task 상태로 값을 에 전달하는 JsonPath 데 사용했습니다.

다음에는 Amazon ECS 작업을 실행하고 완료할 때까지 기다리는 Task 상태가 포함됩니다.

```
{
 "StartAt": "Run an ECS Task and wait for it to complete",
 "States": {
   "Run an ECS Task and wait for it to complete": {
     "Type": "Task",
     "Resource": "arn:aws:states:::ecs:runTask.sync",
     "Parameters": {
                "Cluster": "cluster-arn",
                "TaskDefinition": "job-id",
                "Overrides": {
                    "ContainerOverrides": [
                        {
                            "Name": "container-name",
                            "Command.$": "$.commands"
                        }
                    ]
                }
            },
     "End": true
    }
  }
}
```

ContainerOverrides의 "Command.$": "$.commands" 줄은 상태 입력에서 컨테이너로 명령을 전달합니다.

이전 예제의 경우 실행에 대한 입력이 다음과 같은 경우 각 명령이 컨테이너 재정의로 전달됩니다.

```
{
  "commands": [
    "test command 1",
    "test command 2",
    "test command 3"
  ]
}
```

다음에는 Amazon ECS 작업을 실행한 다음 작업 토큰이 반환될 때까지 기다리는 Task 상태가 포함됩니다. 작업 토큰을 사용하여 콜백 대기 섹션을 참조하십시오.

```
{
   "StartAt":"Manage ECS task",
   "States":{
      "Manage ECS task":{
         "Type":"Task",
         "Resource":"arn:aws:states:::ecs:runTask.waitForTaskToken",
         "Parameters":{
            "LaunchType":"FARGATE",
            "Cluster":"cluster-arn",
            "TaskDefinition":"job-id",
            "Overrides":{
               "ContainerOverrides":[
                  {
                     "Name":"container-name",
                     "Environment":[
                        {
                           "Name":"TASK_TOKEN_ENV_VARIABLE",
                           "Value.$":"$$.Task.Token"
                        }
                     ]
                  }
               ]
            }
         },
         "End":true
      }
   }
}
```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Step Functions를 사용하여 Amazon EKS 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

최적화된 아마존 EKS 통합이 아마존 EKS SDK AWS 통합과 어떻게 다른가

- 작업 실행(.sync) 통합 패턴이 지원됩니다.
- 요청 및 응답 통합 패턴에 대한 최적화는 없습니다.
- 작업 토큰을 사용하여 콜백 대기 통합 패턴은 지원되지 않습니다.

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 통합 서비스용 IAM 정책

Step Functions는 Amazon Elastic Kubernetes Service와 통합할 수 있도록 두 가지 유형의 서비스 통합 API를 제공합니다. 하나는 Amazon EKS API를 사용하여 Amazon EKS 클러스터를 만들고 관리할 수 있도록 합니다. 다른 하나는 Kubernetes API를 사용하여 클러스터와 상호 작용하고 작업을 애플리케이션 워크플로의 일부로 실행할 수 있도록 합니다. Step Functions를 사용하여 만든 Amazon EKS 클러스터, eksctl 도구 또는 Amazon EKS 콘솔로 만든 Amazon EKS 클러스터 또는 유사한 방법으로 Kubernetes API 통합을 사용할 수 있습니다. 자세한 내용은 Amazon EKS 사용 설명서의 Amazon EKS 클러스터 생성을 참조하세요.

Note

Step Functions EKS 통합은 퍼블릭 엔드포인트 액세스 권한이 있는 Kubernetes API만 지원합니다. 기본적으로 EKS 클러스터 API 서버 엔드포인트에는 퍼블릭 액세스 권한이 있습니다. 자세한 내용은 Amazon EKS 사용 설명서의 Amazon EKS 클러스터 엔드포인트 액세스 제어를 참조하세요.

Step Functions는 실행이 중지된 경우 Amazon EKS 클러스터를 자동으로 종료하지 않습니다. Amazon EKS 클러스터가 종료되기 전에 상태 시스템이 중지하면 클러스터가 무기한 계속 실행될 수 있으므로 추가 요금이 발생할 수 있습니다. 이를 방지하려면 생성된 모든 Amazon EKS 클러스터를 올바르게 종료해야 합니다. 자세한 내용은 다음을 참조하세요.

- Amazon EKS 사용 설명서의 클러스터 삭제
- 서비스 통합 패턴의 작업 실행(.sync)

> **Note**
>
> Step Functions에는 작업의 최대 입력 또는 결과 데이터 크기에 대한 할당량이 있습니다. 이를 통해 다른 서비스에 데이터를 보내거나 다른 서비스로부터 데이터를 받을 때 UTF-8 인코딩 문자열로 데이터 256KB까지 제한됩니다. 상태 시스템 실행과 관련된 할당량 섹션을 참조하십시오.

Kubernetes API 통합

Step Functions는 다음 Kubernetes API를 지원합니다.

RunJob

`eks:runJob` 서비스 통합을 사용하면 Amazon EKS 클러스터에서 작업을 실행할 수 있습니다. 이 `eks:runJob.sync` 변형을 사용하면 작업이 완료될 때까지 기다릴 수 있으며 선택적으로 로그를 검색할 수 있습니다.

Kubernetes API 서버는 상태 시스템에서 사용하는 IAM 역할에 권한을 부여해야 합니다. 자세한 정보는 권한을 참조하세요.

작업 실행(`.sync`) 패턴의 경우 작업 상태는 폴링을 통해 결정됩니다. Step Functions는 처음에 1분당 약 1회 속도로 폴링합니다. 결국 이 속도는 5분당 약 1회 폴링으로 느려집니다. 더 자주 폴링해야 하거나 폴링 전략을 더 세밀하게 제어해야 하는 경우 `eks:call` 통합을 사용하여 작업 상태를 쿼리하면 됩니다.

`eks:runJob` 통합은 `batch/v1` Kubernetes 작업에만 해당됩니다. 자세한 내용은 Kubernetes 문서의 작업을 참조하세요. 사용자 지정 리소스를 포함하여 다른 Kubernetes 리소스를 관리하려면 `eks:call` 서비스 통합을 사용합니다. the section called “작업 상태 설문 조사 (Lambda AWS Batch,) ” 샘플 프로젝트에 나와 있는 것처럼 Step Functions를 사용하여 폴링 루프를 빌드할 수 있습니다.

지원되는 파라미터는 다음과 같습니다.

- `ClusterName`: 직접적으로 호출하려는 Amazon EKS 클러스터의 이름입니다.
  - `Type: String`
  - 필수 항목 여부: 예
- `CertificateAuthority`: 클러스터와 통신하는 데 필요한 Base64로 인코딩된 인증서 데이터입니다. 이 값은 아마존 EKS 콘솔에서 가져오거나 아마존 EKS API를 사용하여 얻을 수 있습니다. DescribeCluster

  - Type: String
  - 필수 항목 여부: 예
- Endpoint: Kubernetes API 서버에 대한 엔드포인트 URL입니다. 이 값은 아마존 EKS 콘솔에서 가져오거나 아마존 EKS API를 사용하여 얻을 수 있습니다. DescribeCluster
  - Type: String
  - 필수 항목 여부: 예
- Namespace: 작업을 실행할 네임스페이스입니다. 제공되지 않으면 default 네임스페이스가 사용됩니다.
  - Type: String
  - 필수 항목 여부: 아니요
- Job: Kubernetes 작업 정의입니다. Kubernets 문서의 작업을 참조하세요.
  - Type: JSON 또는 String
  - 필수 항목 여부: 예
- LogOptions: 선택적 로그 검색을 제어하는 옵션 집합입니다. 실행 작업(.sync) 서비스 통합 패턴을 사용하여 작업 완료를 기다리는 경우에만 적용 가능합니다.
  - Type: JSON
  - 필수 항목 여부: 아니요
  - 로그는 logs 키 아래의 응답에 포함됩니다. 작업 내에 포드가 여러 개 있을 수 있으며 포드마다 컨테이너가 여러 개 있을 수 있습니다.

```
{
  ...
  "logs": {
    "pods": {
      "pod1": {
        "containers": {
          "container1": {
            "log": <log>
          },
          ...
        }
      },
      ...
    }
  }
```

  - 로그 검색은 최선의 노력을 기반으로 수행됩니다. 로그를 검색하는 동안에 오류가 발생하는 경우 `log` 필드 대신 `error` 및 `cause` 필드가 표시됩니다.
- `LogOptions.RetrieveLogs`: 작업 완료 후 로그 검색을 활성화합니다. 기본적으로 로그는 검색되지 않습니다.
  - Type: `Boolean`
  - 필수 항목 여부: 아니요
- `LogOptions.RawLogs`: `RawLogs`를 true로 설정하면 로그는 JSON으로 파싱하려고 시도하지 않아도 원시 문자열로 반환됩니다. 기본적으로 로그는 가능한 경우 JSON으로 역직렬화됩니다. 경우에 따라 이러한 파싱으로 인해 많은 자릿수가 포함된 숫자의 정밀도 제한과 같은 원치 않는 변경 사항이 발생할 수 있습니다.
  - Type: `Boolean`
  - 필수 항목 여부: 아니요
- `LogOptions.LogParameters`: Kubernetes API의 읽기 로그 API는 로그 검색을 제어하는 쿼리 파라미터를 지원합니다. 예를 들어 `tailLines` 또는 `limitBytes`를 사용하여 검색된 로그의 크기를 제한하고 Step Functions 데이터 크기 할당량 이내로 유지할 수 있습니다. 자세한 내용은 Kubernetes API 참조의 읽기 로그 섹션을 참조하세요.
  - Type: `List of Strings`에 대한 `String` 맵
  - 필수 항목 여부: 아니요
  - 예제

```
"LogParameters": {
  "tailLines": [ "6" ]
}
```

다음 예제에는 작업을 실행하고 작업이 완료될 때까지 기다린 다음 작업 로그를 검색하는 `Task` 상태가 포함됩니다.

```
{
  "StartAt": "Run a job on EKS",
  "States": {
    "Run a job on EKS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:runJob.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
```

```
        "CertificateAuthority": "ANPAJ2UCCR6DPCEXAMPLE",
        "Endpoint": "https://AKIAIOSFODNN7EXAMPLE.yl4.us-east-1.eks.amazonaws.com",
        "LogOptions": {
          "RetrieveLogs": true
        },
        "Job": {
          "apiVersion": "batch/v1",
          "kind": "Job",
          "metadata": {
            "name": "example-job"
          },
          "spec": {
            "backoffLimit": 0,
            "template": {
              "metadata": {
                "name": "example-job"
              },
              "spec": {
                "containers": [
                  {
                    "name": "pi-2000",
                    "image": "perl",
                    "command": [ "perl" ],
                    "args": [
                      "-Mbignum=bpi",
                      "-wle",
                      "print bpi(2000)"
                    ]
                  }
                ],
                "restartPolicy": "Never"
              }
            }
          }
        },
        "End": true
    }
  }
}
```

`Call`

`eks:call` 서비스 통합을 사용하면 Kubernetes API를 사용하여 Kubernets API 엔드포인트를 통해 Kubernetes 리소스 객체를 읽고 쓸 수 있습니다.

Kubernetes API 서버는 상태 시스템에서 사용하는 IAM 역할에 권한을 부여해야 합니다. 자세한 정보는 권한을 참조하세요.

사용 가능한 작업에 대한 자세한 내용은 Kubernetes API 참조를 확인하세요.

지원되는 `Call` 파라미터는 다음과 같습니다.

- `ClusterName`: 직접적으로 호출하려는 Amazon EKS 클러스터의 이름입니다.
  - `Type`: 문자열 목록
  - 필수 항목 여부: 예
- `CertificateAuthority`: 클러스터와 통신하는 데 필요한 Base64로 인코딩된 인증서 데이터입니다. 이 값은 아마존 EKS 콘솔에서 가져오거나 아마존 EKS API를 사용하여 얻을 수 있습니다. DescribeCluster
  - `Type: String`
  - 필수 항목 여부: 예
- `Endpoint`: Kubernetes API 서버에 대한 엔드포인트 URL입니다. Amazon EKS 콘솔에서 또는 Amazon EKS API를 사용하여 이 값을 찾을 수 있습니다. DescribeCluster
  - `Type: String`
  - 필수 항목 여부: 예
- `Method`: 요청의 HTTP 메서드입니다. `GET`, `POST`, `PUT`, `DELETE`, `HEAD` 또는 `PATCH` 중 하나입니다.
  - `Type: String`
  - 필수 항목 여부: 예
- `Path`: Kubernetes REST API 작업의 HTTP 경로입니다.
  - `Type: String`
  - 필수 항목 여부: 예
- `QueryParameters`: Kubernetes REST API 작업의 HTTP 쿼리 파라미터입니다.
  - `Type: List of Strings`에 대한 `String` 맵
  - 필수 여부: 아니요
  - 예제

```
"QueryParameters": {
  "labelSelector": [ "job-name=example-job" ]
}
```

- RequestBody: Kubernetes REST API 작업의 HTTP 메시지 본문입니다.
  - Type: JSON 또는 String
  - 필수 여부: 아니요

다음에는eks:call을 사용하여 example-job 작업에 속하는 포드를 나열하는 Task 상태가 포함됩니다.

```
{
  "StartAt": "Call EKS",
  "States": {
    "Call EKS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:call",
      "Parameters": {
        "ClusterName": "MyCluster",
        "CertificateAuthority": "ANPAJ2UCCR6DPCEXAMPLE",
        "Endpoint": "https://444455556666.yl4.us-east-1.eks.amazonaws.com",
        "Method": "GET",
        "Path": "/api/v1/namespaces/default/pods",
        "QueryParameters": {
          "labelSelector": [
            "job-name=example-job"
          ]
        }
      },
      "End": true
    }
  }
}
```

다음에는 eks:call을 사용하여 example-job 작업을 삭제하고 propagationPolicy를 설정하여 작업 포드도 삭제되었는지 확인하는 Task 상태가 포함됩니다.

```
{
  "StartAt": "Call EKS",
  "States": {
```

```
    "Call EKS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:call",
      "Parameters": {
        "ClusterName": "MyCluster",
        "CertificateAuthority": "ANPAJ2UCCR6DPCEXAMPLE",
        "Endpoint": "https://444455556666.yl4.us-east-1.eks.amazonaws.com",
        "Method": "DELETE",
        "Path": "/apis/batch/v1/namespaces/default/jobs/example-job",
        "QueryParameters": {
          "propagationPolicy": [
            "Foreground"
          ]
        }
      },
      "End": true
    }
  }
}
```

지원되는 Amazon EKS API

지원되는 Amazon EKS API 및 구문은 다음과 같습니다.

- CreateCluster
  - 요청 구문
  - 응답 구문

  eks:createCluster 서비스 통합을 통해 Amazon EKS 클러스터가 생성될 때 IAM 역할은 관리자(system:masters 권한이 있음)로 Kubernetes RBAC 인증 테이블에 추가됩니다. 처음에는 해당 IAM 엔터티만 Kubernetes API 서버를 직접적으로 호출할 수 있습니다. 자세한 내용은 다음을 참조하세요.

  - Amazon EKS 사용 설명서의 클러스터의 사용자 또는 IAM 역할 관리
  - 권한 섹션

  Amazon EKS는 Amazon EKS에서 자동으로 다른 서비스를 직접적으로 호출하는 데 필요한 권한이 포함된 서비스 연결 역할을 사용합니다. 이러한 서비스 연결 역할이 아직 계정에 없으면 Step Functions에서 사용하는 IAM 역할에 iam:CreateServiceLinkedRole 권한을 추가해야 합니다. 자세한 내용은 Amazon EKS 사용 설명서의 서비스 연결 역할 사용을 참조하세요.

Step Functions에서 사용하는 IAM 역할에는 클러스터 IAM 역할을 Amazon EKS로 전달할 수 있는 `iam:PassRole` 권한이 있어야 합니다. 자세한 내용은 Amazon EKS 사용 설명서의 Amazon EKS 클러스터 IAM 역할을 참조하세요.

- `DeleteCluster`
  - 요청 구문
  - 응답 구문

  클러스터를 삭제하기 전에 Fargate 프로필이나 노드 그룹을 삭제해야 합니다.
- `CreateFargateProfile`
  - 요청 구문
  - 응답 구문

  Amazon EKS는 Amazon EKS에서 자동으로 다른 서비스를 직접적으로 호출하는 데 필요한 권한이 포함된 서비스 연결 역할을 사용합니다. 이러한 서비스 연결 역할이 아직 계정에 없으면 Step Functions에서 사용하는 IAM 역할에 `iam:CreateServiceLinkedRole` 권한을 추가해야 합니다. 자세한 내용은 Amazon EKS 사용 설명서의 서비스 연결 역할 사용을 참조하세요.

  일부 리전에서는 Amazon EKS on Fargate를 사용하지 못할 수 있습니다. 리전 가용성은 Amazon EKS 사용 설명서의 Fargate 섹션을 참조하세요.

  Step Functions에서 사용하는 IAM 역할에는 포드 실행 IAM 역할을 Amazon EKS로 전달할 수 있는 `iam:PassRole` 권한이 있어야 합니다. 자세한 내용은 Amazon EKS 사용 설명서의 포드 실행 역할을 참조하세요.
- `DeleteFargateProfile`
  - 요청 구문
  - 응답 구문
- `CreateNodegroup`
  - 요청 구문
  - 응답 구문

  Amazon EKS는 Amazon EKS에서 자동으로 다른 서비스를 직접적으로 호출하는 데 필요한 권한이 포함된 서비스 연결 역할을 사용합니다. 이러한 서비스 연결 역할이 아직 계정에 없으면 Step Functions에서 사용하는 IAM 역할에 `iam:CreateServiceLinkedRole` 권한을 추가해야 합니다. 자세한 내용은 Amazon EKS 사용 설명서의 서비스 연결 역할 사용을 참조하세요.

Step Functions에서 사용하는 IAM 역할에는 노드 IAM 역할을 Amazon EKS로 전달할 수 있는 `iam:PassRole` 권한이 있어야 합니다. 자세한 내용은 Amazon EKS 사용 설명서의 서비스 연결 역할 사용을 참조하세요.

- DeleteNodegroup
  - 요청 구문
  - 응답 구문

다음에는 Amazon EKS 클러스터를 만드는 Task가 포함됩니다.

```
{
  "StartAt": "CreateCluster.sync",
  "States": {
    "CreateCluster.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createCluster.sync",
      "Parameters": {
        "Name": "MyCluster",
        "ResourcesVpcConfig": {
          "SubnetIds": [
            "subnet-053e7c47012341234",
            "subnet-027cfea4b12341234"
          ]
        },
        "RoleArn": "arn:aws:iam::123456789012:role/MyEKSClusterRole"
      },
      "End": true
    }
  }
}
```

다음에는 Amazon EKS 클러스터를 삭제하는 Task 상태가 포함됩니다.

```
{
  "StartAt": "DeleteCluster.sync",
  "States": {
    "DeleteCluster.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteCluster.sync",
      "Parameters": {
        "Name": "MyCluster"
```

```
      },
      "End": true
    }
  }
}
```

다음에는 Fargate 프로필을 만드는 Task 상태가 포함됩니다.

```
{
  "StartAt": "CreateFargateProfile.sync",
  "States": {
    "CreateFargateProfile.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createFargateProfile.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "FargateProfileName": "MyFargateProfile",
        "PodExecutionRoleArn": "arn:aws:iam::123456789012:role/
MyFargatePodExecutionRole",
        "Selectors": [{
            "Namespace": "my-namespace",
            "Labels": { "my-label": "my-value" }
          }]
      },
      "End": true
    }
  }
}
```

다음에는 Fargate 프로필을 삭제하는 Task 상태가 포함됩니다.

```
{
  "StartAt": "DeleteFargateProfile.sync",
  "States": {
    "DeleteFargateProfile.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteFargateProfile.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "FargateProfileName": "MyFargateProfile"
      },
      "End": true
    }
```

```
  }
}
```

다음에는 노드 그룹을 만드는 Task 상태가 포함됩니다.

```
{
  "StartAt": "CreateNodegroup.sync",
  "States": {
    "CreateNodegroup.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createNodegroup.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "NodegroupName": "MyNodegroup",
        "NodeRole": "arn:aws:iam::123456789012:role/MyNodeInstanceRole",
        "Subnets": ["subnet-09fb51df01234", "subnet-027cfea4b1234"]
      },
      "End": true
    }
  }
}
```

다음에는 노드 그룹을 삭제하는 Task 상태가 포함됩니다.

```
{
  "StartAt": "DeleteNodegroup.sync",
  "States": {
    "DeleteNodegroup.sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteNodegroup.sync",
      "Parameters": {
        "ClusterName": "MyCluster",
        "NodegroupName": "MyNodegroup"
      },
      "End": true
    }
  }
}
```

권한

`eks:createCluster` 서비스 통합을 통해 Amazon EKS 클러스터가 생성될 때 IAM 역할은 관리자(`system:masters` 권한이 있음)로 Kubernetes RBAC 인증 테이블에 추가됩니다. 처음에는 해당 IAM 엔터티만 Kubernetes API 서버를 직접적으로 호출할 수 있습니다. 예를 들어 Step Functions 상태 시스템과 동일한 역할을 가정하거나 권한을 추가 IAM 엔티티에 부여하도록 Kubernetes를 구성하지 않는 한 kubectl을 사용하여 Kubernetes API 서버와 상호 작용할 수 없습니다. 자세한 내용은 Amazon EKS 사용 설명서의 클러스터의 사용자 또는 IAM 역할 관리를 참조하세요.

사용자 또는 역할과 같은 추가 IAM 엔티티를 kube-system 네임스페이스의 `aws-auth ConfigMap`에 추가하여 해당 엔티티에 대한 권한을 추가할 수 있습니다. Step Functions에서 클러스터를 만드는 경우 `eks:call` 서비스 통합을 사용합니다.

다음에는 `aws-auth ConfigMap`을 만들고 `system:masters` 권한을 `arn:aws:iam::123456789012:user/my-user` 사용자와 IAM 역할 `arn:aws:iam::123456789012:role/my-role`에 부여하는 `Task` 상태가 포함됩니다.

```
{
  "StartAt": "Add authorized user",
  "States": {
    "Add authorized user": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:call",
      "Parameters": {
        "ClusterName": "MyCluster",
        "CertificateAuthority": "LS0tLS1CRUd...UtLS0tLQo=",
        "Endpoint": "https://444455556666.yl4.us-east-1.eks.amazonaws.com",
        "Method": "POST",
        "Path": "/api/v1/namespaces/kube-system/configmaps",
        "RequestBody": {
           "apiVersion": "v1",
           "kind": "ConfigMap",
           "metadata": {
              "name": "aws-auth",
              "namespace": "kube-system"
           },
           "data": {
             "mapUsers": "[{ \"userarn\": \"arn:aws:iam::123456789012:user/my-user\",
 \"username\": \"my-user\", \"groups\": [ \"system:masters\" ] } ]",
             "mapRoles": "[{ \"rolearn\": \"arn:aws:iam::123456789012:role/my-role\",
 \"username\": \"my-role\", \"groups\": [ \"system:masters\" ] } ]"
           }
```

```
        }
      },
      "End": true
    }
  }
```

> **Note**
>
> IAM 역할의 ARN이 /service-role/ 경로(예: `arn:aws:iam::123456789012:role/service-role/my-role`)가 포함된 형식으로 표시됩니다. `aws-auth`에 역할을 나열할 때 이 service-role 경로 토큰을 포함해서는 안 됩니다.

클러스터를 처음 만들 때는 `aws-auth ConfigMap`이 존재하지 않지만 Fargate 프로필을 만들면 자동으로 추가됩니다. `aws-auth`의 현재 값을 검색하고 추가 권한을 추가하고, 새 버전을 PUT할 수 있습니다. 일반적으로 Fargate 프로필 전에 `aws-auth`를 만드는 것이 더 쉽습니다.

클러스터가 Step Functions 외부에서 생성된 경우 Kubernetes API 서버와 통신하려면 kubectl을 구성하면 됩니다. 그런 다음 `kubectl apply -f aws-auth.yaml`을 사용하여 새 `aws-auth ConfigMap`을 만들거나 `kubectl edit -n kube-system configmap/aws-auth`를 사용하여 이미 있는 서버를 편집합니다. 자세한 내용은 다음을 참조하세요.

- Amazon EKS 사용 설명서의 Amazon EKS용 kubeconfig 생성
- Amazon EKS 사용 설명서의 클러스터의 사용자 또는 IAM 역할 관리

Kubernetes에 IAM 역할에 대한 충분한 권한이 없으면 `eks:call` 또는 `eks:runJob` 서비스 통합이 실패하고 다음 오류가 발생합니다.

```
Error:
EKS.401

Cause:
{
  "ResponseBody": {
    "kind": "Status",
    "apiVersion": "v1",
    "metadata": {},
    "status": "Failure",
    "message": "Unauthorized",
```

```
    "reason": "Unauthorized",
    "code": 401
  },
  "StatusCode": 401,
  "StatusText": "Unauthorized"
}
```

Step Functions를 사용하여 Amazon EMR 직접 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 아마존 EMR 통합이 아마존 EMR SDK 통합과 다른 점 AWS
>
> 최적화된 Amazon EMR 서비스 통합에는 기본 Amazon EMR API를 래핑하는 사용자 지정된 API 집합이 있습니다. 이 때문에 Amazon AWS EMR SDK 서비스 통합과 크게 다릅니다. 또한 작업 실행(.sync) 통합 패턴이 지원됩니다.

Amazon AWS Step Functions EMR과 통합하려면 제공된 Amazon EMR 서비스 통합 API를 사용합니다. 서비스 통합 API는 해당 Amazon EMR API와 유사하지만 전달되는 필드와 반환되는 응답에 약간의 차이가 있습니다.

Step Functions는 실행이 중지된 경우 Amazon EMR 클러스터를 자동으로 종료하지 않습니다. Amazon EMR 클러스터가 종료되기 전에 상태 시스템이 중지하면 클러스터가 무기한 계속 실행될 수 있으므로 추가 요금이 발생할 수 있습니다. 이를 방지하려면 생성된 모든 Amazon EMR 클러스터를 올바르게 종료해야 합니다. 자세한 내용은 다음을 참조하세요.

- Amazon EMR 사용 설명서의 클러스터 종료 제어
- 서비스 통합 패턴 작업 실행(.sync) 섹션

> Note
>
> `emr-5.28.0`에 따라 클러스터를 만들 때 `StepConcurrencyLevel` 파라미터를 지정하여 단일 클러스터에서 여러 단계를 동시에 실행할 수 있습니다. Step Functions `Map` 및 `Parallel` 상태를 사용하여 동시에 작업을 클러스터에 제출할 수 있습니다.

Amazon EMR 서비스 통합 가용성은 Amazon EMR API 가용성에 따라 달라질 수 있습니다. 특별 리전에서 제한 사항은 Amazon EMR 설명서를 확인하세요.

> **Note**
>
> Amazon EMR과 통합을 위해 Step Functions는 처음 10분과 이후 300초 동안 작업 폴링 빈도를 60초로 하드 코딩합니다.

다음 표에서는 각 서비스 통합 API와 해당 Amazon EMR API 간의 차이점을 설명합니다.

Amazon EMR 서비스 통합 API 및 해당 Amazon EMR API

Amazon EMR 서비스 통합 API	해당 EMR API	차이
createCluster 클러스터(작업 흐름)를 생성하고 실행을 시작합니다. Amazon EMR은 서비스 연결 역할이라고 하는 고유한 유형의 IAM 역할에 직접 연결됩니다. `createCluster` 및 `createCluster.sync` 작업을 수행하려면 서비스 연결 역할인 `AWSServiceRoleForEMRCleanup` 을 생성하도록 구성된 필수 권한이 있어야 합니다. IAM 권한 정책에 추가할 수 있는 문을 포함하여 이에 대한 자세한 내용은 Amazon EMR에 서비스 연결 역할 사용을 참조하세요.	runJobFlow	`createCluster` 다음을 제외하고 와 동일한 요청 구문을 사용합니다. runJobFlow • `Instances.KeepJobFlowAliveWhenNoSteps` 필드는 필수이며 부울 값 `TRUE`가 있어야 합니다. • `Steps` 필드는 허용되지 않습니다. • `Instances.InstanceFleets[index].Name` 필드가 제공되어야 하며 선택적 `modifyInstanceFleetByName` 커넥터 API가 사용되는 경우 고유해야 합니다. • `Instances.InstanceGroups[index].Name` 필드가 제공되어야 하며 선택적 `modifyInstanceGroupByName` API

Amazon EMR 서비스 통합 API	해당 EMR API	차이

		가 사용되는 경우 고유해야 합니다.
		
		응답은 다음과 같습니다.
		
		`{ "ClusterId": "string" }`
		
		Amazon EMR에서 다음을 사용합니다.
		
		`{ "JobFlowId": "string" }`

createCluster.sync 클러스터(작업 흐름)를 생성하고 실행을 시작합니다.	runJobFlow	`createCluster` 와 동일하지만 클러스터가 `WAITING` 상태에 도달할 때까지 기다려야 합니다.

setClusterTermination보호
 	
클러스터의 EC2 인스턴스가 사용자 개입, API 호출 또는 작업 흐름 오류 발생으로 인하여 종료될 수 없도록 클러스터(작업 흐름)를 잠급니다.
	setTerminationProtection	요청은 다음을 사용합니다.
		
		`{ "ClusterId": "string" }`
		
		Amazon EMR에서 다음을 사용합니다.
		
		`{ "JobFlowIds": ["string"] }`

Amazon EMR 서비스 통합 API	해당 EMR API	차이

terminateCluster
 	
클러스터(작업 흐름)를 종료합니다.
	terminateJobFlows	요청은 다음을 사용합니다.
		
		`{ "ClusterId": "string" }`
		
		Amazon EMR에서 다음을 사용합니다.
		
		`{ "JobFlowIds": ["string"] }`

terminateCluster.sync 클러스터(작업 흐름)를 종료합니다.	terminateJobFlows	`terminateCluster` 와 동일하지만 클러스터가 종료될 때까지 기다려야 합니다.

Amazon EMR 서비스 통합 API	해당 EMR API	차이

addStep
 	
실행 중인 클러스터에 새 단계를 추가합니다.
 	
필요한 경우 이 API를 사용하는 중에 `ExecutionRoleArn` 파라미터를 지정할 수도 있습니다.
	addJobFlow단계	요청에서 `"ClusterId"` 키를 사용합니다. Amazon EMR에서 `"JobFlowId"` 를 사용합니다. 요청은 단일 단계를 사용합니다.
		
		`{ "Step": <"StepConfig object"> }`
		
		Amazon EMR에서 다음을 사용합니다.
		
		`{ "Steps": [<StepConfig objects>] }`
		
		응답은 다음과 같습니다.
		
		`{ "StepId": "string" }`
		
		Amazon EMR에서 다음을 반환합니다.
		
		`{ "StepIds": [<strings >] }`

Amazon EMR 서비스 통합 API	해당 EMR API	차이
addStep.sync 실행 중인 클러스터에 새 단계를 추가합니다. 필요한 경우 이 API를 사용하는 중에 `ExecutionRoleArn` 파라미터를 지정할 수도 있습니다.	addJobFlow단계	addStep과 동일하지만 단계가 완료될 때까지 기다려야 합니다.

Amazon EMR 서비스 통합 API	해당 EMR API	차이

cancelStep
 	
실행 중인 클러스터에서 보류 중인 단계를 취소합니다.
	cancelSteps	요청은 다음을 사용합니다.
		
		`{ "StepId": "string" }`
		
		Amazon EMR에서 다음을 사용합니다.
		
		`{ "StepIds": [<strings>] }`
		
		응답은 다음과 같습니다.
		
		`{ "CancelStepsInfo": <CancelStepsInfo object> }`
		
		Amazon EMR에서 다음을 사용합니다.
		
		`{ "CancelStepsInfoList": [<CancelStepsInfo objects>] }`

Amazon EMR 서비스 통합 API	해당 EMR API	차이
modifyInstanceFleetByName 지정된 `InstanceFleetName` 을 사용하여 인스턴스 플릿에 대한 대상 온디맨드 및 대상 스팟 용량을 수정합니다.	modifyInstanceFleet	요청은 다음을 제외하고 `modifyInstanceFleet` 과 동일합니다. • `Instance.InstanceFleetId` 필드는 허용되지 않습니다. • 런타임 시 `InstanceFleetId` 는 `ListInstanceFleets` 를 호출하고 결과를 구문 분석하여 서비스 통합에 의해 자동으로 결정됩니다.

Amazon EMR 서비스 통합 API	해당 EMR API	차이

modifyInstanceGroupByName
 	
인스턴스 그룹의 구성 설정 및 노드 수를 수정합니다.
	modifyInstanceGroups	요청은 다음과 같습니다.
		
		`{ "ClusterId": "string", "InstanceGroup": <InstanceGroupModifyConfig object> }`
		
		Amazon EMR에서 다음 목록을 사용합니다.
		
		`{ "ClusterId": ["string"], "InstanceGroups": [<InstanceGroupModifyConfig objects>] }`
		
		`InstanceGroupModifyConfig` 객체 내에서 `InstanceGroupId` 필드는 허용되지 않습니다.
		
		새 필드 `InstanceGroupName` 이 추가되었습니다. 런타임 시 `InstanceGroupId` 는 `ListInstanceGroups` 를 호출하고 결과를 구문 분석하여 서비스 통합에 의해 자동으로 결정됩니다.

다음은 클러스터를 생성하는 Task 상태를 포함합니다.

```
"Create_Cluster": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:createCluster.sync",
    "Parameters": {
        "Name": "MyWorkflowCluster",
        "VisibleToAllUsers": true,
        "ReleaseLabel": "emr-5.28.0",
        "Applications": [
            {
                "Name": "Hive"
            }
        ],
        "ServiceRole": "EMR_DefaultRole",
        "JobFlowRole": "EMR_EC2_DefaultRole",
        "LogUri": "s3n://aws-logs-123456789012-us-east-1/elasticmapreduce/",
        "Instances": {
            "KeepJobFlowAliveWhenNoSteps": true,
            "InstanceFleets": [
                {
                    "InstanceFleetType": "MASTER",
                    "Name": "MASTER",
                    "TargetOnDemandCapacity": 1,
                    "InstanceTypeConfigs": [
                        {
                            "InstanceType": "m4.xlarge"
                        }
                    ]
                },
                {
                    "InstanceFleetType": "CORE",
                    "Name": "CORE",
                    "TargetOnDemandCapacity": 1,
                    "InstanceTypeConfigs": [
                        {
                            "InstanceType": "m4.xlarge"
                        }
                    ]
                }
            ]
        }
    },
    "End": true
}
```

다음은 종료 보호를 활성화하는 Task 상태를 포함합니다.

```
"Enable_Termination_Protection": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:setClusterTerminationProtection",
    "Parameters": {
        "ClusterId.$": "$.ClusterId",
        "TerminationProtected": true
    },
    "End": true
}
```

다음은 클러스터에 단계를 제출하는 Task 상태를 포함합니다.

```
"Step_One": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:addStep.sync",
    "Parameters": {
        "ClusterId.$": "$.ClusterId",
        "ExecutionRoleArn": "arn:aws:iam::123456789012:role/myEMR-execution-role",
        "Step": {
            "Name": "The first step",
            "ActionOnFailure": "CONTINUE",
            "HadoopJarStep": {
                "Jar": "command-runner.jar",
                "Args": [
                    "hive-script",
                    "--run-hive-script",
                    "--args",
                    "-f",
                    "s3://<region>.elasticmapreduce.samples/cloudfront/code/
Hive_CloudFront.q",
                    "-d",
                    "INPUT=s3://<region>.elasticmapreduce.samples",
                    "-d",
                    "OUTPUT=s3://<mybucket>/MyHiveQueryResults/"
                ]
            }
        }
    },
    "End": true
}
```

다음은 단계를 취소하는 Task 상태를 포함합니다.

```
"Cancel_Step_One": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:cancelStep",
    "Parameters": {
        "ClusterId.$": "$.ClusterId",
        "StepId.$": "$.AddStepsResult.StepId"
    },
    "End": true
}
```

다음은 클러스터를 종료하는 Task 상태를 포함합니다.

```
"Terminate_Cluster": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:terminateCluster.sync",
    "Parameters": {
        "ClusterId.$": "$.ClusterId"
    },
    "End": true
}
```

다음은 인스턴스 그룹에 대해 클러스터를 확장 또는 축소하는 Task 상태를 포함합니다.

```

"ModifyInstanceGroupByName": {
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:modifyInstanceGroupByName",
    "Parameters": {
        "ClusterId": "j-1234567890123",
        "InstanceGroupName": "MyCoreGroup",
        "InstanceGroup": {
            "InstanceCount": 8
        }
    },
    "End": true
}
```

다음은 인스턴스 플릿에 대해 클러스터를 확장 또는 축소하는 Task 상태를 포함합니다.

```
"ModifyInstanceFleetByName": {
```

```
    "Type": "Task",
    "Resource": "arn:aws:states:::elasticmapreduce:modifyInstanceFleetByName",
    "Parameters": {
        "ClusterId": "j-1234567890123",
        "InstanceFleetName": "MyCoreFleet",
        "InstanceFleet": {
            "TargetOnDemandCapacity": 8,
            "TargetSpotCapacity": 0
        }
    },
    "End": true
}
```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

EKS에서 Amazon EMR에 전화하십시오. AWS Step Functions

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

EKS 기반 최적화된 Amazon EMR 통합이 EKS 기반 Amazon EMR SDK 통합과 어떻게 다른가요? AWS

- 작업 실행(.sync) 통합 패턴이 지원됩니다.
- 요청 및 응답 통합 패턴에 대한 최적화는 없습니다.
- 작업 토큰을 사용하여 콜백 대기 통합 패턴은 지원되지 않습니다.

Note

Amazon EMR과 통합을 위해 Step Functions는 처음 10분과 이후 300초 동안 작업 폴링 빈도를 60초로 하드 코딩합니다.

EKS의 Amazon AWS Step Functions EMR과 통합하려면 EKS 기반 Amazon EMR 서비스 통합 API를 사용하십시오. 서비스 통합 API는 해당 Amazon EMR on EKS API와 동일하지만 다음 표와 같이 모든 API에서 모든 통합 패턴을 지원하지 않습니다.

API	요청 및 응답	작업 실행(.sync)
CreateVirtualCluster	✓	
DeleteVirtualCluster	✓	✓
StartJobRun	✓	✓

지원되는 Amazon EMR on EKS API는 다음과 같습니다.

> **Note**
>
> Step Functions에는 작업의 최대 입력 또는 결과 데이터 크기에 대한 할당량이 있습니다. 이를 통해 다른 서비스에 데이터를 보내거나 다른 서비스로부터 데이터를 받을 때 UTF-8 인코딩 문자열로 데이터 256KB까지 제한됩니다. 상태 시스템 실행과 관련된 할당량 섹션을 참조하십시오.

- CreateVirtualCluster
  - 요청 구문
  - 지원되는 파라미터
  - 응답 구문
- DeleteVirtualCluster
  - 요청 구문
  - 지원되는 파라미터
  - 응답 구문
- StartJobRun
  - 요청 구문
  - 지원되는 파라미터
  - 응답 구문

다음에는 가상 클러스터를 만드는 Task 상태가 포함됩니다.

```
"Create_Virtual_Cluster": {
```

```
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-containers:createVirtualCluster",
  "Parameters": {
    "Name": "MyVirtualCluster",
    "ContainerProvider": {
      "Id": "EKSClusterName",
      "Type": "EKS",
      "Info": {
        "EksInfo": {
          "Namespace": "Namespace"
        }
      }
    }
  },
  "End": true
}
```

다음에는 작업을 가상 클러스터에 제출하고 완료될 때까지 기다리는 Task 상태가 포함됩니다.

```
"Submit_Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-containers:startJobRun.sync",
    "Parameters": {
      "Name": "MyJobName",
      "VirtualClusterId.$": "$.VirtualClusterId",
      "ExecutionRoleArn": "arn:aws:iam::<accountId>:role/job-execution-role",
      "ReleaseLabel": "emr-6.2.0-latest",
      "JobDriver": {
        "SparkSubmitJobDriver": {
          "EntryPoint": "s3://<mybucket>/jobs/trip-count.py",
          "EntryPointArguments": [
            "60"
          ],
          "SparkSubmitParameters": "--conf spark.driver.cores=2 --conf
 spark.executor.instances=10 --conf spark.kubernetes.pyspark.pythonVersion=3 --conf
 spark.executor.memory=10G --conf spark.driver.memory=10G --conf spark.executor.cores=1
 --conf spark.dynamicAllocation.enabled=false"
        }
      },
      "ConfigurationOverrides": {
        "ApplicationConfiguration": [
          {
            "Classification": "spark-defaults",
```

```
            "Properties": {
              "spark.executor.instances": "2",
              "spark.executor.memory": "2G"
            }
          }
        ],
        "MonitoringConfiguration": {
          "PersistentAppUI": "ENABLED",
          "CloudWatchMonitoringConfiguration": {
            "LogGroupName": "MyLogGroupName",
            "LogStreamNamePrefix": "MyLogStreamNamePrefix"
          },
          "S3MonitoringConfiguration": {
            "LogUri": "s3://<mylogsbucket>"
          }
        }
      },
      "Tags": {
        "taskType": "jobName"
      }
    },
    "End": true
}
```

다음에는 가상 클러스터를 삭제하고 삭제가 완료될 때까지 기다리는 Task 상태가 포함됩니다.

```
"Delete_Virtual_Cluster": {
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-containers:deleteVirtualCluster.sync",
  "Parameters": {
    "Id.$": "$.VirtualClusterId"
  },
  "End": true
}
```

다른 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. AWS 통합 서비스용 IAM 정책

Step Functions 사용을 통한 Amazon EMR Serverless 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

최적화된 EMR Serverless 통합과 EMR Serverless AWS SDK 통합 간의 차이점

- 최적화 EMR Serverless 서비스 통합에는 기본 EMR Serverless API를 래핑하는 사용자 지정된 API 집합이 있습니다. 이러한 사용자 지정으로 인해 최적화된 EMR Serverless 통합은 EMR Serverless AWS SDK 서비스 통합과 크게 다릅니다. 또한 최적화된 EMR Serverless 통합은 작업 실행(.sync) 통합 패턴을 지원합니다.
- 작업 토큰을 사용하여 콜백 대기 통합 패턴은 지원되지 않습니다.

이 주제의 내용

EMR Serverless 서비스 통합 API

AWS Step Functions를 EMR Serverless와 통합하려면 다음과 같은 EMR Serverless 서비스 통합 API 6개를 사용하면 됩니다. 이러한 서비스 통합 API는 해당 EMR Serverless API와 유사하지만 전달되는 필드와 반환되는 응답에 약간의 차이가 있습니다.

다음 표에는 각 서비스 통합 API와 해당 EMR Serverless API 간의 차이점이 나와 있습니다.

EMR Serverless 서비스 통합 API 및 해당 EMR Serverless API

EMR Serverless 서비스 통합 API	해당 EMR Serverless API	차이
createApplication 애플리케이션을 생성합니다. EMR Serverless는 서비스 연결 역할이라고 하는 고유한 유형의 IAM 역할에 연결됩니다. `createApplication` 및 `createApplication.sync` 작업을 수행하려면 서비스 연결 역할인 `AWS ServiceRoleForAmaz`	CreateApplication	None

EMR Serverless 서비스 통합 API	해당 EMR Serverless API	차이
`onEMRServerless` 을 생성하도록 구성된 필수 권한이 있어야 합니다. IAM 권한 정책에 추가할 수 있는 문을 포함하여 이에 대한 자세한 내용은 EMR Serverless에 서비스 연결 역할 사용을 참조하세요.		
createApplication.sync 애플리케이션을 생성합니다.	CreateApplication	EMR Serverless API와 EMR Serverless 서비스 통합 API의 요청과 응답 간에는 차이가 없습니다. 하지만 createApplication.sync는 애플리케이션이 `CREATED` 상태에 도달할 때까지 기다립니다.
startApplication 지정된 애플리케이션을 시작하고 구성된 경우 애플리케이션의 초기 용량을 초기화합니다.	StartApplication	EMR Serverless API 응답에는 어떠한 데이터도 포함되지 않지만 EMR Serverless 서비스 통합 API 응답에는 다음 데이터가 포함됩니다. `{ "ApplicationId": "string" }`

EMR Serverless 서비스 통합 API	해당 EMR Serverless API	차이
startApplication.sync 지정된 애플리케이션을 시작하고 구성된 경우 초기 용량을 초기화합니다.	StartApplication	EMR Serverless API 응답에는 어떠한 데이터도 포함되지 않지만 EMR Serverless 서비스 통합 API 응답에는 다음 데이터가 포함됩니다. `{ "ApplicationId": "string" }` 또한 startApplication.sync는 애플리케이션이 STARTED 상태에 도달할 때까지 기다립니다.
stopApplication 지정된 애플리케이션을 중지하고 구성된 경우 초기 용량을 해제합니다. 애플리케이션을 중지하기 전에 예약되고 실행 중인 모든 작업을 완료하거나 취소해야 합니다.	StopApplication	EMR Serverless API 응답에는 어떠한 데이터도 포함되지 않지만 EMR Serverless 서비스 통합 API 응답에는 다음 데이터가 포함됩니다. `{ "ApplicationId": "string" }`

EMR Serverless 서비스 통합 API	해당 EMR Serverless API	차이
stopApplication.sync 지정된 애플리케이션을 중지하고 구성된 경우 초기 용량을 해제합니다. 애플리케이션을 중지하기 전에 예약되고 실행 중인 모든 작업을 완료하거나 취소해야 합니다.	StopApplication	EMR Serverless API 응답에는 어떠한 데이터도 포함되지 않지만 EMR Serverless 서비스 통합 API 응답에는 다음 데이터가 포함됩니다. `{ "ApplicationId": "string" }` 또한 stopApplication.sync는 애플리케이션이 STOPPED 상태에 도달할 때까지 기다립니다.
deleteApplication 애플리케이션을 삭제합니다. 애플리케이션을 삭제하려면 애플리케이션이 STOPPED 또는 CREATED 상태여야 합니다.	DeleteApplication	EMR Serverless API 응답에는 어떠한 데이터도 포함되지 않지만 EMR Serverless 서비스 통합 API 응답에는 다음 데이터가 포함됩니다. `{ "ApplicationId": "string" }`

EMR Serverless 서비스 통합 API	해당 EMR Serverless API	차이
deleteApplication.sync 애플리케이션을 삭제합니다. 애플리케이션을 삭제하려면 애플리케이션이 `STOPPED` 또는 `CREATED` 상태여야 합니다.	DeleteApplication	EMR Serverless API 응답에는 어떠한 데이터도 포함되지 않지만 EMR Serverless 서비스 통합 API 응답에는 다음 데이터가 포함됩니다. `{ "ApplicationId": "string" }` 또한 stopApplication.sync는 애플리케이션이 `TERMINATED` 상태에 도달할 때까지 기다립니다.
startJobRun 작업 실행을 시작합니다.	StartJobRun	None
startJobRun.sync 작업 실행을 시작합니다.	StartJobRun	EMR Serverless API와 EMR Serverless 서비스 통합 API의 요청과 응답 간에는 차이가 없습니다. startJobRun그러나.sync는 애플리케이션이 상태에 도달할 때까지 기다립니다. `SUCCESS`
cancelJobRun 작업 실행을 취소합니다.	CancelJobRun	None

EMR Serverless 서비스 통합 API	해당 EMR Serverless API	차이
cancelJobRun.sync 작업 실행을 취소합니다.	CancelJobRun	EMR Serverless API와 EMR Serverless 서비스 통합 API의 요청과 응답 간에는 차이가 없습니다. cancelJobRun그러나.sync는 애플리케이션이 상태에 도달할 때까지 기다립니다. `CANCELLED`

EMR 서버리스 통합 사용 사례

최적화된 EMR Serverless 서비스 통합의 경우 단일 애플리케이션을 만든 다음 해당 애플리케이션을 사용하여 여러 작업을 실행하는 것이 좋습니다. 예를 들어 단일 상태 머신에 동일한 애플리케이션을 사용하는 여러 startJobRun요청을 포함할 수 있습니다. 다음 태스크 상태 상태 예제에서는 Step Functions를 사용하여 EMR Serverless API를 통합하는 사용 사례를 보여줍니다. 다른 EMR Serverless 사용 사례는 Amazon EMR Serverless란 무엇입니까? 참조하세요.

> **Tip**
>
> 여러 작업을 실행하기 위해 통합되는 상태 머신의 예를 사용자 AWS 계정컴퓨터에 배포하려면을 참조하십시오EMR Serverless 작업 실행. EMR Serverless

- 애플리케이션 생성
- 애플리케이션 시작
- 애플리케이션 중지
- 애플리케이션을 삭제합니다
- 애플리케이션에서 작업 시작
- 애플리케이션에서 작업 취소

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은을 참조하십시오통합 서비스용 IAM 정책.

다음 사용 사례에 표시된 예제에서 *####* 텍스트를 리소스별 정보로 바꿉니다. 예를 들어 EMR Serverless 애플리케이션의 ID (예:) *yourApplicationId*로 바꾸십시오00yv7iv71inak893.

애플리케이션 생성

다음 Task 상태 예제는 createApplication.sync 서비스 통합 API를 사용하여 애플리케이션을 만듭니다.

```
"Create_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:createApplication.sync",
    "Parameters": {
        "Name": "MyApplication",
        "ReleaseLabel": "emr-6.9.0",
        "Type": "SPARK"
    },
    "End": true
}
```

애플리케이션 시작

다음 Task 상태 예제는 startApplication.sync 서비스 통합 API를 사용하여 애플리케이션을 시작합니다.

```
"Start_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:startApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
    "End": true
}
```

애플리케이션 중지

다음 Task 상태 예제는 stopApplication.sync 서비스 통합 API를 사용하여 애플리케이션을 중지합니다.

```
"Stop_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:stopApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
```

```
    "End": true
}
```

애플리케이션을 삭제합니다

다음 Task 상태 예제는 deleteApplication.sync 서비스 통합 API를 사용하여 애플리케이션을 삭제합니다.

```
"Delete_Application": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:deleteApplication.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId"
    },
    "End": true
}
```

애플리케이션에서 작업 시작

다음 작업 상태 startJobRun예제는.sync 서비스 통합 API를 사용하여 애플리케이션에서 작업을 시작합니다.

```
"Start_Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:startJobRun.sync",
    "Parameters": {
        "ApplicationId": "yourApplicationId",
        "ExecutionRoleArn": "arn:aws:iam::123456789012:role/myEMRServerless-execution-
role",
        "JobDriver": {
            "SparkSubmit": {
                "EntryPoint": "s3://<mybucket>/sample.py",
                "EntryPointArguments": ["1"],
                "SparkSubmitParameters": "--conf spark.executor.cores=4 --conf
 spark.executor.memory=4g --conf spark.driver.cores=2 --conf spark.driver.memory=4g --
conf spark.executor.instances=1"
            }
        }
    },
    "End": true
}
```

애플리케이션에서 작업 취소

다음 작업 상태 cancelJobRun예제는.sync 서비스 통합 API를 사용하는 애플리케이션에서 작업을 취소합니다.

```
"Cancel_Job": {
    "Type": "Task",
    "Resource": "arn:aws:states:::emr-serverless:cancelJobRun.sync",
    "Parameters": {
        "ApplicationId.$": "$.ApplicationId",
        "JobRunId.$": "$.JobRunId"
    },
    "End": true
}
```

EventBridge Step 함수를 사용한 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 EventBridge 통합과 EventBridge AWS SDK 통합의 차이점
>
> - 실행 ARN 및 상태 시스템 ARN은 각 `PutEventsRequestEntry`의 `Resources` 필드에 자동으로 추가됩니다.
> - `PutEvents`의 응답에 0이 아닌 `FailedEntryCount`가 포함된 경우 `Task` 상태가 실패하고 `EventBridge.FailedEntry` 오류가 발생합니다.

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은을 참조하십시오통합 서비스용 IAM 정책.

Step Functions는 EventBridge 아마존과의 통합을 위한 서비스 통합 API를 제공합니다. 이를 통해 Step Functions 워크플로에서 직접 사용자 지정 이벤트를 전송하여 이벤트 기반 애플리케이션을 빌드할 수 있습니다.

`PutEvents`API를 사용하려면 전송할 이벤트의 특정 패턴과 일치하는 EventBridge 규칙을 계정에 생성해야 합니다. 예를 들면, 다음과 같이 할 수 있습니다.

- 규칙과 일치하는 이벤트를 수신하고 인쇄하는 Lambda 함수를 계정에 생성하십시오. EventBridge

- 특정 이벤트 패턴과 일치하고 Lambda 함수를 대상으로 하는 기본 이벤트 버스의 계정에 EventBridge 규칙을 생성합니다.

자세한 내용은 다음을 참조하세요.

- EventBridge 사용 설명서에 Amazon EventBridge 이벤트 추가. PutEvents
- 서비스 통합 패턴의 작업 토큰을 사용하여 콜백 대기

> **Note**
>
> Step Functions에는 작업의 최대 입력 또는 결과 데이터 크기에 대한 할당량이 있습니다. 이를 통해 다른 서비스에 데이터를 보내거나 다른 서비스로부터 데이터를 받을 때 UTF-8 인코딩 문자열로 데이터 256KB까지 제한됩니다. 상태 시스템 실행과 관련된 할당량 섹션을 참조하십시오.

지원되는 EventBridge API

지원되는 EventBridge API 및 구문은 다음과 같습니다.

- PutEvents
  - 요청 구문
  - 지원되는 파라미터는 다음과 같습니다.
    - Entries
  - 응답 구문

다음에는 사용자 지정 이벤트를 전송하는 Task가 포함됩니다.

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::events:putEvents",
  "Parameters": {
    "Entries": [
      {
        "Detail": {
          "Message": "MyMessage"
        },
```

```
        "DetailType": "MyDetailType",
        "EventBusName": "MyEventBus",
        "Source": "my.source"
      }
    ]
  },
  "End": true
}
```

오류 처리

PutEvents API는 항목 배열을 입력으로 허용한 다음 결과 항목 배열을 반환합니다. PutEvents 작업이 성공하면 항목이 하나 이상 실패하더라도 PutEvents에서 HTTP 200 응답을 반환합니다. PutEvents에서 FailedEntryCount 필드에 실패한 항목 수를 반환합니다.

Step Functions는 FailedEntryCount가 0보다 큰지 여부를 확인합니다. 값이 0보다 크면 Step Functions에서 상태가 실패하고 EventBridge.FailedEntry 오류가 발생합니다. 이를 통해 실패한 항목이 있을 때 추가 상태를 사용하여 응답의 FailedEntryCount를 분석할 필요 없이 Task 상태에서 Step Functions의 기본 제공 오류 처리 기능을 사용하여 이 항목을 포착하거나 재시도할 수 있습니다.

> **Note**
>
> 멱등성을 구현했고 모든 항목을 안전하게 재시도할 수 있는 경우 Step Functions의 재시도 로직을 사용할 수 있습니다. Step Functions는 재시도하기 전에 PutEvents 입력 배열에서 성공한 항목을 제거하지 않습니다. 대신 원래 항목 배열을 사용하여 재시도합니다.

Step Functions를 사용한 AWS Glue 작업 관리

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> **최적화된 AWS Glue 통합과 AWS GlueAWS SDK 통합의 차이점**
>
> - 작업 실행(.sync) 통합 패턴을 사용할 수 있습니다.
> - JobName 필드는 요청에서 추출되어 응답에 삽입됩니다. 응답에는 보통 JobRunID만 포함됩니다.

지원되는 AWS Glue API:

- StartJobRun

> 의 매개변수는 Step Functions 다음과 같이 표현됩니다. PascalCase
> 네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

다음은 작업을 시작하는 `Task` 상태를 포함합니다. AWS Glue

```
"Glue StartJobRun": {
      "Type": "Task",
      "Resource": "arn:aws:states:::glue:startJobRun.sync",
      "Parameters": {
        "JobName": "GlueJob-JTrRO5l98qMG"
      },
      "Next": "ValidateOutput"
    },
```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Step Functions를 사용한 AWS Glue DataBrew 작업 관리

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

DataBrew 통합을 사용하여 분석 및 기계 학습 워크플로에 데이터 정리 및 데이터 정규화 단계를 추가할 수 있습니다.

지원되는 DataBrew API:

- StartJobRun

다음은 `Task` DataBrew 요청-응답 작업을 시작하는 상태를 포함합니다.

```
"DataBrew StartJobRun": {
            "Type": "Task",
```

```
            "Resource": "arn:aws:states:::databrew:startJobRun",
            "Parameters": {
               "Name": "sample-proj-job-1"
            },
            "Next": "NEXT_STATE"
         },
```

다음은 동기화 작업을 시작하는 Task 상태를 포함합니다. DataBrew

```
"DataBrew StartJobRun": {
           "Type": "Task",
           "Resource": "arn:aws:states:::databrew:startJobRun.sync",
           "Parameters": {
              "Name": "sample-proj-job-1"
           },
           "Next": "NEXT_STATE"
         },
```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Step Functions를 사용하여 Lambda 간접 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 Lambda 통합이 Lambda SDK 통합과 다른 점 AWS
>
> - 응답 Payload 필드는 이스케이프된 Json에서 Json으로 파싱됩니다.
> - 응답에 FunctionError 필드가 포함되어 있거나 Lambda 함수 내에서 예외가 발생하면 작업이 실패합니다.

상태 입력, 출력, 결과 관리에 대한 자세한 내용은 Step Functions에서 입력 및 출력 처리 단원을 참조하십시오.

지원되는 API: AWS Lambda

- Invoke

- 요청 구문
- 지원되는 파라미터
  - ClientContext
  - FunctionName
  - InvocationType
  - Qualifier
  - Payload
- 응답 구문

> 의 매개변수는 Step Functions 다음과 같이 표현됩니다. PascalCase
>
> 네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

다음에는 Lambda 함수를 간접적으로 호출하는 `Task` 상태가 포함됩니다.

```
{
   "StartAt":"CallLambda",
   "States":{
      "CallLambda":{
         "Type":"Task",
         "Resource":"arn:aws:states:::lambda:invoke",
         "Parameters":{
            "FunctionName":"arn:aws:lambda:us-east-1:123456789012:function:MyFunction"
         },
         "End":true
      }
   }
}
```

다음에는 콜백 서비스 통합 패턴을 구현하는 `Task` 상태가 포함됩니다.

```
{
   "StartAt":"GetManualReview",
   "States":{
      "GetManualReview":{
         "Type":"Task",
```

```
         "Resource":"arn:aws:states:::lambda:invoke.waitForTaskToken",
         "Parameters":{
            "FunctionName":"arn:aws:lambda:us-east-1:123456789012:function:get-model-review-decision",
            "Payload":{
               "model.$":"$.new_model",
               "token.$":"$$.Task.Token"
            },
            "Qualifier":"prod-v1"
         },
         "End":true
      }
   }
}
```

Lambda 함수를 간접적으로 호출하면 함수가 완료될 때까지 실행이 대기합니다. 콜백 작업으로 Lambda 함수를 간접적으로 호출하면 Lambda 함수가 실행을 완료하고 결과를 반환할 때까지 하트비트 제한 시간이 계산되지 않습니다. Lambda 함수가 실행되는 동안에는 하트비트 제한 시간이 적용되지 않습니다.

다음 예제와 같이 `InvocationType` 파라미터를 사용하여 Lambda를 비동기적으로 호출할 수도 있습니다.

> **Note**
>
> Lambda 함수의 비동기 호출의 경우 하트비트 제한 시간이 즉시 시작됩니다.

```
{

  "Comment": "A Hello World example of the Amazon States Language using Pass states",
  "StartAt": "Hello",
  "States": {
    "Hello": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-1:123456789012:function:echo",
        "InvocationType": "Event"
      },
      "End": true
    }
```

```
  }
}
```

Task 결과가 반환되면 함수 출력이 메타데이터 사전 내에서 중첩됩니다. 예:

```
{

   "ExecutedVersion":"$LATEST",
   "Payload":"FUNCTION OUTPUT",
   "SdkHttpMetadata":{
      "HttpHeaders":{
         "Connection":"keep-alive",
         "Content-Length":"4",
         "Content-Type":"application/json",
         "Date":"Fri, 26 Mar 2021 07:42:02 GMT",
         "X-Amz-Executed-Version":"$LATEST",
         "x-amzn-Remapped-Content-Length":"0",
         "x-amzn-RequestId":"0101aa0101-1111-111a-aa55-1010aaa1010",
         "X-Amzn-Trace-Id":"root=1-1a1a000a2a2-fe0101aa10ab;sampled=0"
      },
      "HttpStatusCode":200
   },
   "SdkResponseMetadata":{
      "RequestId":"6b3bebdb-9251-453a-ae45-512d9e2bf4d3"
   },
   "StatusCode":200
}
```

또는 함수 ARN을 "리소스" 필드에 직접 지정하여 Lambda 함수를 간접적으로 호출할 수 있습니다. 이런 방식으로 Lambda 함수를 간접적으로 호출할 때는 `.waitForTaskToken`을 지정할 수 없으며 작업 결과에 함수 출력만 포함됩니다.

```
{
   "StartAt":"CallFunction",
   "States":{
      "CallFunction": {
         "Type":"Task",
         "Resource":"arn:aws:lambda:us-east-1:123456789012:function:HelloFunction",
         "End": true
      }
   }
}
```

`Resource` 필드의 ARN에서 해당 옵션을 지정하여 특정 Lambda 함수 버전 또는 별칭을 간접적으로 호출할 수 있습니다. Lambda 설명서에서 다음을 참조하세요.

- AWS Lambda 버전 관리
- AWS Lambda 별칭

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Step AWS Elemental MediaConvert Functions를 사용한 관리

> Step Functions를 사용해 실험하고 MediaConvert
>
> 비디오 클립의 시작 부분에서 길이를 알 수 없는 SMTPE 색상 막대를 감지하여 제거하는 워크플로우에서 MediaConvert 최적화된 통합을 사용하는 방법을 알아보십시오. 2024년 4월 12일자 블로그 게시물 읽기: 다음을 포함한 로우 코드 워크플로우 AWS Elemental MediaConvert

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 통합이 표준 AWS SDK 통합과 다른 점
>
> - 작업 실행(.sync) 통합 패턴을 사용할 수 있습니다.
> - 최적화 요청 및 응답 또는 작업 토큰을 사용하여 콜백 대기 통합 패턴은 없습니다.

지원되는 API: MediaConvert

- `CreateJob`
  - 요청 구문
  - 지원되는 파라미터:
    - `Role`(필수)
    - `Settings`(필수)
    - `CreateJobRequest` (선택 사항)
  - 응답 구문 — 스키마 참조 CreateJobResponse

다음은 작업을 제출하고 MediaConvert 작업이 완료될 때까지 기다리는 Task 상태를 포함합니다.

```
{
    "StartAt": "MediaConvert_CreateJob",
    "States": {
        "MediaConvert_CreateJob": {
        "Type": "Task",
        "Resource": "arn:aws:states:::mediaconvert:createJob.sync",
        "Parameters": {
            "Role": "arn:aws:iam::111122223333:role/Admin",
            "Settings": {
            "OutputGroups": [
                {
                "Outputs": [
                    {
                    "ContainerSettings": {
                        "Container": "MP4"
                    },
                    "VideoDescription": {
                        "CodecSettings": {
                        "Codec": "H_264",
                        "H264Settings": {
                            "MaxBitrate": 1000,
                            "RateControlMode": "QVBR",
                            "SceneChangeDetect": "TRANSITION_DETECTION"
                        }
                        }
                    },
                    "AudioDescriptions": [
                        {
                        "CodecSettings": {
                            "Codec": "AAC",
                            "AacSettings": {
                            "Bitrate": 96000,
                            "CodingMode": "CODING_MODE_2_0",
                            "SampleRate": 48000
                            }
                        }
                        }
                    ]
                    }
                ],
                "OutputGroupSettings": {
                    "Type": "FILE_GROUP_SETTINGS",
```

```
                    "FileGroupSettings": {
                    "Destination": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/"
                    }
                }
                }
            ],
            "Inputs": [
                {
                "AudioSelectors": {
                    "Audio Selector 1": {
                    "DefaultSelection": "DEFAULT"
                    }
                },
                "FileInput": "s3://DOC-EXAMPLE-SOURCE-BUCKET/DOC-EXAMPLE-SOURCE_FILE"
                }
            ]
            }
        },
        "End": true
        }
    }
}
```

Step Functions와 함께 MediaConvert 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. IAM 정책은 다음과 같습니다. AWS Elemental MediaConvert

> 의 매개변수는 Step Functions 다음과 같이 표현됩니다. PascalCase
>
> 네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

Step SageMaker Functions를 사용한 관리

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 SageMaker 통합과 SageMaker AWS SDK 통합의 차이점
>
> - 작업 실행(.sync) 통합 패턴이 지원됩니다.
> - 요청 및 응답 통합 패턴에 대한 최적화는 없습니다.

- 작업 토큰을 사용하여 콜백 대기 통합 패턴은 지원되지 않습니다.

지원되는 SageMaker API 및 구문:

- CreateEndpoint
  - 요청 구문
  - 지원되는 파라미터:
    - EndpointConfigName
    - EndpointName
    - Tags
  - 응답 구문
- CreateEndpointConfig
  - 요청 구문
  - 지원되는 파라미터:
    - EndpointConfigName
    - KmsKeyId
    - ProductionVariants
    - Tags
  - 응답 구문
- CreateHyperParameterTuningJob

> **Note**
>
> 이 API 작업은 .sync 통합 패턴을 지원합니다.

  - 요청 구문
  - 지원되는 파라미터:
    - HyperParameterTuningJobConfig
    - HyperParameterTuningJobName
    - Tags
    - TrainingJobDefinition

    - WarmStartConfig
  - 응답 구문
- CreateLabelingJob

> **Note**
>
> 이 API 작업은 .sync 통합 패턴을 지원합니다.

  - 요청 구문
  - 지원되는 파라미터:
    - HumanTaskConfig
    - InputConfig
    - LabelAttributeName
    - LabelCategoryConfigS3Uri
    - LabelingJobAlgorithmsConfig
    - LabelingJobName
    - OutputConfig
    - RoleArn
    - StoppingConditions
    - Tags
  - 응답 구문
- CreateModel
  - 요청 구문
  - 지원되는 파라미터:
    - Containers
    - EnableNetworkIsolation
    - ExecutionRoleArn
    - ModelName
    - PrimaryContainer
    - Tags
    - VpcConfig

- CreateProcessingJob

> **Note**
>
> 이 API 작업은 .sync 통합 패턴을 지원합니다.

  - 요청 구문
  - 지원되는 파라미터:
    - AppSpecification
    - Environment
    - ExperimentConfig
    - NetworkConfig
    - ProcessingInputs
    - ProcessingJobName
    - ProcessingOutputConfig
    - ProcessingResources
    - RoleArn
    - StoppingCondition
    - Tags
  - 응답 구문
- CreateTrainingJob

> **Note**
>
> 이 API 작업은 .sync 통합 패턴을 지원합니다.

  - 요청 구문
  - 지원되는 파라미터:
    - AlgorithmSpecification
    - HyperParameters
    - InputDataConfig

    - OutputDataConfig
    - ResourceConfig
    - RoleArn
    - StoppingCondition
    - Tags
    - TrainingJobName
    - VpcConfig
  - 응답 구문
- CreateTransformJob

> **Note**
>
> 이 API 작업은 .sync 통합 패턴을 지원합니다.

> **Note**
>
> AWS Step Functions 에 대한 CreateTransformJob 정책을 자동으로 생성하지 않습니다. 생성된 역할에는 인라인 정책을 연결해야 합니다. 자세한 내용은 이 예제 IAM 정책(CreateTrainingJob)을 참조하세요.

  - 요청 구문
  - 지원되는 파라미터:
    - BatchStrategy
    - Environment
    - MaxConcurrentTransforms
    - MaxPayloadInMB
    - ModelName
    - Tags
    - TransformInput
    - TransformJobName
    - TransformOutput

    - TransformResources
  - 응답 구문
- UpdateEndpoint
  - 요청 구문
  - 지원되는 파라미터:
    - EndpointConfigName
    - EndpointName
  - 응답 구문

SageMaker Transform Job 예제

다음은 DataSource 및 에 대한 Amazon S3 위치를 지정하는 Amazon SageMaker 변환 작업을 생성하는 Task 상태를 포함합니다TransformOutput.

```
{
"SageMaker CreateTransformJob": {
  "Type": "Task",
  "Resource": "arn:aws:states:::sagemaker:createTransformJob.sync",
  "Parameters": {
    "ModelName": "SageMakerCreateTransformJobModel-9iFBKsYti9vr",
    "TransformInput": {
      "CompressionType": "None",
      "ContentType": "text/csv",
      "DataSource": {
        "S3DataSource": {
          "S3DataType": "S3Prefix",
          "S3Uri": "s3://my-s3bucket-example-1/TransformJobDataInput.txt"
        }
      }
    },
    "TransformOutput": {
      "S3OutputPath": "s3://my-s3bucket-example-1/TransformJobOutputPath"
    },
    "TransformResources": {
      "InstanceCount": 1,
      "InstanceType": "ml.m4.xlarge"
    },
    "TransformJobName": "sfn-binary-classification-prediction"
  },
```

```
  "Next": "ValidateOutput"
},
```

SageMaker 교육 Job 예제

다음은 Amazon SageMaker 교육 작업을 생성하는 Task 상태를 포함합니다.

```
{
   "SageMaker CreateTrainingJob":{
      "Type":"Task",
      "Resource":"arn:aws:states:::sagemaker:createTrainingJob.sync",
      "Parameters":{
         "TrainingJobName":"search-model",
         "ResourceConfig":{
            "InstanceCount":4,
            "InstanceType":"ml.c4.8xlarge",
            "VolumeSizeInGB":20
         },
         "HyperParameters":{
            "mode":"batch_skipgram",
            "epochs":"5",
            "min_count":"5",
            "sampling_threshold":"0.0001",
            "learning_rate":"0.025",
            "window_size":"5",
            "vector_dim":"300",
            "negative_samples":"5",
            "batch_size":"11"
         },
         "AlgorithmSpecification":{
            "TrainingImage":"...",
            "TrainingInputMode":"File"
         },
         "OutputDataConfig":{
            "S3OutputPath":"s3://bucket-name/doc-search/model"
         },
         "StoppingCondition":{
            "MaxRuntimeInSeconds":100000
         },
         "RoleArn":"arn:aws:iam::123456789012:role/docsearch-stepfunction-iam-role",
         "InputDataConfig":[
            {
               "ChannelName":"train",
```

```
            "DataSource":{
               "S3DataSource":{
                  "S3DataType":"S3Prefix",
                  "S3Uri":"s3://bucket-name/doc-search/interim-data/training-data/",
                  "S3DataDistributionType":"FullyReplicated"
               }
            }
         }
      ]
   },
   "Retry":[
      {
         "ErrorEquals":[
            "SageMaker.AmazonSageMakerException"
         ],
         "IntervalSeconds":1,
         "MaxAttempts":100,
         "BackoffRate":1.1
      },
      {
         "ErrorEquals":[
            "SageMaker.ResourceLimitExceededException"
         ],
         "IntervalSeconds":60,
         "MaxAttempts":5000,
         "BackoffRate":1
      },
      {
         "ErrorEquals":[
            "States.Timeout"
         ],
         "IntervalSeconds":1,
         "MaxAttempts":5,
         "BackoffRate":1
      }
   ],
   "Catch":[
      {
         "ErrorEquals":[
            "States.ALL"
         ],
         "ResultPath":"$.cause",
         "Next":"Sagemaker Training Job Error"
      }
```

```
      ],
      "Next":"Delete Interim Data Job"
   }
}
```

SageMaker 라벨링 작업 예제

다음은 Amazon SageMaker 라벨 제작 작업을 생성하는 Task 상태를 포함합니다.

```
{
  "StartAt": "SageMaker CreateLabelingJob",
  "TimeoutSeconds": 3600,
  "States": {
    "SageMaker CreateLabelingJob": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sagemaker:createLabelingJob.sync",
      "Parameters": {
        "HumanTaskConfig": {
          "AnnotationConsolidationConfig": {
            "AnnotationConsolidationLambdaArn": "arn:aws:lambda:us-
west-2:123456789012:function:ACS-TextMultiClass"
          },
          "NumberOfHumanWorkersPerDataObject": 1,
          "PreHumanTaskLambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:PRE-
TextMultiClass",
          "TaskDescription": "Classify the following text",
          "TaskKeywords": [
            "tc",
            "Labeling"
          ],
          "TaskTimeLimitInSeconds": 300,
          "TaskTitle": "Classify short bits of text",
          "UiConfig": {
            "UiTemplateS3Uri": "s3://s3bucket-example/TextClassification.template"
          },
          "WorkteamArn": "arn:aws:sagemaker:us-west-2:123456789012:workteam/private-
crowd/ExampleTesting"
        },
        "InputConfig": {
          "DataAttributes": {
            "ContentClassifiers": [
```

```
              "FreeOfPersonallyIdentifiableInformation",
              "FreeOfAdultContent"
            ]
          },
          "DataSource": {
            "S3DataSource": {
              "ManifestS3Uri": "s3://s3bucket-example/manifest.json"
            }
          }
        },
        "LabelAttributeName": "Categories",
        "LabelCategoryConfigS3Uri": "s3://s3bucket-example/labelcategories.json",
        "LabelingJobName": "example-job-name",
        "OutputConfig": {
          "S3OutputPath": "s3://s3bucket-example/output"
        },
        "RoleArn": "arn:aws:iam::123456789012:role/service-role/AmazonSageMaker-
ExecutionRole",
        "StoppingConditions": {
          "MaxHumanLabeledObjectCount": 10000,
          "MaxPercentageOfInputDatasetLabeled": 100
        }
      },
      "Next": "ValidateOutput"
    },
    "ValidateOutput": {
      "Type": "Choice",
      "Choices": [
        {
          "Not": {
            "Variable": "$.LabelingJobArn",
            "StringEquals": ""
          },
          "Next": "Succeed"
        }
      ],
      "Default": "Fail"
    },
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail",
      "Error": "InvalidOutput",
```

```
      "Cause": "Output is not what was expected. This could be due to a service outage
 or a misconfigured service integration."
    }
  }
}
```

SageMaker 처리 작업 예제

다음은 Amazon SageMaker 처리 작업을 생성하는 Task 상태를 포함합니다.

```
{
  "StartAt": "SageMaker CreateProcessingJob Sync",
  "TimeoutSeconds": 3600,
  "States": {
    "SageMaker CreateProcessingJob Sync": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sagemaker:createProcessingJob.sync",
      "Parameters": {
        "AppSpecification": {
          "ImageUri": "737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-scikit-
learn:0.20.0-cpu-py3"
        },
        "ProcessingResources": {
          "ClusterConfig": {
            "InstanceCount": 1,
            "InstanceType": "ml.t3.medium",
            "VolumeSizeInGB": 10
          }
        },
        "RoleArn": "arn:aws:iam::123456789012:role/SM-003-
CreateProcessingJobAPIExecutionRole",
        "ProcessingJobName.$": "$.id"
      },
      "Next": "ValidateOutput"
    },
    "ValidateOutput": {
      "Type": "Choice",
      "Choices": [
        {
          "Not": {
            "Variable": "$.ProcessingJobArn",
            "StringEquals": ""
          },
```

```
          "Next": "Succeed"
        }
      ],
      "Default": "Fail"
    },
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail",
      "Error": "InvalidConnectorOutput",
      "Cause": "Connector output is not what was expected. This could be due to a
 service outage or a misconfigured connector."
    }
  }
}
```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Step Functions를 사용하여 Amazon SNS 직접 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 아마존 SNS 통합이 아마존 SNS AWS SDK 통합과 어떻게 다른가
> 요청 및 응답 또는 작업 토큰을 사용하여 콜백 대기 통합 패턴에 대한 최적화는 없습니다.

지원되는 Amazon SNS API는 다음과 같습니다.

> **Note**
>
> Step Functions에는 작업의 최대 입력 또는 결과 데이터 크기에 대한 할당량이 있습니다. 이를 통해 다른 서비스에 데이터를 보내거나 다른 서비스로부터 데이터를 받을 때 UTF-8 인코딩 문자열로 데이터 256KB까지 제한됩니다. 상태 시스템 실행과 관련된 할당량 섹션을 참조하십시오.

- Publish

- 요청 구문
- 지원되는 파라미터
  - Message
  - MessageAttributes
  - MessageStructure
  - PhoneNumber
  - Subject
  - TargetArn
  - TopicArn
- 응답 구문

> 의 파라미터는 Step Functions 다음과 같이 표현됩니다. PascalCase
>
> 네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

다음에는 Amazon Simple Notification Service(SNS) 주제에 게시하는 `Task` 상태가 포함됩니다.

```
{
 "StartAt": "Publish to SNS",
 "States": {
   "Publish to SNS": {
     "Type": "Task",
     "Resource": "arn:aws:states:::sns:publish",
     "Parameters": {
       "TopicArn": "arn:aws:sns:us-east-1:123456789012:myTopic",
       "Message.$": "$.input.message",
       "MessageAttributes": {
         "my_attribute_no_1": {
           "DataType": "String",
           "StringValue": "value of my_attribute_no_1"
         },
         "my_attribute_no_2": {
           "DataType": "String",
           "StringValue": "value of my_attribute_no_2"
         }
```

```
        }
      },
      "End": true
    }
  }
}
```

동적 값 전달. 위 예제를 수정하여 이 JSON 페이로드의 속성을 동적으로 전달할 수 있습니다.

```
{
"input": {
  "message": "Hello world"
},
"SNSDetails": {
  "attribute1": "some value",
  "attribute2": "some other value",
}
}
```

`.$`를 `StringValue` 필드에 추가합니다.

```
"MessageAttributes": {
  "my_attribute_no_1": {
      "DataType": "String",
      "StringValue.$": "$.SNSDetails.attribute1"
  },
  "my_attribute_no_2": {
      "DataType": "String",
      "StringValue.$": "$.SNSDetails.attribute2"
  }
```

다음에는 Amazon SNS 주제에 게시된 다음 작업 토큰이 반환될 때까지 기다리는 `Task` 상태가 포함됩니다. 작업 토큰을 사용하여 콜백 대기 섹션을 참조하십시오.

```
{
   "StartAt":"Send message to SNS",
   "States":{
      "Send message to SNS":{
         "Type":"Task",
         "Resource":"arn:aws:states:::sns:publish.waitForTaskToken",
         "Parameters":{
            "TopicArn":"arn:aws:sns:us-east-1:123456789012:myTopic",
```

```
            "Message":{
               "Input.$":"$",
               "TaskToken.$":"$$.Task.Token"
            }
         },
         "End":true
      }
   }
}
```

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 통합 서비스용 IAM 정책

Step Functions를 사용하여 Amazon SQS 직접 호출

Step Functions는 Amazon States Language (ASL) 에서 직접 특정 AWS 서비스를 제어할 수 있습니다. 자세한 내용은 다른 서비스와 함께 사용 및 파라미터를 서비스 API에 전달 섹션을 참조하세요.

> 최적화된 Amazon SQS 통합이 Amazon SQS SDK 통합과 다른 점 AWS
> 요청 및 응답 또는 작업 토큰을 사용하여 콜백 대기 통합 패턴에 대한 최적화는 없습니다.

지원되는 Amazon SQS API는 다음과 같습니다.

> **Note**
>
> Step Functions에는 작업의 최대 입력 또는 결과 데이터 크기에 대한 할당량이 있습니다. 이를 통해 다른 서비스에 데이터를 보내거나 다른 서비스로부터 데이터를 받을 때 UTF-8 인코딩 문자열로 데이터 256KB까지 제한됩니다. 상태 시스템 실행과 관련된 할당량 섹션을 참조하십시오.

- `SendMessage`

  지원되는 파라미터:

  - `DelaySeconds`
  - `MessageAttribute`
  - `MessageBody`

- MessageDeduplicationId
- MessageGroupId
- QueueUrl
- Response syntax

> 의 파라미터는 다음과 같이 표현됩니다. Step Functions PascalCase
>
> 네이티브 서비스 API가 CamelCase에 있는 경우에도 (예: API `startSyncExecution` 작업) 다음과 같은 매개변수를 PascalCase 지정합니다. `StateMachineArn`

다음에는 Amazon Simple Queue Service(Amazon SQS) 메시지를 전송하는 `Task` 상태가 포함됩니다.

```
{
 "StartAt": "Send to SQS",
 "States": {
   "Send to SQS": {
     "Type": "Task",
     "Resource": "arn:aws:states:::sqs:sendMessage",
     "Parameters": {
       "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/myQueue",
       "MessageBody.$": "$.input.message",
       "MessageAttributes": {
         "my_attribute_no_1": {
           "DataType": "String",
           "StringValue": "attribute1"
         },
         "my_attribute_no_2": {
           "DataType": "String",
           "StringValue": "attribute2"
         }
       }
     },
     "End": true
    }
  }
}
```

다음에는 Amazon SQS 대기열에 게시한 다음 작업 토큰이 반환될 때까지 기다리는 `Task` 상태가 포함됩니다. 작업 토큰을 사용하여 콜백 대기 섹션을 참조하십시오.

```
{
   "StartAt":"Send message to SQS",
   "States":{
      "Send message to SQS":{
         "Type":"Task",
         "Resource":"arn:aws:states:::sqs:sendMessage.waitForTaskToken",
         "Parameters":{
            "QueueUrl":"https://sqs.us-east-1.amazonaws.com/123456789012/myQueue",
            "MessageBody":{
               "Input.$":"$",
               "TaskToken.$":"$$.Task.Token"
            }
         },
         "End":true
      }
   }
}
```

Amazon SQS에서 메시지를 수신하는 방법에 대한 자세한 내용은 Amazon Simple Queue Service 개발자 안내서의 메시지 수신 및 삭제를 참조하세요.

다른 AWS 서비스와 Step Functions 함께 사용할 때 IAM 권한을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. 통합 서비스용 IAM 정책

통합 서비스로서의 AWS Step Functions 실행 관리

Step Functions는 서비스 통합으로 자체 API와 통합됩니다. 이를 통해 Step Functions는 진행 중인 실행의 작업 상태에서 직접 상태 시스템의 새로운 실행을 시작할 수 있습니다 새 워크플로를 작성할 때 기본 워크플로의 복잡성을 줄이고 공통 프로세스를 재사용하려면 중첩된 워크플로 실행을 사용하십시오.

최적화된 Step Functions 통합과 Step Functions AWS SDK 통합의 차이점

- 작업 실행(.sync) 통합 패턴을 사용할 수 있습니다.

  요청 및 응답 또는 작업 토큰을 사용하여 콜백 대기 통합 패턴에 대한 최적화는 없습니다.

자세한 내용은 다음을 참조하십시오.

- 작업에서 실행 시작
- 다른 서비스와 함께 사용
- 파라미터를 서비스 API에 전달

지원되는 Step Functions API 및 구문은 다음과 같습니다.

- StartExecution
  - 요청 구문
  - 지원되는 파라미터
    - Input
    - Name
    - StateMachineArn
  - 응답 구문

다음은 다른 상태 머신의 실행을 시작하고 완료될 때까지 기다리는 Task 상태를 포함합니다.

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution.sync:2",
   "Parameters":{
      "Input":{
        "Comment": "Hello world!"
       },
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Name":"ExecutionName"
   },
   "End":true
}
```

다음은 다른 상태 머신의 실행을 시작하는 Task 상태를 포함합니다.

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution",
   "Parameters":{
      "Input":{
        "Comment": "Hello world!"
```

```
       },
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Name":"ExecutionName"
   },
   "End":true
}
```

다음에는 콜백 서비스 통합 패턴을 구현하는 Task 상태가 포함됩니다.

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution.waitForTaskToken",
   "Parameters":{
      "Input":{
        "Comment": "Hello world!",
        "token.$": "$$.Task.Token"
       },
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
      "Name":"ExecutionName"
   },
   "End":true
}
```

중첩된 워크플로 실행을 시작한 상위 실행과 연결하려면 컨텍스트 객체에서 가져온 실행 ID를 포함하는 특수하게 명명된 파라미터를 전달하십시오. 중첩 실행을 시작할 때 AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID라는 매개 변수를 사용하십시오. 매개 변수 이름에 .$를 추가하고 $$.Execution.Id를 사용하여 컨텍스트 개체에서 ID를 참조하여 실행 ID를 전달하십시오. 자세한 정보는 컨텍스트 객체 액세스을 참조하세요.

```
{
   "Type":"Task",
   "Resource":"arn:aws:states:::states:startExecution.sync",
   "Parameters":{
      "Input":{
        "Comment": "Hello world!",
        "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
       },
      "StateMachineArn":"arn:aws:states:us-
east-1:123456789012:stateMachine:HelloWorld",
```

```
      "Name":"ExecutionName"
   },
   "End":true
}
```

중첩된 상태 머신은 다음을 반환합니다.

Resource	출력
startExecution.sync	String
startExecution.sync:2	JSON

둘 다 중첩된 상태 머신이 완료될 때까지 기다리지만, 다른 `Output` 형식을 반환합니다. 예를 들어 `{ "MyKey": "MyValue" }` 객체를 반환하는 Lambda 함수를 만드는 경우 다음과 같은 응답을 얻습니다.

startExecution.sync의 경우:

```
{
   <other fields>
   "Output": "{ \"MyKey\": \"MyValue\" }"
}
```

startExecution.sync:2의 경우:

```
{
   <other fields>
   "Output": {
      "MyKey": "MyValue"
   }
}
```

중첩된 상태 시스템에 대한 IAM 권한 구성

상위 상태 시스템은 폴링과 이벤트를 사용하여 하위 상태 시스템에서 실행을 완료했는지 확인합니다. 폴링에는 해당 권한이 필요한 `states:DescribeExecution` 반면, Step EventBridge Functions를 통해 전송된 이벤트에는 `events:PutTargetsevents:PutRule`, 및 `events:DescribeRule` 에

대한 권한이 필요합니다. IAM 역할에서 이러한 권한이 누락되면 상위 상태 시스템에서 하위 상태 시스템 실행 완료를 인식할 때까지 지연이 발생할 수 있습니다.

단일 중첩 워크플로 실행을 위해 `StartExecution`을 직접적으로 호출하는 상태 시스템의 경우 해당 상태 시스템에 대한 권한을 제한하는 IAM 정책을 사용합니다.

자세한 내용은 Step Functions에 대한 IAM 권한을 참조하세요.

타사 API 호출

HTTP 태스크는 워크플로에서 Salesforce 및 Stripe 등 모든 퍼블릭 타사 API를 호출할 수 있는 일종의 태스크 상태 상태입니다. 타사 API를 호출하려면 `arn:aws:states:::http:invoke` 리소스와 함께 태스크 상태를 사용합니다. 그런 다음 API URL, 사용할 방법, 인증 세부 정보 등 API 엔드포인트 구성 세부 정보를 입력합니다.

Workflow Studio를 사용하여 HTTP 태스크가 포함된 상태 머신을 빌드하는 경우, Workflow Studio는 HTTP 태스크에 대한 IAM 정책이 포함된 실행 역할을 자동으로 생성합니다. 자세한 정보는 Workflow Studio에서 HTTP 태스크를 테스트하기 위한 역할을 참조하세요.

주제

- HTTP 태스크 정의
- HTTP 태스크 필드
- HTTP 태스크에 대한 인증
- EventBridge 연결 및 HTTP 태스크 정의 데이터 병합
- 요청 본문에 URL 인코딩 적용
- HTTP 태스크를 실행하기 위한 IAM 권한
- HTTP 태스크 예제
- HTTP 태스크 테스트
- 지원되지 않는 HTTP 태스크 응답

HTTP 태스크 정의

ASL 정의는 `http:invoke` 리소스가 있는 HTTP 태스크를 나타냅니다. 다음 HTTP 태스크 정의는 모든 고객 목록을 반환하는 Stripe API를 호출합니다.

```
"Call third-party API": {
```

```
  "Type": "Task",
  "Resource": "arn:aws:states:::http:invoke",
  "Parameters": {
    "ApiEndpoint": "https://api.stripe.com/v1/customers",
    "Authentication": {
      "ConnectionArn": "arn:aws:events:us-east-2:123456789012:connection/
Stripe/81210c42-8af1-456b-9c4a-6ff02fc664ac"
    },
    "Method": "GET"
  },
  "End": true
}
```

HTTP 태스크 필드

HTTP 태스크의 정의에는 다음 필드가 포함됩니다.

`Resource`(필수)

태스크 유형을 지정하려면 `Resource` 필드에 해당 ARN을 입력합니다. HTTP 태스크의 경우 다음과 같이 `Resource` 필드를 지정합니다.

```
"Resource": "arn:aws:states:::http:invoke"
```

`Parameters`(필수)

호출하려는 타사 API에 대한 정보를 입력하는 `ApiEndpoint`, `Method`, `ConnectionArn` 필드가 포함되어 있습니다. `Parameters`에는 `Headers`, `QueryParameters` 등 옵션 필드도 포함되어 있습니다.

`Parameters`필드에서와 같이 정적 JSON과 JsonPath구문의 조합을 지정할 수 있습니다. 자세한 정보는 파라미터를 서비스 API에 전달을 참조하세요.

`ApiEndpoint`(필수)

호출하려는 타사 API의 URL을 지정합니다. URL에 쿼리 파라미터를 추가하려면 `QueryParameters` 필드를 사용합니다. 다음 예제에서는 Stripe API를 호출하여 모든 고객 목록을 가져오는 방법을 보여 줍니다.

```
"ApiEndpoint":"https://api.stripe.com/v1/customers"
```

JsonPath구문을 사용하여 타사 API URL이 포함된 JSON 노드를 선택하여 참조 경로를 지정할 수도 있습니다. 예를 들어 특정 고객 ID를 사용하여 Stripe의 API 중 하나를 호출하고 싶다고 가정해 보겠습니다. 다음 상태 입력을 제공했다고 생각해 보세요.

```
{
    "customer_id": "1234567890",
    "name": "John Doe"
}
```

Stripe API를 사용하여 이 고객 ID의 세부 정보를 검색하려면 다음 예제와 같이 `ApiEndpoint`를 지정합니다. 이 예제에서는 내장 함수와 참조 경로를 사용합니다.

```
"ApiEndpoint.$":"States.Format('https://api.stripe.com/v1/customers/{}',
 $.customer_id)"
```

런타임 시 Step Functions는 다음처럼 `ApiEndpoint` 값을 확인합니다.

```
https://api.stripe.com/v1/customers/1234567890
```

`Method`(필수)

타사 API를 호출하는 데 사용할 HTTP 메서드를 지정합니다. HTTP 태스크에서 GET, POST, POST, POST, POST, POST, HEAD 중 하나를 지정할 수 있습니다.

예를 들어 GET 메서드를 사용하려면 다음과 같이 `Method` 필드를 지정합니다.

```
"Method": "GET"
```

참조 경로를 사용하여 런타임에 메서드를 지정할 수도 있습니다. 예를 들어 **`"Method.$": "$.myHTTPMethod"`**입니다.

`Authentication`(필수)

타사 API 호출을 인증하는 방법을 지정하는 `ConnectionArn` 필드를 포함합니다. Step Functions는 Amazon EventBridge의 연결 리소스를 사용하여 특정 `ApiEndpoint`의 인증을 지원합니다.

`ConnectionArn`(필수)

EventBridge 연결 ARN을 지정합니다.

HTTP 태스크에는 API 제공자의 인증 자격 증명을 안전하게 관리하는 EventBridge 연결이 필요합니다. 연결은 타사 API를 인증하는 데 사용할 권한 부여 유형과 보안 인증을 지정합니다. 연결을 사용하면 API 키 등의 암호가 상태 머신 정의에 하드 코딩되는 것을 방지할 수 있습니다. 연결 시 `Headers`, `QueryParameters`, `RequestBody` 파라미터를 지정할 수도 있습니다.

EventBridge 연결을 생성할 때 인증 세부 정보를 제공합니다. HTTP 태스크에서 인증이 작동하는 방식에 대한 자세한 내용은 HTTP 태스크에 대한 인증 섹션을 참조하세요.

다음 예제는 HTTP 태스크 정의에서 `Authentication` 필드를 지정하는 방법을 보여 줍니다.

```
"Authentication": {
  "ConnectionArn": "arn:aws:events:us-east-2:123456789012:connection/
Stripe/81210c42-8af1-456b-9c4a-6ff02fc664ac"
}
```

`Headers` (선택 사항)

API 엔드포인트에 추가 컨텍스트와 메타데이터를 입력합니다. 헤더를 문자열 또는 JSON 배열로 지정할 수 있습니다.

EventBridge 연결에서 헤더를 지정하고 HTTP 태스크에서 `Headers` 필드를 지정할 수 있습니다. `Headers` 필드에는 API 공급자에 대한 인증 세부 정보를 포함하지 않는 것이 좋습니다. 이러한 세부 정보는 EventBridge 연결에 포함하기 바랍니다.

Step Functions는 EventBridge 연결에서 지정한 헤더를 HTTP 태스크 정의에서 지정한 헤더에 추가합니다. 정의 및 연결에 동일한 헤더 키가 있는 경우, Step Functions는 EventBridge 연결에 지정된 해당 값을 해당 헤더에 사용합니다. Step Functions가 데이터 병합을 수행하는 방법에 대한 자세한 내용은 EventBridge 연결 및 HTTP 태스크 정의 데이터 병합 섹션을 참조하세요.

다음 예제는 타사 API 호출에 포함될 헤더를 지정합니다. `content-type`

```
"Headers": {
  "content-type": "application/json"
}
```

참조 경로를 사용하여 런타임에 헤더를 지정할 수도 있습니다. 예를 들어 **`"Headers.$": "$.myHTTPHeaders"`**입니다.

Step Functions는 `User-Agent`, `Range`, `Host` 헤더를 설정합니다. Step Functions는 호출하는 API를 기반으로 `Host` 헤더 값을 설정합니다. 다음은 이러한 헤더 값의 예제입니다.

```
User-Agent: Amazon|StepFunctions|HttpInvoke|us-east-1,
Range: bytes=0-262144,
Host: api.stripe.com
```

HTTP 태스크 정의에는 다음 헤더를 사용할 수 없습니다. 이러한 헤더를 사용하면 HTTP 태스크에 `States.Runtime` 오류가 발생하여 실패합니다.

- A-IM
- Accept-Charset
- Accept-Datetime
- Accept-Encoding
- Cache-Control
- 연결
- Content-Encoding
- Content-MD5
- 날짜
- Expect
- 전달됨
- From
- Host
- HTTP2-Settings
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- 오리진(Origin)
- Pragma
- Proxy-Authorization
- 참조자

- Server
- TE
- 트레일러
- Transfer-Encoding
- 업그레이드
- Via
- 경고
- x-forwarded-*
- x-amz-*
- x-amzn-*

`QueryParameters` (선택 사항)

API URL 끝에 키-값 쌍을 삽입합니다. 쿼리 파라미터를 문자열, JSON 배열 또는 JSON 객체로 지정할 수 있습니다. Step Functions는 타사 API를 호출할 때 쿼리 파라미터를 자동으로 URL로 인코딩합니다.

예를 들어 Stripe API를 호출하여 미국 달러(USD)로 거래를 하는 고객을 검색한다고 가정해 보겠습니다. 다음 `QueryParameters`를 상태 입력으로 제공했다고 생각해 보세요.

```
"QueryParameters": {
  "currency": "usd"
}
```

런타임 시 Step Functions는 다음과 같이 API URL에 `QueryParameters`를 추가합니다.

```
https://api.stripe.com/v1/customers/search?currency=usd
```

참조 경로를 사용하여 런타임에 쿼리 파라미터를 지정할 수도 있습니다. 예를 들어 **`"QueryParameters.$": "$.myQueryParameters"`**입니다.

EventBridge 연결에서 쿼리 파라미터를 지정한 경우 Step Functions는 HTTP 태스크 정의에서 지정한 쿼리 파라미터에 이러한 쿼리 파라미터를 추가합니다. 정의 및 연결에 동일한 쿼리 파라미터 키가 있는 경우, Step Functions는 EventBridge 연결에 지정된 해당 값을 해당 헤더에 사용합니다. Step Functions가 데이터 병합을 수행 방법에 대한 자세한 내용은 EventBridge 연결 및 HTTP 태스크 정의 데이터 병합 섹션을 참조하세요.

`Transform` (선택 사항)

`RequestBodyEncoding` 및 `RequestEncodingOptions` 필드를 포함합니다. 기본적으로 Step Functions는 요청 본문을 JSON 데이터로 API 엔드포인트에 보냅니다.

API 공급자가 `form-urlencoded` 요청 본문을 수락하는 경우 `Transform` 필드를 사용하여 요청 본문의 URL 인코딩을 지정합니다. 또한 `content-type` 헤더를 `application/x-www-form-urlencoded`로 지정해야 합니다. 그러면 Step Functions는 요청 본문을 자동으로 URL로 인코딩합니다.

`RequestBodyEncoding`

요청 본문의 URL 인코딩을 지정합니다. `NONE` 또는 `URL_ENCODED` 값 중 하나를 지정할 수 있습니다.

- `NONE` - HTTP 요청 본문은 `RequestBody` 필드의 직렬화된 JSON이 됩니다. 이것이 기본값입니다.
- `URL_ENCODED` - HTTP 요청 본문은 `RequestBody` 필드의 URL로 인코딩된 양식 데이터입니다.

`RequestEncodingOptions`

`RequestBodyEncoding`을 `URL_ENCODED`로 설정한 경우 요청 본문의 배열에 사용할 인코딩 옵션을 결정합니다.

Step Functions는 다음의 배열 인코딩 옵션을 지원합니다. 이러한 옵션에 대한 자세한 정보와 예제는 요청 본문에 URL 인코딩 적용 섹션을 참조하세요.

- `INDICES` - 배열 요소의 인덱스 값을 사용하여 배열을 인코딩합니다. 기본적으로 Step Functions는 이 인코딩 옵션을 사용합니다.
- `REPEAT` - 배열의 각 항목에서 키를 반복합니다.
- `COMMAS` - 키의 모든 값을 쉼표로 구분된 값 목록으로 인코딩합니다.
- `BRACKETS` - 배열의 각 항목에서 키를 반복하고 키에 대괄호(`[]`)를 추가하여 배열임을 나타냅니다.

다음 예제에서는 요청 본문 데이터의 URL 인코딩을 설정합니다. 또한 요청 본문의 배열에 `COMMAS` 인코딩 옵션을 사용하도록 지정합니다.

```
"Transform": {
  "RequestBodyEncoding": "URL_ENCODED",
  "RequestEncodingOptions": {
    "ArrayFormat": "COMMAS"
```

```
  }
}
```

RequestBody (선택 사항)

상태 입력에 입력하는 JSON 데이터를 수용합니다. RequestBody에서는 정적 JSON과 JsonPath구문의 조합을 지정할 수 있습니다. 예를 들어 다음 상태 입력을 제공한다고 가정해 봅니다.

```
{
    "CardNumber": "1234567890",
    "ExpiryDate": "09/25"
}
```

런타임 시 요청 본문에서 CardNumber 및 ExpiryDate의 이러한 값을 사용하려면 요청 본문에 다음의 JSON 데이터를 지정하면 됩니다.

```
"RequestBody": {
  "Card": {
    "Number.$": "$.CardNumber",
    "Expiry.$": "$.ExpiryDate",
    "Name": "John Doe",
    "Address": "123 Any Street, Any Town, USA"
  }
}
```

호출하려는 타사 API에 form-urlencoded 요청 본문이 필요한 경우 요청 본문 데이터에 URL 인코딩을 지정해야 합니다. 자세한 정보는 요청 본문에 URL 인코딩 적용을 참조하세요.

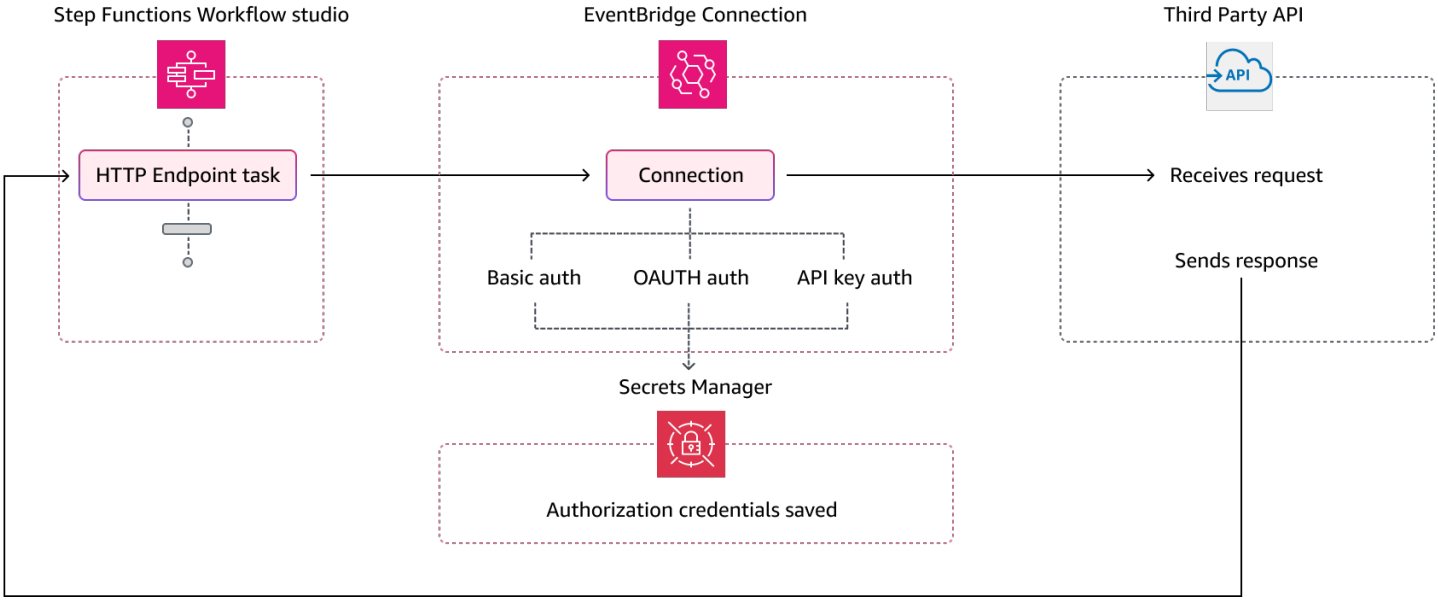
HTTP 태스크에 대한 인증

HTTP 태스크에는 API 제공자의 인증 자격 증명을 안전하게 관리하는 EventBridge 연결이 필요합니다. 연결은 타사 API를 인증하는 데 사용할 권한 부여 유형과 보안 인증을 지정합니다. 연결을 사용하면 API 키 등의 암호가 상태 머신 정의에 하드 코딩되는 것을 방지할 수 있습니다. EventBridge 연결은 기본, OAuth 및 API 키 인증 체계를 지원합니다.

EventBridge 연결을 생성할 때 인증 세부 정보를 제공합니다. 또한 API에 대한 권한 부여에 필요한 헤더, 본문 및 쿼리 파라미터를 포함할 수 있습니다. 타사 API를 호출하는 모든 HTTP 태스크에 연결 ARN을 포함해야 합니다.

연결을 생성하고 권한 부여 매개변수를 추가하면 비밀이 EventBridge 생성됩니다 AWS Secrets Manager. 이 시크릿에는 연결 및 권한 부여 파라미터를 암호화된 형태로 EventBridge 저장합니다. 연결을 성공적으로 만들거나 업데이트하려면 Secrets Manager를 사용할 권한이 AWS 계정 있는 서버를 사용해야 합니다. 상태 머신이 EventBridge 연결에 액세스하는 데 필요한 IAM 권한에 대한 자세한 내용은 을 참조하십시오HTTP 태스크를 실행하기 위한 IAM 권한.

다음 이미지는 Step Functions가 EventBridge 연결을 사용하여 타사 API 호출에 대한 권한 부여를 처리하는 방법을 보여 줍니다.



EventBridge 연결 및 HTTP 태스크 정의 데이터 병합

HTTP 태스크를 호출할 때 EventBridge 연결 및 HTTP 태스크 정의의 데이터를 지정할 수 있습니다. 이 데이터에는`Headers`, `QueryParameters`, `RequestBody` 파라미터가 포함됩니다. 타사 API를 호출하기 전에 Step Functions는 요청 본문이 문자열이고 연결 본문 파라미터가 비어 있지 않은 경우를 제외하고 모든 경우에 요청 본문을 연결 본문 파라미터와 병합합니다. 이 경우 HTTP 태스크에 `States.Runtime` 오류가 발생하여 실패합니다.

HTTP 태스크 정의 및 EventBridge 연결에 지정된 중복 키가 있는 경우 Step Functions는 HTTP 태스크의 값을 연결의 값으로 덮어씁니다.

다음 목록에는 타사 API를 호출하기 전에 Step Functions가 데이터를 병합하는 방법이 나와 있습니다.

- 헤더 - Step Functions는 연결에서 지정한 헤더를 HTTP 태스크의 `Headers` 필드에 있는 헤더에 추가합니다. 헤더 키 간에 충돌이 있으면 Step Functions는 연결에 지정된 값을 해당 헤더에 사용합니다. 예를 들어 HTTP 태스크 정의와 EventBridge 연결 모두에서 `content-type` 헤더를 지정한 경우 Step Functions는 연결에 지정된 `content-type` 헤더 값을 사용합니다.

- 쿼리 파라미터 - Step Functions는 연결에서 지정한 모든 쿼리 파라미터를 HTTP 태스크의 `QueryParameters` 필드에 있는 쿼리 파라미터에 추가합니다. 쿼리 파라미터 키 간에 충돌이 있으면 Step Functions는 연결에 지정된 값을 해당 쿼리 파라미터에 사용합니다. 예를 들어 HTTP 태스크 정의와 EventBridge 연결 모두에서 `maxItems` 쿼리 파라미터를 지정한 경우 Step Functions는 연결에 지정된 `maxItems` 쿼리 파라미터 값을 사용합니다.
- 본문 파라미터
  - Step Functions는 연결에 지정된 모든 요청 본문 값을 HTTP 태스크의 `RequestBody` 필드에 있는 요청 본문에 추가합니다. 요청 본문 키 간에 충돌이 있으면 Step Functions는 연결에 지정된 값을 요청 본문에 사용합니다. 예를 들어 HTTP 태스크 정의와 EventBridge 연결 모두에서 `RequestBody`의 `Mode` 필드를 지정했다고 가정해 봅니다. Step Functions는 연결에 지정된 `Mode` 필드 값을 사용합니다.
  - 요청 본문을 JSON 객체 대신 문자열로 지정하고 EventBridge 연결에 요청 본문도 포함된 경우 Step Functions는 두 위치에 지정된 요청 본문을 병합할 수 없습니다. `States.Runtime` 오류가 발생하여 HTTP 태스크가 실패합니다.

  Step Functions는 요청 본문 병합이 완료된 후 모든 변환을 적용하고 요청 본문을 직렬화합니다.

다음 예제에서는 HTTP 태스크와 EventBridge 연결 모두에서 `Headers`, `QueryParameters`, `RequestBody` 필드를 설정합니다.

HTTP 태스크 정의

```
{
  "Comment": "Data merging example for HTTP Task and EventBridge connection",
  "StartAt": "ListCustomers",
  "States": {
    "ListCustomers": {
      "Type": "Task",
      "Resource": "arn:aws:states:::http:invoke",
      "Parameters": {
        "Authentication": {
          "ConnectionArn": "arn:aws:events:us-
east-1:123456789012:connection/Example/81210c42-8af1-456b-9c4a-6ff02fc664ac"
        },
        "ApiEndpoint": "https:/example.com/path",
        "Method": "GET",
        "Headers": {
          "Request-Id": "my_request_id",
          "Header-Param": "state_machine_header_param"
```

```
        },
        "RequestBody": {
          "Job": "Software Engineer",
          "Company": "AnyCompany",
          "BodyParam": "state_machine_body_param"
        },
        "QueryParameters": {
          "QueryParam": "state_machine_query_param"
        }
      }
    }
  }
}
```

EventBridge 연결

```
{
  "AuthorizationType": "API_KEY",
  "AuthParameters": {
    "ApiKeyAuthParameters": {
      "ApiKeyName": "ApiKey",
      "ApiKeyValue": "key_value"
    },
    "InvocationHttpParameters": {
      "BodyParameters": [
        {
          "Key": "BodyParam",
          "Value": "connection_body_param"
        }
      ],
      "HeaderParameters": [
        {
          "Key": "Header-Param",
          "Value": "connection_header_param"
        }
      ],
      "QueryStringParameters": [
        {
          "Key": "QueryParam",
          "Value": "connection_query_param"
        }
      ]
    }
```

```
  }
}
```

이 예시에서는 HTTP 태스크 및 EventBridge 연결에 중복 키가 지정됩니다. 따라서 Step Functions는 HTTP 태스크의 값을 연결의 값으로 덮어씁니다. 다음 코드 조각은 Step Functions가 타사 API로 보내는 HTTP 요청을 보여 줍니다.

```
POST /path?QueryParam=connection_query_param HTTP/1.1
Apikey: key_value
Content-Length: 79
Content-Type: application/json; charset=UTF-8
Header-Param: connection_header_param
Host: example.com
Range: bytes=0-262144
Request-Id: my_request_id
User-Agent: Amazon|StepFunctions|HttpInvoke|us-east-1

{"Job":"Software Engineer","Company":"AnyCompany","BodyParam":"connection_body_param"}
```

요청 본문에 URL 인코딩 적용

기본적으로 Step Functions는 요청 본문을 JSON 데이터로 API 엔드포인트에 보냅니다. 타사 API 공급자가 `form-urlencoded` 요청 본문을 요구하는 경우 요청 본문에 URL 인코딩을 지정해야 합니다. 그러면 Step Functions는 선택한 URL 인코딩 옵션에 따라 요청 본문을 자동으로 URL로 인코딩합니다.

`Transform` 필드를 사용하여 URL 인코딩을 지정합니다. 이 필드에는 요청 본문에 URL 인코딩을 적용할지 여부를 지정하는 `RequestBodyEncoding` 필드가 포함됩니다. `RequestBodyEncoding` 필드를 지정하면 타사 API를 호출하기 전에 JSON 요청 본문이 Step Functions 요청 본문으로 변환됩니다. 또한 URL로 인코딩된 데이터를 수용하는 API는 `content-type` 헤더를 예상하기 때문에 `content-type` 헤더를 `application/x-www-form-urlencoded`로 지정해야 합니다.

요청 본문에 배열을 인코딩하기 위해 Step Functions는 다음의 배열 인코딩 옵션을 제공합니다.

- `INDICES` - 배열의 각 항목에서 키를 반복하고 키에 대괄호([])를 추가하여 배열임을 나타냅니다. 이 대괄호에는 배열 요소의 인덱스가 포함됩니다. 인덱스를 추가하면 배열 요소의 순서를 지정하는 데 도움이 됩니다. 기본적으로 Step Functions는 이 인코딩 옵션을 사용합니다.

  예를 들어 요청 본문에 다음 배열이 포함된 경우를 알아봅니다.

  ```
  {"array": ["a","b","c","d"]}
  ```

Step Functions는 배열을 다음 문자열로 인코딩합니다.

```
array[0]=a&array[1]=b&array[2]=c&array[3]=d
```

- REPEAT - 배열의 각 항목에서 키를 반복합니다.

  예를 들어 요청 본문에 다음 배열이 포함된 경우를 알아봅니다.

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions는 배열을 다음 문자열로 인코딩합니다.

  ```
  array=a&array=b&array=c&array=d
  ```

- COMMAS - 키의 모든 값을 쉼표로 구분된 값 목록으로 인코딩합니다.

  예를 들어 요청 본문에 다음 배열이 포함된 경우를 알아봅니다.

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions는 배열을 다음 문자열로 인코딩합니다.

  ```
  array=a,b,c,d
  ```

- BRACKETS - 배열의 각 항목에서 키를 반복하고 키에 대괄호([])를 추가하여 배열임을 나타냅니다.

  예를 들어 요청 본문에 다음 배열이 포함된 경우를 알아봅니다.

  ```
  {"array": ["a","b","c","d"]}
  ```

  Step Functions는 배열을 다음 문자열로 인코딩합니다.

  ```
  array[]=a&array[]=b&array[]=c&array[]=d
  ```

HTTP 태스크를 실행하기 위한 IAM 권한

상태 머신 실행 역할에는 타사 API를 호출하기 위한 HTTP 태스크에 대한 `states:InvokeHTTPEndpoint`, `events:RetrieveConnectionCredentials`,

`secretsmanager:GetSecretValue`, `secretsmanager:DescribeSecret` 권한이 있어야 합니다. 다음 IAM 정책 예제에서는 Stripe API를 호출하기 위해 상태 머신 역할에 필요한 최소 권한을 부여합니다. 또한 이 IAM 정책은 Secrets Manager에 저장된 이 EventBridge 연결의 암호를 포함하여 특정 연결에 액세스할 수 있는 권한을 상태 시스템 역할에 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Action": "states:InvokeHTTPEndpoint",
            "Resource": "arn:aws:states:us-
east-2:123456789012:stateMachine:myStateMachine",
            "Condition": {
                "StringEquals": {
                    "states:HTTPMethod": "GET"
                },
                "StringLike": {
                    "states:HTTPEndpoint": "https://api.stripe.com/*"
                }
            }
        },
        {
            "Sid": "Statement2",
            "Effect": "Allow",
            "Action": [
                "events:RetrieveConnectionCredentials",
            ],
            "Resource": "arn:aws:events:us-
east-2:123456789012:connection/oauth_connection/aeabd89e-d39c-4181-9486-9fe03e6f286a"
        },
        {
            "Sid": "Statement3",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:events!connection/*"
        }
    ]
```

```
}
```

HTTP 태스크 예제

다음 상태 머신 정의는 Headers, QueryParameters, Transform, RequestBody 파라미터가 포함된 HTTP 태스크를 보여 줍니다. HTTP 태스크는 Stripe API(https://api.stripe.com/v1/invoices)를 호출하여 인보이스를 생성합니다. 또한 HTTP 태스크는 INDICES 인코딩 옵션을 사용하여 요청 본문의 URL 인코딩을 지정합니다.

EventBridge 연결을 생성했는지 확인하세요. 다음 예제에서는 BASIC 권한 부여 유형을 사용하여 만든 연결을 보여 줍니다.

```
{
    "Type": "BASIC",
    "AuthParameters": {
        "BasicAuthParameters": {
            "Password": "myPassword",
            "Username": "myUsername"
         },
    }
}
```

텍스트를 리소스별 정보로 바꿔야 합니다.

```
{
  "Comment": "A state machine that uses HTTP Task",
  "StartAt": "CreateInvoiceAPI",
  "States": {
    "CreateInvoiceAPI": {
      "Type": "Task",
      "Resource": "arn:aws:states:::http:invoke",
      "Parameters": {
        "ApiEndpoint": "https://api.stripe.com/v1/invoices",
        "Method": "POST",
        "Authentication": {
          "ConnectionArn": ""arn:aws:events:us-east-2:123456789012:connection/
Stripe/81210c42-8af1-456b-9c4a-6ff02fc664ac"
        },
        "Headers": {
          "Content-Type": "application/x-www-form-urlencoded"
        },
        "RequestBody": {
```

```
          "customer.$": "$.customer_id",
          "description": "Monthly subscription",
          "metadata": {
            "order_details": "monthly report data"
          }
        },
        "Transform": {
          "RequestBodyEncoding": "URL_ENCODED",
          "RequestEncodingOptions": {
            "ArrayFormat": "INDICES"
          }
        }
      },
      "Retry": [
        {
          "ErrorEquals": [
            "States.Http.StatusCode.429",
            "States.Http.StatusCode.503",
            "States.Http.StatusCode.504",
            "States.Http.StatusCode.502"
          ],
          "BackoffRate": 2,
          "IntervalSeconds": 1,
          "MaxAttempts": 3,
          "JitterStrategy": "FULL"
        }
      ],
      "Catch": [
        {
          "ErrorEquals": [
            "States.Http.StatusCode.404",
            "States.Http.StatusCode.400",
            "States.Http.StatusCode.401",
            "States.Http.StatusCode.409",
            "States.Http.StatusCode.500"
          ],
          "Comment": "Handle all non 200 ",
          "Next": "HandleInvoiceFailure"
        }
      ],
      "End": true
    }
  }
```

```
}
```

이 상태 머신을 실행하려면 다음 예제와 같이 고객 ID를 입력합니다.

```
{
    "customer_id": "1234567890"
}
```

다음 예제는 Step Functions가 Stripe API로 보내는 HTTP 요청을 보여 줍니다.

```
POST /v1/invoices HTTP/1.1
Authorization: Basic <base64 of username and password>
Content-Type: application/x-www-form-urlencoded
Host: api.stripe.com
Range: bytes=0-262144
Transfer-Encoding: chunked
User-Agent: Amazon|StepFunctions|HttpInvoke|us-east-1

description=Monthly%20subscription&metadata%5Border_details%5D=monthly%20report
%20data&customer=1234567890
```

HTTP 태스크 테스트

콘솔, SDK 또는 를 통해 TestStateAPI를 사용하여 HTTP 태스크를 AWS CLI 테스트할 수 있습니다. 다음 절차는 Step Functions 콘솔에서 TestState API를 사용하는 방법을 설명합니다. HTTP 태스크가 예상대로 작동할 때까지 API 요청, 응답 및 권한 부여 세부 정보를 반복해서 테스트할 수 있습니다.

Step Functions 콘솔에서 HTTP 태스크 상태 테스트

1. Step Functions 콘솔을 엽니다.
2. 상태 머신 생성을 선택하여 상태 머신 생성을 시작하거나 HTTP 태스크가 포함된 기존 상태 머신을 선택합니다.

   기존 상태 머신에서 태스크를 테스트하는 경우 4단계를 참조하세요.
3. Workflow Studio의 디자인 모드에서 HTTP 태스크를 시각적으로 구성합니다. 또는 코드 모드를 선택하여 로컬 개발 환경에서 상태 머신 정의를 복사하여 붙여넣을 수 있습니다.
4. 디자인 모드인 Workflow Studio의 Inspector 패널에서 테스트 상태를 선택합니다.
5. 테스트 상태 대화 상자에서 다음을 수행합니다.

a. 실행 역할에서 상태를 테스트할 실행 역할을 선택합니다. HTTP 태스크에 대해 충분한 권한을 갖춘 역할이 없으면 Workflow Studio에서 HTTP 태스크를 테스트하기 위한 역할 섹션을 확인하여 역할을 생성합니다.

b. (선택 사항) 선택한 상태에서 테스트에 필요한 모든 JSON 입력을 제공합니다.

c. 검사 레벨에서 기본값인 정보를 그대로 유지합니다. 이 레벨은 API 직접 호출 및 상태 출력의 상태를 보여 줍니다. 이는 API 응답을 빠르게 확인하는 데 유용합니다.

d. 테스트 시작을 선택합니다.

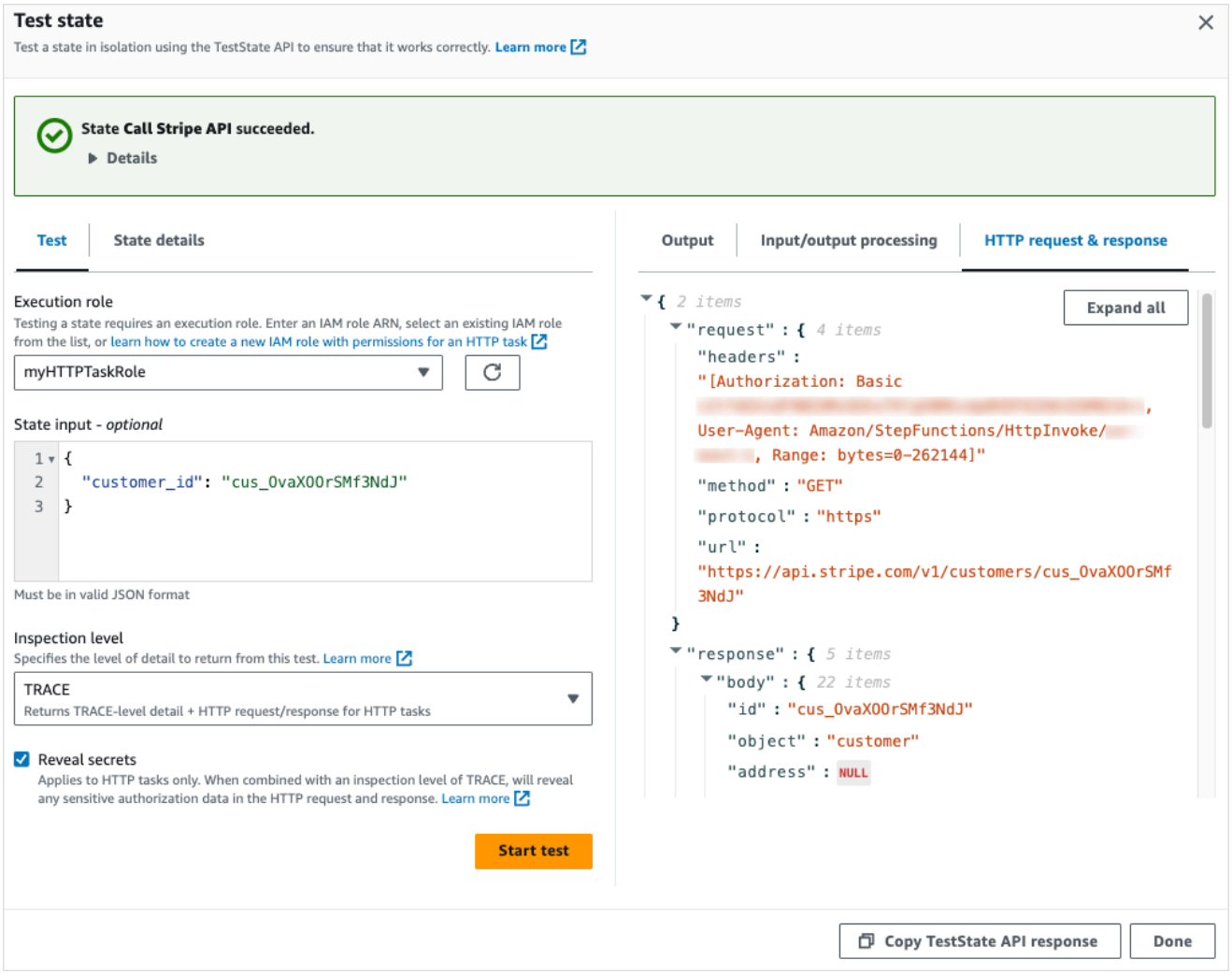
e. 테스트가 성공하면 상태 출력이 테스트 상태 대화 상자 오른쪽에 나타납니다. 테스트가 실패하면 오류가 나타납니다.

   대화 상자의 상태 세부 정보 탭에서 상태 정의와 EventBridge 연결 링크를 볼 수 있습니다.

f. 검사 레벨을 추적으로 변경합니다. 이 레벨은 원시 HTTP 요청 및 응답을 보여 주며 헤더, 쿼리 파라미터 및 기타 API별 세부 정보를 확인하는 데 유용합니다.

g. 비밀 공개 확인란을 선택합니다. 이 설정을 추적과 함께 사용하면 API 키와 같이 EventBridge 연결에 삽입되는 민감한 데이터를 볼 수 있습니다. 콘솔에 액세스하는 데 사용하는 IAM 사용자 ID에는 `states:RevealSecrets` 작업을 수행할 권한이 있어야 합니다. 이 권한이 없으면 Step Functions에서 테스트가 시작될 때 액세스 거부 오류가 발생합니다. IAM 권한을 설정하는 `states:RevealSecrets` 정책에 대한 예제는 IAM TestState API 사용 권한 섹션을 참조하세요.

   다음 이미지는 성공한 HTTP 태스크 테스트를 보여 줍니다. 이 상태의 검사 레벨은 TRACE로 설정되어 있습니다. 다음 이미지의 HTTP 요청 및 응답 탭은 타사 API 호출의 결과를 보여 줍니다.



h. 테스트 시작을 선택합니다.

i. 테스트가 성공하면 HTTP 요청 및 응답 탭에서 HTTP 세부 정보를 볼 수 있습니다.

지원되지 않는 HTTP 태스크 응답

반환된 응답이 다음 조건 중 하나에 해당하면 HTTP 태스크에서 `States.Runtime` 오류가 발생하여 작업이 실패합니다.

- 응답에는 `application/octet-stream`, `image/*`, `video/*`, `audio/*`의 콘텐츠 유형 헤더가 포함되어 있습니다.
- 응답은 유효한 문자열로 읽을 수 없습니다. 이진 데이터 또는 이미지 데이터를 예로 들 수 있습니다.

서비스 통합 패턴

AWS Step Functions Amazon States 언어로 서비스와 직접 통합됩니다. 세 가지 서비스 통합 패턴을 사용하여 이러한 AWS 서비스를 제어할 수 있습니다

- 서비스를 직접적으로 호출하고 Step Functions에서 HTTP 응답을 가져온 후 즉시 다음 상태로 진행할 수 있습니다.
- 서비스를 직접적으로 호출하고 작업이 완료될 때까지 Step Functions가 기다리도록 합니다.
- 작업 토큰으로 서비스를 직접적으로 호출하고 Step Functions는 해당 토큰이 페이로드와 함께 반환될 때까지 기다리도록 합니다.

이러한 각 서비스 통합 패턴은 작업 정의의 `"Resource"` 필드에 URI를 만드는 방법에 의해 제어됩니다.

통합 서비스를 호출하는 방법

- 요청 및 응답
- 작업 실행(.sync)
- 작업 토큰을 사용하여 콜백 대기

통합 서비스의 구성 AWS Identity and Access Management (IAM) 에 대한 자세한 내용은 을 참조하십시오. 통합 서비스용 IAM 정책

요청 및 응답

작업 상태의 `"Resource"` 문자열에서 서비스를 지정하고 리소스만 제공하면 Step Functions는 HTTP 응답을 기다린 후 다음 상태로 진행합니다. Step Functions는 작업이 완료될 때까지 기다리지 않습니다.

다음 예제에서는 Amazon SNS 주제를 게시하는 방법을 보여줍니다.

```
"Send message to SNS":{
   "Type":"Task",
   "Resource":"arn:aws:states:::sns:publish",
   "Parameters":{
      "TopicArn":"arn:aws:sns:us-east-1:123456789012:myTopic",
      "Message":"Hello from Step Functions!"
   },
```

```
   "Next":"NEXT_STATE"
}
```

이 예제는 Amazon SNS의 Publish API를 참조합니다. 워크플로는 Publish API를 호출한 후 다음 상태로 진행합니다.

> **Tip**
>
> 요청 응답 서비스 통합 패턴을 사용하는 샘플 워크플로를 사용자 AWS 계정환경에 배포하려면 모듈 2 - AWS Step Functions 워크숍의 요청 응답을 참조하십시오.

작업 실행(.sync)

Amazon AWS Batch ECS와 같은 통합 서비스의 경우 Step Functions는 요청이 완료될 때까지 기다린 후 다음 상태로 진행할 수 있습니다. Step Functions가 기다리도록 하려면 리소스 URI 뒤에 .sync 접미사를 추가하여 작업 상태 정의에서 "Resource" 필드를 지정합니다.

예를 들어, AWS Batch 작업을 제출할 때는 이 예제와 같이 상태 시스템 정의의 "Resource" 필드를 사용하십시오.

```
"Manage Batch task": {
  "Type": "Task",
  "Resource": "arn:aws:states:::batch:submitJob.sync",
  "Parameters": {
    "JobDefinition": "arn:aws:batch:us-east-2:123456789012:job-definition/
testJobDefinition",
    "JobName": "testJob",
    "JobQueue": "arn:aws:batch:us-east-2:123456789012:job-queue/testQueue"
  },
  "Next": "NEXT_STATE"
}
```

리소스 Amazon 리소스 이름(ARN)에 추가된 .sync 부분이 있으면 이는 Step Functions가 작업이 완료될 때까지 기다린다는 의미입니다. AWS Batch submitJob을 호출한 후 워크플로가 일시 중지됩니다. 작업이 완료되면 Step Functions는 다음 상태로 진행합니다. 자세한 내용은 AWS Batch 샘플 프로젝트를 참조하십시오배치 작업 관리(AWS Batch, Amazon SNS).

이 (.sync) 서비스 통합 패턴을 사용하는 작업이 중단되고 Step Functions가 작업을 취소할 수 없으면 통합 서비스에서 추가 요금이 발생할 수 있습니다. 다음과 같은 경우에 작업을 중단할 수 있습니다.

- 상태 시스템 실행이 중지되었습니다.
- 확인할 수 없는 오류가 발생하여 Parallel 상태의 다른 브랜치가 실패합니다.
- 확인할 수 없는 않는 오류가 발생하여 Map 상태 반복이 실패합니다.

Step Functions는 작업을 취소하기 위해 최선을 다합니다. 예를 들어 Step Functions `states:startExecution.sync` 작업이 중단되면 Step Functions `StopExecution` API 작업을 직접적으로 호출합니다. 하지만 Step Functions에서 작업을 취소하지 못할 수도 있습니다. 이에 대한 이유는 다음과 같지만 이에 국한되지는 않습니다.

- IAM 실행 역할에 해당 API 직접 호출을 수행할 수 있는 권한이 없습니다.
- 일시적으로 서비스가 중단되었습니다.

`.sync` 서비스 통합 패턴을 사용하면 Step Functions는 할당된 할당량과 이벤트를 사용하는 폴링을 사용하여 작업 상태를 모니터링합니다. 동일한 계정 내에서 `.sync` 호출하는 경우 Step Functions는 EventBridge 이벤트를 사용하고 상태에 지정된 API를 폴링합니다. `Task` 크로스 계정 `.sync` 간접 호출의 경우 Step Functions는 폴링만 사용합니다. 예를 들어`states:StartExecution.sync`, 의 경우 Step Functions는 DescribeExecutionAPI에서 폴링을 수행하고 할당된 할당량을 사용합니다.

> **Tip**
>
> Run a Job (.sync) 서비스 통합 패턴을 사용하는 샘플 워크플로를 사용자 AWS 계정환경에 배포하려면 워크숍의 모듈 3 - Run a Job (.sync) 을 AWS Step Functions 참조하십시오.

작업이 완료될 때까지 대기(`.sync`)를 지원하는 통합 서비스 목록을 보려면 Step Functions를 위한 최적화된 통합 단원을 참조하십시오.

> **Note**
>
> `.sync` 패턴을 사용하는 서비스 통합에는 추가 IAM 권한이 필요합니다. 자세한 정보는 통합 서비스용 IAM 정책을 참조하세요.

경우에 따라 Step Functions가 작업이 완전히 완료되기 전까지 워크플로를 계속하기를 원할 수 있습니다. 작업 토큰을 사용하여 콜백 대기 서비스 통합 패턴을 사용할 때와 같은 방법으로 이 작업을 수행할 수 있습니다. 이렇게 하려면 작업 토큰을 작업에 전달한 다음 `SendTaskSuccess` 또는

SendTaskFailure API 직접 호출을 사용하여 반환합니다. Step Functions는 해당 호출에서 제공한 데이터를 사용하여 작업을 완료하고 작업 모니터링을 중지하며 워크플로를 계속합니다.

작업 토큰을 사용하여 콜백 대기

콜백 작업은 작업 토큰이 반환될 때까지 워크플로를 일시 중지하는 방법을 제공합니다. 작업은 사람의 승인을 대기하거나, 타사와 통합하거나 레거시 시스템을 호출해야 할 수 있습니다. 이러한 작업의 경우 워크플로 실행이 1년 서비스 할당량(상태 제한과 관련된 할당량 참조)에 도달할 때까지 Step Functions를 일시 중지하고 외부 프로세스나 워크플로가 완료될 때까지 기다릴 수 있습니다. 이러한 상황에서 Step Functions를 사용하면 AWS SDK 서비스 통합과 일부 최적화 서비스 통합에 태스크 토큰을 전달할 수 있습니다. 작업은 SendTaskSuccess 또는 SendTaskFailure 호출을 통해 해당 작업 토큰을 다시 수신할 때까지 일시 중지됩니다.

콜백 작업 토큰을 사용하는 Task 상태 시간이 초과되면 새로운 무작위 토큰이 생성됩니다. 컨텍스트 객체에서 작업 토큰에 액세스할 수 있습니다.

> **Note**
>
> 작업 토큰은 1자 이상, 1024자 이하여야 합니다.

AWS SDK .waitForTaskToken 통합과 함께 사용하려면 사용하는 API에 태스크 토큰을 배치할 파라미터 필드가 있어야 합니다.

> **Note**
>
> 동일한 계정 내 보안 주체로부터 작업 토큰을 전달해야 합니다. AWS 다른 계정의 보안 주체로부터 토큰을 보내면 토큰이 작동하지 않습니다. AWS

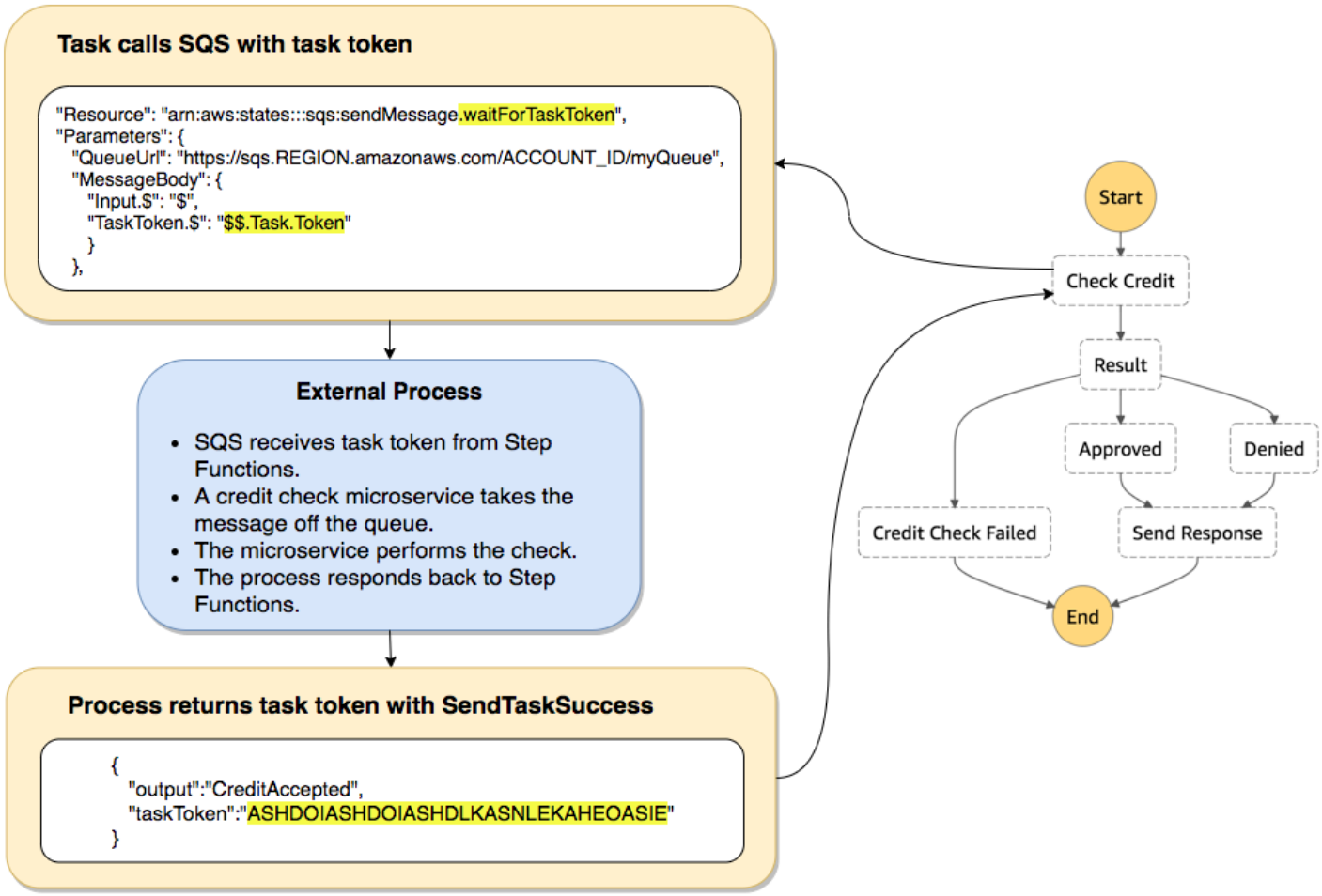
> **Tip**
>
> 콜백 태스크 토큰 서비스 통합 패턴을 사용하는 샘플 워크플로를 사용자에게 AWS 계정배포하려면 모듈 4 - 워크샵의 태스크 토큰을 사용한 콜백 대기 섹션을 참조하십시오. AWS Step Functions

작업 토큰 대기(.waitForTaskToken)를 지원하는 통합 서비스 목록은 Step Functions를 위한 최적화된 통합 단원을 참조하십시오.

주제

작업 토큰 예제

이 예제에서 Step Functions 워크플로는 외부 마이크로서비스와 통합하여 승인 워크플로의 일부로 크레딧 확인을 수행해야 합니다. Step Functions는 작업 토큰이 메시지의 일부로 포함된 Amazon SQS 메시지를 게시합니다. 외부 시스템은 Amazon SQS와 통합되고 대기열에서 메시지를 폴링합니다. 이 작업이 완료되면 결과와 원래 작업 토큰을 반환합니다. 그러면 Step Functions가 워크플로를 계속합니다.



Amazon SQS를 참조하는 작업 정의의 `"Resource"` 필드 끝에 `.waitForTaskToken`이 추가되어 있습니다.

```
"Send message to SQS": {
  "Type": "Task",
  "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
  "Parameters": {
    "QueueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/myQueue",
    "MessageBody": {
        "Message": "Hello from Step Functions!",
        "TaskToken.$": "$$.Task.Token"
     }
  },
  "Next": "NEXT_STATE"
}
```

이는 Step Functions에게 일시 중지하고 작업 토큰을 기다리라고 지시합니다. `.waitForTaskToken`을 사용하여 리소스를 지정하면 특정 경로 지정(`$$.Task.Token`)이 있는 상태 정의의 `"Parameters"` 필드에서 작업 토큰에 액세스할 수 있습니다. 처음 `$$.`는 경로가 컨텍스트 객체에 액세스하고 진행 중인 실행에서 현재 작업에 대한 작업 토큰을 가져오도록 지정합니다

작업이 완료되면 외부 서비스가 `taskToken`이 포함되어 있는 `SendTaskSuccess` 또는 `SendTaskFailure`를 직접적으로 호출합니다. 그런 다음에만 워크플우는 다음 상태로 계속합니다.

> **Note**
>
> 프로세스가 `SendTaskSuccess` 또는 `SendTaskFailure`와 함께 작업 토큰을 전달하는 데 실패하는 경우 무기한 대기하지 않도록 하려면 대기 작업에 대한 하트비트 시간 제한 구성 단원을 참조하십시오.

컨텍스트 객체에서 토큰 가져오기

컨텍스트 객체는 실행에 대한 정보가 포함된 내부 JSON 객체입니다. 상태 입력과 마찬가지로, 실행 중에 `"Parameters"` 필드의 경로를 사용하여 이 객체에 액세스할 수 있습니다. 작업 정의 내에서 액세스하는 경우 작업 토큰을 포함하여 특정 실행에 대한 정보가 포함됩니다.

```
{
    "Execution": {
        "Id": "arn:aws:states:us-
east-1:123456789012:execution:stateMachineName:executionName",
        "Input": {
           "key": "value"
```

```
        },
        "Name": "executionName",
        "RoleArn": "arn:aws:iam::123456789012:role...",
        "StartTime": "2019-03-26T20:14:13.192Z"
    },
    "State": {
        "EnteredTime": "2019-03-26T20:14:13.192Z",
        "Name": "Test",
        "RetryCount": 3
    },
    "StateMachine": {
        "Id": "arn:aws:states:us-east-1:123456789012:stateMachine:stateMachineName",
        "Name": "name"
    },
    "Task": {
        "Token": "h7XRiCdLtd/83p1E0dMccoxlzFhglsdkzpK9mBVKZsp7d9yrT1W"
    }
}
```

작업 정의의 "Parameters" 필드 내부에서 특정 경로를 사용하여 작업 토큰에 액세스할 수 있습니다. 입력 또는 컨텍스트 객체에 액세스하려면 먼저 .$를 파라미터 이름에 추가하여 파라미터가 경로가 되도록 지정합니다. 다음은 "Parameters" 지정에서 입력 및 컨텍스트 객체의 노드를 지정합니다.

```
"Parameters": {
   "Input.$": "$",
   "TaskToken.$": "$$.Task.Token"
 },
```

두 경우 모두 .$를 파라미터 이름에 추가하면 Step Functions에 경로를 예상하라고 지시하는 것입니다. 첫 번째 경우 "$"는 전체 입력이 포함된 경로입니다. 두 번째 경우, $$.는 경로가 컨텍스트 객체에 액세스하도록 지정하고 $$.Task.Token은 파라미터를 진행 중인 실행의 컨텍스트 객체에 있는 작업 토큰의 값으로 설정합니다.

Amazon SQS 예제에서 "Resource" 필드의 .waitForTaskToken은 Step Functions에 작업 토큰이 반환될 때까지 기다리라고 지시합니다. "TaskToken.$": "**$$.Task.Token**" 파라미터는 해당 토큰을 Amazon SQS 메시지의 일부로 전달합니다.

```
"Send message to SQS": {
  "Type": "Task",
  "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
  "Parameters": {
    "QueueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/myQueue",
```

```
    "MessageBody": {
        "Message": "Hello from Step Functions!",
        "TaskToken.$": "$$.Task.Token"
     }
  },
  "Next": "NEXT_STATE"
}
```

컨텍스트 객체에 대한 자세한 내용은 이 가이드의 컨텍스트 객체 단원에 있는 입/출력 처리 항목을 참조하십시오.

대기 작업에 대한 하트비트 시간 제한 구성

작업 토큰을 대기하고 있는 작업은 실행이 1년 서비스 할당량에 도달할 때까지 대기합니다(상태 제한과 관련된 할당량 참조). 실행 멈춤을 방지하기 위해 상태 머신 정의에서 하트비트 제한 시간 간격을 구성할 수 있습니다. `HeartbeatSeconds` 필드를 사용항 제한 시간 간격을 지정합니다.

```
{
  "StartAt": "Push to SQS",
  "States": {
    "Push to SQS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
      "HeartbeatSeconds": 600,
      "Parameters": {
        "MessageBody": { "myTaskToken.$": "$$.Task.Token" },
        "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/push-based-queue"
      },
      "ResultPath": "$.SQS",
      "End": true
    }
  }
}
```

이 상태 시스템 정의에서 작업은 메시지를 Amazon SQS로 푸시하고 제공된 작업 토큰을 사용하여 외부 프로세스가 다시 직접적으로 호출할 때까지 기다립니다. `"HeartbeatSeconds": 600` 필드는 하트비트 제한 시간 간격을 10분으로 설정합니다. 이 작업은 이러한 API 작업 중 하나를 사용하여 작업 토큰이 반환될 때까지 대기합니다.

- `SendTaskSuccess`
- `SendTaskFailure`

- SendTaskHeartbeat

대기 중인 작업이 10분 기간 이내에 유효한 작업 토큰을 수신하지 않으면 이 작업은 `States.Timeout` 오류 이름과 함께 실패합니다.

자세한 내용은 콜백 작업 샘플 프로젝트인 콜백 패턴 예제(Amazon SQS, Amazon SNS, Lamda) 를 참조하십시오.

파라미터를 서비스 API에 전달

`Task` 상태의 `Parameters` 필드를 사용하여 서비스 API로 전달할 파라미터를 제어합니다.

`Parameters` 필드 내에서는 API 작업에 배열 파라미터 복수형을 사용해야 합니다. 예를 들어 Amazon EC2와 통합을 위해 `DescribeSnapshots` API 작업의 필터 필드를 사용하는 경우 필드를 `Filters`로 정의해야 합니다. 복수형을 사용하지 않으면 Step Functions에서 다음과 같은 오류를 반환합니다.

```
The field Filter is not supported by Step Functions.
```

정적 JSON을 파라미터로 전달

상태 머신 정의에 JSON 객체를 직접 포함시켜 파라미터로 리소스에 전달할 수 있습니다.

예를 들어 AWS Batch에 대한 `SubmitJob` API의 `RetryStrategy` 파라미터를 설정하려면 파라미터에 다음을 포함시킬 수 있습니다.

```
"RetryStrategy": {
  "attempts": 5
}
```

정적 JSON이 포함된 여러 개의 파라미터를 전달할 수 있습니다. 보다 완벽한 예제로서 다음은 *myTopic*이라는 Amazon SNS 주제에 게시하는 작업의 사양에 대한 `Resource` 및 `Parameters` 필드입니다.

```
"Resource": "arn:aws:states:::sns:publish",
  "Parameters": {
     "TopicArn": "arn:aws:sns:us-east-2:123456789012:myTopic",
     "Message": "test message",
     "MessageAttributes": {
       "my attribute no 1": {
```

```
          "DataType": "String",
          "StringValue": "value of my attribute no 1"
        },
        "my attribute no 2": {
          "DataType": "String",
          "StringValue": "value of my attribute no 2"
        }
      }
  },
```

Path를 사용하여 상태 입력을 파라미터로 전달

Path를 사용하여 상태 입력의 일부분을 파라미터로 전달할 수 있습니다. 경로는 $로 시작하는 문자열이며 JSON 텍스트 내에서 구성 요소를 식별하는 데 사용됩니다. Step Functions 경로는 JsonPath 구문을 사용합니다.

파라미터에서 경로를 사용하도록 지정하려면 파라미터 이름을 .$로 끝냅니다. 예를 들어 상태 입력에서 message라는 노드 내의 텍스트를 포함한 경우 경로를 사용하여 해당 텍스트를 파라미터로 전달할 수 있습니다.

다음 상태 입력을 고려하세요.

```
{
  "comment": "A message in the state input",
  "input": {
    "message": "foo",
    "otherInfo": "bar"
  },
  "data": "example"
}
```

message라는 노드의 값을 파라미터로 전달하려면 다음 구문을 지정합니다.

```
"Parameters": {"myMessage.$": "$.input.message"},
```

그러면 Step Functions에서 foo 값을 파라미터로 전달합니다.

Step Functions에서 파라미터 사용에 대한 자세한 내용은 다음을 참조하세요.

- 입/출력 처리
- InputPath, 파라미터 및 ResultSelector

파라미터로서 Pass 컨텍스트 객체 노드

정적 콘텐츠 및 정적 입력의 노드 외에도 컨텍스트 객체의 노드를 파라미터로 전달할 수 있습니다. 컨텍스트 객체는 상태 머신 실행 중에 존재하는 동적 JSON 데이터입니다. 여기에는 상태 머신과 현재 실행에 대한 정보가 포함됩니다. 상태 정의의 `"Parameters"` 필드에서 경로를 사용하여 컨텍스트 객체에 액세스할 수 있습니다.

컨텍스트 객체와 `"Parameters"` 필드에서 해당 데이터에 액세스하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- 컨텍스트 객체
- 컨텍스트 객체 액세스
- 컨텍스트 객체에서 토큰 가져오기

지원되는 AWS SDK 통합에 대한 변경 로그

다음 표에는 서비스가 Step Functions와 처음 통합된 시기와 해당 통합 API가 가장 최근에 업데이트된 시기가 요약되어 있습니다. 통합 사용에 대한 자세한 내용은 을 참조하십시오. AWS SDK 서비스 통합

> **Important**
>
> API 작업 지원은 분기별로 릴리스됩니다. 새 매개변수와 같이 이미 지원되는 작업에 대한 업데이트는 즉시 제공되지 않을 수 있습니다.

Service	초기 지원	Updated	
AWS AppFabric	2024년 1월 18일		
B2B Data Interchange	2024년 1월 18일		
AWS Data Exports	2024년 1월 18일		
Amazon Bedrock	2024년 1월 18일		
Amazon Bedrock Agents	2024년 1월 18일		

Service	초기 지원	Updated	
Amazon Bedrock Runtime Agents	2024년 1월 18일		
Amazon Bedrock Runtime	2024년 1월 18일		
Amazon CloudFront KeyValueStore	2024년 1월 18일		
Amazon CodeGuru Security	2024년 1월 18일		
AWS Cost Optimization Hub	2024년 1월 18일		
Amazon DataZone	2024년 1월 18일		
Amazon EKS Auth	2024년 1월 18일		
AWS Entity Resolution	2024년 1월 18일		
AWS 프리 티어	2024년 1월 18일		
Amazon Inspector Scan	2024년 1월 18일		
AWS Launch Wizard	2024년 1월 18일		
Amazon Managed Blockchain Query	2024년 1월 18일		
AWS Elemental MediaPackage V2	2024년 1월 18일		
AWS HealthImaging	2024년 1월 18일		
Network Manager	2024년 1월 18일		

Service	초기 지원	Updated	
AWS Payment Cryptography	2024년 1월 18일		
AWS Payment Cryptography Data	2024년 1월 18일		
AWS Private CA Connector for Active Directory	2024년 1월 18일		
Amazon Q Business	2024년 1월 18일		
Amazon Q Connect	2024년 1월 18일		
AWS re:Post	2024년 1월 18일		
Amazon Timestream Query	2024년 1월 18일		
Amazon Timestream Write	2024년 1월 18일		
Trusted Advisor	2024년 1월 18일		
Verified Permissions	2024년 1월 18일		
Amazon WorkSpaces Thin Client	2024년 1월 18일		
AWS CloudTrail Data	2023년 6월 16일		
Amazon CloudWatch Internet Monitor	2023년 6월 16일		
Amazon Interacti ve Video Service RealTime	2023년 6월 16일		

Service	초기 지원	Updated	
AWS IoT TwinMaker	2023년 6월 16일		
Amazon OpenSearch Ingestion	2023년 6월 16일		
AWS Telco Network Builder	2023년 6월 16일		
Amazon VPC Lattice	2023년 6월 16일		
AWS Backup Storage	2023년 2월 17일		
Amazon Chime Media Pipelines	2023년 2월 17일		
Amazon Chime Voice	2023년 2월 17일		
AWS Clean Rooms	2023년 2월 17일	2024년 1월 18일	
Amazon CodeCatalyst	2023년 2월 17일		
Amazon Connect Cases	2023년 2월 17일		
AWS Control Tower	2023년 2월 17일		
Amazon DocumentDB Elastic Clusters	2023년 2월 17일		
Amazon EMR Serverless	2023년 2월 17일		
Amazon IVS Chat	2023년 2월 17일		
Amazon Kendra Intelligent Ranking	2023년 2월 17일		
AWS HealthOmics	2023년 2월 17일		

Service	초기 지원	Updated	
Amazon Redshift Serverless	2023년 2월 17일		
Amazon Security Lake	2023년 2월 17일		
AWS Health	2023년 2월 17일		
AWS IoT FleetWise	2023년 2월 17일		
AWS IoT RoboRunner	2023년 2월 17일		
AWS Mainframe Modernization	2023년 2월 17일		
AWS Migration Hub Orchestrator	2023년 2월 17일		
AWS Private 5G	2023년 2월 17일		
AWS 리소스 탐색기	2023년 2월 17일		
AWS SimSpace Weaver	2023년 2월 17일		
AWS Support App	2023년 2월 17일		
CloudWatch Observability Access Manager	2023년 2월 17일		
EventBridge Pipes	2023년 2월 17일		
EventBridge Scheduler	2023년 2월 17일		
IAM Roles Anywhere	2023년 2월 17일		

Service	초기 지원	Updated	
Kinesis Video WebRTC Storage	2023년 2월 17일		
License Manager Linux Subscriptions	2023년 2월 17일		
License Manager User Subscriptions	2023년 2월 17일		
OpenSearch Serverless	2023년 2월 17일		
Route 53 ARC Zonal Shift	2023년 2월 17일		
SageMaker Geospatial	2023년 2월 17일		
SageMaker Metrics	2023년 2월 17일		
Systems Manager for SAP	2023년 2월 17일		
AWS Account Management	2022년 4월 19일		
AWS Amplify	2021년 9월 30일		
AWS App Mesh	2021년 9월 30일		
AWS App Runner	2021년 9월 30일	2023년 2월 17일	
AWS AppConfig	2021년 9월 30일		
AWS AppConfig Data	2022년 4월 19일		
AWS AppSync	2021년 9월 30일	2023년 2월 17일	

Service	초기 지원	Updated	
AWS Application Discovery Service	2021년 9월 30일		
AWS Application Migration Service	2021년 9월 30일		
AWS Audit Manager	2021년 9월 30일		
AWS Auto Scaling Plans	2021년 9월 30일		
AWS Backup	2021년 9월 30일	2023년 2월 17일	
AWS Backup gateway	2022년 4월 19일	2023년 2월 17일	
AWS Batch	2021년 9월 30일	2023년 2월 17일	
AWS Billing Conductor	2022년 7월 26일	2023년 2월 17일	
AWS Budgets	2021년 9월 30일		
AWS Certificate Manager	2021년 9월 30일		
AWS Private Certificate Authority	2021년 9월 30일		
AWS Cloud Map	2021년 9월 30일		
AWS Cloud9	2021년 9월 30일		
AWS CloudFormation	2021년 9월 30일	2023년 2월 17일	
AWS CloudHSM	2021년 9월 30일		
AWS CloudHSM	2021년 9월 30일		
AWS CloudTrail	2021년 9월 30일	2023년 2월 17일	

Service	초기 지원	Updated	
AWS Cloud Control	2022년 4월 19일		
AWS CodeBuild	2021년 9월 30일		
AWS CodeCommit	2021년 9월 30일	2023년 2월 17일	
AWS CodeDeploy	2021년 9월 30일		
AWS CodePipeline	2021년 9월 30일		
AWS CodeStar	2021년 9월 30일		
AWS CodeStar	2021년 9월 30일		
AWS CodeStar	2021년 9월 30일		
AWS Compute Optimizer	2021년 9월 30일	2023년 2월 17일	
AWS Config	2021년 9월 30일	2022년 7월 26일	
AWS Cost Explorer Service	2021년 9월 30일	2023년 2월 17일	
AWS Cost and Usage Report	2021년 9월 30일		
AWS Data Exchange	2021년 9월 30일	2022년 7월 26일	
AWS Data Pipeline	2021년 9월 30일		
AWS DataSync	2021년 9월 30일	2022년 7월 26일	
AWS Database Migration Service	2021년 9월 30일		
AWS Device Farm	2021년 9월 30일		
AWS Direct Connect	2021년 9월 30일		

Service	초기 지원	Updated	
AWS Directory Service	2021년 9월 30일	2023년 2월 17일	
AWS EC2 Instance Connect	2021년 9월 30일		
AWS Elastic Beanstalk	2021년 9월 30일		
AWS Elemental MediaLive	2021년 9월 30일		
AWS Elemental MediaPackage	2021년 9월 30일		
AWS Elemental MediaPackage VOD	2021년 9월 30일		
AWS Elemental MediaStore	2021년 9월 30일		
AWS Fault Injection Service	2021년 9월 30일		
AWS Firewall Manager	2021년 9월 30일	2023년 2월 17일	
AWS Glue	2021년 9월 30일	2023년 2월 17일	
AWS Glue DataBrew	2021년 9월 30일		
AWS IoT Greengrass	2021년 9월 30일		
AWS Ground Station	2021년 9월 30일	2023년 2월 17일	
AWS Identity and Access Management	2021년 9월 30일		

Service	초기 지원	Updated	
AWS IoT	2021년 9월 30일	2023년 2월 17일	
AWS IoT 1-Click	2021년 9월 30일		
AWS IoT Analytics	2021년 9월 30일		
AWS IoT Core Device Advisor	2021년 9월 30일		
AWS IoT Events	2021년 9월 30일		
AWS IoT Events 데이터	2021년 9월 30일		
AWS IoT Fleet Hub	2021년 9월 30일		
AWS IoT Greengrass Version 2	2021년 9월 30일		
AWS IoT Jobs Data Plane	2021년 9월 30일		
AWS IoT Secure Tunneling	2021년 9월 30일		
AWS IoT SiteWise	2021년 9월 30일	2023년 2월 17일	
AWS IoT Things Graph	2021년 9월 30일		
AWS IoT Wireless	2021년 9월 30일	2023년 2월 17일	
AWS Key Management Service	2021년 9월 30일	2022년 7월 26일	
AWS Lake Formation	2021년 9월 30일	2023년 2월 17일	
AWS Lambda	2021년 9월 30일	2023년 2월 17일	

Service	초기 지원	Updated	
AWS License Manager	2021년 9월 30일	2023년 2월 17일	
AWS Marketplace	2021년 9월 30일	2023년 2월 17일	
AWS Marketplace Commerce Analytics	2021년 9월 30일		
AWS Marketplace Entitlement Service	2021년 9월 30일		
AWS Elemental MediaTailor	2021년 9월 30일	2022년 7월 26일	
AWS Migration Hub	2021년 9월 30일		
AWS Migration Hub Config	2021년 9월 30일		
AWS Migration Hub 전략 권장 사항	2022년 4월 19일	2023년 2월 17일	
AWS Mobile	2021년 9월 30일		
AWS Network Firewall	2021년 9월 30일		
AWS OpsWorks	2021년 9월 30일		
AWS OpsWorks CM	2021년 9월 30일		
AWS Organizations	2021년 9월 30일	2023년 2월 17일	
AWS Outposts	2021년 9월 30일		
AWS Panorama	2022년 4월 19일	2023년 2월 17일	
Amazon Relational Database Service Performance Insights	2021년 9월 30일		

Service	초기 지원	Updated	
AWS 가격표	2021년 9월 30일		
Amazon Relational Database Service	2021년 9월 30일		
AWS Resilience Hub	2022년 4월 19일		
AWS Resource Access Manager	2021년 9월 30일		
AWS Resource Groups	2021년 9월 30일	2023년 2월 17일	
AWS Resource Groups Tagging API	2021년 9월 30일		
AWS RoboMaker	2021년 9월 30일		
AWS IAM Identity Center	2021년 9월 30일	2023년 2월 17일	
AWS SSO OIDC	2021년 9월 30일		
AWS Secrets Manager	2021년 9월 30일		
AWS Security Token Service	2021년 9월 30일		
AWS Security Hub	2021년 9월 30일		
AWS Server Migration Service	2021년 9월 30일		
AWS Service Catalog	2021년 9월 30일		
AWS Service Catalog AppRegistry	2021년 9월 30일	2023년 2월 17일	

Service	초기 지원	Updated	
AWS Shield	2021년 9월 30일		
AWS Signer	2021년 9월 30일		
AWS IAM Identity Center	2021년 9월 30일		
AWS IAM Identity Center Admin	2021년 9월 30일		
AWS Step Functions	2021년 9월 30일	2023년 2월 17일	
AWS Storage Gateway	2021년 9월 30일		
AWS Support	2021년 9월 30일		
AWS Transfer Family	2021년 9월 30일	2023년 2월 17일	
AWS WAF	2021년 9월 30일		
AWS WAF Regional	2021년 9월 30일		
AWS WAFV2	2021년 9월 30일		
AWS Well-Architected Tool	2021년 9월 30일	2023년 2월 17일	
AWS X-Ray	2021년 9월 30일	2023년 2월 17일	
AWS Marketplace Metering Service	2021년 9월 30일		
AWS Serverless Application Repository	2021년 9월 30일		
AWS Identity and Access Management Access Analyzer	2021년 9월 30일		

Service	초기 지원	Updated	
Alexa for Business	2021년 9월 30일		
Amazon API Gateway	2021년 9월 30일	2023년 2월 17일	
Amazon API Gateway	2021년 9월 30일		
Amazon AppIntegrations	2021년 9월 30일		
Amazon AppStream 2.0	2021년 9월 30일		
Amazon AppFlow	2021년 9월 30일	2023년 2월 17일	
Amazon Athena	2021년 9월 30일	2023년 2월 17일	
Amazon Augmented AI	2021년 9월 30일		
Amazon Braket	2021년 9월 30일		
Amazon Chime	2021년 9월 30일		
Amazon Chime Meetings	2022년 4월 19일	2023년 2월 17일	
Amazon Cloud Directory	2021년 9월 30일		
Amazon CloudFront	2021년 9월 30일	2023년 2월 17일	
Amazon CloudSearch	2021년 9월 30일		
Amazon CloudWatch	2021년 9월 30일	2023년 2월 17일	
Amazon CloudWatch Application Insights	2021년 9월 30일		
CloudWatch Evidently	2022년 4월 19일		

Service	초기 지원	Updated	
Amazon CloudWatch Logs	2021년 9월 30일		
Amazon CloudWatch RUM	2022년 4월 19일	2023년 2월 17일	
Amazon CloudWatch Synthetics	2021년 9월 30일		
Amazon CodeGuru Profiler	2021년 9월 30일		
Amazon CodeGuru Reviewer	2021년 9월 30일		
Amazon Cognito	2021년 9월 30일		
Amazon Cognito Identity Provider	2021년 9월 30일		
Amazon Cognito Sync	2021년 9월 30일		
Amazon Comprehend	2021년 9월 30일	2023년 2월 17일	
Amazon Comprehend Medical	2021년 9월 30일		
Amazon Connect Contact Lens	2021년 9월 30일		
Amazon Connect Participant Service	2021년 9월 30일		
Amazon Connect	2021년 9월 30일	2023년 2월 17일	
Amazon Connect Voice ID	2022년 4월 19일		

Service	초기 지원	Updated	
Amazon Connect Wisdom	2022년 4월 19일		
Amazon Data Lifecycle Manager	2021년 9월 30일		
Amazon Detective	2021년 9월 30일		
Amazon DevOps Guru	2021년 9월 30일	2022년 7월 26일	
Amazon DocumentDB (with MongoDB compatibility)	2021년 9월 30일		
Amazon DynamoDB	2021년 9월 30일	2023년 2월 17일	
Amazon DynamoDB Streams	2021년 9월 30일		
Amazon EC2 Container Registry	2021년 9월 30일		
Amazon EC2 Container Service	2021년 9월 30일	2023년 2월 17일	
Amazon EC2 Systems Manager	2021년 9월 30일	2023년 2월 17일	
Amazon EMR	2021년 9월 30일	2023년 2월 17일	
Amazon ElastiCache	2021년 9월 30일		
Amazon Elastic Inference	2021년 9월 30일		
Amazon Elastic Block Store	2021년 9월 30일		

Service	초기 지원	Updated	
Amazon Elastic Compute Cloud	2021년 9월 30일	2023년 2월 17일	
Amazon Elastic Container Registry Public	2021년 9월 30일		
Amazon Elastic File System	2021년 9월 30일		
Amazon Elastic Kubernetes Service	2021년 9월 30일	2023년 2월 17일	
Amazon EMR	2021년 9월 30일		
Amazon Elastic Transcoder	2021년 9월 30일		
Amazon OpenSearch Service	2021년 9월 30일	2023년 2월 17일	
Amazon OpenSearch Service	2022년 4월 19일	2023년 2월 17일	
Amazon EventBridge	2021년 9월 30일	2023년 2월 17일	
Amazon FSx	2021년 9월 30일	2023년 2월 17일	
Amazon Forecast Query	2021년 9월 30일	2023년 2월 17일	
Amazon Forecast Service	2021년 9월 30일	2023년 2월 17일	
Amazon Fraud Detector	2021년 9월 30일		
Amazon GameLift	2021년 9월 30일	2023년 2월 17일	

Service	초기 지원	Updated	
Amazon GameSparks	2022년 7월 26일		
Amazon S3 Glacier	2021년 9월 30일		
Amazon GuardDuty	2021년 9월 30일		
AWS HealthLake	2021년 9월 30일		
Amazon Honeycode	2021년 9월 30일		
Amazon Inspector	2021년 9월 30일		
Amazon Inspector V2	2022년 4월 19일		
Amazon Interactive Video Service	2021년 9월 30일		
Amazon Kendra	2021년 9월 30일		
Amazon Kinesis	2021년 9월 30일		
Amazon Kinesis Analytics	2021년 9월 30일		
Amazon Kinesis Analytics V2	2021년 9월 30일		
Amazon Kinesis Firehose	2021년 9월 30일		
Amazon Kinesis Video Signaling Channels	2021년 9월 30일		
Amazon Kinesis Video Streams	2021년 9월 30일	2023년 2월 17일	

Service	초기 지원	Updated	
Amazon Kinesis Video Streams Archived Media	2021년 9월 30일		
Amazon Kinesis video stream	2021년 9월 30일		
Amazon Lex Model Building Service	2021년 9월 30일		
Amazon Lex Model Building Service V2	2021년 9월 30일	2023년 2월 17일	
Amazon Lex	2021년 9월 30일		
Amazon Lex Runtime V2	2021년 9월 30일		
Amazon Lightsail	2021년 9월 30일	2023년 2월 17일	
Amazon Location Service	2021년 9월 30일	2023년 2월 17일	
Amazon Lookout for Equipment	2021년 9월 30일		
Amazon Lookout for Metrics	2021년 9월 30일	2023년 2월 17일	
Amazon Lookout for Vision	2021년 9월 30일		
Amazon MQ	2021년 9월 30일		
Amazon Macie	2021년 9월 30일		
Amazon Macie 2	2021년 9월 30일	2023년 2월 17일	

Service	초기 지원	Updated	
Amazon Managed Blockchain	2021년 9월 30일	2023년 2월 17일	
Amazon Managed Grafana	2022년 4월 19일	2023년 2월 17일	
Amazon Managed Service for Prometheus	2021년 9월 30일	2023년 2월 17일	
Amazon Managed Streaming for Apache Kafka	2021년 9월 30일	2023년 2월 17일	
Amazon Managed Streaming for Apache Kafkax	2022년 4월 19일		
Amazon Managed Workflows for Apache Airflow	2021년 9월 30일		
Amazon Mechanical Turk	2021년 9월 30일		
Amazon MemoryDB for Redis	2022년 4월 19일	2023년 2월 17일	
Amazon Nimble Studio	2021년 9월 30일		
Amazon Personalize	2021년 9월 30일	2023년 2월 17일	
Amazon Personalize Events	2021년 9월 30일		

Service	초기 지원	Updated	
Amazon Personalize Runtime	2021년 9월 30일		
Amazon Pinpoint	2021년 9월 30일		
Amazon Pinpoint Email Service	2021년 9월 30일		
Amazon Pinpoint SMS and Voice Service	2021년 9월 30일		
Amazon Pinpoint SMS and Voice V2 Service	2022년 7월 26일		
Amazon Polly	2021년 9월 30일		
Amazon QLDB	2021년 9월 30일		
Amazon QLDB Session	2021년 9월 30일		
Amazon QuickSight	2021년 9월 30일	2023년 2월 17일	
Amazon Redshift	2021년 9월 30일		
Amazon Redshift Data API	2021년 9월 30일		
Amazon Rekognition	2021년 9월 30일	2023년 2월 17일	
Amazon Relational Database Service	2021년 9월 30일	2023년 2월 17일	
Amazon Route 53	2021년 9월 30일		

Service	초기 지원	Updated	
Amazon Route 53 Recovery Control Config	2021년 9월 30일	2022년 7월 26일	
Amazon Route 53 Domains	2021년 9월 30일	2023년 2월 17일	
Amazon Route 53 Resolver	2021년 9월 30일		
Amazon S3 on Outposts	2021년 9월 30일	2022년 7월 26일	
Amazon SageMaker Runtime Feature Store Runtime	2021년 9월 30일		
Amazon SageMaker Runtime Runtime	2021년 9월 30일		
Amazon SageMaker	2021년 9월 30일	2023년 2월 17일	
Amazon SageMaker Edge Manager	2021년 9월 30일		
Amazon Simple Email Service	2021년 9월 30일		
Amazon Simple Email Service V2	2021년 9월 30일	2023년 2월 17일	
Amazon Simple Notification Service	2021년 9월 30일	2023년 2월 17일	
Amazon Simple Queue Service	2021년 9월 30일	2023년 2월 17일	

Service	초기 지원	Updated	
Amazon Simple Storage Service	2021년 9월 30일	2023년 2월 17일	
Amazon Simple Workflow Service	2021년 9월 30일		
Amazon Textract	2021년 9월 30일	2023년 2월 17일	
Amazon Transcribe	2021년 9월 30일		
Amazon Translate	2021년 9월 30일	2023년 2월 17일	
Amazon WorkDocs	2021년 9월 30일	2023년 2월 17일	
Amazon WorkMail	2021년 9월 30일	2023년 2월 17일	
Amazon WorkMail Message Flow	2021년 9월 30일		
Amazon WorkSpaces	2021년 9월 30일	2023년 2월 17일	
Amazon WorkSpaces Web	2022년 4월 19일	2023년 2월 17일	
Amplify	2021년 9월 30일		
Amplify UI Builder	2022년 4월 19일	2023년 2월 17일	
Application Auto Scaling	2021년 9월 30일		
Amazon EC2 Auto Scaling	2021년 9월 30일	2023년 2월 17일	
CodeArtifact	2021년 9월 30일		
DynamoDB Accelerator	2021년 9월 30일		

Service	초기 지원	Updated	
EC2 Image Builder	2021년 9월 30일		
AWS Elastic Disaster Recovery	2022년 4월 19일	2023년 2월 17일	
Elastic Load Balancing	2021년 9월 30일		
Elastic Load Balancing V2	2021년 9월 30일		
MediaConnect	2021년 9월 30일		
Amazon S3 Control	2021년 9월 30일	2023년 2월 17일	
Recycle Bin for Amazon EBS	2022년 4월 19일	2023년 2월 17일	
Savings Plans	2021년 9월 30일		
Amazon EventBridge Schema Registry	2021년 9월 30일		
Service Quotas	2021년 9월 30일		
AWS Snowball	2021년 9월 30일		

Step Functions를 위한 샘플 프로젝트

AWS Step Functions 콘솔에서 다음 스타터 템플릿 중 하나를 선택하여 상태 시스템을 AWS 계정에 배포할 수 있습니다. 이러한 스타터 템플릿은 워크플로 프로토타입과 정의, 프로젝트에 대한 모든 관련 AWS 리소스를 자동으로 만드는 바로 실행 가능한 샘플 프로젝트입니다.

이러한 샘플 프로젝트를 사용하여 있는 그대로 배포 및 실행하거나 워크플로 프로토타입을 사용하여 이를 기반으로 빌드할 수 있습니다. 이러한 프로젝트에서 빌드하면 Step Functions는 워크플로 프로토타입을 만들지만 워크플로 정의에 나열된 리소스를 배포하지는 않습니다.

각 샘플 프로젝트를 배포하면 샘플 프로젝트는 완전히 기능하는 상태 시스템을 프로비저닝하고 실행할 상태 시스템의 관련 리소스를 만듭니다. 샘플 프로젝트를 만들면 Step Functions는 AWS CloudFormation을 사용하여 상태 시스템에서 참조하는 관련 리소스를 만듭니다.

주제

- 배치 작업 관리(AWS Batch, Amazon SNS)
- 컨테이너 작업 관리(Amazon ECS, Amazon SNS)
- 데이터 레코드 전송(Lambda, DynamoDB, Amazon SQS)
- 작업 상태 설문 조사 (Lambda AWS Batch,)
- 태스크 타이머(Lambda, Amazon SNS)
- 콜백 패턴 예제(Amazon SQS, Amazon SNS, Lamda)
- Amazon EMR 작업 관리
- EMR Serverless 작업 실행
- 워크플로에서 워크플로 시작(Step Functions, Lambda)
- Map 상태를 사용하여 데이터 동적 처리
- Distributed Map을 사용하여 CSV 파일 처리
- Distributed Map을 사용하여 Amazon S3 버킷의 데이터 처리
- 기계 학습 모델 훈련
- 기계 학습 모델 튜닝
- Amazon SQS에서 대용량 메시지 처리(Express 워크플로)
- 선택적 체크포인트 예(Express 워크플로)
- AWS CodeBuild 프로젝트 구축 (CodeBuild, 아마존 SNS)

- 데이터 전처리 및 기계 학습 모델 학습
- Lambda 오케스트레이션 예제
- Athena 쿼리 시작
- 여러 쿼리 실행(Amazon Athena, Amazon SNS)
- 대규모 데이터세트 쿼리 (아마존 아테나, 아마존 S3 AWS Glue, 아마존 SNS)
- 데이터를 최신으로 유지 (아마존 아테나, 아마존 S3,) AWS Glue
- Amazon EKS 클러스터 관리
- API Gateway 직접 호출
- API Gateway 통합을 사용하여 Fargate에서 실행되는 마이크로서비스 직접 호출
- 에 맞춤 이벤트 보내기 EventBridge
- 동기 Express 워크플로 간접 호출
- Amazon Redshift(Lambda, Amazon Redshift Data API)를 사용하여 ETL/ELT 워크플로 실행
- Step Functions 및 오류 처리 기능이 있는 AWS Batch 사용
- AWS Batch 작업 팬 아웃
- AWS Batch 람다와 함께
- Amazon Bedrock을 사용하여 AI 프롬프트 체이닝 수행

배치 작업 관리(AWS Batch, Amazon SNS)

이 샘플 프로젝트는 AWS Batch 작업을 제출하고, 작업의 성공 또는 실패 여부에 따라 Amazon SNS 알림을 전송하는 방법을 보여줍니다. 이 샘플 프로젝트를 배포하면 AWS Step Functions 상태 머신, AWS Batch 작업 및 Amazon SNS 주제가 생성됩니다.

이 프로젝트에서 Step Functions은 상태 시스템을 사용하여 AWS Batch 작업을 동기식으로 호출합니다. 그런 다음 작업이 성공 또는 실패하기까지 대기한 다음 작업의 성공 또는 실패 여부에 대한 메시지와 함께 Amazon SNS 주제를 전송합니다.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

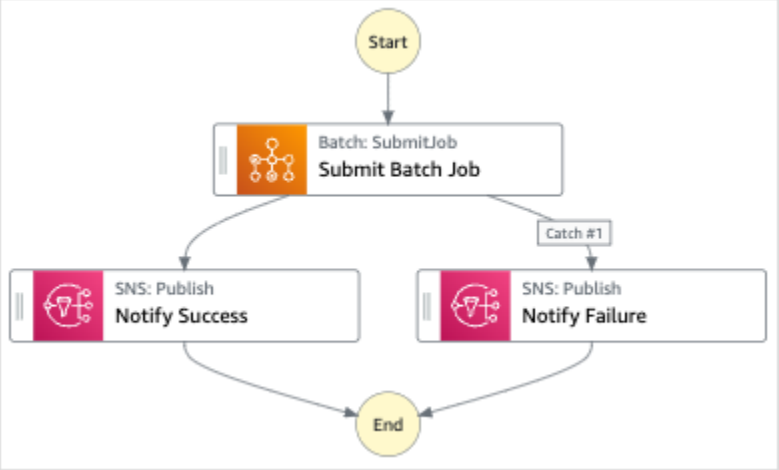
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Manage a batch job`**을 입력한 다음 반환된 검색 결과에서 배치 작업 관리를 선택합니다.

3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - AWS Batch 직업
   - Amazon SNS 주제
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 배치 작업 관리 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 파라미터를 해당 리소스에 직접 전달하여 Amazon AWS Batch SNS와 통합됩니다.

이 예제 상태 머신을 살펴보고 `Resource` 필드의 Amazon 리소스 이름 (ARN) 에 연결하고 서비스 API에 `Parameters` 전달하여 Step Functions가 Amazon SNS를 제어하는 AWS Batch 방법을 살펴봅니다.

다른 AWS 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
  "Comment": "An example of the Amazon States Language for notification on an AWS Batch
 job completion",
  "StartAt": "Submit Batch Job",
  "TimeoutSeconds": 3600,
  "States": {
    "Submit Batch Job": {
      "Type": "Task",
      "Resource": "arn:aws:states:::batch:submitJob.sync",
      "Parameters": {
        "JobName": "BatchJobNotification",
        "JobQueue": "arn:aws:batch:us-east-1:123456789012:job-queue/
BatchJobQueue-7049d367474b4dd",
        "JobDefinition": "arn:aws:batch:us-east-1:123456789012:job-definition/
BatchJobDefinition-74d55ec34c4643c:1"
      },
      "Next": "Notify Success",
      "Catch": [
          {
            "ErrorEquals": [ "States.ALL" ],
            "Next": "Notify Failure"
          }
      ]
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "Batch job submitted through Step Functions succeeded",
        "TopicArn": "arn:aws:sns:us-east-1:123456789012:batchjobnotificatiointemplate-
SNSTopic-1J757CVBQ2KHM"
      },
      "End": true
    },
    "Notify Failure": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "Batch job submitted through Step Functions failed",
        "TopicArn": "arn:aws:sns:us-east-1:123456789012:batchjobnotificatiointemplate-
SNSTopic-1J757CVBQ2KHM"
      },
      "End": true
```

```
    }
  }
}
```

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:ap-northeast-1:123456789012:ManageBatchJob-SNSTopic-
JHLYYG7AZPZI"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "batch:SubmitJob",
                "batch:DescribeJobs",
                "batch:TerminateJob"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:ap-northeast-1:123456789012:rule/
StepFunctionsGetEventsForBatchJobsRule"
            ],
            "Effect": "Allow"
```

```
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

컨테이너 작업 관리(Amazon ECS, Amazon SNS)

이 샘플 프로젝트에서는 AWS Fargate 작업을 실행한 다음 해당 작업의 성공 또는 실패 여부에 따라 Amazon SNS 알림을 전송하는 방법을 보여줍니다. 이 샘플 프로젝트를 배포하면 AWS Step Functions 상태 시스템, Fargate 클러스터 및 Amazon SNS 주제가 생성됩니다.

이 프로젝트에서 Step Functions는 상태 시스템을 사용하여 Fargate 작업을 동기식으로 직접 호출합니다. 그런 다음 작업이 성공 또는 실패하기까지 대기한 다음 작업의 성공 또는 실패 여부에 대한 메시지와 함께 Amazon SNS 주제를 전송합니다.

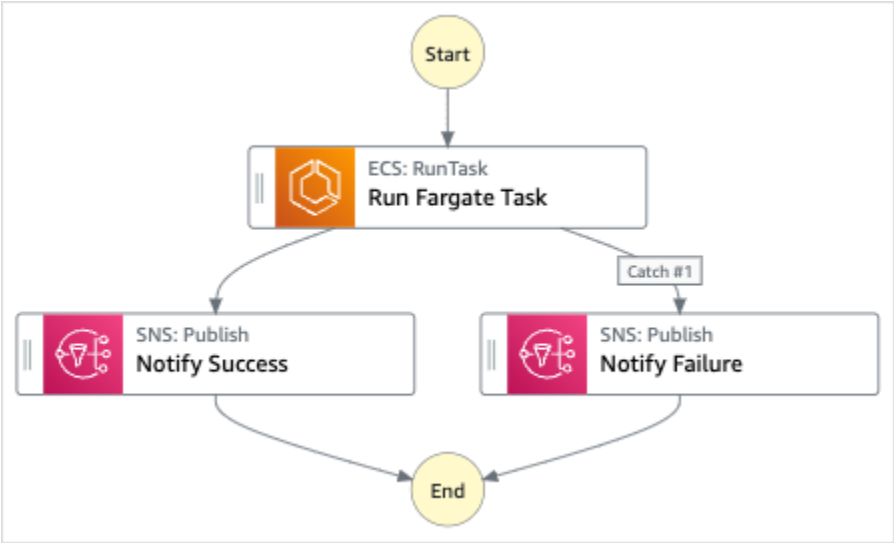
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Manage a container task`**를 입력한 다음 반환된 검색 결과에서 컨테이너 태스크 관리를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - AWS Fargate 클러스터
   - Amazon SNS 주제
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 컨테이너 태스크 관리 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

   - 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

     > **Tip**
     >
     > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

      > **Note**
      >
      > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

      데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

      > **Note**
      >
      > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 파라미터를 해당 리소스에 직접 전달하여 Amazon AWS Fargate SNS와 통합됩니다. 이 예제 상태 시스템을 살펴보고 Step Functions에서 상태 시스템을 사용하여 Fargate 작업을 동기식으로 직접 호출하고 작업이 성공하거나 실패할 때까지 기다리고 작업의 성공 또는 실패 여부에 대한 메시지와 함께 Amazon SNS 주제를 전송하는 방법을 확인합니다.

다른 AWS 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
  "Comment": "An example of the Amazon States Language for notification on an AWS
 Fargate task completion",
  "StartAt": "Run Fargate Task",
  "TimeoutSeconds": 3600,
  "States": {
    "Run Fargate Task": {
      "Type": "Task",
      "Resource": "arn:aws:states:::ecs:runTask.sync",
      "Parameters": {
        "LaunchType": "FARGATE",
        "Cluster": "arn:aws:ecs:ap-northeast-1:123456789012:cluster/
FargateTaskNotification-ECSCluster-VHLR20IF9IMP",
        "TaskDefinition": "arn:aws:ecs:ap-northeast-1:123456789012:task-definition/
FargateTaskNotification-ECSTaskDefinition-13YOJT8Z2LY5Q:1",
        "NetworkConfiguration": {
          "AwsvpcConfiguration": {
            "Subnets": [
              "subnet-07e1ad3abcfce6758",
              "subnet-04782e7f34ae3efdb"
```

```
            ],
            "AssignPublicIp": "ENABLED"
          }
        }
      },
      "Next": "Notify Success",
      "Catch": [
          {
            "ErrorEquals": [ "States.ALL" ],
            "Next": "Notify Failure"
          }
      ]
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "AWS Fargate Task started by Step Functions succeeded",
        "TopicArn": "arn:aws:sns:ap-northeast-1:123456789012:FargateTaskNotification-
SNSTopic-1XYW5YD5V0M7C"
      },
      "End": true
    },
    "Notify Failure": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "AWS Fargate Task started by Step Functions failed",
        "TopicArn": "arn:aws:sns:ap-northeast-1:123456789012:FargateTaskNotification-
SNSTopic-1XYW5YD5V0M7C"
      },
      "End": true
    }
  }
}
```

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 모든 권한만 포함시키는 것이 좋습니다.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:ap-northeast-1:123456789012:FargateTaskNotification-
SNSTopic-1XYW5YD5V0M7C"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "arn:aws:ecs:ap-northeast-1:123456789012:task-definition/
FargateTaskNotification-ECSTaskDefinition-13YOJT8Z2LY5Q:1"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "ecs:StopTask",
                "ecs:DescribeTasks"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:ap-northeast-1:123456789012:rule/
StepFunctionsGetEventsForECSTaskRule"
            ],
            "Effect": "Allow"
        }
    ]
```

```
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

데이터 레코드 전송(Lambda, DynamoDB, Amazon SQS)

이 샘플 프로젝트에서는 Step Functions 상태 시스템을 사용하여 Amazon DynamoDB 테이블에서 항목을 반복해서 읽고 이러한 항목을 Amazon SQS 대기열로 보내는 방법을 보여줍니다. 이 샘플 프로젝트를 배포하면 Step Functions 상태 시스템, DynamoDB 테이블, AWS Lambda 함수 및 Amazon SQS 대기열이 생성됩니다.

이 프로젝트에서 Step Functions는 Lambda 함수를 사용하여 DynamoDB 테이블을 채웁니다. 또한 상태 시스템은 `for` 루프를 사용하여 각 항목을 읽은 다음 각 항목을 Amazon SQS 대기열로 보냅니다.

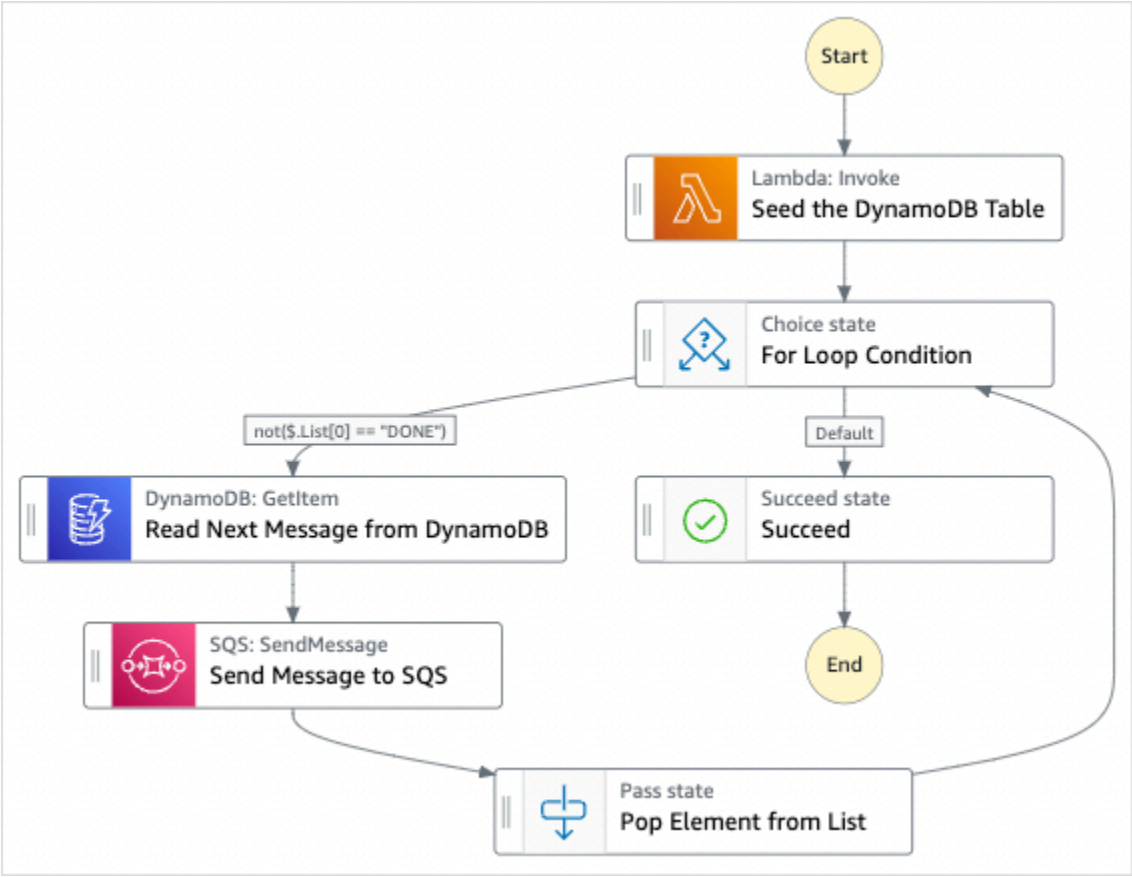
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Transfer data records`**를 입력한 다음 반환된 검색 결과에서 데이터 레코드 전송을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - DynamoDB 테이블을 시딩할 수 있는 Lambda 함수
   - Amazon SQS 대기열
   - DynamoDB 테이블
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 데이터 레코드 전송 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.
    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 시스템은 파라미터를 리소스에 직접 전달하여 DynamoDB 및 Amazon SQS와 통합됩니다.

이 예제 상태 시스템을 살펴보고 `Resource` 필드의 Amazon 리소스 이름(ARN)과 연결하고 `Parameters`를 서비스 API에 전달하여 Step Functions에서 DynamoDB 및 Amazon SQS를 제어하는 방법을 확인합니다.

다른 AWS 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "Comment" : "An example of the Amazon States Language for reading messages from a
 DynamoDB table and sending them to SQS",
  "StartAt": "Seed the DynamoDB Table",
  "TimeoutSeconds": 3600,
  "States": {
    "Seed the DynamoDB Table": {
      "Type": "Task",
```

```
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:sqsconnector-
SeedingFunction-T3U43VYDU5OQ",
      "ResultPath": "$.List",
      "Next": "For Loop Condition"
    },
    "For Loop Condition": {
      "Type": "Choice",
      "Choices": [
        {
          "Not": {
            "Variable": "$.List[0]",
            "StringEquals": "DONE"
          },
          "Next": "Read Next Message from DynamoDB"
        }
      ],
      "Default": "Succeed"
    },
    "Read Next Message from DynamoDB": {
      "Type": "Task",
      "Resource": "arn:aws:states:::dynamodb:getItem",
      "Parameters": {
        "TableName": "sqsconnector-DDBTable-1CAFOJWP8QD6I",
        "Key": {
          "MessageId": {"S.$": "$.List[0]"}
        }
      },
      "ResultPath": "$.DynamoDB",
      "Next": "Send Message to SQS"
    },
    "Send Message to SQS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sqs:sendMessage",
      "Parameters": {
        "MessageBody.$": "$.DynamoDB.Item.Message.S",
        "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/sqsconnector-
SQSQueue-QVGQBW134PWK"
      },
      "ResultPath": "$.SQS",
      "Next": "Pop Element from List"
    },
    "Pop Element from List": {
      "Type": "Pass",
      "Parameters": {
```

```
        "List.$": "$.List[1:]"
      },
      "Next": "For Loop Condition"
    },
    "Succeed": {
      "Type": "Succeed"
    }
  }
}
```

파라미터 전달 및 결과 관리에 대한 자세한 내용은 다음을 참조하십시오.

- 파라미터를 서비스 API에 전달
- ResultPath

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 모든 권한만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:ap-northeast-1:123456789012:table/
TransferDataRecords-DDBTable-3I41R5L5EAGT"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": [
                "arn:aws:sqs:ap-northeast-1:123456789012:TransferDataRecords-SQSQueue-
BKWXTS09LIW1"
            ],
```

```
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:invokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:ap-
northeast-1:123456789012:function:TransferDataRecords-SeedingFunction-VN4KY2TPAZSR"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

작업 상태 설문 조사 (Lambda AWS Batch,)

이 샘플 프로젝트는 AWS Batch 작업 폴러를 생성합니다. 작업을 확인하는 AWS Step Functions 상태 루프를 만드는 AWS Lambda 데 사용하는 `Wait` 상태 머신을 구현합니다. AWS Batch

이 샘플 프로젝트는 Step Functions 워크플로가 작업을 제출하고 AWS Batch 작업이 완료될 때까지 기다렸다가 성공적으로 종료되도록 모든 리소스를 만들고 구성합니다.

> **Note**
>
> 또한 Lambda 함수를 사용하지 않고 이 패턴을 구현할 수 있습니다. AWS Batch 직접 제어에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

이 샘플 프로젝트는 상태 머신, Lambda 함수 2개, 대기열 1개를 생성하고 AWS Batch 관련 IAM 권한을 구성합니다.

다른 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. AWS 다른 서비스와 AWS Step Functions 함께 사용

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

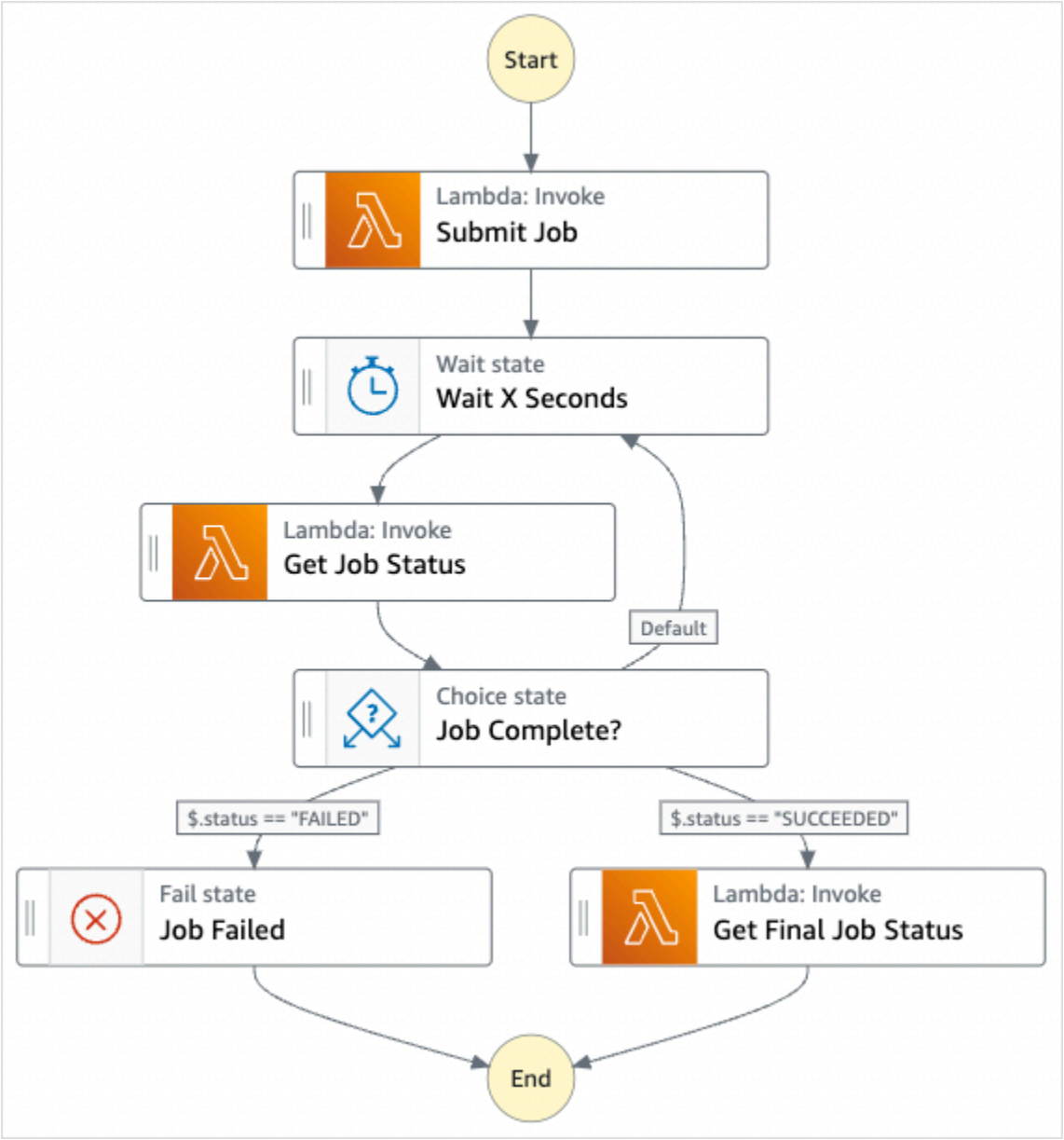
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.

2. 검색 상자에 **`Job Poller`**를 입력한 다음 반환된 검색 결과에서 작업 폴러를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - 작업을 제출하고 AWS Batch AWS Batch 제출된 작업의 현재 상태 및 최종 작업 완료 상태를 가져오는 세 개의 Lambda 함수.
   - 작업 AWS Batch
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 작업 폴러 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

  > **Tip**
  >
  > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

  준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

  이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

  샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

  > **Important**
  >
  > 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

모든 리소스가 프로비저닝되고 배포되면 다음과 비슷한 예제 입력과 함께 실행 시작 대화 상자가 표시됩니다.

```
{
    "jobName": "my-job",
    "jobDefinition": "arn:aws:batch:us-east-2:123456789012:job-definition/
SampleJobDefinition-343f54b445d5312:1",
    "jobQueue": "arn:aws:batch:us-east-2:123456789012:job-queue/
SampleJobQueue-4d9d696031e1449",
    "wait_time": 60
```

```
}
```

> **Note**
>
> `wait_time`은 `Wait` 상태에 60초마다 루프를 실행하도록 명령합니다.

- 실행 시작 대화 상자에서 다음을 수행합니다.

  1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

     > **Note**
     >
     > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

  2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

     데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

     > **Note**
     >
     > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

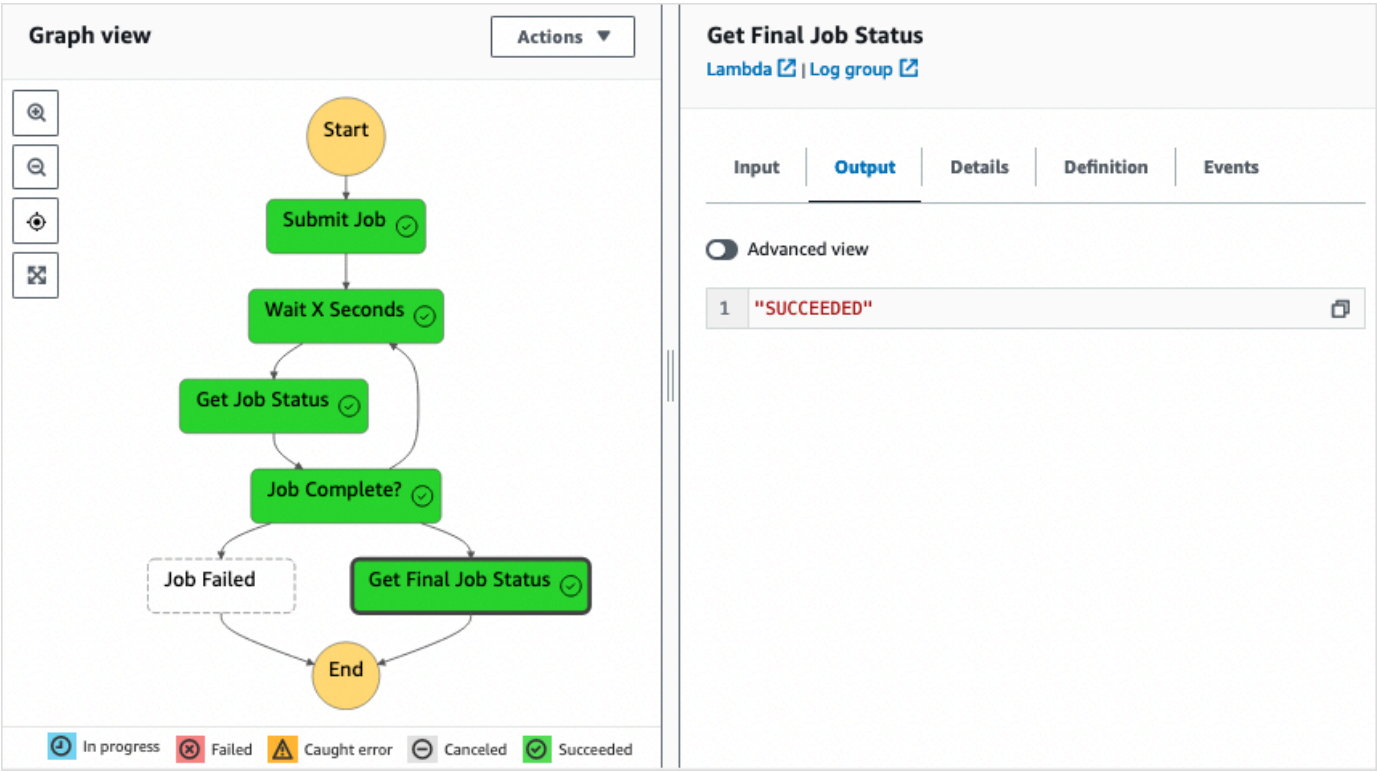
  3. 실행 시작을 선택합니다.

  4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

     실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예를 들어, AWS Batch 작업 상태 변경 및 실행 반복 결과를 보려면 [출력] 탭을 선택합니다.

다음 이미지에서는 그래프 보기의 실행 상태 그래프를 보여줍니다. 또한 출력 탭에서 선택한 단계의 실행 출력도 보여줍니다.



예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 와 통합되어 작업을 AWS Lambda 제출합니다. AWS Batch 이 예제 상태 머신을 살펴보고 Step Functions가 Lambda AWS Batch및 를 제어하는 방법을 알아보십시오.

다른 AWS 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
    "Comment": "An example of the Amazon States Language that runs an AWS Batch job and
 monitors the job until it completes.",
    "StartAt": "Submit Job",
    "States": {
      "Submit Job": {
        "Type": "Task",
```

```
        "Resource": "arn:aws:lambda:us-
east-1:111122223333:function:StepFunctionsSample-JobStatusPol-SubmitJobFunction-
jDaYcl4cx55r",
        "ResultPath": "$.guid",
        "Next": "Wait X Seconds"
      },
      "Wait X Seconds": {
        "Type": "Wait",
        "SecondsPath": "$.wait_time",
        "Next": "Get Job Status"
      },
      "Get Job Status": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:us-
east-1:111122223333:function:StepFunctionsSample-JobStatusPoll-
CheckJobFunction-1JkJwY10vonI",
        "Next": "Job Complete?",
        "InputPath": "$.guid",
        "ResultPath": "$.status"
      },
      "Job Complete?": {
        "Type": "Choice",
        "Choices": [
          {
            "Variable": "$.status",
            "StringEquals": "FAILED",
            "Next": "Job Failed"
          },
          {
            "Variable": "$.status",
            "StringEquals": "SUCCEEDED",
            "Next": "Get Final Job Status"
          }
        ],
        "Default": "Wait X Seconds"
      },
      "Job Failed": {
        "Type": "Fail",
        "Cause": "AWS Batch Job Failed",
        "Error": "DescribeJob returned FAILED"
      },
      "Get Final Job Status": {
        "Type": "Task",
```

```
        "Resource": "arn:aws::lambda:us-
east-1:111122223333:function:StepFunctionsSample-JobStatusPoll-
CheckJobFunction-1JkJwY10vonI",
        "InputPath": "$.guid",
        "End": true
      }
    }
  }
```

태스크 타이머(Lambda, Amazon SNS)

이 샘플 프로젝트는 작업 타이머를 생성합니다. 상태를 구현하는 AWS Step Functions 상태 머신을 구현하고 Amazon Simple Notification Service (Amazon SNS) 알림을 보내는 AWS Lambda 함수를 사용합니다. `Wait` Wait 상태는 하나의 작업 단위를 수행하기 위해 트리거를 기다리는 상태 유형입니다.

> **Note**
>
> 이 샘플 프로젝트는 Amazon Simple Notification Service (Amazon SNS) 알림을 보내는 AWS Lambda 기능을 구현합니다. Amazon States Language에서 직접 Amazon SNS 알림을 보낼 수도 있습니다. 다른 서비스와 AWS Step Functions 함께 사용 섹션을 참조하십시오.

이 샘플 프로젝트는 상태 머신, Lambda 함수 및 Amazon SNS 주제를 생성하고 관련 (IAM) 권한을 구성합니다. AWS Identity and Access Management 작업 타이머 샘플 프로젝트를 사용하여 생성되는 리소스에 대한 자세한 내용은 다음을 참조하십시오.

다른 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. AWS 다른 서비스와 AWS Step Functions 함께 사용

- AWS CloudFormation 사용 설명서
- Amazon Simple Notification Service 개발자 안내서
- AWS Lambda 개발자 안내서
- IAM 시작 안내서

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

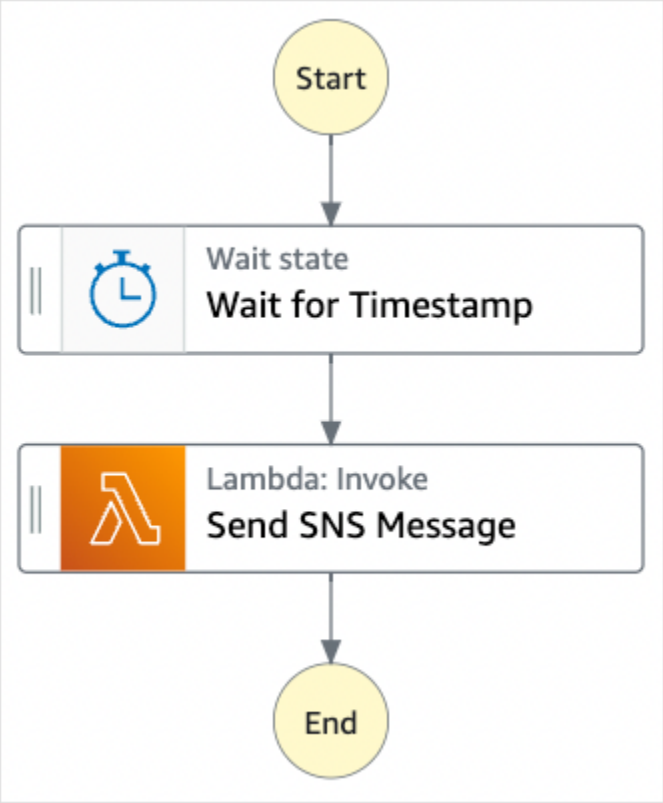
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.

2. 검색 상자에 **`Task Timer`**를 입력한 다음 반환되는 검색 결과에서 태스크 타이머를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon SNS 알림을 보내는 Lambda 함수
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 태스크 타이머 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

모든 리소스가 프로비저닝되고 배포되면 다음과 비슷한 예제 입력과 함께 실행 시작 대화 상자가 표시됩니다.

```
{
    "jobName": "my-job",{
  "topic": "arn:aws:sns:us-east-2:123456789012:StepFunctionsSample-TaskTimercc68840e-c3d3-42a8-911e-821b7ce248e5-SNSTopic-44UjcFxzhACT",
  "message": "HelloWorld",
  "timer_seconds": 10
}
```

- 실행 시작 대화 상자에서 다음을 수행합니다.

  1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

     > **Note**
     >
     > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

  2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

     데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

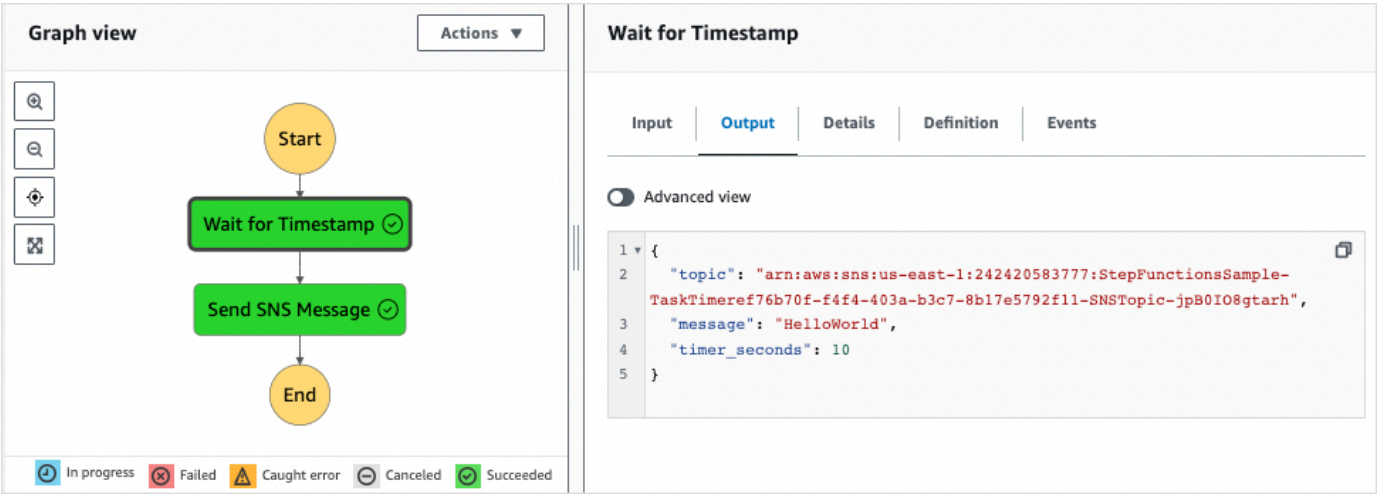
     > **Note**
     >
     > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

  3. 실행 시작을 선택합니다.

  4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

     실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예를 들어 다음 이미지에서는 선택한 타임스탬프 대기 단계의 출력을 보여줍니다. 이 단계의 출력은 입력으로 SNS 메시지 전송 단계로 전달됩니다.



콜백 패턴 예제(Amazon SQS, Amazon SNS, Lamda)

이 샘플 프로젝트는 작업을 AWS Step Functions 일시 중지하고 외부 프로세스가 작업 시작 시 생성된 작업 토큰을 반환할 때까지 기다리는 방법을 보여줍니다.

이 샘플 프로젝트를 배포하고 실행을 시작하면 다음 단계가 수행됩니다.

1. Step Functions는 작업 토큰이 포함된 메시지를 Amazon Simple Queue Service(Amazon SQS) 대기열에 전달합니다.
2. 그런 다음 Step Functions는 일시 중지하고 해당 토큰이 반환될 때까지 기다립니다.
3. Amazon SQS 대기열은 동일한 작업 `SendTaskSuccess`토큰으로 호출하는 AWS Lambda 함수를 트리거합니다.
4. 작업 토큰이 수신되면 워크플로우가 계속됩니다.
5. `"Notify Success"` 작업은 콜백을 수신했다는 Amazon Simple Notification Service(SNS) 메시지를 게시합니다.

Step Functions에서 콜백 패턴을 구현하는 방법은 작업 토큰을 사용하여 콜백 대기 섹션을 참조하세요.

다른 AWS 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

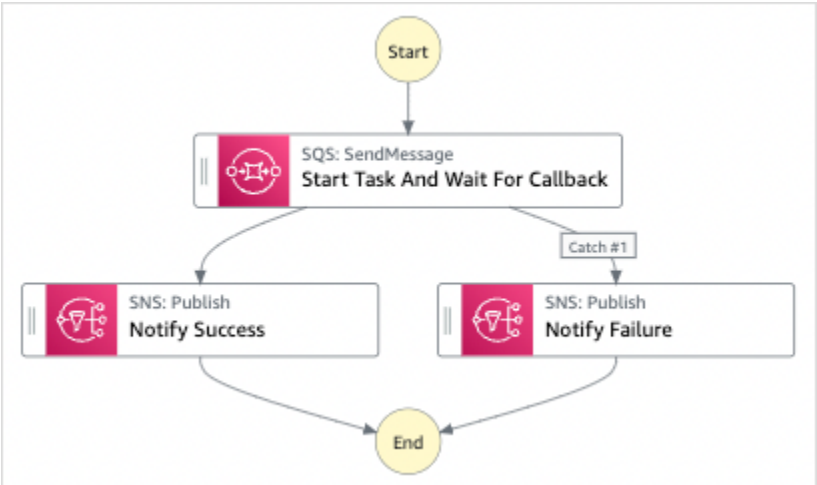
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Callback pattern example`**을 입력한 다음 반환된 검색 결과에서 콜백 패턴 예제를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon SQS 메시지 대기열
   - Step Functions API 작업을 호출하는 Lambda 함수입니다. SendTaskSuccess
   - 워크플로를 계속할 수 있는 여부를 나타내는 작업의 성공 또는 실패 여부를 알려주는 Amazon SNS 주제
   - 스테이트 AWS Step Functions 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 콜백 패턴 예제 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

- 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

  Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

  > **Important**
  >
  > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

  > **Tip**
  >
  > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

  준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

  이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

  샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

  > **Important**
  >
  > 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

      > **Note**
      >
      > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

      데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

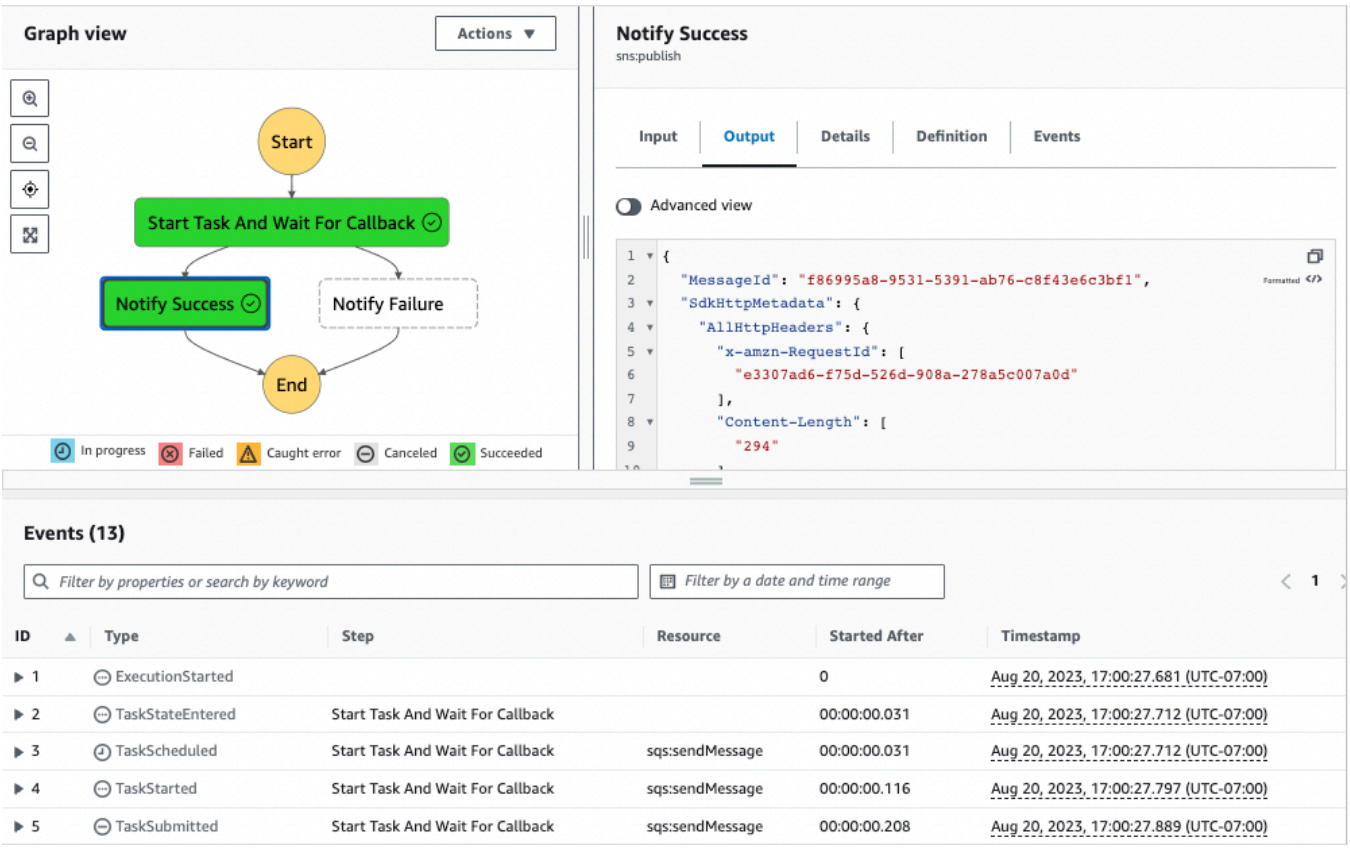
      > **Note**
      >
      > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

   3. 실행 시작을 선택합니다.
   4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

      실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예를 들어 Step Functions에서 워크플로를 어떻게 진행했고 Amazon SQS로부터 콜백을 수신했는지 검토하려면 이벤트의 항목을 검토합니다. 다음 이미지에서는 성공 알림 단계의 실행 출력을 보

여줍니다. 또한 실행 이벤트 내역의 처음 5개 이벤트도 보여줍니다. 각 이벤트를 확장하면 해당 이벤트에 대한 세부 정보를 볼 수 있습니다.



Lamda 콜백 예제

이 샘플 프로젝트의 구성 요소가 함께 작동하는 방식을 보려면 계정에 배포된 리소스를 확인하세요. AWS 예를 들어 다음은 작업 토큰을 사용하여 Step Functions를 직접적으로 호출하는 Lambda 함수입니다.

```
console.log('Loading function');
const aws = require('aws-sdk');

exports.lambda_handler = (event, context, callback) => {
    const stepfunctions = new aws.StepFunctions();

    for (const record of event.Records) {
        const messageBody = JSON.parse(record.body);
        const taskToken = messageBody.TaskToken;

        const params = {
```

```
            output: "\"Callback task completed successfully.\"",
            taskToken: taskToken
        };

        console.log(`Calling Step Functions to complete callback task with params
 ${JSON.stringify(params)}`);

        stepfunctions.sendTaskSuccess(params, (err, data) => {
            if (err) {
                console.error(err.message);
                callback(err.message);
                return;
            }
            console.log(data);
            callback(null);
        });
    }
};
```

Amazon EMR 작업 관리

이 샘플 프로젝트는 Amazon EMR AWS Step Functions 및 통합을 보여줍니다.

Amazon EMR 클러스터를 만들고 여러 단계를 추가하여 실행한 다음 클러스터를 종료하는 방법을 보여줍니다.

> **Important**
>
> Amazon EMR에는 무료 요금 티어가 없습니다. 샘플 프로젝트를 실행하면 비용이 발생합니다. Amazon EMR 요금 페이지에서 요금 정보를 확인할 수 있습니다. Amazon EMR 서비스 통합 가용성은 Amazon EMR API 가용성에 따라 달라질 수 있습니다. 이로 인해 일부 AWS 지역에서는 이 샘플 프로젝트가 제대로 작동하지 않을 수 있습니다. 특별 리전에서의 제한 사항은 Amazon EMR 설명서를 참조하세요.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

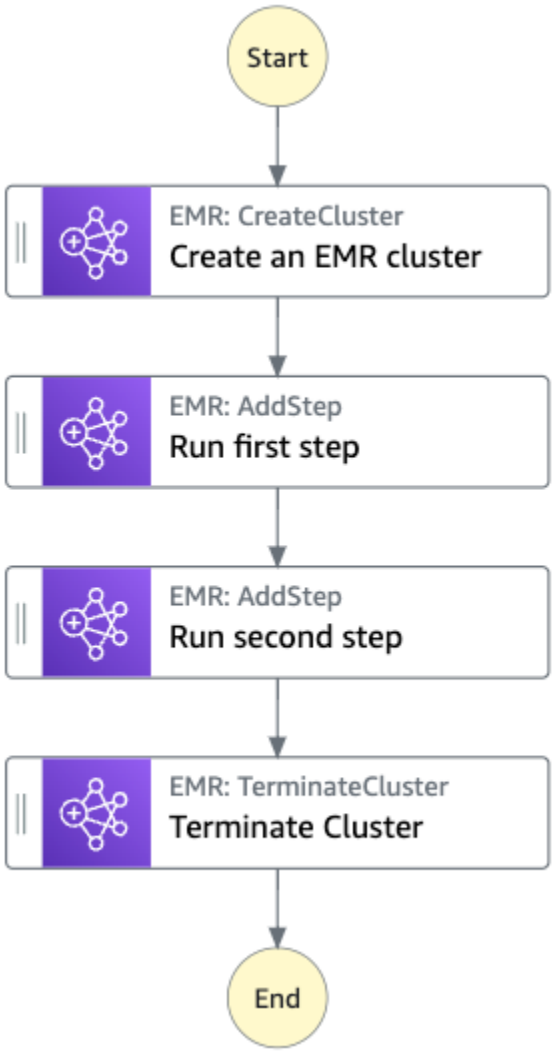
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **Manage an EMR job**을 입력한 다음 반환된 검색 결과에서 EMR 작업 관리를 선택합니다.

3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon S3 버킷
   - Amazon EMR 클러스터
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 EMR 작업 관리 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

파라미터를 리소스에 직접 해당 리소스에 전달하면 이 샘플 프로젝트의 상태 시스템은 Amazon EMR과 통합됩니다. 이 예제 상태 시스템을 살펴봐 Step Functions에서 상태 시스템을 사용하여 Amazon EMR 작업을 동기식으로 직접 호출하고 작업이 성공하거나 실패할 때까지 기다린 다음 클러스터를 종료하는 방식을 확인합니다.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "Comment": "An example of the Amazon States Language for running jobs on Amazon EMR",
```

```
  "StartAt": "Create an EMR cluster",
  "States": {
    "Create an EMR cluster": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::elasticmapreduce:createCluster.sync",
      "Parameters": {
        "Name": "ExampleCluster",
        "VisibleToAllUsers": true,
        "ReleaseLabel": "emr-5.26.0",
        "Applications": [
          { "Name": "Hive" }
        ],
        "ServiceRole": "<EMR_SERVICE_ROLE>",
        "JobFlowRole": "<EMR_EC2_INSTANCE_PROFILE>",
        "LogUri": "s3://<EMR_LOG_S3_BUCKET>/logs/",
        "Instances": {
          "KeepJobFlowAliveWhenNoSteps": true,
          "InstanceFleets": [
            {
              "Name": "MyMasterFleet",
              "InstanceFleetType": "MASTER",
              "TargetOnDemandCapacity": 1,
              "InstanceTypeConfigs": [
                {
                  "InstanceType": "m5.xlarge"
                }
              ]
            },
            {
              "Name": "MyCoreFleet",
              "InstanceFleetType": "CORE",
              "TargetOnDemandCapacity": 1,
              "InstanceTypeConfigs": [
                {
                  "InstanceType": "m5.xlarge"
                }
              ]
            }
          ]
        }
      },
      "ResultPath": "$.cluster",
      "Next": "Run first step"
    },
```

```
    "Run first step": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::elasticmapreduce:addStep.sync",
      "Parameters": {
        "ClusterId.$": "$.cluster.ClusterId",
        "Step": {
          "Name": "My first EMR step",
          "ActionOnFailure": "CONTINUE",
          "HadoopJarStep": {
            "Jar": "command-runner.jar",
            "Args": ["<COMMAND_ARGUMENTS>"]
          }
        }
      },
      "Retry" : [
        {
          "ErrorEquals": [ "States.ALL" ],
          "IntervalSeconds": 1,
          "MaxAttempts": 3,
          "BackoffRate": 2.0
        }
      ],
      "ResultPath": "$.firstStep",
      "Next": "Run second step"
    },
    "Run second step": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::elasticmapreduce:addStep.sync",
      "Parameters": {
        "ClusterId.$": "$.cluster.ClusterId",
        "Step": {
          "Name": "My second EMR step",
          "ActionOnFailure": "CONTINUE",
          "HadoopJarStep": {
            "Jar": "command-runner.jar",
            "Args": ["<COMMAND_ARGUMENTS>"]
          }
        }
      },
      "Retry" : [
        {
          "ErrorEquals": [ "States.ALL" ],
          "IntervalSeconds": 1,
          "MaxAttempts": 3,
```

```
          "BackoffRate": 2.0
        }
      ],
      "ResultPath": "$.secondStep",
      "Next": "Terminate Cluster"
    },
    "Terminate Cluster": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::elasticmapreduce:terminateCluster",
      "Parameters": {
        "ClusterId.$": "$.cluster.ClusterId"
      },
      "End": true
    }
  }
}
```

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 모든 권한만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:RunJobFlow",
                "elasticmapreduce:DescribeCluster",
                "elasticmapreduce:TerminateJobFlows"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::123456789012:role/StepFunctionsSample-EMRJobManagement-
EMRServiceRole-ANPAJ2UCCR6DPCEXAMPLE",
```

```
                "arn:aws:iam::123456789012:role/StepFunctionsSample-
EMRJobManagementWJALRXUTNFEMI-ANPAJ2UCCR6DPCEXAMPLE-
EMREc2InstanceProfile-1ANPAJ2UCCR6DPCEXAMPLE"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:sa-east-1:123456789012:rule/
StepFunctionsGetEventForEMRRunJobFlowRule"
            ]
        }
    ]
}
```

다음 정책은 addStep에 충분한 권한이 있는지 확인합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:AddJobFlowSteps",
                "elasticmapreduce:DescribeStep",
                "elasticmapreduce:CancelSteps"
            ],
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
```

```
                "arn:aws:events:sa-east-1:123456789012:rule/
StepFunctionsGetEventForEMRAddJobFlowStepsRule"
            ]
        }
    ]
}
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

EMR Serverless 작업 실행

이 샘플 프로젝트에서는 EMR Serverless 애플리케이션을 만들고 시작하는 방법을 보여줍니다. 또한 이 프로젝트에서는 해당 애플리케이션 내에서 여러 작업을 실행할 수 있는 방법을 보여줍니다.

이 샘플 프로젝트는 스테이트 머신과 지원 AWS 리소스를 생성하고 관련 IAM 권한을 구성합니다. 이 샘플 프로젝트를 살펴보고 Step Functions 상태 시스템을 사용하여 EMR Serverless 작업을 실행하는 방법을 알아보거나 자체 프로젝트의 시작점으로 사용합니다.

> **Important**
>
> EMR Serverless에는 무료 요금 티어가 없습니다. 샘플 프로젝트를 실행하면 비용이 발생합니다. 요금 정보는 Amazon EMR Serverless 요금 페이지에서 확인할 수 있습니다.
> 또한 EMR Serverless 서비스 통합 가용성은 EMR Serverless API 가용성에 따라 달라질 수 있습니다. 이로 인해 일부 AWS 리전에서는 이 샘플 프로젝트가 올바르게 작동하지 않거나 사용하지 못할 수 있습니다. AWS 리전의 EMR Serverless 가용성은 기타 고려 사항 주제를 참조하세요.

AWS CloudFormation 템플릿과 추가 리소스

CloudFormation 템플릿을 사용하여 이 샘플 프로젝트를 배포합니다. 이 템플릿은 다음과 같은 리소스를 사용자 사이트에 생성합니다. AWS 계정

- Step Functions 상태 시스템
- 상태 시스템에 대한 실행 역할. 이 역할은 상태 머신이 다른 AWS 서비스 리소스 (예: EMR Serverless CreateApplication작업) 에 액세스하는 데 필요한 권한을 부여합니다.

- EMR Serverless에 대한 작업 실행 역할. 이 역할은 사용자 대신 다른 서비스를 호출할 때 EMR Serverless 작업 실행에서 사용할 수 있도록 권한을 부여합니다.

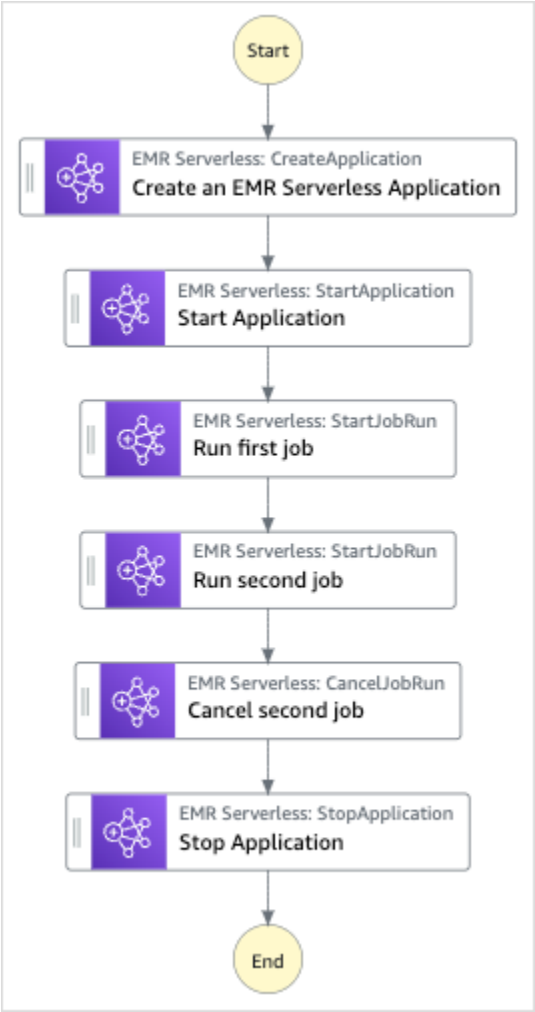
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`EMR Serverless`**를 입력한 다음 반환된 검색 결과에서 EMR Serverless 작업 실행을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Step Functions 상태 머신
   - 관련된 AWS Identity and Access Management(IAM) 역할

   다음 이미지에서는 EMR Serverless 작업 실행 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

> **Note**
>
> Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.
3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

워크플로에서 워크플로 시작(Step Functions, Lambda)

이 샘플 프로젝트는 스테이트 머신을 사용하여 다른 AWS Step Functions 스테이트 머신 실행을 시작하는 방법을 보여줍니다. 다른 상태 시스템에서 상태 시스템 실행을 시작하는 방법은 작업 상태에서 워크플로 실행 시작 섹션을 참조하세요.

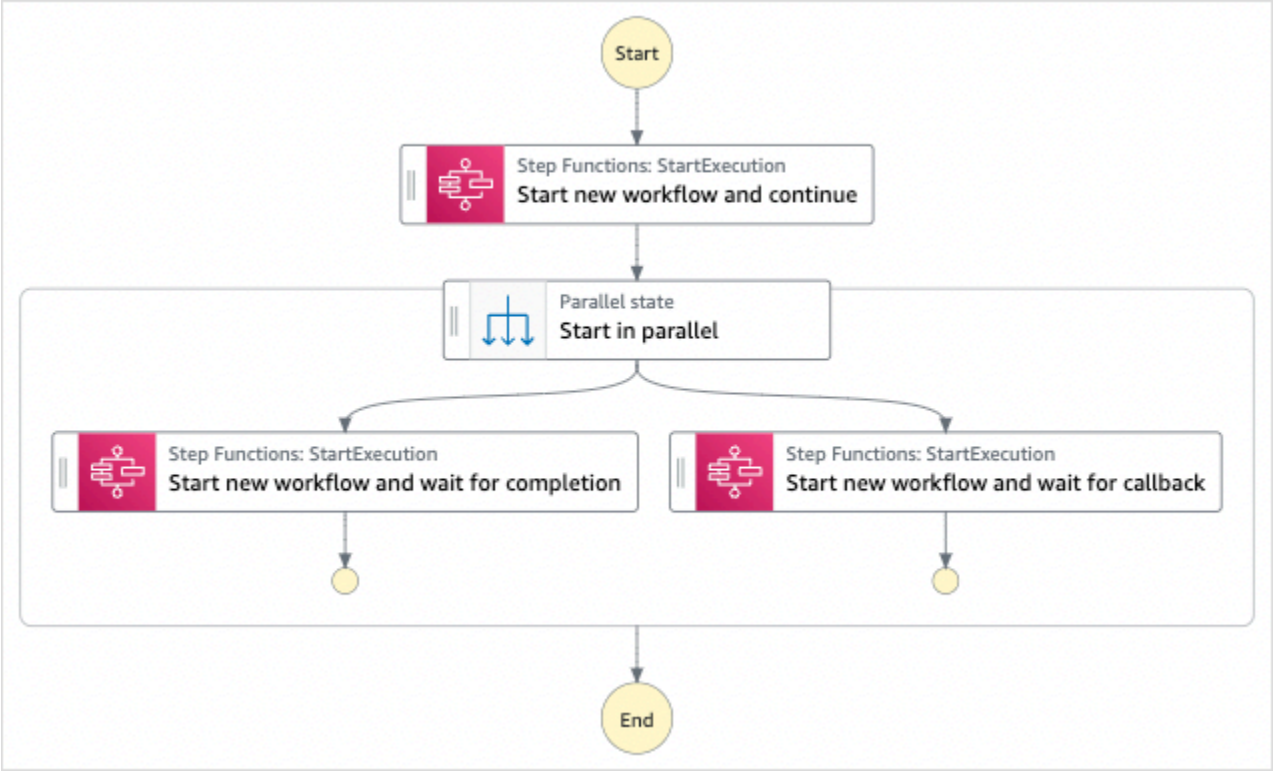
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Start a workflow within a workflow`**를 입력한 다음 반환된 검색 결과에서 워크플로에서 워크플로 시작을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

이 샘플 프로젝트는 다음 리소스를 배포합니다.

- 추가 상태 시스템. 이 상태 시스템 실행은 실행 중인 상태 시스템에서 시작됩니다.
- 콜백 Lambda 함수. 이 함수는 추가 상태 시스템에서 콜백 메커니즘을 구현하는 데 사용됩니다.
- AWS Step Functions 스테이트 머신
- 관련 AWS Identity and Access Management (IAM) 역할

다음 이미지에서는 워크플로에서 워크플로 시작 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

- 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

  Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 매개변수를 해당 리소스에 직접 AWS Lambda 전달하여 다른 상태 머신을 통합합니다.

이 예제 상태 시스템을 살펴보고 Step Functions에서 다른 상태 시스템에 대한 `StartExecution` API 작업을 직접적으로 호출하는 방법을 확인합니다. 다른 상태 머신의 두 인스턴스를 병렬로 시작합니다. 하나는 작업 실행(.sync) 패턴을 사용하고 다른 하나는 작업 토큰을 사용하여 콜백 대기 패턴을 사용합니다.

다른 AWS 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
  "Comment": "An example of combining workflows using a Step Functions StartExecution
 task state with various integration patterns.",
  "StartAt": "Start new workflow and continue",
  "States": {
    "Start new workflow and continue": {
      "Comment": "Start an execution of another Step Functions state machine and
 continue",
      "Type": "Task",
      "Resource": "arn:aws:states:::states:startExecution",
      "Parameters": {
        "StateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:NestingPatternAnotherStateMachine-HZ9gtgspmdun",
        "Input": {
          "NeedCallback": false,
          "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
        }
      },
      "Next": "Start in parallel"
    },
    "Start in parallel": {
      "Comment": "Start two executions of the same state machine in parallel",
      "Type": "Parallel",
      "End": true,
      "Branches": [
        {
          "StartAt": "Start new workflow and wait for completion",
          "States": {
            "Start new workflow and wait for completion": {
              "Comment": "Start an execution of the same
 'NestingPatternAnotherStateMachine' and wait for its completion",
              "Type": "Task",
              "Resource": "arn:aws:states:::states:startExecution.sync",
              "Parameters": {
                "StateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:NestingPatternAnotherStateMachine-HZ9gtgspmdun",
                "Input": {
                  "NeedCallback": false,
                  "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
                }
              },
              "OutputPath": "$.Output",
              "End": true
```

```
            }
          }
        },
        {
          "StartAt": "Start new workflow and wait for callback",
          "States": {
            "Start new workflow and wait for callback": {
              "Comment": "Start an execution and wait for it to call back with a task
 token",
              "Type": "Task",
              "Resource": "arn:aws:states:::states:startExecution.waitForTaskToken",
              "Parameters": {
                "StateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:NestingPatternAnotherStateMachine-HZ9gtgspmdun",
                "Input": {
                  "NeedCallback": true,
                  "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id",
                  "TaskToken.$": "$$.Task.Token"
                }
              },
              "End": true
            }
          }
        }
      ]
    }
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Map 상태를 사용하여 데이터 동적 처리

이 샘플 프로젝트는 맵 상태를 사용하여 동적 병렬화를 보여줍니다.

이 프로젝트에서 Step Functions는 AWS Lambda 함수를 사용하여 Amazon SQS 대기열에서 메시지를 가져와서 해당 메시지의 JSON 배열을 상태에 전달합니다. Map 대기열에 있는 메시지마다 상태 시스템은 메시지를 DynamoDB에 쓰고 다른 Lambda 함수를 간접적으로 호출하여 Amazon SQS에서 메시지를 제거한 다음 메시지를 Amazon SNS 주제에 게시합니다.

Map 상태 및 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 맵
- 다른 서비스와 AWS Step Functions 함께 사용

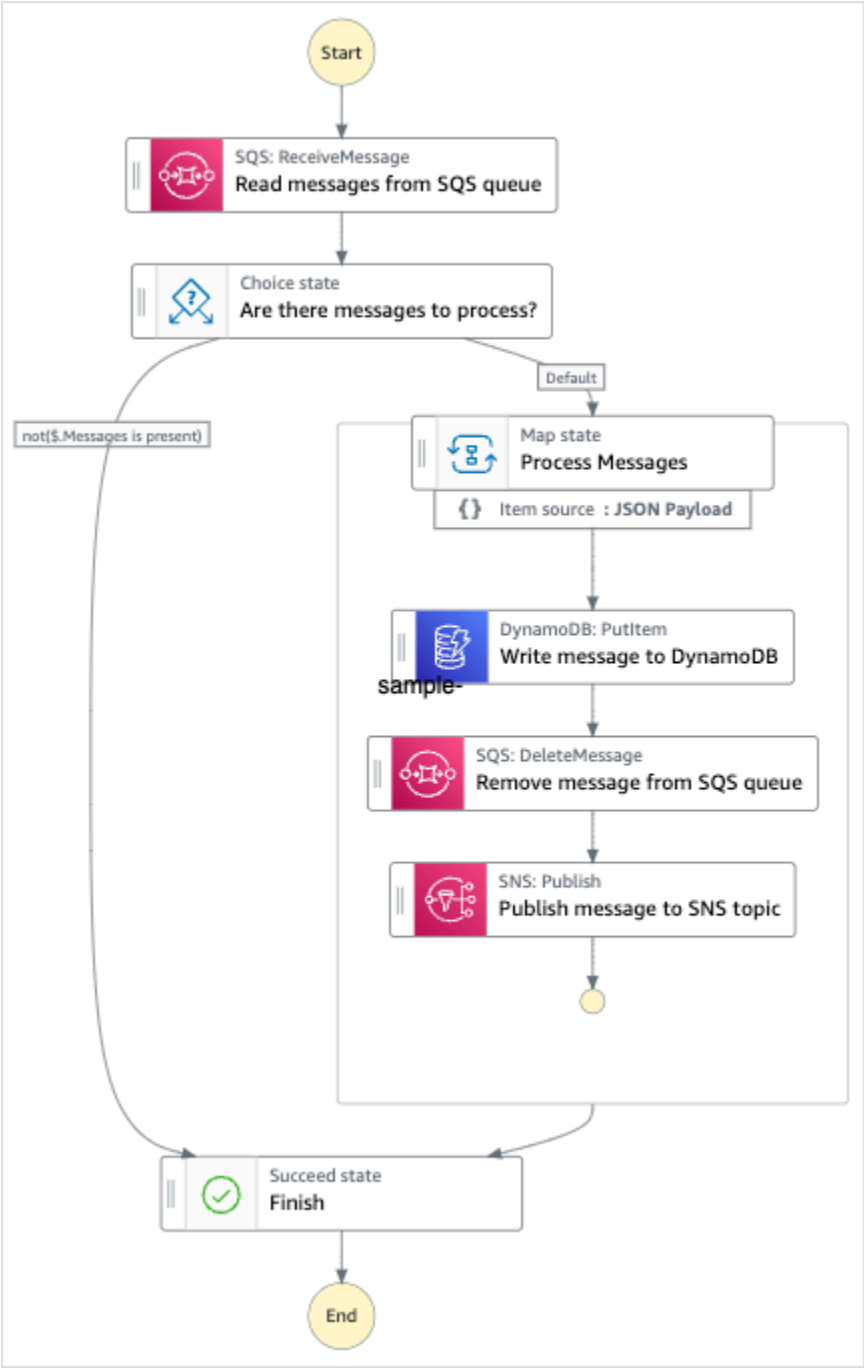
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Dynamically process data with a Map state`**를 입력한 다음 반환된 검색 결과에서 Map 상태에서 데이터를 동적으로 처리를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Map 상태에서 메시지를 반복적으로 읽고 제거하는 Amazon SQS 대기열
   - Map 상태에서 메시지를 반복적으로 쓰는 DynamoDB 테이블
   - Step Functions가 Amazon SQS 대기열에서 읽은 메시지를 게시하는 Amazon SNS 주제
   - 두 가지 AWS Lambda 기능
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 Map 상태에서 데이터를 동적으로 처리 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하

여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

샘플 프로젝트의 리소스가 배포되면 항목을 Amazon SQS 대기열에 추가하고 Amazon SNS 주제를 구독한 후에 상태 시스템을 실행해야 합니다.

2단계: Amazon SNS 주제 구독

1. Amazon SNS 콘솔을 엽니다.
2. 주제를 선택하고 `Map` 상태 샘플 프로젝트에서 생성한 주제를 선택합니다.

이름은 MapSampleProj-SNStoPic-1CQO4HQ3IR1kn과 비슷합니다.

3. 구독 생성을 선택합니다.

   구독 생성 페이지가 나타나며, 해당 주제에 대한 주제 ARN이 나열됩니다.

4. 프로토콜에서 이메일을 선택합니다.
5. 엔드포인트에 주제를 구독할 이메일 주소를 입력합니다.
6. 구독 생성을 선택합니다.

> **Note**
>
> 활성화되기 전에 이메일에서 구독을 확인합니다.

7. 관련된 계정에서 구독 확인 이메일을 열고 구독 확인 URL을 엽니다.

   구독이 확인됨 페이지가 표시됩니다.

3단계: Amazon SQS 대기열에 메시지 추가

1. Amazon SQS 콘솔을 엽니다.
2. Map 상태 샘플 프로젝트에서 생성한 대기열을 선택합니다.

   이름은 MapSampleProj-sqsqueue-1Udic9VZdorn7과 비슷할 것입니다.

3. [메시지 전송 및 수신(Send and receive messages)]을 선택합니다.
4. 메시지 전송 및 수신 페이지에서 메시지를 입력하고 메시지 전송을 선택합니다.
5. 다른 메시지를 입력하고 메시지 전송을 선택합니다. Amazon SQS 대기열에 메시지 여러 개가 있을 때까지 메시지를 계속 입력합니다.

4단계: 상태 시스템 실행

> **Note**
>
> Amazon SNS의 대기열은 일정하게 유지됩니다. 최상의 결과를 얻으려면 대기열을 채우고 상태 시스템을 실행하는 동안 잠시 기다리십시오.

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

      > **Note**
      >
      > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

      데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

      > **Note**
      >
      > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

   3. 실행 시작을 선택합니다.
   4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

      실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 시스템 코드

이 샘플 프로젝트의 상태 시스템은 파라미터를 리소스에 직접 전달하여 Amazon SQS, Amazon SNS 및 Lambda와 통합됩니다.

이 예제 상태 시스템을 살펴보고 `Resource` 필드의 Amazon 리소스 이름(ARN)과 연결하고 `Parameters`를 서비스 API에 전달하여 Step Functions에서 Lambda, DynamoDB 및 Amazon SNS를 제어하는 방법을 확인합니다.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "Comment": "An example of the Amazon States Language for reading messages from an SQS
 queue and iteratively processing each message.",
  "StartAt": "Read messages from SQS Queue",
  "States": {
    "Read messages from SQS Queue": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "MapSampleProj-ReadFromSQSQueueLambda-1MY3M63RMJVA9"
      },
      "Next": "Are there messages to process?"
    },
    "Are there messages to process?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$",
          "StringEquals": "No messages",
          "Next": "Finish"
        }
      ],
      "Default": "Process messages"
    },
    "Process messages": {
      "Type": "Map",
      "Next": "Finish",
      "ItemsPath": "$",
      "Parameters": {
        "MessageNumber.$": "$$.Map.Item.Index",
        "MessageDetails.$": "$$.Map.Item.Value"
      },
      "Iterator": {
        "StartAt": "Write message to DynamoDB",
        "States": {
```

```
          "Write message to DynamoDB": {
            "Type": "Task",
            "Resource": "arn:aws:states:::dynamodb:putItem",
            "ResultPath": null,
            "Parameters": {
              "TableName": "MapSampleProj-DDBTable-YJDJ1MKIN6C5",
              "ReturnConsumedCapacity": "TOTAL",
              "Item": {
                "MessageId": {
                  "S.$": "$.MessageDetails.MessageId"
                },
                "Body": {
                  "S.$": "$.MessageDetails.Body"
                }
              }
            },
            "Next": "Remove message from SQS queue"
          },
          "Remove message from SQS queue": {
            "Type": "Task",
            "Resource": "arn:aws:states:::lambda:invoke",
            "InputPath": "$.MessageDetails",
            "ResultPath": null,
            "Parameters": {
              "FunctionName": "MapSampleProj-DeleteFromSQSQueueLambda-198J2839ZO5K2",
              "Payload": {
                "ReceiptHandle.$": "$.ReceiptHandle"
              }
            },
            "Next": "Publish message to SNS topic"
          },
          "Publish message to SNS topic": {
            "Type": "Task",
            "Resource": "arn:aws:states:::sns:publish",
            "InputPath": "$.MessageDetails",
            "Parameters": {
              "Subject": "Message from Step Functions!",
              "Message.$": "$.Body",
              "TopicArn": "arn:aws:sns:us-east-1:012345678910:MapSampleProj-
SNSTopic-1CQO4HQ3IR1KN"
            },
            "End": true
          }
        }
```

```
      }
    },
    "Finish": {
      "Type": "Succeed"
    }
  }
}
```

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:012345678901:function:MapSampleProj-
ReadFromSQSQueueLambda-1MY3M63RMJVA9",
                "arn:aws:lambda:us-east-1:012345678901:function:MapSampleProj-
DeleteFromSQSQueueLambda-198J2839ZO5K2"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:012345678901:table/MapSampleProj-DDBTable-
YJDJ1MKIN6C5"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
```

```
            "Resource": [
                "arn:aws:sns:us-east-1:012345678901:MapSampleProj-
SNSTopic-1CQO4HQ3IR1KN"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Distributed Map을 사용하여 CSV 파일 처리

이 샘플 프로젝트에서는 Distributed Map 상태를 사용하여 Lambda 함수를 통해 생성된 CSV 파일의 행 10,000개 이상을 반복하는 방법을 보여줍니다. CSV 파일에는 고객 주문의 배송 정보가 포함되며 Amazon S3 버킷에 저장됩니다. Distributed Map은 데이터 분석을 위해 CSV 파일의 행 10개 배치를 반복합니다.

Distributed Map에는 지연된 주문을 감지하는 Lambda 함수가 포함되어 있습니다. 또한 Distributed Map에는 지연된 주문을 일괄 처리하고 이러한 지연된 주문을 배열로 반환하는 Inline Map도 포함되어 있습니다. 지연된 주문마다 Inline Map은 메시지를 Amazon SQS 대기열로 보냅니다. 마지막으로, 이 샘플 프로젝트는 맵 실행 결과를 AWS 계정의 다른 Amazon S3 버킷에 저장합니다.

Distributed Map을 사용하면 하위 워크플로 실행을 한 번에 최대 10,000개까지 동시에 실행할 수 있습니다. 이 샘플 프로젝트에서 Distributed Map의 최대 동시성은 1,000개로 설정되어 있으며 이 경우 하위 워크플로 동시 실행은 1,000개로 제한됩니다.

이 샘플 프로젝트는 스테이트 머신과 지원 AWS 리소스를 생성하고 관련 IAM 권한을 구성합니다. 이 샘플 프로젝트를 살펴보고 Distributed Map을 사용하여 대규모 병렬 워크로드를 오케스트레이션하는 방법을 알아보거나 자체 프로젝트의 시작점으로 사용합니다.

AWS CloudFormation 템플릿 및 추가 리소스

CloudFormation 템플릿을 사용하여 이 샘플 프로젝트를 배포합니다. 이 템플릿은 다음 리소스를 사용자 내에 생성합니다 AWS 계정.

- Step Functions 상태 시스템
- 상태 시스템에 대한 실행 역할. 이 역할은 상태 머신이 Lambda 함수의 Invoke 작업과 같은 다른 리소스 AWS 서비스 및 리소스에 액세스하는 데 필요한 권한을 부여합니다.

- 고객 주문 세부 정보가 포함된 CSV 파일을 생성하는 Lambda 함수 `CSVGeneratorFunction`
- CSV 생성기 Lambda 함수에 대한 실행 역할. 이 역할은 함수에 다른 항목에 액세스할 수 있는 권한을 부여합니다. AWS 서비스
- 생성된 CSV 파일을 저장하는 Amazon S3 입력 버킷
- CSV 파일 데이터를 분석하고 지연된 주문을 감지하는 지연된 주문 감지 Lambda 함수
- 지연된 주문 Lambda 함수에 대한 실행 역할. 이 역할은 함수에 다른 기능에 액세스할 수 있는 권한을 AWS 서비스 부여합니다.
- 고객 주문의 분석 결과를 저장하는 Amazon S3 출력 버킷
- Step Functions에서 지연된 모든 주문에 대해 메시지를 보내는 Amazon SQS 대기열. 이러한 메시지에는 고객 및 주문 ID가 포함됩니다.
- 상태 머신의 실행 기록과 관련된 정보를 저장하는 CloudWatch 로그 그룹입니다.

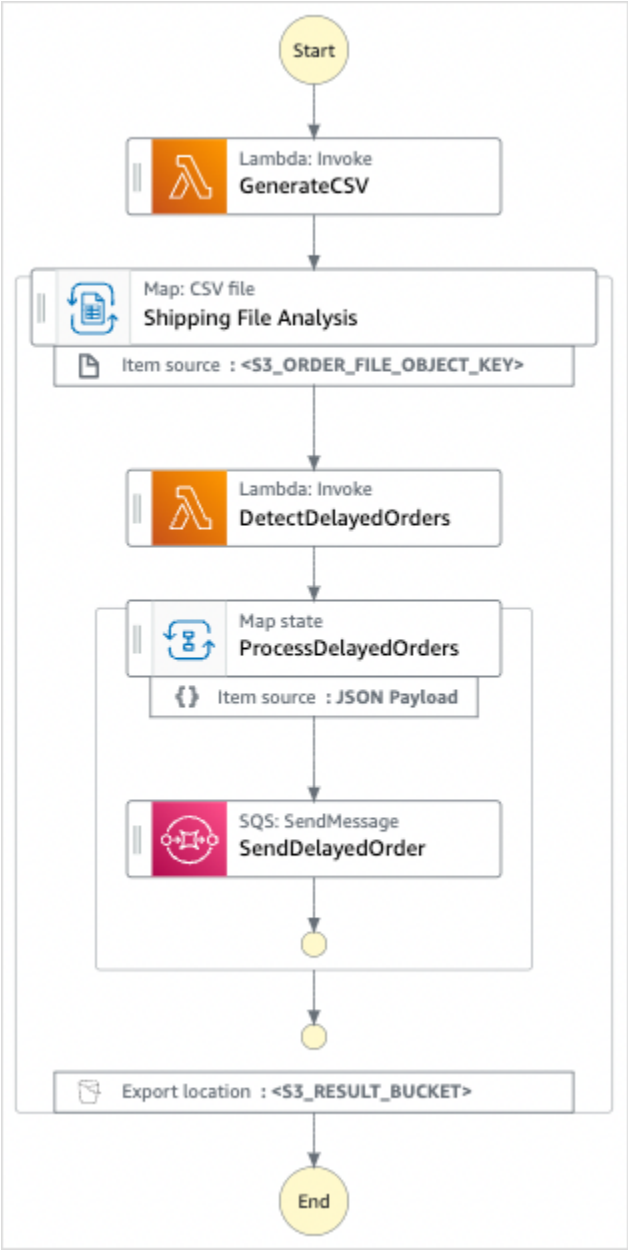
> **Important**
>
> 각 서비스마다 표준 요금이 적용됩니다.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Distributed Map to process a CSV file in S3`**를 입력한 다음 반환된 검색 결과에서 S3에서 CSV 파일을 처리하는 분산 맵을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트에 생성될 리소스에 대한 자세한 내용은 AWS CloudFormation 템플릿 및 추가 리소스를 참조하세요.

   다음 이미지에서 S3에서 CSV 파일을 처리하는 분산 맵 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행합니다.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트

합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

모든 리소스가 프로비저닝되고 배포된 후에 상태 시스템을 실행할 수 있습니다.

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 입력 값을 JSON 형식으로 입력하여 샘플 프로젝트를 실행합니다.

데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

> **Note**
>
> 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

b. 실행 시작을 선택합니다.

c. (선택 사항) Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행이 완료되면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의 등 각 상태의 세부 정보를 각각 봅니다.

   - 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요를 참조하세요.
   - 콘솔에서 Distributed Map 상태 실행을 보는 방법에 대한 자세한 내용은 맵 실행 검사를 참조하세요.

d. (선택 사항) Amazon S3 버킷으로 내보낸 실행 결과를 검토합니다. 이러한 결과에는 실행 입력 및 출력, ARN, 실행 상태와 같은 데이터가 포함됩니다. 자세한 내용은 ResultWriter을(를) 참조하세요.

Distributed Map을 사용하여 Amazon S3 버킷의 데이터 처리

이 샘플 프로젝트에서는 Distributed Map 상태를 사용하여 과거 날씨 데이터 분석 및 매월 지구상에서 평균 기온이 가장 높은 기상 관측소 식별과 같은 대규모 데이터를 처리하는 방법을 보여줍니다. 날씨 데이터는 12,000개가 넘는 CSV 파일에 기록되며 이러한 파일은 Amazon S3 버킷에 저장됩니다.

이 샘플 프로젝트에는 Distributed S3 copy NOA Data 및 ProcessNOAAData라는 Distributed Map 상태 2개가 포함되어 있습니다. 분산된 S3 사본 NOA 데이터는 이름이 지정된 noaa-gsod-pds퍼블릭 Amazon S3 버킷의 CSV 파일을 반복하여 사용자의 Amazon S3 버킷에 복사합니다. AWS 계정 ProcessNOAAData는 복사된 파일을 반복하며 온도 분석을 수행하는 Lambda 함수를 포함합니다.

샘플 프로젝트는 먼저 ListObjectsV2 API 작업을 호출하여 Amazon S3 버킷의 콘텐츠를 확인합니다. 이 직접 호출에 대한 응답으로 반환된 키 수를 기준으로 샘플 프로젝트는 다음 중 하나를 결정합니다.

- 키 수가 1보다 크거나 같으면 프로젝트는 ProcessNOAAData 상태로 전환됩니다. 이 분산 맵 상태에는 매월 평균 기온이 가장 높았던 기상 관측소를 찾는 TemperatureFunction이라는 Lambda 함수가 포함되어 있습니다. 이 함수는 키로 `year-month`가 있는 사전과 값으로 기상 관측소에 대한 정보가 포함된 사전을 반환합니다.
- 반환된 키 수가 1을 초과하지 않는 경우 분산 S3 사본 NOA 데이터 상태는 공용 버킷의 모든 객체를 noaa-gsod-pds나열하고 개별 객체를 계정의 다른 버킷에 100개씩 반복적으로 복사합니다. Inline Map은 객체를 반복 복사합니다.

  모든 객체가 복사되면 프로젝트는 날씨 데이터를 처리하기 위한 ProcessNOAAData 상태로 전환됩니다.

샘플 프로젝트는 최종적으로 Lambda 함수에서 반환된 결과를 최종 집계하고 결과를 테이블에 쓰는 리듀서 TemperatureFunction함수로 전환됩니다. Amazon DynamoDB

Distributed Map을 사용하면 하위 워크플로 실행을 한 번에 최대 10,000개까지 동시에 실행할 수 있습니다. 이 샘플 프로젝트에서 ProcessNOAAData Distributed Map의 최대 동시성은 3,000개로 설정되어 있으며 이 경우 하위 워크플로 동시 실행은 3,000개로 제한됩니다.

이 샘플 프로젝트는 스테이트 머신과 지원 AWS 리소스를 생성하고 관련 IAM 권한을 구성합니다. 이 샘플 프로젝트를 살펴보고 Distributed Map을 사용하여 대규모 병렬 워크로드를 오케스트레이션하는 방법을 알아보거나 자체 프로젝트의 시작점으로 사용합니다.

> **Important**
>
> 미국 동부(버지니아 북부) 리전에서만 이 샘플 프로젝트를 사용할 수 있습니다.

AWS CloudFormation 템플릿 및 추가 리소스

CloudFormation 템플릿을 사용하여 이 샘플 프로젝트를 배포합니다. 이 템플릿은 다음 리소스를 사용자 내에 생성합니다 AWS 계정.

- Step Functions 상태 시스템
- 상태 시스템에 대한 실행 역할. 이 역할은 상태 머신이 Lambda 함수의 Invoke 작업과 같은 다른 리소스 AWS 서비스 및 리소스에 액세스하는 데 필요한 권한을 부여합니다.
- `NOAADataBucket`이라는 Amazon S3 버킷. 이 버킷에는 날씨 데이터가 있는 CSV 파일이 포함되어 있습니다.

- 날씨 데이터의 최종 집계를 수행하고 결과를 Amazon DynamoDB 테이블에 기록하는 `ReducerFunction` Lambda 함수입니다.
- 리듀서 Lambda 함수에 대한 실행 역할. 이 역할은 함수에 다른 항목에 액세스할 수 있는 권한을 부여합니다. AWS 서비스
- 날씨 분석 결과를 저장하는 `ResultsBucket` Amazon S3 출력 버킷
- `ReducerFunction`에서 반환한 결과가 포함된 `ResultsDynamoDBTable` DynamoDB 테이블
- 가장 높은 월 평균 기온을 찾는 `TemperatureFunction` Lambda 함수
- Lambda 함수에 대한 실행 역할. 이 역할은 함수에 다른 기능에 액세스할 수 있는 권한을 AWS 서비스 부여합니다.
- 상태 머신의 실행 기록과 관련된 정보를 저장하는 CloudWatch 로그 그룹입니다.

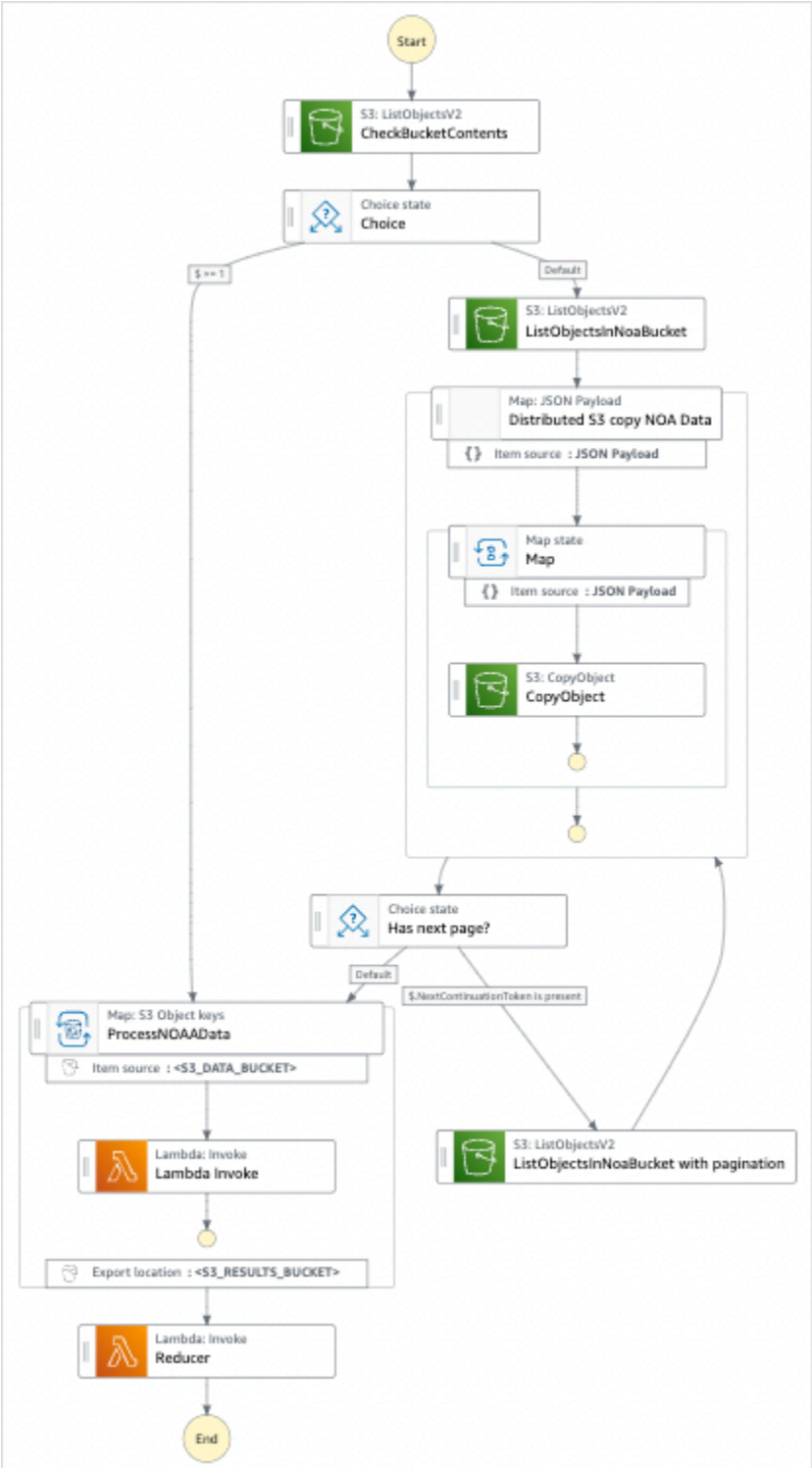
> **Important**
>
> 각 서비스마다 표준 요금이 적용됩니다.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Distributed Map to process files in S3`**를 입력한 다음 반환된 검색 결과에서 S3에서 파일을 처리하는 분산 맵을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트에 생성될 리소스에 대한 자세한 내용은 AWS CloudFormation 템플릿 및 추가 리소스를 참조하세요.

   다음 이미지에서 S3에서 파일을 처리하는 분산 맵 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행합니다.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

모든 리소스가 프로비저닝되고 배포된 후에 상태 시스템을 실행할 수 있습니다.

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.

2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   a. (선택 사항) 입력 값을 JSON 형식으로 입력하여 샘플 프로젝트를 실행합니다.

      데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

      > **Note**
      >
      > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

   b. 실행 시작을 선택합니다.

   c. (선택 사항) Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

      실행이 완료되면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의 등 각 상태의 세부 정보를 각각 봅니다.

      - 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요를 참조하세요.
      - 콘솔에서 Distributed Map 상태 실행을 보는 방법에 대한 자세한 내용은 맵 실행 검사를 참조하세요.

   d. (선택 사항) Amazon S3 버킷으로 내보낸 실행 결과를 검토합니다. 이러한 결과에는 실행 입력 및 출력, ARN, 실행 상태와 같은 데이터가 포함됩니다. 자세한 내용은 ResultWriter을(를) 참조하세요.

기계 학습 모델 훈련

이 샘플 프로젝트는 기계 학습 모델을 사용하고 SageMaker 학습시키는 방법과 테스트 데이터세트를 일괄 변환하는 방법을 보여줍니다. AWS Step Functions

이 프로젝트에서 Step Functions는 Lambda 함수를 사용하여 테스트 데이터 세트로 Amazon S3 버킷을 시딩합니다. 그런 다음 SageMaker 서비스 통합을 사용하여 기계 학습 모델을 학습시키고 일괄 변환을 수행합니다.

Step Functions 서비스 통합에 대한 SageMaker 자세한 내용은 다음을 참조하십시오.

- 다른 서비스와 AWS Step Functions 함께 사용
- Step SageMaker Functions를 사용한 관리

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 SageMaker 요금을 참조하십시오.

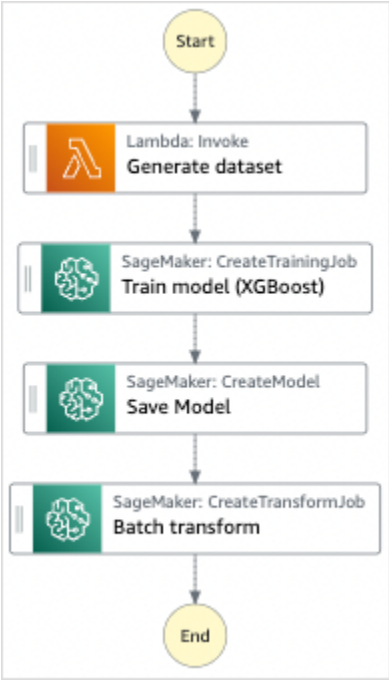
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Train a machine learning model`**을 입력한 다음 반환된 검색 결과에서 기계 학습 모델 학습시키기를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - AWS Lambda 함수
   - Amazon Simple Storage Service(S3) 버킷
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 기계 학습 모델 학습시키기 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

   - 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

        > **Note**
        >
        > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

    2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

        데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

> **Note**
>
> 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 해당 리소스에 파라미터를 직접 AWS Lambda 전달하여 통합하고, Amazon S3 버킷을 교육 데이터 소스 및 출력으로 SageMaker 사용합니다.

이 예제 상태 머신을 살펴보고 Step Functions가 Lambda SageMaker 및 를 제어하는 방법을 살펴보세요.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "StartAt": "Generate dataset",
  "States": {
    "Generate dataset": {
      "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:TrainAndBatchTransform-SeedingFunction-17RNSOTG97HPV",
      "Type": "Task",
      "Next": "Train model (XGBoost)"
    },
    "Train model (XGBoost)": {
      "Resource": "arn:aws:states:::sagemaker:createTrainingJob.sync",
      "Parameters": {
        "AlgorithmSpecification": {
```

```
          "TrainingImage": "433757028032.dkr.ecr.us-west-2.amazonaws.com/
xgboost:latest",
          "TrainingInputMode": "File"
        },
        "OutputDataConfig": {
          "S3OutputPath": "s3://trainandbatchtransform-s3bucket-1jn1le6gadwfz/models"
        },
        "StoppingCondition": {
          "MaxRuntimeInSeconds": 86400
        },
        "ResourceConfig": {
          "InstanceCount": 1,
          "InstanceType": "ml.m4.xlarge",
          "VolumeSizeInGB": 30
        },
        "RoleArn": "arn:aws:iam::123456789012:role/TrainAndBatchTransform-
SageMakerAPIExecutionRole-Y9IX3DLF6EUO",
        "InputDataConfig": [
          {
            "DataSource": {
              "S3DataSource": {
                "S3DataDistributionType": "ShardedByS3Key",
                "S3DataType": "S3Prefix",
                "S3Uri": "s3://trainandbatchtransform-s3bucket-1jn1le6gadwfz/csv/
train.csv"
              }
            },
            "ChannelName": "train",
            "ContentType": "text/csv"
          }
        ],
        "HyperParameters": {
          "objective": "reg:logistic",
          "eval_metric": "rmse",
          "num_round": "5"
        },
        "TrainingJobName.$": "$$.Execution.Name"
      },
      "Type": "Task",
      "Next": "Save Model"
    },
    "Save Model": {
      "Parameters": {
        "PrimaryContainer": {
```

```
          "Image": "433757028032.dkr.ecr.us-west-2.amazonaws.com/xgboost:latest",
          "Environment": {},
          "ModelDataUrl.$": "$.ModelArtifacts.S3ModelArtifacts"
        },
        "ExecutionRoleArn": "arn:aws:iam::123456789012:role/TrainAndBatchTransform-
SageMakerAPIExecutionRole-Y9IX3DLF6EUO",
        "ModelName.$": "$.TrainingJobName"
      },
      "Resource": "arn:aws:states:::sagemaker:createModel",
      "Type": "Task",
      "Next": "Batch transform"
    },
    "Batch transform": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sagemaker:createTransformJob.sync",
      "Parameters": {
        "ModelName.$": "$$.Execution.Name",
        "TransformInput": {
          "CompressionType": "None",
          "ContentType": "text/csv",
          "DataSource": {
            "S3DataSource": {
              "S3DataType": "S3Prefix",
              "S3Uri": "s3://trainandbatchtransform-s3bucket-1jn1le6gadwfz/csv/
test.csv"
            }
          }
        },
        "TransformOutput": {
          "S3OutputPath": "s3://trainandbatchtransform-s3bucket-1jn1le6gadwfz/output"
        },
        "TransformResources": {
          "InstanceCount": 1,
          "InstanceType": "ml.m4.xlarge"
        },
        "TransformJobName.$": "$$.Execution.Name"
      },
      "End": true
    }
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

다음 정책을 통해 Lambda 함수에서 샘플 데이터를 사용하여 Amazon S3 버킷을 시딩할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
```

```
            "Resource": "arn:aws:s3:::trainandbatchtransform-s3bucket-1jn1le6gadwfz/*",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

기계 학습 모델 튜닝

이 샘플 프로젝트는 머신 러닝 모델의 하이퍼파라미터를 튜닝하고 테스트 데이터세트를 일괄 변환하는 데 사용하는 SageMaker 방법을 보여줍니다.

이 프로젝트에서 Step Functions는 Lambda 함수를 사용하여 테스트 데이터 세트로 Amazon S3 버킷을 시딩합니다. 그런 다음 서비스 통합을 사용하여 하이퍼파라미터 조정 작업을 생성합니다. SageMaker 그런 다음 Lambda 함수를 사용하여 데이터 경로를 추출하고, 튜닝 모델을 저장하고, 모델 이름을 추출한 다음, 일괄 변환 작업을 실행하여 추론을 수행합니다. SageMaker

Step Functions 서비스 통합에 대한 SageMaker 자세한 내용은 다음을 참조하십시오.

- 다른 서비스와 AWS Step Functions 함께 사용
- Step SageMaker Functions를 사용한 관리

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 SageMaker요금을 참조하십시오.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

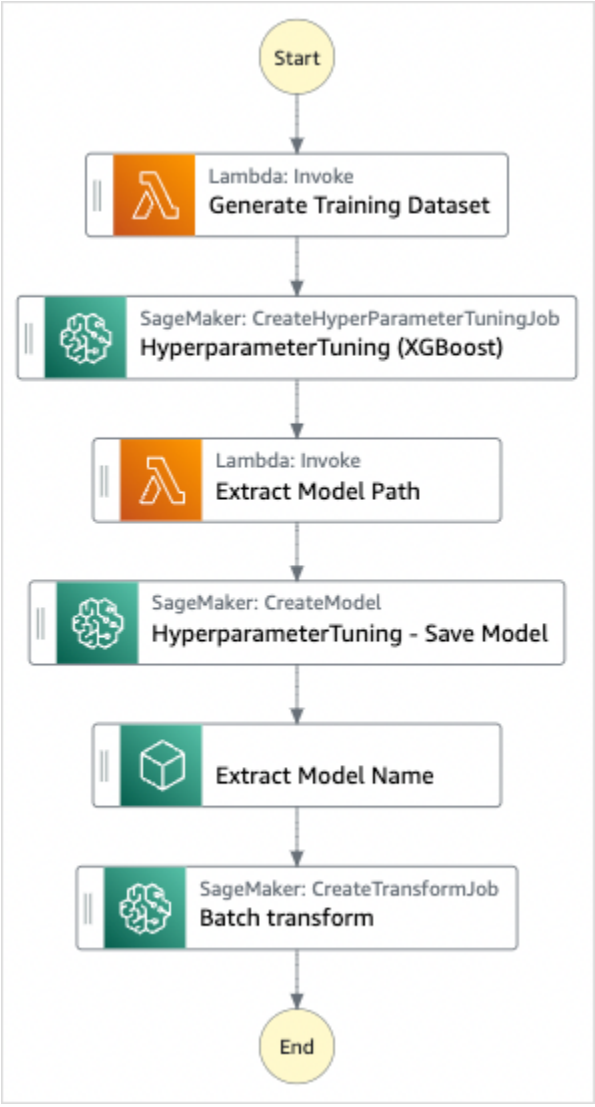
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Tune a machine learning model`**을 입력한 다음 반환된 검색 결과에서 기계 학습 모델 조정을 선택합니다.

3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - 세 가지 AWS Lambda 기능
   - Amazon Simple Storage Service(S3) 버킷
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 기계 학습 모델 조정 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 해당 리소스에 파라미터를 직접 AWS Lambda 전달하여 통합하고, Amazon S3 버킷을 교육 데이터 소스 및 출력으로 SageMaker 사용합니다.

이 예제 상태 머신을 살펴보고 Step Functions가 Lambda SageMaker 및 를 제어하는 방법을 살펴보세요.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
```

```
    "StartAt": "Generate Training Dataset",
    "States": {
        "Generate Training Dataset": {
            "Resource": "arn:aws:lambda:us-
west-2:012345678912:function:StepFunctionsSample-SageMa-
LambdaForDataGeneration-1TF67BUE5A12U",
            "Type": "Task",
            "Next": "HyperparameterTuning (XGBoost)"
        },
        "HyperparameterTuning (XGBoost)": {
            "Resource":
 "arn:aws:states:::sagemaker:createHyperParameterTuningJob.sync",
            "Parameters": {
                "HyperParameterTuningJobName.$": "$.body.jobName",
                "HyperParameterTuningJobConfig": {
                    "Strategy": "Bayesian",
                    "HyperParameterTuningJobObjective": {
                        "Type": "Minimize",
                        "MetricName": "validation:rmse"
                    },
                    "ResourceLimits": {
                        "MaxNumberOfTrainingJobs": 2,
                        "MaxParallelTrainingJobs": 2
                    },
                    "ParameterRanges": {
                        "ContinuousParameterRanges": [{
                                "Name": "alpha",
                                "MinValue": "0",
                                "MaxValue": "1000",
                                "ScalingType": "Auto"
                            },
                            {
                                "Name": "gamma",
                                "MinValue": "0",
                                "MaxValue": "5",
                                "ScalingType": "Auto"
                            }
                        ],
                        "IntegerParameterRanges": [{
                                "Name": "max_delta_step",
                                "MinValue": "0",
                                "MaxValue": "10",
                                "ScalingType": "Auto"
                            },
```

```
                            {
                                "Name": "max_depth",
                                "MinValue": "0",
                                "MaxValue": "10",
                                "ScalingType": "Auto"
                            }
                        ]
                    }
                },
                "TrainingJobDefinition": {
                    "AlgorithmSpecification": {
                        "TrainingImage": "433757028032.dkr.ecr.us-west-2.amazonaws.com/
xgboost:latest",
                        "TrainingInputMode": "File"
                    },
                    "OutputDataConfig": {
                        "S3OutputPath": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/models"
                    },
                    "StoppingCondition": {
                        "MaxRuntimeInSeconds": 86400
                    },
                    "ResourceConfig": {
                        "InstanceCount": 1,
                        "InstanceType": "ml.m4.xlarge",
                        "VolumeSizeInGB": 30
                    },
                    "RoleArn": "arn:aws:iam::012345678912:role/StepFunctionsSample-
SageM-SageMakerAPIExecutionRol-1MNH1VS5CGGOG",
                    "InputDataConfig": [{
                        "DataSource": {
                            "S3DataSource": {
                                "S3DataDistributionType": "FullyReplicated",
                                "S3DataType": "S3Prefix",
                                "S3Uri": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/csv/train.csv"
                            }
                        },
                        "ChannelName": "train",
                        "ContentType": "text/csv"
                    },
                    {
                        "DataSource": {
                            "S3DataSource": {
```

```
                                "S3DataDistributionType": "FullyReplicated",
                                "S3DataType": "S3Prefix",
                                "S3Uri": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/csv/validation.csv"
                            }
                        },
                        "ChannelName": "validation",
                        "ContentType": "text/csv"
                    }],
                    "StaticHyperParameters": {
                        "precision_dtype": "float32",
                        "num_round": "2"
                    }
                }
            },
            "Type": "Task",
            "Next": "Extract Model Path"
        },
        "Extract Model Path": {
            "Resource": "arn:aws:lambda:us-
west-2:012345678912:function:StepFunctionsSample-SageM-LambdaToExtractModelPath-
V0R37CVARUS9",
            "Type": "Task",
            "Next": "HyperparameterTuning - Save Model"
        },
        "HyperparameterTuning - Save Model": {
            "Parameters": {
                "PrimaryContainer": {
                    "Image": "433757028032.dkr.ecr.us-west-2.amazonaws.com/
xgboost:latest",
                    "Environment": {},
                    "ModelDataUrl.$": "$.body.modelDataUrl"
                },
                "ExecutionRoleArn": "arn:aws:iam::012345678912:role/
StepFunctionsSample-SageM-SageMakerAPIExecutionRol-1MNH1VS5CGGOG",
                "ModelName.$": "$.body.bestTrainingJobName"
            },
            "Resource": "arn:aws:states:::sagemaker:createModel",
            "Type": "Task",
            "Next": "Extract Model Name"
        },
        "Extract Model Name": {
```

```
            "Resource": "arn:aws:lambda:us-
west-2:012345678912:function:StepFunctionsSample-SageM-
LambdaToExtractModelName-8FUOB30SM5EM",
            "Type": "Task",
            "Next": "Batch transform"
        },
        "Batch transform": {
            "Type": "Task",
            "Resource": "arn:aws:states:::sagemaker:createTransformJob.sync",
            "Parameters": {
                "ModelName.$": "$.body.jobName",
                "TransformInput": {
                    "CompressionType": "None",
                    "ContentType": "text/csv",
                    "DataSource": {
                        "S3DataSource": {
                            "S3DataType": "S3Prefix",
                            "S3Uri": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/csv/test.csv"
                        }
                    }
                },
                "TransformOutput": {
                    "S3OutputPath": "s3://stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/output"
                },
                "TransformResources": {
                    "InstanceCount": 1,
                    "InstanceType": "ml.m4.xlarge"
                },
                "TransformJobName.$": "$.body.jobName"
            },
            "End": true
        }
    }
}
```

다른 AWS 서비스와 함께 Step Functions를 사용할 때 IAM을 구성하는 방법은 통합 서비스용 IAM 정책 섹션을 참조하세요.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

다음 IAM 정책은 상태 머신에 연결되며, 상태 머신 실행에서 필요한 SageMaker Lambda 및 Amazon S3 리소스에 액세스할 수 있도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sagemaker:CreateHyperParameterTuningJob",
                "sagemaker:DescribeHyperParameterTuningJob",
                "sagemaker:StopHyperParameterTuningJob",
                "sagemaker:ListTags",
                "sagemaker:CreateModel",
                "sagemaker:CreateTransformJob",
                "iam:PassRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-west-2:012345678912:function:StepFunctionsSample-
SageMa-LambdaForDataGeneration-1TF67BUE5A12U",
                "arn:aws:lambda:us-west-2:012345678912:function:StepFunctionsSample-
SageM-LambdaToExtractModelPath-V0R37CVARUS9",
                "arn:aws:lambda:us-west-2:012345678912:function:StepFunctionsSample-
SageM-LambdaToExtractModelName-8FUOB30SM5EM"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
```

```
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:*:*:rule/
StepFunctionsGetEventsForSageMakerTrainingJobsRule",
                "arn:aws:events:*:*:rule/
StepFunctionsGetEventsForSageMakerTransformJobsRule",
                "arn:aws:events:*:*:rule/
StepFunctionsGetEventsForSageMakerTuningJobsRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

다음 IAM 정책은 HyperparameterTuning 상태의 TrainingJobDefinition 및 HyperparameterTuning 필드에서 참조됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "sagemaker:DescribeHyperParameterTuningJob",
                "sagemaker:StopHyperParameterTuningJob",
                "sagemaker:ListTags"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
```

```
            ],
            "Resource": "arn:aws:s3:::stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f",
            "Effect": "Allow"
        }
    ]
}
```

다음 IAM 정책을 통해 Lambda 함수에서 샘플 데이터를 사용하여 Amazon S3 버킷을 시딩할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::stepfunctionssample-sagemak-
bucketformodelanddata-80fblmdlcs9f/*",
            "Effect": "Allow"
        }
    ]
}
```

다른 AWS 서비스와 함께 Step Functions를 사용할 때 IAM을 구성하는 방법은 통합 서비스용 IAM 정책 섹션을 참조하세요.

Amazon SQS에서 대용량 메시지 처리(Express 워크플로)

이 샘플 프로젝트는 AWS Step Functions 익스프레스 워크플로를 사용하여 Amazon Simple Queue Service (Amazon SQS) 와 같은 대용량 이벤트 소스의 메시지 또는 데이터를 처리하는 방법을 보여줍

니다. Express 워크플로는 매우 빠른 속도로 시작할 수 있기 때문에 대용량 이벤트 처리 또는 스트리밍 데이터 워크로드에 적합합니다.

다음은 이벤트 소스에서 상태 머신을 실행하는 데 일반적으로 사용되는 두 가지 방법입니다.

- 이벤트 소스에서 CloudWatch 이벤트가 발생할 때마다 상태 머신 실행을 시작하도록 Amazon Events 규칙을 구성합니다. 자세한 내용은 이벤트를 트리거하는 CloudWatch 이벤트 규칙 생성을 참조하십시오.
- 이벤트 소스를 Lambda 함수에 매핑하고 함수 코드를 작성하여 상태 머신을 실행합니다. 이벤트 소스가 이벤트를 내보낼 때마다 이 AWS Lambda 함수가 호출되어 스테이트 머신 실행이 시작됩니다. 자세한 내용은 Amazon SQS에서 AWS Lambda 사용을 참조하세요.

이 샘플 프로젝트는 두 번째 방법을 사용하여 Amazon SQS 대기열에서 메시지를 보낼 때마다 실행을 시작합니다. 유사한 구성을 사용하여 Amazon Simple Storage Service(S3), Amazon DynamoDB 및 Amazon Kinesis와 같은 다른 이벤트 소스에서 Express 워크플로 실행을 트리거할 수 있습니다.

Express 워크플로 및 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 표준 워크플로와 Express 워크플로 비교
- 다른 서비스와 AWS Step Functions 함께 사용
- 할당량

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

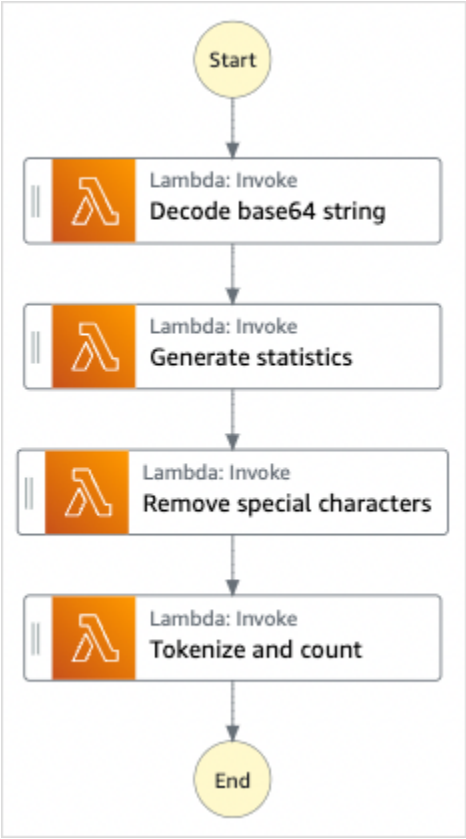
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Process high-volume messages from SQS`**를 입력한 다음 반환된 검색 결과에서 SQS에서 대용량 메시지 처리를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Lambda 함수 4개
   - Amazon SQS 대기열

- AWS Step Functions 스테이트 머신
- 관련 AWS Identity and Access Management (IAM) 역할

다음 이미지에서는 SQS에서 대용량 메시지 처리 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행합니다.

- 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

  Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행 트리거

1. Amazon SQS 콘솔을 엽니다.
2. 샘플 프로젝트에서 생성한 대기열을 선택합니다.

   이름은 Example-SQSQueue-wJalrXUtnFEMI와 유사합니다.
3. 대기열 작업 목록에서 메시지 전송을 선택합니다.
4. 복사 버튼을 사용하여 다음 메시지를 복사한 다음 메시지 전송 창에 입력하고 메시지 전송을 선택합니다.

> **Note**
>
> 이 샘플 메시지에서 `input:` 줄은 페이지에 맞게 줄바꿈을 사용하여 서식이 지정되었습니다. [복사] 버튼을 사용합니다. 또는 줄바꿈 없이 한 줄로 입력되었는지 확인합니다.

```
{
      "input":
 "QW5kIGxpa2UgdGhlIGJhc2VsZXNzIGZhYnJpYyBvZiB0aGlzIHZpc2lvbiwgVGhlIGNsb3VkLWNhcHBlZCB0b3dlc

  91cyBwYWxhY2VzLCBUaGUgc29sZW1uIHRlbXBsZXMsIHRoZSBncmVhdCBnbG9iZSBpdHNlbGbigJQgWWVhLCBhbGw

  ZXJpdOKAlHNoYWxsIGRpc3NvbHZlLCBBbmQgbGlrZSB0aGlzIGluc3Vic3RhbnRpYWwgcGFnZWFudCBmYWRlZCwgT

  FjayBiZWhpbmQuIFdlIGFyZSBzdWNoIHN0dWZmIEFzIGRyZWFtcyBhcmUgbWFkZSBvbiwgYW5kIG91ciBsaXR0bGU

  ZGVkIHdpdGggYSBzbGVlcC4gU2lyLCBJIGFtIHZleGVkLiBCZWFyIHdpdGggbXkgd2Vha25lc3MuIE15IG9sZCBic

  xlZC4gQmUgbm90IGRpc3R1cmJlZCB3aXRoIG15IGluZmlybWl0eS4gSWYgeW91IGJlIHBsZWFzZWQsIHJldGlyZSB

  QW5kIHRoZXJlIHJlcG9zZS4gQSB0dXJuIG9yIHR3byBJ4oCZbGwgd2FsayBUbyBzdGlsbCBteSBiZWF0aW5nIG1pb
}
```

5. 닫기를 선택하세요.
6. Step Functions 콘솔을 엽니다.
7. Amazon CloudWatch Logs 로그 그룹으로 이동하여 로그를 검사하십시오. 로그 그룹 이름은 예시-ExpressLogGroup -WJALRXUTNFEMI와 같이 표시됩니다.

Lambda 함수 코드 예제

다음은 시작 Lambda 함수가 SDK를 사용하여 상태 머신 실행을 시작하는 방법을 보여주는 Lambda 함수 코드입니다. AWS

```
import boto3

def lambda_handler(event, context):
    message_body = event['Records'][0]['body']
    client = boto3.client('stepfunctions')
    response = client.start_execution(
```

```
        stateMachineArn='${ExpressStateMachineArn}',
        input=message_body
    )
```

예제 상태 머신 코드

이 샘플 프로젝트의 Express 워크플로는 텍스트 처리를 위한 Lambda 함수 집합으로 구성됩니다.

다른 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. AWS 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "Comment": "An example of using Express workflows to run text processing for each
 message sent from an SQS queue.",
  "StartAt": "Decode base64 string",
  "States": {
    "Decode base64 string": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<BASE64_DECODER_LAMBDA_FUNCTION_NAME>",
        "Payload.$": "$"
      },
      "Next": "Generate statistics"
    },
    "Generate statistics": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<TEXT_STATS_GENERATING_LAMBDA_FUNCTION_NAME>",
        "Payload.$": "$"
      },
      "Next": "Remove special characters"
    },
    "Remove special characters": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<STRING_CLEANING_LAMBDA_FUNCTION_NAME>",
        "Payload.$": "$"
```

```
      },
      "Next": "Tokenize and count"
    },
    "Tokenize and count": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<TOKENIZING_AND_WORD_COUNTING_LAMBDA_FUNCTION_NAME>",
        "Payload.$": "$"
      },
      "End": true
    }
  }
}
```

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:123456789012:function:example-
Base64DecodeLambda-wJalrXUtnFEMI",
                "arn:aws:lambda:us-east-1:123456789012:function:example-
StringCleanerLambda-je7MtGbClwBF",
                "arn:aws:lambda:us-east-1:123456789012:function:example-
TokenizerCounterLambda-wJalrXUtnFEMI",
                "arn:aws:lambda:us-east-1:123456789012:function:example-
GenerateStatsLambda-je7MtGbClwBF"
            ],
            "Effect": "Allow"
        }
    ]
```

```
}
```

다음 정책은 로그에 대한 충분한 권한이 있는지 확인합니다. CloudWatch

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

선택적 체크포인트 예(Express 워크플로)

이 샘플 프로젝트는 선택적 체크포인트를 수행하는 모의 전자 상거래 워크플로를 실행하여 표준 및 Express 워크플로를 결합하는 방법을 보여줍니다. 이 샘플 프로젝트를 배포하면 표준 워크플로 상태 시스템, 중첩된 Express 워크플로 상태 시스템, AWS Lambda 함수, Amazon Simple Queue Service(Amazon SQS) 대기열 및 Amazon Simple Notification Service(SNS) 주제가 생성됩니다.

Express 워크플로, 중첩된 워크플로 및 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 표준 워크플로와 Express 워크플로 비교
- 작업 상태에서 워크플로 실행 시작

- 다른 서비스와 AWS Step Functions 함께 사용

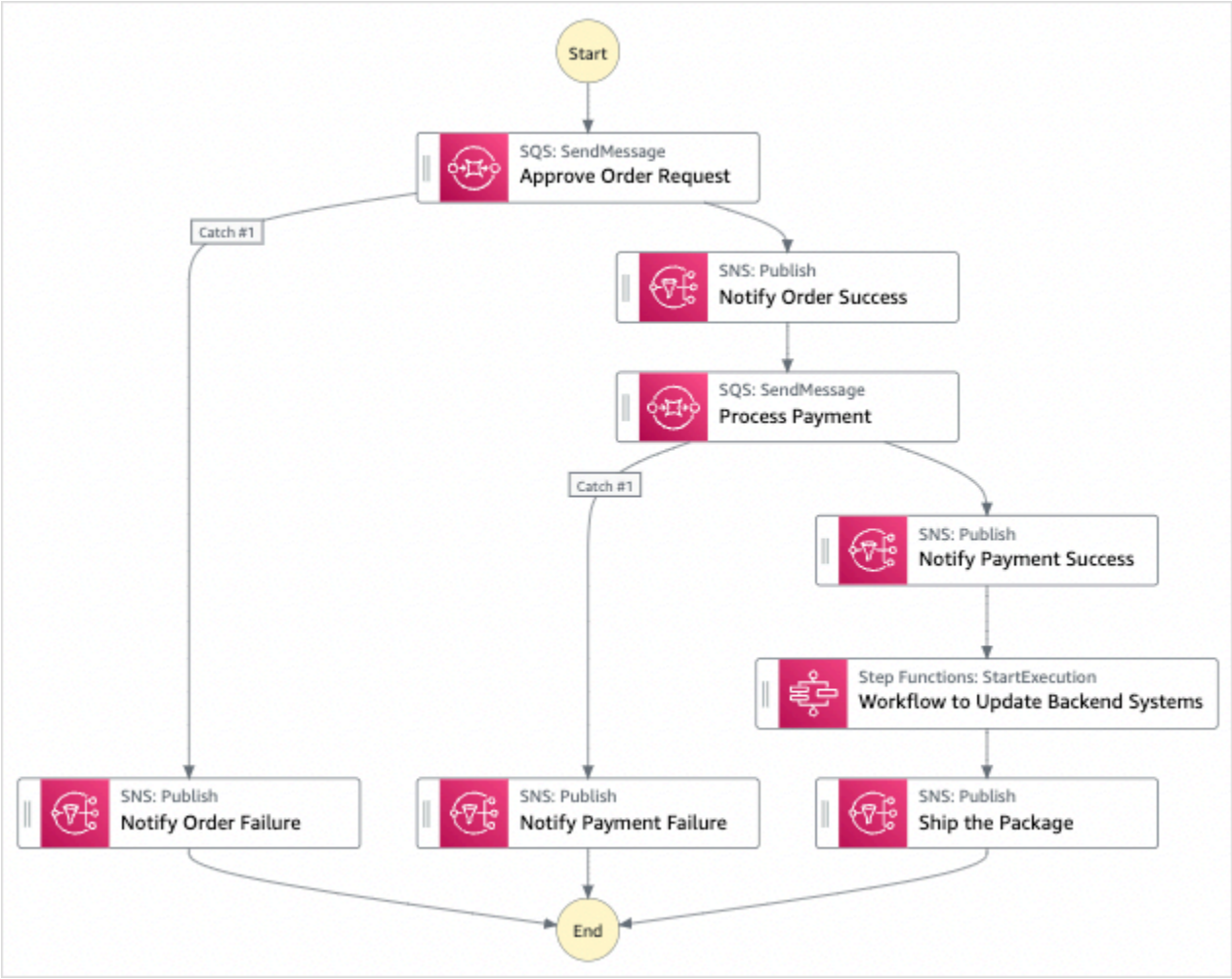
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Selective checkpointing example`**을 입력한 다음 반환된 검색 결과에서 선택적 체크포인트 수행 예를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - AWS Lambda 함수
   - Amazon SQS 대기열
   - Amazon SNS 주제
   - 표준 유형의 AWS Step Functions 스테이트 머신.
   - Express 유형의 Step Functions 상태 시스템
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 선택적 체크포인트 수행 예 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

  > **Tip**
  >
  > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

  준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

  이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

  샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

  > **Important**
  >
  > 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

샘플 프로젝트의 리소스를 배포한 후 다음을 수행합니다.

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

4. CloudWatch 로그 로그 그룹으로 이동하여 로그를 살펴보세요. 로그 그룹 이름은 예시-ExpressLogGroup -WJALRXUTNFEMI와 같이 표시됩니다.

상위 상태 머신 코드 예제(표준 워크플로)

이 샘플 프로젝트의 상태 시스템은 Amazon SQS, Amazon SNS 및 Step Functions Express 워크플로와 통합됩니다.

이 예제 상태 시스템을 살펴보고 Step Functions에서 Amazon SQS 및 Amazon SNS의 입력을 처리한 다음 중첩된 Express 워크플로 상태 시스템을 사용하여 백엔드 시스템을 업데이트하는 방법을 확인합니다.

다른 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오. AWS 다른 서비스와 AWS Step Functions 함께 사용

```
{
    "Comment": "An example of combining standard and express workflows to run a mock e-
commerce workflow that does selective checkpointing.",
    "StartAt": "Approve Order Request",
    "States": {
        "Approve Order Request": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sqs:sendMessage.waitForTaskToken",
            "Parameters": {
                "QueueUrl": "<SQS_QUEUE_URL>",
                "MessageBody": {
                    "MessageTitle": "Order Request received. Pausing workflow to wait
 for manual approval. ",
                    "TaskToken.$": "$$.Task.Token"
                }
            },
            "Next": "Notify Order Success",
            "Catch": [
                {
                    "ErrorEquals": [
                        "States.ALL"
                    ],
                    "Next": "Notify Order Failure"
                }
            ]
        },
        "Notify Order Success": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Order has been approved. Resuming workflow.",
                "TopicArn": "<SNS_ARN>"
            },
            "Next": "Process Payment"
        },
        "Notify Order Failure": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Order not approved. Order failed.",
```

```
                "TopicArn": "<SNS_ARN>"
            },
            "End": true
        },
        "Process Payment": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sqs:sendMessage.waitForTaskToken",
            "Parameters": {
                "QueueUrl": "<SQS_QUEUE_URL>",
                "MessageBody": {
                    "MessageTitle": "Payment sent to third-party for processing.
 Pausing workflow to wait for response.",
                    "TaskToken.$": "$$.Task.Token"
                }
            },
            "Next": "Notify Payment Success",
            "Catch": [
                {
                    "ErrorEquals": [
                        "States.ALL"
                    ],
                    "Next": "Notify Payment Failure"
                }
            ]
        },
        "Notify Payment Success": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Payment processing succeeded. Resuming workflow.",
                "TopicArn": "<SNS_ARN>"
            },
            "Next": "Workflow to Update Backend Systems"
        },
        "Notify Payment Failure": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Payment processing failed.",
                "TopicArn": "<SNS_ARN>"
            },
            "End": true
        },
        "Workflow to Update Backend Systems": {
```

```
            "Comment": "Starting an execution of an Express workflow to handle backend
 updates. Express workflows are fast and cost-effective for steps where checkpointing
 isn't required.",
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::states:startExecution.sync",
            "Parameters": {
                "StateMachineArn": "<UPDATE_DATABASE_EXPRESS_STATE_MACHINE_ARN>",
                "Input": {
                    "AWS_STEP_FUNCTIONS_STARTED_BY_EXECUTION_ID.$": "$$.Execution.Id"
                }
            },
            "Next": "Ship the Package"
        },
        "Ship the Package": {
            "Type": "Task",
            "Resource": "arn:<PARTITION>:states:::sns:publish",
            "Parameters": {
                "Message": "Order and payment received, database is updated and the
 package is ready to ship.",
                "TopicArn": "<SNS_ARN>"
            },
            "End": true
        }
    }
}
```

상위 상태 시스템에 대한 IAM 역할 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

Amazon SNS 정책:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:us-east-1:123456789012:Checkpoint-SNSTopic-
wJalrXUtnFEMI",
```

```
            "Effect": "Allow"
        }
    ]
}
```

Amazon SQS 정책:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:us-east-1:123456789012:Checkpoint-SQSQueue-
je7MtGbClwBF",
            "Effect": "Allow"
        }
    ]
}
```

상태 실행 정책:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "states:StartExecution",
                "states:DescribeExecution",
                "states:StopExecution"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": "arn:aws:events:us-east-1:123456789012:rule/
StepFunctionsGetEventsForStepFunctionsExecutionRule",
```

```
            "Effect": "Allow"
        }
    ]
}
```

중첩 상태 머신의 상태 머신 코드 예제(Express 워크플로)

이 샘플 프로젝트의 상태 머신은 상위 상태 머신에서 호출할 때 백엔드 정보를 업데이트합니다.

이 예제 상태 시스템을 살펴보고 Step Functions에서 모의 전자 상거래 백엔드 시스템의 여러 구성 요소를 업데이트하는 방법을 확인합니다.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용


```
{
  "StartAt": "Update Order History",
  "States": {
    "Update Order History": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "Checkpoint-UpdateDatabaseLambdaFunction-wJalrXUtnFEMI",
```

```
        "Payload": {
          "Message": "Update order history."
        }
      },
      "Next": "Update Data Warehouse"
    },
    "Update Data Warehouse": {
      "Type" : "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "Checkpoint-UpdateDatabaseLambdaFunction-wJalrXUtnFEMI",
        "Payload": {
          "Message": "Update data warehouse."
        }
      },
      "Next": "Update Customer Profile"
    },
    "Update Customer Profile": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "Checkpoint-UpdateDatabaseLambdaFunction-wJalrXUtnFEMI",
        "Payload": {
          "Message": "Update customer profile."
        }
      },
      "Next": "Update Inventory"
    },
    "Update Inventory": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Parameters": {
        "FunctionName": "Checkpoint-UpdateDatabaseLambdaFunction-wJalrXUtnFEMI",
        "Payload": {
          "Message": "Update inventory."
        }
      },
      "End": true
    }
  }
}
```

하위 상태 시스템에 대한 IAM 역할 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:123456789012:function:Example-
UpdateDatabaseLambdaFunction-wJalrXUtnFEMI"
            ],
            "Effect": "Allow"
        }
    ]
}
```

다음 정책은 로그에 대한 CloudWatch 충분한 권한이 있는지 확인합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
```

```
        }
    ]
}
```

다른 AWS 서비스와 함께 Step Functions를 사용할 때 IAM을 구성하는 방법은 통합 서비스용 IAM 정책 섹션을 참조하세요.

AWS CodeBuild 프로젝트 구축 (CodeBuild, 아마존 SNS)

이 샘플 프로젝트는 프로젝트를 빌드하고 테스트를 실행한 AWS CodeBuild 다음 Amazon SNS 알림을 보내는 AWS Step Functions 데 사용하는 방법을 보여줍니다.

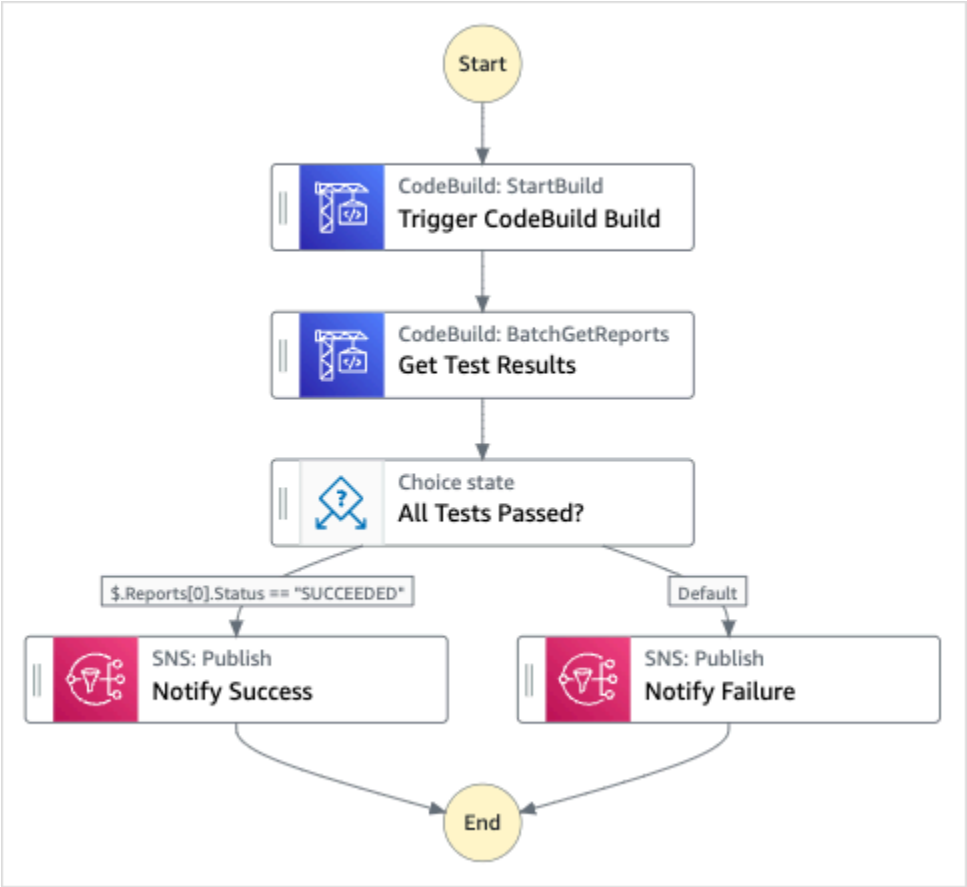
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 입력한 다음 반환된 검색 결과에서 CodeBuild 빌드 시작을 선택합니다. **`Start a CodeBuild build`**
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - AWS CodeBuild 빌드
   - Amazon SNS 주제
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지는 CodeBuild 빌드 시작 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 Amazon CodeBuild SNS와 통합됩니다.

이 예제 상태 머신을 탐색하여 Step Functions가 어떻게 상태 머신을 사용하여 CodeBuild 프로젝트를 빌드하고, 작업의 성공 또는 실패 여부에 대한 메시지와 함께 Amazon SNS 주제를 전송하는 방법을 살펴봅니다.

Step Functions가 다른 AWS 서비스를 제어하는 방법에 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
  "Comment": "An example of using CodeBuild to run tests, get test results and send a
 notification.",
  "StartAt": "Trigger CodeBuild Build",
  "States": {
    "Trigger CodeBuild Build": {
      "Type": "Task",
      "Resource": "arn:aws:states:::codebuild:startBuild.sync",
      "Parameters": {
        "ProjectName": "CodeBuildProject-Dtw1jBhEYGDf"
```

```
      },
      "Next": "Get Test Results"
    },
    "Get Test Results": {
      "Type": "Task",
      "Resource": "arn:aws:states:::codebuild:batchGetReports",
      "Parameters": {
        "ReportArns.$": "$.Build.ReportArns"
      },
      "Next": "All Tests Passed?"
    },
    "All Tests Passed?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.Reports[0].Status",
          "StringEquals": "SUCCEEDED",
          "Next": "Notify Success"
        }
      ],
      "Default": "Notify Failure"
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "CodeBuild build tests succeeded",
        "TopicArn": "arn:aws:sns:sa-east-1:123456789012:StepFunctionsSample-
CodeBuildExecution3da9ead6-bc1f-4441-99ac-591c140019c4-SNSTopic-EVYLVNGW85JP"
      },
      "End": true
    },
    "Notify Failure": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "CodeBuild build tests failed",
        "TopicArn": "arn:aws:sns:sa-east-1:123456789012:StepFunctionsSample-
CodeBuildExecution3da9ead6-bc1f-4441-99ac-591c140019c4-SNSTopic-EVYLVNGW85JP"
      },
      "End": true
    }
  }
```

```
}
```

다른 AWS 서비스와 함께 Step Functions를 사용할 때 IAM을 구성하는 방법은 통합 서비스용 IAM 정책 섹션을 참조하세요.

데이터 전처리 및 기계 학습 모델 학습

이 샘플 프로젝트는 데이터를 사용하고 데이터를 SageMaker 전처리하며 AWS Step Functions 기계 학습 모델을 훈련시키는 방법을 보여줍니다.

이 프로젝트에서 Step Functions는 Lambda 함수를 사용하여 테스트 데이터 세트와 데이터 처리용 Python 스크립트로 Amazon S3 버킷을 시딩합니다. 그런 다음 SageMaker 서비스 통합을 사용하여 기계 학습 모델을 학습시키고 일괄 변환을 수행합니다.

Step Functions 서비스 통합에 대한 SageMaker 자세한 내용은 다음을 참조하십시오.

- 다른 서비스와 AWS Step Functions 함께 사용
- Step SageMaker Functions를 사용한 관리

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 SageMaker 요금을 참조하십시오.

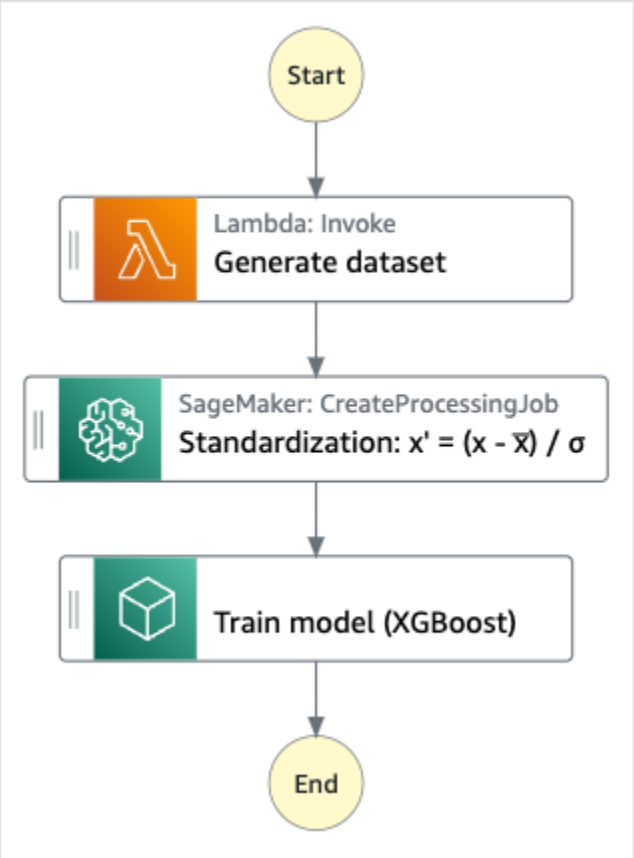
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Preprocess data and train a machine learning model`**을 입력한 다음 반환된 검색 결과에서 데이터 전처리 및 기계 학습 모델 학습을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

이 샘플 프로젝트는 다음 리소스를 배포합니다.

- AWS Lambda 함수
- Amazon S3 버킷
- AWS Step Functions 스테이트 머신
- 관련 AWS Identity and Access Management (IAM) 역할

다음 이미지에서는 데이터 전처리 및 기계 학습 모델 학습 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하

여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

   > **Note**
   >
   > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 해당 리소스에 파라미터를 직접 AWS Lambda 전달하여 통합하고, Amazon S3 버킷을 교육 데이터 소스 및 출력으로 SageMaker 사용합니다.

이 예제 상태 머신을 살펴보고 Step Functions가 Lambda SageMaker 및 를 제어하는 방법을 살펴보세요.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "StartAt": "Generate dataset",
  "States": {
    "Generate dataset": {
      "Resource": "arn:aws:lambda:sa-east-1:1234567890:function:FeatureTransform-
LambaForDataGeneration-17M8LX7IO9LUW",
      "Type": "Task",
      "Next": "Standardization: x' = (x - x̄) / σ"
    },
    "Standardization: x' = (x - x̄) / σ": {
      "Resource": "arn:aws:states:::sagemaker:createProcessingJob.sync",
      "Parameters": {
        "ProcessingResources": {
          "ClusterConfig": {
            "InstanceCount": 1,
            "InstanceType": "ml.m5.xlarge",
            "VolumeSizeInGB": 10
          }
        },
        "ProcessingInputs": [
            {
                "InputName": "input-1",
                "S3Input": {
                    "S3Uri": "s3://featuretransform-bucketforcodeanddata-1jn1le6gadwfz/
input/raw.csv",
                    "LocalPath": "/opt/ml/processing/input",
                    "S3DataType": "S3Prefix",
                    "S3InputMode": "File",
                    "S3DataDistributionType": "FullyReplicated",
                    "S3CompressionType": "None"
                }
            },
            {
                "InputName": "code",
                "S3Input": {
                    "S3Uri": "s3://featuretransform-bucketforcodeanddata-1jn1le6gadwfz/
code/transform.py",
                    "LocalPath": "/opt/ml/processing/input/code",
                    "S3DataType": "S3Prefix",
                    "S3InputMode": "File",
```

```
                    "S3DataDistributionType": "FullyReplicated",
                    "S3CompressionType": "None"
                }
            }
        ],
        "ProcessingOutputConfig": {
            "Outputs": [
                {
                    "OutputName": "train_data",
                    "S3Output": {
                        "S3Uri": "s3://featuretransform-
bucketforcodeanddata-1jn1le6gadwfz/train",
                        "LocalPath": "/opt/ml/processing/output/train",
                        "S3UploadMode": "EndOfJob"
                    }
                }
            ]
        },
        "AppSpecification": {
            "ImageUri": "737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-scikit-
learn:0.20.0-cpu-py3",
            "ContainerEntrypoint": [
                "python3",
                "/opt/ml/processing/input/code/transform.py"
            ]
        },
        "StoppingCondition": {
            "MaxRuntimeInSeconds": 300
        },
        "RoleArn": "arn:aws:iam::1234567890:role/SageMakerAPIExecutionRole-
AIDACKCEVSQ6C2EXAMPLE",
        "ProcessingJobName.$": "$$.Execution.Name"
      },
      "Type": "Task",
      "Next": "Train model (XGBoost)"
    },
    "Train model (XGBoost)": {
      "Resource": "arn:aws:states:::sagemaker:createTrainingJob.sync",
      "Parameters": {
        "AlgorithmSpecification": {
          "TrainingImage": "855470959533.dkr.ecr.sa-east-1.amazonaws.com/
xgboost:latest",
          "TrainingInputMode": "File"
        },
```

```
        "OutputDataConfig": {
          "S3OutputPath": "s3://featuretransform-bucketforcodeanddata-1jn1le6gadwfz/
models"
        },
        "StoppingCondition": {
          "MaxRuntimeInSeconds": 86400
        },
        "ResourceConfig": {
          "InstanceCount": 1,
          "InstanceType": "ml.m5.xlarge",
          "VolumeSizeInGB": 30
        },
        "RoleArn": "arn:aws:iam::1234567890:role/SageMakerAPIExecutionRole-
AIDACKCEVSQ6C2EXAMPLE",
        "InputDataConfig": [
          {
            "DataSource": {
              "S3DataSource": {
                "S3DataDistributionType": "ShardedByS3Key",
                "S3DataType": "S3Prefix",
                "S3Uri": "s3://featuretransform-bucketforcodeanddata-1jn1le6gadwfz"
              }
            },
            "ChannelName": "train",
            "ContentType": "text/csv"
          }
        ],
        "HyperParameters": {
          "objective": "reg:logistic",
          "eval_metric": "rmse",
          "num_round": "5"
        },
        "TrainingJobName.$": "$$.Execution.Name"
      },
      "Type": "Task",
      "End": true
    }
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

다음 정책을 통해 Lambda 함수에서 샘플 데이터를 사용하여 Amazon S3 버킷을 시딩할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::featuretransform-
bucketforcodeanddata-1jn1le6gadwfz/*",
            "Effect": "Allow"
```

```
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Lambda 오케스트레이션 예제

이 샘플 프로젝트는 Step Functions 상태 머신에서 AWS Lambda 함수를 통합하는 방법을 보여줍니다.

이 프로젝트에서 Step Functions는 Lambda 함수를 사용하여 주가를 확인하고 추천 매수 또는 매도 거래를 결정합니다. 그런 다음 사용자에게 이 추천이 제공되고 사용자는 주식을 매수 또는 매도할지 선택할 수 있습니다. 거래 결과는 SNS 주제를 통해 반환됩니다.

Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 다른 서비스와 AWS Step Functions 함께 사용
- 다음에 대한 IAM 정책:
  - 다음에 대한 IAM 정책 AWS Lambda
  - 아마존 SQS에 대한 IAM 정책
  - 아마존 SNS를 위한 IAM 정책

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 요금을 참조하십시오.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

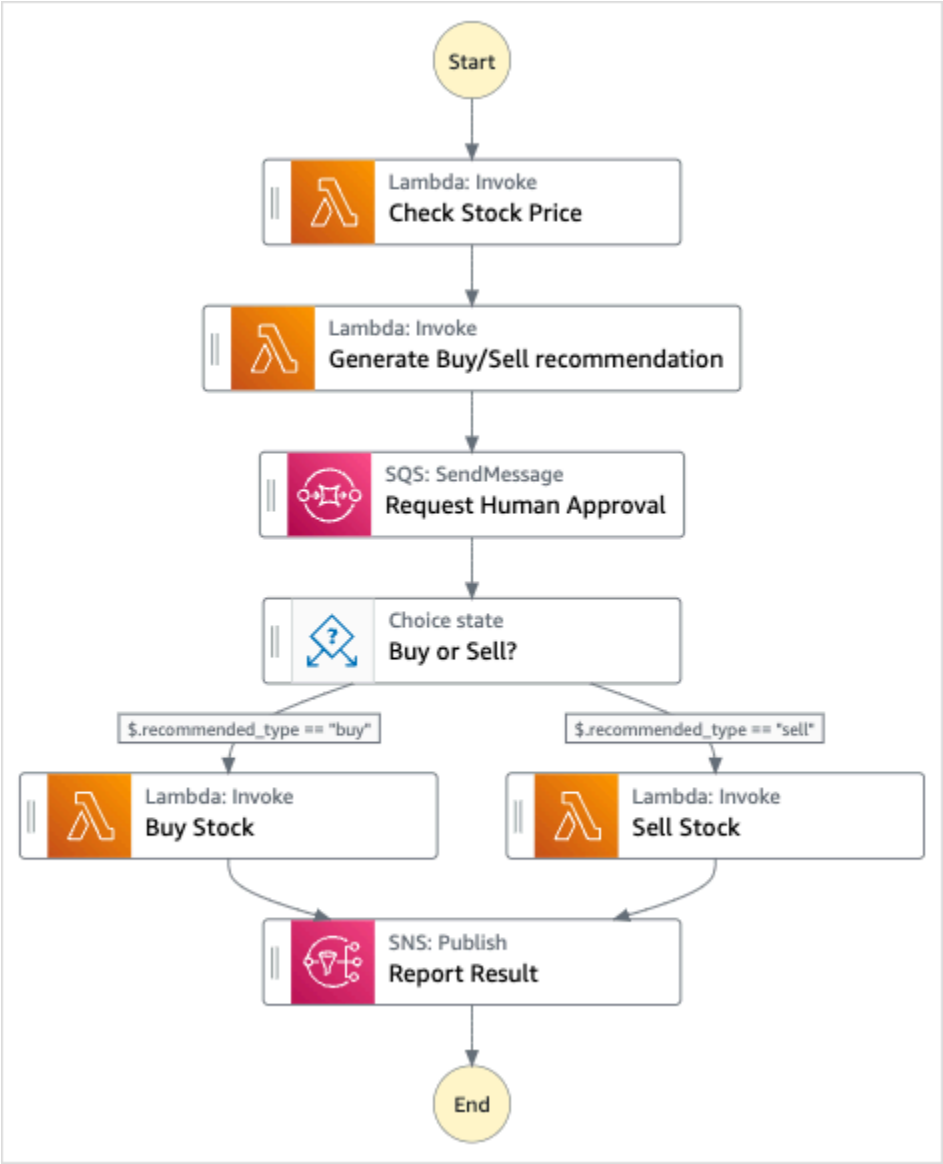
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Orchestrate Lambda functions`**를 입력한 다음 반환된 검색 결과에서 Lambda 함수 오케스트레이션을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.

4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Lambda 함수 5개
   - Amazon Simple Queue Service 대기열
   - Amazon Simple Notification Service 주제
   - AWS Step Functions 상태 머신
   - 관련된 AWS Identity and Access Management(IAM) 역할

   다음 이미지에서는 Lambda 함수 오케스트레이션 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

모든 리소스가 프로비저닝되고 배포되면 실행 시작 대화 상자가 표시됩니다.

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.
    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

상태 시스템 및 실행 정보

이 샘플 프로젝트의 상태 머신은 파라미터를 해당 리소스에 직접 AWS Lambda 전달하여 와 통합하고, Amazon SQS 대기열을 사용하여 사람의 승인 요청을 관리하고, Amazon SNS 주제를 사용하여 쿼리 결과를 반환합니다.

Step Functions 실행에서 JSON 텍스트를 입력으로 수신하고 해당 입력을 워크플로의 첫 번째 상태에 전달합니다. 개별 상태는 JSON 데이터를 입력으로 수신하고 일반적으로 JSON 데이터를 출력으로 다음 상태로 전달합니다. 이 샘플 프로젝트에서는 각 단계의 출력이 입력으로 워크플로의 다음 단계에 전달됩니다. 예를 들어 구매/판매 추천 생성 단계에서는 주가 확인 단계 출력을 입력으로 수신합니다.

또한 구매/판매 추천 생성 단계 출력은 입력으로 사람의 승인 단계를 모방한 다음 단계인 사람 승인 요청으로 전달됩니다.

> **Note**
>
> 단계에서 반환한 출력과 단계에 전달된 입력을 보려면 워크플로 실행에 대한 실행 세부 정보 페이지를 여세요. 단계 세부 정보 섹션에는 보기 모드에서 선택한 각 단계의 입력 및 출력이 표시됩니다.

사람 승인 단계를 구현하려면 일반적으로 작업 토큰이 반환될 때까지 워크플로 실행을 일시 중지합니다. 이 샘플 프로젝트에서는 메시지가 Amazon SQS 대기열로 전달되며 이 대기열은 콜백 기능을 처리하도록 정의된 Lambda 함수의 트리거로 사용됩니다. 메시지에는 작업 토큰과 이전 단계에서 반환한 출력이 포함됩니다. Lambda 함수는 메시지의 페이로드와 함께 간접적으로 호출됩니다. `SendTaskSuccess` API 직접 호출을 통해 작업 토큰을 다시 수신할 때까지 워크플로 실행이 일시 중지됩니다. 작업 토큰에 대한 자세한 내용은 작업 토큰을 사용하여 콜백 대기 섹션을 참조하세요.

다음 `StepFunctionsSample-HelloLambda-ApproveSqsLambda` 함수 코드에서는 Step Functions 상태 시스템을 통해 Amazon SQS 대기열에서 제출한 모든 작업을 자동으로 승인하도록 정의되는 방식을 보여줍니다.

콜백 기능을 처리하고 작업 토큰을 반환하는 샘플 Lambda 함수 코드

```
exports.lambdaHandler = (event, context, callback) => {
  const stepfunctions = new aws.StepFunctions();

  // For every record in sqs queue
  for (const record of event.Records) {
      const messageBody = JSON.parse(record.body);
      const taskToken = messageBody.TaskToken;

      const params = {
          output: "\"approved\"",
          taskToken: taskToken
      };

      console.log(`Calling Step Functions to complete callback task with params
 ${JSON.stringify(params)}`);

      // Approve
      stepfunctions.sendTaskSuccess(params, (err, data) => {
```

```
          if (err) {
              console.error(err.message);
              callback(err.message);
              return;
          }
          console.log(data);
          callback(null);
      });
  }
};
```

이 예제 상태 시스템을 살펴보고 Step Functions에서 Lambda 및 Amazon SQS를 제어하는 방법을 확인합니다.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
    "StartAt": "Check Stock Price",
    "States": {
        "Check Stock Price": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLam-
CheckStockPriceLambda-444455556666",
            "Next": "Generate Buy/Sell recommendation"
        },
        "Generate Buy/Sell recommendation": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-Hello-
GenerateBuySellRecommend-123456789012",
            "ResultPath": "$.recommended_type",
            "Next": "Request Human Approval"
        },
        "Request Human Approval": {
            "Type": "Task",
            "Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken",
            "Parameters": {
                "QueueUrl": "https://sqs.us-west-1.amazonaws.com/111122223333/
StepFunctionsSample-HelloLambda4444-5555-6666-RequestHumanApprovalSqs-777788889999",
                "MessageBody": {
                    "Input.$": "$",
```

```
                    "TaskToken.$": "$$.Task.Token"
                }
            },
            "ResultPath": null,
            "Next": "Buy or Sell?"
        },
        "Buy or Sell?": {
            "Type": "Choice",
            "Choices": [
                {
                    "Variable": "$.recommended_type",
                    "StringEquals": "buy",
                    "Next": "Buy Stock"
                },
                {
                    "Variable": "$.recommended_type",
                    "StringEquals": "sell",
                    "Next": "Sell Stock"
                }
            ]
        },
        "Buy Stock": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLambda-
BuyStockLambda-000000000000",
            "Next": "Report Result"
        },
        "Sell Stock": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLambda-
SellStockLambda-111111111111",
            "Next": "Report Result"
        },
        "Report Result": {
            "Type": "Task",
            "Resource": "arn:aws:states:::sns:publish",
            "Parameters": {
                "TopicArn": "arn:aws:sns:us-west-1:111122223333:StepFunctionsSample-
HelloLambda1111-2222-3333-ReportResultSnsTopic-222222222222",
                "Message": {
                    "Input.$": "$"
                }
```

```
            },
            "End": true
        }
    }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLam-
CheckStockPriceLambda-444455556666",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-Hello-
GenerateBuySellRecommend-123456789012",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLambda-
BuyStockLambda-777788889999",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-1:111122223333:function:StepFunctionsSample-HelloLambda-
SellStockLambda-000000000000",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "sqs:SendMessage*"
            ],
            "Resource": "arn:aws:sqs:us-west-1:111122223333:StepFunctionsSample-
HelloLambda4444-5555-6666-RequestHumanApprovalSqs-111111111111",
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:us-west-1:111122223333:StepFunctionsSample-
HelloLambda1111-2222-3333-ReportResultSnsTopic-222222222222",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Athena 쿼리 시작

표준 워크플로를 기반으로 하는 이 샘플 프로젝트에서는 Step Functions와 Amazon Athena를 사용하여 Athena 쿼리를 시작하고 쿼리 결과가 포함된 알림을 보내는 방법을 보여줍니다.

이 프로젝트에서 Step Functions는 Lambda 함수와 크롤러를 사용하여 AWS Glue 예제 데이터 세트를 생성합니다. 그런 다음 Athena 서비스 통합을 사용하여 쿼리를 수행하고 SNS 주제를 사용하여 결과를 반환합니다.

Athena 및 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 다른 서비스와 AWS Step Functions 함께 사용
- Step Functions를 사용하여 Athena 직접 호출

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 Athena 요금을 참조하십시오.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Start an Athena query`**를 입력한 다음 반환된 검색 결과에서 Athena 쿼리 시작을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon Athena 쿼리
   - AWS Glue 크롤러
   - Amazon SNS 주제
   - AWS Step Functions 상태 머신
   - 관련된 AWS Identity and Access Management(IAM) 역할

   다음 이미지에서는 Athena 쿼리 시작 샘플 프로젝트의 워크플로 그래프를 보여줍니다.

5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 AWS Lambda Athena와 통합되어 해당 리소스에 매개변수를 직접 전달하고 SNS 주제를 사용하여 쿼리 결과를 반환합니다.

이 예제 상태 시스템을 살펴보고 Step Functions에서 Lambda 및 Athena를 제어하는 방법을 확인합니다.

다른 AWS 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
```

```
  "StartAt": "Generate example log",
  "States": {
    "Generate example log": {
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:StepFunctionsSample-
Athena-LambdaForDataGeneration-AKIAIOSFODNN7EXAMPLE",
      "Type": "Task",
      "Next": "Run Glue crawler"
    },
    "Run Glue crawler": {
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:StepFunctionsSample-
Athen-LambdaForInvokingCrawler-AKIAI44QH8DHBEXAMPLE",
      "Type": "Task",
      "Next": "Start an Athena query"
    },
    "Start an Athena query": {
      "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
      "Parameters": {
        "QueryString": "SELECT * FROM \"athena-sample-project-db-wJalrXUtnFEMI\".\"log
\" limit 1",
        "WorkGroup": "stepfunctions-athena-sample-project-workgroup-wJalrXUtnFEMI"
      },
      "Type": "Task",
      "Next": "Get query results"
    },
    "Get query results": {
      "Resource": "arn:aws:states:::athena:getQueryResults",
      "Parameters": {
        "QueryExecutionId.$": "$.QueryExecution.QueryExecutionId"
      },
      "Type": "Task",
      "Next": "Send query results"
    },
    "Send query results": {
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "TopicArn": "arn:aws:sns:us-east-1:111122223333:StepFunctionsSample-
AthenaDataQueryd1111-2222-3333-777788889999-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE",
        "Message": {
          "Input.$": "$.ResultSet.Rows"
        }
      },
      "Type": "Task",
      "End": true
    }
```

```
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:111122223333:function:StepFunctionsSample-
Athena-LambdaForDataGeneration-AKIAIOSFODNN7EXAMPLE",
                "arn:aws:lambda:us-east-1:111122223333:function:StepFunctionsSample-
Athen-LambdaForInvokingCrawler-AKIAI44QH8DHBEXAMPLE"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-1:111122223333:StepFunctionsSample-
AthenaDataQueryd1111-2222-3333-777788889999-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "athena:getQueryResults",
                "athena:startQueryExecution",
                "athena:stopQueryExecution",
                "athena:getQueryExecution",
```

```
                "athena:getDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:111122223333:workgroup/stepfunctions-athena-
sample-project-workgroup-wJalrXUtnFEMI",
                "arn:aws:athena:us-east-1:111122223333:datacatalog/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:DeleteDatabase",
                "glue:CreateTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
```

```
            ],
            "Resource": [
                "arn:aws:glue:us-east-1:111122223333:database/*",
                "arn:aws:glue:us-east-1:111122223333:table/*",
                "arn:aws:glue:us-east-1:111122223333:catalog"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

여러 쿼리 실행(Amazon Athena, Amazon SNS)

이 샘플 프로젝트에서는 Athena 쿼리를 연속해서 실행한 다음 오류를 병렬로 처리한 다음 쿼리의 성공 또는 실패 여부에 따라 Amazon SNS 알림을 보내는 방법을 보여줍니다.

이 프로젝트에서 Step Functions는 상태 시스템을 사용하여 Athena 쿼리를 동기적으로 실행합니다. 쿼리 결과가 반환되면 Athena 쿼리 2개가 동시에 실행되는 병렬 상태로 전환됩니다. 그런 다음 작업이 성공 또는 실패하기까지 대기한 다음 작업의 성공 또는 실패 여부에 대한 메시지와 함께 Amazon SNS 주제를 전송합니다.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

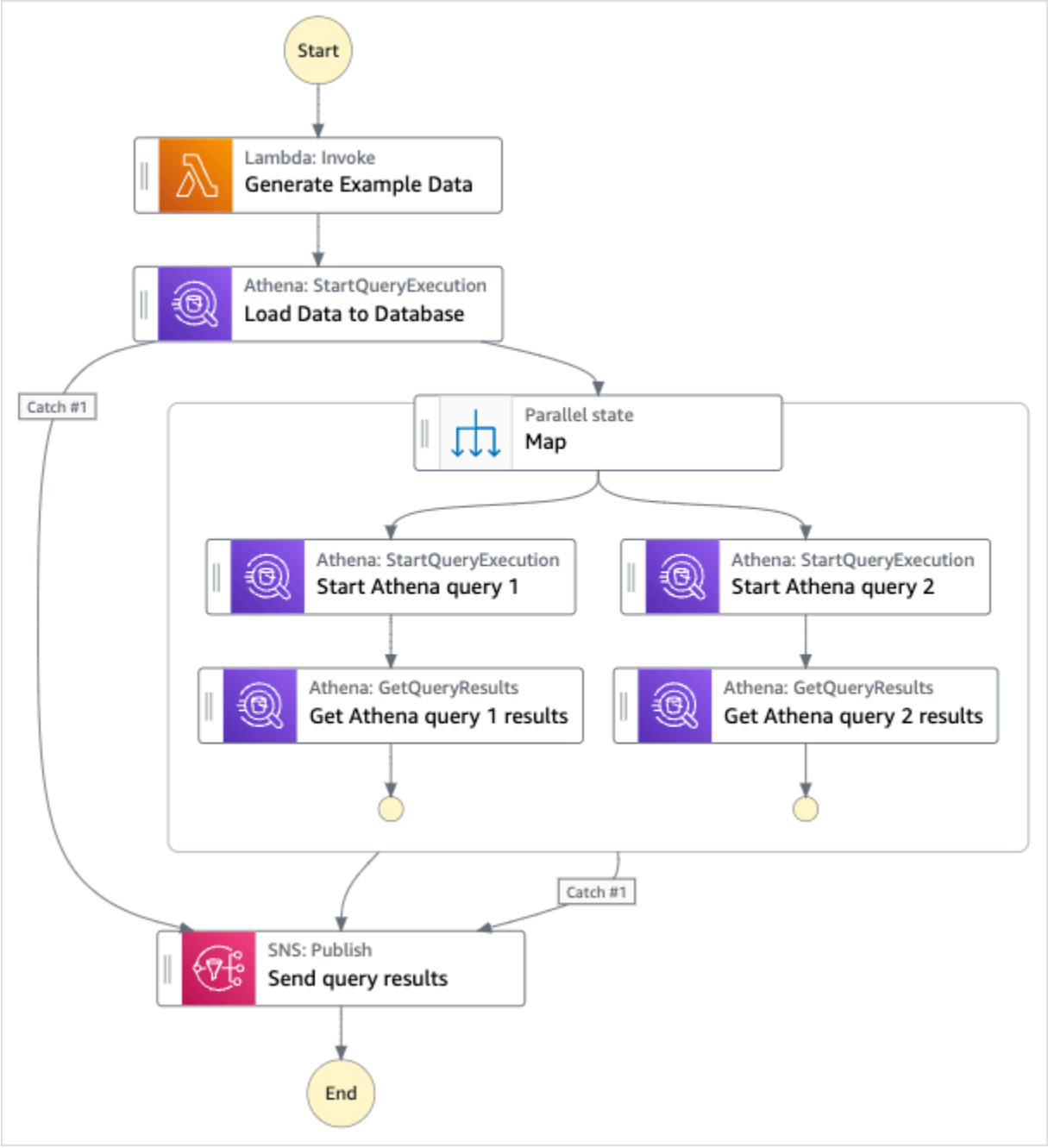
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Execute multiple queries`**를 입력한 다음 반환된 검색 결과에서 여러 쿼리 실행을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon Athena 쿼리

- Amazon SNS 주제
- AWS Step Functions 상태 머신
- 관련된 AWS Identity and Access Management(IAM) 역할

다음 이미지에서는 여러 쿼리 실행 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

- 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

  Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

  > **Important**
  >
  > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

  > **Tip**
  >
  > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

  준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

  이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

  샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

  > **Important**
  >
  > 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

        > **Note**
        >
        > Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

    2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

        데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

    3. 실행 시작을 선택합니다.
    4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

        실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 시스템은 파라미터를 리소스에 직접 전달하여 Amazon Athena 및 Amazon SNS와 통합됩니다.

이 예제 상태 시스템을 살펴보고 `Resource` 필드의 Amazon 리소스 이름(ARN)과 연결하고 `Parameters`를 서비스 API에 전달하여 Step Functions에서 Amazon Athena 및 Amazon SNS를 제어하는 방법을 확인합니다.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
    "Comment": "An example of using Athena to execute queries in sequence and parallel,
 with error handling and notifications.",
    "StartAt": "Generate Example Data",
    "States": {
      "Generate Example Data": {
        "Type": "Task",
        "Resource": "arn:aws:states:::lambda:invoke",
        "OutputPath": "$.Payload",
        "Parameters": {
          "FunctionName": "<ATHENA_FUNCTION_NAME>"
        },
        "Next": "Load Data to Database"
      },
      "Load Data to Database": {
        "Type": "Task",
        "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
        "Parameters": {
          "QueryString": "<ATHENA_QUERYSTRING>",
          "WorkGroup": "<ATHENA_WORKGROUP>"
        },
        "Catch": [
          {
            "ErrorEquals": [
              "States.ALL"
            ],
            "Next": "Send query results"
          }
        ],
        "Next": "Map"
      },
      "Map": {
        "Type": "Parallel",
        "ResultSelector": {
          "Query1Result.$": "$[0].ResultSet.Rows",
          "Query2Result.$": "$[1].ResultSet.Rows"
        },
        "Catch": [
          {
            "ErrorEquals": [
```

```
              "States.ALL"
            ],
            "Next": "Send query results"
          }
        ],
        "Branches": [
          {
            "StartAt": "Start Athena query 1",
            "States": {
              "Start Athena query 1": {
                "Type": "Task",
                "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
                "Parameters": {
                  "QueryString": "<ATHENA_QUERYSTRING>",
                  "WorkGroup": "<ATHENA_WORKGROUP>"
                },
                "Next": "Get Athena query 1 results"
              },
              "Get Athena query 1 results": {
                "Type": "Task",
                "Resource": "arn:aws:states:::athena:getQueryResults",
                "Parameters": {
                  "QueryExecutionId.$": "$.QueryExecution.QueryExecutionId"
                },
                "End": true
              }
            }
          },
          {
            "StartAt": "Start Athena query 2",
            "States": {
              "Start Athena query 2": {
                "Type": "Task",
                "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
                "Parameters": {
                    "QueryString": "<ATHENA_QUERYSTRING>",
                    "WorkGroup": "<ATHENA_WORKGROUP>"
                },
                "Next": "Get Athena query 2 results"
              },
              "Get Athena query 2 results": {
                "Type": "Task",
                "Resource": "arn:aws:states:::athena:getQueryResults",
                "Parameters": {
```

```
                  "QueryExecutionId.$": "$.QueryExecution.QueryExecutionId"
                },
                "End": true
              }
            }
          }
        ],
        "Next": "Send query results"
      },
      "Send query results": {
        "Type": "Task",
        "Resource": "arn:aws:states:::sns:publish",
        "Parameters": {
          "Message.$": "$",
          "TopicArn": "<SNS_TOPIC_ARN>"
        },
        "End": true
      }
    }
  }
```

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

AthenaStartQueryExecution

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:startQueryExecution",
                "athena:stopQueryExecution",
                "athena:getQueryExecution",
                "athena:getDataCatalog"
            ],
            "Resource": [
```

```
                "arn:aws:athena:us-east-2:123456789012:workgroup/stepfunctions-athena-
sample-project-workgroup-ztuvu9yuix",
                "arn:aws:athena:us-east-2:123456789012:datacatalog/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:DeleteDatabase",
                "glue:CreateTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
```

```
            "Resource": [
                "arn:aws:glue:us-east-2:123456789012:catalog",
                "arn:aws:glue:us-east-2:123456789012:database/*",
                "arn:aws:glue:us-east-2:123456789012:table/*",
                "arn:aws:glue:us-east-2:123456789012:userDefinedFunction/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AthenaGetQueryResults

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:getQueryResults"
            ],
            "Resource": [
                "arn:aws:us-east-2:123456789012:workgroup/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        }
```

```
    ]
}
```

SNSPublish

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-2:123456789012:StepFunctionsSample-
AthenaMultipleQueriese1ec229b-5cbe-4754-a8a8-078474bac878-SNSTopic-9AID0HEJT7TH"
            ]
        }
    ]
}
```

LambdaInvokeFunction

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-2:123456789012:function:StepFunctionsSample-
Athen-LambdaForStringGeneratio-GQFQjN7mE9gl:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
```

```
                "arn:aws:lambda:us-east-2:123456789012:function:StepFunctionsSample-
Athen-LambdaForStringGeneratio-GQFQjN7mE9gl"
            ]
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

대규모 데이터세트 쿼리 (아마존 아테나, 아마존 S3 AWS Glue, 아마존 SNS)

이 샘플 프로젝트는 Amazon S3에서 대규모 데이터 세트를 수집하여 AWS Glue Crawlers를 통해 파티셔닝한 다음 해당 파티션에 대해 Amazon Athena 쿼리를 실행하는 방법을 보여줍니다.

이 프로젝트에서 Step Functions 상태 머신은 Amazon S3의 대규모 데이터 세트를 분할하는 AWS Glue 크롤러를 호출합니다. AWS Glue 크롤러가 성공 메시지를 반환하면 워크플로는 해당 파티션에 대해 Athena 쿼리를 실행합니다. 쿼리 실행이 성공적으로 완료되면 Amazon SNS 알림이 Amazon SNS 주제로 전송됩니다.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

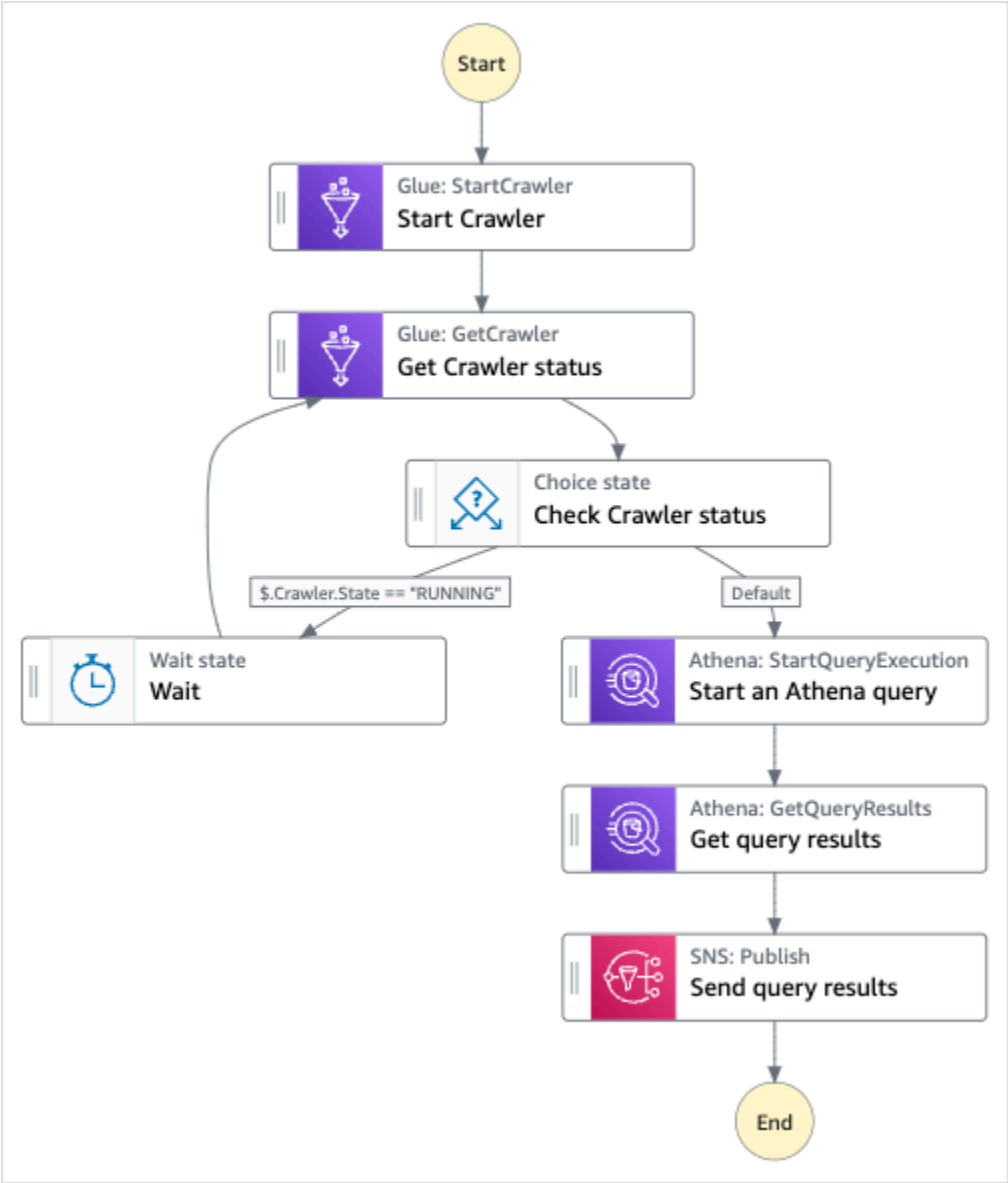
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Query large datasets`**를 입력한 다음 반환된 검색 결과에서 대량의 데이터 세트 쿼리를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon S3 버킷
   - AWS Glue 크롤러
   - Amazon SNS 주제
   - AWS Step Functions 상태 머신

- 관련된 AWS Identity and Access Management(IAM) 역할

다음 이미지에서는 대량의 데이터 세트 쿼리 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

- 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

  Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하

여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

   > **Note**
   >
   > Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

3. 실행 시작을 선택합니다.

4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 해당 리소스에 파라미터를 직접 전달하여 Amazon S3 AWS Glue, Amazon Athena 및 Amazon SNS와 통합됩니다.

이 예제 상태 머신을 살펴보고 Step Functions가 `Resource` 필드의 Amazon 리소스 이름 (ARN) 에 연결하고 서비스 API에 `Parameters` 전달하여 Amazon S3, Amazon Athena 및 Amazon SNS를 제어하는 방법을 살펴봅니다. AWS Glue

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
```

```
   "Comment": "An example demonstrates how to ingest a large data set in Amazon S3 and
partition it through aws Glue Crawlers, then execute Amazon Athena queries against
that partition.",
   "StartAt": "Start Crawler",
   "States": {
     "Start Crawler": {
       "Type": "Task",
       "Next": "Get Crawler status",
       "Parameters": {
         "Name": "<GLUE_CRAWLER_NAME>"
       },
       "Resource": "arn:aws:states:::aws-sdk:glue:startCrawler"
     },
     "Get Crawler status": {
       "Type": "Task",
       "Parameters": {
         "Name": "<GLUE_CRAWLER_NAME>"
       },
       "Resource": "arn:aws:arn:aws:states:::aws-sdk:glue:getCrawler",
       "Next": "Check Crawler status"
     },
     "Check Crawler status": {
       "Type": "Choice",
       "Choices": [
         {
           "Variable": "$.Crawler.State",
           "StringEquals": "RUNNING",
           "Next": "Wait"
         }
       ],
       "Default": "Start an Athena query"
     },
     "Wait": {
       "Type": "Wait",
       "Seconds": 30,
       "Next": "Get Crawler status"
     },
     "Start an Athena query": {
       "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
       "Parameters": {
           "QueryString": "<ATHENA_QUERYSTRING>",
           "WorkGroup": "<ATHENA_WORKGROUP>"
       },
       "Type": "Task",
```

```
        "Next": "Get query results"
      },
      "Get query results": {
        "Resource": "arn:aws:states:::athena:getQueryResults",
        "Parameters": {
          "QueryExecutionId.$": "$.QueryExecution.QueryExecutionId"
        },
        "Type": "Task",
        "Next": "Send query results"
      },
      "Send query results": {
        "Resource": "arn:aws:states:::sns:publish",
        "Parameters": {
          "TopicArn": "<SNS_TOPIC_ARN>",
          "Message": {
            "Input.$": "$.ResultSet.Rows"
          }
        },
        "Type": "Task",
        "End": true
      }
    }
  }
```

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

AthenaGetQueryResults

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:getQueryResults"
            ],
            "Resource": [
                "arn:aws:athena:us-east-2:123456789012:workgroup/*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        }
    ]
}
```

AthenaStartQueryExecution

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:startQueryExecution",
                "athena:stopQueryExecution",
                "athena:getQueryExecution",
                "athena:getDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-2:123456789012:workgroup/stepfunctions-athena-
sample-project-workgroup-8v7bshiv70",
                "arn:aws:athena:us-east-2:123456789012:datacatalog/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
```

```
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:DeleteDatabase",
                "glue:CreateTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-east-2:123456789012:catalog",
                "arn:aws:glue:us-east-2:123456789012:database/*",
                "arn:aws:glue:us-east-2:123456789012:table/*",
                "arn:aws:glue:us-east-2:123456789012:userDefinedFunction/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
```

```
        }
    ]
}
```

SNSPublish

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-2:123456789012:StepFunctionsSample-
AthenaIngestLargeDataset92bc4949-abf8-4a1e-9236-5b7c81b3efa3-SNSTopic-8Y5ZLI5AASXV"
            ]
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

데이터를 최신으로 유지 (아마존 아테나, 아마존 S3,) AWS Glue

이 샘플 프로젝트는 AWS Glue Catalog로 대상 테이블을 쿼리하여 현재 데이터를 가져온 다음 Amazon Athena를 사용하여 다른 소스의 새 데이터로 업데이트하는 방법을 보여줍니다.

이 프로젝트에서는 Step Functions 상태 머신이 AWS Glue Catalog를 호출하여 Amazon S3 버킷에 대상 테이블이 있는지 확인합니다. 테이블이 없으면 새 테이블이 생성됩니다. 그런 다음 Step Functions에서 Athena 쿼리를 실행하여 다른 데이터 소스의 대상 테이블에 행을 추가합니다. 먼저 대상 테이블을 쿼리하여 최신 날짜를 가져온 다음 최신 데이터의 소스 테이블을 쿼리하여 대상 테이블에 삽입합니다.

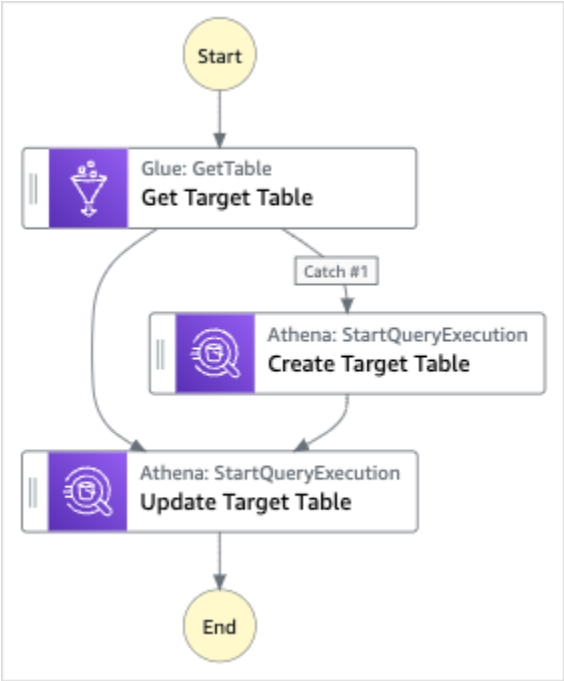
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **Keep data up to date**를 입력한 다음 반환된 검색 결과에서 데이터를 최신 상태로 유지를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon S3 버킷
   - Amazon Athena 쿼리
   - AWS Glue Data Catalog 직접 호출
   - AWS Step Functions 상태 머신
   - 관련된 AWS Identity and Access Management(IAM) 역할

   다음 이미지에서는 데이터를 최신 상태로 유지 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.

6. 다음 중 하나를 수행하십시오.

- 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

  Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

  > **Important**
  >
  > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

  > **Tip**
  >
  > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

  준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

  이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

  샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

  > **Important**
  >
  > 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

        > **Note**
        >
        > Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

    2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

        데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

    3. 실행 시작을 선택합니다.
    4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

        실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 해당 리소스에 파라미터를 직접 전달하여 Amazon S3 및 Amazon Athena와 통합됩니다. AWS Glue

이 예제 상태 머신을 살펴보면서 Step Functions가 `Resource` 필드의 Amazon 리소스 이름 (ARN) 에 연결하고 서비스 API에 `Parameters` 전달하여 Amazon S3와 Amazon Athena를 제어하는 방법을 살펴봅니다. AWS Glue

다른 AWS 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
    "Comment": "An example demonstrates how to use Athena to query a target table to
 get current data, then update it with new data from other sources.",
    "StartAt": "Get Target Table",
    "States": {
      "Get Target Table": {
        "Type": "Task",
        "Parameters": {
          "DatabaseName": "<GLUE_DATABASE_NAME>",
          "Name": "target"
        },
        "Catch": [
          {
            "ErrorEquals": [
              "Glue.EntityNotFoundException"
            ],
            "Next": "Create Target Table"
          }
        ],
        "Resource": "arn:aws:states:::aws-sdk:glue:getTable",
        "Next": "Update Target Table"
      },
      "Create Target Table": {
        "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
        "Parameters": {
            "QueryString": "<ATHENA_QUERYSTRING>",
            "WorkGroup": "<ATHENA_WORKGROUP>"
        },
        "Type": "Task",
        "Next": "Update Target Table"
      },
      "Update Target Table": {
        "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
        "Parameters": {
            "QueryString": "<ATHENA_QUERYSTRING>",
            "WorkGroup": "<ATHENA_WORKGROUP>"
        },
        "Type": "Task",
        "End": true
      }
```

```
    }
  }
```

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

AthenaStartQueryExecution

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:startQueryExecution",
                "athena:stopQueryExecution",
                "athena:getQueryExecution",
                "athena:getDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-2:123456789012:workgroup/stepfunctions-athena-
sample-project-workgroup-26ujlyawxg",
                "arn:aws:athena:us-east-2:123456789012:datacatalog/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:DeleteDatabase",
                "glue:CreateTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition"
            ],
            "Resource": [
                "arn:aws::glue:us-east-2:123456789012:catalog",
                "arn:aws::glue:us-east-2:123456789012:database/*",
                "arn:aws::glue:us-east-2:123456789012:table/*",
                "arn:aws::glue:us-east-2:123456789012:userDefinedFunction/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Amazon EKS 클러스터 관리

이 샘플 프로젝트에서는 Step Functions 및 Amazon Elastic Kubernetes Service를 사용하여 노드 그룹이 있는 Amazon EKS 클러스터를 만들고 Amazon EKS에서 작업을 실행한 다음 출력을 검사하는 방법을 보여줍니다. 완료되면 노드 그룹과 Amazon EKS 클러스터가 제거됩니다.

Step Functions와 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 다른 서비스와 AWS Step Functions 함께 사용
- Step Functions를 사용하여 Amazon EKS 호출

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 Amazon EKS 요금을 참조하십시오.

1단계: 상태 시스템 만들기 및 리소스 프로비저닝

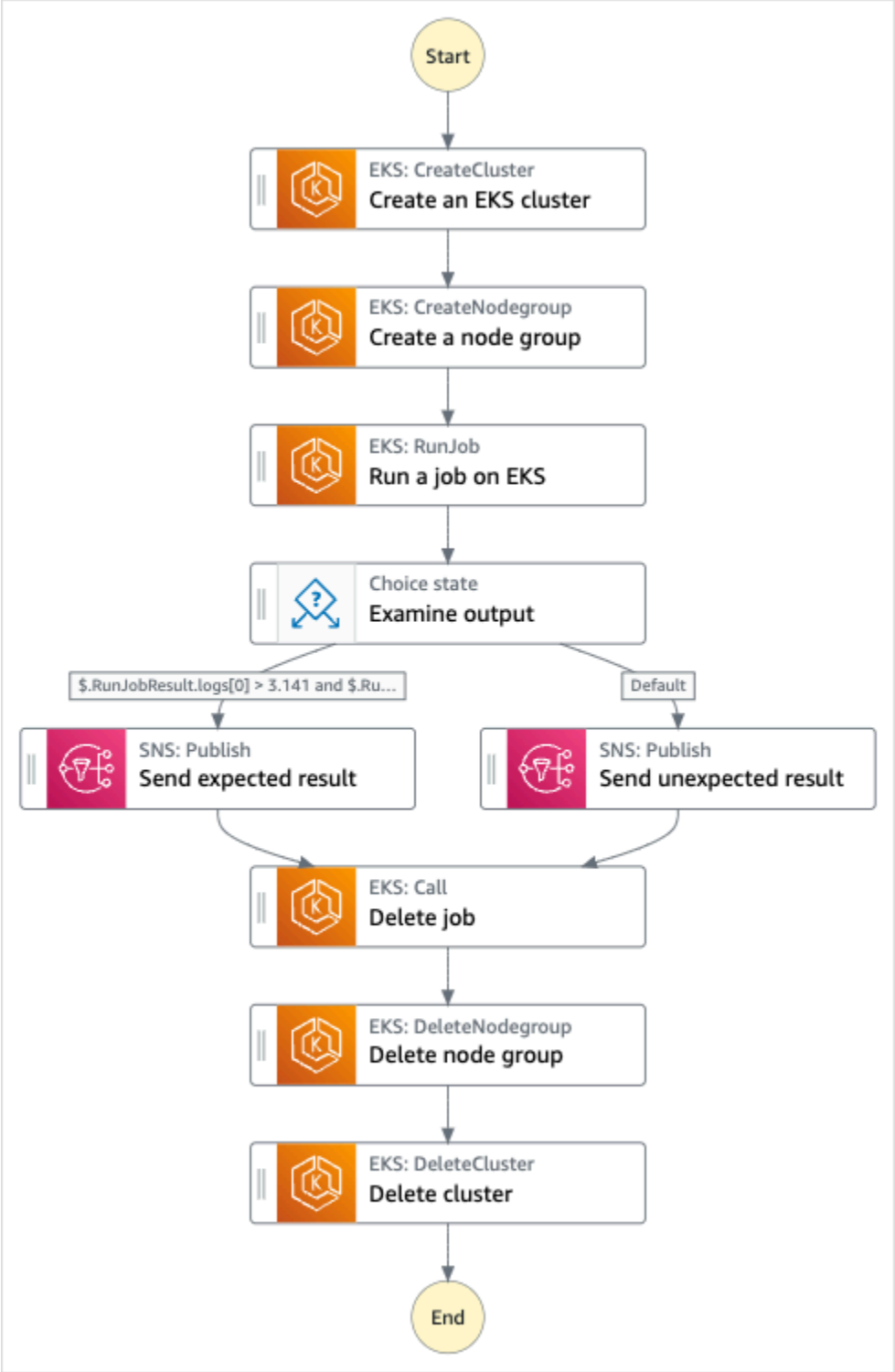
1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Manage an EKS cluster`**를 입력한 다음 반환된 검색 결과에서 EKS 클러스터 관리를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon Elastic Kubernetes Service 클러스터
   - Amazon SNS 주제

- AWS Step Functions 상태 머신
- 관련된 AWS Identity and Access Management(IAM) 역할

다음 이미지에서는 EKS 클러스터 관리 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

   - 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

     > **Tip**
     >
     > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

     준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

     이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

     샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

     > **Important**
     >
     > 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

      > **Note**
      >
      > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

      데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

      > **Note**
      >
      > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

   3. 실행 시작을 선택합니다.
   4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

      실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

Amazon EKS 클러스터와 노드 그룹을 만들면 이 샘플 프로젝트의 상태 시스템이 Amazon EKS와 통합되고 SNS 주제를 사용하여 결과를 반환합니다.

이 예제 상태 시스템을 살펴보고 Step Functions에서 Amazon EKS 클러스터와 노드 그룹을 관리하는 방법을 확인합니다.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "Comment": "An example of the Amazon States Language for running Amazon EKS Cluster",
  "StartAt": "Create an EKS cluster",
  "States": {
    "Create an EKS cluster": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createCluster.sync",
      "Parameters": {
        "Name": "ExampleCluster",
        "ResourcesVpcConfig": {
          "SubnetIds": [
            "subnet-0aacf887d9f00e6a7",
            "subnet-0e5fc41e7507194ab"
          ]
        },
        "RoleArn": "arn:aws:iam::111122223333:role/StepFunctionsSample-EKSClusterManag-
EKSServiceRole-ANPAJ2UCCR6DPCEXAMPLE"
      },
      "Retry": [{
        "ErrorEquals": [ "States.ALL" ],
        "IntervalSeconds": 30,
        "MaxAttempts": 2,
        "BackoffRate": 2
      }],
      "ResultPath": "$.eks",
      "Next": "Create a node group"
    },
    "Create a node group": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:createNodegroup.sync",
      "Parameters": {
        "ClusterName": "ExampleCluster",
```

```
        "NodegroupName": "ExampleNodegroup",
        "NodeRole": "arn:aws:iam::111122223333:role/StepFunctionsSample-EKSClusterMan-
NodeInstanceRole-ANPAJ2UCCR6DPCEXAMPLE",
        "Subnets": [
          "subnet-0aacf887d9f00e6a7",
          "subnet-0e5fc41e7507194ab"]
      },
      "Retry": [{
        "ErrorEquals": [ "States.ALL" ],
        "IntervalSeconds": 30,
        "MaxAttempts": 2,
        "BackoffRate": 2
      }],
      "ResultPath": "$.nodegroup",
      "Next": "Run a job on EKS"
    },
    "Run a job on EKS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:runJob.sync",
      "Parameters": {
        "ClusterName": "ExampleCluster",
        "CertificateAuthority.$": "$.eks.Cluster.CertificateAuthority.Data",
        "Endpoint.$": "$.eks.Cluster.Endpoint",
        "LogOptions": {
          "RetrieveLogs": true
        },
        "Job": {
          "apiVersion": "batch/v1",
          "kind": "Job",
          "metadata": {
            "name": "example-job"
          },
          "spec": {
            "backoffLimit": 0,
            "template": {
              "metadata": {
                "name": "example-job"
              },
              "spec": {
                "containers": [
                  {
                    "name": "pi-20",
                    "image": "perl",
                    "command": [
```

```
                    "perl"
                  ],
                  "args": [
                    "-Mbignum=bpi",
                    "-wle",
                    "print '{ ' . '\"pi\": '. bpi(20) . ' }';"
                  ]
                }
              ],
              "restartPolicy": "Never"
            }
          }
        }
      }
    },
    "ResultSelector": {
      "status.$": "$.status",
      "logs.$": "$.logs..pi"
    },
    "ResultPath": "$.RunJobResult",
    "Next": "Examine output"
  },
  "Examine output": {
    "Type": "Choice",
    "Choices": [
      {
        "And": [
          {
            "Variable": "$.RunJobResult.logs[0]",
            "NumericGreaterThan": 3.141
          },
          {
            "Variable": "$.RunJobResult.logs[0]",
            "NumericLessThan": 3.142
          }
        ],
        "Next": "Send expected result"
      }
    ],
    "Default": "Send unexpected result"
  },
  "Send expected result": {
    "Type": "Task",
    "Resource": "arn:aws:states:::sns:publish",
```

```
      "Parameters": {
        "TopicArn": "arn:aws:sns:sa-east-1:111122223333:StepFunctionsSample-
EKSClusterManagement123456789012-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE",
        "Message": {
          "Input.$": "States.Format('Saw expected value for pi: {}',
 $.RunJobResult.logs[0])"
        }
      },
      "ResultPath": "$.SNSResult",
      "Next": "Delete job"
    },
    "Send unexpected result": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "TopicArn": "arn:aws:sns:sa-east-1:111122223333:StepFunctionsSample-
EKSClusterManagement123456789012-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE",
        "Message": {
          "Input.$": "States.Format('Saw unexpected value for pi: {}',
 $.RunJobResult.logs[0])"
        }
      },
      "ResultPath": "$.SNSResult",
      "Next": "Delete job"
    },
    "Delete job": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:call",
      "Parameters": {
        "ClusterName": "ExampleCluster",
        "CertificateAuthority.$": "$.eks.Cluster.CertificateAuthority.Data",
        "Endpoint.$": "$.eks.Cluster.Endpoint",
        "Method": "DELETE",
        "Path": "/apis/batch/v1/namespaces/default/jobs/example-job"
      },
      "ResultSelector": {
        "status.$": "$.ResponseBody.status"
      },
      "ResultPath": "$.DeleteJobResult",
      "Next": "Delete node group"
    },
    "Delete node group": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteNodegroup.sync",
```

```
      "Parameters": {
        "ClusterName": "ExampleCluster",
        "NodegroupName": "ExampleNodegroup"
      },
      "Next": "Delete cluster"
    },
    "Delete cluster": {
      "Type": "Task",
      "Resource": "arn:aws:states:::eks:deleteCluster.sync",
      "Parameters": {
        "Name": "ExampleCluster"
      },
      "End": true
    }
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:CreateCluster"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:DeleteCluster"
            ],
```

```
            "Resource": "arn:aws:eks:sa-east-1:111122223333:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::111122223333:role/StepFunctionsSample-EKSClusterManag-
EKSServiceRole-ANPAJ2UCCR6DPCEXAMPLE"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:sa-east-1:111122223333:StepFunctionsSample-
EKSClusterManagement123456789012-SNSTopic-ANPAJ2UCCR6DPCEXAMPLE"
            ]
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

API Gateway 직접 호출

이 샘플 프로젝트에서는 Step Functions를 사용하여 API Gateway를 직접적으로 호출하고 호출이 성공했는지 확인하는 방법을 보여줍니다.

API Gateway 및 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 다른 서비스와 AWS Step Functions 함께 사용
- Step Functions를 사용하여 API Gateway 직접 호출

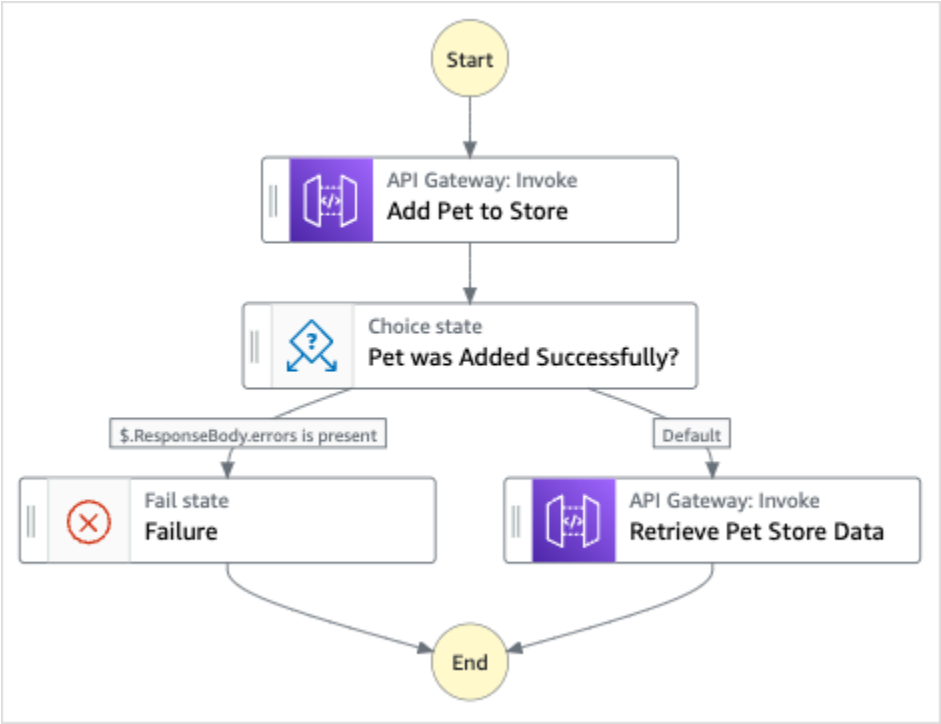
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Make a call to API Gateway`**를 입력한 다음 반환된 검색 결과에서 API Gateway 호출을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - 상태 시스템에서 호출하는 Amazon API Gateway REST API입니다.
   - AWS Step Functions 상태 머신
   - 관련된 AWS Identity and Access Management(IAM) 역할

   다음 이미지에서는 API Gateway 호출 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

   - 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

        > **Note**
        >
        > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

    2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

        데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

> **Note**
>
> 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

API Gateway REST API를 직접적으로 호출하고 필요한 모든 파라미터를 전달하면 이 샘플 프로젝트의 상태 시스템은 API Gateway와 통합됩니다.

이 예제 상태 시스템을 살펴보고 Step Functions가 API Gateway와 상호 작용하는 방식을 알아봅니다.

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "Comment": "Calling APIGW REST Endpoint",
  "StartAt": "Add Pet to Store",
  "States": {
    "Add Pet to Store": {
      "Type": "Task",
      "Resource": "arn:aws:states:::apigateway:invoke",
      "Parameters": {
        "ApiEndpoint": "<POST_PETS_API_ENDPOINT>",
        "Method": "POST",
        "Stage": "default",
        "Path": "pets",
        "RequestBody.$": "$.NewPet",
        "AuthType": "IAM_ROLE"
```

```
      },
      "ResultSelector": {
        "ResponseBody.$": "$.ResponseBody"
      },
      "Next": "Pet was Added Successfully?"
    },
    "Pet was Added Successfully?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.ResponseBody.errors",
          "IsPresent": true,
          "Next": "Failure"
        }
      ],
      "Default": "Retrieve Pet Store Data"
    },
    "Failure": {
      "Type": "Fail"
    },
    "Retrieve Pet Store Data": {
      "Type": "Task",
      "Resource": "arn:aws:states:::apigateway:invoke",
      "Parameters": {
        "ApiEndpoint": "<GET_PETS_API_ENDPOINT>",
        "Method": "GET",
        "Stage": "default",
        "Path": "pets",
        "AuthType": "IAM_ROLE"
      },
      "ResultSelector": {
        "Pets.$": "$.ResponseBody"
      },
      "ResultPath": "$.ExistingPets",
      "End": true
    }
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-west-1:111122223333:c8hqe4kdg5/default/GET/
pets",
                "arn:aws:execute-api:us-west-1:111122223333:c8hqe4kdg5/default/POST/
pets"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

API Gateway 통합을 사용하여 Fargate에서 실행되는 마이크로서비스 직접 호출

이 샘플 프로젝트는 Step Functions를 사용하여 API Gateway를 호출하여 서비스와 상호 작용하고 호출 성공 여부를 확인하는 방법을 보여줍니다. AWS Fargate

API Gateway 및 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 다른 서비스와 AWS Step Functions 함께 사용
- Step Functions를 사용하여 API Gateway 직접 호출

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 요금을 참조하십시오.

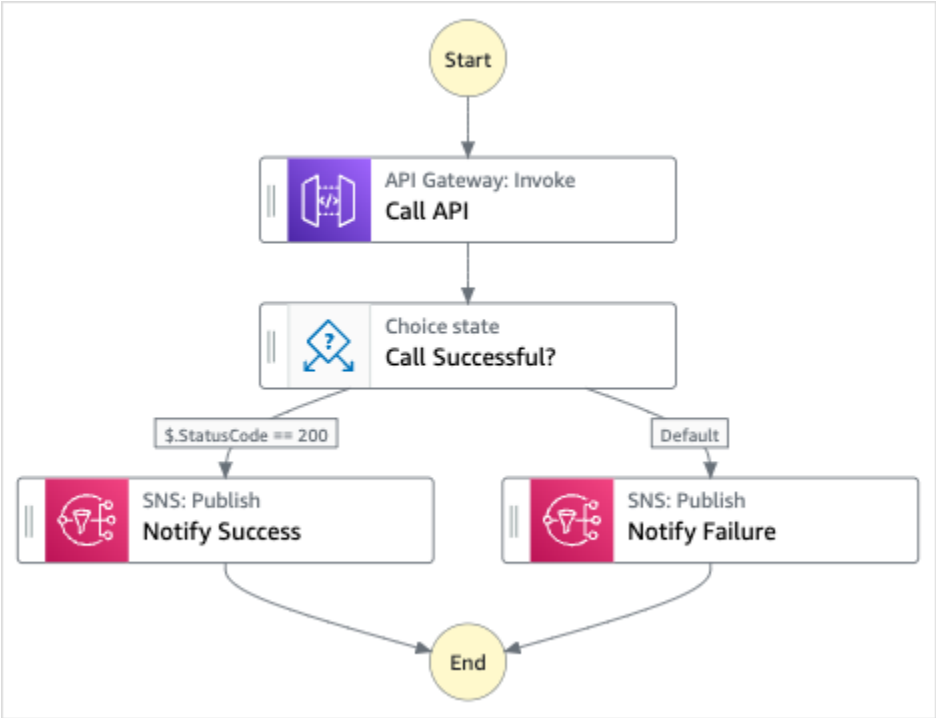
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Call a microservice with API Gateway`**를 입력한 다음 반환된 검색 결과에서 API Gateway를 사용하여 마이크로서비스 호출을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - 상태 시스템에서 호출하는 Amazon API Gateway HTTP API.
   - Amazon API Gateway Amazon VPC 링크.
   - Amazon Virtual Private Cloud.
   - Application Load Balancer.
   - Fargate 클러스터.
   - Amazon SNS 주제
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할
   - 이러한 리소스가 함께 작동하려면 몇 가지 추가 서비스가 필요합니다.

   다음 이미지에서는 API Gateway를 사용하여 마이크로서비스 호출 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

   - 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

        > **Note**
        >
        > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

    2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

        데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

> **Note**
>
> 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 시스템은 Fargate의 서비스에 연결된 API Gateway HTTP API를 직접적으로 호출하여 API Gateway와 통합됩니다. 이는 프라이빗 서브넷에서 호스팅되며 프라이빗 애플리케이션 로드 밸런서를 통해 액세스됩니다.

이 예제 상태 시스템을 살펴보고 Step Functions가 API Gateway와 상호 작용하고 결과를 반환하는 방식을 알아봅니다.

다른 AWS 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "Comment": "Calling APIGW HTTP Endpoint",
  "StartAt": "Call API",
  "States": {
    "Call API": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::apigateway:invoke",
      "Parameters": {
        "ApiEndpoint": "<API_ENDPOINT>",
        "Method": "GET",
        "AuthType": "IAM_ROLE"
      },
```

```
      "Next": "Call Successful?"
    },
    "Call Successful?": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.StatusCode",
          "NumericEquals": 200,
          "Next": "Notify Success"
        }
      ],
      "Default": "Notify Failure"
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::sns:publish",
      "Parameters": {
        "Message": "Call was successful",
        "TopicArn": "<SNS_TOPIC_ARN>"
      },
      "End": true
    },
    "Notify Failure": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::sns:publish",
      "Parameters": {
        "Message": "Call was not successful",
        "TopicArn": "<SNS_TOPIC_ARN>"
      },
      "End": true
    }
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 상태 시스템 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함됩니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-east-1:111122223333:apigw-ecs-sample-2000-
SNSTopic-444455556666"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:111122223333:444444444444/*/GET/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "ec2:AttachNetworkInterface",
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:Describe*",
                "ec2:DetachNetworkInterface",
                "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:Describe*",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:RegisterTargets"
            ],
            "Resource": "*",
```

```
            "Effect": "Allow"
        }
    ]
}
```

```
{
    "Statement": [
        {
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

에 맞춤 이벤트 보내기 EventBridge

이 샘플 프로젝트는 Step Functions를 사용하여 여러 대상 (Amazon EventBridge, Amazon Simple Notification Service AWS Lambda, Amazon Simple Queue Service) 이 있는 규칙과 일치하는 이벤트 버스로 사용자 지정 이벤트를 보내는 방법을 보여줍니다.

Step Functions와 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 다른 서비스와 AWS Step Functions 함께 사용
- EventBridge Step 함수를 사용한 호출

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.

신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 EventBridge 요금을 참조하십시오.

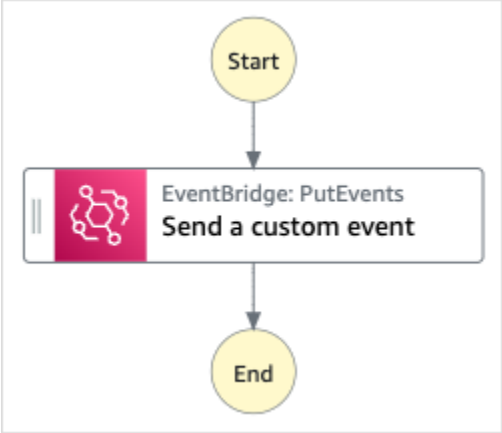
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Send a custom event to EventBridge`**를 입력한 다음 반환된 검색 결과에서 EventBridge로 사용자 지정 이벤트 전송을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon EventBridge 이벤트
   - Amazon SNS 주제
   - Amazon SQS 대기열
   - Lambda 기능
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 EventBridge로 사용자 지정 이벤트 전송 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

   - 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

     > **Tip**
     >
     > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

     준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

      > **Note**
      >
      > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

      데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

      > **Note**
      >
      > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

   3. 실행 시작을 선택합니다.

4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 커스텀 이벤트를 이벤트 EventBridge 버스로 보내는 방식으로 통합됩니다. EventBridge 이벤트 버스로 전송된 이벤트는 Amazon SNS 주제 및 Amazon SQS 대기열에 메시지를 보내는 Lambda 함수를 트리거하는 EventBridge 규칙과 일치합니다.

이 예제 스테이트 머신을 살펴보고 Step Functions가 어떻게 관리되는지 살펴보세요 EventBridge.

다른 AWS 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
  "Comment": "An example of the Amazon States Language for sending a custom event to
 Amazon EventBridge",
  "StartAt": "Send a custom event",
  "States": {
    "Send a custom event": {
      "Resource": "arn:<PARTITION>:states:::events:putEvents",
      "Type": "Task",
      "Parameters": {
        "Entries": [{
          "Detail": {
            "Message": "Hello from Step Functions!"
          },
          "DetailType": "MessageFromStepFunctions",
          "EventBusName": "<EVENT_BUS_NAME>",
          "Source": "my.statemachine"
        }]
      },
      "End": true
    }
  }
```

```
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "events:PutEvents"
            ],
            "Resource": [
                "arn:aws:events:us-east-1:1234567890:event-bus/stepfunctions-
sampleproject-eventbus"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

동기 Express 워크플로 간접 호출

이 샘플 프로젝트에서는 Amazon API Gateway를 통해 동기 Express 워크플로를 간접적으로 호출하여 직원 데이터베이스를 관리하는 방법을 보여줍니다.

이 프로젝트에서 Step Functions는 API Gateway 엔드포인트를 사용하여 Step Functions 동기 Express 워크플로를 시작합니다. 그런 다음 DynamoDB를 사용하여 직원 데이터베이스에서 직원을 검색, 추가 및 제거합니다.

Step Functions 동기 Express 워크플로에 대한 자세한 내용은 동기 및 비동기 Express 워크플로 섹션을 참조하세요.

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 Step Functions 요금을 참조하십시오.

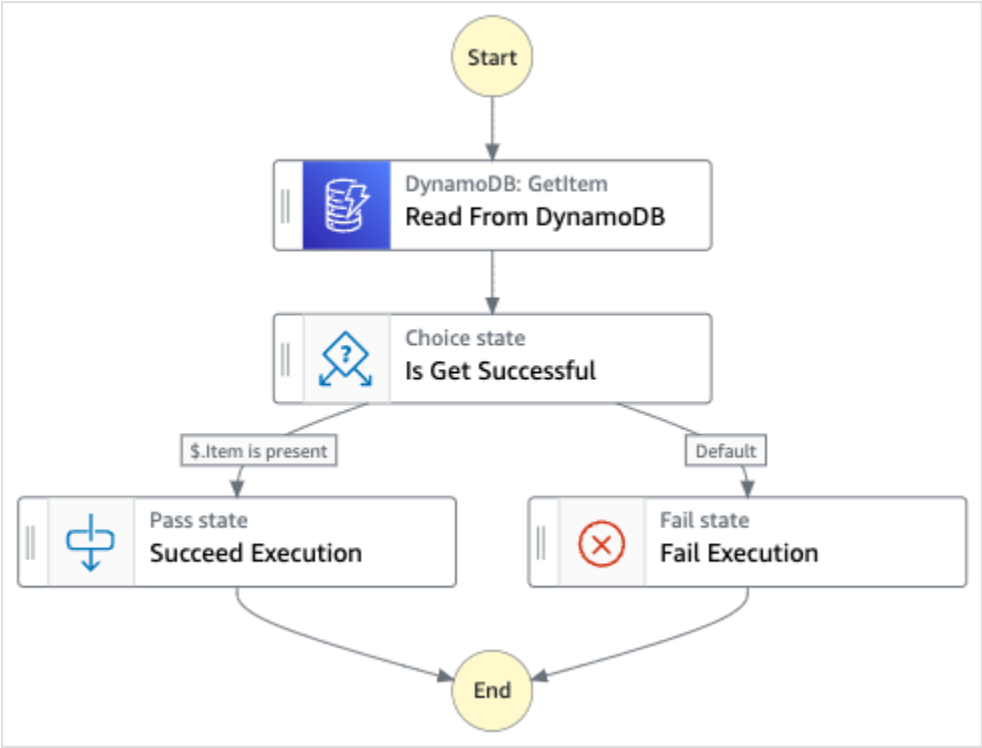
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Invoke Synchronous Express Workflows through API Gateway`**를 입력한 다음 반환된 검색 결과에서 API Gateway를 통해 동기 Express 워크플로 간접 호출을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - 상태 시스템에서 직접적으로 호출하는 Amazon API Gateway HTTPS API
   - Amazon DynamoDB 테이블
   - 세 개의 AWS Step Functions 스테이트 머신.
   - 관련 AWS Identity and Access Management (IAM) 역할.

   다음 이미지에서는 API Gateway를 통해 동기 Express 워크플로 간접 호출 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

   - 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

        > **Note**
        >
        > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

    2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

        데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

> **Note**
>
> 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 시스템은 API Gateway를 사용하여 동기 Express 워크플로를 간접적으로 호출해 API Gateway 및 DynamoDB와 통합된 다음 DynamoDB를 사용하여 직원 데이터베이스를 업데이트하거나 직원 데이터베이스에서 데이터를 읽습니다.

이 예제 상태 시스템을 살펴보고 Step Functions가 DynamoDB에서 읽고 직원 정보를 검색하는 방법을 확인합니다.

API Gateway를 사용하여 Step Functions를 간접적으로 호출하는 방법에 대한 자세한 내용은 다음을 참조하세요.

- Step Functions를 사용하여 API Gateway 직접 호출
- API Gateway 개발자 안내서의 프라이빗 게이트웨이를 간접적으로 호출하는 방법

다른 AWS 서비스를 제어할 AWS Step Functions 수 있는 방법에 대한 자세한 내용은 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

```
{
  "Comment": "This state machine returns an employee entry from DynamoDB",
  "StartAt": "Read From DynamoDB",
  "States": {
    "Read From DynamoDB": {
```

```
      "Type": "Task",
      "Resource": "arn:aws:states:::dynamodb:getItem",
      "Parameters": {
        "TableName": "StepFunctionsSample-
SynchronousExpressWorkflowAKIAIOSFODNN7EXAMPLE-DynamoDBTable-ANPAJ2UCCR6DPCEXAMPLE",
        "Key": {
          "EmployeeId": {"S.$": "$.employee"}
        }
      },
      "Retry": [
        {
          "ErrorEquals": [
            "DynamoDB.AmazonDynamoDBException"
          ],
          "IntervalSeconds": 3,
          "MaxAttempts": 2,
          "BackoffRate": 1.5
        }
      ],
      "Next": "Is Get Successful"
    },
    "Is Get Successful": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.Item",
          "IsPresent": true,
          "Next": "Succeed Execution"
        }
      ],
      "Default": "Fail Execution"
    },
    "Succeed Execution": {
      "Type": "Pass",
      "Parameters" : {
        "employee.$": "$.Item.EmployeeId.S",
        "jobTitle.$": "$.Item.JobTitle.S"
      },
      "End": true
    },
    "Fail Execution": {
      "Type": "Fail",
      "Error": "EmployeeDoesNotExist"
    }
```

```
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem"
```

```
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111122223333:table/Write"
            ]
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Amazon Redshift(Lambda, Amazon Redshift Data API)를 사용하여 ETL/ELT 워크플로 실행

이 샘플 프로젝트에서는 Step Functions 및 Amazon Redshift Data API를 사용하여 Amazon Redshift 데이터 웨어하우스로 데이터를 로드하는 ETL/ELT 워크플로를 실행하는 방법을 보여줍니다.

이 프로젝트에서 Step Functions는 AWS Lambda 함수와 Amazon Redshift Data API를 사용하여 필요한 데이터베이스 객체를 생성하고 예제 데이터 세트를 생성한 다음 차원 테이블과 팩트 테이블을 로드하는 두 개의 작업을 병렬로 실행합니다. 두 차원 로드 작업 모두 성공적으로 종료되면 Step Functions는 팩트 테이블에 대한 로드 작업을 실행하고 검증 작업을 실행한 다음 Amazon Redshift 클러스터를 일시 중지합니다.

> **Note**
>
> ETL 로직을 수정하여 Amazon S3와 같은 다른 소스에서 데이터를 받을 수 있으므로 COPY 명령을 사용하여 Amazon S3에서 Amazon Redshift 테이블로 데이터를 복사할 수 있습니다.

Amazon Redshift 및 Step Functions 서비스 통합에 대한 자세한 내용은 다음을 참조하세요.

- 다른 서비스와 AWS Step Functions 함께 사용
- Amazon Redshift Data API 사용
- Amazon Redshift Data API 서비스
- Lambda를 사용하는 Step Functions 상태 시스템 만들기
- 다음에 대한 IAM 정책:
  - 다음에 대한 IAM 정책 AWS Lambda

- Amazon Redshift Data API에 대한 액세스 권한 부여

> **Note**
>
> 이 샘플 프로젝트를 사용할 때 요금이 발생할 수 있습니다.
> 신규 AWS 사용자의 경우 프리 티어를 사용할 수 있습니다. 이 계층에서 특정 사용 수준 미만의 서비스는 무료입니다. AWS 비용 및 프리 티어에 대한 자세한 내용은 AWS Step Functions 요금을 참조하십시오.

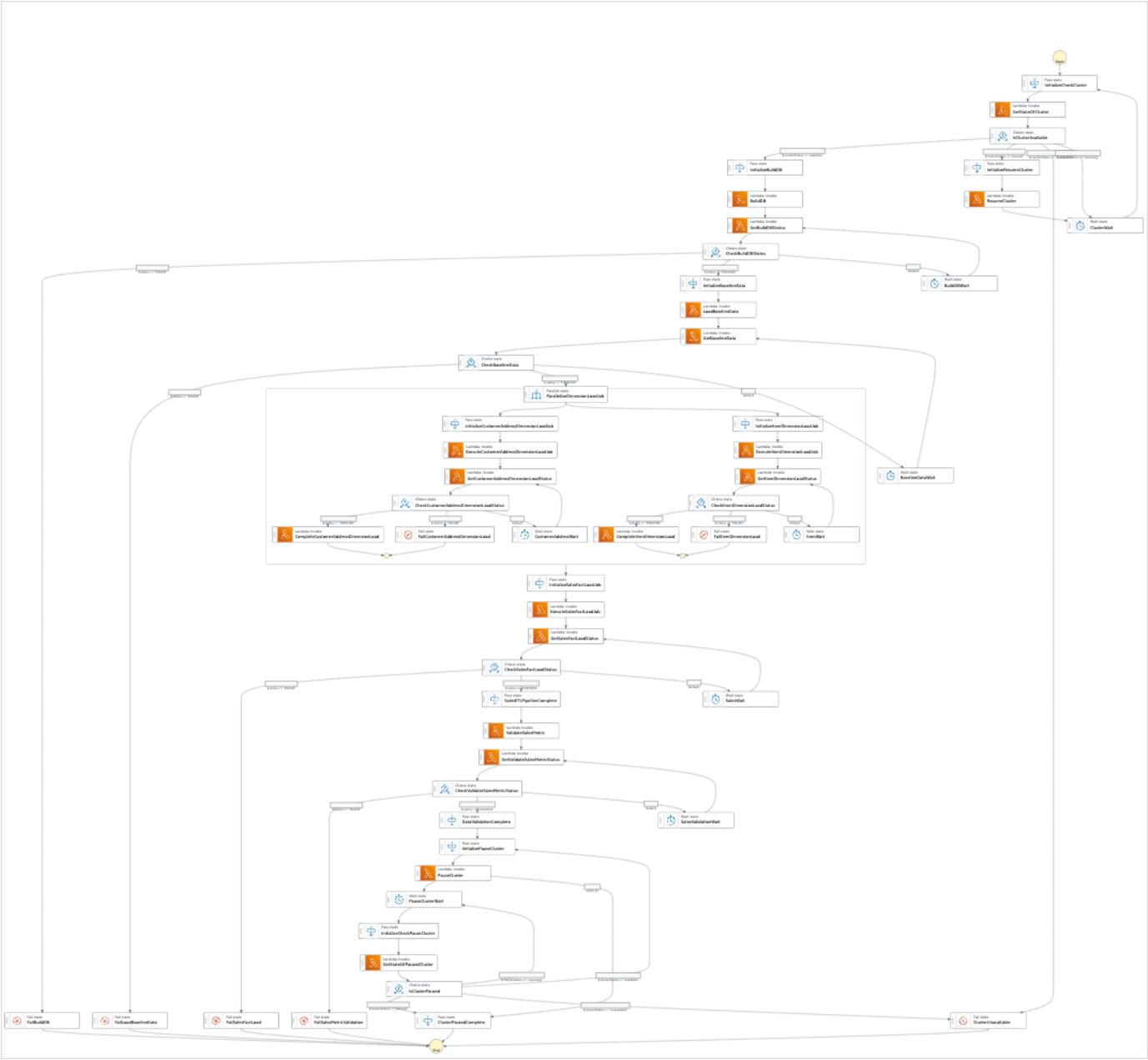
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`ETL job in Amazon Redshift`**를 입력한 다음 반환된 검색 결과에서 Amazon Redshift에서의 ETL 작업을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Amazon Redshift 클러스터
   - 2개의 Lambda 함수
   - Amazon Redshift 스키마
   - Amazon Redshift 테이블 5개
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할.

   다음 이미지에서는 Amazon Redshift에서의 ETL 작업 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트

합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.
    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 ETL 로직을 해당 리소스에 InputPath 직접 전달하고 Amazon Redshift Data API를 사용하여 비동기적으로 실행되는 AWS Lambda 방식으로 통합됩니다.

이 예제 상태 머신을 살펴보고 Step Functions가 Amazon Redshift 데이터 API를 제어하는 AWS Lambda 방법을 살펴보십시오.

다른 AWS 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
```

```
"Comment": "A simple ETL workflow for loading dimension and fact tables",
  "StartAt": "InitializeCheckCluster",
  "States": {
    "InitializeCheckCluster": {
      "Type": "Pass",
      "Next": "GetStateOfCluster",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "operation": "status"
        }
      }
    },
    "GetStateOfCluster": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "IsClusterAvailable",
      "InputPath": "$",
      "ResultPath": "$.clusterStatus"
    },
    "IsClusterAvailable": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "available",
          "Next": "InitializeBuildDB"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "paused",
          "Next": "InitializeResumeCluster"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "unavailable",
          "Next": "ClusterUnavailable"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "resuming",
```

```
          "Next": "ClusterWait"
        }
      ]
    },
    "ClusterWait": {
      "Type": "Wait",
      "Seconds": 720,
      "Next": "InitializeCheckCluster"
    },
    "InitializeResumeCluster": {
      "Type": "Pass",
      "Next": "ResumeCluster",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "operation": "resume"
        }
      }
    },
    "ResumeCluster": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "ClusterWait",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "InitializeBuildDB": {
      "Type": "Pass",
      "Next": "BuildDB",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "redshift_database": "dev",
          "redshift_user": "awsuser",
          "redshift_schema": "tpcds",
          "action": "build_database",
          "sql_statement": [
            "create schema if not exists {0} authorization {1};",
            "create table if not exists {0}.customer",
            "(c_customer_sk          int4     not null encode az64",
            ",c_customer_id          char(16) not null encode zstd",
```

```
",c_current_addr_sk      int4              encode az64",
",c_first_name           char(20)          encode zstd",
",c_last_name            char(30)          encode zstd",
",primary key (c_customer_sk)",
") distkey(c_customer_sk);",
"--",
"create table if not exists {0}.customer_address",
"(ca_address_sk    int4     not null encode az64",
",ca_address_id    char(16) not null encode zstd",
",ca_state         char(2)           encode zstd",
",ca_zip           char(10)          encode zstd",
",ca_country       varchar(20)       encode zstd",
",primary key (ca_address_sk)",
") distkey(ca_address_sk);",
"--",
"create table if not exists {0}.date_dim",
"(d_date_sk           integer  not null encode az64",
",d_date_id           char(16) not null encode zstd",
",d_date              date              encode az64",
",d_day_name          char(9)           encode zstd",
",primary key (d_date_sk)",
") diststyle all;",
"--",
"create table if not exists {0}.item",
"(i_item_sk        int4     not null encode az64",
",i_item_id        char(16) not null encode zstd",
",i_rec_start_date date              encode az64",
",i_rec_end_date   date              encode az64",
",i_current_price  numeric(7,2)      encode az64",
",i_category       char(50)          encode zstd",
",i_product_name   char(50)          encode zstd",
",primary key (i_item_sk)",
") distkey(i_item_sk) sortkey(i_category);",
"--",
"create table if not exists {0}.store_sales",
"(ss_sold_date_sk       int4",
",ss_item_sk            int4 not null encode az64",
",ss_customer_sk        int4          encode az64",
",ss_addr_sk            int4          encode az64",
",ss_store_sk           int4          encode az64",
",ss_ticket_number      int8 not null encode az64",
",ss_quantity           int4          encode az64",
",ss_net_paid           numeric(7,2)  encode az64",
",ss_net_profit         numeric(7,2)  encode az64",
```

```
            ",primary key (ss_item_sk, ss_ticket_number)",
            ") distkey(ss_item_sk) sortkey(ss_sold_date_sk);"
          ]
        }
      }
    },
    "BuildDB": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "GetBuildDBStatus",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "GetBuildDBStatus": {
      "Type": "Task",
      "Next": "CheckBuildDBStatus",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "InputPath": "$",
      "ResultPath": "$.status"
    },
    "CheckBuildDBStatus": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.status",
          "StringEquals": "FAILED",
          "Next": "FailBuildDB"
        },
        {
          "Variable": "$.status",
          "StringEquals": "FINISHED",
          "Next": "InitializeBaselineData"
        }
      ],
      "Default": "BuildDBWait"
    },
    "BuildDBWait": {
      "Type": "Wait",
```

```
      "Seconds": 15,
      "Next": "GetBuildDBStatus"
    },
    "FailBuildDB": {
      "Type": "Fail",
      "Cause": "Database Build Failed",
      "Error": "Error"
    },
    "InitializeBaselineData": {
      "Type": "Pass",
      "Next": "LoadBaselineData",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "redshift_database": "dev",
          "redshift_user": "awsuser",
          "redshift_schema": "tpcds",
          "action": "load_baseline_data",
          "sql_statement": [
            "begin transaction;",
            "truncate table {0}.customer;",
            "insert into {0}.customer
(c_customer_sk,c_customer_id,c_current_addr_sk,c_first_name,c_last_name)",
            "values",
            "(7550,'AAAAAAAAOHNBAAAA',9264662,'Michelle','Deaton'),",
            "(37079,'AAAAAAAAHNAJAAAA',13971208,'Michael','Simms'),",
            "(40626,'AAAAAAAACLOJAAAA',1959255,'Susan','Ryder'),",
            "(2142876,'AAAAAAAAMJCLACAA',7644556,'Justin','Brown');",
            "analyze {0}.customer;",
            "--",
            "truncate table {0}.customer_address;",
            "insert into {0}.customer_address
(ca_address_sk,ca_address_id,ca_state,ca_zip,ca_country)",
            "values",
            "(13971208,'AAAAAAAAIAPCFNAA','NE','63451','United States'),",
            "(7644556,'AAAAAAAAMIFKEHAA','SD','58883','United States'),",
            "(9264662,'AAAAAAAAGBOFNIAA','CA','99310','United States');",
            "analyze {0}.customer_address;",
            "--",
            "truncate table {0}.item;",
            "insert into {0}.item
(i_item_sk,i_item_id,i_rec_start_date,i_rec_end_date,i_current_price,i_category,i_product_name
            "values",
```

```
 "(3417,'AAAAAAAAIFNAAAAA','1997-10-27',NULL,14.29,'Electronics','ationoughtesepri
 '),",
            "(9615,'AAAAAAAAOIFCAAAA','1997-10-27',NULL,9.68,'Home','antioughtcallyn
 st'),",
            "(3693,'AAAAAAAAMGOAAAAA','2001-03-12',NULL,2.10,'Men','prin
 stcallypri'),",

 "(3630,'AAAAAAAAMCOAAAAA','2001-10-27',NULL,2.95,'Electronics','barpricallypri'),",

 "(16506,'AAAAAAAAIHAEAAAA','2001-10-27',NULL,3.85,'Home','callybaranticallyought'),",

 "(7866,'AAAAAAAAILOBAAAA','2001-10-27',NULL,12.60,'Jewelry','callycallyeingation');",
            "--",
            "analyze {0}.item;",
            "truncate table {0}.date_dim;",
            "insert into {0}.date_dim (d_date_sk,d_date_id,d_date,d_day_name)",
            "values",
            "(2450521,'AAAAAAAAJFEGFCAA','1997-03-13','Thursday'),",
            "(2450749,'AAAAAAAANDFGFCAA','1997-10-27','Monday'),",
            "(2451251,'AAAAAAAADDHGFCAA','1999-03-13','Saturday'),",
            "(2451252,'AAAAAAAAEDHGFCAA','1999-03-14','Sunday'),",
            "(2451981,'AAAAAAAANAKGFCAA','2001-03-12','Monday'),",
            "(2451982,'AAAAAAAAOAKGFCAA','2001-03-13','Tuesday'),",
            "(2452210,'AAAAAAAACPKGFCAA','2001-10-27','Saturday'),",
            "(2452641,'AAAAAAAABKMGFCAA','2003-01-01','Wednesday'),",
            "(2452642,'AAAAAAAACKMGFCAA','2003-01-02','Thursday');",
            "--",
            "analyze {0}.date_dim;",
            "--  commit and End transaction",
            "commit;",
            "end transaction;"
          ]
        }
      }
    },
    "LoadBaselineData": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "GetBaselineData",
      "InputPath": "$",
```

```
      "ResultPath": "$"
    },
    "GetBaselineData": {
      "Type": "Task",
      "Next": "CheckBaselineData",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "InputPath": "$",
      "ResultPath": "$.status"
    },
    "CheckBaselineData": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.status",
          "StringEquals": "FAILED",
          "Next": "FailLoadBaselineData"
        },
        {
          "Variable": "$.status",
          "StringEquals": "FINISHED",
          "Next": "ParallelizeDimensionLoadJob"
        }
      ],
      "Default": "BaselineDataWait"
    },
    "BaselineDataWait": {
      "Type": "Wait",
      "Seconds": 20,
      "Next": "GetBaselineData"
    },
    "FailLoadBaselineData": {
      "Type": "Fail",
      "Cause": "Load Baseline Data Failed",
      "Error": "Error"
    },
    "ParallelizeDimensionLoadJob": {
      "Type": "Parallel",
      "Next": "InitializeSalesFactLoadJob",
      "ResultPath": "$.status",
      "Branches": [
        {
```

```
          "StartAt": "InitializeCustomerAddressDimensionLoadJob",
          "States": {
            "InitializeCustomerAddressDimensionLoadJob": {
              "Type": "Pass",
              "Next": "ExecuteCustomerAddressDimensionLoadJob",
              "Result": {
                "input": {
                  "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
                  "redshift_database": "dev",
                  "redshift_user": "awsuser",
                  "redshift_schema": "tpcds",
                  "action": "load_customer_address",
                  "sql_statement": [
                    "begin transaction;",
                    "/* Create a staging table to hold the input data. Staging table is
created with BACKUP NO option for faster inserts and also data temporary */",
                    "drop table if exists {0}.stg_customer_address;",
                    "create table if not exists {0}.stg_customer_address",
                    "(ca_address_id    varchar(16)  encode zstd",
                    ",ca_state         varchar(2)   encode zstd",
                    ",ca_zip           varchar(10)  encode zstd",
                    ",ca_country       varchar(20)  encode zstd",
                    ")",
                    "backup no",
                    "diststyle even;",
                    "/* Ingest data from source */",
                    "insert into {0}.stg_customer_address
(ca_address_id,ca_state,ca_zip,ca_country)",
                    "values",
                    "('AAAAAAAACFBBAAAA','NE','','United States'),",
                    "('AAAAAAAAGAEFAAAA','NE','61749','United States'),",
                    "('AAAAAAAAPJKKAAAA','OK','','United States'),",
                    "('AAAAAAAAMIHGAAAA','AL','','United States');",
                    "/* Perform UPDATE for existing data with refreshed attribute
values */",
                    "update {0}.customer_address",
                    "   set ca_state = stg_customer_address.ca_state,",
                    "       ca_zip = stg_customer_address.ca_zip,",
                    "       ca_country = stg_customer_address.ca_country",
                    "  from {0}.stg_customer_address",
                    " where customer_address.ca_address_id =
stg_customer_address.ca_address_id;",
                    "/* Perform insert for new rows  */",
                    "insert into {0}.customer_address",
```

```
                    "(ca_address_sk",
                    ",ca_address_id",
                    ",ca_state",
                    ",ca_zip",
                    ",ca_country",
                    ")",
                    "with max_customer_address_sk as",
                    "(select max(ca_address_sk) max_ca_address_sk",
                    "from {0}.customer_address)",
                    "select row_number() over (order by
 stg_customer_address.ca_address_id) + max_customer_address_sk.max_ca_address_sk as
 ca_address_sk",
                    ",stg_customer_address.ca_address_id",
                    ",stg_customer_address.ca_state",
                    ",stg_customer_address.ca_zip",
                    ",stg_customer_address.ca_country",
                    "from {0}.stg_customer_address,",
                    "max_customer_address_sk",
                    "where stg_customer_address.ca_address_id not in (select
 customer_address.ca_address_id from {0}.customer_address);",
                    "/* Commit and End transaction */",
                    "commit;",
                    "end transaction;"
                  ]
                }
              }
            },
            "ExecuteCustomerAddressDimensionLoadJob": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "Next": "GetCustomerAddressDimensionLoadStatus",
              "InputPath": "$",
              "ResultPath": "$"
            },
            "GetCustomerAddressDimensionLoadStatus": {
              "Type": "Task",
              "Next": "CheckCustomerAddressDimensionLoadStatus",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
```

```
              "InputPath": "$",
              "ResultPath": "$.status"
            },
            "CheckCustomerAddressDimensionLoadStatus": {
              "Type": "Choice",
              "Choices": [
                {
                  "Variable": "$.status",
                  "StringEquals": "FAILED",
                  "Next": "FailCustomerAddressDimensionLoad"
                },
                {
                  "Variable": "$.status",
                  "StringEquals": "FINISHED",
                  "Next": "CompleteCustomerAddressDimensionLoad"
                }
              ],
              "Default": "CustomerAddressWait"
            },
            "CustomerAddressWait": {
              "Type": "Wait",
              "Seconds": 5,
              "Next": "GetCustomerAddressDimensionLoadStatus"
            },
            "CompleteCustomerAddressDimensionLoad": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "End": true
            },
            "FailCustomerAddressDimensionLoad": {
              "Type": "Fail",
              "Cause": "ETL Workflow Failed",
              "Error": "Error"
            }
          }
        },
        {
          "StartAt": "InitializeItemDimensionLoadJob",
          "States": {
            "InitializeItemDimensionLoadJob": {
              "Type": "Pass",
```

```
              "Next": "ExecuteItemDimensionLoadJob",
              "Result": {
                "input": {
                  "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
                  "redshift_database": "dev",
                  "redshift_user": "awsuser",
                  "redshift_schema": "tpcds",
                  "action": "load_item",
                  "sql_statement": [
                    "begin transaction;",
                    "/* Create a staging table to hold the input data. Staging table is
created with BACKUP NO option for faster inserts and also data temporary */",
                    "drop table if exists {0}.stg_item;",
                    "create table if not exists {0}.stg_item",
                    "(i_item_id        varchar(16) encode zstd",
                    ",i_rec_start_date date encode zstd",
                    ",i_rec_end_date   date encode zstd",
                    ",i_current_price  numeric(7,2) encode zstd",
                    ",i_category       varchar(50) encode zstd",
                    ",i_product_name   varchar(50) encode zstd",
                    ")",
                    "backup no",
                    "diststyle even;",
                    "/* Ingest data from source */",
                    "insert into {0}.stg_item",

"(i_item_id,i_rec_start_date,i_rec_end_date,i_current_price,i_category,i_product_name)",
                    "values",

"('AAAAAAAAABJBAAAA','2000-10-27',NULL,4.10,'Books','ationoughtesecally'),",
                    "('AAAAAAAAOPKBAAAA','2001-10-27',NULL,4.22,'Books','ableoughtn
stcally'),",
                    "('AAAAAAAAHGPAAAAA','1997-10-27',NULL,29.30,'Books','priesen
stpri'),",

"('AAAAAAAAICMAAAAA','2001-10-27',NULL,1.93,'Books','eseoughtoughtpri'),",

"('AAAAAAAAGPGBAAAA','2001-10-27',NULL,9.96,'Books','bareingeinganti'),",
                    "('AAAAAAAAANBEBAAAA','1997-10-27',NULL,2.25,'Music','n
steseoughtanti'),",

"('AAAAAAAACLAAAAAA','2001-10-27',NULL,1.71,'Home','bareingought'),",

"('AAAAAAAAOBBDAAAA','2001-10-27',NULL,5.55,'Books','callyationantiableought');",
```

```
                    "/
*************************************************************************************
                    "** Type 2 is maintained for i_current_price column.",
                    "** Update all attributes for the item when the price is not
 changed",
                    "** Sunset existing active item record with current i_rec_end_date
 and insert a new record when the price does not match",

 "*************************************************************************************
                    "update {0}.item",
                    "   set i_category = stg_item.i_category,",
                    "       i_product_name = stg_item.i_product_name",
                    "  from {0}.stg_item",
                    " where item.i_item_id = stg_item.i_item_id",
                    "   and item.i_rec_end_date is null",
                    "   and item.i_current_price = stg_item.i_current_price;",
                    "insert into {0}.item",
                    "(i_item_sk",
                    ",i_item_id",
                    ",i_rec_start_date",
                    ",i_rec_end_date",
                    ",i_current_price",
                    ",i_category",
                    ",i_product_name",
                    ")",
                    "with max_item_sk as",
                    "(select max(i_item_sk) max_item_sk",
                    "   from {0}.item)",
                    "select row_number() over (order by stg_item.i_item_id) +
 max_item_sk as i_item_sk",
                    "      ,stg_item.i_item_id",
                    "      ,trunc(sysdate) as i_rec_start_date",
                    "      ,null as i_rec_end_date",
                    "      ,stg_item.i_current_price",
                    "      ,stg_item.i_category",
                    "      ,stg_item.i_product_name",
                    "  from {0}.stg_item, {0}.item, max_item_sk",
                    " where item.i_item_id = stg_item.i_item_id",
                    "   and item.i_rec_end_date is null",
                    "   and item.i_current_price <> stg_item.i_current_price;",
                    "/* Sunset penultimate records that were inserted as type 2 */",
                    "update {0}.item",
                    "   set i_rec_end_date = trunc(sysdate)",
                    "  from {0}.stg_item",
```

```
                    " where item.i_item_id = stg_item.i_item_id",
                    "   and item.i_rec_end_date is null",
                    "   and item.i_current_price <> stg_item.i_current_price;",
                    "/* Commit and End transaction */",
                    "commit;",
                    "end transaction;"
                  ]
                }
              }
            },
            "ExecuteItemDimensionLoadJob": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "Next": "GetItemDimensionLoadStatus",
              "InputPath": "$",
              "ResultPath": "$"
            },
            "GetItemDimensionLoadStatus": {
              "Type": "Task",
              "Next": "CheckItemDimensionLoadStatus",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "InputPath": "$",
              "ResultPath": "$.status"
            },
            "CheckItemDimensionLoadStatus": {
              "Type": "Choice",
              "Choices": [
                {
                  "Variable": "$.status",
                  "StringEquals": "FAILED",
                  "Next": "FailItemDimensionLoad"
                },
                {
                  "Variable": "$.status",
                  "StringEquals": "FINISHED",
                  "Next": "CompleteItemDimensionLoad"
                }
              ],
```

```
              "Default": "ItemWait"
            },
            "ItemWait": {
              "Type": "Wait",
              "Seconds": 5,
              "Next": "GetItemDimensionLoadStatus"
            },
            "CompleteItemDimensionLoad": {
              "Type": "Task",
              "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
              "TimeoutSeconds": 180,
              "HeartbeatSeconds": 60,
              "End": true
            },
            "FailItemDimensionLoad": {
              "Type": "Fail",
              "Cause": "ETL Workflow Failed",
              "Error": "Error"
            }
          }
        }
      ]
    },
    "InitializeSalesFactLoadJob": {
      "Type": "Pass",
      "Next": "ExecuteSalesFactLoadJob",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "redshift_database": "dev",
          "redshift_user": "awsuser",
          "redshift_schema": "tpcds",
          "snapshot_date": "2003-01-02",
          "action": "load_sales_fact",
          "sql_statement": [
            "begin transaction;",
            "/* Create a stg_store_sales staging table */",
            "drop table if exists {0}.stg_store_sales;",
            "create table {0}.stg_store_sales",
            "(sold_date             date encode zstd",
            ",i_item_id             varchar(16) encode zstd",
            ",c_customer_id         varchar(16) encode zstd",
            ",ca_address_id         varchar(16) encode zstd",
```

```
            ",ss_ticket_number      integer encode zstd",
            ",ss_quantity           integer encode zstd",
            ",ss_net_paid           numeric(7,2) encode zstd",
            ",ss_net_profit         numeric(7,2) encode zstd",
            ")",
            "backup no",
            "diststyle even;",
            "/* Ingest data from source */",
            "insert into {0}.stg_store_sales",

"(sold_date,i_item_id,c_customer_id,ca_address_id,ss_ticket_number,ss_quantity,ss_net_paid,ss_
            "values",

"('2003-01-02','AAAAAAAAIFNAAAAA','AAAAAAAAOHNBAAAA','AAAAAAAAGBOFNIAA',1403191,13,5046.37,150

"('2003-01-02','AAAAAAAAIFNAAAAA','AAAAAAAAOHNBAAAA','AAAAAAAAGBOFNIAA',1403191,13,2103.72,-12

"('2003-01-02','AAAAAAAAILOBAAAA','AAAAAAAAOHNBAAAA','AAAAAAAAGBOFNIAA',1403191,13,959.10,-130

"('2003-01-02','AAAAAAAAILOBAAAA','AAAAAAAAHNAJAAAA','AAAAAAAAIAPCFNAA',1403191,13,962.65,-475

"('2003-01-02','AAAAAAAAMCOAAAAA','AAAAAAAAHNAJAAAA','AAAAAAAAIAPCFNAA',1201746,17,111.60,-241

"('2003-01-02','AAAAAAAAMCOAAAAA','AAAAAAAAHNAJAAAA','AAAAAAAAIAPCFNAA',1201746,17,4013.02,-11

"('2003-01-02','AAAAAAAAMCOAAAAA','AAAAAAAAMJCLACAA','AAAAAAAAMIFKEHAA',1201746,17,2689.12,-55

"('2003-01-02','AAAAAAAAMGOAAAAA','AAAAAAAAMJCLACAA','AAAAAAAAMIFKEHAA',193971,18,1876.89,-556
            "/* Delete any rows from target store_sales for the input date for
idempotency */",
            "delete from {0}.store_sales where ss_sold_date_sk in (select d_date_sk
from {0}.date_dim where d_date='{1}');",
            "/* Insert data from staging table to the target table */",
            "insert into {0}.store_sales",
            "(ss_sold_date_sk",
            ",ss_item_sk",
            ",ss_customer_sk",
            ",ss_addr_sk",
            ",ss_ticket_number",
            ",ss_quantity",
            ",ss_net_paid",
            ",ss_net_profit",
            ")",
            "select date_dim.d_date_sk ss_sold_date_sk",
```

```
            "      ,item.i_item_sk ss_item_sk",
            "      ,customer.c_customer_sk ss_customer_sk",
            "      ,customer_address.ca_address_sk ss_addr_sk",
            "      ,ss_ticket_number",
            "      ,ss_quantity",
            "      ,ss_net_paid",
            "      ,ss_net_profit",
            "  from {0}.stg_store_sales as store_sales",
            "  inner join {0}.date_dim on store_sales.sold_date = date_dim.d_date",
            "  left join {0}.item on store_sales.i_item_id = item.i_item_id and
 item.i_rec_end_date is null",
            "  left join {0}.customer on store_sales.c_customer_id =
 customer.c_customer_id",
            "  left join {0}.customer_address on store_sales.ca_address_id =
 customer_address.ca_address_id;",
            "/* Drop staging table */",
            "drop table if exists {0}.stg_store_sales;",
            "/* Commit and End transaction */",
            "commit;",
            "end transaction;"
          ]
        }
      }
    },
    "ExecuteSalesFactLoadJob": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "GetSalesFactLoadStatus",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "GetSalesFactLoadStatus": {
      "Type": "Task",
      "Next": "CheckSalesFactLoadStatus",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "InputPath": "$",
      "ResultPath": "$.status"
    },
```

```
    "CheckSalesFactLoadStatus": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.status",
          "StringEquals": "FAILED",
          "Next": "FailSalesFactLoad"
        },
        {
          "Variable": "$.status",
          "StringEquals": "FINISHED",
          "Next": "SalesETLPipelineComplete"
        }
      ],
      "Default": "SalesWait"
    },
    "SalesWait": {
      "Type": "Wait",
      "Seconds": 5,
      "Next": "GetSalesFactLoadStatus"
    },
    "FailSalesFactLoad": {
      "Type": "Fail",
      "Cause": "ETL Workflow Failed",
      "Error": "Error"
    },
    "ClusterUnavailable": {
      "Type": "Fail",
      "Cause": "Redshift cluster is not available",
      "Error": "Error"
    },
    "SalesETLPipelineComplete": {
      "Type": "Pass",
      "Next": "ValidateSalesMetric",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "redshift_database": "dev",
          "redshift_user": "awsuser",
          "redshift_schema": "tpcds",
          "snapshot_date": "2003-01-02",
          "action": "validate_sales_metric",
          "sql_statement": [
```

```
            "select 1/count(1) from {0}.store_sales where ss_sold_date_sk in (select
 d_date_sk from {0}.date_dim where d_date='{1}')"
          ]
        }
      }
    },
    "ValidateSalesMetric": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "GetValidateSalesMetricStatus",
      "InputPath": "$",
      "ResultPath": "$"
    },
    "GetValidateSalesMetricStatus": {
      "Type": "Task",
      "Next": "CheckValidateSalesMetricStatus",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftDataApi-AIDACKCEVSQ6C2EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "InputPath": "$",
      "ResultPath": "$.status"
    },
    "CheckValidateSalesMetricStatus": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.status",
          "StringEquals": "FAILED",
          "Next": "FailSalesMetricValidation"
        },
        {
          "Variable": "$.status",
          "StringEquals": "FINISHED",
          "Next": "DataValidationComplete"
        }
      ],
      "Default": "SalesValidationWait"
    },
    "SalesValidationWait": {
      "Type": "Wait",
```

```
      "Seconds": 5,
      "Next": "GetValidateSalesMetricStatus"
    },
    "FailSalesMetricValidation": {
      "Type": "Fail",
      "Cause": "Data Validation Failed",
      "Error": "Error"
    },
    "DataValidationComplete": {
      "Type": "Pass",
      "Next": "InitializePauseCluster"
    },
    "InitializePauseCluster": {
      "Type": "Pass",
      "Next": "PauseCluster",
      "Result": {
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "operation": "pause"
        }
      }
    },
    "PauseCluster": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "PauseClusterWait",
      "InputPath": "$",
      "ResultPath": "$.clusterStatus",
      "Catch": [
        {
          "ErrorEquals": [
            "States.ALL"
          ],
          "Next": "ClusterPausedComplete"
        }
      ]
    },
    "InitializeCheckPauseCluster": {
      "Type": "Pass",
      "Next": "GetStateOfPausedCluster",
      "Result": {
```

```
        "input": {
          "redshift_cluster_id": "cfn36-redshiftcluster-AKIAI44QH8DHBEXAMPLE",
          "operation": "status"
        }
      }
    },
    "GetStateOfPausedCluster": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE",
      "TimeoutSeconds": 180,
      "HeartbeatSeconds": 60,
      "Next": "IsClusterPaused",
      "InputPath": "$",
      "ResultPath": "$.clusterStatus"
    },
    "IsClusterPaused": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "available",
          "Next": "InitializePauseCluster"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "paused",
          "Next": "ClusterPausedComplete"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "unavailable",
          "Next": "ClusterUnavailable"
        },
        {
          "Variable": "$.clusterStatus",
          "StringEquals": "resuming",
          "Next": "PauseClusterWait"
        }
      ]
    },
    "PauseClusterWait": {
      "Type": "Wait",
      "Seconds": 720,
```

```
      "Next": "InitializeCheckPauseCluster"
    },
    "ClusterPausedComplete": {
      "Type": "Pass",
      "End": true
    }
  }
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-east-1:111122223333:function:CFN36-RedshiftDataApi-
AIDACKCEVSQ6C2EXAMPLE",
                "arn:aws:lambda:us-east-1:111122223333:function:CFN36-
RedshiftOperations-AKIAIOSFODNN7EXAMPLE"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Step Functions 및 오류 처리 기능이 있는 AWS Batch 사용

이 샘플 프로젝트에서는 Step Functions를 사용하여 오류 처리 기능이 있는 상태 시스템을 사용하는 AWS Batch 작업을 실행하는 방법을 보여줍니다.

이 프로젝트에서 Step Functions은 상태 시스템을 사용하여 AWS Batch 작업을 동기식으로 호출합니다. 그런 다음 작업이 성공 또는 실패하기까지 기다리고 작업이 실패하면 오류를 재시도 및 포착한 다음 작업 성공 또는 실패 여부에 대한 메시지와 함께 Amazon SNS 주제를 전송합니다.

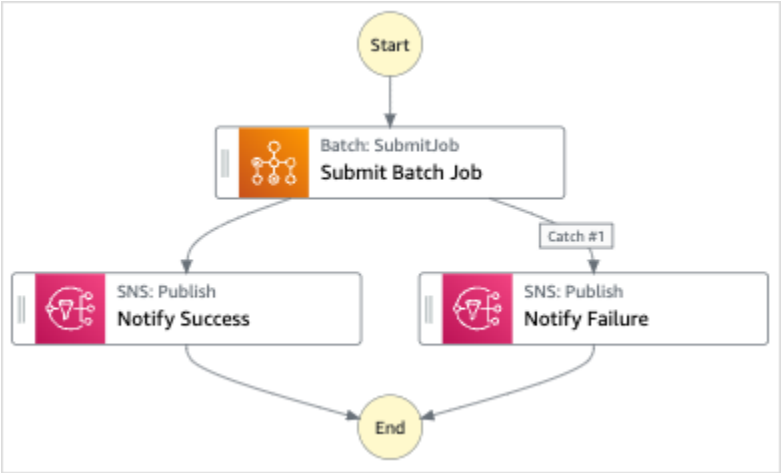
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Manage a batch job`**을 입력한 다음 반환된 검색 결과에서 배치 작업 관리를 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - AWS Batch 직업
   - Amazon SNS 주제
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 배치 작업 관리 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

     > **Important**
     >
     > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

   - 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

     > **Tip**
     >
     > 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

     준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

   1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

      > **Note**
      >
      > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

   2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

      데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

      > **Note**
      >
      > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

   3. 실행 시작을 선택합니다.

4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 파라미터를 해당 리소스에 직접 전달하여 Amazon AWS Batch SNS와 통합됩니다.

이 예제 상태 머신을 살펴보고 `Resource` 필드의 Amazon 리소스 이름 (ARN) 에 연결하고 서비스 API에 `Parameters` 전달하여 Step Functions가 Amazon SNS를 제어하는 AWS Batch 방법을 살펴봅니다.

다른 AWS 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
  "Comment": "An example of the Amazon States Language for notification on an AWS Batch
 job completion",
  "StartAt": "Submit Batch Job",
  "TimeoutSeconds": 3600,
  "States": {
    "Submit Batch Job": {
      "Type": "Task",
      "Resource": "arn:aws:states:::batch:submitJob.sync",
      "Parameters": {
        "JobName": "BatchJobNotification",
        "JobQueue": "arn:aws:batch:us-west-2:123456789012:job-queue/
BatchJobQueue-123456789abcdef",
        "JobDefinition": "arn:aws:batch:us-west-2:123456789012:job-definition/
BatchJobDefinition-123456789abcdef:1"
      },
      "Next": "Notify Success",
      "Retry": [
        {
          "ErrorEquals": [
```

```
            "States.ALL"
          ],
          "IntervalSeconds": 30,
          "MaxAttempts": 2,
          "BackoffRate": 1.5
        }
      ],
      "Catch": [
        {
          "ErrorEquals": [ "States.ALL" ],
          "Next": "Notify Failure"
        }
      ]
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "Batch job submitted through Step Functions succeeded",
        "TopicArn": "arn:aws:sns:us-west-2:123456789012:StepFunctionsSample-
BatchJobManagement12345678-9abc-def0-1234-567890abcdef-SNSTopic-A2B3C4D5E6F7G"
      },
      "End": true
    },
    "Notify Failure": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "Message": "Batch job submitted through Step Functions failed",
        "TopicArn": "arn:aws:sns:us-west-2:123456789012:StepFunctionsSample-
BatchJobManagement12345678-9abc-def0-1234-567890abcdef-SNSTopic-A2B3C4D5E6F7G"
      },
      "End": true
    }
  }
}
```

IAM 예제

샘플 프로젝트에서 생성된 이 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

Example **BatchJobNotificationAccessPolicy**

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-west-2:123456789012:StepFunctionsSample-
BatchJobManagement12345678-9abc-def0-1234-567890abcdef-SNSTopic-A2B3C4D5E6F7G"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "batch:SubmitJob",
                "batch:DescribeJobs",
                "batch:TerminateJob"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:us-west-2:123456789012:rule/
StepFunctionsGetEventsForBatchJobsRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

AWS Batch 작업 팬 아웃

이 샘플 프로젝트는 Step Functions의 맵 상태를 사용하여 AWS Batch 작업을 팬아웃하는 방법을 보여줍니다.

이 프로젝트에서 Step Functions는 상태 머신을 사용하여 Lambda 함수를 호출하여 간단한 사전 처리를 수행한 다음 상태를 사용하여 여러 AWS Batch 작업을 병렬로 호출합니다. 맵

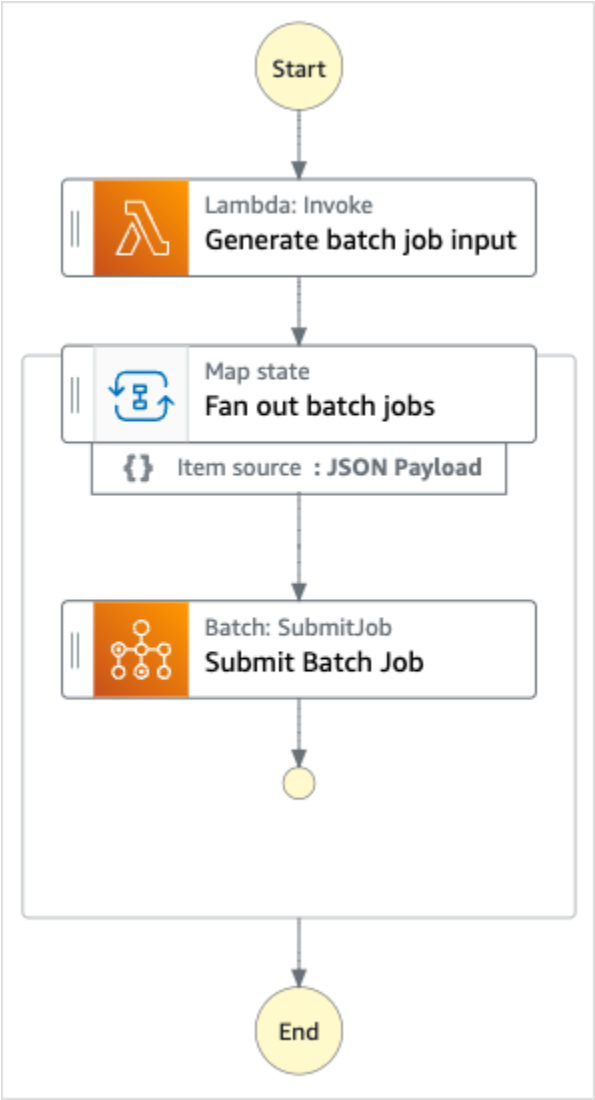
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Fan out a batch job`**을 입력한 다음 반환된 검색 결과에서 배치 작업 팬아웃을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Lambda 기능
   - AWS Batch 작업 대기열
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지에서는 배치 작업 팬아웃 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

> **Note**
>
> Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

   > **Note**
   >
   > 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 파라미터를 해당 리소스에 직접 전달하여 Amazon AWS Batch SNS와 통합됩니다.

이 예제 상태 머신을 살펴보고 `Resource` 필드의 Amazon 리소스 이름 (ARN) 에 연결하고 서비스 API에 `Parameters` 전달하여 Step Functions가 Amazon SNS를 제어하는 AWS Batch 방법을 살펴봅니다.

다른 AWS 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
  "Comment": "An example of the Amazon States Language for fanning out AWS Batch job",
  "StartAt": "Generate batch job input",
  "TimeoutSeconds": 3600,
  "States": {
    "Generate batch job input": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.Payload",
      "Parameters": {
        "FunctionName": "<GENERATE_BATCH_JOB_INPUT_LAMBDA_FUNCTION_NAME>"
      },
      "Next": "Fan out batch jobs"
    },
    "Fan out batch jobs": {
      "Comment": "Start multiple executions of batch job depending on pre-processed
 data",
      "Type": "Map",
      "End": true,
      "ItemsPath": "$",
      "Parameters": {
        "BatchNumber.$": "$$.Map.Item.Value"
      },
      "Iterator": {
        "StartAt": "Submit Batch Job",
        "States": {
          "Submit Batch Job": {
            "Type": "Task",
            "Resource": "arn:aws:states:::batch:submitJob.sync",
            "Parameters": {
              "JobName": "BatchJobFanOut",
              "JobQueue": "<BATCH_QUEUE_ARN>",
              "JobDefinition": "<BATCH_JOB_DEFINITION_ARN>"
            },
            "End": true
          }
        }
      }
    }
  }
}
```

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

Example **BatchJobFanOutAccessPolicy**

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "batch:SubmitJob",
                "batch:DescribeJobs",
                "batch:TerminateJob"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:us-west-2:123456789012:rule/
StepFunctionsGetEventsForBatchJobsRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Example **InvokeGenerateBatchJobMapLambdaPolicy**

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
```

```
            ],
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:StepFunctionsSample-BatchJobFa-
GenerateBatchJobMap-444455556666",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

AWS Batch 람다와 함께

이 샘플 프로젝트는 Step Functions를 사용하여 함수를 사용하여 데이터를 사전 처리한 다음 작업을 AWS Batch 오케스트레이션하는 방법을 보여줍니다. AWS Lambda

이 프로젝트에서 Step Functions는 상태 시스템을 사용하여 Lambda 함수를 간접적으로 호출해 AWS Batch 작업이 제출되기 전에 간단한 사전 처리를 수행합니다. 이전 작업의 결과 또는 성공 여부에 따라 여러 작업을 간접적으로 호출할 수 있습니다.

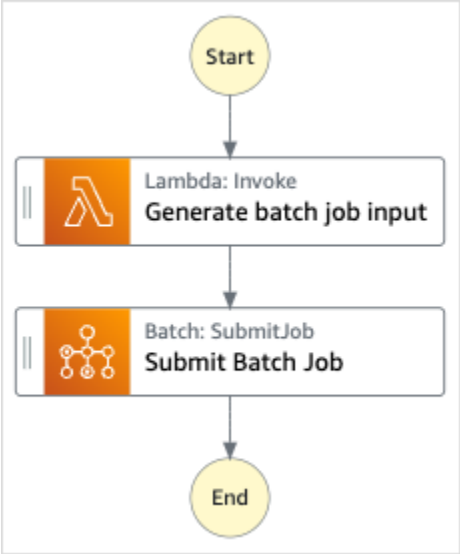
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **`Batch job with Lambda`**를 입력한 다음 반환된 검색 결과에서 Lambda를 사용한 Batch 작업을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - Lambda 기능
   - AWS Batch 작업
   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

다음 이미지에서는 Lambda를 사용한 Batch 작업 샘플 프로젝트의 워크플로 그래프를 보여줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

- 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

  Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

  > **Important**
  >
  > 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 고유한 실행 이름을 자동으로 생성합니다.

        > **Note**
        >
        > Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

    2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

        데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.

> **Note**
>
> 배포한 데모 프로젝트에 미리 채워진 실행 입력 데이터가 포함되어 있는 경우 해당 입력을 사용하여 상태 시스템을 실행하세요.

3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

예제 상태 머신 코드

이 샘플 프로젝트의 상태 머신은 파라미터를 해당 리소스에 직접 전달하여 Amazon AWS Batch SNS와 통합됩니다.

이 예제 상태 머신을 살펴보고 `Resource` 필드의 Amazon 리소스 이름 (ARN) 에 연결하고 서비스 API에 `Parameters` 전달하여 Step Functions가 Amazon SNS를 제어하는 AWS Batch 방법을 살펴봅니다.

다른 AWS 서비스를 제어하는 방법에 AWS Step Functions 대한 자세한 내용은 을 참조하십시오다른 서비스와 AWS Step Functions 함께 사용.

```
{
  "Comment": "An example of the Amazon States Language for using batch job with pre-
processing lambda",
  "StartAt": "Generate batch job input",
  "TimeoutSeconds": 3600,
  "States": {
    "Generate batch job input": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.batch_input",
      "Parameters": {
        "FunctionName": "<GENERATE_BATCH_JOB_INPUT_LAMBDA_FUNCTION_NAME>"
```

```
      },
      "Next": "Submit Batch Job"
    },
    "Submit Batch Job": {
      "Type": "Task",
      "Resource": "arn:aws:states:::batch:submitJob.sync",
      "Parameters": {
        "JobName": "BatchJobFanOut",
        "JobQueue": "<BATCH_QUEUE_ARN>",
        "JobDefinition": "<BATCH_JOB_DEFINITION_ARN>",
        "Parameters.$": "$.batch_input"
      },
      "End": true
    }
  }
}
```

IAM 예제

샘플 프로젝트에서 생성된 이러한 예제 AWS Identity and Access Management (IAM) 정책에는 스테이트 머신 및 관련 리소스를 실행하는 데 필요한 최소 권한이 포함되어 있습니다. IAM 정책에 필요한 정책만 포함시키는 것이 좋습니다.

Example **`BatchJobWithLambdaAccessPolicy`**

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-west-2:123456789012:ManageBatchJob-SNSTopic-
JHLYYG7AZPZI"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "batch:SubmitJob",
                "batch:DescribeJobs",
                "batch:TerminateJob"
```

```
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:us-west-2:123456789012:rule/
StepFunctionsGetEventsForBatchJobsRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Example **InvokeGenerateBatchJobMapLambdaPolicy**

```
{
    "Statement": [
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:StepFunctionsSample-BatchWithL-
GenerateBatchJobMap-444455556666",
            "Effect": "Allow"
        }
    ]
}
```

Step Functions를 다른 AWS 서비스와 함께 사용할 때 IAM을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오통합 서비스용 IAM 정책.

Amazon Bedrock을 사용하여 AI 프롬프트 체이닝 수행

이 샘플 프로젝트는 Amazon Bedrock으로 통합하여 AI 프롬프트 체이닝을 수행하는 방법을 보여 줍니다. 이 샘플 프로젝트는 Amazon Bedrock을 사용하여 고품질 챗봇을 구축하는 방법을 보여 줍니다. 프

로젝트는 몇 가지 프롬프트를 연결하여 제공된 순서대로 해결합니다. 이러한 프롬프트를 연결하면 고도로 엄선된 응답을 제공하는 데 사용되는 언어 모델의 기능이 향상됩니다.

이 샘플 프로젝트는 스테이트 머신과 지원 AWS 리소스를 생성하고 관련 IAM 권한을 구성합니다. 이 샘플 프로젝트를 살펴보고 Step Functions 상태 머신으로 Amazon Bedrock 최적화 서비스 통합을 사용하는 방법에 대해 알아보거나 이를 자체 프로젝트의 시작점으로 사용합니다.

주제

AWS CloudFormation 템플릿과 추가 리소스

CloudFormation 템플릿을 사용하여 이 샘플 프로젝트를 배포합니다. 이 템플릿은 다음과 같은 리소스를 사용자 사이트에 생성합니다. AWS 계정

- Step Functions 상태 시스템
- 상태 시스템에 대한 실행 역할. 이 역할은 상태 머신이 다른 AWS 서비스 리소스 (예: Amazon Bedrock InvokeModel작업) 에 액세스하는 데 필요한 권한을 부여합니다.

필수 조건

이 샘플 프로젝트는 Cohere Command 대규모 언어 모델(LLM)을 사용합니다. 이 샘플 프로젝트를 성공적으로 실행하려면 Amazon Bedrock 콘솔에서 이 LLM에 대한 액세스 권한을 추가해야 합니다. 모델 액세스 권한을 추가하려면 다음 내용을 따릅니다.

1. Amazon Bedrock 콘솔을 엽니다.
2. 기본 탐색 창에서 모델 액세스를 선택합니다.
3. 모델 액세스 관리를 선택합니다.
4. Cohere 옆의 확인란을 선택합니다.
5. 액세스 요청을 선택합니다. Cohere 모델의 액세스 상태는 액세스 권한 부여됨으로 표시됩니다.

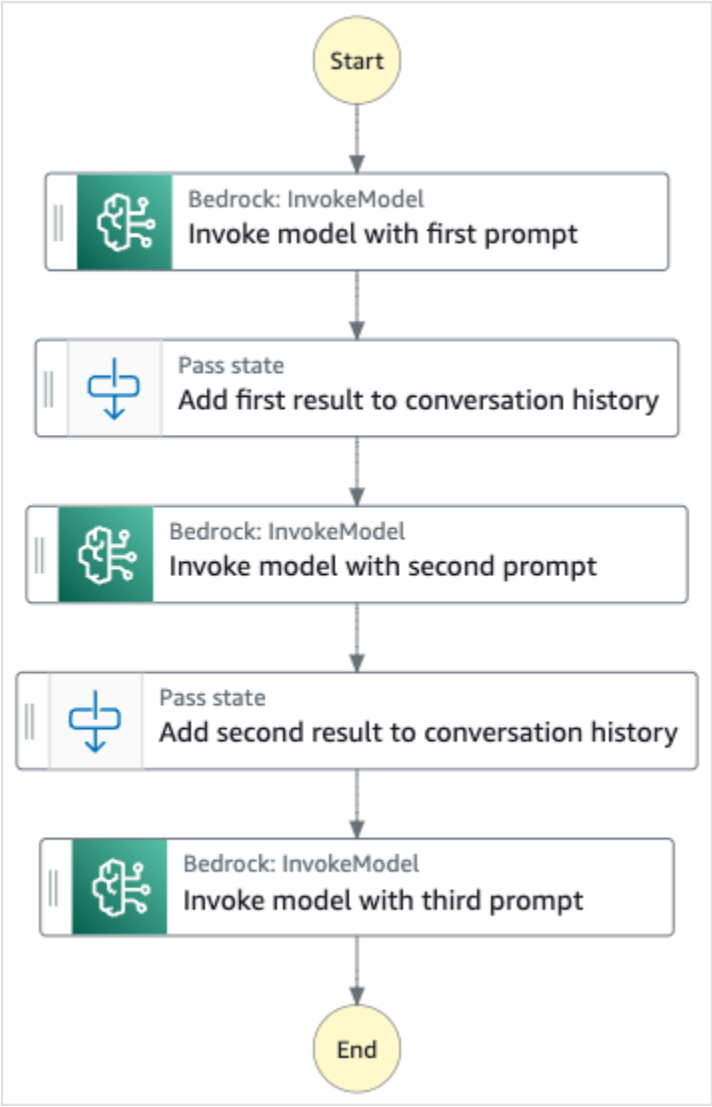
1단계: 상태 시스템 만들기 및 리소스 프로비저닝

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 검색 상자에 **bedrock**을 입력한 다음 반환된 검색 결과에서 Bedrock으로 AI 프롬프트 체이닝 수행을 선택합니다.
3. 다음을 선택하여 계속 진행합니다.
4. Step Functions는 선택한 샘플 프로젝트에 AWS 서비스 사용된 항목을 나열합니다. 또한 샘플 프로젝트의 워크플로 그래프도 보여줍니다. 이 프로젝트를 사용자 프로젝트에 AWS 계정 배포하거나 자체 프로젝트를 빌드하기 위한 출발점으로 사용하세요. 진행하려는 방식에 따라 데모 실행 또는 이를 기반으로 구축을 선택합니다.

   이 샘플 프로젝트는 다음 리소스를 배포합니다.

   - AWS Step Functions 스테이트 머신
   - 관련 AWS Identity and Access Management (IAM) 역할

   다음 이미지는 Bedrock으로 AI 프롬프트 체이닝 수행 샘플 프로젝트의 워크플로 그래프를 보여 줍니다.



5. 템플릿 사용을 선택하여 계속 선택합니다.
6. 다음 중 하나를 수행하십시오.

   - 이를 기반으로 구축을 선택한 경우 Step Functions는 선택한 샘플 프로젝트의 워크플로 프로토타입을 만듭니다. Step Functions는 워크플로 정의에 나열된 리소스를 배포하지 않습니다.

     Workflow Studio의 디자인 모드에서 상태 브라우저의 상태를 끌어서 놓아 워크플로 프로토타입을 계속 빌드합니다. 또는 VS Code와 유사한 통합 코드 편집기를 제공하는 코드 모드로 전환하여 Step Functions 콘솔 내에서 상태 시스템의 Amazon States Language(ASL) 정의를 업데이트합니다. Workflow Studio를 사용하여 상태 시스템 빌드에 대한 자세한 내용은 Workflow Studio 사용 섹션을 참조하세요.

> **Important**
>
> 워크플로를 실행하기 전에 샘플 프로젝트에 사용된 리소스의 자리 표시자 Amazon 리소스 이름(ARN)을 업데이트해야 합니다.

- 데모 실행을 선택한 경우 Step Functions는 템플릿을 사용하여 해당 AWS CloudFormation 템플릿에 나열된 AWS 리소스를 사용자에게 배포하는 읽기 전용 샘플 프로젝트를 만듭니다. AWS 계정

> **Tip**
>
> 샘플 프로젝트의 상태 시스템 정의를 보려면 코드를 선택하세요.

준비가 되면 배포 및 실행을 선택하여 샘플 프로젝트를 배포하고 리소스를 만듭니다.

이러한 리소스 및 관련 IAM 권한을 만드는 데 최대 10분이 걸릴 수 있습니다. 리소스를 배포하는 동안 CloudFormation Stack ID 링크를 열어 프로비저닝되는 리소스를 확인할 수 있습니다.

샘플 프로젝트의 모든 리소스가 생성된 후에 상태 시스템 페이지에 새 샘플 프로젝트가 나열됩니다.

> **Important**
>
> 템플릿에서 사용되는 각 서비스에는 표준 요금이 적용될 수 있습니다. CloudFormation

2단계: 상태 시스템 실행

1. 상태 시스템 페이지에서 샘플 프로젝트를 선택합니다.
2. 샘플 프로젝트 페이지에서 실행 시작을 선택합니다.
3. 실행 시작 대화 상자에서 다음을 수행합니다.

    1. (선택 사항) 실행을 식별하려면 이름 상자에 해당 실행의 이름을 지정하면 됩니다. 기본적으로 Step Functions는 자동으로 고유한 실행 이름을 생성합니다.

> **Note**
>
> Step Functions를 사용하면 비 ASCII 문자가 포함된 상태 시스템, 실행, 활동 및 레이블 이름을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

2. (선택 사항) 입력 상자에 입력 값을 JSON 형식으로 입력하여 워크플로를 실행합니다.

   데모 실행을 선택한 경우에는 실행 입력을 제공할 필요가 없습니다.
3. 실행 시작을 선택합니다.
4. Step Functions 콘솔은 실행 ID가 제목인 페이지로 이동합니다. 이 페이지를 실행 세부 정보 페이지라고 합니다. 실행이 진행되는 동안 또는 완료된 후에 이 페이지에서 실행 결과를 검토할 수 있습니다.

   실행 결과를 검토하려면 그래프 보기에서 개별 상태를 선택한 다음 단계 세부 정보 창에서 개별 탭을 선택하여 입력, 출력 및 정의가 포함된 각 상태의 세부 정보를 각각 봅니다. 실행 세부 정보 페이지에서 볼 수 있는 실행 정보에 대한 자세한 내용은 실행 세부 정보 페이지 - 인터페이스 개요 섹션을 참조하세요.

할당량

AWS Step Functions 특정 기간 동안의 API 작업 수 또는 정의할 수 있는 상태 머신의 수와 같은 특정 상태 머신 파라미터의 크기에 할당량을 할당합니다. 잘못 구성된 상태 머신이 시스템의 리소스를 모두 사용하지 않도록 하기 위해 이러한 할당량이 설계된 것이지만 많은 수가 하드 할당량은 아닙니다.

서비스 할당량 증가를 요청하려면 다음 중 하나를 수행하면 됩니다.

- Service Quotas 콘솔(https://console.aws.amazon.com/servicequotas/home)을 엽니다. Service Quotas 콘솔을 사용하여 할당량 증가를 요청하는 방법은 Service Quotas 사용 설명서의 할당량 증가 요청을 참조하세요.
- 의 Support Center 페이지를 사용하여 지역별로 AWS Step Functions 제공하는 리소스의 할당량 증가를 요청하십시오. AWS Management Console 자세한 내용은AWS 일반 참조의 AWS Service Quotas를 참조하십시오.

> **Note**
>
> 상태 머신 실행이나 활동 실행의 특정 단계가 너무 오래 걸리는 경우 상태 머신 시간 제한을 구성하여 시간 초과 이벤트를 발생시킬 수 있습니다.

주제

일반 할당량

할당량	설명
Step Functions에서 이름	상태 머신, 실행 및 활동 태스크의 이름은 80자를 초과하면 안 됩니다. 이러한 이름은 계정 및 AWS 지역별로 고유해야 하며 다음 내용을 포함하지 않아야 합니다. • 공백 • 와일드카드 문자 (? *) • 괄호 문자(< > { } [ ]) • 특수 문자 (" # % \ ^ | ~ ` $ & , ; : /) • 제어 문자(\\u0000 - \\u001f 또는 \\u007f - \\u009f). 상태 시스템이 Express 유형이면 상태 시스템 실행 여러 개에 같은 이름을 제공할 수 있습니다. Step Functions는 여러 실행 이름이 같더라도 Express 상태 시스템 실행마다 고유한 실행 ARN을 생성합니다. Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.

계정과 관련된 할당량

Resource	기본 할당량	증가 가능
등록된 상태 머신 최대 수	10,000개	25,000
등록된 활동 최대 수	10,000개	15,000
최대 요청 크기	요청당 1MB. Step Functions API 요청당 전체 데이터 크기로, 요청 헤더와 다른 모든 연결된 요청 데이터가 포함됩니다.	하드 할당량
계정당 최대 열린 실행 수	각 AWS 리전에서 AWS 계정 마다 1,000,000회 실행. 이를 초과하면 `Execution LimitExceeded` 오류가 발생합니다. Express 워크플로에는 적용되지 않습니다.	수백만
최대 오픈 맵 실행 수 오픈 맵 실행은 시작되었지만 아직 완료되지 않은 맵 실행입니다. 예정된 맵 런은 열린 맵 런의 총 수가 기본 할당량인 1000개 미만이 될 때까지 MapRunStarted이벤트 기간 동안 대기합니다.	1000 이 할당량은 Distributed Map 상태에 적용됩니다.	하드 할당량
최대 맵 실행 redrives 수	1000 이 할당량은 Distributed Map 상태에 적용됩니다.	하드 할당량
병렬 Map Run 하위 실행 최대 수	10,000개	하드 할당량

HTTP 태스크 관련 할당량

Step Functions 서비스 대역폭을 유지하기 위해 토큰 버킷 체계를 사용하여 HTTP 태스크가 제한됩니다.

Resource	버킷 크기	초당 다시 채우기 속도
HTTP 태스크	300	300

다음 테이블에는 HTTP 태스크 기간 동안의 할당량이 나열되어 있습니다.

Resource	기본 할당량
HTTP 태스크 기간 HTTP 태스크 기간은 HTTP 태스크가 HTTP 요청을 보내고 응답을 받는 데 걸리는 시간을 말합니다.	60초 이 수는 하드 할당량이며 변경할 수 없습니다.

상태 제한과 관련된 할당량

서비스 대역폭을 유지하기 위해 토큰 버킷 체계를 사용하여 Step Functions 상태 변환을 제한합니다. 표준 워크플로와 Express 워크플로의 상태 전환 제한은 서로 다릅니다. 표준 워크플로 할당량은 소프트 할당량이며 증가될 수 있습니다.

> **Note**
>
> `StateTransition`서비스 지표에 대한 스로틀링은 `ExecutionThrottled` Amazon에서와 같이 보고됩니다. CloudWatch 자세한 내용은 지표를 참조하십시오. `ExecutionThrottled` CloudWatch

	Standard		Express	
서비스 지표	버킷 크기	초당 다시 채우기 속도	버킷 크기	초당 다시 채우기 속도
`StateTransition` - 미국 동부(버지니아 북부), 미국 서부(오레곤) 및 유럽(아일랜드)	5,000	5,000	무제한	무제한
`StateTransition` - 다른 모든 리전	800	800	무제한	무제한

API 작업 제한과 관련된 할당량

서비스 대역폭을 유지하기 위해 토큰 버킷을 사용하여 일부 Step Functions API 작업이 제한됩니다. 이러한 할당량은 소프트 할당량이며 증가될 수 있습니다.

> **Note**
>
> 제한 할당량은 계정별, 지역별입니다. AWS
> AWS Step Functions 언제든지 버킷 크기와 리필 비율을 모두 늘릴 수 있습니다.

	Standard		Express	
API 이름	버킷 크기	초당 다시 채우기 속도	버킷 크기	초당 다시 채우기 속도
`StartExecution` - 미국 동부(버지니아 북부), 미국 서부(오	1,300	300	6,000	6,000

	Standard		Express	
API 이름	버킷 크기	초당 다시 채우기 속도	버킷 크기	초당 다시 채우기 속도
레곤) 및 유럽(아일랜드)				
`StartExecution` - 다른 모든 리전	800	150	6,000	6,000

API 관련 TestState 할당량

API 이름	할당량	증가 가능
TestState	1개의 초당 트랜잭션(TPS)	하드 할당량

기타 할당량

이러한 할당량은 소프트 할당량이며 증가될 수 있습니다.

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 이름	버킷 크기	초당 다시 채우기 속도	버킷 크기	초당 다시 채우기 속도
`CreateActivity`	100	1	100	1
`CreateStateMachine`	100	1	100	1
`DeleteActivity`	100	1	100	1

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 이름	버킷 크기	초당 다시 채우기 속도	버킷 크기	초당 다시 채우기 속도
DeleteStateMachine	100	1	100	1
DescribeActivity	200	1	200	1
DescribeExecution	300	15	250	10
DescribeStateMachine	200	20	200	20
DescribeStateMachineForExecution	200	1	200	1
GetActivityTask	3,000	500	1,500	300
GetExecutionHistory	400	20	400	20
ListActivities	100	10	100	5
ListExecutions	200	5	100	2
ListStateMachines	100	5	100	5

	In US East (N. Virginia), US West (Oregon), and Europe (Ireland)		All other regions	
API 이름	버킷 크기	초당 다시 채우기 속도	버킷 크기	초당 다시 채우기 속도
`ListTagsForResource`	100	1	100	1
`SendTaskFailure`	3,000	500	1,500	300
`SendTaskHeartbeat`	3,000	500	1,500	300
`SendTaskSuccess`	3,000	500	1,500	300
StartSyncExecution	동기 Express 실행 API 직접 호출은 기존 계정 용량 한도에 영향을 주지 않습니다. Step Functions는 온디맨드 용량을 제공하고 지속적인 워크로드에 따라 자동으로 규모를 조정합니다. 용량이 확보될 때까지 워크로드 급증이 제한될 수 있습니다. 제한이 발생하면 잠시 후 다시 시도하세요. 동기 Express 워크플로에 대한 자세한 내용은 동기 및 비동기 Express 워크플로 섹션을 참조하세요.			
`StopExecution`	1,000	200	500	25
`TagResource`	200	1	200	1
`UntagResource`	200	1	200	1
`UpdateStateMachine`	100	1	100	1

상태 시스템 실행과 관련된 할당량

다음 표에서는 상태 시스템 실행과 관련된 할당량을 설명합니다. 상태 시스템 실행 할당량은 실행 내역 보존 시간 할당량을 제외하고 변경될 수 없는 하드 할당량입니다.

할당량	표준	Express
최대 실행 시간	1년. 최대 1년 이상 실행되면 States.Timeout 오류와 함께 실행이 실패하고 지표가 생성됩니다. ExecutionsTimedOut CloudWatch	5분 최대 5분 이상 실행되면 States.Timeout 오류와 함께 실행이 실패하고 지표가 생성됩니다. ExecutionsTimedOut CloudWatch
최대 실행 기록 크기	단일 상태 시스템 실행 내역에 있는 이벤트 25,000개입니다. 실행 내역이 이 할당량에 도달할 경우 실행이 실패합니다. 이를 피하려면 내역 할당량 도달 방지 단원을 참조하십시오.	무제한.
최대 실행 유휴 시간	1년(최대 실행 시간에 따라 제한됨)	5분(최대 실행 시간에 따라 제한됨)
실행 내역 보존 기간	실행이 종료된 지 90일 후. 이 기간이 지나면 실행 내역을 더 이상 검색하거나 볼 수 없습니다. Step Functions에서 유지하는 닫힌 실행 수에 대한 추가 할당량은 없습니다. 규정 준수, 조직 또는 규제 기관 요구 사항을 충족하려면 할당량 요청을 보내 실행 내역 보존 기간을 30일로 줄이면 됩니다. 이렇게 하려면 를 사용하고 새 케이스를 생성하십시오. AWS Support Center Console	실행 기록을 보려면 Amazon CloudWatch Logs 로깅을 구성해야 합니다. 자세한 정보는 CloudWatch Logs를 사용하여 로깅을 참조하세요.

할당량	표준	Express
	보존 기간을 30일로 줄이는 변경 사항은 리전의 계정 수준에서 적용됩니다.	
실행 redrivable 기간 Redrivable 기간은 지정된 표준 워크플로 실행을 redrive할 수 있는 시간을 의미합니다. 이 기간은 상태 시스템에서 실행을 완료한 날부터 시작합니다.	14일. 이 하드 할당량은 Distributed Map 상태에 적용됩니다.	현재 Redrive는 Express 워크플로에는 지원되지 않습니다.

작업 실행과 관련된 할당량

다음 표에는 작업 실행과 관련된 할당량이 설명되어 있습니다. 이는 모두 변경할 수 없는 고정 할당량입니다.

할당량	표준	Express
최대 작업 실행 시간	1년(최대 실행 시간에 따라 제한됨)	5분(최대 실행 시간에 따라 제한됨)
Step Functions가 대기열에 작업을 유지하는 최대 시간	1년(최대 실행 시간에 따라 제한됨)	5분(최대 실행 시간에 따라 제한됨)
Amazon 리소스 이름(ARN)당 최대 활동 폴러 수	ARN당 `GetActivityTask` 를 호출하는 폴러 1,000개. 이 할당량을 초과하면 "활동 작업에 대해 동시에 폴링 중인 작업자 최대 수에 도달했습니다" 오류가 나타납니다.	Express 워크플로에는 적용되지 않습니다.
작업, 상태 또는 실행에 대한 최대 입력 또는 결과	UTF-8로 인코딩된 문자열로 데이터 256KB. 이 할당량은 작업을 예약하거나 상태를 입력하거나 실행을 시작할 때 작업(활	UTF-8로 인코딩된 문자열로 데이터 256KB. 이 할당량은 작업을 예약하거나 상태를 입력하거나 실행을 시작할 때 작업(활

할당량	표준	Express
	동, Lambda 함수 또는 통합 서비스), 상태 또는 실행 출력 및 입력 데이터에 영향을 미칩니다.	동, Lambda 함수 또는 통합 서비스), 상태 또는 실행 출력 및 입력 데이터에 영향을 미칩니다.

버전 및 별칭과 관련된 할당량

Resource	기본 할당량
게시된 상태 시스템 버전 최대 수	상태 시스템당 1,000개입니다. 이 소프트 한도 증가를 요청하려면 AWS Management Console의 지원 센터 페이지를 사용합니다.
최대 상태 시스템 별칭 수	상태 시스템당 100개입니다. 이 소프트 한도 증가를 요청하려면 AWS Management Console의 지원 센터 페이지를 사용합니다.

태그 지정과 관련된 제한

Step Functions 리소스에 태그를 지정할 때는 이러한 제한에 주의하세요.

> **Note**
>
> 다른 할당량과 마찬가지로 태그 지정 제한을 증가시킬 수 없습니다.

제한	설명
리소스당 최대 태그 수	50

제한	설명
최대 키 길이	UTF-8 형식의 유니코드 문자 128자
최대 값 길이	UTF-8 형식의 유니코드 문자 256자
접두사 제한	`aws:`접두사는 사용하도록 예약되어 있으므로 태그 이름이나 값에 사용하지 마십시오. AWS 이 접두사가 지정된 태그 이름이나 값은 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 할당량에 포함되지 않습니다.
문자 제한	태그에는 유니코드 문자, 숫자, 공백 또는 `_ . : / = + - @` 등의 기호만 사용할 수 있습니다.

로그인 및 모니터링 AWS Step Functions

로깅 및 모니터링은 Step Functions와 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요합니다. Step Functions에서 여러 가지 도구를 사용할 수 있습니다.

> **Tip**
>
> 샘플 워크플로를 AWS 계정 배포하고 워크플로 실행의 메트릭, 로그 및 트레이스를 모니터링하는 방법을 알아보려면 모듈 12 - AWS Step Functions 워크숍 관찰 가능성을 참조하십시오.

주제

- Step Functions를 사용하여 모니터링하기 CloudWatch
- EventBridge Step Functions의 실행 상태 변경에 대한 (CloudWatch 이벤트)
- 를 사용하여 API 호출 녹음하기 AWS CloudTrail
- CloudWatch Logs를 사용하여 로깅
- AWS X-Ray 및 Step Functions
- AWS 사용자 알림와 함께 AWS Step Functions 사용

Step Functions를 사용하여 모니터링하기 CloudWatch

모니터링은 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 AWS Step Functions 있어 중요한 부분입니다. 멀티포인트 장애를 디버깅하려면 사용하는 AWS 서비스에서 최대한 많은 모니터링 데이터를 수집해야 합니다. Step Functions 모니터링을 시작하기 전에 다음 질문에 대해 답하는 모니터링 계획을 작성해야 합니다.

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

다음 단계는 환경의 정상 성능에 대한 기준을 설정하는 것입니다. 이렇게 하려면 다양한 시간과 다양한 부하 조건에서 성능을 측정해야 합니다. Step Functions를 모니터링할 때 과거 모니터링 데이터를 저장하는 것이 좋습니다. 그러한 데이터는 현재 성능 데이터와 비교하고, 일반 성능 패턴과 성능 이상을 식별하고, 문제 해결 방법을 제안하는 기준이 될 수 있습니다.

예를 들어 Step Functions를 사용하면 하트비트 타임아웃으로 인해 실패한 액티비티 또는 AWS Lambda 태스크의 수를 모니터링할 수 있습니다. 성능이 설정된 기준 아래로 떨어지면 하트비트 간격을 변경해야 할 수 있습니다.

기준선을 설정하려면 최소한 다음 지표를 모니터링해야 합니다.

- `ActivitiesStarted`
- `ActivitiesTimedOut`
- `ExecutionsStarted`
- `ExecutionsTimedOut`
- `LambdaFunctionsStarted`
- `LambdaFunctionsTimedOut`

다음 섹션에서는 Step Functions가 아마존에 제공하는 메트릭을 설명합니다 CloudWatch. 이러한 지표를 사용하면 상태 시스템 및 활동을 추적하고 임계값에 경보를 설정할 수 있습니다. 를 사용하여 지표를 볼 수 있습니다 AWS Management Console.

시간 간격을 보고하는 지표

Step Functions CloudWatch 지표 중 일부는 시간 간격이며 항상 밀리초 단위로 측정됩니다. 이러한 지표는 일반적으로 사용자가 알기 쉬운 이름을 사용하여 상태 시스템, 활동 및 Lambda 함수 제한 시간을 설정하는 실행 단계에 해당합니다.

예를 들어, `ActivityRunTime` 지표는 활동이 실행을 시작한 후 완료될 때까지 걸리는 시간을 측정합니다. 동일한 기간에 대해 제한 시간 값을 설정할 수 있습니다.

CloudWatch 콘솔에서 시간 간격 지표의 표시 통계로 평균을 선택하면 최상의 결과를 얻을 수 있습니다.

개수를 보고하는 지표

Step Functions CloudWatch 지표 중 일부는 결과를 개수로 보고합니다. 예를 들어, `ExecutionsFailed`는 실패한 상태 시스템 실행 수를 기록합니다.

Step Functions는 모든 스테이트 머신 실행에 대해 두 개의 `ExecutionsStarted` 메트릭을 내보냅니다. 이로 인해 모든 상태 머신 실행에 대한 `ExecutionsStarted` 지표 SampleCount통계에 값 2가 표시됩니다. SampleCount 통계에는 실행 완료 `ExecutionStarted=1` 시점과 `ExecutionStarted=0` 시기가 표시됩니다.

> **Tip**
>
> 콘솔에 카운트를 보고하는 지표의 디스플레이 통계로 Sum을 선택하는 것이 좋습니다. CloudWatch

실행 지표

`AWS/States`네임스페이스에는 모든 Step Functions 실행에 대한 다음 지표가 포함됩니다. 이는 한 지역의 계정 전체에 적용되는 차원 없는 지표입니다.

지표	설명
`OpenExecutionCount`	현재 진행 중인 실행의 대략적인 수 - 계정에서 현재 진행 중인 워크플로입니다. 워크플로가 최대 실행 한도에 도달하는 시기에 대한 통찰력을 제공하여 표준 워크플로를 `StartExecution` 호출하거나 표준 워크플로를 호출할 때 ExecutionLimitExceeded오류가 발생하지 않도록 하기 `RedriveExecution` 위한 것입니다. 지표는 활성 워크플로 상태 전환에 따라 달라지므로 낮은 수준에서는 예상치가 관찰된 실행 워크플로 수와 일치하지 않을 수 있습니다.
`OpenExecutionLimit`	열려 있는 최대 실행 수입니다. 자세한 정보는 계정과 관련된 할당량을 참조하세요. 이 제한은 익스프레스 워크플로우에는 적용되지 않습니다.

버전 또는 별칭이 있는 스테이트 머신의 실행 메트릭

버전 또는 별칭을 사용하여 상태 머신을 실행하는 경우 Step Functions는 다음 지표를 내보냅니다. 이 ExecutionThrottled 지표는 실행이 제한되는 경우에만 생성됩니다. 이러한 지표에는 특정 상태 머신을 StateMachineArn 식별하기 위한 a가 포함됩니다.

지표	설명
ExecutionTime	실행 시작 시간과 종료 시간 사이의 간격 (밀리초)
ExecutionThrottled	StateEntered 병목 현상이 발생한 이벤트 및 재시도 횟수. 이 항목은 StateTransition 조절과 관련됩니다. 자세한 정보는 상태 제한과 관련된 할당량을 참조하세요.
ExecutionsAborted	중단되거나 종료된 실행 수.
ExecutionsFailed	실패한 실행 수.
ExecutionsStarted	시작된 실행 수.
ExecutionsSucceeded	성공적으로 완료된 실행 수.
ExecutionsTimedOut	어떤 이유로든 제한 시간이 초과된 실행 수.

Express 워크플로의 실행 지표

AWS/States 네임스페이스에는 Step Functions Express 워크플로 실행에 대한 다음 지표가 포함되어 있습니다.

지표	설명
ExpressExecutionMemory	Express 워크플로에서 소비한 총 메모리입니다.
ExpressExecutionBilledDuration	Express 워크플로 요금이 청구되는 기간입니다.
ExpressExecutionBilledMemory	Express 워크플로에 요금이 청구되는 메모리 사용량입니다.

표준 워크플로의 Redrive 실행 지표

상태 시스템 실행을 redrive하면 Step Functions에서 다음 지표를 내보냅니다.

모든 redriven 실행에 대해 `Executions*` 지표를 내보냅니다. 예를 들어 redriven 실행이 중단되었다고 가정해보겠습니다. 이 실행은 `RedrivenExecutionsAborted` 및 `ExecutionsAborted` 모두에 대해 0이 아닌 데이터포인트를 내보냅니다.

지표	설명
`ExecutionsRedriven`	redriven실행 횟수.
`RedrivenExecutionsAborted`	취소되거나 redriven 종료된 실행 수입니다.
`RedrivenExecutionsTimedOut`	어떤 redriven 이유로든 제한 시간이 초과된 실행 수.
`RedrivenExecutionsSucceeded`	성공적으로 완료된 redriven 실행 수입니다.
`RedrivenExecutionsFailed`	실패한 redriven 실행 수.

Step Functions 실행 지표에 대한 차원

측정기준	설명
`StateMachineArn`	해당 실행에 사용되는 상태 시스템의 Amazon 리소스 이름(ARN)입니다.

버전이 있는 실행 차원

측정기준	설명
`StateMachineArn`	버전으로 실행이 시작된 상태 시스템의 Amazon 리소스 이름(ARN)입니다.

측정기준	설명
Version	실행을 시작하는 데 사용된 상태 시스템 버전입니다.

별칭이 있는 실행 차원

측정기준	설명
StateMachineArn	별칭으로 실행이 시작된 상태 시스템의 Amazon 리소스 이름(ARN)입니다.
Alias	실행을 시작하는 데 사용된 상태 시스템 별칭입니다.

버전 및 별칭에 대한 리소스 수 지표

AWS/States 네임스페이스에는 상태 시스템의 버전 및 별칭 수에 대한 다음 지표가 포함되어 있습니다.

지표	설명
AliasCount	상태 머신에 대해 생성된 별칭 수입니다. 상태 시스템마다 별칭을 최대 100개까지 만들 수 있습니다.
VersionCount	상태 컴퓨터에 게시된 버전 수. 상태 시스템 버전을 최대 1,000개까지 게시할 수 있습니다.

버전 및 별칭에 대한 리소스 수 지표의 차원

측정기준	설명
ResourceArn	버전이나 별칭이 있는 상태 시스템의 Amazon 리소스 이름(ARN)입니다.

활동 지표

AWS/States 네임스페이스에는 Step Functions 활동에 대한 다음 지표가 포함되어 있습니다.

지표	설명
ActivityRunTime	활동 시작 시간과 종료 시간 사이의 간격 (밀리초).
ActivityScheduleTime	활동이 스케줄 상태로 유지되는 간격 (밀리초)
ActivityTime	활동이 예약된 시간과 활동이 종료되는 시간 사이의 간격 (밀리초)
ActivitiesFailed	실패한 활동 수.
ActivitiesHeartbeatTimedOut	하트비트 타임아웃으로 인해 타임아웃된 액티비티 수입니다.
ActivitiesScheduled	스케줄링된 활동 수.
ActivitiesStarted	시작된 활동 수.
ActivitiesSucceeded	성공적으로 완료한 활동 수.
ActivitiesTimedOut	마감 시 제한 시간이 초과된 활동 수입니다.

Step Functions 활동 지표의 차원

측정기준	설명
ActivityArn	활동 ARN

Lambda 함수 지표

AWS/States 네임스페이스에는 Step Functions Lambda 함수에 대한 다음 지표가 포함되어 있습니다.

지표	설명
LambdaFunctionRunTime	Lambda 함수가 시작된 시간과 종료되는 시간 사이의 간격 (밀리초).
LambdaFunctionScheduleTime	Lambda 함수가 스케줄 상태로 유지되는 간격 (밀리초).
LambdaFunctionTime	Lambda 함수가 스케줄링된 시간과 종료 시간 사이의 간격 (밀리초).
LambdaFunctionsFailed	실패한 Lambda 함수 수입니다.
LambdaFunctionsScheduled	스케줄링된 Lambda 함수의 개수.
LambdaFunctionsStarted	시작된 Lambda 함수 수입니다.
LambdaFunctionsSucceeded	성공적으로 완료된 Lambda 함수 수입니다.
LambdaFunctionsTimedOut	종료 시 제한 시간이 초과된 Lambda 함수 수입니다.

Step Functions Lambda 함수 지표의 차원

측정기준	설명
LambdaFunctionArn	Lambda 함수의 ARN입니다.

> **Note**
>
> Lambda 함수 지표를 `Resource` 필드에 Lambda 함수 ARN을 지정하는 Task 상태에 내보냅니다. `"Resource": "arn:aws:states:::lambda:invoke"`를 사용하는 Task 상태에서

대신 서비스 통합 지표를 내보냅니다. 자세한 정보는 Step Functions를 사용하여 Lambda 간접 호출을 참조하세요.

서비스 통합 지표

AWS/States 네임스페이스에는 Step Functions 서비스 통합에 대한 다음과 같은 지표가 포함되어 있습니다. 자세한 정보는 다른 서비스와 AWS Step Functions 함께 사용을 참조하세요.

측정치	설명
ServiceIntegrationRunTime	서비스 태스크가 시작된 시간과 종료되는 시간 사이의 간격 (밀리초).
ServiceIntegrationScheduleTime	서비스 태스크가 스케줄 상태로 유지되는 간격 (밀리초)
ServiceIntegrationTime	서비스 태스크가 예약된 시간과 종료되는 시간 사이의 간격 (밀리초)
ServiceIntegrationsFailed	실패한 서비스 작업 수.
ServiceIntegrationsScheduled	예약된 서비스 작업 수.
ServiceIntegrationsStarted	시작된 서비스 태스크 수.
ServiceIntegrationsSucceeded	성공적으로 완료된 서비스 태스크 수.
ServiceIntegrationsTimedOut	마감 시 제한 시간이 초과된 서비스 작업 수

Step Functions 서비스 통합 지표의 차원

측정기준	설명
ServiceIntegrationResourceArn	통합 서비스의 리소스 ARN입니다.

서비스 지표

AWS/States 네임스페이스에는 Step Functions 서비스에 대한 다음과 같은 지표가 포함되어 있습니다.

지표	설명
ThrottledEvents	병목 현상이 발생한 요청 수.
ProvisionedBucketSize	초당 사용 가능한 요청 수.
ProvisionedRefillRate	버킷에 허용된 초당 요청 수입니다.
ConsumedCapacity	초당 요청 수.

Step Functions 서비스 지표의 차원

측정기준	설명
ServiceMetric	State Transitions 수치를 보여 주는 데이터를 필터링합니다.

API 지표

AWS/States 네임스페이스에는 Step Functions API에 대한 다음과 같은 지표가 포함되어 있습니다.

지표	설명
ThrottledEvents	병목 현상이 발생한 요청 수입니다.
ProvisionedBucketSize	초당 사용 가능한 요청 수.
ProvisionedRefillRate	버킷에 허용된 초당 요청 수입니다.
ConsumedCapacity	초당 요청 수.

Step Functions API 지표의 차원

측정기준	설명
APIName	지정한 API 이름의 API에 대한 데이터를 필터링합니다.

최선을 다한 CloudWatch 지표 전달

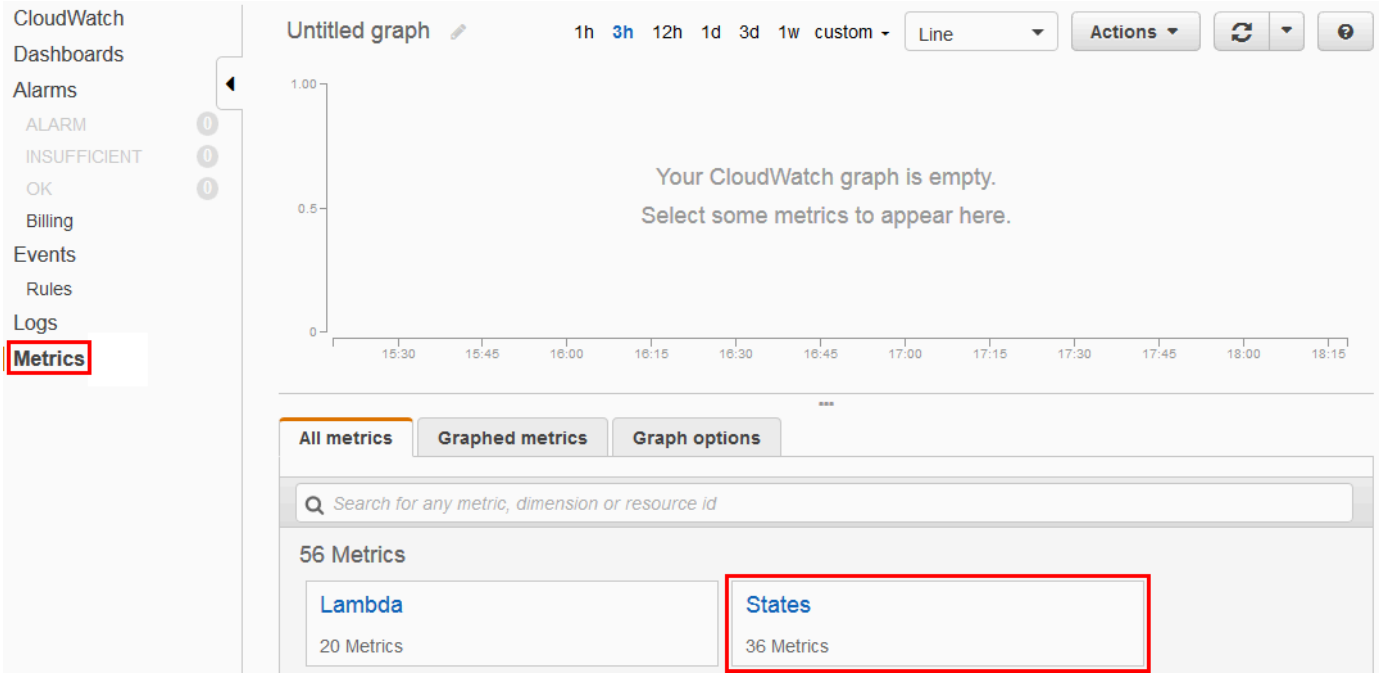
CloudWatch 지표를 전송하고자 최선을 다할 것입니다.

모든 지표가 제때 전송될 것이라고 보장할 수는 없습니다. 특정 요청에 대한 데이터 요소는 그 요청이 실제로 처리되었을 때보다 더 늦은 타임스탬프와 함께 반환될 수도 있습니다. 1분 동안의 데이터 포인트가 사용 가능해지기까지 지연되거나 전혀 전달되지 않을 수 있습니다. CloudWatch CloudWatch요청 지표를 통해 상태 시스템 실행을 거의 실시간으로 파악할 수 있습니다. 모든 실행 관련 지표를 완벽하게 제공한다는 의미는 아닙니다.

완벽한 전송을 보장할 수 없는 그 특성에 따라 결제 및 비용 관리 대시보드에 제공되는 보고서에는 실행 지표에 나타나지 않는 액세스 요청이 하나 이상 포함될 수 있습니다.

Step Functions에 대한 지표 보기

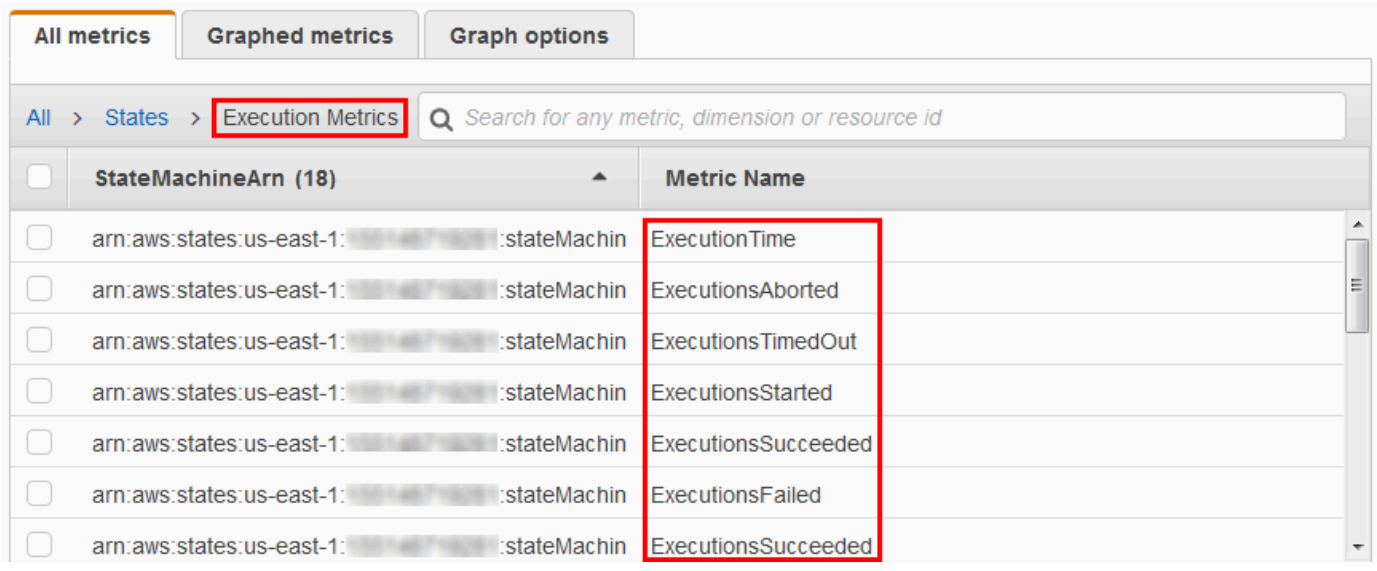
1. 에 AWS Management Console 로그인하고 콘솔을 엽니다. CloudWatch
2. 지표를 선택하고 모든 지표 탭에서 상태를 선택합니다.



최근에 실행을 실행했다면 다음과 같은 유형의 지표가 최대 4개까지 표시됩니다.

- 실행 지표
- 활동 기능 지표
- Lambda 함수 지표
- 서비스 통합 지표

3. 지표 목록을 보려는 지표 유형을 선택합니다.



- 지표 이름 또는 StateMachineArn기준으로 지표를 정렬하려면 열 제목을 사용하세요.

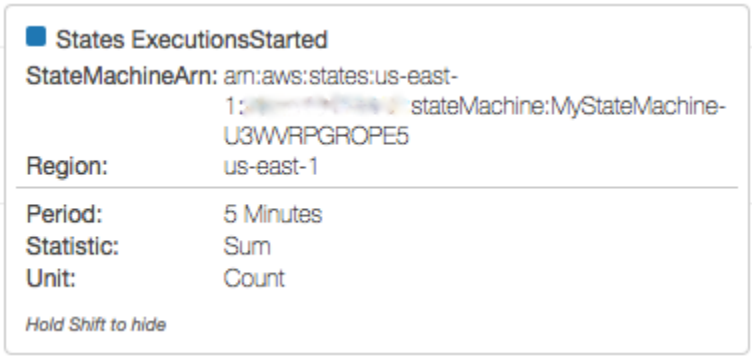
- 지표에 대한 그래프를 보려면 목록에서 지표 옆의 확인란을 선택합니다. 그래프 보기 위의 시간 범위 컨트롤을 사용하여 그래프 파라미터를 변경할 수 있습니다.

  상대값 또는 절대값(특정 날짜 및 시간)을 사용하여 사용자 지정 시간 범위를 선택할 수 있습니다. 또한 드롭다운 목록을 사용하여 값을 선, 스택형 영역 또는 숫자(값)으로 표시할 수 있습니다.

- 그래프에 대한 세부 정보를 보려면 그래프 아래 나타나는 지표 색상 코드 위에 마우스 포인터를 놓으십시오.

  지표의 세부 정보가 표시됩니다.

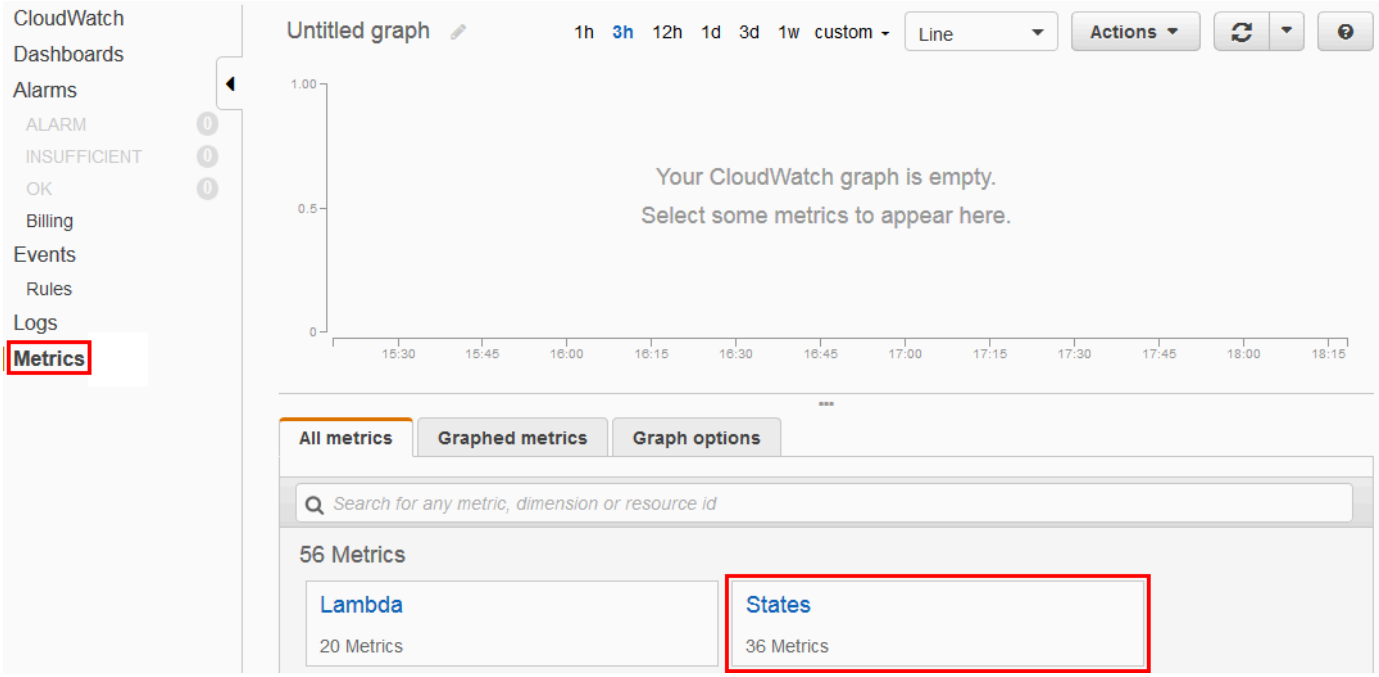
CloudWatch 메트릭 사용에 대한 자세한 내용은 Amazon 사용 CloudWatch 설명서의 Amazon CloudWatch 메트릭스 사용을 참조하십시오.

Step Functions에 대한 경보 설정

Amazon CloudWatch 경보를 사용하여 작업을 수행할 수 있습니다. 예를 들어 경보 임계 값 도달 시기를 알고 싶으면 `StateMachinesFailed` 지표가 특정 임계 값을 초과할 때 알림을 Amazon SNS 주제에 보내거나 이메일을 보내도록 경보를 설정하면 됩니다.

지표에 대해 경보를 설정하려면

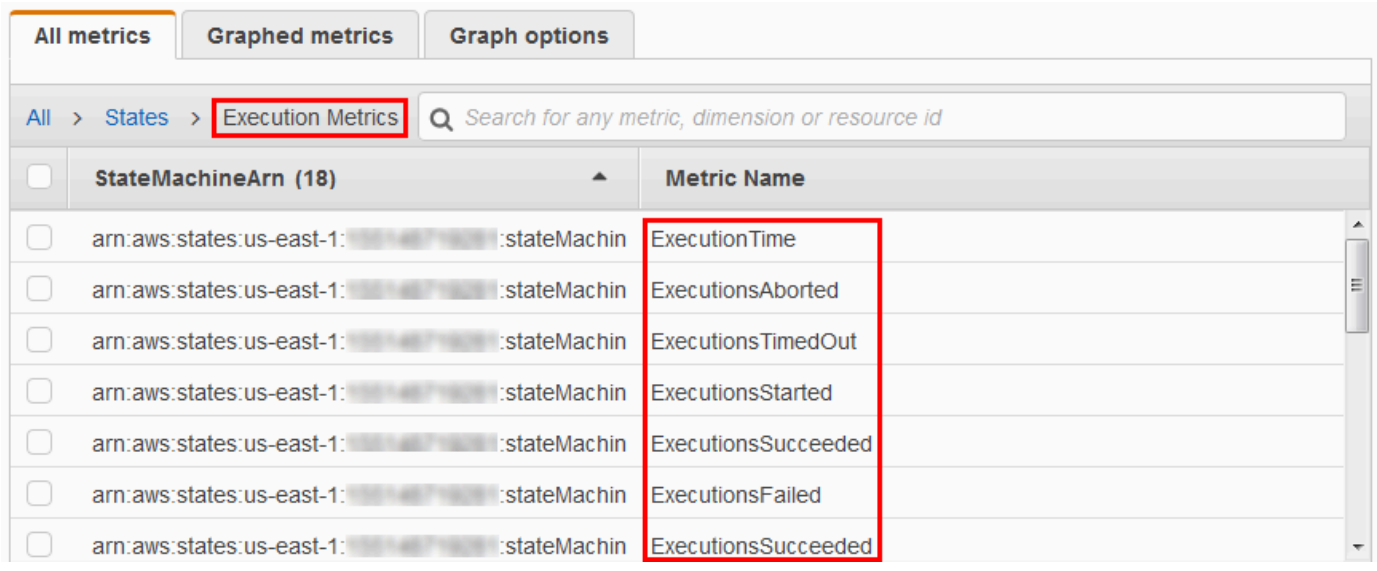
1. 에 AWS Management Console 로그인하고 CloudWatch 콘솔을 엽니다.
2. 지표를 선택하고 모든 지표 탭에서 상태를 선택합니다.



최근에 실행을 실행했다면 다음과 같은 유형의 지표가 최대 4개까지 표시됩니다.

- 실행 지표
- 활동 기능 지표
- Lambda 함수 지표
- 서비스 통합 지표

3. 지표 목록을 보려는 지표 유형을 선택합니다.

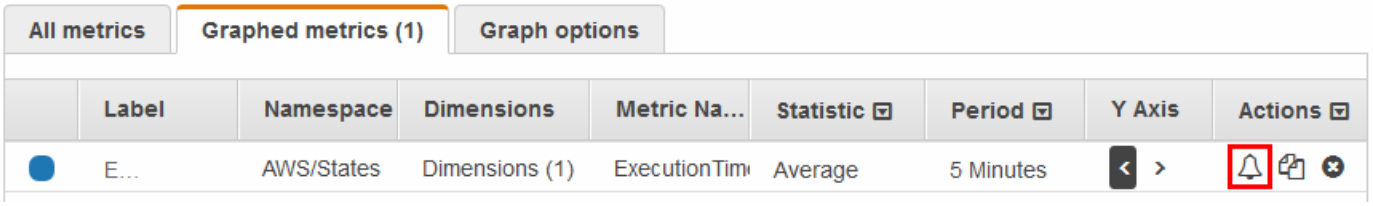


4. 지표를 선택한 후 그래프로 표시된 지표를 선택합니다.

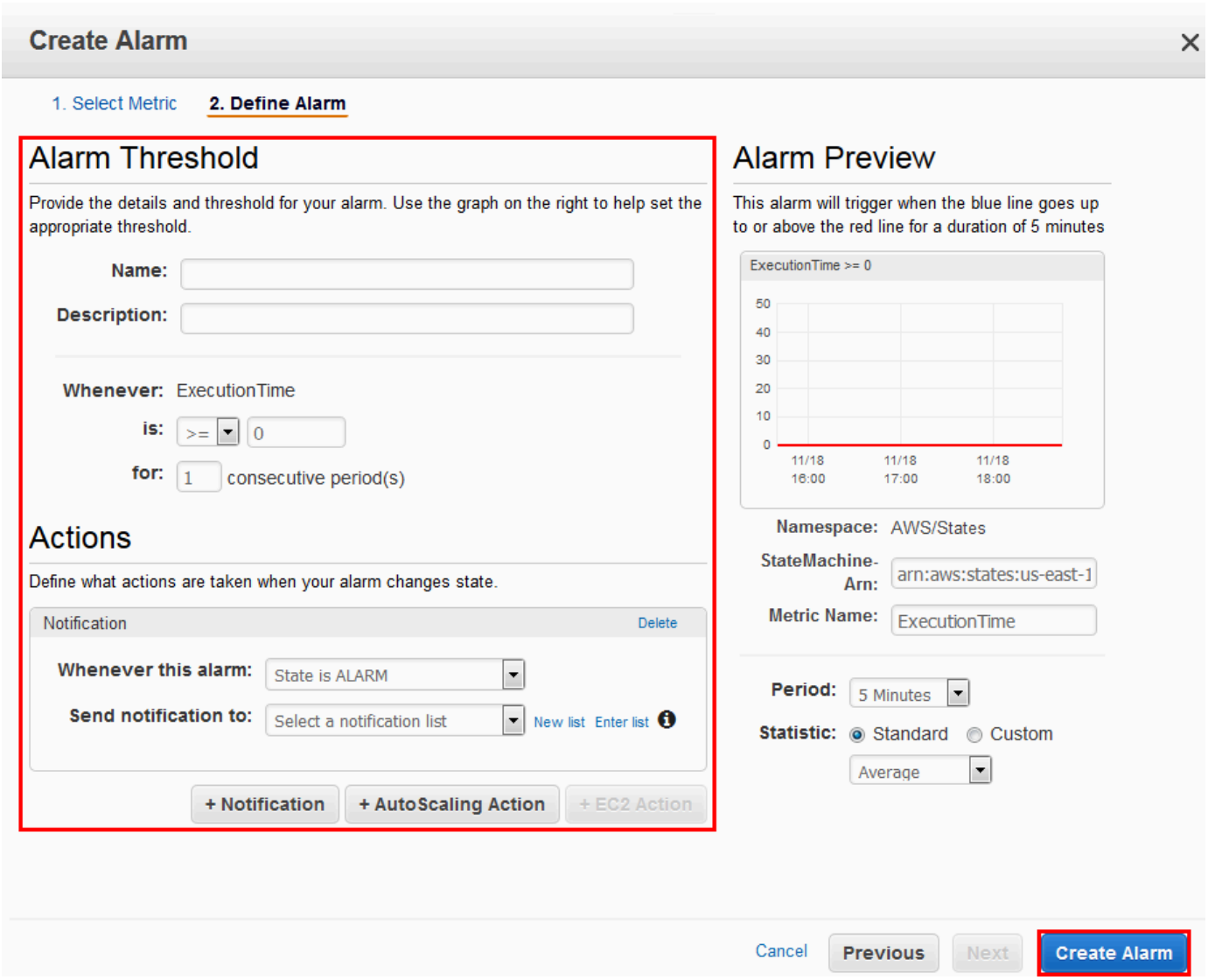
5. 목록에서 지표 옆의

을

선택합니다.



경보 생성 페이지가 표시됩니다.



6. 경보 임계값 및 작업에 값을 입력한 후 경보 생성을 선택합니다.

CloudWatch 경보 설정 및 사용에 대한 자세한 내용은 Amazon 사용 CloudWatch 설명서의 Amazon CloudWatch 경보 생성을 참조하십시오.

EventBridge Step Functions의 실행 상태 변경에 대한 (CloudWatch 이벤트)

EventBridge Amazon은 AWS 리소스의 상태 변경에 응답할 수 있게 해주는 AWS 서비스입니다. 예를 들어, 다음 두 가지 방법을 EventBridge 사용하여 Step Functions 표준 워크플로우의 실행 상태 변경에 응답할 수 있습니다.

- Step Functions 상태 머신의 실행 상태가 변경될 때 발생하는 이벤트에 반응하도록 EventBridge 규칙을 구성할 수 있습니다. 이를 통해 DescribeExecution API를 사용하여 지속적으로 폴링하지 않고도 워크플로를 모니터링할 수 있습니다. 상태 시스템 실행의 변화에 따라 EventBridge 대상을 사용하여 새 상태 시스템 실행을 시작하고, AWS Lambda 함수를 호출하고, Amazon Simple Notification Service (Amazon SNS) 주제에 메시지를 게시하는 등의 작업을 수행할 수 있습니다.
- 에서 Step Functions 상태 머신을 대상으로 구성할 수도 EventBridge 있습니다. 이를 통해 다른 AWS 서비스의 이벤트에 대한 응답으로 Step Functions 워크플로 실행을 트리거할 수 있습니다.

  자세한 내용은 Amazon EventBridge 사용 설명서를 참조하십시오.

하지만 익스프레스 워크플로는 이벤트를 에 EventBridge 내보내지 않습니다. CloudWatch 로그를 사용하여 익스프레스 워크플로의 실행을 모니터링할 수 있습니다. 이렇게 하려면 상태 시스템 실행 세부 정보 페이지에서 모니터링 및 로깅 탭을 선택합니다. 모니터링 탭에서 실행 기간, 실행 오류, 청구 메모리와 같은 이벤트 CloudWatch 지표를 볼 수 있습니다. 로깅 탭에서는 최근 로그와 로깅 구성을 볼 수 있습니다.

> **Tip**
>
> 익스프레스 워크플로우의 예제를 AWS 계정 배포하고 익스프레스 워크플로를 모니터링하는 방법을 알아보려면 AWS Step Functions 워크숍의 익스프레스 워크플로우 모니터링 모듈을 참조하십시오.

EventBridge 페이로드

EventBridge 이벤트 정의에 입력 속성을 포함할 수 있습니다. 일부 이벤트의 경우 EventBridge 이벤트 정의에 출력 속성을 포함할 수도 있습니다.

- 로 전송된 이스케이프 입력과 이스케이프된 출력을 합한 EventBridge 값이 248KB를 초과하는 경우 입력은 제외됩니다. 마찬가지로 이스케이프 처리된 출력이 248KB를 초과하면 출력이 제외됩니다. 이는 이벤트 할당량의 결과입니다. EventBridge
- `inputDetails` 및 `outputDetails` 속성을 사용하여 페이로드가 잘렸는지 여부를 확인할 수 있습니다. 자세한 내용은`CloudWatchEventsExecutionDataDetails` 데이터 유형을 참조하세요.
- 표준 워크플로의 경우 를 사용하여 전체 입력 및 출력을 볼 수 있습니다. DescribeExecution
- Express 워크플로에 `DescribeExecution`을 사용할 수 없습니다. 전체 입력/출력을 확인하려면 Express 워크플로를 표준 워크플로로 래핑하면 됩니다. 다른 옵션은 Amazon S3 ARN을 사용하는 것입니다. ARN 사용에 대한 자세한 내용은 the section called “대용량 페이로드를 전달하는 대신 Amazon S3 ARN 사용”을 참조하세요.

주제

Step Functions 이벤트 예제

다음은 Step Functions가 이벤트를 보내는 예제입니다 EventBridge.

주제

각각의 경우, 이벤트 데이터의 `detail` 섹션은 `DescribeExecution` API와 동일한 정보를 제공합니다. `status` 필드는 방출된 이벤트에 따라 `RUNNING`, `SUCCEEDED`, `FAILED`, `TIMED_OUT`, `ABORTED` 중 하나인 이벤트가 송신된 시점의 실행 상태를 나타냅니다.

실행 시작

```
{
    "version": "0",
```

```
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws::states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "RUNNING",
        "startDate": 1551225271984,
        "stopDate":  null,
        "input": "{}",
        "inputDetails": {
             "included": true
        },
        "output": null,
        "outputDetails": null
    }
}
```

실행 성공

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
```

```
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "SUCCEEDED",
        "startDate": 1547148840101,
        "stopDate": 1547148840122,
        "input": "{}",
        "inputDetails": {
             "included": true
        },
        "output": "\"Hello World!\"",
        "outputDetails": {
             "included": true
        }
    }
}
```

실행 실패

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "FAILED",
        "startDate": 1551225146847,
        "stopDate": 1551225151881,
        "input": "{}",
```

```
        "inputDetails": {
             "included": true
        },
        "output": null,
        "outputDetails": null
    }
}
```

실행 시간 초과

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "TIMED_OUT",
        "startDate": 1551224926156,
        "stopDate": 1551224927157,
        "input": "{}",
        "inputDetails": {
             "included": true
        },
        "output": null,
        "outputDetails": null
    }
}
```

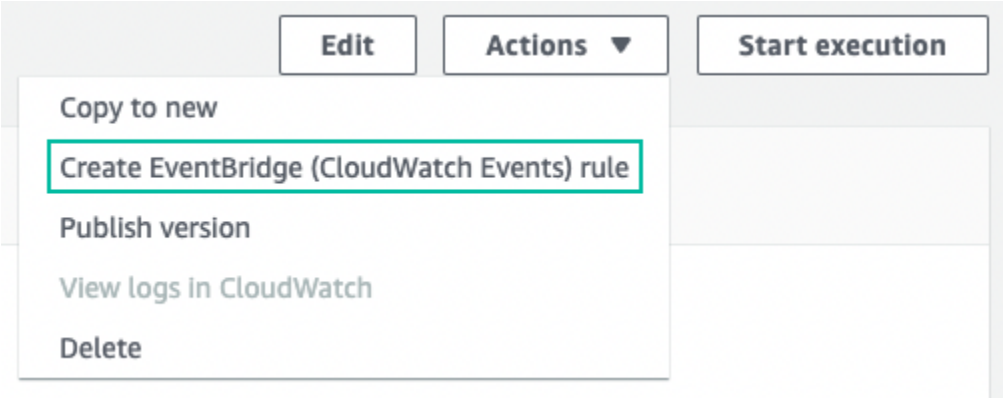
실행 중단

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73e60175",
    "detail-type": "Step Functions Execution Status Change",
    "source": "aws.states",
    "account": "123456789012",
    "time": "2019-02-26T19:42:21Z",
    "region": "us-east-2",
    "resources": [
      "arn:aws:states:us-east-2:123456789012:execution:state-machine-name:execution-
name"
    ],
    "detail": {
        "executionArn": "arn:aws:states:us-east-2:123456789012:execution:state-machine-
name:execution-name",
        "stateMachineArn": "arn:aws:states:us-east-2:123456789012:stateMachine:state-
machine",
        "name": "execution-name",
        "status": "ABORTED",
        "startDate": 1551225014968,
        "stopDate": 1551225017576,
        "input": "{}",
        "inputDetails": {
             "included": true
        },
        "output": null,
        "outputDetails": null
    }
}
```

Step Functions 이벤트를 EventBridge 콘솔로 라우팅하기 EventBridge

다음 지침을 사용하여 특정 Step Functions 상태 시스템 실행이 성공적으로 완료될 때마다 Step Functions 상태 시스템 실행을 트리거하는 방법을 알아봅니다. Amazon EventBridge 콘솔을 사용하여 실행을 트리거하려는 상태 머신을 지정합니다.

1. 스테이트 머신의 세부 정보 페이지에서 [Actions] 을 선택한 다음 [Create EventBridge (CloudWatch Events) 규칙] 을 선택합니다.



또는 https://console.aws.amazon.com/events/ 에서 EventBridge 콘솔을 열 수도 있습니다. 탐색 창의 버스 아래에서 규칙을 선택합니다.

2. 규칙 생성을 선택합니다. 그러면 규칙 세부 정보 정의 페이지가 열립니다.
3. 규칙의 이름(예: *StepFunctionsEventRule*)을 입력하고 선택적으로 규칙에 대한 설명을 입력합니다.
4. 이벤트 버스 및 규칙 유형에서는 기본 선택을 유지합니다.
5. 다음을 선택합니다. 그러면 이벤트 패턴 작성 페이지가 열립니다.
6. 이벤트 소스에서 이벤트 또는 EventBridge 파트너 AWS 이벤트의 기본 선택을 유지합니다.
7. 샘플 이벤트 및 생성 방법 섹션의 기본 선택을 유지합니다.
8. 이벤트 패턴에서 다음을 수행합니다.
   a. 이벤트 소스 드롭다운 목록에서 AWS 서비스의 기본 선택을 유지합니다.
   b. AWS 서비스 드롭다운 목록에서 Step Functions를 선택합니다.
   c. 이벤트 유형 드롭다운 목록에서 Step Functions 실행 상태 변경을 선택합니다.
   d. (선택 사항) 특정 상태, 상태 시스템 Amazon 리소스 이름(ARN) 또는 실행 ARN을 구성합니다. 이 절차의 경우 특정 상태를 선택한 다음 드롭다운 목록에서 성공을 선택합니다.
9. 다음을 선택합니다. 그러면 대상 선택 페이지가 열립니다.
10. 대상 유형에서 AWS 서비스의 기본 선택을 유지합니다.
11. 대상 선택 드롭다운 목록에서 AWS 서비스를 선택합니다. 예를 들어 Lambda 함수를 시작하거나 Step Functions 상태 시스템을 실행할 수 있습니다. 이 절차의 경우 Step Functions 상태 시스템을 선택합니다.
12. 상태 시스템 드롭다운 목록에서 상태 시스템을 선택합니다.
13. 실행 역할에서 이 특정 리소스에 대해 새 역할 생성 기본 선택을 유지합니다.
14. 다음을 선택합니다. 그러면 태그 구성 페이지가 열립니다.

15. 다음을 다시 선택합니다. 그러면 검토 및 생성 페이지가 열립니다.
16. 규칙의 세부 정보를 검토하고 규칙 생성을 선택합니다.

    규칙이 생성되고 모든 Amazon 규칙을 나열하는 EventBridge 규칙 페이지가 표시됩니다.

를 사용하여 API 호출 녹음하기 AWS CloudTrail

AWS Step Functions 사용자 AWS CloudTrail, 역할 또는 담당자가 수행한 작업의 기록을 제공하는 서비스와 통합되어 AWS 서비스있습니다. CloudTrail 모든 API 호출을 Step Functions 이벤트로 캡처합니다. 캡처된 호출에는 Step Functions 콘솔에서의 호출 및 Step Functions API 작업에 대한 코드 호출이 포함됩니다. 에서 수집한 CloudTrail 정보를 사용하여 요청을 받은 사람 Step Functions, 요청한 IP 주소, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 신원 정보를 이용하면 다음을 쉽게 알아볼 수 있습니다.

- 요청을 루트 사용자로 했는지 사용자 보안 인증으로 했는지 여부.
- IAM Identity Center 사용자를 대신하여 요청이 이루어졌는지 여부입니다.
- 역할 또는 연동 사용자를 위한 임시 보안 인증으로 요청을 생성하였는지.
- 다른 AWS 서비스에서 요청했는지 여부.

CloudTrail 계정을 만들 AWS 계정 때 활성화되며 자동으로 CloudTrail 이벤트 기록에 액세스할 수 있습니다. CloudTrail 이벤트 기록은 지난 90일간의 기록된 관리 이벤트를 보고, 검색하고, 다운로드할 수 있고, 변경할 수 없는 기록을 제공합니다. AWS 리전자세한 내용은 사용 설명서의 CloudTrail 이벤트 기록 사용을 참조하십시오.AWS CloudTrail 이벤트 기록 조회에는 CloudTrail 요금이 부과되지 않습니다.

AWS 계정 지난 90일 동안 진행 중인 이벤트 기록을 보려면 트레일 또는 CloudTrail호수 이벤트 데이터 저장소를 생성하세요.

CloudTrail 트레일

트레일을 사용하면 CloudTrail Amazon S3 버킷에 로그 파일을 전송할 수 있습니다. 를 사용하여 생성된 모든 트레일은 멀티 AWS Management Console 리전입니다. AWS CLI를 사용하여 단일 리전 또는 다중 리전 추적을 생성할 수 있습니다. 계정의 모든 활동을 기록할 수 있으므로 멀티 리전 트레일을 생성하는 것이 좋습니다 AWS 리전 . 단일 리전 추적을 생성하는 경우 추적의 AWS 리전에 로

깅된 이벤트만 볼 수 있습니다. 추적에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 Creating a trail for your AWS 계정 및 Creating a trail for an organization을 참조하세요.

트레일을 CloudTrail 생성하여 진행 중인 관리 이벤트의 사본 하나를 Amazon S3 버킷으로 무료로 전송할 수 있지만 Amazon S3 스토리지 요금이 부과됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하십시오. Amazon S3 요금에 대한 자세한 내용은 Amazon S3 요금을 참조하세요.

CloudTrail 레이크 이벤트 데이터 스토어

CloudTrail Lake를 사용하면 이벤트에 대한 SQL 기반 쿼리를 실행할 수 있습니다. CloudTrail Lake는 행 기반 JSON 형식의 기존 이벤트를 Apache ORC 형식으로 변환합니다. ORC는 빠른 데이터 검색에 최적화된 열 기반 스토리지 형식입니다. 이벤트는 이벤트 데이터 스토어로 집계되며, 이벤트 데이터 스토어는 고급 이벤트 선택기를 적용하여 선택한 기준을 기반으로 하는 변경 불가능한 이벤트 컬렉션입니다. 이벤트 데이터 스토어에 적용하는 선택기는 어떤 이벤트가 지속되고 쿼리할 수 있는지 제어합니다. CloudTrail Lake에 대한 자세한 내용은 사용 설명서의 Lake 사용을 참조하십시오. AWS CloudTrail AWS CloudTrail

CloudTrail Lake 이벤트 데이터 저장 및 쿼리로 인해 비용이 발생합니다. 이벤트 데이터 스토어를 생성할 때 이벤트 데이터 스토어에 사용할 요금 옵션을 선택합니다. 요금 옵션에 따라 이벤트 모으기 및 저장 비용과 이벤트 데이터 스토어의 기본 및 최대 보존 기간이 결정됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하십시오.

의 데이터 이벤트 CloudTrail

데이터 이벤트는 리소스 기반 또는 리소스에서 수행된 리소스 작업에 대한 정보를 제공합니다(예: Amazon S3 객체 읽기 또는 쓰기). 이를 데이터 영역 작업이라고도 합니다. 데이터 이벤트가 대량 활동인 경우도 있습니다. 기본적으로 데이터 이벤트를 기록하지 CloudTrail 않습니다. CloudTrail 이벤트 기록에는 데이터 이벤트가 기록되지 않습니다.

데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하십시오.

CloudTrail 콘솔 또는 CloudTrail API 작업을 사용하여 Step Functions 리소스 유형에 대한 데이터 이벤트를 기록할 수 있습니다. AWS CLI데이터 이벤트를 로깅하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 Logging data events with the AWS Management Console 및 Logging data events with the AWS Command Line Interface를 참조하세요.

다음 표에는 데이터 이벤트를 기록할 수 있는 Step Functions 리소스 유형이 나열되어 있습니다. 데이터 이벤트 유형 열에는 CloudTrail 콘솔의 데이터 이벤트 유형 목록에서 선택할 수 있는 값이 표시됩니

다. resources.type 값 열에는 또는 `resources.type` API를 사용하여 고급 이벤트 선택기를 구성할 때 지정하는 값이 표시됩니다. AWS CLI CloudTrail 데이터 API 로깅 대상 CloudTrail 열에는 해당 리소스 유형에 대해 로깅된 API 호출이 표시됩니다. CloudTrail

`eventName`, `readOnly` 및 `resources.ARN` 필드를 필터링하여 중요한 이벤트만 로깅하도록 고급 이벤트 선택기를 구성할 수 있습니다. 이러한 필드에 대한 자세한 내용은 AWS CloudTrail API 참조의 AdvancedFieldSelector 섹션을 참조하세요.

데이터 이벤트 유형	resources.type 값	로깅된 데이터 API CloudTrail
Step Functions 상태 시스템	`AWS::StepFunctions::StateMachine`	• InvokeHTTPEndpoint

의 관리 이벤트 CloudTrail

관리 이벤트는 내 리소스에 대해 수행되는 관리 작업에 대한 정보를 제공합니다 AWS 계정. 이를 제어 영역 작업이라고도 합니다. 기본적으로 관리 이벤트를 CloudTrail 기록합니다.

스테이트 머신

- CreateStateMachine
- ListStateMachines
- DescribeStateMachine
- UpdateStateMachine
- DeleteStateMachine
- ValidateStateMachineDefinition
- TestState

스테이트 머신 앨리어스

- CreateStateMachineAlias
- ListStateMachineAliases
- DescribeStateMachineAlias
- UpdateStateMachineAlias
- DeleteStateMachineAlias

스테이트 머신 버전

- ListStateMachineVersions
- PublishStateMachineVersion
- DeleteStateMachineVersion

실행

- StartExecution
- StartSyncExecution
- RedriveExecution
- ListExecutions
- DescribeExecution
- GetExecutionHistory
- DescribeStateMachineForExecution
- StopExecution

활동

- CreateActivity
- ListActivities
- DescribeActivity
- DeleteActivity
- GetActivityTask

태스크 토큰

- SendTaskSuccess
- SendTaskHeartbeat
- SendTaskFailure

MapRun

- ListMapRuns

- DescribeMapRun
- UpdateMapRun

태그

- ListTagsForResource
- TagResource
- UntagResource

이벤트 예

이벤트는 모든 소스의 단일 요청을 나타내며 요청된 API 작업, 작업 날짜 및 시간, 요청 파라미터 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 트레이스가 아니므로 이벤트가 특정 순서로 표시되지 않습니다.

다음 예제는 이를 보여주는 CloudTrail 데이터 이벤트를 보여줍니다. `InvokeHTTPEndpoint`


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "accountId": "123456789012",
        "invokedBy": "states.amazonaws.com"
    },
    "eventTime": "2024-05-01T01:23:45Z",
    "eventSource": "states.amazonaws.com",
    "eventName": "InvokeHTTPEndpoint",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "states.amazonaws.com",
    "userAgent": "states.amazonaws.com",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "readOnly": false,
    "resources": [
        {
            "accountId": "123456789012",
            "type": "AWS::StepFunctions::StateMachine",
            "ARN": "arn:aws:states:us-
east-1:123456789012:stateMachine:ExampleStateMachine"
        }
```

```
    ],
    "eventType": "AwsServiceEvent",
    "managementEvent": false,
    "recipientAccountId": "123456789012",
    "serviceEventDetails": {
        "httpMethod": "GET",
        "httpEndpoint": "https://example.com"
    },
    "eventCategory": "Data"
}
```

다음 예제는 작업을 보여주는 CloudTrail 관리 이벤트를 보여줍니다. `CreateStateMachine`

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAJYDLDBVBI4EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/test-user",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "test-user"
    },
    "eventTime": "2024-05-01T01:23:45Z",
    "eventSource": "states.amazonaws.com",
    "eventName": "CreateStateMachine",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "name": "MyStateMachine",
        "definition": "HIDDEN_DUE_TO_SECURITY_REASONS",
        "roleArn": "arn:aws:iam::123456789012:role/MyStateMachineRole",
        "type": "STANDARD",
        "loggingConfiguration": {
            "level": "OFF",
            "includeExecutionData": false
        },
        "tags": [],
        "tracingConfiguration": {
            "enabled": false
        },
        "publish": false
```

```
    },
    "responseElements": {
        "stateMachineArn": "arn:aws:states:us-
east-1:123456789012:stateMachine:MyStateMachine",
        "creationDate": "May 1, 2024 1:23:45 AM"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```


CloudTrail 레코드 내용에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 레코드 내용을 참조하십시오.

CloudWatch Logs를 사용하여 로깅

표준 워크플로는 AWS Step Functions의 실행 내역을 기록하지만 선택적으로 Amazon CloudWatch Logs에 로깅을 구성할 수 있습니다.

표준 워크플로처럼 Express 워크플로는 AWS Step Functions의 실행 내역을 기록하지 않습니다. Express 워크플로의 실행 내역과 결과를 보려면 Amazon CloudWatch Logs에 로깅을 구성해야 합니다. 로그를 게시해도 실행이 차단되거나 실행 속도가 느려지지 않습니다.

> **Note**
>
> 로깅을 구성하면 CloudWatch Logs 요금이 적용되며 판매 로그 비율로 요금이 결제됩니다. 자세한 내용은 CloudWatch 요금의 로그 탭에서 판매 로그를 참조하세요.

로깅 구성

Step Functions 콘솔을 사용하여 표준 워크플로를 만들 때 CloudWatch Logs에 로깅을 활성화하도록 구성되지 않습니다. Step Functions 콘솔을 사용하여 생성된 Express 워크플로는 기본적으로 CloudWatch Logs에 로깅을 활성화하도록 구성됩니다.

Express 워크플로의 경우 Step Functions는 CloudWatch Logs에 필요한 AWS Identity and Access Management(IAM) 정책이 있는 역할을 만들 수 있습니다. API, CLI 또는 AWS CloudFormation을 사용

하여 표준 워크플로나 Express 워크플로를 만드는 경우 Step Functions는 기본적으로 로깅을 활성화하지 않으므로 역할에 필요한 권한이 있는지 확인해야 합니다.

콘솔에서 시작된 실행마다 Step Functions는 해당 실행별 로그 이벤트를 가져오도록 올바른 필터가 구성된 CloudWatch Logs에 대한 링크를 제공합니다.

로깅을 구성하려면 CreateStateMachine 또는 UpdateStateMachine을 사용할 때 LoggingConfiguration 파라미터를 전달하면 됩니다. CloudWatch Logs Insights를 사용하여 CloudWatch Logs의 데이터를 자세히 분석할 수 있습니다. 자세한 내용은 CloudWatch Logs Insights를 사용한 로그 데이터 분석을 참조하십시오.

CloudWatch Logs 페이로드

실행 내역 이벤트에 해당 정의의 입력 또는 출력 속성이 포함될 수 있습니다. CloudWatch Logs로 전송된 이스케이프된 입력이나 이스케이프된 출력이 248KB를 초과하면 CloudWatch Logs 할당량으로 인해 잘립니다.

- `inputDetails` 및 `outputDetails` 속성을 검토하여 페이로드가 잘렸는지 여부를 확인할 수 있습니다. 자세한 내용은`HistoryEventExecutionDataDetails` 데이터 유형을 참조하세요.
- 표준 워크플로의 경우 `GetExecutionHistory`를 사용하여 전체 실행 내역을 확인할 수 있습니다.
- Express 워크플로에 `GetExecutionHistory`을 사용할 수 없습니다. 전체 입력 및 출력을 확인하려면 Amazon S3 ARN을 사용하면 됩니다. 자세한 내용은 the section called “대용량 페이로드를 전달하는 대신 Amazon S3 ARN 사용” 섹션을 참조하세요.

CloudWatch Logs에 로깅하기 위한 IAM 정책

또한 다음 예제와 같이 CloudWatch Logs에 로깅할 수 있는 적절한 권한이 있도록 상태 시스템 실행 IAM 역할을 구성해야 합니다.

IAM 정책 예시

다음은 권한을 구성하는 데 사용할 수 있는 정책의 예입니다. 다음 예제와 같이 CreateLogDelivery 및 DescribeLogGroups와 같은 CloudWatch API 작업은 Amazon CloudWatch Logs에서 정의한 리소스 유형을 지원하지 않으므로 `Resource` 필드에 *를 지정해야 합니다. 자세한 내용은 Amazon CloudWatch Logs에서 정의한 작업을 참조하세요.

- CloudWatch 리소스에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 CloudWatch Logs 리소스 및 작업을 참조하세요.

- CloudWatch Logs로 로그를 전송하도록 설정하는 데 필요한 권한에 대한 자세한 내용은 CloudWatch Logs로 전송된 로그 섹션의 사용자 권한을 참조하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:CreateLogStream",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutLogEvents",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

CloudWatch Logs에 액세스할 수 없음

CloudWatch Logs에 액세스할 수 없으면 다음을 완료했는지 확인합니다.

1. CloudWatch Logs에 로깅할 수 있는 적절한 권한이 있도록 상태 시스템 실행 IAM 역할을 구성했습니다.

   CreateStateMachine 또는 UpdateStateMachine 요청을 사용하는 경우 앞선 예제와 같이 권한이 포함된 `roleArn` 파라미터에 IAM 역할을 지정했는지 확인합니다.

2. CloudWatch Logs 리소스 정책이 CloudWatch Logs 리소스 정책의 5120자 제한을 초과하지 않는지 확인했습니다.

   문자 제한을 초과한 경우 CloudWatch Logs 리소스 정책에서 불필요한 권한을 제거하거나 로그 그룹 이름 앞에 `/aws/vendedlogs` 접두사를 추가합니다. 그러면 리소스 정책에 문자를 더 추가하지 않고도 로그 그룹에 권한이 부여됩니다. Step Functions 콘솔에서 로그 그룹을 만들면 로그 그룹

이름에 접두사 `/aws/vendedlogs/states`가 추가됩니다. 자세한 내용은 Amazon CloudWatch Logs 리소스 정책 크기 제한 섹션을 참조하세요.

로그 수준

`OFF`, `ALL`, `ERROR` 또는 `FATAL`중에서 선택할 수 있습니다. `OFF`로 설정하면 어떠한 이벤트 유형도 로깅되지 않고, `ALL`로 설정하면 모든 이벤트 유형이 로깅됩니다. `ERROR` 및 `FATAL`에 대해서는 다음 표를 참조하십시오.

이러한 로그 수준을 기반으로 Express 워크플로 실행에 대해 표시되는 실행 데이터에 대한 자세한 내용은 콘솔에서의 표준 및 Express 워크플로 실행을 참조하세요.

이벤트 유형	**ALL**	**ERROR**	**FATAL**	**OFF**
ChoiceStateEntered	✓			
ChoiceStateExited	✓			
ExecutionAborted	✓	✓	✓	
ExecutionFailed	✓	✓	✓	
ExecutionStarted	✓			
ExecutionSucceeded	✓			
ExecutionTimedOut	✓	✓	✓	
FailStateEntered	✓	✓		
LambdaFunctionFailed	✓	✓		

이벤트 유형	ALL	ERROR	FATAL	OFF
LambdaFunctionScheduled	✓			
LambdaFunctionScheduleFailed	✓	✓		
LambdaFunctionStarted	✓			
LambdaFunctionStartFailed	✓	✓		
LambdaFunctionSucceeded	✓			
LambdaFunctionTimedOut	✓	✓		
MapIterationAborted	✓	✓		
MapIterationFailed	✓	✓		
MapIterationStarted	✓			
MapIterationSucceeded	✓			
MapRunAborted	✓	✓		
MapRunFailed	✓	✓		
MapStateAborted	✓	✓		

이벤트 유형	ALL	ERROR	FATAL	OFF
MapStateEntered	✓			
MapStateExited	✓			
MapStateFailed	✓	✓		
MapStateStarted	✓			
MapStateSucceeded	✓			
ParallelStateAborted	✓	✓		
ParallelStateEntered	✓			
ParallelStateExited	✓			
ParallelStateFailed	✓	✓		
ParallelStateStarted	✓			
ParallelStateSucceeded	✓			
PassStateEntered	✓			
PassStateExited	✓			
SucceedStateEntered	✓			

이벤트 유형	ALL	ERROR	FATAL	OFF
SucceedStateExited	✓			
TaskFailed	✓	✓		
TaskScheduled	✓			
TaskStarted	✓			
TaskStartFailed	✓	✓		
TaskStateAborted	✓	✓		
TaskStateEntered	✓			
TaskStateExited	✓			
TaskSubmitFailed	✓	✓		
TaskSubmitted	✓			
TaskSucceeded	✓			
TaskTimedOut	✓	✓		
WaitStateAborted	✓	✓		
WaitStateEntered	✓			
WaitStateExited	✓			

AWS X-Ray 및 Step Functions

AWS X-Ray를 사용하여 상태 시스템 구성 요소를 시각화하고 성능 병목 현상을 식별하며 오류가 발생한 요청 문제를 해결할 수 있습니다. 상태 시스템은 트레이스 데이터를 X-Ray로 보내고 X-Ray는 데이터를 처리하여 서비스 맵과 검색 가능한 트레이스 요약을 생성합니다.

상태 머신에 X-Ray를 활성화하면 X-Ray를 사용할 수 있는 모든 AWS 지역의 Step Functions에서 실행되는 요청을 추적할 수 있습니다. 이를 통해 전체 Step Functions 요청에 대한 상세 개요를 확인할 수 있습니다. Step Functions는 업스트림 서비스에서 트레이스 ID를 전달하지 않은 경우에도 상태 시스템 실행을 위해 트레이스를 X-Ray로 전송합니다. X-Ray 서비스 맵을 사용하여 X-Ray와 통합된 모든 AWS 서비스를 포함하여 요청의 지연 시간을 볼 수 있습니다. 또한 샘플링 규칙을 구성하여 지정한 기준에 따라 어떤 요청을 어떤 샘플링 속도로 기록할지를 X-Ray에 지시할 수도 있습니다.

상태 시스템에 X-Ray를 활성화하지 않고 업스트림 서비스에서 트레이스 ID를 전달하지 않으면 Step Functions는 상태 시스템 실행을 위해 트레이스를 X-Ray에 전송하지 않습니다. 하지만 업스트림 서비스에서 트레이스 ID를 전달하는 경우 Step Functions는 상태 시스템 실행을 위해 트레이스를 X-Ray로 전송합니다.

두 가지가 모두 지원되는 지역에서는 Step Functions와 AWS X-Ray 함께 사용할 수 있습니다. X-Ray 및 Step Functions의 리전 지원에 대한 자세한 내용은 Step Functions와 X-Ray 엔드포인트 및 할당량 페이지를 참조하세요.

> **X-Ray 및 Step Functions의 통합 할당량**
>
> 최대 7일 동안 데이터를 트레이스에 추가하고 X-Ray에서 트레이스 데이터를 저장하는 기간인 30일 이전의 트레이스 데이터를 쿼리할 수 있습니다. 트레이스에는 X-Ray 할당량이 적용됩니다. 다른 할당량 외에도 X-Ray는 Step Functions 상태 시스템의 최소 보장 트레이스 크기인 100KB를 제공합니다. 100KB 이상의 트레이스 데이터가 X-Ray에 제공되면 트레이스가 정지될 수 있습니다. 다른 X-Ray 할당량에 대한 자세한 내용은 X-Ray 엔드포인트 및 할당량 페이지의 서비스 할당량 섹션을 참조하세요.

> **Important**
>
> Step Functions는 Distributed Map 상태에서 시작된 하위 워크플로 실행의 X-Ray 트레이스를 지원하지 않습니다. 이러한 실행의 경우 트레이스 문서 크기 한도를 초과하기 쉽기 때문입니다.

주제

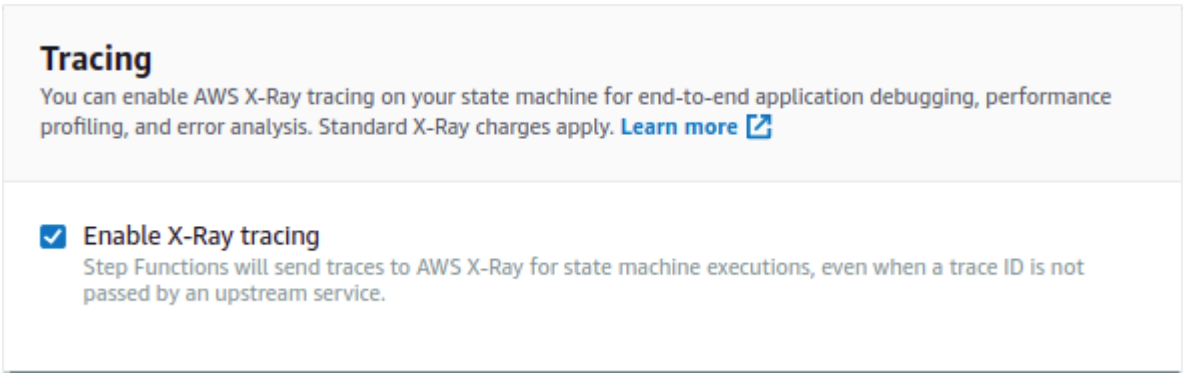
설정 및 구성

상태 시스템을 만들 때 X-Ray 트레이싱 활성화

세부 정보 지정 페이지에서 X-Ray 추적 활성화를 선택하여 새 상태 시스템을 만들 때 X-Ray 트레이싱을 활성화할 수 있습니다.

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 작성 방법 선택 페이지에서 적절한 옵션을 선택하여 상태 시스템을 만듭니다. 샘플 프로젝트 실행을 선택하면 상태 시스템을 만드는 동안 X-Ray 트레이싱을 활성화할 수 없으며 상태 시스템을 만든 후에 X-Ray 트레이싱을 활성화해야 합니다. 기존 상태 시스템에서 X-Ray를 활성화하는 방법에 대한 자세한 내용은 기존 상태 시스템에서 X-Ray 활성화를 참조하세요.

   다음을 선택합니다.
3. 세부 정보 지정 페이지에서 상태 시스템을 구성합니다.
4. X-Ray 추적 활성화를 선택합니다.



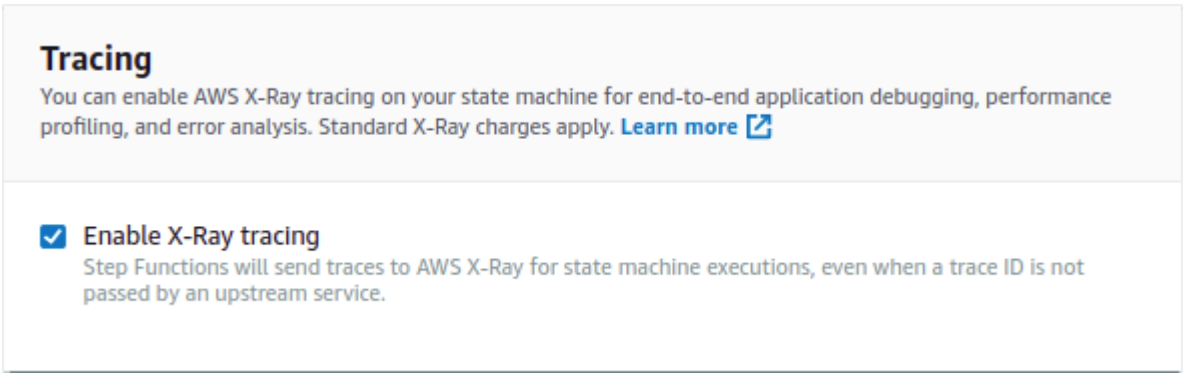
이제 Step Functions 상태 시스템에서 상태 시스템 실행을 위해 트레이스를 X-Ray로 전송합니다.

> **Note**
>
> 기존 IAM 역할을 사용하는 경우 X-Ray 쓰기가 허용되는지 확인해야 합니다. 필요한 권한에 대한 자세한 내용은 X-Ray에 대한 IAM 정책을 참조하세요.

기존 상태 시스템에서 X-Ray 활성화

기존 상태 시스템에서 X-Ray 활성화하기

1. Step Functions 콘솔에서 트레이싱을 활성화하려는 상태 시스템을 선택합니다.
2. 편집을 선택합니다.
3. X-Ray 추적 활성화를 선택합니다.



추가적으로 변경해야 할 수도 있다는 알림이 표시됩니다.

> **Note**
>
> 기존 상태 시스템에 X-Ray를 활성화할 때는 트레이스를 수행할 수 있도록 X-Ray에 충분한 권한을 부여하는 IAM 정책이 있는지 확인해야 합니다. 수동으로 추가하거나 생성할 수 있습니다. 자세한 내용은 다음에 대한 IAM 정책 AWS X-Ray에 대한 IAM 정책을 참조하세요.

4. (선택 사항) X-Ray 권한이 포함되도록 상태 시스템의 새 역할을 자동 생성합니다.
5. 저장을 선택합니다.

Step Function에 대한 X-Ray 트레이싱 구성

X-Ray Tracing이 활성화된 상태에서 상태 머신을 처음 실행하면 X-Ray Tracing의 기본 구성 값이 사용됩니다. AWS X-Ray 애플리케이션으로 전송되는 모든 요청에 대해 데이터를 수집하지는 않습니다. 대신 통계적으로 유의미한 요청 수에 대한 데이터를 수집합니다. 기본 값은 매초 최초 요청과 추가 요청의 5% 기록입니다. 초당 하나의 요청은 리저버입니다. 이는 서비스가 요청을 처리 중인 동안 하나 이상의 트레이스가 매초 기록되도록 합니다. 5퍼센트는 리저버 크기를 넘는 추가 요청이 샘플링되는 비율입니다.

시작할 때 서비스 요금이 발생하지 않도록 하려는 경우 기본 샘플링 비율은 보수적입니다. 기본 샘플링 규칙을 수정하고 서비스 또는 요청의 속성에 따라 샘플링을 적용하는 추가 규칙을 구성하도록 X-Ray를 구성할 수 있습니다.

예를 들어 샘플링을 비활성화하고 상태나 핸들 AWS 계정 또는 트랜잭션을 수정하는 호출에 대한 모든 요청을 추적하고 싶을 수 있습니다. 백그라운드 폴링, 상태 확인 또는 연결 유지 관리와 같은 대량 읽기 전용 직접 호출의 경우 낮은 비율로 샘플링하면서도 발생하는 문제를 관찰할 수 있을 만큼 충분한 데이터를 계속 확보할 수 있습니다.

상태 시스템에 대한 샘플링 규칙 구성하기

1. X-Ray 콘솔로 이동합니다.
2. 샘플링을 선택합니다.
3. 샘플링 규칙 생성을 선택하여 규칙을 생성합니다.

   규칙 이름을 선택하여 규칙을 편집합니다.

   규칙을 선택하고 작업메뉴를 사용하여 규칙을 삭제합니다.

기존 샘플링 규칙의 일부(예: 이름 및 우선순위)를 변경할 수 없습니다. 대신 기존 규칙을 추가 또는 복제하고 원하는 대로 변경한 다음 새 규칙을 사용합니다.

X-Ray 샘플링 규칙 및 다양한 파라미터를 구성하는 방법에 대한 자세한 내용은 X-Ray 콘솔에서 샘플링 규칙 구성을 참조하세요.

업스트림 서비스 통합

Express, 동기 및 표준 워크플로와 같은 Step Functions 워크플로의 실행을 업스트림 서비스와 통합하려면 `traceHeader`를 설정해야 합니다. 이 작업은 API Gateway에서 HTTP API를 사용하는 경우 자동으로 수행됩니다. 하지만 Lambda 함수 또는 SDK를 사용하는 경우에는 `StartExecution` 또는 `StartSyncExecution` API 직접 호출에서 직접 `traceHeader`를 설정해야 합니다.

`traceHeader`를 `\p{ASCII}#` 형식으로 지정해야 합니다. 또한 Step Functions에서 같은 트레이스 ID를 사용하도록 하려면 형식을 `Root={TRACE_ID};Sampled={1 or 0}`으로 지정해야 합니다. Lambda 함수를 사용하는 경우 `TRACE_ID`를 현재 세그먼트의 트레이스 ID로 바꾸고 샘플링 필드를 샘플링 모드가 true인 경우에는 `1`로, 샘플링 모드가 false인 경우에는 `0`으로 설정합니다. 트레이스 ID를 이 형식으로 제공하면 완전한 트레이스를 얻을 수 있습니다.

다음은 `traceHeader`를 지정하는 방법을 보여주기 위해 Python으로 작성된 예제입니다.

```
state_machine = config.get_string_paramter("STATE_MACHINE_ARN")
    if (xray_recorder.current_subsegment() is not None and
        xray_recorder.current_subsegment().sampled) :
        trace_id = "Root={};Sampled=1".format(
            xray_recorder.current_subsegment().trace_id
        )
    else:
        trace_id = "Root=not enabled;Sampled=0"
    LOGGER.info("trace %s", trace_id)

    # execute it
    response = states.start_sync_execution(
        stateMachineArn=state_machine,
        input=event['body'],
        name=context.aws_request_id,
        traceHeader=trace_id
    )
    LOGGER.info(response)
```

개념

X-Ray 콘솔

AWS X-Ray 콘솔을 사용하면 애플리케이션이 제공하는 요청에 대한 서비스 맵과 추적을 볼 수 있습니다. X-Ray가 상태 시스템에 활성화되면 콘솔에 액세스하여 X-Ray에서 수집한 세부 정보를 볼 수 있습니다.

상태 시스템 실행을 위해 X-Ray 콘솔에 액세스하는 방법은 X-Ray 콘솔 보기를 참조하세요.

X-Ray 콘솔에 대한 자세한 내용은 X-Ray 콘솔 설명서를 참조하세요.

세그먼트, 하위 세그먼트 및 트레이스

세그먼트는 상태 시스템에 대한 요청 정보를 기록합니다. 여기에는 상태 시스템에서 수행하는 작업과 같은 정보가 포함되며 다운스트림 호출에 대한 정보가 포함된 하위 세그먼트도 포함될 수 있습니다.

트레이스는 단일 요청에 의해 생성된 모든 세그먼트를 수집합니다.

샘플링

효율적인 트레이스를 보장하고 애플리케이션에서 처리하는 요청의 대표적 샘플을 제공하기 위해 X-Ray는 샘플링 알고리즘을 적용하여 트레이스되는 요청을 결정합니다. 샘플링 규칙을 편집하여 이를 변경할 수 있습니다.

지표

상태 시스템의 경우 X-Ray는 간접 호출 시간, 상태 전환 시간, Step Functions의 전체 실행 시간 및 이 실행 시간의 차이를 측정합니다. X-Ray 콘솔을 통해 이 정보에 액세스할 수 있습니다.

분석

AWS X-Ray Analytics 콘솔은 추적 데이터를 해석하기 위한 대화형 도구입니다. 현재 트레이스 집합과 연결된 지표 및 필드의 그래프와 패널을 클릭하면 더욱 세부적인 필터로 활성 데이터 세트를 구체화할 수 있습니다. 이를 통해 상태 시스템이 수행되는 방식을 분석하고 성능 문제를 신속하게 찾고 식별할 수 있습니다.

X-Ray 분석에 대한 자세한 내용은 Analytics 콘솔과의 상호 작용을 참조하십시오. AWS X-Ray

Step Functions 서비스 통합 및 X-Ray

Step Functions와 통합되는 일부 AWS 서비스는 요청에 추적 헤더를 추가하거나, X-Ray 데몬을 실행하거나, 샘플링 결정을 내리고 추적 데이터를 X-Ray에 업로드하는 AWS X-Ray 방식으로 통합을 제공합니다. 다른 것들은 SDK를 사용하여 계측해야 합니다. AWS X-Ray 몇몇 서비스는 아직 X-Ray 통합을 지원하지 않습니다. Step Functions와 서비스 통합을 사용할 때 완전한 트레이스 데이터를 제공하려면 X-Ray 통합이 필요합니다.

기본 X-Ray 지원

기본 X-Ray 지원과 서비스 통합에는 다음이 포함됩니다.

- Amazon Simple Notification Service
- Amazon Simple Queue Service
- AWS Lambda
- AWS Step Functions

계측 필요

X-Ray 기기가 필요한 서비스 통합:

- Amazon Elastic Container Service
- AWS Batch
- AWS Fargate

클라이언트 측 트레이스 전용

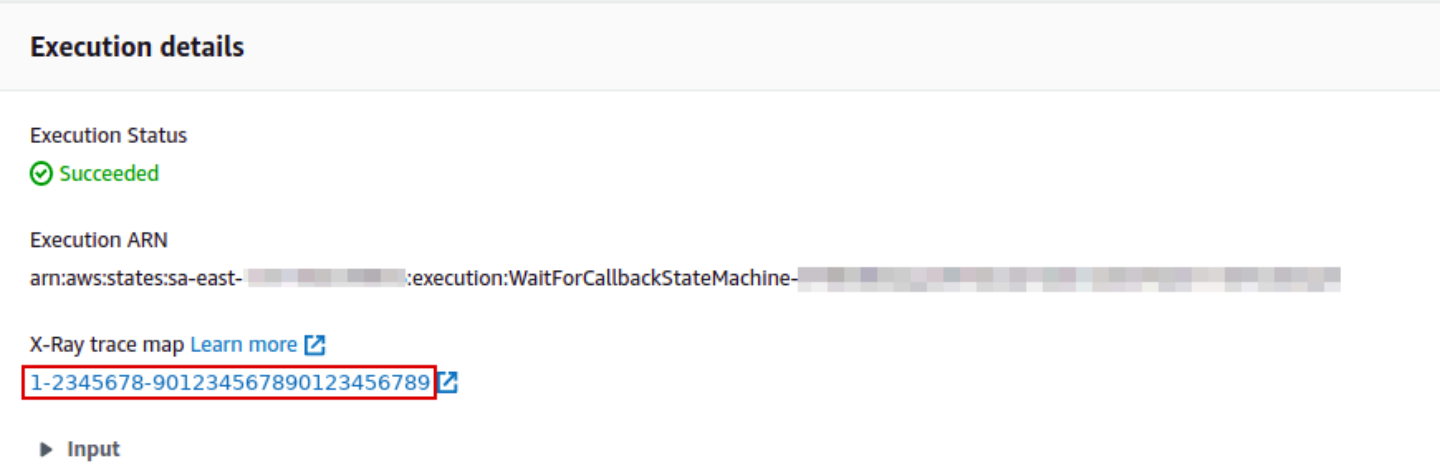
다른 서비스 통합은 X-Ray 트레이스를 지원하지 않습니다. 하지만 클라이언트 측 트레이스를 계속 수집할 수 있습니다.

- Amazon DynamoDB
- Amazon EMR
- 아마존 SageMaker
- AWS CodeBuild
- AWS Glue

X-Ray 콘솔 보기

X-Ray는 서비스로부터 데이터를 세그먼트로 수신합니다. X-Ray는 공통 요청이 있는 세그먼트를 트레이스로 그룹화합니다. X-Ray는 트레이스를 처리하여 애플리케이션을 시각적으로 표현하는 서비스 그래프를 생성합니다.

상태 시스템 실행을 시작한 후 실행 세부 정보 섹션에서 X-Ray 트레이스 맵 링크를 선택하면 X-Ray 트레이스를 볼 수 있습니다.



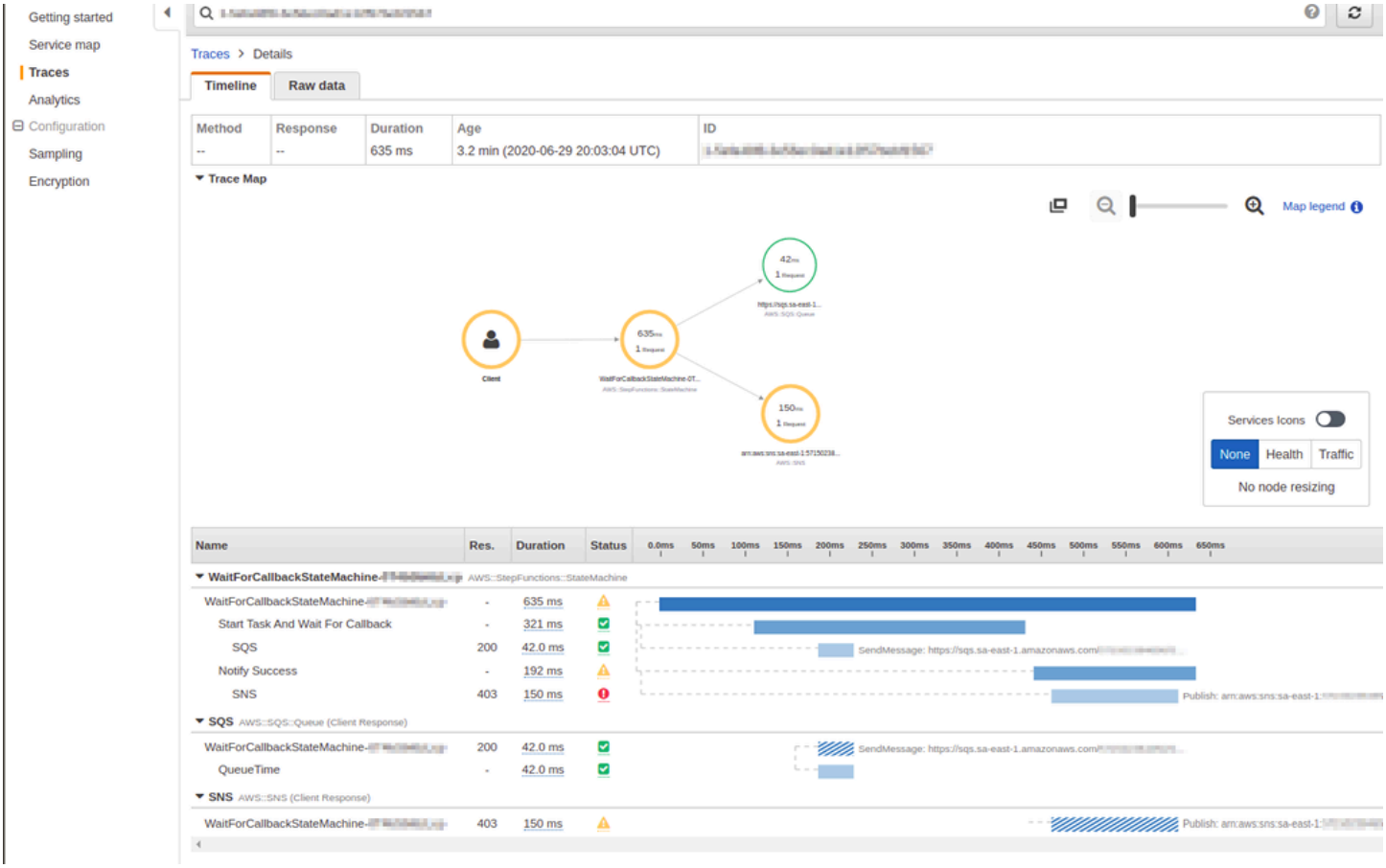
상태 시스템에 X-Ray를 활성화한 후에는 X-Ray 콘솔에서 해당 실행에 대한 트레이싱 정보를 볼 수 있습니다.

Step Function에 대한 X-Ray 트레이싱 정보 보기

다음 단계에서는 X-Ray를 활성화하고 실행을 실행한 후 콘솔에 표시되는 정보의 종류를 보여줍니다. 콜백 패턴 예제(Amazon SQS, Amazon SNS, Lamda) 샘플 프로젝트의 X-Ray 트레이스가 표시됩니다.

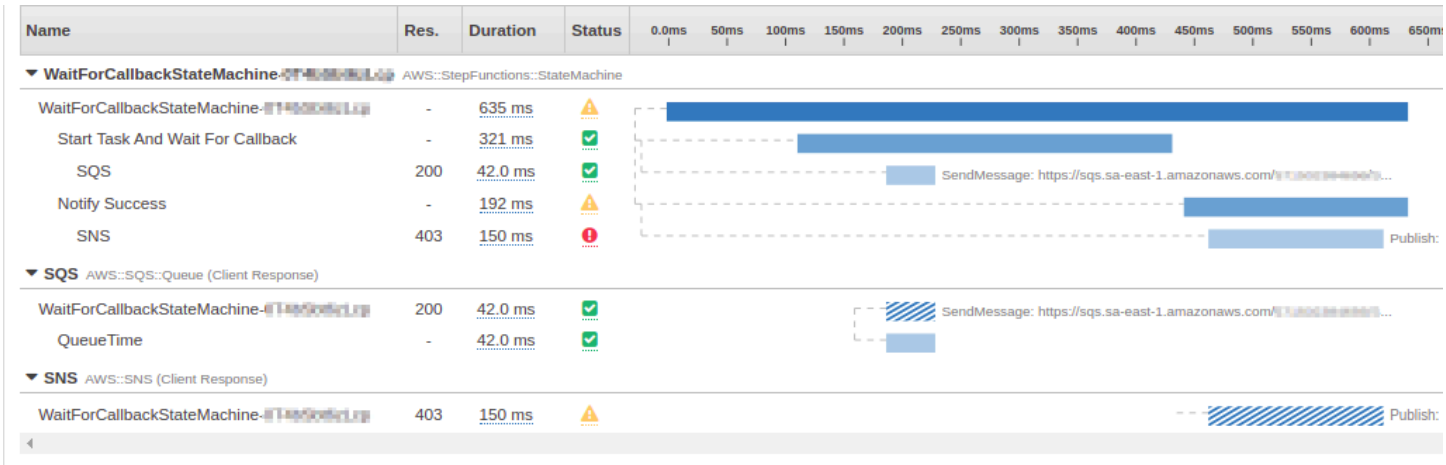
트레이스

실행이 완료되면 X-Ray 콘솔로 이동합니다. 여기서 X-Ray 트레이스 페이지가 표시됩니다. 이 페이지는 서비스 맵 개요뿐만 아니라 상태 시스템의 트레이스 및 세그먼트 정보도 표시됩니다.

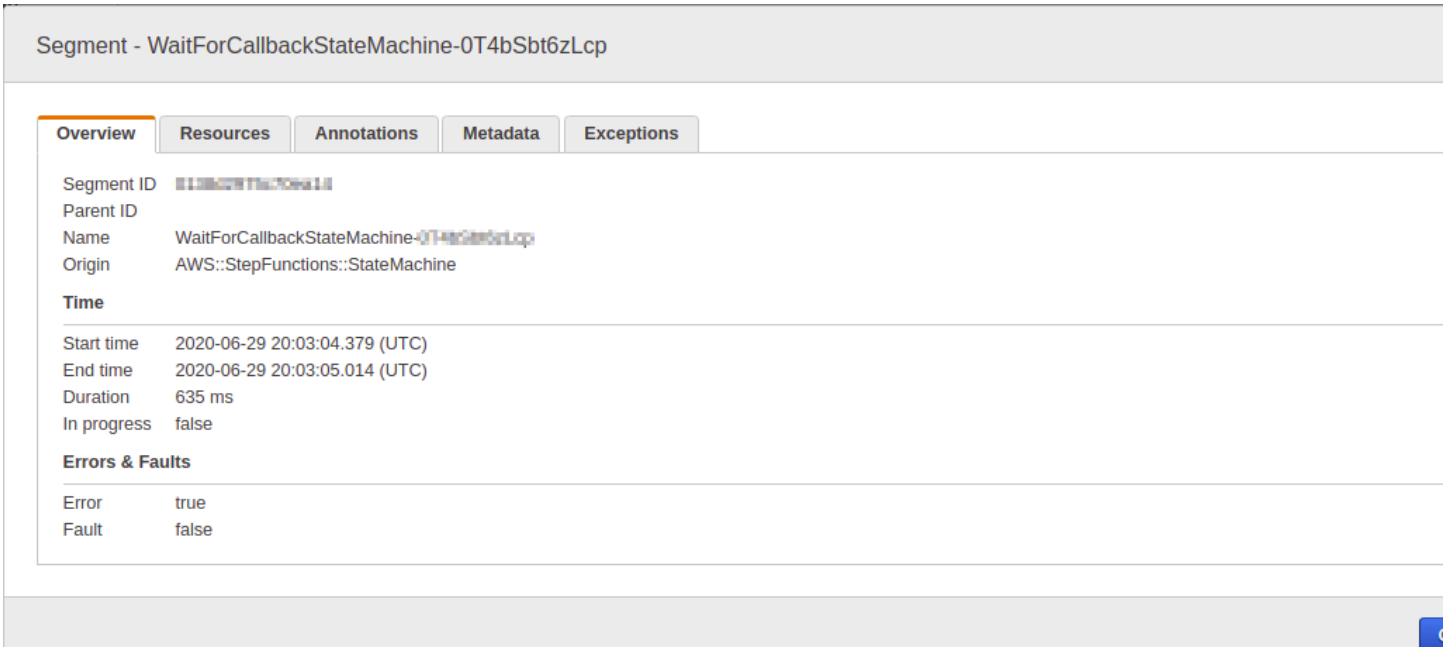


서비스 맵

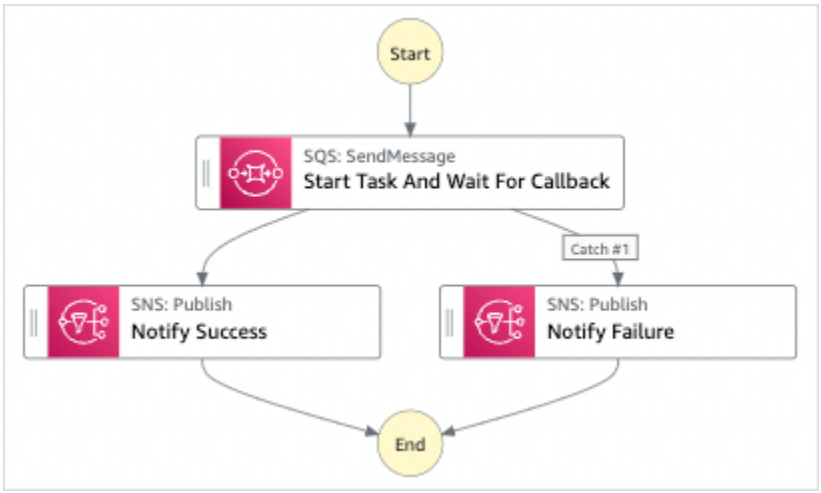
X-Ray 콘솔의 서비스 맵을 사용하면 지연 시간이 긴 연결과 같은 오류가 발생한 서비스를 식별하고 실패한 요청의 트레이스를 확인할 수 있습니다.



트레이스 맵에서 서비스 노드를 선택하여 해당 노드에 대한 요청을 보거나 두 노드 간의 엣지를 선택하여 해당 연결로 이동하는 요청을 볼 수 있습니다. 여기에서는 `WaitForCallBack` 노드가 선택되었으므로 해당 실행과 응답 상태에 대한 추가 정보를 볼 수 있습니다.



X-Ray 서비스 맵이 상태 시스템과 상관 관계를 지정하는 방법을 확인할 수 있습니다. X-Ray를 지원하는 경우 Step Functions에서 직접적으로 호출하는 각 서비스 통합에 대한 서비스 맵 노드가 있습니다.



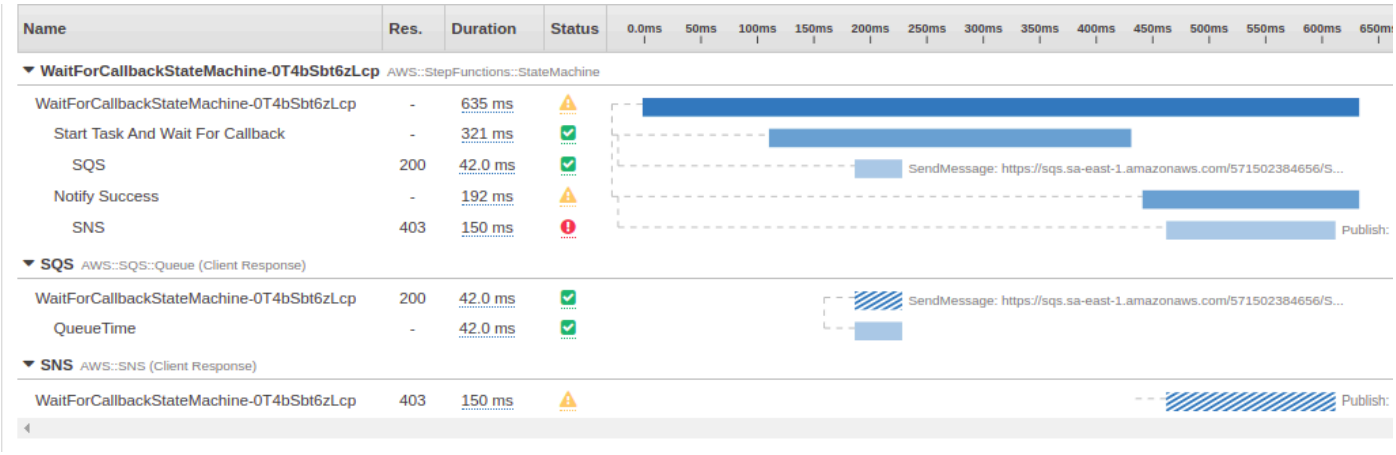
세그먼트 및 하위 세그먼트

트레이스는 단일 요청에 의해 생성된 세그먼트 컬렉션입니다. 각 세그먼트는 리소스 이름, 요청의 세부 정보 및 완료된 작업의 세부 정보를 제공합니다. 트레이스 페이지에서 세그먼트를 확인할 수 있으며 이 페이지를 확장하면 해당 하위 세그먼트도 확인할 수 있습니다. 세그먼트나 하위 세그먼트를 선택하여 해당 세그먼트에 대한 자세한 정보를 볼 수 있습니다.

각 탭을 선택하여 세그먼트 및 하위 세그먼트에 대한 정보가 표시되는 방법을 확인하세요.

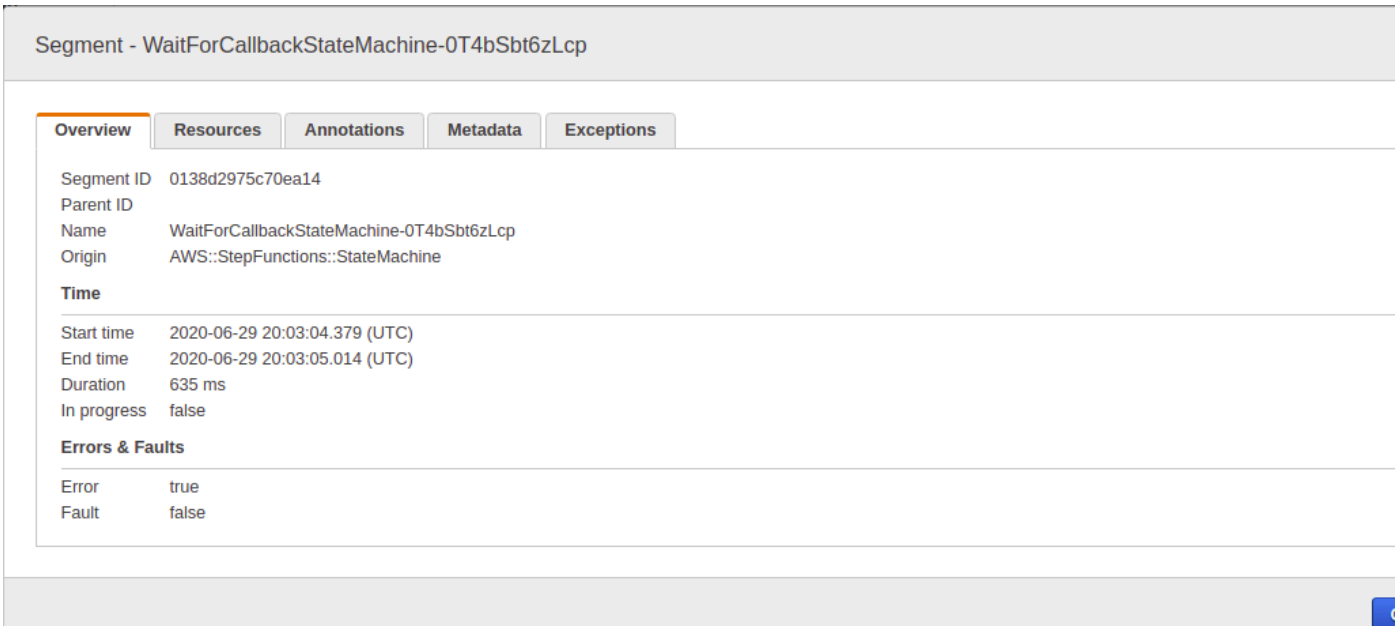
Overview of Segments

이 상태 시스템의 세그먼트 및 하위 세그먼트 개요입니다. 서비스 맵에는 노드마다 다른 세그먼트가 있습니다.



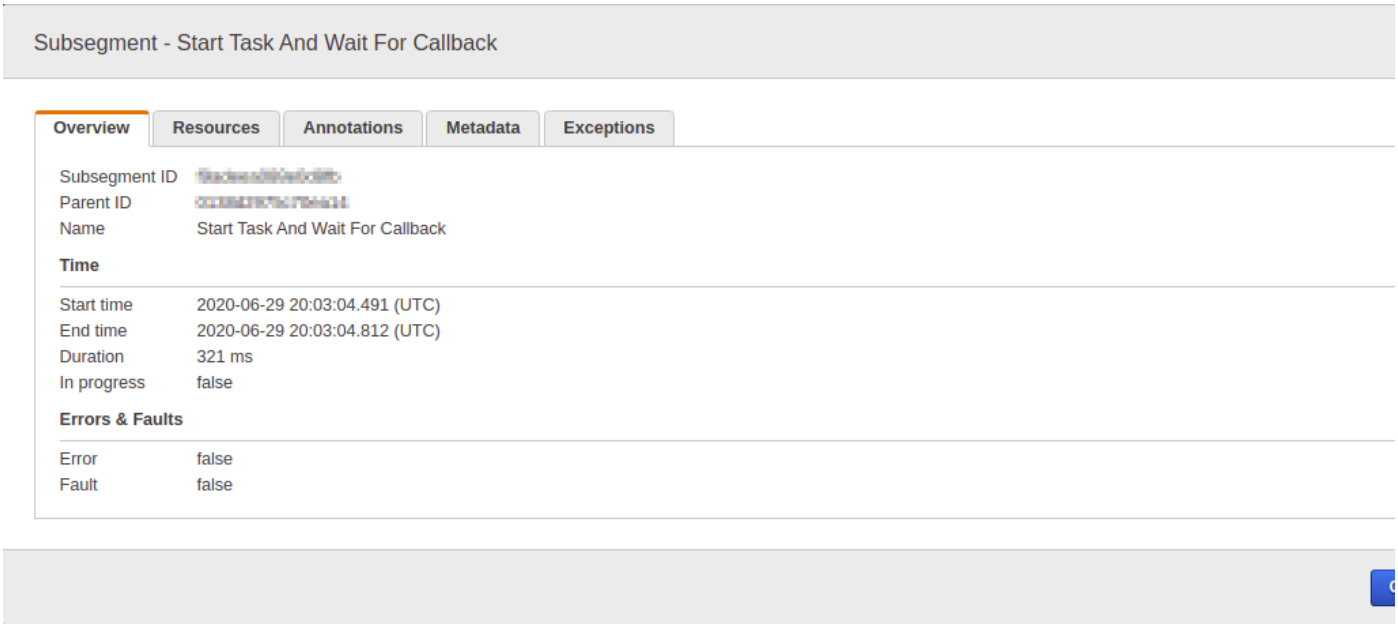
View segment detail

세그먼트를 선택하면 리소스 이름, 요청의 세부 정보 및 완료된 작업의 세부 정보가 제공됩니다.



View subsegment detail

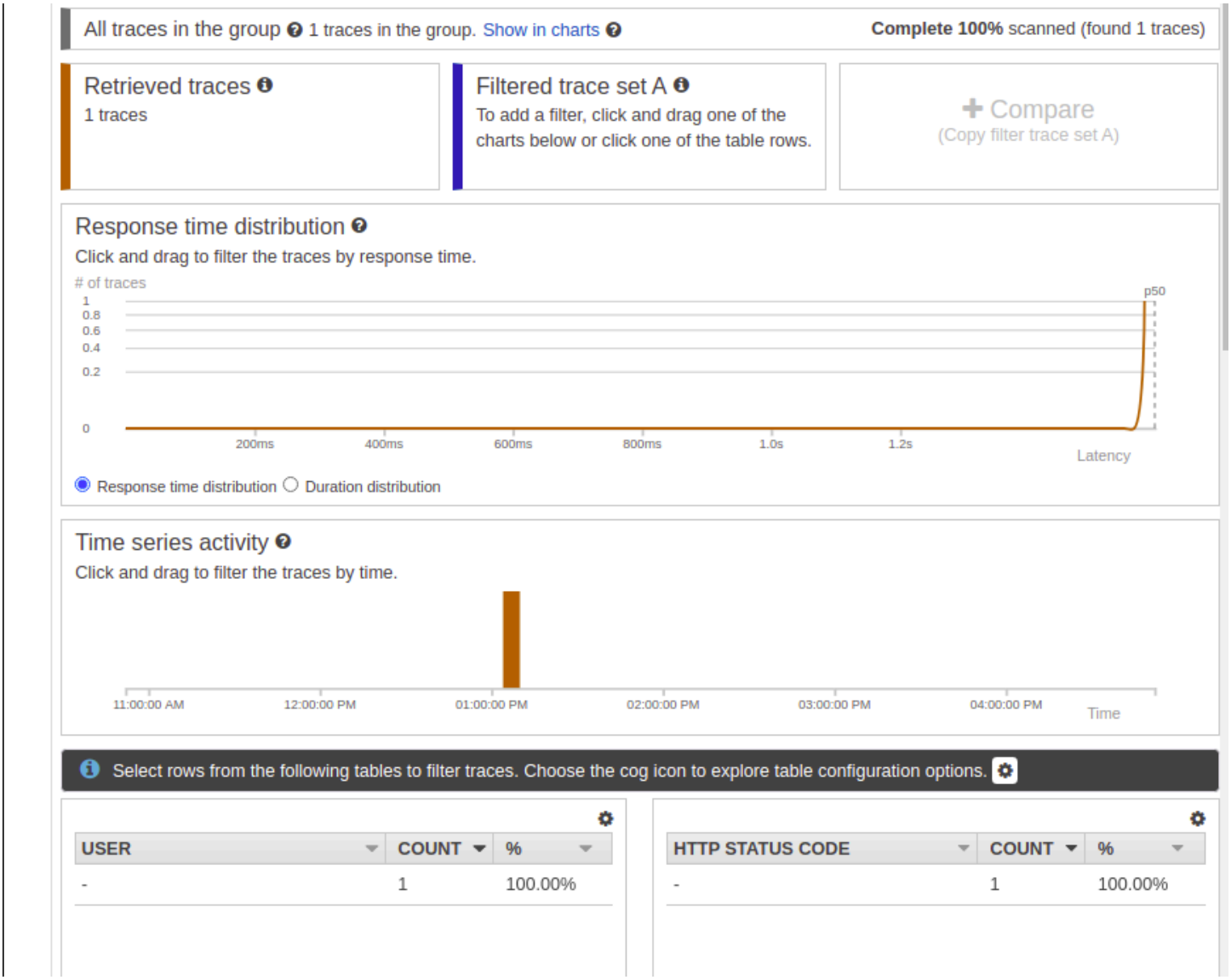
세그먼트는 완료된 작업에 대한 데이터를 하위 세그먼트로 구분할 수 있습니다. 하위 세그먼트를 선택하면 보다 자세한 타이밍 정보와 세부 정보를 볼 수 있습니다. 하위 세그먼트에는 AWS 서비스 호출, 외부 HTTP API 또는 SQL 데이터베이스에 대한 추가 세부 정보가 포함될 수 있습니다.



분석

AWS X-Ray Analytics 콘솔은 추적 데이터를 해석하기 위한 대화형 도구입니다. 이 콘솔을 사용하면 상태 시스템이 수행되는 방식을 더욱 쉽게 이해할 수 있습니다. 콘솔에서는 대화형 응답 시간 및 시계열 그래프를 통해 트레이스를 탐색, 분석 및 시각화할 수 있습니다. 이를 통해 성능 및 지연 시간 문제를 빠르게 찾을 수 있습니다.

현재 트레이스 집합과 연결된 지표 및 필드의 그래프와 패널을 클릭하면 더욱 세부적인 필터로 활성 데이터 세트를 구체화할 수 있습니다.

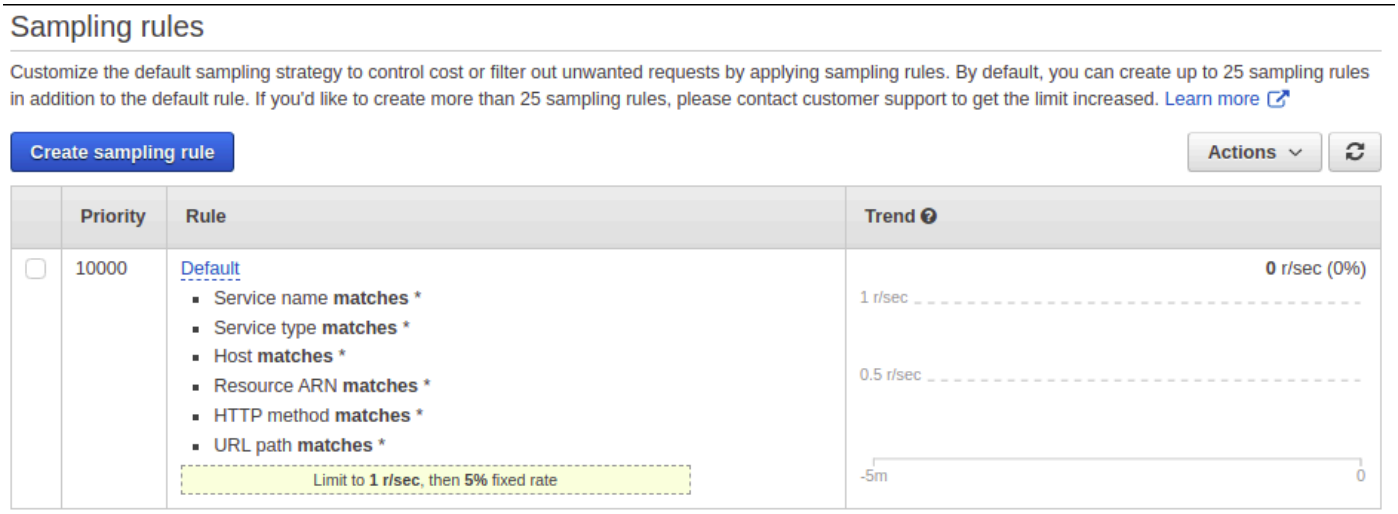


구성

X-Ray 콘솔에서 샘플링 및 암호화 옵션을 구성할 수 있습니다.

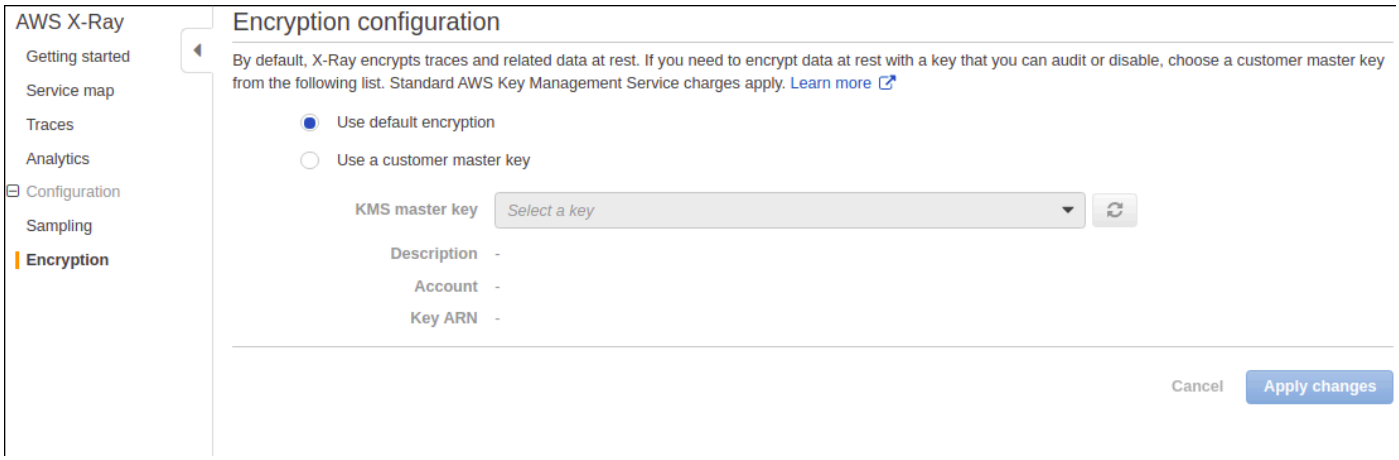
Sampling

샘플링을 선택하여 샘플링 속도와 구성에 대한 세부 정보를 봅니다. 샘플링 규칙을 변경하여 기록할 데이터 양을 제어하고 특정 요구 사항에 맞게 샘플링 동작을 수정할 수 있습니다.



Encryption

암호화를 선택하여 암호화 설정을 수정합니다. X-Ray에서 트레이스와 저장된 날짜를 암호화하는 기본 설정을 사용하거나 필요한 경우 고객 마스터 키를 선택할 수 있습니다. 후자의 경우 표준 AWS KMS 요금이 적용됩니다.



트레이스 맵 또는 서비스 맵에 데이터가 없으면 어떻게 되나요?

X-Ray를 활성화했지만 X-Ray 콘솔에 데이터가 표시되지 않으면 다음을 확인합니다.

- IAM 역할은 X-Ray에 기록할 수 있도록 올바르게 설정되었습니다.
- 샘플링 규칙에서 데이터 샘플링을 허용합니다.
- 새로 생성되거나 수정된 IAM 역할이 적용되기까지 짧은 지연이 발생할 수 있으므로 몇 분 후에 트레이스 또는 서비스 맵을 다시 확인합니다.

- X-Ray Trace 패널에 Data Not Found (데이터 없음) 가 AWS Security Token Service 표시되면 IAM 계정 설정을 확인하고 원하는 지역에 활성화되어 있는지 확인하십시오. 자세한 내용은 IAM 사용 설명서의 활성화 및 비활성화를 AWS STS 참조하십시오. AWS 리전

AWS 사용자 알림와 함께 AWS Step Functions 사용

AWS 사용자 알림을 사용하여 AWS Step Functions 이벤트에 대한 알림을 받을 전송 채널을 설정할 수 있습니다. 이벤트가 지정한 규칙과 일치하면 알림을 받습니다. 이메일, AWS Chatbot 채팅 알림 또는 AWS Console Mobile Application 푸시 알림을 비롯한 여러 채널을 통해 이벤트에 대한 알림을 받을 수 있습니다. 콘솔 알림 센터에서도 알림을 볼 수 있습니다. 사용자 알림은 집계를 지원하므로 특정 이벤트 중에 받는 알림 수를 줄일 수 있습니다.

보안: AWS Step Functions

이 섹션에서는 AWS Step Functions 보안 및 인증에 대한 정보를 제공합니다.

Step Functions는 IAM을 사용하여 다른 AWS 서비스 및 리소스에 대한 액세스를 제어합니다. IAM의 작동 방식에 대한 개요는 IAM 사용 설명서의 액세스 관리 개요를 참조하세요. 보안 인증 정보에 대한 개요는 Amazon Web Services 일반 참조의 AWS 보안 인증 정보를 참조하세요.

데이터 보호: AWS Step Functions

AWS 공동 책임 모델 의 데이터 보호에 적용됩니다 AWS Step Functions. 이 모델에 설명된 대로 AWS는 모든 데이터를 실행하는 글로벌 인프라를 보호하는 역할을 AWS 클라우드합니다. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하십시오.

데이터 보호를 위해 AWS 계정 자격 증명을 보호하고 AWS IAM Identity Center OR AWS Identity and Access Management (IAM) 을 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 멀티 팩터 인증 설정(MFA)을 사용하세요.
- SSL/TLS를 사용하여 리소스와 통신하세요. AWS TLS 1.2는 필수이며 TLS 1.3를 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다. AWS CloudTrail
- 포함된 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용하십시오 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 AWS 통해 액세스할 때 FIPS 140-2로 검증된 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용하십시오. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-2를 참조하십시오.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 콘솔 AWS CLI, API 또는 AWS 서비스 AWS SDK를 사용하여 Step Functions나 기타 기능을 사용하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍

스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 보안 인증 정보를 URL에 포함시켜서는 안 됩니다.

에서의 암호화 AWS Step Functions

유휴 데이터 암호화

Step Functions는 항상 저장 데이터를 암호화합니다. 저장된 AWS Step Functions 데이터는 투명한 서버 측 암호화를 사용하여 암호화됩니다. 이를 사용하면 중요한 데이터 보호와 관련된 운영 부담 및 복잡성을 줄일 수 있습니다. 저장 시 암호화를 사용하면 암호화 규정 준수 및 규제 요구 사항을 충족하는 보안에 민감한 애플리케이션을 구축할 수 있습니다.

전송 중 암호화

Step Functions는 서비스와 다른 통합 AWS 서비스 간에 전송 중인 데이터를 암호화합니다(다른 서비스와 AWS Step Functions 함께 사용 참조). Step Functions 및 통합 서비스 간에 전달되는 모든 데이터는 전송 계층 보안(TLS)을 통해 암호화됩니다.

AWS Step Functions의 자격 증명 및 액세스 관리

에 액세스하려면 AWS Step Functions 요청을 인증하는 데 사용할 AWS 수 있는 자격 증명이 필요합니다. 이러한 자격 증명에는 다른 AWS 리소스에서 이벤트 데이터를 검색하는 등 AWS 리소스에 액세스할 수 있는 권한이 있어야 합니다. 다음 섹션에서는 리소스에 액세스할 수 있는지 대상을 제어하여 리소스를 보호할 수 있도록 AWS Identity and Access Management (IAM) 및 Step Functions를 사용하는 방법에 대한 세부 정보를 제공합니다.

AWS Identity and Access Management (IAM) 은 관리자가 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 AWS 도와줍니다. IAM 관리자는 리소스를 사용할 수 있는 인증 (로그인) 및 권한 부여 (권한 보유) 를 받을 수 있는 사용자를 제어합니다. Step Functions IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

고객

사용하는 방식 AWS Identity and Access Management (IAM) 은 수행하는 작업에 따라 다릅니다. Step Functions

서비스 사용자 - Step Functions 서비스를 사용하여 작업을 수행하는 경우 관리자가 필요한 자격 증명과 권한을 제공합니다. 더 많은 Step Functions 기능을 사용하여 작업을 수행함에 따라 추가 권한이 필

요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. Step Functions의 기능에 액세스할 수 없는 경우 ID 및 액세스 문제 해결 AWS Step Functions을 참조하세요.

서비스 관리자 — 회사에서 Step Functions 리소스를 담당하는 경우 전체 액세스 권한이 있을 수 Step Functions있습니다. 서비스 사용자가 액세스해야 하는 Step Functions 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 IAM을 어떻게 사용할 수 있는지 자세히 Step Functions알아보려면 을 참조하십시오IAM과의 AWS Step Functions 작동 방식.

IAM 관리자 - IAM 관리자라면 Step Functions에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 Step Functions ID 기반 정책의 예를 보려면 을 참조하십시오. 아이덴티티 기반 정책 예시 AWS Step Functions

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 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 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 서명을 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) 사용을 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 작업을 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 Service페더레이션 ID에 AWS 계정액세스하면 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 교체를 참조하십시오.

IAM 그룹은 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 수임할 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 역할은 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 사용을 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 세트를 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 서비스경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 계정 간 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하십시오.
- 서비스 간 액세스 — 일부는 다른 기능을 사용합니다. AWS 서비스 AWS 서비스예를 들어 서비스에서 직접적 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 세션을 참조하세요.

  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하십시오.
  - 서비스 연결 역할 — 서비스 연결 역할은 에 연결된 서비스 역할의 한 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.
- 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 부여를 참조하십시오.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용 설명서의 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 개요를 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 생성을 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 선택을 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 지정해야 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) 개요를 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 엔터티(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 엔터티에 대한 권한 경계를 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 방식을 참조하십시오.
- 세션 정책 - 세션 정책은 역할 또는 페더레이션 사용자에 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 정책을 참조하십시오.

여러 정책 타입

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련되어 있을 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하십시오.

액세스 통제

요청을 인증하는 데 유효한 보안 인증 정보가 있더라도 권한이 없으면 Step Functions 리소스를 만들거나 액세스할 수 없습니다. 예를 들어, Step Functions 규칙과 관련된 Amazon Simple Notification Service (Amazon SNS) 및 Amazon Simple Queue 서비스 (Amazon SQS) 대상을 AWS Lambda호출할 권한이 있어야 합니다.

다음 섹션에서는 Step Functions에 대한 권한을 관리하는 방법을 설명합니다.

- 상태 시스템을 위한 IAM 역할 만들기
- 관리자가 아닌 사용자에 대한 세부 IAM 권한 만들기
- Step Functions용 Amazon VPC 엔드포인트
- 통합 서비스용 IAM 정책

- Distributed Map 상태를 사용하기 위한 IAM 정책

에 대한 정책 조치 Step Functions

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 `Action`요소는 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.

Step Functions 작업 목록을 보려면 서비스 권한 부여 AWS Step Functions참조의 정의된 리소스를 참조하십시오.

정책 조치는 조치 앞에 다음 접두사를 Step Functions 사용합니다.

```
states
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "states:action1",
      "states:action2"
         ]
```

Step Functions ID 기반 정책의 예를 보려면 을 참조하십시오. 아이덴티티 기반 정책 예시 AWS Step Functions

에 대한 정책 리소스 Step Functions

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": "*"
```

Step Functions 리소스 유형 및 해당 ARN 목록을 보려면 서비스 권한 부여 AWS Step Functions 참조에 정의된 작업을 참조하십시오. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 AWS Step Functions에서 정의한 리소스를 참조하세요.

Step Functions ID 기반 정책의 예를 보려면 을 참조하십시오. 아이덴티티 기반 정책 예시 AWS Step Functions

에 대한 정책 조건 키 Step Functions

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 연산자를 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 경우 는 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 태그를 참조하십시오.

AWS 글로벌 조건 키 및 서비스별 조건 키를 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 AWS 설명서의 글로벌 조건 컨텍스트 키를 참조하십시오.

Step Functions 조건 키 목록을 보려면 서비스 권한 부여 참조의 조건 키를 참조하십시오 AWS Step Functions. 조건 키를 사용할 수 있는 작업 및 리소스를 알아보려면 리소스 정의 기준을 참조하세요 AWS Step Functions.

Step Functions ID 기반 정책의 예를 보려면 을 참조하십시오. 아이덴티티 기반 정책 예시 AWS Step Functions

내 ACL Step Functions

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 포함 Step Functions

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 엔티티 (사용자 또는 역할) 및 여러 AWS 리소스에 태그를 첨부할 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 요소에 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) 사용을 참조하십시오.

다음과 같은 임시 자격 증명 사용 Step Functions

임시 보안 인증 지원	예

임시 자격 증명을 사용하여 로그인하면 일부 자격 증명이 AWS 서비스 작동하지 않습니다. 임시 자격 증명을 사용하는 방법을 AWS 서비스 비롯한 추가 정보는 IAM 사용 설명서의 IAM과AWS 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 다음 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.

서비스 간 사용자 권한: Step Functions

전달 액세스 세션(FAS) 지원	예

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 세션을 참조하세요.

Step Functions의 서비스 역할

서비스 역할 지원	예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하십시오.

> **Warning**
>
> 서비스 역할의 권한을 변경하면 Step Functions 기능이 중단될 수 있습니다. 서비스 역할을 편집하기 위한 지침이 Step Functions 제공되는 경우에만 서비스 역할을 편집하십시오.

서비스 연결 역할은 다음과 같습니다. Step Functions

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 서비스를 참조하십시오. 서비스 연결 역할 열에서 Yes(이)가 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 Yes(네) 링크를 선택합니다.

IAM과의 AWS Step Functions 작동 방식

IAM을 사용하여 액세스를 Step Functions관리하기 전에 어떤 IAM 기능과 함께 사용할 수 있는지 알아보세요. Step Functions

함께 사용할 수 있는 IAM 기능 AWS Step Functions

IAM 특성	Step Functions 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예

IAM 특성	Step Functions 지원
정책 리소스	예
정책 조건 키(서비스별)	예
ACLs	아니요
ABAC(정책 내 태그)	부분
임시 보안 인증	예
보안 주체 권한	예
서비스 역할	예
서비스 연결 역할	아니요

Step Functions 및 기타 AWS 서비스가 대부분의 IAM 기능과 어떻게 작동하는지 자세히 알아보려면 IAM 사용 설명서의 IAM과 함께 작동하는AWS 서비스를 참조하십시오.

아이덴티티 기반 정책 예시 AWS Step Functions

기본적으로 사용자 및 역할에는 Step Functions 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 AWS API를 사용하여 작업을 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 ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하십시오.

각 리소스 유형의 ARN 형식을 비롯하여 에서 정의한 Step Functions작업 및 리소스 유형에 대한 자세한 내용은 서비스 권한 부여 참조의 작업, 리소스 및 조건 키를 참조하십시오. AWS Step Functions

주제

- 정책 모범 사례
- Step Functions 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

정책 모범 사례

ID 기반 정책은 누군가가 계정에서 Step Functions 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부를 결정합니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하여 최소 권한 권한으로 이동 — 사용자와 워크로드에 권한을 부여하려면 여러 일반적인 사용 사례에 권한을 부여하는 AWS 관리형 정책을 사용하세요. 해당 내용은 에서 사용할 수 있습니다. AWS 계정사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 더 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책 또는 직무에 대한AWS 관리형 정책을 참조하십시오.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 태스크를 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 사용 설명서의 IAM의 정책 및 권한을 참조하십시오.
- 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. 예를 AWS 서비스들어 특정 작업을 통해 서비스 작업을 사용하는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 AWS CloudFormation있습니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하십시오.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer 정책 검증을 참조하십시오.
- 멀티 팩터 인증 (MFA) 필요 - IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 AWS 계정 MFA를 활성화하십시오. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA 보호 API 액세스 구성을 참조하십시오.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

Step Functions 콘솔 사용

AWS Step Functions 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한을 통해 내 Step Functions 리소스의 세부 정보를 나열하고 볼 수 있어야 AWS 계정합니다. 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 필요는 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 Step Functions 콘솔을 계속 사용할 수 있도록 하려면 엔티티에 Step Functions *ConsoleAccess* 또는 *ReadOnly* AWS 관리형 정책도 연결하세요. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하십시오.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 AWS CLI 또는 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": "*"
        }
    ]
}
```

ID 기반 정책은 다음과 같습니다. Step Functions

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 생성을 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 참조를 참조하십시오.

다음에 대한 ID 기반 정책 예제 Step Functions

Step Functions ID 기반 정책의 예를 보려면 을 참조하십시오. 아이덴티티 기반 정책 예시 AWS Step Functions

내 리소스 기반 정책 Step Functions

리소스 기반 정책 지원	아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 문서입니다. 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 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 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 액세스를 참조하십시오.

AWS 관리형 정책 대상 AWS Step Functions

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 정책을 정의하여 권한을 줄이는 것이 좋습니다.

관리형 정책에 정의된 권한은 변경할 수 없습니다. AWS AWS 관리형 정책에 정의된 권한을 업데이트하는 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 사용 설명서의 AWS 관리형 정책을 참조하십시오.

AWS 관리형 정책: AWSStepFunctionsConsoleFullAccess

`AWSStepFunctionsConsoleFullAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 ##### Step Functions 콘솔을 사용할 수 있는 사용자 액세스를 허용하는 권한을 부여합니다. 완전한 콘솔 환경을 위해 사용자는 서비스에서 위임할 수 있는 다른 IAM 역할에 대한 iam:PassRole 권한도 필요할 수 있습니다.

AWS 관리형 정책: AWSStepFunctionsReadOnlyAccess

`AWSStepFunctionsReadOnlyAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 사용자 또는 역할이 상태 머신, 활동, 실행, 활동, 태그 MapRuns, 상태 머신 별칭 및 버전을 나열하고 설명할 수 있는 *## ##* 권한을 부여합니다. 또한 이 정책은 사용자가 제공하는 상태 시스템 정의의 구문을 검사할 수 있는 권한을 부여합니다.

AWS 관리형 정책: AWSStepFunctionsFullAccess

`AWSStepFunctionsFullAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다.

이 정책은 사용자 또는 역할에 Step Functions API를 사용할 수 있는 *##* 권한을 부여합니다. 전체 액세스 권한을 얻으려면 사용자에게 서비스에서 위임할 수 있는 하나 이상의 IAM 역할에 대한 *iam:PassRole* 권한이 있어야 합니다.

Step Functions 관리형 정책 업데이트 AWS

이 서비스가 이러한 변경 사항을 추적하기 시작한 Step Functions 이후의 AWS 관리형 정책 업데이트에 대한 세부 정보를 볼 수 있습니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 Step Functions 문서 기록 페이지에서 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
AWSStepFunctionsReadOnlyAccess - 기존 정책에 대한 업데이트	Step Functions `states:ValidateStateMachineDefinition` API 작업을 호출하여 제공하는 상태 시스템 정의의 구문을 확인할 수 있는 새 권한이 추가되었습니다.	2024년 4월 25일
AWSStepFunctionsReadOnlyAccess-기존 정책 업데이트	Step Functions 태그 (ListTagsForResource), 분산 맵 (ListMap실행,), 버전 및 별칭 (,, DescribeMapRun) 과 관련된 데이터를 나열하고 읽을 수 있는 새 권한이 추가되었습니다. DescribeState MachineAlias ListStateMachineAliases ListStateMachineVersions	2024년 4월 2일

변경 사항	설명	날짜
Step Functions 변경 사항 추적 시작	Step Functions AWS 관리형 정책의 변경 사항 추적을 시작했습니다.	2024년 4월 2일

상태 시스템을 위한 IAM 역할 만들기

AWS Step Functions 코드를 실행하고 AWS 리소스에 액세스할 수 있습니다 (예: AWS Lambda 함수 호출). 보안 유지를 위해 사용자는 IAM 역할을 사용하여 이러한 리소스에 대한 Step Functions 액세스 권한을 부여해야 합니다.

이 가이드의 내용을 통해 상태 머신을 생성한 AWS 지역에 유효한 자동 생성된 IAM 역할을 활용할 수 있습니다. Step Functions 자습서 하지만 상태 시스템에 대한 자체 IAM 역할을 만들 수 있습니다.

사용할 상태 시스템에 대한 IAM 정책을 만들 때 정책에 상태 시스템에게 위임할 권한이 포함되어야 합니다. 기존 AWS 관리형 정책을 예로 사용하거나 특정 요구 사항에 맞는 사용자 지정 정책을 처음부터 만들 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 생성을 참조하세요.

상태 시스템에 대한 자체 IAM 역할을 만들려면 이 섹션의 단계를 수행합니다.

이 예제에서는 Lambda 함수를 간접적으로 호출할 수 있는 권한을 사용하여 IAM 역할을 만듭니다.

Step Functions의 역할 만들기

1. IAM 콘솔에 로그인한 다음 역할, 역할 생성을 선택합니다.
2. AWS 서비스의 신뢰할 수 있는 엔터티 선택 페이지에 있는 목록에서 Step Functions를 선택한 다음, 다음: 권한을 선택합니다.
3. 연결된 권한 정책 페이지에서 다음: 검토를 선택합니다.
4. 검토 페이지에서 역할 이름에 `StepFunctionsLambdaRole`을 입력한 다음 역할 생성을 선택합니다.

   역할 목록에 IAM 역할이 표시됩니다.

IAM 권한과 정책에 대한 자세한 내용은 IAM 사용 설명서의 액세스 관리를 참조하세요.

교차 서비스 혼동된 대리자 문제 예방

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에게 작업을 수행하도록 강요할 수 있는 보안 문제입니다. AWS에서는 서비스 간 사칭으로 인해 대리인 문제가 혼동될 수 있습니다. 교차 서비스 가장은 한 서비스(호출하는 서비스)가 다른 서비스(호출되는 서비스)를 호출할 때 발생할 수 있습니다. 이러한 유형의 가장은 크로스 계정과 교차 서비스에서 발생할 수 있습니다. 직접적으로 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다.

대리인이 혼동하지 않도록 계정 내 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 통해 모든 서비스의 데이터를 보호할 수 있는 도구를 AWS 제공합니다. 이 섹션에서는 다음과 관련된 서비스 간 혼동 대리 예방에 초점을 맞추고 있지만, IAM AWS Step Functions사용 설명서의 혼동된 대리인 문제 섹션에서 이 주제에 대해 자세히 알아볼 수 있습니다.

Step Functions에서 리소스에 대한 액세스 권한을 다른 서비스에 제공하는 권한을 제한하려면 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키를 사용하는 것이 좋습니다. 하나의 리소스만 교차 서비스 액세스와 연결되도록 허용하려는 경우 `aws:SourceArn`를 사용하십시오. 해당 계정의 모든 리소스가 교차 서비스 사용과 연결되도록 허용하려는 경우 `aws:SourceAccount`을 사용하세요.

혼동된 대리자 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 ARN이 포함된 `aws:SourceArn` 전역 조건 컨텍스트 키를 사용하는 것입니다. 리소스의 전체 ARN을 모르거나 여러 리소스를 지정하는 경우 ARN의 알 수 없는 부분에 와일드카드 문자(*)를 사용한 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용합니다. 예를 들어 `arn:aws:states:*:111122223333:*`입니다.

다음은 혼동된 대리자 문제를 예방하기 위해 Step Functions에서 `aws:SourceArn` 또는 `aws:SourceAccount`를 사용하는 방법을 보여주는 신뢰할 수 있는 정책의 예제입니다.

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

```
                "aws:SourceArn":"arn:aws:states:us-east-1:111122223333:stateMachine:*"
            },
            "StringEquals":{
                "aws:SourceAccount":"111122223333"
            }
        }
    }
  ]
}
```

인라인 정책 연결

Step Functions는 Task 상태에서 직접 다른 서비스를 제어할 수 있습니다. 인라인 정책을 연결하면 Step Functions가 제어해야 하는 서비스의 API 작업에 액세스할 수 있습니다.

1. IAM 콘솔을 열고 역할(Roles)을 선택한 다음 Step Functions 역할을 검색하고 해당 역할을 선택합니다.
2. 인라인 정책 추가를 선택합니다.
3. 시각적 편집기 또는 JSON 탭을 사용하여 역할에 대한 정책을 생성합니다.

다른 AWS 서비스를 제어하는 방법에 대한 자세한 내용은 AWS Step Functions 을 참조하십시오. 다른 서비스와 AWS Step Functions 함께 사용

> **Note**
>
> Step Functions 콘솔에서 만든 IAM 정책의 예제는 통합 서비스용 IAM 정책 섹션을 참조하세요.

관리자가 아닌 사용자에 대한 세부 IAM 권한 만들기

IAM의 기본 관리형 정책 (예ReadOnly:) 은 모든 유형의 AWS Step Functions 권한을 완전히 다루지는 않습니다. 이 단원에서는 이들 다른 유형의 권한을 설명하고 몇 가지 예제 구성을 제공합니다.

Step Functions에는 4가지 권한 범주가 있습니다. 사용자에게 제공하려는 액세스에 따라 이 범주의 권한을 사용하여 액세스를 제어할 수 있습니다.

서비스 수준 권한

특정 리소스에서 작동하지 않는 API 구성 요소에 적용하세요.

상태 시스템 수준 권한

특정 리소스에 작용하는 모든 API 구성 요소에 적용됩니다.

실행 수준 권한

특정 실행에 작용하는 모든 API 구성 요소에 적용됩니다.

작업 수준 권한

특정 작업에 작용하거나, 작업의 특정 인스턴스에 작용하는 모든 API 구성 요소에 적용됩니다.

서비스 수준 권한

이 권한 수준은 특정 리소스에서 작동하지 않는 모든 API 작업에 적용됩니다. 여기에는CreateStateMachine,,CreateActivity, ListStateMachinesListActivities, 등이 포함됩니다ValidationStateMachineDefinition.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:ListStateMachines",
        "states:ListActivities",
        "states:CreateStateMachine",
        "states:CreateActivity",
        "states:ValidationStateMachineDefinition",
      ],
      "Resource": [
        "arn:aws:states:*:*:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam:::role/my-execution-role"
      ]
    }
```

```
  ]
}
```

상태 시스템 수준 권한

이 권한 수준은 특정 상태 시스템에 작용하지 않는 모든 API 작업에 적용됩니다. 이러한 API 작업에는 요청의 일부로 상태 시스템의 Amazon 리소스 이름(ARN)이 필요합니다(예: DeleteStateMachine, DescribeStateMachine, StartExecution 및 ListExecutions).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeStateMachine",
        "states:StartExecution",
        "states:DeleteStateMachine",
        "states:ListExecutions",
        "states:UpdateStateMachine",
        "states:TestState",
        "states:RevealSecrets"
      ],
      "Resource": [
        "arn:aws:states:*:*:stateMachine:StateMachinePrefix*"
      ]
    }
  ]
}
```

실행 수준 권한

이 권한 수준은 특정 실행에 작용하지 않는 모든 API 작업에 적용됩니다. 이러한 API 작업은 요청의 일부로 실행의 ARN을 요구합니다(예: DescribeExecution, GetExecutionHistory, StopExecution).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "states:DescribeExecution",
        "states:DescribeStateMachineForExecution",
        "states:GetExecutionHistory",
        "states:StopExecution"
      ],
      "Resource": [
        "arn:aws:states:*:*:execution:*:ExecutionPrefix*"
      ]
    }
  ]
}
```

작업 수준 권한

이 권한 수준은 특정 작업에 작용하거나 해당 작업의 특정 인스턴스에 작용하는 모든 API 작업에 적용됩니다. 이러한 API 작업에는 요청의 일부로 작업의 ARN 또는 인스턴스 토큰이 필요합니다(예: DeleteActivity, DescribeActivity, GetActivityTask 및 SendTaskHeartbeat).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeActivity",
        "states:DeleteActivity",
        "states:GetActivityTask",
        "states:SendTaskHeartbeat"
      ],
      "Resource": [
        "arn:aws:states:*:*:activity:ActivityPrefix*"
      ]
    }
  ]
}
```

워크플로의 다른 AWS 계정 리소스에 액세스

Step Functions는 AWS 계정 워크플로우에서 서로 다르게 구성된 리소스에 대한 계정 간 액세스를 제공합니다. Step Functions 서비스 통합을 사용하면 AWS 리소스 기반 정책이나 교차 계정 호출을 AWS 서비스 지원하지 않는 경우에도 모든 계정 간 리소스를 호출할 수 있습니다.

예를 들어, 개발과 테스트라는 두 개의 회사를 한 곳에 AWS 계정소유하고 있다고 가정해 보겠습니다. AWS 리전크로스 계정 액세스를 사용하면 개발 계정의 워크플로에서 Amazon S3 버킷, Amazon DynamoDB 테이블, 테스트 계정에서 사용할 수 있는 Lambda 함수와 같은 리소스에 액세스할 수 있습니다.

> **Important**
>
> IAM 역할 및 리소스 기반 정책은 단일 파티션 내에서만 계정 간에 액세스 권한을 위임합니다. 예를 들어 표준 `aws` 파티션의 미국 서부(캘리포니아 북부)에 계정이 있다고 가정합니다. `aws-cn` 파티션의 중국(베이징)에도 계정이 있습니다. 중국(베이징)의 계정에서 Amazon S3 리소스 기반 정책을 사용하여 표준 `aws` 계정의 사용자에 대한 액세스를 허용할 수 없습니다.

크로스 계정 액세스에 대한 자세한 내용은 IAM 사용 설명서의 크로스 계정 정책 평가 로직을 참조하세요.

각 AWS 계정 단계는 자체 리소스를 완벽하게 제어하지만 Step Functions를 사용하면 코드를 사용자 지정할 필요 없이 워크플로의 단계를 재구성, 교체, 추가 또는 제거할 수 있습니다. 프로세스가 변경되거나 애플리케이션이 발전하는 경우에도 이 작업을 수행할 수 있습니다.

또한 여러 계정에서 사용할 수 있도록 중첩된 상태 시스템 실행을 간접적으로 호출할 수 있습니다. 이렇게 하면 워크플로를 효율적으로 분리하고 격리할 수 있습니다. 다른 계정의 또 다른 Step Functions 워크플로에 액세스하는 워크플로에서 `.sync` 서비스 통합 패턴을 사용하면 Step Functions는 할당된 할당량을 소비하는 폴링을 사용합니다. 자세한 정보는 작업 실행(.sync)을 참조하세요.

> **Note**
>
> 현재 Step Functions에서는 지역 간 AWS SDK 통합 및 지역 간 AWS 리소스 액세스를 사용할 수 없습니다.

내용

이 주제의 주요 개념

실행 역할

Step Functions가 코드를 실행하고 AWS 리소스에 액세스하는 데 사용하는 IAM 역할 (예: AWS Lambda 함수의 Invoke 작업)

서비스 통합

워크플로의 특정 Task 상태 내에서 호출할 수 있는 AWS SDK 통합 API 작업.

소스 계정

상태 AWS 계정 머신을 소유하고 실행을 시작한 호스트.

대상 계정

이를 통해 계정 간 통화를 AWS 계정 할 수 있습니다.

대상 역할

상태 시스템이 대상 계정에서 소유하는 리소스를 직접적으로 호출할 수 있도록 위임하는 대상 계정의 IAM 역할입니다.

작업 실행(.sync)

서비스를 호출하는 데 사용되는 서비스 통합 패턴 (예 AWS Batch:). 또한 Step Functions 상태 시스템은 작업이 완료될 때까지 대기한 후 다음 상태로 진행합니다. Step Functions가 대기해야 함을 나타내려면 Task 상태 정의의 Resource 필드에 .sync 접미사를 추가합니다.

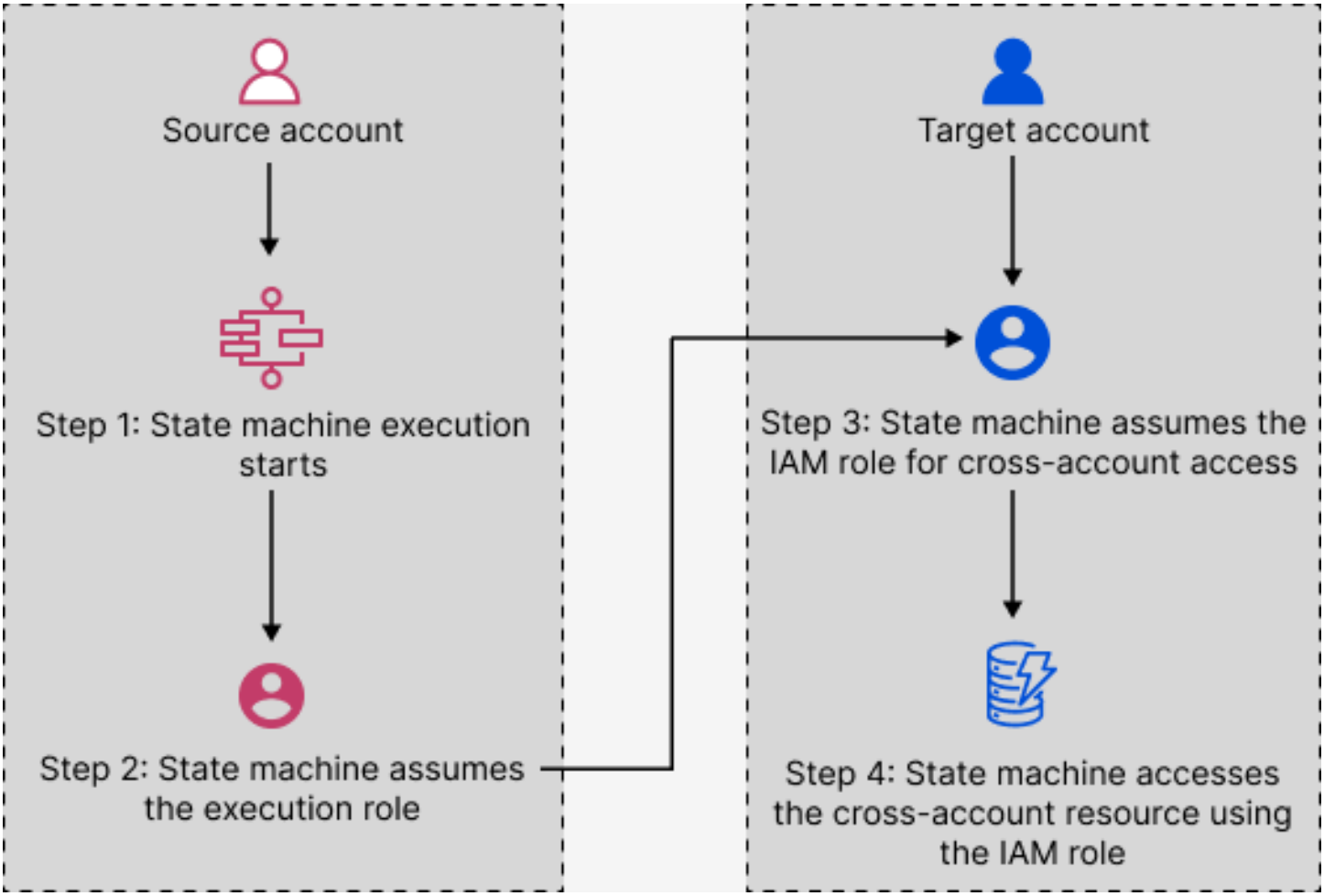
크로스 계정 리소스 간접 호출

워크플로에서 크로스 계정 리소스를 간접적으로 호출하려면 다음을 수행합니다.

1. 리소스가 포함된 대상 계정에 IAM 역할을 만듭니다. 이 역할은 상태 시스템이 포함된 소스 계정에 대상 계정 리소스에 액세스할 수 있는 권한을 부여합니다.
2. Task 상태 정의에서 크로스 계정 리소스를 간접적으로 호출하기 전에 상태 시스템에서 위임할 대상 IAM 역할을 지정합니다.
3. 소스 계정에서 일시적으로 이 역할을 위임할 수 있도록 대상 IAM 역할의 신뢰 정책을 수정합니다. 신뢰 정책에는 소스 계정에 정의된 상태 시스템의 Amazon 리소스 이름(ARN)이 포함되어야 합니다. 또한 대상 IAM 역할에서 AWS 리소스를 호출할 적절한 권한을 정의하십시오.
4. 대상 IAM 역할을 위임하는 데 필요한 권한이 포함되도록 소스 계정 실행 역할을 업데이트합니다.

예시는 자습서: 교차 AWS 계정 리소스 액세스단원을 참조하세요.

> **Note**
>
> 여러 AWS 계정의 리소스에 액세스하기 위해 IAM 역할을 위임하도록 상태 시스템을 구성할 수 있습니다. 하지만 상태 시스템은 한 번에 IAM 역할 하나만 위임할 수 있습니다.



자습서: 교차 AWS 계정 리소스 액세스

Step Functions의 크로스 계정 액세스 지원을 통해 다양한 AWS 계정에서 구성된 리소스를 공유할 수 있습니다. 이 자습서에서는 Production이라는 계정에 정의된 크로스 계정 Lambda 함수에 액세스하는 프로세스를 안내합니다. 이 함수는 Development라는 계정의 상태 시스템에서 간접적으로 호출됩니다. 이 자습서에서는 Development 계정을 소스 계정이라고 하고 Production 계정을 대상 IAM 역할이 포함된 대상 계정이라고 합니다.

먼저 Task 상태 정의에서 크로스 계정 Lambda 함수를 간접적으로 호출하기 전에 상태 시스템에서 위임해야 하는 대상 IAM 역할을 지정합니다. 그런 다음, 소스 계정에서 대상 역할을 일시적으로 위임할 수 있도록 대상 IAM 역할의 신뢰 정책을 수정합니다. 또한 AWS 리소스를 호출하려면 대상 IAM 역할에 적절한 권한을 정의하십시오. 마지막으로, 소스 계정 실행 역할을 업데이트하여 대상 역할을 위임하는 데 필요한 권한을 지정합니다.

여러 AWS 계정의 리소스에 액세스하기 위해 IAM 역할을 위임하도록 상태 시스템을 구성할 수 있습니다. 하지만 상태 시스템은 Task 상태 정의에 따라 한 번에 IAM 역할 하나만 위임할 수 있습니다.

> **Note**
>
> 현재 Step Functions에서는 지역 간 AWS SDK 통합 및 지역 간 AWS 리소스 액세스를 사용할 수 없습니다.

내용

사전 조건

- 이 자습서에서는 Lambda 함수의 예제를 사용하여 크로스 계정 액세스를 설정하는 방법을 보여줍니다. 다른 AWS 리소스를 사용할 수 있지만 다른 계정에서 리소스를 구성했는지 확인하세요.

  > **Important**
  >
  > IAM 역할 및 리소스 기반 정책은 단일 파티션 내에서만 계정 간에 액세스 권한을 위임합니다. 예를 들어 표준 aws 파티션의 미국 서부(캘리포니아 북부)에 계정이 있다고 가정합니다. aws-cn 파티션의 중국(베이징)에도 계정이 있습니다. 중국(베이징)의 계정에서 Amazon S3 리소스 기반 정책을 사용하여 표준 aws 계정의 사용자에 대한 액세스를 허용할 수 없습니다.

- 텍스트 파일에 크로스 계정 리소스의 Amazon 리소스 이름(ARN)을 기록합니다. 이 자습서의 후반부에서 상태 시스템의 Task 상태 정의에 이 ARN을 제공합니다. 다음은 Lambda 함수 ARN의 예제입니다.

```
arn:aws:lambda:us-east-2:123456789012:function:functionName
```

- 상태 시스템에서 위임해야 하는 대상 IAM 역할을 만들었는지 확인합니다.

1단계: Task 상태 정의를 업데이트하여 대상 역할 지정

워크플로의 `Task` 상태에서 크로스 계정 Lambda 함수를 간접적으로 호출하기 전에 상태 시스템에서 위임해야 하는 ID가 포함된 `Credentials` 필드를 추가합니다.

다음 절차에서는 `Echo`라는 크로스 계정 Lambda 함수에 액세스하는 방법을 보여줍니다. 다음 단계에 따라 모든 AWS 리소스를 호출할 수 있습니다.

1. Step Functions 콘솔을 열고 상태 시스템 생성을 선택합니다.
2. 작성 방법 선택 페이지에서 시각적으로 워크플로 설계를 선택하고 기본 선택 사항을 모두 유지합니다.
3. Workflow Studio를 열려면 다음을 선택합니다.
4. 작업 탭에서 `Task` 상태를 끌어 캔버스에 놓습니다. 그러면 이 `Task` 상태를 사용하는 크로스 계정 Lambda 함수가 간접적으로 호출됩니다.
5. 구성 탭에서 다음을 수행합니다.
   a. 상태 이름을 **`Cross-account call`**로 변경합니다.
   b. 함수 이름에 함수 이름 입력을 선택한 다음 상자에 Lambda 함수 ARN을 입력합니다. 예를 들어 `arn:aws:lambda:us-east-2:111122223333:function:Echo`입니다.
   c. IAM 역할 ARN 제공에 대상 IAM 역할 ARN을 지정합니다. 예를 들어 `arn:aws:iam::111122223333:role/LambdaRole`입니다.

   > **Tip**
   > 또는 IAM 역할 ARN이 포함된 상태의 JSON 입력에 기존 키-값 페어에 대한 참조 경로를 지정할 수도 있습니다. 이렇게 하려면 상태 입력에서 런타임 시 IAM 역할 ARN 가져오기를 선택합니다. 참조 경로를 사용하여 값을 지정하는 예제는 JSONPath를 IAM 역할 ARN으로 지정 섹션을 참조하세요.

6. 다음을 선택합니다.
7. 생성된 코드 검토 페이지에서 다음을 선택합니다.

8. 상태 시스템 설정 지정 페이지에서 이름, 권한 및 로깅 수준과 같은 새 상태 시스템에 대한 세부 정보를 지정합니다.
9. 상태 머신 생성을 선택합니다.
10. 상태 시스템의 IAM 역할 ARN과 상태 시스템 ARN을 텍스트 파일에 기록해 둡니다. 대상 계정의 신뢰 정책에 이러한 ARN을 제공해야 합니다.

이제 Task 상태 정의가 다음 정의와 비슷하게 표시되어야 합니다.

```
{
  "StartAt": "Cross-account call",
  "States": {
    "Cross-account call": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "Credentials": {
        "RoleArn": "arn:aws:iam::111122223333:role/LambdaRole"
      },
      "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-2:111122223333:function:Echo",
      },
      "End": true
    }
  }
}
```

2단계: 대상 역할의 신뢰 정책 업데이트

IAM 역할은 대상 계정에 있어야 하며 원본 계정이 이 역할을 일시적으로 위임할 수 있도록 신뢰 정책을 수정해야 합니다. 또한 대상 IAM 역할을 위임할 수 있는 사용자를 제어할 수 있습니다.

신뢰 관계를 생성한 후에는 소스 계정의 사용자가 AWS Security Token Service (AWS STS) AssumeRoleAPI 작업을 사용할 수 있습니다. 이 작업은 대상 계정의 AWS 리소스에 액세스할 수 있는 임시 보안 자격 증명을 제공합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 콘솔의 탐색 창에서 역할을 선택한 다음 검색 상자를 사용하여 대상 IAM 역할을 검색합니다. 예를 들어 *LambdaRole*입니다.
3. 신뢰 관계 탭을 선택합니다.

4. 신뢰 정책 편집을 선택하고 다음 신뢰 정책을 붙여넣습니다. AWS 계정 번호와 IAM 역할 ARN을 교체해야 합니다. `sts:ExternalId` 필드는 역할을 위임할 수 있는 사용자를 추가로 제어합니다. 상태 머신의 이름에는 AWS Security Token Service `AssumeRole` API가 지원하는 문자만 포함되어야 합니다. 자세한 내용은 AWS Security Token Service API AssumeRole참조를 참조하십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/ExecutionRole"  // The source
 account's state machine execution role ARN
      },
      "Condition": {  // Control which account and state machine can assume the
 target IAM role

        "StringEquals": {
          "sts:ExternalId": "arn:aws:states:us-
east-1:123456789012:stateMachine:testCrossAccount"   //// ARN of the state machine
 that will assume the role.
        }
      }
    }
  ]
}
```

5. 이 창을 열어 두고 다음 단계로 진행하여 추가 작업을 수행합니다.

3단계: 대상 역할에 필요한 권한 추가

IAM 정책에서 권한은 특정 요청의 허용 또는 거부 여부를 결정합니다. 대상 IAM 역할에는 Lambda 함수를 간접적으로 호출할 수 있는 올바른 권한이 있어야 합니다.

1. 권한 탭을 선택합니다.
2. 권한 추가를 선택하고 인라인 정책 생성을 선택합니다.
3. JSON 탭을 선택하고 기존 콘텐츠를 다음 권한으로 바꿉니다. Lambda 함수 ARN을 교체해야 합니다.

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:Echo"  // The
 cross-account AWS resource being accessed
    }
  ]
}
```

4. 정책 검토를 선택합니다.
5. 정책 검토 페이지에 권한 이름을 입력한 다음 정책 생성을 선택합니다.

4단계: 대상 역할을 위임할 수 있는 권한을 실행 역할에 추가

Step Functions는 모든 계정 간 서비스 통합에 대한 AssumeRole정책을 자동으로 생성하지 않습니다. AWS 계정하나 이상에서 대상 IAM 역할을 위임할 수 있도록 상태시스템 실행 역할에 필요한 권한을 추가해야 합니다.

1. IAM 콘솔(https://console.aws.amazon.com/iam/)에서 상태 시스템 실행 역할을 엽니다. 방법:
   a. 1단계에서 만든 상태 시스템을 소스 계정에서 엽니다.
   b. 상태 시스템 세부 정보 페이지에서 IAM 역할 ARN을 선택합니다.
2. 권한 탭에서 권한 추가를 선택한 다음 인라인 정책 생성을 선택합니다.
3. JSON 탭을 선택하고 기존 콘텐츠를 다음 권한으로 바꿉니다. Lambda 함수 ARN을 교체해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::111122223333:role/LambdaRole"  // The target role
 to be assumed
    }
  ]
}
```

4. 정책 검토를 선택합니다.

5. 정책 검토 페이지에 권한 이름을 입력한 다음 정책 생성을 선택합니다.

.sync 통합 패턴에 대한 크로스 계정 액세스

워크플로에서 `.sync` 서비스 통합 패턴을 사용하면 Step Functions에서 간접적으로 호출된 크로스 계정 리소스를 폴링하여 작업이 완료되었는지 확인합니다. 이로 인해 실제 작업 완료 시간과 Step Functions에서 작업 완료를 인식하는 시간 사이에서 약간의 지연이 발생합니다. 이 폴링 루프를 완료하려면 대상 IAM 역할에 `.sync` 간접 호출에 필요한 권한이 있어야 합니다. 이렇게 하려면 대상 IAM 역할에 소스 계정에서 권한을 위임할 수 있게 해주는 신뢰 정책이 있어야 합니다. 또한 대상 IAM 역할에는 폴링 루프를 완료하는 데 필요한 권한이 있어야 합니다.

> **Note**
>
> 중첩된 Express 워크플로의 경우 현재 `arn:aws:states:::states:startExecution.sync`는 지원되지 않습니다. 대신 `arn:aws:states:::aws-sdk:sfn:startSyncExecution`을 사용하세요.

.sync 직접 호출에 대한 신뢰 정책 업데이트

다음 예제와 같이 대상 IAM 역할의 신뢰 정책을 업데이트합니다. `sts:ExternalId` 필드는 역할을 위임할 수 있는 사용자를 추가로 제어합니다. 스테이트 머신 이름에는 AWS Security Token Service `AssumeRole` API가 지원하는 문자만 포함되어야 합니다. 자세한 내용은 AWS Security Token Service API AssumeRole참조를 참조하십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": "arn:aws:iam::sourceAccountID:role/InvokeRole",
      },
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "arn:aws:states:us-
east-2:sourceAccountID:stateMachine:stateMachineName"
        }
      }
```

```
    }
  ]
}
```

.sync 직접 호출에 필요한 권한

상태 시스템에 필요한 권한을 부여하려면 대상 IAM 역할에 필요한 권한을 업데이트합니다. 자세한 정보는 the section called “통합 서비스용 IAM 정책”을 참조하세요. 예제 정책의 Amazon EventBridge 권한은 필요하지 않습니다. 예를 들어 상태 시스템을 시작하려면 다음 권한을 추가합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "arn:aws:states:region:accountID:stateMachine:stateMachineName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:StopExecution"
      ],
      "Resource": [
        "arn:aws:states:region:accountID:execution:stateMachineName:*"
      ]
    }
  ]
}
```

Step Functions용 Amazon VPC 엔드포인트

Amazon VPC (Virtual Private Cloud) 를 사용하여 AWS 리소스를 호스팅하는 경우, Amazon VPC와 워크플로를 연결할 수 있습니다. AWS Step Functions 퍼블릭 인터넷을 사용하지 않고도 Step Functions 워크플로에서 이 연결을 사용할 수 있습니다. Amazon VPC 엔드포인트는 표준 워크플로, Express 워크플로 및 동기식 Express 워크플로에서 지원됩니다.

Amazon VPC를 사용하면 사용자 지정 가상 네트워크에서 AWS 리소스를 시작할 수 있습니다. VPC를 사용하여 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 등의 네트워크 설정을 제어할 수 있습니다. VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

Amazon VPC를 Step Functions에 연결하려면 먼저 VPC를 다른 서비스에 연결할 수 있는 인터페이스 VPC 엔드포인트를 정의해야 합니다. AWS 이 엔드포인트를 이용하면 인터넷 게이트웨이나 NAT(네트워크 주소 변환) 인스턴스 또는 VPN 연결 없이도 안정적이고 확장 가능하게 연결됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하십시오.

엔드포인트 만들기

, AWS Command Line Interface (AWS CLI), AWS SDK AWS Management Console, AWS Step Functions API 또는 를 사용하여 VPC에 AWS Step Functions 엔드포인트를 생성할 수 있습니다. AWS CloudFormation

Amazon VPC 콘솔 또는 AWS CLI를 사용한 엔드포인트 생성 및 구성에 대한 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하십시오.

> **Note**
>
> 엔드포인트를 만들 때 VPC를 연결할 서비스로 Step Functions를 지정합니다. Amazon VPC 콘솔에서 서비스 이름은 지역에 따라 AWS 다릅니다. 예를 들어 미국 동부(버지니아 북부)를 선택한 경우 표준 워크플로 및 Express 워크플로의 서비스 이름은 com.amazonaws.us-east-1.states이고 동기식 Express 워크플로의 서비스 이름은 com.amazonaws.us-east-1.sync-states입니다.

> **Note**
>
> 프라이빗 DNS를 통해 SDK에서 엔드포인트를 재정의하지 않고도 VPC 엔드포인트를 사용할 수 있습니다. 하지만 동기식 Express 워크플로의 SDK에서 엔드포인트를 재정의하려면 `DisableHostPrefixInjection` 구성을 `true`로 설정해야 합니다. 예제(Java SDK V2):
>
> ```
> SfnClient.builder()
>   .endpointOverride(URI.create("https://vpce-{vpceId}.sync-states.us-
> east-1.vpce.amazonaws.com"))
>   .overrideConfiguration(ClientOverrideConfiguration.builder()
>
>  .advancedOptions(ImmutableMap.of(SdkAdvancedClientOption.DISABLE_HOST_PREFIX_INJECTION,
>  true))
> ```

```
        .build())
    .build();
```

를 사용하여 엔드포인트를 생성하고 구성하는 방법에 대한 자세한 내용은 사용 AWS CloudFormation 설명서의 AWS: :EC2: :VPCendPoint 리소스를 참조하십시오.AWS CloudFormation

Amazon VPC 엔드포인트 정책

Step Functions에 대한 연결 액세스를 제어하려면 Amazon VPC 엔드포인트를 생성하는 동안 AWS Identity and Access Management (IAM) 엔드포인트 정책을 연결할 수 있습니다. 엔드포인트 정책 여러 개를 연결하여 복잡한 IAM 규칙을 만들 수 있습니다. 자세한 내용은 다음을 참조하십시오.

- Step Functions에 대한 Amazon Virtual Private Cloud 엔드포인트 정책
- 관리자가 아닌 사용자에 대한 세부 IAM 권한 만들기
- VPC 엔드포인트로 서비스에 대한 액세스 제어

Step Functions에 대한 Amazon Virtual Private Cloud 엔드포인트 정책

Step Functions에 대한 Amazon VPC 엔드포인트 정책을 만들어 다음을 지정할 수 있습니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

다음 예제에서는 한 사용자가 상태 시스템을 만들 수 있도록 허용하고 상태 시스템을 삭제할 수 있는 다른 모든 사용자 권한을 거부하는 Amazon VPC 엔드포인트 정책을 보여줍니다. 또한 예제 정책은 모든 사용자에게 실행 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "*Execution",
            "Resource": "*",
            "Effect": "Allow",
            "Principal": "*"
        },
```

```
        {
            "Action": "states:CreateStateMachine",
            "Resource": "*",
            "Effect": "Allow",
            "Principal": {
              "AWS": "arn:aws:iam::123456789012:user/MyUser"
            }
        },
        {
            "Action": "states:DeleteStateMachine",
            "Resource": "*",
            "Effect": "Deny",
            "Principal": "*"
        }
    ]
}
```

엔드포인트 정책 생성에 대한 자세한 내용은 다음을 참조하십시오.

- 관리자가 아닌 사용자에 대한 세부 IAM 권한 만들기
- VPC 엔드포인트로 서비스에 대한 액세스 제어

통합 서비스용 IAM 정책

AWS Step Functions 콘솔에서 스테이트 머신을 생성할 때 Step Functions는 다음과 같이 스테이트 머신 정의에 사용된 리소스를 기반으로 AWS Identity and Access Management (IAM) 정책을 생성합니다.

- 스테이트 머신이 Optimized 통합 중 하나를 사용하는 경우 Step Functions는 스테이트 머신에 필요한 권한 및 역할을 포함하는 정책을 생성합니다. (예외: MediaConvert 통합을 위해서는 권한을 수동으로 설정해야 IAM 정책은 다음과 같습니다. AWS Elemental MediaConvert 합니다. 참조)
- 상태 머신이 AWS SDK 통합 중 하나를 사용하는 경우 부분 권한이 있는 IAM 역할이 생성됩니다. 이후 IAM 콘솔을 사용하여 누락된 역할 정책을 추가할 수 있습니다.

다음 예제에서는 Step Functions가 상태 시스템 정의를 기반으로 IAM 정책을 생성하는 방법을 보여줍니다. *[[resourceName]]* 등과 같은 예제 코드의 항목은 상태 머신 정의에 나열된 정적 리소스로 대체됩니다. 정적 리소스가 여러 개 있는 경우 각 IAM 역할에 대한 항목이 있습니다.

동적 리소스와 정적 리소스 비교

정적 리소스는 상태 머신의 작업 상태에서 직접 정의됩니다. 작업 상태에서 직접 호출할 리소스에 대한 정보를 포함하면 Step Functions에서 해당 리소스에 대한 IAM 역할만 만듭니다.

동적 리소스는 상태 입력으로 전달되고 Path를 사용하여 액세스되는 리소스입이다(경로 참조). 동적 리소스를 작업으로 전달하면 Step Functions에서 `"Resource": "*"`를 지정하는 더 많은 권한이 있는 정책을 만듭니다.

작업 실행 패턴을 사용하는 작업에 대한 추가 권한

작업 실행 패턴(`.sync`로 끝남)을 사용하는 작업의 경우 연결된 서비스의 API 작업에서 응답을 수신하고 모니터링하려면 추가 권한이 필요합니다. 관련 정책에는 요청 응답 또는 콜백 대기 패턴을 사용하는 작업보다 더 많은 권한이 포함됩니다. 동기식 작업에 대한 자세한 내용은 서비스 통합 패턴 섹션을 참조하세요.

> **Note**
>
> Run a Job (.sync) 패턴을 지원하는 서비스 통합에 대한 추가 권한을 제공해야 합니다.

Step Functions는 연결된 서비스에서 작업이 실행될 때 폴링 및 이벤트 등 두 가지 방법을 사용하여 작업 상태를 모니터링합니다.

폴링에는 `ecs:DescribeTasks` 또는 `glue:GetJobRun`과 같은 `Describe` 또는 `Get` API 작업에 대한 권한이 필요합니다. 역할에서 이러한 권한이 누락되면 Step Functions에서 작업 상태를 확인하지 못할 수 있습니다. 이는 일부 Run a Job (.sync) 서비스 통합이 EventBridge 이벤트를 지원하지 않고 일부 서비스는 최선을 다해 이벤트를 전송하기만 하기 때문입니다.

AWS 서비스에서 EventBridge Amazon으로 전송된 이벤트는 관리형 규칙을 사용하여 Step Functions로 전달되며,`events:PutTargets`,`events:PutRule`, 에 대한 권한이 필요합니다`events:DescribeRule`. 역할에서 이러한 권한이 누락되면 Step Functions에서 작업 완료를 인식하기까지 지연이 발생할 수 있습니다. EventBridge 이벤트에 대한 자세한 내용은 AWS 서비스의 이벤트를 참조하십시오.

> **Note**
>
> 폴링과 이벤트를 모두 지원하는 작업 실행(.sync) 작업의 경우에도 여전히 이벤트를 사용하여 작업을 제대로 완료할 수 있습니다. 이는 폴링에 필요한 권한이 역할에 없는 경우에도 발생

할 수 있습니다. 이 경우 폴링 권한이 잘못되었거나 누락된 것을 즉시 알지 못할 수도 있습니다. 드문 경우이긴 하지만 Step Functions에서 이벤트를 전달하거나 처리할 수 없으면 실행이 중단될 수 있습니다. 폴링 권한이 올바르게 구성되었는지 확인하려면 다음과 같은 방법으로 EventBridge 이벤트가 없는 환경에서 실행을 실행할 수 있습니다.

- Step Functions에 이벤트를 전달하는 역할을 하는 관리형 규칙을 EventBridge 삭제하십시오. 이 관리형 규칙은 계정의 모든 상태 시스템에서 공유되므로 다른 상태 시스템에 의도치 않은 영향이 미치지 않도록 테스트 또는 개발 계정에서만 이 작업을 수행해야 합니다. 대상 서비스의 정책 템플릿에서 `events:PutRule`에 사용되는 `Resource` 필드를 검사하여 삭제할 특정 관리형 규칙을 식별할 수 있습니다. 관리형 규칙은 다음에 서비스 통합을 사용하는 상태 시스템을 만들거나 업데이트할 때 다시 생성됩니다. EventBridge 규칙 삭제에 대한 자세한 내용은 규칙 비활성화 또는 삭제를 참조하십시오.
- 작업 실행(.sync) 작업을 완료하기 위한 이벤트 사용을 지원하지 않는 Step Functions Local을 사용하여 테스트합니다. Step Functions Local을 사용하려면 상태 시스템에서 사용하는 IAM 역할을 위임해야 합니다. 신뢰 관계를 편집해야 할 수도 있습니다. `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` 및 `AWS_SESSION_TOKEN` 환경 변수를 위임된 역할 값으로 설정한 다음 java -jar StepFunctionsLocal.jar를 사용하여 Step Functions Local을 시작합니다. 마지막으로 --endpoint-url 매개 변수와 AWS CLI 함께 를 사용하여 상태 머신을 만들고, 실행을 시작하고, 실행 기록을 가져옵니다. 자세한 정보는 상태 시스템을 로컬로 테스트을 참조하세요.

작업 실행(.sync) 패턴을 사용하는 작업이 중지되면 Step Functions는 최선을 다해 작업을 취소합니다. 이렇게 하려면 `batch:TerminateJob` 또는 `eks:DeleteCluster`와 같은 `Cancel`, `Stop`, `Terminate` 또는 `Delete` API 작업에 대한 권한이 필요합니다. 역할에 이러한 권한이 없으면 Step Functions에서 작업을 취소할 수 없으며 작업을 계속 실행하는 동안 추가 요금이 발생할 수 있습니다. 작업 중지 방법에 자세한 내용은 작업 실행을 참조하세요.

IAM 역할을 만드는 데 사용되는 정책 템플릿

다음 주제에는 Step Functions에서 자동으로 새 역할을 만들 때 사용되는 정책 템플릿이 포함되어 있습니다.

> **Note**
>
> 이 템플릿을 검토하여 Step Functions가 IAM 정책을 생성하는 방법을 이해하고 다른 AWS 서비스를 사용할 때 Step Functions에 대한 IAM 정책을 수동으로 생성하는 방법의 예로 들

어 보십시오. Step Functions 서비스 통합에 대한 자세한 내용은 다른 서비스와 AWS Step Functions 함께 사용 섹션을 참조하세요.

주제

아마존 API 게이트웨이에 대한 IAM 정책

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

리소스:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": [
                "arn:[[region]]:[[accountId]]:*"
            ]
        }
    ]
}
```

다음 코드 예제에서는 API Gateway를 직접 호출하기 위한 리소스 정책을 보여줍니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "states.amazonaws.com"
            },
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:<region>:<account-id>:<api-id>/<stage-
name>/<HTTP-VERB>/<resource-path-specifier>",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": [
                        "<SourceStateMachineArn>"
                    ]
                }
            }
        }
    ]
}
```

아마존 아테나에 대한 IAM 정책

다음 예제 템플릿은 상태 시스템 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

`StartQueryExecution`

정적 리소스

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:startQueryExecution",
            "athena:stopQueryExecution",
            "athena:getQueryExecution",
            "athena:getDataCatalog"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/[[workGroup]]",
            "arn:aws:athena:{{region}}:{{accountId}}:datacatalog/*"
        ]
    },
        {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload",
            "s3:CreateBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
```

```
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue:CreateDatabase",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:UpdateDatabase",
            "glue:DeleteDatabase",
            "glue:CreateTable",
            "glue:UpdateTable",
            "glue:GetTable",
            "glue:GetTables",
            "glue:DeleteTable",
            "glue:BatchDeleteTable",
            "glue:BatchCreatePartition",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:BatchGetPartition",
            "glue:DeletePartition",
            "glue:BatchDeletePartition"
        ],
        "Resource": [
            "arn:aws:glue:{{region}}:{{accountId}}:catalog",
            "arn:aws:glue:{{region}}:{{accountId}}:database/*",
            "arn:aws:glue:{{region}}:{{accountId}}:table/*",
            "arn:aws:glue:{{region}}:{{accountId}}:userDefinedFunction/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess"
        ],
        "Resource": [
            "*"
        ]
    }
]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:startQueryExecution",
            "athena:getDataCatalog"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/[[workGroup]]",
            "arn:aws:athena:{{region}}:{{accountId}}:datacatalog/*"
        ]
    },
        {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload",
            "s3:CreateBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue:CreateDatabase",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:UpdateDatabase",
            "glue:DeleteDatabase",
            "glue:CreateTable",
            "glue:UpdateTable",
            "glue:GetTable",
            "glue:GetTables",
```

```
            "glue:DeleteTable",
            "glue:BatchDeleteTable",
            "glue:BatchCreatePartition",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:BatchGetPartition",
            "glue:DeletePartition",
            "glue:BatchDeletePartition"
        ],
        "Resource": [
            "arn:aws:glue:{{region}}:{{accountId}}:catalog",
            "arn:aws:glue:{{region}}:{{accountId}}:database/*",
            "arn:aws:glue:{{region}}:{{accountId}}:table/*",
            "arn:aws:glue:{{region}}:{{accountId}}:userDefinedFunction/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess"
        ],
        "Resource": [
            "*"
        ]
    }
]
}
```

동적 리소스

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:startQueryExecution",
            "athena:stopQueryExecution",
```

```
            "athena:getQueryExecution",
            "athena:getDataCatalog"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*",
            "arn:aws:athena:{{region}}:{{accountId}}:datacatalog/*"
        ]
    },
        {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:ListBucket",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload",
            "s3:CreateBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue:CreateDatabase",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:UpdateDatabase",
            "glue:DeleteDatabase",
            "glue:CreateTable",
            "glue:UpdateTable",
            "glue:GetTable",
            "glue:GetTables",
            "glue:DeleteTable",
            "glue:BatchDeleteTable",
            "glue:BatchCreatePartition",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:BatchGetPartition",
```

```
            "glue:DeletePartition",
            "glue:BatchDeletePartition"
        ],
        "Resource": [
            "arn:aws:glue:{{region}}:{{accountId}}:catalog",
            "arn:aws:glue:{{region}}:{{accountId}}:database/*",
            "arn:aws:glue:{{region}}:{{accountId}}:table/*",
            "arn:aws:glue:{{region}}:{{accountId}}:userDefinedFunction/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lakeformation:GetDataAccess"
        ],
        "Resource": [
            "*"
        ]
    }
]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:startQueryExecution",
            "athena:getDataCatalog"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*",
            "arn:aws:athena:{{region}}:{{accountId}}:datacatalog/*"
        ]
    },
        {
        "Effect": "Allow",
        "Action": [
            "s3:GetBucketLocation",
            "s3:GetObject",
```

```
            "s3:ListBucket",
            "s3:ListBucketMultipartUploads",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload",
            "s3:CreateBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue:CreateDatabase",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:UpdateDatabase",
            "glue:DeleteDatabase",
            "glue:CreateTable",
            "glue:UpdateTable",
            "glue:GetTable",
            "glue:GetTables",
            "glue:DeleteTable",
            "glue:BatchDeleteTable",
            "glue:BatchCreatePartition",
            "glue:CreatePartition",
            "glue:UpdatePartition",
            "glue:GetPartition",
            "glue:GetPartitions",
            "glue:BatchGetPartition",
            "glue:DeletePartition",
            "glue:BatchDeletePartition"
        ],
        "Resource": [
            "arn:aws:glue:{{region}}:{{accountId}}:catalog",
            "arn:aws:glue:{{region}}:{{accountId}}:database/*",
            "arn:aws:glue:{{region}}:{{accountId}}:table/*",
            "arn:aws:glue:{{region}}:{{accountId}}:userDefinedFunction/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
```

```
            "lakeformation:GetDataAccess"
        ],
        "Resource": [
            "*"
        ]
    }
]
}
```

StopQueryExecution

리소스

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:stopQueryExecution"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*"
        ]
    }
    ]
}
```

GetQueryExecution

리소스

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:getQueryExecution"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*"
```

```
        ]
    }
    ]
}
```

GetQueryResults

리소스

```
{
    "Version": "2012-10-17",
    "Statement":[
    {
        "Effect": "Allow",
        "Action": [
            "athena:getQueryResults"
        ],
        "Resource": [
            "arn:aws:athena:{{region}}:{{accountId}}:workgroup/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::*"
        ]
    }
    ]
}
```

IAM 정책은 다음과 같습니다. AWS Batch

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

리소스 수준 액세스 AWS Batch 제어를 부분적으로 지원하므로 반드시 사용해야 합니다. `"Resource": "*"`

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:SubmitJob",
                "batch:DescribeJobs",
                "batch:TerminateJob"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
               "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForBatchJobsRule"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "batch:SubmitJob"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

IAM policies for Amazon Bedrock

콘솔을 사용하여 상태 시스템을 만들면 Step Functions에서 필요한 최소 권한으로 상태 시스템의 실행 역할을 자동으로 만듭니다. 자동으로 생성된 이러한 IAM 역할은 상태 머신을 AWS 리전 생성하는 시점에만 유효합니다.

다음 예제 템플릿은 스테이트 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

IAM 정책을 만들 때는 정책에 와일드카드를 포함하지 않는 것이 좋습니다. 보안 모범 사례로 정책 범위를 최대한 좁혀야 합니다. 런타임 중에 특정 입력 파라미터를 알 수 없는 경우에만 동적 정책을 사용해야 합니다.

이 주제의 내용

- Step Functions를 사용해 Amazon Bedrock을 통합할 때 IAM 정책의 예

Step Functions를 사용해 Amazon Bedrock을 통합할 때 IAM 정책의 예

다음 섹션에서는 특정 파운데이션 모델 또는 프로비저닝 모델에 사용하는 Amazon Bedrock API를 기반으로 필요한 IAM 권한을 설명합니다. 또한 이 섹션에는 전체 액세스 권한을 부여하는 정책의 예가 포함되어 있습니다.

텍스트를 리소스별 정보로 바꿔야 합니다.

- IAM를 사용하여 특정 기반 모델에 액세스하는 정책 예제 InvokeModel
- IAM를 사용하여 특정 프로비저닝된 모델에 액세스하는 정책 예제 InvokeModel
- 사용할 전체 액세스 IAM 정책 예제 InvokeModel
- 특정 파운데이션 모델에 기본 모델로 액세스하는 IAM 정책의 예
- 특정 사용자 지정 모델에 기본 모델로 액세스하는 IAM 정책의 예
- CreateModelCustomizationJob.sync를 사용하기 위한 전체 액세스 IAM 정책 예제
- IAM CreateModelCustomizationJob.sync를 사용하여 특정 기반 모델에 액세스하는 정책 예제
- IAM.sync를 사용하여 CreateModelCustomizationJob 사용자 지정 모델에 액세스하는 정책 예제
- .sync를 사용하기 CreateModelCustomizationJob 위한 전체 액세스 IAM 정책 예제

IAM를 사용하여 특정 기반 모델에 액세스하는 정책 예제 InvokeModel

다음은 InvokeModelAPI 작업을 amazon.titan-text-express-v1 사용하여 이름이 지정된 특정 기반 모델에 액세스하는 상태 머신의 IAM 정책 예제입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "InvokeModel1",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/amazon.titan-text-express-v1"
            ]
        }
    ]
}
```

IAM를 사용하여 특정 프로비저닝된 모델에 액세스하는 정책 예제 InvokeModel

다음은 API 작업을 사용하여 이름이 c2oi931ulksx 지정된 특정 프로비저닝된 모델에 액세스하는 상태 머신의 IAM 정책 예제입니다. InvokeModel

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Sid": "InvokeModel1",
      "Action": [
        "bedrock:InvokeModel"
      ],
      "Resource": [
        "arn:aws:bedrock:us-east-2:123456789012:provisioned-model/c2oi931ulksx"
      ]
    }
  ]
}
```

사용할 전체 액세스 IAM 정책 예제 InvokeModel

다음은 InvokeModelAPI 작업을 사용할 때 전체 액세스를 제공하는 상태 머신의 IAM 정책 예제입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "InvokeModel1",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:provisioned-model/*"
            ]
        }
    ]
}
```

특정 파운데이션 모델에 기본 모델로 액세스하는 IAM 정책의 예

다음은 상태 머신이 CreateModelCustomizationJobAPI 작업을 사용하여 기본 `amazon.titan-text-express-v1` 모델로 명명된 특정 기반 모델에 액세스하는 방법에 대한 IAM 정책 예제입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/amazon.titan-text-express-v1",
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

특정 사용자 지정 모델에 기본 모델로 액세스하는 IAM 정책의 예

다음은 상태 머신이 CreateModelCustomizationJobAPI 작업을 사용하여 기본 모델로서 특정 사용자 지정 모델에 액세스하는 IAM 정책 예제입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/[[roleName]]"
            ]
        }
    ]
}
```

CreateModelCustomizationJob.sync를 사용하기 위한 전체 액세스 IAM 정책 예제

다음은 CreateModelCustomizationJobAPI 작업을 사용할 때 전체 액세스를 제공하는 상태 머신의 IAM 정책 예제입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

IAM CreateModelCustomizationJob.sync를 사용하여 특정 기반 모델에 액세스하는 정책 예제

CreateModelCustomizationJob다음은.sync API 작업을 amazon.titan-text-express-v1 사용하여 이름이 지정된 특정 기반 모델에 액세스하는 상태 머신의 IAM 정책 예제입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/amazon.titan-text-express-
v1",
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "bedrock:GetModelCustomizationJob",
                "bedrock:StopModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob3",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

IAM.sync를 사용하여 CreateModelCustomizationJob 사용자 지정 모델에 액세스하는 정책 예제

CreateModelCustomizationJob다음은.sync API 작업을 사용하여 사용자 지정 모델에 액세스하는 상태 머신의 IAM 정책 예제입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "bedrock:GetModelCustomizationJob",
                "bedrock:StopModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob3",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

.sync를 사용하기 CreateModelCustomizationJob 위한 전체 액세스 IAM 정책 예제

CreateModelCustomizationJob다음은.sync API 작업을 사용할 때 전체 액세스를 제공하는 상태 머신의 IAM 정책 예제입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob1",
            "Action": [
                "bedrock:CreateModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2::foundation-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:custom-model/*",
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob2",
            "Action": [
                "bedrock:GetModelCustomizationJob",
                "bedrock:StopModelCustomizationJob"
            ],
            "Resource": [
                "arn:aws:bedrock:us-east-2:123456789012:model-customization-job/*"
            ]
        },
        {
            "Effect": "Allow",
            "Sid": "CreateModelCustomizationJob3",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/myRole"
            ]
        }
    ]
}
```

다음에 대한 IAM 정책 AWS CodeBuild

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

리소스:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:sa-east-1:123456789012:StepFunctionsSample-
CodeBuildExecution1111-2222-3333-wJalrXUtnFEMI-SNSTopic-bPxRfiCYEXAMPLEKEY"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "codebuild:StartBuild",
                "codebuild:StopBuild",
                "codebuild:BatchGetBuilds",
                "codebuild:BatchGetReports"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:sa-east-1:123456789012:rule/
StepFunctionsGetEventForCodeBuildStartBuildRule"
            ],
            "Effect": "Allow"
        }
    ]
}
```

StartBuild

정적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StartBuild",
        "codebuild:StopBuild",
        "codebuild:BatchGetBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule",
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventForCodeBuildStartBuildRule"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StartBuild"
      ],
      "Resource": [
```

```
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    }
  ]
}
```

동적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StartBuild",
        "codebuild:StopBuild",
        "codebuild:BatchGetBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule",
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventForCodeBuildStartBuildRule"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StartBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    }
  ]
}
```

StopBuild

정적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:StopBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    }
  ]
}
```

동적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "codebuild:StopBuild"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
    }
  ]
}
```

BatchDeleteBuilds

정적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchDeleteBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
      ]
    }
  ]
}
```

동적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchDeleteBuilds"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:project/*"
      ]
```

```
    }
  ]
}
```

BatchGetReports

정적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetReports"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:[[accountId]]:report-group/[[reportName]]"
      ]
    }
  ]
}
```

동적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetReports"
      ],
      "Resource": [
        "arn:aws:codebuild:[[region]]:*:report-group/*"
      ]
    }
  ]
}
```

StartBuildBatch

정적 리소스

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StartBuildBatch",
                "codebuild:StopBuildBatch",
                "codebuild:BatchGetBuildBatches"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventForCodeBuildStartBuildBatchRule"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StartBuildBatch"
            ],
            "Resource": [
```

```
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

동적 리소스

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StartBuildBatch",
                "codebuild:StopBuildBatch",
                "codebuild:BatchGetBuildBatches"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventForCodeBuildStartBuildBatchRule"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StartBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

StopBuildBatch

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StopBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StopBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

RetryBuildBatch

정적 리소스

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:RetryBuildBatch",
                "codebuild:StopBuildBatch",
                "codebuild:BatchGetBuildBatches"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "codebuild:RetryBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

동적 리소스

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:RetryBuildBatch",
                "codebuild:StopBuildBatch",
                "codebuild:BatchGetBuildBatches"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:RetryBuildBatch"
```

```
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
}
```

DeleteBuildBatch

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:DeleteBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/[[projectName]]"
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:DeleteBuildBatch"
            ],
            "Resource": [
                "arn:aws:codebuild:[[region]]:[[accountId]]:project/*"
            ]
        }
    ]
```

```
}
```

아마존 DynamoDB에 대한 IAM 정책

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:[[region]]:[[accountId]]:table/[[tableName]]"
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

모든 DynamoDB API 작업에 대한 IAM 정책에 대한 자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB의 IAM 정책을 참조하세요. 또한 DynamoDB용 PartiQL의 IAM 정책에 대한 자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB용 PartiQL의 IAM 정책을 참조하세요.

아마존 ECS를 위한 IAM 정책/AWS Fargate

다음 예제 템플릿은 상태 시스템 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

작업 제출 전까지 `TaskId` 값이 알려지지 않으므로 Step Functions는 권한이 더 많이 부여된 `"Resource": "*"` 정책을 만듭니다.

> **Note**
>
> "*" IAM 정책에도 불구하고 Step Functions에서 시작한 Amazon Elastic Container Service(Amazon ECS) 작업만 중지할 수 있습니다.

Run a Job (.sync)

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "arn:aws:ecs:[[region]]:
[[accountId]]:task-definition/[[taskDefinition]]:[[revisionNumber]]"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ecs:StopTask",
                "ecs:DescribeTasks"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
               "arn:aws:events:[[region]]:
[[accountId]]:rule/StepFunctionsGetEventsForECSTaskRule"
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask",
                "ecs:StopTask",
                "ecs:DescribeTasks"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
```

```
                "events:DescribeRule"
            ],
            "Resource": [
               "arn:aws:events:[[region]]:
[[accountId]]:rule/StepFunctionsGetEventsForECSTaskRule"
            ]
        }
    ]
}
```

Request Response and Callback (.waitForTaskToken)

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
                "arn:aws:ecs:[[region]]:
[[accountId]]:task-definition/[[taskDefinition]]:[[revisionNumber]]"
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

예약된 Amazon ECS 작업에 작업 실행 역할, 작업 역할 또는 작업 역할 재정의를 사용해야 하는 경우 각 작업 실행 역할, 작업 역할 또는 작업 역할 재정의에 대한 `iam:PassRole` 권한을 호출 엔티티의 CloudWatch Events IAM 역할 (이 경우 Step Functions) 에 추가해야 합니다.

아마존 EKS의 IAM 정책

다음 예제 템플릿은 상태 시스템 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

`CreateCluster`

리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:CreateCluster"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:DeleteCluster"
            ],
            "Resource": "arn:aws:eks:sa-east-1:444455556666:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::444455556666:role/StepFunctionsSample-EKSClusterManag-
EKSServiceRole-ANPAJ2UCCR6DPCEXAMPLE"
            ],
```

```
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

CreateNodeGroup

리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeSubnets",
                "eks:CreateNodegroup"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeNodegroup",
                "eks:DeleteNodegroup"
            ],
            "Resource": "arn:aws:eks:sa-east-1:444455556666:nodegroup/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": "arn:aws:iam::444455556666:role/*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::444455556666:role/StepFunctionsSample-EKSClusterMan-
NodeInstanceRole-ANPAJ2UCCR6DPCEXAMPLE"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

DeleteCluster

리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DeleteCluster",
                "eks:DescribeCluster"
            ],
            "Resource": [
                "arn:aws:eks:sa-east-1:444455556666:cluster/ExampleCluster"
            ]
        }
    ]
}
```

DeleteNodegroup

리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "eks:DeleteNodegroup",
                "eks:DescribeNodegroup"
            ],
            "Resource": [
                "arn:aws:eks:sa-east-1:444455556666:nodegroup/ExampleCluster/
ExampleNodegroup/*"
            ]
        }
    ]
}
```

Step Functions에서 Amazon EKS 사용에 대한 자세한 내용은 Step Functions를 사용하여 Amazon EKS 호출 섹션을 참조하세요.

아마존 EMR에 대한 IAM 정책

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

`addStep`

정적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:AddJobFlowSteps",
        "elasticmapreduce:DescribeStep",
        "elasticmapreduce:CancelSteps"
      ],
      "Resource": [
        "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
      ]
    }
  ]
}
```

동적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:AddJobFlowSteps",
        "elasticmapreduce:DescribeStep",
        "elasticmapreduce:CancelSteps"
      ],
      "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
    }
  ]
}
```

cancelStep

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:CancelSteps",
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:CancelSteps",
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
```

```
        }
    ]
}
```

createCluster

정적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:RunJobFlow",
        "elasticmapreduce:DescribeCluster",
        "elasticmapreduce:TerminateJobFlows"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": [
        "arn:aws:iam::{{account}}:role/[[roleName]]"
      ]
    }
  ]
}
```

setClusterTerminationProtection

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:SetTerminationProtection",
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
```

```
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "elasticmapreduce:SetTerminationProtection",
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        }
    ]
}
```

modifyInstanceFleetByName

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceFleet",
                "elasticmapreduce:ListInstanceFleets"
            ],
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

동적 리소스

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceFleet",
                "elasticmapreduce:ListInstanceFleets"
            ],
            "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
        }
    ]
}
```

`modifyInstanceGroupByName`

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceGroups",
                "elasticmapreduce:ListInstanceGroups"
            ],
            "Resource": [

 "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:ModifyInstanceGroups",
```

```
                "elasticmapreduce:ListInstanceGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

terminateCluster

정적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:TerminateJobFlows",
        "elasticmapreduce:DescribeCluster"
      ],
      "Resource": [
        "arn:aws:elasticmapreduce:[[region]]:[[accountId]]:cluster/[[clusterId]]"
      ]
    }
  ]
}
```

동적 리소스

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:TerminateJobFlows",
        "elasticmapreduce:DescribeCluster"
      ],
      "Resource": "arn:aws:elasticmapreduce:*:*:cluster/*"
    }
  ]
}
```

EKS 기반 아마존 EMR에 대한 IAM 정책

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

`CreateVirtualCluster`

리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:CreateVirtualCluster"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::{{accountId}}:role/aws-service-role/emr-
containers.amazonaws.com/AnAWSServiceRoleForAmazonEMRContainers",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "emr-containers.amazonaws.com"
                }
            }
        }
    ]
}
```

`DeleteVirtualCluster`

정적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": [
        "emr-containers:DeleteVirtualCluster",
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DeleteVirtualCluster"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]"
      ]
    }
  ]
}
```

동적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DeleteVirtualCluster",
```

```
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DeleteVirtualCluster"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ]
    }
  ]
}
```

`StartJobRun`

정적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]"
      ],
```

```
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "[[executionRoleArn]]"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeJobRun",
        "emr-containers:CancelJobRun"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]/jobruns/*"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/
[[virtualClusterId]]"
      ],
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "[[executionRoleArn]]"
          ]
        }
      }
    }
  ]
```

```
}
```

동적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ],
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "[[executionRoleArn]]"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeJobRun",
        "emr-containers:CancelJobRun"
      ],
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ]
    }
  ]
}
```

Request Response

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": [
        "arn:aws:emr-containers:{{region}}:{{accountId}}:/virtualclusters/*"
      ],
      "Condition": {
        "StringEquals": {
          "emr-containers:ExecutionRoleArn": [
            "[[executionRoleArn]]"
          ]
        }
      }
    }
  ]
}
```

Amazon EMR Serverless의 IAM 정책

콘솔을 사용하여 상태 시스템을 만들면 Step Functions에서 필요한 최소 권한으로 상태 시스템의 실행 역할을 자동으로 만듭니다. 자동으로 생성된 이러한 IAM 역할은 상태 머신을 생성한 시점에만 유효합니다. AWS 리전

다음 예제 템플릿은 스테이트 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

IAM 정책을 만들 때는 정책에 와일드카드를 포함하지 않는 것이 좋습니다. 보안 모범 사례로 정책 범위를 최대한 좁혀야 합니다. 런타임 중에 특정 입력 파라미터를 알 수 없는 경우에만 동적 정책을 사용해야 합니다.

또한 관리자는 관리자가 아닌 사용자에게 상태 시스템을 실행할 수 있는 실행 역할을 부여할 때 주의해야 합니다. 정책을 직접 만드는 경우 실행 역할에 passRole 정책을 포함하는 것이 좋습니다. 또한 실행 역할에 `aws:SourceARN` 및 `aws:SourceAccount` 컨텍스트 키를 추가하는 것이 좋습니다.

이 주제의 내용

- Step Functions와 EMR 서버리스 통합에 대한 IAM 정책 예제

Step Functions와 EMR 서버리스 통합에 대한 IAM 정책 예제

- IAM 정책 예제: CreateApplication

- IAM 정책 예제: StartApplication
- IAM 정책 예시: StopApplication
- IAM 정책 예시: DeleteApplication
- IAM 정책 예시: StartJobRun
- IAM 정책 예시: CancelJobRun

IAM 정책 예제: CreateApplication

다음은 상태가 있는 상태 머신을 위한 IAM 정책 예제입니다. CreateApplication 태스크 상태

> **Note**
>
> 계정에서 처음으로 애플리케이션을 생성할 때 IAM 정책에 CreateServiceLinkedRole 권한을 지정해야 합니다. 이후에는 이 권한을 추가할 필요가 없습니다. 에 대한 CreateServiceLinkedRole 자세한 내용은 https://docs.aws.amazon.com/IAM/latest/APIReference/ 을 참조하십시오 CreateServiceLinkedRole.

다음 정책의 정적 리소스와 동적 리소스는 동일합니다.

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:CreateApplication"
            ],
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:GetApplication",
                "emr-serverless:DeleteApplication"
            ],
```

```
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:{{accountId}}:rule/
StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::{{accountId}}:role/aws-service-role/ops.emr-
serverless.amazonaws.com/AWS ServiceRoleForAmazonEMRServerless*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"
                }
            }
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",

   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:CreateApplication"
            ],
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::{{accountId}}:role/aws-service-role/ops.emr-
serverless.amazonaws.com/AWS ServiceRoleForAmazonEMRServerless*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"
                }
            }
        }
    ]
}
```

IAM 정책 예제: StartApplication

정적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 정적 리소스에 대한 IAM 정책 예제입니다. StartApplication 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartApplication",
               "emr-serverless:GetApplication",
               "emr-serverless:StopApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        }
    ]
}
```

동적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 동적 리소스에 대한 IAM 정책 예제입니다. StartApplication 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
```

```
            "Action": [
                "emr-serverless:StartApplication",
                "emr-serverless:GetApplication",
                "emr-serverless:StopApplication"
             ],
            "Resource": [
                 "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
             ]
        },
        {
             "Effect": "Allow",
             "Action": [
                 "events:PutTargets",
                 "events:PutRule",
                 "events:DescribeRule"
             ],
             "Resource": [
                 "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
             ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:StartApplication"
             ],
            "Resource": [
                 "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
             ]
        }
    ]
}
```

IAM 정책 예시: StopApplication

정적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 정적 리소스에 대한 IAM 정책 예제입니다. StopApplication 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StopApplication",
               "emr-serverless:GetApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
```

```
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StopApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        }
    ]
}
```

동적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 동적 리소스에 대한 IAM 정책 예제입니다. StopApplication 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StopApplication",
               "emr-serverless:GetApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
```

```
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StopApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        }
    ]
}
```

IAM 정책 예시: DeleteApplication

정적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 정적 리소스에 대한 IAM 정책 예제입니다. DeleteApplication 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:DeleteApplication",
               "emr-serverless:GetApplication"
            ],
```

```
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:DeleteApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        }
    ]
}
```

동적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 동적 리소스에 대한 IAM 정책 예제입니다. DeleteApplication 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:DeleteApplication",
               "emr-serverless:GetApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessApplicationRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:DeleteApplication"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
```

```
        }
    ]
}
```

IAM 정책 예시: StartJobRun

정적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 정적 리소스에 대한 IAM 정책 예제입니다. StartJobRun 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "[[jobExecutionRoleArn]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "emr-serverless:GetJobRun",
                "emr-serverless:CancelJobRun"
```

```
            ],
            "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]/jobruns/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessJobRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "[[jobExecutionRoleArn]]"
            ],
            "Condition": {
```

```
                "StringEquals": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        }
    ]
}
```

동적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 동적 리소스에 대한 IAM 정책 예제입니다. StartJobRun 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartJobRun",
               "emr-serverless:GetJobRun",
               "emr-serverless:CancelJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "[[jobExecutionRoleArn]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessJobRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:StartJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "[[jobExecutionRoleArn]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        }
    ]
}
```

IAM 정책 예시: CancelJobRun

정적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 정적 리소스에 대한 IAM 정책 예제입니다.
CancelJobRun 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:CancelJobRun",
               "emr-serverless:GetJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]/jobruns/[[jobRunId]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessJobRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
```

```
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:CancelJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/
[[applicationId]]/jobruns/[[jobRunId]]"
            ]
        }
    ]
}
```

동적 리소스

다음은 상태가 있는 상태 머신을 사용할 때의 동적 리소스에 대한 IAM 정책 예제입니다. CancelJobRun 태스크 상태

Run a Job (.sync)

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:CancelJobRun",
               "emr-serverless:GetJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
```

```
                "arn:aws:events:{{region}}:
{{accountId}}:rule/StepFunctionsGetEventsForEMRServerlessJobRule"
            ]
        }
    ]
}
```

Request Response

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
           "Effect": "Allow",
           "Action": [
               "emr-serverless:CancelJobRun"
            ],
           "Resource": [
                "arn:aws:emr-serverless:{{region}}:{{accountId}}:/applications/*"
            ]
        }
    ]
}
```

아마존용 IAM 정책 EventBridge

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

`PutEvents`

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "events:PutEvents"
            ],
            "Resource": [
```

```
                "arn:aws:events:us-east-1:123456789012:event-bus/stepfunctions-
sampleproject-eventbus"
            ],
            "Effect": "Allow"
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "events:PutEvents"
            ],
            "Resource": "arn:aws:events:*:*:event-bus/*"
        }
    ]
}
```

Step EventBridge Functions와 함께 사용하는 방법에 대한 자세한 내용은 을 참조하십시오 EventBridge Step 함수를 사용한 호출.

다음에 대한 IAM 정책 AWS Lambda

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

AWS Step Functions 스테이트 머신 정의를 기반으로 IAM 정책을 생성합니다. `function1`및 `function2` 를 호출하는 두 개의 AWS Lambda 작업 상태가 있는 상태 머신의 경우 두 함수에 대한 `lambda:Invoke` 권한이 있는 정책을 사용해야 합니다.

방법은 다음 예제와 같습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:[[region]]:[[accountId]]:function:[[function1]]",
                "arn:aws:lambda:[[region]]:[[accountId]]:function:[[function2]]"
            ]
        }
    ]
}
```

IAM 정책은 다음과 같습니다. AWS Elemental MediaConvert

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 설정해야 하는 방법을 보여줍니다. IAM 콘솔을 사용하여 누락된 역할 정책을 추가할 수 있습니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

리소스 수준 액세스 MediaConvert 제어를 부분적으로 지원하므로 반드시 사용해야 합니다. `"Resource": "*"`

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "mediaconvert:CreateJob",
                "mediaconvert:GetJob",
                "mediaconvert:CancelJob"
            ],
            "Resource": "*"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
               "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForMediaConvertJobRule"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "mediaconvert:CreateJob"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM 정책은 다음과 같습니다. AWS Glue

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

AWS Glue 리소스 기반 제어 기능이 없습니다.

Run a Job (.sync)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartJobRun",
                "glue:GetJobRun",
                "glue:GetJobRuns",
                "glue:BatchStopJobRun"
            ],
            "Resource": "*"
        }
    ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:StartJobRun"
            ],
            "Resource": "*"
        }
    ]
}
```

에 대한 IAM 정책 AWS Glue DataBrew

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

Run a Job (.sync)

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "databrew:startJobRun",
                "databrew:listJobRuns",
                "databrew:stopJobRun"
            ],
            "Resource": [
                "arn:aws:databrew:{{region}}:{{accountId}}:job/*"
            ]
        }
    ]
}
```

Request Response

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "databrew:startJobRun"
            ],
            "Resource": [
                "arn:aws:databrew:{{region}}:{{accountId}}:job/*"
            ]
        }
    ]
}
```

아마존용 IAM 정책 SageMaker

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

> **Note**
>
> 이 예제는 ML 컴퓨팅 인스턴스에 배포하거나 일괄 변환 작업을 위해 모델 아티팩트와 docker 이미지에 액세스하는 데 SageMaker 사용하는 IAM 역할의 Amazon 리소스 이름 (ARN) 을 참조하십시오. *[[roleArn]]* 자세한 내용은 Amazon SageMaker 역할을 참조하십시오.

`CreateTrainingJob`

정적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTrainingJob",
        "sagemaker:DescribeTrainingJob",
        "sagemaker:StopTrainingJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:[[region]]:[[accountId]]:training-
job/[[trainingJobName]]*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:ListTags"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
```

```
      "Resource": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule",
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForSageMakerTrainingJobsRule"
      ]
    }
  ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTrainingJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:[[region]]:[[accountId]]:training-
job/[[trainingJobName]]*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "sagemaker:ListTags"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    }
  ]
}
```

동적 리소스

.sync or .waitForTaskToken

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTrainingJob",
        "sagemaker:DescribeTrainingJob",
        "sagemaker:StopTrainingJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:[[region]]:[[accountId]]:training-job/*"
      ]
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:ListTags"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule",
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForSageMakerTrainingJobsRule"
      ]
    }
  ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTrainingJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:[[region]]:[[accountId]]:training-job/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:ListTags"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    }
  ]
}
```

`CreateTransformJob`

> **Note**
>
> AWS Step Functions 와 SageMaker 통합되는 상태 머신을 생성할 `CreateTransformJob` 때 적용되는 정책은 자동으로 생성되지 않습니다. 다음 IAM 예제 중 하나를 기반으로 만든 역할에 인라인 정책을 연결해야 합니다.

정적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTransformJob",
        "sagemaker:DescribeTransformJob",
        "sagemaker:StopTransformJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:[[region]]:[[accountId]]:transform-
job/[[transformJobName]]*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:ListTags"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
```

```
      "Resource": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule",
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForSageMakerTransformJobsRule"
      ]
    }
  ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTransformJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:[[region]]:[[accountId]]:transform-
job/[[transformJobName]]*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "sagemaker:ListTags"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    }
  ]
}
```

동적 리소스

Run a Job (.sync)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTransformJob",
        "sagemaker:DescribeTransformJob",
        "sagemaker:StopTransformJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:[[region]]:[[accountId]]:transform-job/*"
      ]
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:ListTags"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutTargets",
        "events:PutRule",
        "events:DescribeRule"
      ],
      "Resource": [
        "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForSageMakerTransformJobsRule"
      ]
    }
  ]
}
```

Request Response and Callback (.waitForTaskToken)

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateTransformJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:[[region]]:[[accountId]]:transform-job/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:ListTags"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "[[roleArn]]"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    }
  ]
}
```

아마존 SNS를 위한 IAM 정책

다음 예제 템플릿은 상태 시스템 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

정적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:[[region]]:[[accountId]]:[[topicName]]"
            ]
        }
    ]
}
```

Path 기반 리소스 또는 *TargetArn*이나 *PhoneNumber*로 게시:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "*"
        }
    ]
}
```

아마존 SQS에 대한 IAM 정책

다음 예제 템플릿은 상태 시스템 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

정적 리소스

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": [
                "arn:aws:sqs:[[region]]:[[accountId]]:[[queueName]]"
            ]
        }
    ]
}
```

동적 리소스

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM 정책은 다음과 같습니다. AWS Step Functions

단일 중첩 워크플로 실행을 위해 `StartExecution`을 직접적으로 호출하는 상태 시스템의 경우 해당 상태 시스템에 대한 권한을 제한하는 IAM 정책을 사용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:StartExecution"
            ],
```

```
            "Resource": [

 "arn:aws:states:[[region]]:[[accountId]]:stateMachine:[[stateMachineName]]"
            ]
        }
    ]
}
```

자세한 내용은 다음 자료를 참조하십시오.

- 다른 서비스와 AWS Step Functions 함께 사용
- 파라미터를 서비스 API에 전달
- 통합 서비스로서의 AWS Step Functions 실행 관리

Synchronous

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:StartExecution"
            ],
            "Resource": [
                "arn:aws:states:[[region]]:[[accountId]]:stateMachine:
[[stateMachineName]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "states:DescribeExecution",
                "states:StopExecution"
            ],
            "Resource": [

 "arn:aws:states:[[region]]:[[accountId]]:execution:[[stateMachineName]]:*"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "events:PutTargets",
                "events:PutRule",
                "events:DescribeRule"
            ],
            "Resource": [
               "arn:aws:events:[[region]]:[[accountId]]:rule/
StepFunctionsGetEventsForStepFunctionsExecutionRule"
            ]
        }
    ]
}
```

Asynchronous

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "states:StartExecution"
            ],
            "Resource": [

 "arn:aws:states:[[region]]:[[accountId]]:stateMachine:[[stateMachineName]]"
            ]
        }
    ]
}
```

중첩된 워크플로 실행에 대한 자세한 내용은 작업 상태에서 워크플로 실행 시작 단원을 참조하십시오.

다음에 대한 IAM 정책 AWS X-Ray

다음 예제 템플릿은 상태 머신 정의의 리소스를 기반으로 IAM 정책을 AWS Step Functions 생성하는 방법을 보여줍니다. 자세한 내용은 통합 서비스용 IAM 정책 및 서비스 통합 패턴 섹션을 참조하세요.

X-Ray 추적을 활성화하려면 추적을 허용할 수 있는 적절한 권한이 있는 IAM 정책이 필요합니다. 상태 시스템에서 다른 통합 서비스를 사용하는 경우 추가 IAM 정책이 필요할 수 있습니다. 특정 서비스 통합에 대한 IAM 정책을 참조하세요.

X-Ray 추적이 활성화된 상태 시스템을 만들면 IAM 정책이 자동으로 생성됩니다.

> **Note**
>
> 기존 상태 시스템에 X-Ray 추적을 활성화하는 경우 X-Ray 추적을 활성화할 수 있는 충분한 권한이 있는 정책을 추가해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords",
                "xray:GetSamplingRules",
                "xray:GetSamplingTargets"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Step Functions에서 X-Ray 사용에 대한 자세한 내용은 AWS X-Ray 및 Step Functions 섹션을 참조하세요.

활동 또는 작업 없음

`Activity` 작업만 있거나 작업이 전혀 없는 상태 시스템의 경우 모든 작업과 리소스에 대한 액세스를 거부하는 IAM 정책을 사용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*"
```

```
        }
    ]
}
```

`Activity` 작업 사용에 대한 자세한 내용은 활동 단원을 참조하십시오.

Distributed Map 상태를 사용하기 위한 IAM 정책

Step Functions 콘솔을 사용하여 워크플로를 만들면 Step Functions에서 워크플로 정의의 리소스를 기반으로 IAM 정책을 자동으로 생성할 수 있습니다. 이러한 정책에는 상태 시스템 역할에서 Distributed Map 상태에 대한 `StartExecution` API 작업을 간접적으로 호출하도록 허용하는 데 필요한 최소 권한이 포함되어 있습니다. 또한 이러한 정책에는 Amazon S3 버킷과 객체, Lambda 함수와 같은 AWS 리소스에 액세스하는 데 필요한 최소 권한의 Step Functions가 포함됩니다. IAM 정책에 필요한 권한만 포함하는 것이 좋습니다. 예를 들어 워크플로에 분산 모드의 `Map` 상태가 포함된 경우 정책 범위를 데이터 세트가 포함된 특정 Amazon S3 버킷과 폴더로 좁힙니다.

> **Important**
>
> Distributed Map 상태 입력에 있는 기존 키-값 페어에 대한 참조 경로를 사용하여 Amazon S3 버킷과 객체 또는 접두사를 지정하는 경우 워크플로에 대한 IAM 정책을 업데이트해야 합니다. 정책 범위를 런타임 시 경로에서 확인하는 버킷과 객체 이름으로 좁히세요.

이 주제에서 수행할 작업

- Distributed Map 상태를 실행하기 위한 IAM 정책의 예제
- Distributed Map을 redriving하기 위한 IAM 정책 예제
- Amazon S3 데이터 세트에서 데이터 읽기에 대한 IAM 정책 예제
- Amazon S3 버킷에 데이터를 쓰기 위한 IAM 정책 예제

Distributed Map 상태를 실행하기 위한 IAM 정책의 예제

워크플로에 Distributed Map 상태가 포함된 경우 Step Functions에는 상태 시스템 역할에서 Distributed Map 상태의 `StartExecution` API 작업을 호출할 수 있는 적절한 권한이 있어야 합니다.

다음 IAM 정책 예제에서는 Distributed Map 상태를 실행하는 데 필요한 최소 권한을 상태 시스템 역할에 부여합니다.

> **Note**
>
> *stateMachineName*을 Distributed Map 상태를 사용하는 상태 시스템의 이름으로 바꿔야 합니다. 예를 들어 arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine*입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:StartExecution"
      ],
      "Resource": [
        "arn:aws:states:region:accountID:stateMachine:stateMachineName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "states:DescribeExecution",
        "states:StopExecution"
      ],
      "Resource": "arn:aws:states:region:accountID:execution:stateMachineName:*"
    }
  ]
}
```

Distributed Map을 redriving하기 위한 IAM 정책 예제

상위 워크플로를 redriving하면 맵 실행에서 실패한 하위 워크플로 실행을 다시 시작할 수 있습니다. redriven 상위 워크플로에서 Distributed Map을 비롯한 실패한 모든 상태를 redrives합니다. 상위 워크플로에서 `RedriveExecution` API 작업을 간접적으로 호출하는 데 필요한 최소 권한이 실행 역할에 있는지 확인합니다.

다음 IAM 정책 예제에서는 Distributed Map 상태를 redriving하는 데 필요한 최소 권한을 상태 시스템에 부여합니다.

> **Note**
>
> *stateMachineName*을 Distributed Map 상태를 사용하는 상태 시스템의 이름으로 바꿔야 합니다. 예를 들어 arn:aws:states:*us-east-2*:*123456789012*:stateMachine:*mystateMachine*입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "states:RedriveExecution"
      ],
      "Resource": "arn:aws:states:us-
east-2:123456789012:execution:myStateMachine/myMapRunLabel:*"
    }
  ]
}
```

Amazon S3 데이터 세트에서 데이터 읽기에 대한 IAM 정책 예제

다음 IAM 정책 예제는 ListObjectsV2 및 GetObjectAPI 작업을 사용하여 Amazon S3 데이터 세트에 액세스하는 데 필요한 최소한의 권한을 부여합니다.

Example Amazon S3 객체를 데이터 세트로 사용하는 IAM 정책

다음 예제에서는 *myBucket*이라는 Amazon S3 버킷의 *processImages* 내에 구성된 객체에 액세스할 수 있는 최소 권한을 부여하는 IAM 정책을 보여줍니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket"
```

```
            ],
            "Condition": {
                "StringLike": {
                    "s3:prefix": [
                        "processImages"
                    ]
                }
            }
        }
    ]
}
```

Example CSV 파일을 데이터 세트로 사용하는 IAM 정책

다음 예제에서는 *ratings.csv*라는 CSV 파일에 액세스할 수 있는 최소 권한을 부여하는 IAM 정책을 보여줍니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::myBucket/csvDataset/ratings.csv"
            ]
        }
    ]
}
```

Example Amazon S3 인벤토리를 데이터 세트로 사용하는 IAM 정책

다음 예제에서는 Amazon S3 인벤토리 보고서에 액세스할 수 있는 최소 권한을 부여하는 IAM 정책을 보여줍니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::destination-prefix/source-bucket/config-ID/YYYY-MM-DDTHH-MMZ/manifest.json",
                "arn:aws:s3:::destination-prefix/source-bucket/config-ID/data/*"
            ]
        }
    ]
}
```

Amazon S3 버킷에 데이터를 쓰기 위한 IAM 정책 예제

다음 IAM 정책 예제에서는 PutObject API 작업을 사용하여 하위 워크플로 실행 결과를 Amazon S3 버킷의 *csvJobs* 폴더에 쓰는 데 필요한 최소 권한을 부여합니다.

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

AWS KMS key 암호화된 Amazon S3 버킷에 대한 IAM 권한

Distributed Map 상태는 멀티파트 업로드를 사용하여 하위 워크플로 실행 결과를 Amazon S3 버킷에 씁니다. AWS Key Management Service (AWS KMS) 키를 사용하여 버킷을 암호화하는 경우 IAM 정책에 키에서kms:Decrypt, kms:Encrypt 및 kms:GenerateDataKey 작업을 수행할 수 있는 권한도 포함해야 합니다. 이러한 권한이 필요한 이유는 Amazon S3이 멀티파트 업로드를 완료하기 전에 암호화된 파일 부분에서 데이터를 암호 해독하고 읽어야 하기 때문입니다.

다음 IAM 정책 예제에서는 Amazon S3 버킷을 암호화하는 데 사용되는 키에 대한 `kms:Decrypt`, `kms:Encrypt` 및 `kms:GenerateDataKey` 작업에 권한을 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:Encrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": [
      "arn:aws:kms:us-east-1:123456789012:key/111aa2bb-333c-4d44-5555-a111bb2c33dd"
    ]
  }
}
```

자세한 내용은 AWS 지식 센터의 AWS KMS key를 사용한 암호화를 통해 Amazon S3에 대용량 파일 업로드를 참조하십시오.

IAM 사용자 또는 역할이 AWS 계정 동일한 경우 키 KMS key 정책에 대해 이러한 권한이 있어야 합니다. IAM 사용자 또는 역할이 KMS key와 다른 계정에 속한 경우 키 정책과 IAM 사용자 또는 역할 모두에 대한 권한이 있어야 합니다.

태그 기반 정책

Step Functions는 태그를 기반으로 하는 정책을 지원합니다. 예를 들어 `environment` 키 및 `production` 값과 함께 태그가 포함된 모든 Step Functions 리소스에 대한 액세스를 제한할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "states:TagResource",
                "states:UntagResource",
                "states:DeleteActivity",
                "states:DeleteStateMachine",
```

```
                "states:StopExecution"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/environment": "production"}
            }
        }
    ]
}
```

이 정책은 environment/production으로 태그 지정된 모든 리소스에 대해 상태 시스템 또는 활동을 삭제하고, 실행을 중지하며, 새 태그를 추가하거나 삭제할 수 있는 기능을 거부(Deny)합니다.

태그 기반 권한 부여의 경우 다음 예제와 같은 상태 시스템 실행 리소스는 상태 시스템과 연결된 태그를 상속합니다.

```
arn:<partition>:states:<Region>:<account-id>:execution:<StateMachineName>:<ExecutionId>
```

실행 리소스 ARN을 지정하는 다른 API DescribeExecution또는 호출 시 Step Functions는 태그 기반 인증을 수행하는 동안 상태 머신과 연결된 태그를 사용하여 요청을 수락하거나 거부합니다. 이렇게 하면 상태 시스템 수준에서 상태 시스템 실행에 대한 액세스를 허용하거나 거부할 수 있습니다.

태그 지정에 대한 자세한 내용은 다음을 참조하십시오.

- Step Functions에서 태그 지정
- IAM 태그를 사용한 액세스 제어

ID 및 액세스 문제 해결 AWS Step Functions

다음 정보를 사용하면 Step Functions 및 IAM을 사용할 때 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- Step Functions에서 작업을 수행할 수 있는 권한이 없음
- 저는 IAM을 수행할 권한이 없습니다. PassRole
- 외부 사용자가 내 Step Functions AWS 계정 리소스에 액세스할 수 있도록 허용하고 싶습니다.

Step Functions에서 작업을 수행할 수 있는 권한이 없음

작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음 예제 오류는 mateojackson 사용자가 콘솔을 사용하여 가상 *my-example-widget* 리소스에 대한 세부 정보를 보려고 하지만 가상 states:*GetWidget* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 states:GetWidget on resource: my-example-widget
```

이 경우 Mateo의 정책은 states:*GetWidget* 작업을 사용하여 *my-example-widget* 리소스에 액세스하도록 허용하도록 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

저는 IAM을 수행할 권한이 없습니다. PassRole

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 역할을 Step Functions에 전달할 수 있도록 정책을 업데이트해야 합니다.

새 서비스 역할 또는 서비스 연결 역할을 만드는 대신 기존 역할을 해당 서비스에 전달할 AWS 서비스 수 있는 기능도 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 오류 예제는 marymajor라는 IAM 사용자가 콘솔을 사용하여 Step Functions에서 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 Step Functions AWS 계정 리소스에 액세스할 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 제

어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- Step Functions에서 이러한 기능을 지원하는지 여부를 알아보려면 IAM과의 AWS Step Functions 작동 방식 섹션을 참조하세요.
- 소유한 리소스에 대한 액세스 권한을 AWS 계정 부여하는 방법을 알아보려면 IAM 사용 설명서의 다른 AWS 계정 IAM 사용자에게 액세스 권한 제공을 참조하십시오.
- 제3자에게 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정AWS 계정 소유에 대한 액세스 제공을 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 액세스를 참조하십시오.

로깅 및 모니터링

로그인 및 모니터링에 AWS Step Functions대한 자세한 내용은 섹션을 참조하십시오. 로깅 및 모니터링

규정 준수 검증: AWS Step Functions

제3자 감사자는 여러 규정 AWS 준수 프로그램의 AWS Step Functions 일환으로 보안 및 규정 준수를 평가합니다. 여기에는 SOC, PCI, FedRAMP, HIPAA 등이 포함됩니다.

특정 규정 준수 프로그램 범위 내 AWS 서비스 목록은 규정 준수 프로그램별 범위 내AWS 서비스 규정 준수 참조하십시오. 일반 정보는 AWS 규정 준수 프로그램AWS 보증 프로그램 규정AWS 참조하십시오.

를 사용하여 AWS Artifact타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html 참조하십시오 AWS Artifact.

Step Functions를 사용할 때의 규정 준수 책임은 데이터 민감도, 회사의 규정 준수 목표, 관련 법률과 규정에 따라 결정됩니다. AWS 는 규정 준수를 지원하기 위해 다음과 같은 리소스를 제공합니다.

- 보안 및 규정 준수 퀵 스타트 가이드 - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다. AWS
- Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계 — 이 백서에서는 기업이 HIPAA 준수 애플리케이션을 만드는 AWS 데 사용할 수 있는 방법을 설명합니다.
- AWS 규정 준수 리소스 — 이 통합 문서 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- AWS Config 개발자 안내서의 규칙을 통한 리소스 평가 — 이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub— 이 AWS 서비스는 보안 업계 표준 및 모범 사례를 준수하는지 확인하는 데 도움이 AWS 되는 내부 보안 상태를 종합적으로 보여줍니다.

레질리언스: AWS Step Functions

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 네트워킹으로 연결됩니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 지역 및 가용 영역에 대한 자세한 내용은 글로벌 인프라를 참조하십시오AWS .

AWS 글로벌 인프라 외에도 Step Functions는 데이터 복원력 및 백업 요구 사항을 지원하는 데 도움이 되는 여러 기능을 제공합니다.

의 인프라 보안 AWS Step Functions

관리형 서비스로서 AWS 글로벌 네트워크 보안으로 AWS Step Functions 보호됩니다. AWS 보안 서비스 및 인프라 AWS 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하십시오. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 Security Pillar AWS Well-Architected Framework의 인프라 보호를 참조하십시오.

AWS 게시된 API 호출을 사용하여 네트워크를 통해 액세스할 Step Functions 수 있습니다. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 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 인증을 생성하여 요청에 서명할 수 있습니다.

모든 네트워크 위치에서 AWS API 작업을 호출할 수 있지만 소스 IP 주소에 따른 제한을 포함할 수 있는 리소스 기반 액세스 정책은 Step Functions 지원하지 않습니다. 또한 Step Functions 정책을 사용하여 특정 Amazon Virtual Private Cloud (Amazon VPC) 엔드포인트나 특정 VPC의 액세스를 제어할 수 있습니다. 이를 통해 네트워크 내의 AWS 특정 VPC로부터 특정 Step Functions 리소스에 대한 네트워크 액세스를 효과적으로 분리할 수 있습니다.

의 구성 및 취약성 분석 AWS Step Functions

구성 및 IT 제어는 귀하와 당사 고객 간의 AWS 공동 책임입니다. 자세한 내용은 AWS 공동 책임 모델을 참조하십시오.

에서 Step AWS Data Pipeline Functions로 워크로드 마이그레이션

AWS 2012년에 AWS Data Pipeline 서비스를 시작했습니다. 당시 고객은 다양한 컴퓨팅 옵션을 사용하여 서로 다른 데이터 소스 간에 데이터를 이동할 수 있는 서비스를 원했습니다. 시간이 경과하면서 데이터 전송 요구 사항이 변함에 따라 이러한 요구 사항에 대한 솔루션도 변했습니다. 이제 비즈니스 요구 사항에 가장 부합하는 솔루션을 선택할 수 있습니다. 예를 들어 다음 중 하나를 수행할 수 있습니다.

- Step Functions를 사용하여 워크플로를 여러 AWS 서비스 사이에서 오케스트레이션합니다.
- Amazon Managed Workflows for Apache Airflow(Amazon MWAA)를 사용하여 Apache Airflow의 워크플로 오케스트레이션을 관리합니다.
- Apache Spark 애플리케이션을 실행하고 AWS Glue 오케스트레이션하는 데 사용합니다.

일반적인 사용 사례는 Step AWS Data Pipeline Functions 또는 AWS Glue Amazon MWAA로 마이그레이션할 수 있습니다. 선택한 옵션은 현재 AWS Data Pipeline의 워크로드에 따라 다릅니다. 이 항목에서는 에서 Step Functions로 AWS Data Pipeline 마이그레이션하는 방법을 설명합니다.

주제

AWS Data Pipeline에서 워크로드 마이그레이션

Step Functions는 중요 비즈니스용 애플리케이션의 워크플로를 빌드하는 서버리스 오케스트레이션 서비스입니다. Step Functions의 Workflow Studio를 사용하면 워크플로를 빌드하고 AWS 서비스 250개 이상에서 11,000개가 넘는 API 작업과 통합할 수 있습니다. 여기에는 AWS 서비스 Amazon EMR AWS Lambda, Amazon DynamoDB 등이 포함됩니다. 또한 Step Functions를 사용하여 데이터 처리 파이프라인을 오케스트레이션하고 오류를 처리하며 기본 AWS 서비스에서 한도 제한을 사용할 수 있습니다. 기계 학습 모델을 처리 및 게시하고 마이크로서비스를 오케스트레이션하고 AWS Glue를 사용하여 추출, 전환, 적재(ETL) 워크플로를 처리하는 워크플로를 만들 수 있습니다. 또한 사람의 상호 작용이 필요한 애플리케이션을 위해 장기간 실행되는 자동화된 워크플로를 만들 수 있습니다.

Step Functions는 AWS에서 제공하는 완전 관리형 서비스입니다. 즉, 인프라 유지, 작업자 패치 적용 및 OS 버전 업데이트 관리와 같은 작업을 AWS 에서 관리합니다.

사용 사례가 다음 조건과 일치하는 경우 에서 Step Functions로 AWS Data Pipeline 마이그레이션하는 것이 좋습니다.

- 가용성이 높은 서버리스 워크플로 오케스트레이션 서비스를 선호합니다.
- 단일 작업 실행의 세분화로 요금이 청구되는 솔루션이 필요합니다.
- 워크로드에는 Amazon EMR AWS 서비스, Lambda 또는 DynamoDB와 같은 기타 여러 작업을 오케스트레이션하는 작업이 포함됩니다. AWS Glue
- 워크플로 생성을 위해서는 비주얼 디자이너가 포함된 로우코드 솔루션이 필요합니다. drag-and-drop 이 솔루션을 위해 익숙하지 않은 복잡한 프로그래밍 개념을 배울 필요가 없습니다.
- 11,000개 이상의 API 작업을 AWS 서비스 지원하는 250개 이상의 서비스와 통합되는 서비스가 필요합니다. 또한 이 서비스는 사용자 지정 서비스 및 외부 활동과도 통합되어야 합니다. AWS

Step Functions 및 AWS Data Pipeline 간의 개념 매핑

AWS Data Pipeline 및 Step Functions는 몇 가지 공통된 개념을 공유합니다. 예를 들어 워크플로를 정의하려면 Step AWS Data Pipeline Functions와 Step Functions에서 모두 JSON 형식을 사용합니다. Step Functions에서는 JSON 기반 구조화된 언어인 Amazon States Language를 사용합니다. Amazon States Language(ASL)를 사용하여 워크플로를 정의하고 워크플로의 텍스트 표현이나 시각적 표현으로 전환할 수 있습니다. 이 JSON 기반 형식을 사용하면 워크플로를 간단하게 소스 제어 도구에 저장할 수 있습니다. 또한 여러 버전의 워크플로를 관리하거나 액세스를 제어하거나 CI/CD 방식으로 오케스트레이션을 자동화할 수 있습니다.

다음 표에서는 두 서비스 모두에서 사용되는 주요 개념 간의 매핑을 설명합니다. 왼쪽의 데이터 파이프라인 개념 열에는 의 개념이 나열되어 있고 오른쪽의 Step Functions 개념 열에는 Step Functions의 해당 개념이 나열되어 있습니다. AWS Data Pipeline

데이터 파이프라인 개념	Step Functions 개념
파이프라인	워크플로
파이프라인 정의	Amazon States Language(ASL)
활동	상태 및 태스크 상태

데이터 파이프라인 개념	Step Functions 개념
인스턴스	실행
Attempts	Catcher 및 Retrier
파이프라인 일정	• Amazon EventBridge 스케줄러를 사용한 실행 • 파이프를 통해 트리거된 이벤트 EventBridge
파이프라인 표현식 및 함수	• 내장 함수 • 서비스 통합을 사용하는 Lambda 함수

Step Functions 샘플 프로젝트

Step Functions 소개는 다음 비디오를 참조하세요.

서비스 오케스트레이션을 AWS Step Functions 위한 시작하기

다음 목록에서는 Step Functions를 사용하여 가장 일반적인 AWS Data Pipeline 사용 사례를 구현하는 몇 가지 샘플 프로젝트를 간략하게 설명합니다. 이 샘플 프로젝트를 참조로 사용하여 Step Functions로 AWS Data Pipeline 마이그레이션할 수 있습니다. 또한 이를 상용구로 사용하여 자체 워크플로를 빌드하고 사용 사례에 따라 지원되는 AWS 서비스와 통합할 수 있습니다.

- Amazon EMR 작업 관리
- Amazon EMR Serverless에서 데이터 처리 작업 실행
- Hive/Pig/Hadoop 작업 실행
- 대규모 데이터세트 쿼리 (아마존 아테나, 아마존 S3 AWS Glue, 아마존 SNS)
- Amazon Redshift를 사용하여 ETL/ELT 워크플로 실행
- 크롤러 오케스트레이션 AWS Glue
- Step Functions를 사용하여 쉘 스크립트 실행

Step Functions에 대한 자세한 내용은 다음 주제와 리소스를 참조하세요.

- Step Functions 자습서
- Step Functions를 위한 샘플 프로젝트

- AWS Step Functions 워크숍

요금 비교

AWS Data Pipeline 파이프라인 수와 사용 수준에 따라 가격이 책정됩니다. 하루에 2회 이상 실행되는 활동(높은 빈도)에는 활동당 매월 $1 요금이 청구됩니다. 하루에 1회 미만으로 실행되는 활동(낮은 빈도)에는 활동당 매월 $0.60 요금이 청구됩니다. 비활성 파이프라인 요금은 파이프라인당 $1입니다. 요금에 대한 자세한 내용은 AWS Data Pipeline 요금 페이지를 참조하세요.

Step Functions에는 표준 및 Express 등 두 가지 유형의 워크플로가 있습니다. 워크플로 유형마다 요금 모델이 다릅니다. 이 비교는 표준 워크플로를 기반으로 합니다. 표준 워크플로우가 일반적인 사용 사례와 가장 잘 일치하기 때문입니다. AWS Data Pipeline 표준 워크플로 요금은 상태 전환 1,000회당 $0.025입니다. 비활성 상태 시스템에는 요금이 청구되지 않으며 사용한 만큼만 요금을 지불하면 됩니다. 요금에 대한 자세한 내용은 AWS Step Functions 요금 페이지를 참조하세요.

문제 해결

Step Functions를 사용할 때 문제가 발생하면 다음 문제 해결 리소스를 사용합니다.

주제

일반 문제 해결

상태 시스템을 만들 수 없습니다.

상태 시스템과 연결된 IAM 역할에 충분한 권한이 없을 수 있습니다. AWS 서비스 통합 작업, X-Ray 및 CloudWatch 로깅을 포함하여 IAM 역할 권한을 확인합니다. `.sync` 작업 상태에 추가 권한이 필요합니다.

JsonPath를 사용하여 이전 작업의 출력을 참조할 수 없습니다.

JsonPath의 경우 JSON 키는 `.$`로 끝나야 합니다. 즉, 키-값 페어에서만 JsonPath를 사용할 수 있습니다. 배열과 같은 다른 위치에서 JsonPath를 사용하려는 경우에는 내장 함수를 사용하면 됩니다. 예를 들어 다음과 비슷한 코드를 사용할 수 있습니다.

작업 A 출력:

```
{
    "sample": "test"
}
```

작업 B:

```
{
    "JsonPathSample.$": "$.sample"
}
```

> **Tip**
>
> Step Functions 콘솔의 데이터 흐름 시뮬레이터를 사용하면 JSON 경로 구문을 테스트하고 상태 내에서 데이터가 조작되는 방식을 더 잘 이해하며 데이터가 상태 간에 전달되는 방식을 확인할 수 있습니다.

상태 전환이 지연되었습니다.

표준 워크플로의 경우 상태 전환 수에 제한이 있습니다. 상태 전환 한도를 초과하면 Step Functions는 할당량 버킷이 채워질 때까지 상태 전환을 지연합니다. CloudWatch Metrics 페이지의 실행 지표 섹션에 있는 `ExecutionThrottled` 지표를 검토하여 상태 전환 한도 제한을 모니터링할 수 있습니다.

새로운 표준 워크플로 실행을 시작하면 **`ExecutionLimitExceeded`** 오류가 발생하면서 실패합니다.

Step Functions에서는 각 AWS 리전에서 AWS 계정마다 열린 실행이 1,000,000개로 제한됩니다. 이 한도를 초과하면 Step Functions에서 `ExecutionLimitExceeded` 오류가 발생합니다. Express 워크플로에는 이 한도가 적용되지 않습니다. Amazon CloudWatch 사용 설명서에 있는 다음 CloudWatch Metrics 수학(`ExecutionsStarted - (ExecutionsSucceeded + ExecutionsTimedOut + ExecutionsFailed + ExecutionsAborted)`)을 사용하여 열린 실행 수를 대략적으로 계산할 수 있습니다.

Parallel 상태의 한 브랜치에 실패가 발생하면 전체 실행이 실패합니다.

이는 예상된 동작입니다. Parallel 상태를 사용할 때 실패를 방지하려면 각 브랜치에서 발생한 오류를 포착하도록 Step Functions를 구성합니다.

서비스 통합 문제 해결

다운스트림 서비스에서는 작업이 완료되었지만 Step Functions에서는 Task 상태가 “진행 중”으로 유지되거나 완료가 지연됩니다.

`.sync` 서비스 통합 패턴의 경우 Step Functions는 EventBridge 규칙, 다운스트림 API 또는 이 둘의 조합을 사용하여 다운스트림 작업 상태를 감지합니다. 일부 서비스의 경우 Step Functions는 모니터링할 EventBridge 규칙을 만들지 않습니다. 예를 들어 AWS Glue 서비스 통합의 경우 EventBridge 규칙

을 사용하는 대신 Step Functions에서 `glue:GetJobRun`을 직접적으로 호출합니다. API 직접 호출 빈도로 인해 다운스트림 작업 완료 시간과 Step Functions 작업 완료 시간 간에 차이가 있습니다. Step Functions를 사용하려면 EventBridge 규칙을 관리하고 다운스트림 서비스를 직접적으로 호출할 수 있는 IAM 권한이 필요합니다. 실행 역할에 대한 권한 부족이 작업 완료에 미치는 영향에 대한 자세한 내용은 작업 실행 패턴을 사용하는 작업에 대한 추가 권한을 참조하세요.

중첩된 상태 시스템 실행에서 JSON 출력을 반환하려고 합니다.

Step Functions를 위한 Step Functions 동기 서비스 통합에는 `startExecution.sync` 및 `startExecution.sync:2` 등 2개가 있습니다. 둘 다 중첩된 상태 시스템이 완료될 때까지 기다리지만 다른 `Output` 형식을 반환합니다. `startExecution.sync:2`를 사용하여 `Output`에서 JSON 출력을 반환할 수 있습니다.

다른 계정에서 Lambda 함수를 간접적으로 호출할 수 없습니다.

크로스 계정 지원을 통해 Lambda 함수에 액세스

리전에서 AWS 리소스의 크로스 계정 액세스를 사용할 수 있으면 다음 방법을 사용하여 다른 계정에서 Lambda 함수를 간접적으로 호출합니다.

워크플로에서 크로스 계정 리소스를 간접적으로 호출하려면 다음을 수행합니다.

1. 리소스가 포함된 대상 계정에 IAM 역할을 만듭니다. 이 역할은 상태 시스템이 포함된 소스 계정에 대상 계정 리소스에 액세스할 수 있는 권한을 부여합니다.
2. `Task` 상태 정의에서 크로스 계정 리소스를 간접적으로 호출하기 전에 상태 시스템에서 위임할 대상 IAM 역할을 지정합니다.
3. 소스 계정에서 일시적으로 이 역할을 위임할 수 있도록 대상 IAM 역할의 신뢰 정책을 수정합니다. 신뢰 정책에는 소스 계정에 정의된 상태 시스템의 Amazon 리소스 이름(ARN)이 포함되어야 합니다. 또한 AWS 리소스를 직접적으로 호출하도록 대상 IAM 역할에서 적절한 권한을 정의합니다.
4. 대상 IAM 역할을 위임하는 데 필요한 권한이 포함되도록 소스 계정 실행 역할을 업데이트합니다.

예시는 자습서: 교차 AWS 계정 리소스 액세스에서 확인하세요.

> **Note**
>
> 여러 AWS 계정의 리소스에 액세스하기 위해 IAM 역할을 위임하도록 상태 시스템을 구성할 수 있습니다. 하지만 상태 시스템은 한 번에 IAM 역할 하나만 위임할 수 있습니다.

크로스 계정 리소스를 지정하는 Task 상태 정의의 예제는 Task 상태의 보안 인증 정보 필드 예제를 참조하세요.

크로스 계정 지원 없이 Lambda 함수에 액세스

리전에서 AWS 리소스의 크로스 계정 액세스를 사용할 수 없으면 다음 방법을 사용하여 다른 계정에서 Lambda 함수를 간접적으로 호출합니다.

Task 상태의 Resource 필드에서 arn:aws:states:::lambda:invoke를 사용하고 FunctionArn을 파라미터에 전달합니다. 상태 시스템과 연결된 IAM 역할에는 크로스 계정 Lambda 함수를 간접적으로 호출할 수 있는 올바른 권한(lambda:invokeFunction)이 있어야 합니다.

```
{
   "StartAt":"CallLambda",
   "States":{
      "CallLambda":{
         "Type":"Task",
         "Resource":"arn:aws:states:::lambda:invoke",
         "Parameters":{
            "FunctionName":"arn:aws:lambda:us-west-2:123456789012:function:my-function"
         },
         "End":true
      }
   }
}
```

.waitForTaskToken 상태에서 전달된 작업 토큰을 확인할 수 없습니다.

Task 상태의 Parameters 필드에서는 작업 토큰을 전달해야 합니다. 예를 들어 다음 코드와 비슷한 코드를 사용할 수 있습니다.

```
{
   "StartAt":"taskToken",
   "States":{
      "taskToken":{
         "Type":"Task",
         "Resource":"arn:aws:states:::lambda:invoke.waitForTaskToken",
         "Parameters":{
            "FunctionName":"get-model-review-decision",
            "Payload":{
               "token.$":"$$.Task.Token"
```

```
            },
         },
         "End":true
      }
   }
}
```

> **Note**
>
> 모든 API 작업에서 `.waitForTaskToken`을 사용해 볼 수 있습니다. 하지만 일부 API에는 적절한 파라미터가 없습니다.

활동 문제 해결

상태 시스템 실행이 활동 상태에서 멈췄습니다.

GetActivityTask API 작업을 사용하여 작업 토큰을 폴링할 때까지는 활동 작업 상태가 시작되지 않습니다. 실행 중단을 방지하기 위해 작업 수준 제한 시간을 추가하는 것이 좋습니다. 자세한 내용은 제한 시간을 사용하여 실행 멈춤 방지 섹션을 참조하세요.

상태 시스템이 ActivityScheduled 이벤트에서 멈추는 경우 이는 활동 작업자 플릿에 문제가 있거나 규모가 작았다는 의미입니다. `ActivityScheduleTime` CloudWatch 지표를 모니터링하고 시간이 길어지면 경보를 설정해야 합니다. 하지만 `Activity` 상태가 `ActivityStarted` 상태로 전환되지 않는 중단된 상태 시스템 실행 시간을 초과하려면 상태 시스템 수준에서 제한 시간을 정의합니다. 이렇게 하려면 상태 시스템 정의의 시작 부분, 즉 `TimeoutSeconds` 필드 외부에 `States` 필드를 지정합니다.

작업 토큰을 기다리는 동안 활동 작업자 시간이 초과되었습니다.

작업자는 GetActivityTask API 작업을 사용하여 실행 중인 상태 시스템에서 실행을 예약한 지정된 활동 ARN이 있는 작업을 검색합니다. `GetActivityTask`에서 긴 폴링을 시작하므로 서비스는 HTTP 연결을 열린 상태로 유지하고 작업을 사용할 수 있게 되는 즉시 응답합니다. 서비스가 응답하기 전에 요청을 보류하는 최대 시간은 60초입니다. 60초 내에 사용할 수 있는 작업이 없으면 폴링에서 null 문자열이 포함된 `taskToken`을 반환합니다. 이러한 제한 시간을 방지하려면 API 직접 호출에 사용하는 AWS SDK 또는 클라이언트에서 타임아웃을 65초 이상으로 설정하여 클라이언트 측 소켓을 구성합니다.

Express 워크플로 문제 해결

StartSyncExecution API 직접 호출에서 응답을 받기 전에 애플리케이션 시간이 초과되었습니다.

API 직접 호출에 사용하는 AWS SDK 또는 클라이언트에서 클라이언트 측 소켓 제한 시간을 구성합니다. 응답을 받으려면 제한 시간 값이 Express 워크플로 실행 기간보다 커야 합니다.

Express 워크플로 실패 문제를 해결하기 위해 실행 내역을 볼 수 없습니다.

Express 워크플로는 AWS Step Functions의 실행 내역을 기록하지 않습니다. 대신 CloudWatch 로깅을 활성화해야 합니다. 로깅이 활성화되면 CloudWatch Logs Insights 쿼리를 사용하여 Express 워크플로 실행을 검토할 수 있습니다. 실행 탭에서 활성화 버튼을 선택하면 Step Functions 콘솔에서 Express 워크플로 실행의 실행 내역을 볼 수도 있습니다. 자세한 내용은 Step Functions 콘솔에서 실행 보기 및 디버깅 섹션을 참조하세요.

기간을 기준으로 실행 나열하기:

```
fields ispresent(execution_arn) as exec_arn
| filter exec_arn
| filter type in ["ExecutionStarted", "ExecutionSucceeded", "ExecutionFailed",
 "ExecutionAborted", "ExecutionTimedOut"]
| stats latest(type) as status,
  tomillis(earliest(event_timestamp)) as UTC_starttime,
  tomillis(latest(event_timestamp)) as UTC_endtime,
  latest(event_timestamp) - earliest(event_timestamp) as duration_in_ms  by
 execution_arn
| sort duration desc
```

실패 및 취소된 실행 나열하기:

```
fields ispresent(execution_arn) as isRes | filter type in ["ExecutionFailed",
 "ExecutionAborted", "ExecutionTimedOut"]
```

관련 정보

다음 표에는 이 서비스를 사용할 때 유용하게 참조할 수 있는 관련 리소스가 나열되어 있습니다.

리소스	설명
AWS Step Functions API 참조	API 작업, 파라미터, 데이터 유형 등의 설명 및 반환하는 오류 목록.
AWS Step Functions 명령줄 참조	AWS Step Functions에 사용할 수 있는 AWS CLI 명령에 대한 설명입니다.
Step Functions 제품 정보	Step Functions에 대한 정보를 제공하는 기본 웹 페이지입니다.
토론 포럼	Step Functions 및 기타 AWS 서비스와 관련된 기술 문제를 토론할 수 있는 커뮤니티 기반의 개발자 포럼입니다.
AWS Support 정보	AWS Support에 대한 정보를 제공하는 기본 웹 페이지로, 일대일의 신속한 응대 지원 채널을 통해 AWS 인프라 서비스에서 애플리케이션을 빌드하고 실행하도록 지원합니다.

최근 기능 출시

다음 표에는 새로운 Step Functions 기능을 사용할 수 있는 리전이 나와 있습니다.

시작 날짜	기능 이름	사용 가능한 리전
2023년 11월 26일	퍼블릭 HTTPS 엔드포인트 호출 및 개별 상태 테스트	• 미국 동부(버지니아 북부) - us-east-1 • 미국 서부(오레곤) - us-west-2 • 미국 동부(오하이오) - us-east-2 • 유럽(아일랜드) - eu-west-1 • 유럽(프랑크푸르트) - eu-central-1 • 유럽(스톡홀름) - eu-north-1 • 아시아 태평양(시드니) - ap-southeast-2 • 아시아 태평양(도쿄) - ap-northeast-1 • 아시아 태평양(싱가포르) - ap-southeast-1
2023년 11월 15일	Redrive 실행	이 기능을 사용할 수 AWS 리전의 전체 목록은 리전별 AWS 서비스 페이지의 리전 드롭다운 목록에 있는 옵션을 참조하십시오.
2023년 10월 12일	최적화된 Amazon EMR Serverless 통합	이 기능을 사용할 수 AWS 리전의 전체 목록은 리전별 AWS 서비스 페이지의 리전 드롭다운 목록에 있는 옵션을 참조하십시오.

시작 날짜	기능 이름	사용 가능한 리전
2023년 9월 7일	향상된 오류 처리	이 기능을 사용할 수 AWS 리전의 전체 목록은 리전별 AWS 서비스 페이지의 리전 드롭다운 목록에 있는 옵션을 참조하십시오.
2023년 8월 31일	간소화된 작성 환경을 위한 Workflow Studio 개선 사항	이 기능을 사용할 수 AWS 리전의 전체 목록은 리전별 AWS 서비스 페이지의 리전 드롭다운 목록에 있는 옵션을 참조하십시오.
2023년 6월 22일	버전 및 별칭	이 기능을 사용할 수 AWS 리전의 전체 목록은 리전별 AWS 서비스 페이지의 리전 드롭다운 목록에 있는 옵션을 참조하십시오.
2023년 6월 16일	새 AWS SDK 통합	이 기능을 사용할 수 AWS 리전의 전체 목록은 리전별 AWS 서비스 페이지의 리전 드롭다운 목록에 있는 옵션을 참조하십시오.
2022년 12월 1일	Distributed Map 상태 상태를 사용하여 데이터를 처리하기 위한 대규모 병렬 워크플로 오케스트레이션	이 기능을 사용할 수 AWS 리전의 전체 목록은 리전별 AWS 서비스 페이지의 리전 드롭다운 목록에 있는 옵션을 참조하십시오.

문서 기록

이 섹션에는 AWS Step Functions 개발자 안내서에 대한 주요 변경 사항이 나와 있습니다.

변경 사항	설명	변경 날짜
업데이트	AWS 관리형 정책 업데이트 - 새 권한: `states:ValidateStateMachineDefinition` 제공하는 상태 머신의 구문을 확인하기 위한 새 권한에 대한 정보가 추가되었습니다. 자세한 내용은 AWS 관리형 정책 대상 AWS Step Functions 섹션을 참조하세요.	2024년 4월 29일
새 기능	Step Functions는 다음과 같은 환경에 최적화된 통합을 추가합니다. AWS Elemental MediaConvert AWS Elemental MediaConvert 고객이 미디어 워크플로우에 적합한 코드를 사용하여 자동화할 수 있는 브로드캐스트 수준의 비디오 및 오디오 파일 트랜스코딩을 제공합니다. in을 위한 최적화된 통합을 통해 이제 로우 코드 AWS Step Functions 비주얼 MediaConvert 도구인 Workflow Studio를 사용하여 오케스트레이션할 수 있습니다. 자세한 내용은 Step Functions를 AWS Elemental MediaConvert 사용한 관리 설명서를 참조하십시오.	2024년 4월 12일
업데이트	AWS 관리형 정책 업데이트 - 기존 정책 업데이트: `AWSStepFunctionsReadOnlyAccess` 태그, 분산 맵, 버전 및 별칭에 대한 새로운 읽기 전용 권한에 대한 정보가 추가되었습니다. 자세한 내용은 AWS 관리형 정책 대상 AWS Step Functions 섹션을 참조하세요.	2024년 4월 2일
업데이트	Step Functions는 오픈 워크플로 지표에 대한 지원을 추가합니다. 개방형 워크플로 지표를 통해 이제 진행 중인 표준 워크플로의 수와 공개 워크플로 한도를 계정 수준에서 파악할 수 있습니다. 시작 방법에 관계없이 모든 워크플로의 워크로드	2024년 2월 29일

변경 사항	설명	변경 날짜
	를 관리하여 원활한 워크플로 운영을 보장할 수 있습니다. CloudWatch 경보를 설정하여 워크플로를 모니터링하고 한도에 가까워지면 사전에 알림을 받을 수 있습니다. 알림을 받으면 특정 워크플로를 중지하거나 한도 증가를 요청하는 등의 조치를 취하여 워크플로를 효과적으로 관리할 수 있습니다. 개방형 워크플로 지표는 추가 구성 없이 표준 CloudWatch 워크플로에 사용할 수 있습니다. 자세한 내용은 실행 지표 섹션을 참조하세요.	
업데이트	서비스 통합 추가 및 업데이트. 신규 및 업데이트된 AWS SDK 통합 목록은 을 참조하십시오. 지원되는 AWS SDK 통합에 대한 변경 로그 전체 서비스 목록은 을 참조하십시오. 지원되는 AWS SDK 서비스 통합	2024년 1월 18일
새 기능	Application Composer의 Workflow Studio를 사용하여 AWS CloudFormation 템플릿으로 서버리스 워크플로를 구축할 수 있습니다. 자세한 정보는 Application Composer에서 Workflow Studio 사용을 참조하세요.	2023년 11월 27일
새 기능	이제 Step Functions는 퍼블릭 HTTPS 엔드포인트를 직접 호출하고 새 테스트 상태 API를 사용하여 개별 상태를 테스트할 수 있습니다. 자세한 내용은 다음을 참조하세요. • 타사 API 호출 • TestState API를 사용하여 상태 테스트	2023년 11월 26일
새 기능	Step Functions이 이제 Amazon Bedrock과 통합되었습니다. 자세한 정보는 다음 주제를 참조하세요. • Step Functions 사용을 통한 Amazon Bedrock 호출 • IAM에 대한 Amazon Bedrock 권한 • Amazon Bedrock을 사용하여 AI 프롬프트 체이닝 수행 • 다른 서비스와 AWS Step Functions 함께 사용	2023년 11월 26일

변경 사항	설명	변경 날짜
새 기능	이제 Step Functions를 사용하면 실패 지점부터 표준 유형의 redrive 워크플로를 실행할 수 있습니다. 자세한 내용은 실행 Redriving 및 맵 실행 Redriving 섹션을 참조하세요.	2023년 11월 15일
문서 전용 업데이트	Amazon EventBridge Scheduler를 사용하여 일정에 따라 상태 컴퓨터를 실행하는 방법을 설명하는 새 주제를 게시했습니다. 자세한 정보는 AWS Step Functions에서 Amazon EventBridge 스케줄러 사용을 참조하세요.	2023년 10월 16일
새로운 특성	Step Functions이 이제 Amazon EMR Serverless과 통합되었습니다. 자세한 정보는 다음 주제를 참조하세요. • Step Functions 사용을 통한 Amazon EMR Serverless 호출 • EMR Serverless 작업 실행 • Step Functions를 위한 최적화된 통합 • 다른 서비스와 AWS Step Functions 함께 사용	2023년 10월 12일
문서 전용 업데이트	Amazon EventBridge Scheduler를 사용하여 일정에 따라 상태 시스템을 실행하는 방법에 대한 정보가 추가되었습니다. 자세한 정보는 EventBridge 스케줄러 사용을 참조하세요.	2023년 10월 5일
업데이트	명확하고 간결하며 신규 사용자를 위한 명확한 여정 지도를 설정하기 위해 Distributed Map 상태 주제를 재구성하고 업데이트했습니다. 자세한 정보는 분산 모드에서 Map 상태를 사용하여 대규모 병렬 워크로드 오케스트레이션을 참조하세요.	2023년 10월 6일
수정 사항	AWS CDK v2를 사용하도록 자습서의 코드 샘플을 수정했습니다. 자세한 정보는 AWS CDK를 사용하여 Step Functions용 Lambda 상태 시스템 만들기을 참조하세요.	2023년 9월 19일
업데이트	오류를 명확하게 식별하고 보다 효과적으로 제어로 재시도를 구현하도록 Step Functions에 도입된 오류 처리 기능 개선 사항에 대한 정보가 추가되었습니다. 자세한 내용은 Fail 및 오류 후 재시도 섹션을 참조하세요.	2023년 9월 7일

변경 사항	설명	변경 날짜
업데이트	Step Functions에 워크플로 작성 환경을 간소화하기 위한 Workflow Studio 개선 사항이 추가되었습니다. 자세한 정보는 AWS Step Functions 워크플로 스튜디오을 참조하세요.	2023년 8월 31일
문서 전용 업데이트	`ExecutionsStarted` 지표에 대해 보고된 실제 지표 수의 약 2배에 해당하는 정보가 추가되었습니다. 자세한 정보는 개수를 보고하는 지표을 참조하세요.	2023년 7월 25일
문서 전용 업데이트	Step Functions에 다음과 같은 Distributed Map 상태의 일반적인 사용 사례를 보여주는 새 샘플 프로젝트 2개가 추가되었습니다. • CSV 파일 처리 • Amazon S3 버킷에서 데이터 처리	2023년 7월 17일
문서 전용 업데이트	Terraform을 사용한 상태 시스템 배포에 대한 새 주제를 게시했습니다. 자세한 정보는 Terraform을 사용하여 상태 시스템 배포을 참조하세요.	2023년 7월 5일
문서 전용 업데이트	Amazon EventBridge 인터페이스의 변경 사항과 일치하도록 다음 절차를 업데이트했습니다. • Step Functions 이벤트를 다음으로 라우팅하기 EventBridge • Amazon S3 Events에 대한 응답으로 상태 시스템 실행 시작	2023년 6월 26일
새 기능	이제 Step Functions에서 서버리스 워크플로를 배포하는 동안 복원력이 향상될 수 있도록 여러 상태 시스템 버전과 별칭을 만들 수 있는 기능을 제공합니다. 자세한 정보는 버전 및 별칭을 사용하여 지속적인 배포 관리을 참조하세요.	2023년 6월 22일
문서 전용 업데이트	`TimeoutSeconds` 및 `HeartbeatSeconds` 필드가 서로 어떻게 다른지 설명하기 위해 이러한 필드의 설명을 개선했습니다. 자세한 정보는 Task 상태 필드을 참조하세요.	2023년 6월 22일

변경 사항	설명	변경 날짜
문서 전용 업데이트	일반적으로 Parallel 및 Map 상태에 대한 결과로 반환된 배열의 배열을 평면화하는 방법을 설명하는 새 섹션을 게시했습니다. 자세한 정보는 배열의 배열 평면화을 참조하세요.	2023년 6월 20일
업데이트	Step Functions는 7개 AWS 서비스 및 468개의 새로운 API 작업을 추가하여 AWS SDK 통합에 대한 지원을 확대했습니다. 자세한 내용은 지원되는 AWS SDK 서비스 통합 및 지원되는 AWS SDK 통합에 대한 변경 로그 섹션을 참조하세요.	2023년 6월 16일
문서 전용 업데이트	최근에 출시된 Step AWS 리전 Functions 기능을 사용할 수 있는 기능을 나열하는 새 항목을 게시했습니다. 자세한 정보는 최근 기능 출시을 참조하세요.	2023년 6월 16일
문서 전용 업데이트	Step Functions에는 이제 에 대한 AWS 서비스 섹션이 포함되어 있으며 AWS 사용자 알림, 이 섹션은 AWS 알림을 위한 중앙 위치 역할을 AWS Management Console합니다. 자세한 정보는 Step Functions와 함께 AWS 사용자 알림 사용을 참조하세요.	2023년 5월 4일
문서 전용 업데이트	하위 워크플로 실행 결과를 AWS Key Management Service (AWS KMS) 키로 암호화된 Amazon S3 버킷에 쓰는 데 필요한 권한을 설명하는 새 섹션이 추가되었습니다. 자세한 정보는 AWS KMS key 암호화된 Amazon S3 버킷에 대한 IAM 권한을 참조하세요.	2023년 4월 29일
문서 전용 업데이트	데이터 흐름 시뮬레이터 기능을 설명하는 새 주제가 추가되었습니다. 자세한 정보는 데이터 흐름 시뮬레이터을 참조하세요.	2023년 4월 14일
할당량 업데이트	각 계정의 오픈 맵 실행 기본 할당량(1,000)에 대한 정보가 추가되었습니다. 자세한 정보는 계정과 관련된 할당량을 참조하세요.	2023년 4월 5일

변경 사항	설명	변경 날짜
문서 전용 업데이트	AWS Data Pipeline 워크로드를 Step Functions로 마이그레이션하는 시기를 설명하는 항목이 추가되었습니다. 이 주제에서는 마이그레이션 수행 방법을 설명하는 예제 목록도 제공합니다. 자세한 정보는 에서 Step AWS Data Pipeline Functions로 워크로드 마이그레이션을 참조하세요.	2023년 3월 30일
문서 전용 업데이트	Distributed Map 상태에 X-Ray 트레이싱을 사용할 수 없음에 대한 참고가 추가되었습니다. 자세한 정보는 AWS X-Ray 및 Step Functions을 참조하세요.	2023년 3월 21일
문서 전용 업데이트	Step Functions에서 태그 기반 권한 부여를 처리하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 Step Functions에서 태그 지정 및 태그 기반 정책 섹션을 참조하세요.	2023년 3월 15일
문서 전용 업데이트	Step Functions가 Distributed Map 상태에서 입력으로 사용되는 CSV 파일을 파싱하는 방법에 대한 정보가 추가되었습니다. 자세한 정보는 Amazon S3 버킷에 있는 CSV 파일을 참조하세요.	2023년 3월 14일
문서 전용 업데이트	Step Functions에서 작업 실행(.sync) 패턴에 대한 크로스 계정 간접 호출을 처리하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 작업 실행(.sync)을 참조하세요.	2023년 3월 1일
문서 전용 업데이트	완료된 워크플로 실행의 내역 보존 기간을 90일에서 30일로 단축합니다. 보존 기간 조정 방법에 대한 자세한 내용은 실행 보장 및 상태 시스템 실행과 관련된 할당량 섹션을 참조하세요.	2023년 2월 21일
업데이트	Step Functions는 35개의 AWS 서비스와 1100개의 새로운 API 작업을 추가하여 AWS SDK 통합에 대한 지원을 확대했습니다. 자세한 내용은 지원되는 AWS SDK 서비스 통합 및 지원되는 AWS SDK 통합에 대한 변경 로그 섹션을 참조하세요.	2023년 2월 17일

변경 사항	설명	변경 날짜
문서 전용 업데이트	Step Functions를 사용하여 신용 카드 신청 워크플로를 만드는 절차를 안내하는 시작하기 자습서 시리즈를 게시했습니다. 자세한 정보는 시작하기 AWS Step Functions을 참조하세요.	2022년 12월 30일
새 기능	Step Functions는 새로운 `Map` 상태의 분산 모드를 사용하여 데이터 처리를 위한 대규모 병렬 워크플로를 오케스트레이션하기 위한 지원을 추가합니다. 자세한 정보는 분산 모드에서 Map 상태를 사용하여 대규모 병렬 워크로드 오케스트레이션을 참조하세요.	2022년 12월 1일
새 기능	Step Functions는 이제 다른 계정에 구성된 교차 계정 AWS 리소스에 대한 액세스를 지원합니다. 자세한 내용을 알아보려면 다음 섹션을 참조하세요. • 워크플로의 다른 AWS 계정 리소스에 액세스 • 자습서: 교차 AWS 계정 리소스 액세스 • Task state	2022년 11월 18일
업데이트	이제 Step Functions에서 Express 워크플로 실행을 보고 디버깅할 수 있는 새로운 콘솔 환경을 제공합니다. 자세한 내용은 다음을 참조하세요. • 콘솔에서의 표준 및 Express 워크플로 실행 • Step Functions 콘솔에서 실행 보기 및 디버깅	2022년 10월 18일
업데이트	Amazon EMR 최적화 서비스 통합에 `addStep` 및 `addStep.sync` API를 사용하는 동안 `Execution RoleArn` 파라미터를 선택적으로 지정할 수 있는 지원이 추가되었습니다. 자세한 정보는 Step Functions를 사용하여 Amazon EMR 직접 호출을 참조하세요.	2022년 9월 20일
문서 전용 업데이트	Step Functions를 사용하여 서버리스 워크플로를 빌드하는 동안 비용을 최적화하는 방법에 대한 권장 사항을 제공하는 새 주제가 추가되었습니다. 자세한 정보는 Express 워크플로를 사용하여 비용 최적화을 참조하세요.	2022년 9월 15일

변경 사항	설명	변경 날짜
업데이트	Step Functions는 배열 조작, 데이터 인코딩 및 디코딩, 해시 계산, JSON 데이터 조작, 수학 함수 연산 및 고유 식별자 생성과 같은 데이터 처리 작업을 수행하기 위한 새로운 내장 함수 14개에 대한 지원을 추가합니다. 문서 전용 업데이트: 수행하는 데 도움이 되는 데이터 처리 작업 유형에 따라 기존 내장 함수와 새로 도입된 내장 함수 모두 다음 범주로 그룹화했습니다. • 배열의 내장 함수 • 데이터 인코딩 및 디코딩을 위한 내장 함수 • 해시 계산을 위한 내장 함수 • JSON 데이터 조작을 위한 내장 함수 • 수학 연산을 위한 내장 함수 • 문자열 작업을 위한 내장 연산 • 고유 식별자 생성을 위한 내장 함수 • 일반 연산을 위한 내장 함수 자세한 정보는 내장 함수을 참조하세요.	2022년 8월 31일
업데이트	Step Functions는 세 가지 AWS 서비스 (AWS Billing ConductorAmazon GameSparks, 및) 를 추가하여 AWS SDK 통합에 대한 지원을 확대했습니다. Amazon Pinpoint SMS and Voice V2 자세한 정보는 지원되는 AWS SDK 통합에 대한 변경 로그을 참조하세요.	2022년 7월 26일
문서 전용 업데이트	Step Functions에서 지원하는 AWS SDK 통합에 대한 모든 업데이트의 요약을 포함하는 새 항목이 추가되었습니다. 자세한 내용은 지원되는 AWS SDK 통합에 대한 변경 로그단원을 참조하세요.	2022년 7월 26일

변경 사항	설명	변경 날짜
문서 전용 업데이트	AWS Step Functions 개발자 안내서에는 이제 Express Workflows를 위해 특별히 생성되는 실행 지표에 대한 세부 정보가 포함되어 있습니다. 자세한 정보는 Express 워크플로의 실행 지표을 참조하세요.	2022년 6월 9일

변경 사항	설명	변경 날짜
업데이트	Step Functions 콘솔 개선 사항 이제 콘솔의 실행 세부 정보 페이지가 재설계되어 여러 개선 사항이 포함됩니다. • 실행 실패 원인을 한 눈에 파악할 수 있습니다. • 상태 시스템의 새로운 시각화 모드 2개(테이블 보기 및 이벤트 보기)입니다. 또한 이러한 보기에는 관심 있는 정보만 볼 수 있도록 필터를 적용하는 기능을 제공합니다. 또한 이벤트 타임스탬프를 기준으로 이벤트 보기 콘텐츠를 정렬할 수 있습니다. • 드롭다운 목록을 사용하는 그래프 보기 모드에서 또는 `Map` 상태에 대한 테이블 보기 모드의 트리 뷰에서 여러 `Map` 상태 반복으로 전환할 수 있습니다. • 전체 입력 및 출력 데이터 전송 경로, `Task` 또는 `Parallel` 상태에 대한 재시도를 포함한 워크플로의 각 상태에 대한 심층 정보를 볼 수 있습니다. • 기타 개선 사항으로 상태 시스템 실행 Amazon Resource Name을 복사하고 전체 상태 시스템 전환 수를 보고 실행 세부 정보를 JSON 형식으로 내보내는 옵션 등이 있습니다. 문서 전용 업데이트 실행 세부 정보 페이지에 표시되는 다양한 유형의 정보를 설명하는 새 주제가 추가되었습니다. 또한 이 정보를 검사하는 방법을 보여주는 자습서도 추가되었습니다. 자세한 내용은 다음을 참조하세요. • Step Functions 콘솔에서 실행 보기 및 디버깅 • 자습서: Step Functions 콘솔을 사용하여 상태 시스템 실행 검사	2022년 5월 9일

변경 사항	설명	변경 날짜
업데이트	이제 Step Functions에서 엔터티(서비스 또는 계정)가 다른 엔터티에 의해 작업을 강제 수행할 때 발생하는 혼동된 대리자 보안 문제를 방지하기 위한 해결책을 제공합니다. 자세한 내용은 다음을 참조하세요. • 교차 서비스 혼동된 대리자 문제 예방	2022년 5월 2일

업데이트	• Step Functions는 21개의 서비스를 더 AWS 추가하여 AWS SDK 통합에 대한 지원을 확대했습니다. 자세한 내용은 지원되는 AWS SDK 서비스 통합 섹션을 참조하세요.
	• 문서 전용 업데이트:
	  • Step Functions와의 AWS SDK 서비스 통합을 잘못 수행할 때 생성되는 예외에 존재하는 모든 예외 접두사 목록을 추가했습니다. 자세한 내용은 지원되는 AWS SDK 서비스 통합 섹션을 참조하세요.
	  • 지원되는 SDK 연동을 위해 지원되지 않는 모든 API 작업 목록을 추가했습니다. AWS 자세한 내용은 지원되는 서비스에 지원되지 않는 API 작업 섹션을 참조하세요.
	  • 현재 더 이상 사용되지 않는 지원되는 모든 AWS SDK 통합 목록을 추가했습니다. 자세한 내용은 더 이상 사용되지 않는 AWS SDK 서비스 통합 섹션을 참조하세요.
		2022년 4월 19일

새 기능	Step Functions Local은 이제 AWS SDK 통합 및 서비스 통합 모킹을 지원합니다. 자세한 내용은 다음을 참조하세요. • 모의 서비스 통합 사용	2022년 1월 28일

변경 사항	설명	변경 날짜
새 기능	AWS Step Functions 이제 를 사용하여 동기식 익스프레스 상태 머신을 백엔드 통합으로 사용하는 Amazon API Gateway REST API 생성을 지원합니다. AWS Cloud Development Kit (AWS CDK)자세한 내용은 다음을 참조하세요. • 를 사용하여 동기식 익스프레스 스테이트 머신을 사용하여 API Gateway REST API 만들기 AWS CDK	2021년 12월 10일
업데이트	Step Functions에 Step Functions 및 Amazon Athena의 업그레이드된 콘솔의 통합을 보여주는 새로운 샘플 프로젝트 3개가 추가되었습니다. 자세한 내용은 다음을 참조하세요. • 여러 쿼리 실행(Amazon Athena, Amazon SNS) • 대규모 데이터세트 쿼리 (아마존 아테나, 아마존 S3 AWS Glue, 아마존 SNS) • 데이터를 최신으로 유지 (아마존 아테나, 아마존 S3,) AWS Glue	2021년 11월 22일
새 기능	Step Functions에 동기 Express 워크플로를 위한 Amazon VPC 엔드포인트 지원이 추가되었습니다. 자세한 내용은 다음을 참조하세요. • Step Functions용 Amazon VPC 엔드포인트	2021년 11월 15일
업데이트	AWS Step Functions Step Functions AWS Batch 통합 사용 방법을 보여주는 세 개의 새 샘플 프로젝트가 추가되었습니다. 자세한 내용은 다음을 참조하세요. • AWS Batch 작업 팬 아웃 • AWS Batch 람다와 함께 • Step Functions 및 오류 처리 기능이 있는 AWS Batch 사용	2021년 10월 14일

변경 사항	설명	변경 날짜
새 기능	AWS Step Functions 200개가 넘는 서비스 모두에 API 작업을 사용할 수 있는 AWS SDK 통합이 추가되었습니다. AWS 자세한 내용은 다음을 참조하세요. • AWS SDK 서비스 통합 • AWS SDK 서비스 통합을 사용하여 Amazon S3 버킷 정보 수집	2021년 9월 30일
새 기능	AWS Step Functions 비주얼 워크플로우 디자이너인 워크플로 스튜디오를 추가했습니다. AWS Step Functions 자세한 내용은 다음을 참조하세요. • AWS Step Functions 워크플로 스튜디오 • AWS Step Functions 워크플로 스튜디오 사용 방법 알아보기	2021년 6월 17일
업데이트	AWS Step Functions CodeBuild 통합에 네 개의 새 API `StartBuildBatch` ,`StopBuildBatch` ,`DeleteBuildBatch` , `RetryBuildBatch` 를 추가했습니다. 자세한 내용은 다음을 참조하세요. • AWS CodeBuild Step 함수를 사용한 호출	2021년 6월 4일
새 기능	AWS Step Functions 이제 EventBridge Amazon과 통합됩니다. 자세한 내용은 다음을 참조하세요. • EventBridge Step 함수를 사용한 호출 • Step Functions 및 아마존용 IAM 정책 EventBridge에 대한 IAM 정책 • 에 맞춤 이벤트 보내기 EventBridge하는 방법을 보여주는 샘플 프로젝트	2021년 5월 14일

변경 사항	설명	변경 날짜
업데이트	AWS Step Functions Step Functions와 Amazon Redshift 데이터 API를 사용하여 ETL/ELT 워크플로를 실행하는 방법을 보여주는 새 샘플 프로젝트를 추가했습니다. 자세한 내용은 다음을 참조하세요. • Amazon Redshift(Lambda, Amazon Redshift Data API)를 사용하여 ETL/ELT 워크플로 실행	2021년 4월 16일
새 기능	AWS Step Functions 콘솔에 새로운 데이터 흐름 시뮬레이터가 있습니다. 자세한 내용은 다음을 참조하세요. • Step Functions 콘솔	2021년 4월 8일
새 기능	AWS Step Functions 이제 EKS의 Amazon EMR과 통합됩니다. 자세한 내용은 다음을 참조하세요. • EKS에서 Amazon EMR에 전화하십시오. AWS Step Functions	2021년 3월 29일
업데이트	상태 시스템 정의에 대한 YAML 지원이 AWS Toolkit for Visual Studio Code 및 AWS CloudFormation에 추가되었습니다. 자세한 내용은 다음을 참조하세요. • 정의 형식 지원 • AWS Toolkit for Visual Studio Code	2021년 3월 4일
새 기능	AWS Step Functions 이제 와 통합됩니다. AWS Glue DataBrew자세한 내용은 다음을 참조하세요. • Step Functions를 사용한 AWS Glue DataBrew 작업 관리 • 이게 AWS Glue DataBrew뭐야? DataBrew 개발자 가이드에서	2021년 1월 6일

변경 사항	설명	변경 날짜
새 기능	AWS Step Functions 이제 동기식 익스프레스 워크플로를 사용할 수 있어 마이크로서비스를 쉽게 오케스트레이션할 수 있습니다. 자세한 내용은 다음을 참조하세요. • 동기 및 비동기 Express 워크플로 • 동기 Express 워크플로 간접 호출하는 방법을 보여주는 샘플 프로젝트 • API StartSyncExecution설명서.	2020년 11월 24일
새 기능	AWS Step Functions 이제 Amazon API Gateway와 통합됩니다. 자세한 내용은 다음을 참조하세요. • Step Functions를 사용하여 API Gateway 직접 호출 • Step Functions 및 아마존 API 게이트웨이에 대한 IAM 정책에 대한 IAM 정책 • API Gateway 직접 호출하는 방법을 보여주는 샘플 프로젝트	2020년 11월 17일
새로운 특성	AWS Step Functions 이제 아마존 엘라스틱 쿠버네티스 서비스와 통합됩니다. 자세한 내용은 다음을 참조하세요. • Step Functions를 사용하여 Amazon EKS 호출 • Step Functions 및 아마존 EKS의 IAM 정책에 대한 IAM 정책 • Amazon EKS 클러스터 관리하는 방법을 보여주는 샘플 프로젝트	2020년 11월 16일
새 기능	AWS Step Functions 이제 Amazon Athena와 통합됩니다. 자세한 내용은 다음을 참조하세요. • Step Functions를 사용하여 Athena 직접 호출 • Step Functions 및 아마존 아테나에 대한 IAM 정책에 대한 IAM 정책 • Athena 쿼리 시작하는 방법을 보여주는 샘플 프로젝트	2020년 10월 22일

변경 사항	설명	변경 날짜
새 기능	AWS Step Functions 이제 end-to-end 워크플로 추적을 지원하여 상태 시스템 실행 전반에 대한 완전한 가시성을 제공하고 분산 애플리케이션을 더 쉽게 분석하고 디버깅할 수 있습니다. AWS X-Ray자세한 내용은 다음을 참조하세요. • AWS X-Ray 및 Step Functions • Step Functions 및 다음에 대한 IAM 정책 AWS X-Ray에 대한 IAM 정책 • AWS Step Functions API Reference • TracingConfiguration	2020년 9월 14일

변경 사항	설명	변경 날짜

업데이트	AWS Step Functions 이제 최대 256KB의 페이로드 크기를 UTF-8 인코딩 문자열로 지원합니다. 이를 통해 표준 및 Express 워크플로 모두에서 더 큰 페이로드를 처리할 수 있습니다.
	
	더 큰 페이로드를 사용하기 위해 기존 상태 시스템을 변경할 필요는 없습니다. 하지만 업데이트된 API를 사용하려면 Step Functions SDK 및 Local Runner를 최신 버전으로 업데이트해야 합니다. 자세한 내용은 다음을 참조하세요.
	
	• 할당량
	• the section called “대용량 페이로드를 전달하는 대신 Amazon S3 ARN 사용”
	• `States.DataLimitExceeded`
	• the section called “CloudWatch Logs 페이로드”
	• the section called “EventBridge 페이로드”
	• AWS Step Functions API Reference
	
	  • CloudWatchEventsExecutionDataDetails
	  • HistoryEventExecutionDataDetails
	  • GetExecutionHistory
	  • ActivityScheduledEventDetails
	  • ActivitySucceededEventDetails
	  • CloudWatchEventsExecutionDataDetails
	  • ExecutionSucceededEventDetails
	  • LambdaFunctionScheduledEventDetails
	  • ExecutionSucceededEventDetails
	  • StateEnteredEventDetails
	  • StateExitedEventDetails
	  • TaskSubmittedEventDetails
	  • TaskSucceededEventDetails
		2020년 9월 3일

변경 사항	설명	변경 날짜

업데이트	Amazon States Language가 다음과 같이 업데이트되었습니다.
	• 선택 규칙가 추가되었습니다.
	  • null 비교 연산자 IsNull. IsNull은 JSON null 값을 테스트하며 이전 상태의 출력 null인지 여부를 감지하는 데 사용될 수 있습니다.
	  • , IsBoolean, IsNumeric 및 연산자 4개가 새로 추가되었습니다. IsString IsTimestamp
	  • IsPresent 연산자를 사용하여 필드 존재 여부를 테스트합니다. IsPresent 는 존재하지 않는 키에 액세스하려고 할 때 States.Runtime 오류를 방지하는 데 사용될 수 있습니다.
	  • 와일드카드 패턴 매칭을 통해 와일드카드가 하나 이상 있는 패턴과 문자열을 비교할 수 있습니다.
	  • 지원되는 비교 연산자의 두 변수 비교
	• 이제 TimeoutSecondsPath 및 HeartbeatSecondsPath 필드를 사용하여 고정 값 대신 상태 입력에서 Task 상태의 제한 시간 및 하트비트 값을 동적으로 제공할 수 있습니다. 자세한 내용은 태스크 상태 상태를 참조하세요.
	• 새 ResultSelector 필드는 ResultPath 가 적용되기 전에 상태 결과를 조작할 수 있는 방법을 제공합니다. 이 ResultSelector 필드는 맵, Parallel 및 태스크 상태 상태의 선택적 필드입니다.
	• Task 상태 없이 기본 작업을 수행할 수 있도록 내장 함수가 추가되었습니다. Parameters 및 ResultSelector 필드 내에서 내장 함수를 사용할 수 있습니다.
		2020년 8월 13일

변경 사항	설명	변경 날짜
업데이트	AWS Step Functions 이제 Amazon SageMaker `CreateProcessingJob` API 호출을 지원합니다. 자세한 내용은 다음을 참조하세요. • Step SageMaker Functions를 사용한 관리 • `CreateProcessingJob` 을 보여주는 샘플 프로젝트인 데이터 전처리 및 기계 학습 모델 학습	2020년 8월 4일
새 기능	AWS Step Functions 이제 에서 AWS Serverless Application Model지원되므로 워크플로 오케스트레이션을 서버리스 애플리케이션에 더 쉽게 통합할 수 있습니다. 자세한 내용은 다음을 참조하세요. • AWS Step Functions 그리고 AWS SAM • AWS::Serverless::StateMachine • AWS SAM 정책 템플릿	2020년 5월 27일
새 기능	AWS Step Functions Step Functions 실행을 중첩하기 위한 새로운 동기 호출을 도입했습니다. 새 호출 `arn:aws:states:::states:startExecution.sync:2` 는 JSON 객체를 반환합니다. 원래 호출인 `arn:aws:states:::states:startExecution.sync` 은 계속 지원되며 JSON 이스케이프 문자열을 반환합니다. 자세한 내용은 다음을 참조하세요. • 통합 서비스로서의 AWS Step Functions 실행 관리	2020년 5월 19일
새 기능	AWS Step Functions 이제 와 통합됩니다. AWS CodeBuild 자세한 내용은 다음을 참조하세요. • 다른 서비스와 AWS Step Functions 함께 사용 • AWS CodeBuild Step 함수를 사용한 호출 • Step Functions를 위한 최적화된 통합	2020년 5월 5일

변경 사항	설명	변경 날짜
새 기능	이제 AWS Toolkit for Visual Studio Code에서 Step Functions가 지원되므로 코드 편집기에서 나가지 않고도 상태 시스템 기반 워크플로를 더 쉽게 만들고 시각화할 수 있습니다.	2020년 3월 31일
업데이트	이제 표준 워크플로를 위해 Amazon CloudWatch Logs 로깅을 구성할 수 있습니다. 자세한 내용은 다음을 참조하세요. • CloudWatch Logs를 사용하여 로깅	2020년 2월 25일
새 기능	AWS Step Functions 이제 퍼블릭 IP 주소 없이도 Amazon Virtual Private Cloud (VPC) 에서 직접 액세스할 수 있습니다. 자세한 내용은 다음을 참조하세요. • Step Functions용 Amazon VPC 엔드포인트	2019년 12월 23일

변경 사항	설명	변경 날짜

새 기능	Express 워크플로는 IoT 데이터 수집, 스트리밍 데이터 처리 및 변환, 모바일 애플리케이션 백엔드 등의 대용량 이벤트 처리에 워크로드에 적합한 새로운 워크플로 유형입니다.
	
	자세한 내용은 다음과 같은 신규 및 업데이트된 주제를 검토하십시오.
	
	• 표준 워크플로와 Express 워크플로 비교
	  • 실행 보장
	• 다른 서비스와 AWS Step Functions 함께 사용
	  • Step Functions를 위한 최적화된 통합
	• Amazon SQS에서 대용량 메시지 처리(Express 워크플로)
	• 선택적 체크포인트 예(Express 워크플로)
	• 할당량
	  • 할당량
	• CloudWatch Logs를 사용하여 로깅
	• AWS Step Functions API Reference
	  • CreateStateMachine
	  • UpdateStateMachine
	  • DescribeStateMachine
	  • DescribeStateMachineForExecution
	  • StopExecution
	  • DescribeExecution
	  • GetExecutionHistory
	  • ListExecutions
	  • ListStateMachines
	  • StartExecution
	  • CloudWatchLogsLogGroup
	  • LogDestination
	  • LoggingConfiguration
		2019년 12월 3일

변경 사항	설명	변경 날짜
새 기능	AWS Step Functions 이제 Amazon EMR과 통합됩니다. 자세한 내용은 다음을 참조하세요. • 다른 서비스와 AWS Step Functions 함께 사용 • Step Functions를 사용하여 Amazon EMR 직접 호출 • Step Functions를 위한 최적화된 통합	2019년 11월 19일

업데이트	AWS Step Functions Step AWS Functions 데이터 사이언스 SDK를 출시했습니다. 추가 정보는 다음을 참조하세요.
	• Github의 프로젝트
	• SDK 설명서
	• 다음 예제 노트북은 SageMaker콘솔 및 관련 GitHub 프로젝트에서 사용할 수 있습니다.
	  • `hello_world_workflow.ipynb`
	  • `machine_learning_workflow_abalone.ipynb`
	  • `training_pipeline_pytorch_mnist.ipynb`
		2019년 11월 7일

업데이트	Step Functions는 이제 Amazon을 위한 더 많은 API 작업을 지원하며 SageMaker, 기능을 시연하기 위한 두 개의 새로운 샘플 프로젝트를 포함합니다. 추가 정보는 다음을 참조하세요. • Step SageMaker Functions를 사용한 관리 • 다른 서비스와 AWS Step Functions 함께 사용 • 기계 학습 모델 훈련 • 기계 학습 모델 튜닝	2019년 10월 3일

변경 사항	설명	변경 날짜
새 기능	Step Functiosn에서 `StartExecution` 을 통합 서비스 API로 직접적으로 호출하여 새 워크플로 실행을 시작할 수 있습니다. 다음을 참조하세요. • 작업 상태에서 워크플로 실행 시작 • 통합 서비스로서의 AWS Step Functions 실행 관리 • 다른 서비스와 AWS Step Functions 함께 사용 • Step Functions 워크플로 실행을 시작하기 위한 IAM 정책	2019년 8월 12일
새 기능	Step Functions에는 작업 토큰을 통합 서비스에 전달하고 작업 토큰이 `SendTaskSuccess` 또는 `SendTaskFailure` 와 함께 반환될 때까지 실행을 일시 중지합니다. 다음을 참조하세요. • 서비스 통합 패턴 • 작업 토큰을 사용하여 콜백 대기 • 콜백 패턴 예제(Amazon SQS, Amazon SNS, Lamda) • Step Functions를 위한 최적화된 통합 • 예제 인간 승인 프로젝트 배포 • 서비스 통합 지표 이제 Step Functions가 상태 정의의 `"Parameters"` 필드에서 현재 실행에 대한 동적 정보에 직접 액세스하는 방법을 제공합니다. 다음을 참조하세요. • 컨텍스트 객체 • 파라미터로서 Pass 컨텍스트 객체 노드	2019년 5월 23일
새 기능	Step Functions는 실행 상태 변경에 대한 CloudWatch 이벤트를 지원합니다. 다음을 참조하십시오. • EventBridge Step Functions의 실행 상태 변경에 대한 (CloudWatch 이벤트) • 아마존 CloudWatch 이벤트 사용 설명서	2019년 5월 8일

변경 사항	설명	변경 날짜
새 기능	Step Functions에서 태그를 사용하여 IAM 권한을 지원합니다. 자세한 내용은 다음을 참조하세요. • Step Functions에서 태그 지정 • 태그 기반 정책	2019년 3월 5일
새 기능	이제 Step Functions Local을 사용할 수 있습니다. 테스트 및 개발을 위해 로컬 시스템에서 Step Functions를 실행할 수 있습니다. Step Functions Local을 Java 애플리케이션 또는 도커 이미지로 다운로드할 수 있습니다. 상태 시스템을 로컬로 테스트를 참조하세요.	2019년 2월 4일
새 기능	AWS Step Functions 이제 베이징 및 닝샤 지역에서 사용할 수 있습니다. 지원되는 리전를 참조하세요.	2018년 1월 15일
새 기능	Step Functions에서 비용 할당을 추적하는 데 도움이 되는 리소스 태그 지정을 지원합니다. Details(세부 정보) 페이지에서 또는 API 작업을 통해 상태 머신에 태그를 지정할 수 있습니다. Step Functions에서 태그 지정를 참조하세요.	2019년 1월 7일
새 기능	AWS Step Functions 이제 유럽 (파리) 및 남미 (상파울루) 지역에서 사용할 수 있습니다. 지원되는 리전를 참조하세요.	2018년 12월 13일
새 기능	AWS Step Functions 이제 유럽 (스톡홀름) 지역에서 사용할 수 있습니다. 지원되는 리전 목록은 지원되는 리전 단원을 참조하십시오.	2018년 12월 12일
새 기능	Step Functions는 이제 일부 AWS 서비스와 통합됩니다. 이제 파라미터를 직접 호출하고 Amazon States Language의 Task 상태에서 이러한 통합 서비스의 API로 전달할 수 있습니다. 자세한 내용은 다음을 참조하세요. • 다른 서비스와 AWS Step Functions 함께 사용 • 파라미터를 서비스 API에 전달 • Step Functions를 위한 최적화된 통합	2018년 11월 29일

변경 사항	설명	변경 날짜
업데이트	작업 상태에 대한 설명서에서 `TimeoutSeconds` 및 `HeartbeatSeconds` 의 설명 개선. 태스크 상태를 참조하세요.	2018년 10월 24일
업데이트	최대 실행 내역 크기 한도의 설명을 개선하고 관련 모범 사례 주제 링크를 제공했습니다. • 상태 시스템 실행과 관련된 할당량 • 내역 할당량 도달 방지	2018년 10월 17일
업데이트	AWS Step Functions 설명서에 새 자습서가 추가되었습니다. 을 참조하십시오Amazon S3 Events에 대한 응답으로 상태 시스템 실행 시작.	2018년 9월 25일
업데이트	한도 설명서의 단계 함수 콘솔에 표시된 항목 최대 실행을 제거했습니다. 할당량를 참조하세요.	2018년 9월 13일
업데이트	활동 작업을 위해 폴링할 때의 지연 시간 개선에 대한 모범 사례 주제를 AWS Step Functions 문서에 추가했습니다. 활동 작업을 폴링할 때 지연 시간 방지를 참조하세요.	2018년 8월 30일
업데이트	활동 및 활동 종사자에 대한 AWS Step Functions 주제를 개선했습니다. 활동를 참조하세요.	2018년 8월 29일
업데이트	CloudTrail 통합에 관한 AWS Step Functions 주제를 개선했습니다. 를 사용하여 API 호출 녹음하기 AWS CloudTrail를 참조하세요.	2018년 8월 7일
업데이트	AWS CloudFormation 튜토리얼에 JSON 예제를 추가했습니다. AWS CloudFormation를 사용하여 Step Functions용 Lambda 상태 시스템 만들기를 참조하세요.	2018년 6월 23일
업데이트	Lambda 서비스 오류 처리에 대한 새 주제가 추가되었습니다. Lambda 서비스 예외 처리를 참조하세요.	2018년 6월 20일

변경 사항	설명	변경 날짜
새 기능	AWS Step Functions 이제 아시아 태평양 (뭄바이) 지역에서 사용할 수 있습니다. 지원되는 리전 목록은 지원되는 리전 단원을 참조하십시오.	2018년 6월 28일
새 기능	AWS Step Functions 이제 AWS GovCloud (미국 서부) 지역에서 사용할 수 있습니다. 지원되는 리전 목록은 지원되는 리전 단원을 참조하십시오. AWS GovCloud (미국 서부) 지역에서 Step Functions를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오 AWS GovCloud (US).	2018년 6월 28일
업데이트	`Parallel` 상태의 오류 문제 해결에 대한 설명서 개선 오류 처리를 참조하세요.	2018년 6월 20일
업데이트	Step Functions에서 입력 및 출력을 처리하는 방법에 대한 문서를 개선했습니다. `InputPath` , `ResultPath` 및 `OutputPath` 를 사용하는 방법에 대해 알아보아 자체 워크플로우, 상태 및 작업을 통해 JSON 흐름을 관리합니다. 다음을 참조하세요. • Step Functions에서 입력 및 출력 처리 • ResultPath	2018년 6월 7일
업데이트	병렬 상태의 코드 예제 개선 Parallel를 참조하세요.	2018년 6월 4일
새 기능	이제 에서 CloudWatch API 및 서비스 지표를 모니터링할 수 있습니다. Step Functions를 사용하여 모니터링하기 CloudWatch를 참조하세요.	2018년 5월 25일

변경 사항	설명	변경 날짜
업데이트	`StartExecution` , `StopExecution` 및 `StateTransition` 조절 한도가 다음 리전에서 향상되었습니다. • 미국 동부(버지니아 북부) • US West (Oregon) • 유럽(아일랜드) 자세한 내용은 할당량단원을 참조하세요.	2018년 5월 16일
새 기능	AWS Step Functions 이제 미국 서부 (캘리포니아 북부) 및 아시아 태평양 (서울) 지역에서 사용할 수 있습니다. 지원되는 리전 목록은 지원되는 리전 단원을 참조하십시오.	2018년 5월 5일
업데이트	인터페이스 변경 사항과 일치하도록 절차와 이미지를 업데이트했습니다.	2018년 25월 4일
업데이트	새로운 실행을 시작하여 작업을 계속하는 방법을 보여주는 새로운 자습서를 추가했습니다. 장기 실행 워크플로 실행을 새 실행으로 계속하기를 참조하세요. 이 자습서에서는 일부 서비스 제한을 방지할 수 있는 설계 패턴에 대해 설명합니다. 내역 할당량 도달 방지를 참조하세요.	2018년 4월 19일
업데이트	상태 머신에 대한 개념 정보를 추가함으로써 상태 설명서 소개를 개선했습니다. 상태를 참조하세요.	2018년 3월 9일
업데이트	HTML, PDF 및 Kindle 외에도 AWS Step Functions 개발자 안내서는 에서 사용할 수 있습니다. GitHub 피드백을 남기려면 오른쪽 상단의 GitHub 아이콘을 선택하세요. 	2018년 3월 2일

변경 사항	설명	변경 날짜
업데이트	Step Functions 관련 다른 리소스를 설명하는 주제가 추가되었습니다. 관련 정보를 참조하세요.	2018년 2월 20일

새 기능	• 새 상태 시스템을 만들 때 AWS Step Functions 에서 Lambda 함수에 액세스를 허용하는 IAM 역할을 만든다는 사실을 알아야 합니다.
	• 상태 머신 생성 워크플로우의 사소한 변경 사항을 반영하고자 다음 자습서를 업데이트하였습니다.
	  • Lambda를 사용하는 Step Functions 상태 시스템 만들기
	  • Step Functions를 사용하여 Activity 상태 시스템 만들기
	  • Step Functions 상태 시스템을 사용하여 오류 조건 처리
	  • Lambda를 사용하여 루프를 반복하세요
		2018년 2월 19일

업데이트	Ruby로 작성된 예제 활동을 설명하는 주제를 추가했습니다. 이 구현은 직접 Ruby 활동 작업자를 생성하는 데 사용할 수도 있고 다른 언어로 활동 작업자로 활동 작업자를 생성하기 위한 설계 패턴으로 사용할 수도 있습니다. Ruby 활동 작업자 예제를 참조하세요.	2018년 2월 6일
업데이트	계산 반복에 Lambda 함수를 사용하는 설계 패턴을 설명하는 새로운 자습서가 추가되었습니다. Lambda를 사용하는 Step Functions 상태 시스템 만들기를 참조하세요.	2018년 1월 31일
업데이트	`DescribeStateMachineForExecution` 및 `UpdateStateMachine` API가 포함되도록 IAM 권한에 대한 내용이 업데이트되었습니다. 관리자가 아닌 사용자에 대한 세부 IAM 권한 만들기를 참조하세요.	2018년 1월 26일

변경 사항	설명	변경 날짜
업데이트	새로 사용 가능한 리전(캐나다(중부), 아시아 태평양(싱가포르))이 추가되었습니다. 지원되는 리전를 참조하세요.	2018년 1월 25일
업데이트	IAM에서 Step Functions를 역할로 선택하도록 허용함을 반영하고자 자습서와 절차를 업데이트했습니다.	2018년 1월 24일
업데이트	상태 사이에 대용량 페이로드를 전달하지 않음을 권장하는 새 모범 사례 주제를 추가했습니다. 대용량 페이로드를 전달하는 대신 Amazon S3 ARN 사용를 참조하세요.	2018년 1월 23일
업데이트	상태 머신을 생성하기 위한 업데이트된 인터페이스와 일치하도록 절차를 수정했습니다. • Lambda를 사용하는 Step Functions 상태 시스템 만들기 • Step Functions를 사용하여 Activity 상태 시스템 만들기 • Step Functions 상태 시스템을 사용하여 오류 조건 처리	2018년 1월 17일
새 기능	샘플 프로젝트를 사용하여 상태 머신 및 관련된 모든 AWS 리소스를 프로비저닝할 수 있습니다. Step Functions를 위한 샘플 프로젝트를 참조하세요. 사용 가능한 샘플 프로젝트는 다음과 같습니다. • 작업 상태 설문 조사 (Lambda AWS Batch,) • 태스크 타이머(Lambda, Amazon SNS) Note 이러한 샘플 프로젝트와 관련 설명서는 동일한 기능을 구현을 설명한 자습서를 대체합니다.	2018년 1월 11일

변경 사항	설명	변경 날짜
업데이트	실행 멈춤 방지에 대한 내용을 포함하는 모범 사례 단원을 추가했습니다. Step Functions 모범 사례를 참조하세요.	2018년 1월 5일
업데이트	재시도가 요금에 어떻게 영향을 미칠 수 있는지에 대한 내용이 추가되었습니다. **Note** 재시도는 상태 변환으로 취급됩니다. 상태 전환이 결제에 미치는 영향은 Step Functions 요금을 참조하세요.	2017년 12월 8일
업데이트	리소스 이름과 관련된 추가 정보: **Note** Step Functions를 사용하면 상태 머신, 실행 및 활동의 이름과 ASCII가 아닌 문자가 포함된 레이블을 만들 수 있습니다. 이러한 비 ASCII 이름은 Amazon에서 사용할 수 없습니다. CloudWatch CloudWatch 지표를 추적할 수 있도록 하려면 ASCII 문자만 사용하는 이름을 선택하십시오.	2017년 12월 6일
업데이트	보안 개요 정보가 개선되고 세부 IAM 권한에 대한 주제가 추가되었습니다. 보안: AWS Step Functions 및 관리자가 아닌 사용자에 대한 세부 IAM 권한 만들기 단원을 참조하세요.	2017년 11월 27일
새 기능	기존의 상태 머신을 업데이트할 수 있습니다. 상태 시스템 업데이트를 참조하세요.	2017년 11월 15일

변경 사항	설명	변경 날짜
업데이트	`Lambda.Unknown` 오류를 식별하는 내용을 추가했고, 다음 단원에서 Lambda 설명서 링크를 추가했습니다. • 오류 이름 • 3단계: Catch 필드를 사용하여 상태 시스템 만들기 **Note** Lambda에서 처리되지 않은 오류는 오류 출력에서 `Lambda.Unknown` 으로 보고됩니다. 여기에는 out-of-memory 오류와 함수 타임아웃이 포함됩니다. `Lambda.Unknown` , `States.ALL` 또는 `States.TaskFailed` 를 일치시켜 이러한 오류를 처리할 수 있습니다. Lambda에서 최대 간접 호출 수에 도달하면 오류는 `Lambda.TooManyRequestsException` 입니다. Lambda 함수 오류에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 오류 처리 및 자동 재시도를 참조하세요.	2017년 10월 17일
업데이트	IAM 지침이 명확하게 수정됐으며 모든 자습서의 스크린샷이 업데이트되었습니다.	2017년 10월 11일

변경 사항	설명	변경 날짜

업데이트	• Step Functions 콘솔 변경 사항이 반영되도록 상태 시스템 실행 결과에 대한 스크린샷이 새롭게 추가되었습니다. Lambda 콘솔의 변경 사항이 반영되도록 다음 자습서의 Lambda 지침이 다시 작성되었습니다.
	  • Lambda를 사용하는 Step Functions 상태 시스템 만들기
	  • 작업 상태 poller 생성
	  • 작업 타이머 생성
	  • Step Functions 상태 시스템을 사용하여 오류 조건 처리
	• 상태 머신 생성에 대한 다음 단원의 정보가 수정되고 명확해졌습니다.
	  • Step Functions를 사용하여 Activity 상태 시스템 만들기
		2017년 10월 6일

업데이트	IAM 콘솔의 변경 사항이 반영되도록 다음 섹션의 IAM 지침이 다시 작성되었습니다. • 상태 시스템을 위한 IAM 역할 만들기 • Lambda를 사용하는 Step Functions 상태 시스템 만들기 • 작업 상태 poller 생성 • 작업 타이머 생성 • Step Functions 상태 시스템을 사용하여 오류 조건 처리 • API Gateway를 사용하여 Step Functions API 만들기	2017년 10월 5일
업데이트	상태 머신 데이터 섹션을 다시 작성했습니다.	2017년 9월 28일
새 기능	Step Functions를 사용할 수 있는 모든 리전에서 API 작업 제한과 관련된 제한이 증가합니다.	2017년 9월 18일
업데이트	• 모든 자습서에서 새로운 실행을 시작하는 방법에 대한 정보가 수정되고 명확해졌습니다. • 계정과 관련된 할당량 섹션의 정보가 수정되고 명확해졌습니다.	2017년 9월 14일

변경 사항	설명	변경 날짜
업데이트	Lambda 콘솔의 변경 사항이 반영되도록 다음 자습서가 다시 작성되었습니다. • Lambda를 사용하는 Step Functions 상태 시스템 만들기 • Step Functions 상태 시스템을 사용하여 오류 조건 처리 • 작업 상태 poller 생성	2017년 8월 28일
새 기능	유럽(런던)에서 Step Functions를 사용할 수 있습니다.	2017년 8월 23일
새 기능	상태 머신의 이 시각적 워크플로우를 통해 그래프를 확대하고 축소하며 중앙에 조정할 수 있습니다.	2017년 8월 21일
새 기능	**Important** 실행에서 다른 실행의 이름을 90일 동안 사용할 수 없습니다. 같은 이름으로 여러 번 `StartExecution` 호출하면 새 실행이 실행되지 않습니다. 자세한 내용은 AWS Step Functions API 참조에서 `StartExecution` API 작업의 `name` 파라미터를 참조하세요.	2017년 8월 18일
업데이트	상태 머신 ARN을 API Gateway를 사용하여 Step Functions API 만들기 자습서에 전달하는 다른 방법에 대한 정보를 추가했습니다.	2017년 8월 17일
업데이트	작업 상태 폴러 생성 자습서를 추가했습니다.	2017년 8월 10일

변경 사항	설명	변경 날짜
새 기능	• Step Functions는 `ExecutionThrottled` CloudWatch 메트릭을 내보냅니다. 자세한 정보는 Step Functions를 사용하여 모니터링하기 CloudWatch을 참조하세요. • 상태 제한과 관련된 할당량 단원을 추가했습니다.	2017년 8월 3일
업데이트	1단계: API Gateway에 대한 IAM 역할 만들기 섹션의 지침을 업데이트했습니다.	2017년 18월 7일
업데이트	Choice 섹션의 정보가 수정되고 명확해졌습니다.	2017년 6월 23일
업데이트	다른 AWS 계정의 리소스 사용에 대한 정보를 다음 자습서에 추가했습니다. • Lambda를 사용하는 Step Functions 상태 시스템 만들기 • AWS CloudFormation를 사용하여 Step Functions용 Lambda 상태 시스템 만들기 • Step Functions를 사용하여 Activity 상태 시스템 만들기 • Step Functions 상태 시스템을 사용하여 오류 조건 처리	2017년 6월 22일
업데이트	다음 섹션의 정보가 수정되고 명확해졌습니다. • Step Functions 상태 시스템을 사용하여 오류 조건 처리 • 상태 • Step Functions에서 오류 처리	2017년 6월 21일
업데이트	Step Functions 콘솔의 변경 사항과 일치하도록 모든 자습서가 다시 작성되었습니다.	2017년 6월 12일
새 기능	아시아 태평양(시드니)에서 Step Functions를 사용할 수 있습니다.	2017년 6월 8일
업데이트	Amazon States Language 단원이 재구성되었습니다.	2017년 6월 7일

변경 사항	설명	변경 날짜
업데이트	Step Functions를 사용하여 Activity 상태 시스템 만들기 섹션의 정보가 수정되고 명확해졌습니다.	2017년 6월 6일
업데이트	Retry 및 Catch를 사용하는 상태 시스템 예제 단원의 코드 예제가 수정되었습니다.	2017년 6월 5일
업데이트	AWS 문서 표준을 사용하여 이 안내서를 재구성했습니다.	2017년 5월 31일
업데이트	Parallel 섹션의 정보가 수정되고 명확해졌습니다.	2017년 5월 25일
업데이트	Step Functions에서 입력 및 출력 처리 섹션으로 경로 및 필터 섹션을 병합했습니다.	2017년 5월 24일
업데이트	Step Functions를 사용하여 모니터링하기 CloudWatch 섹션의 정보가 수정되고 명확해졌습니다.	2017년 5월 15일
업데이트	Step Functions를 사용하여 Activity 상태 시스템 만들기 자습서의 `GreeterActivities.java` 작업자 코드를 업데이트했습니다.	2017년 5월 9일
업데이트	무엇입니까 AWS Step Functions? 섹션에 소개 동영상을 추가했습니다.	2017년 4월 19일
업데이트	다음 자습서의 정보가 수정되고 명확해졌습니다. • Lambda를 사용하는 Step Functions 상태 시스템 만들기 • Step Functions를 사용하여 Activity 상태 시스템 만들기 • Step Functions 상태 시스템을 사용하여 오류 조건 처리	2017년 4월 19일
업데이트	Lambda를 사용하는 Step Functions 상태 시스템 만들기 및 Step Functions 상태 시스템을 사용하여 오류 조건 처리 자습서에 Lambda 템플릿에 대한 정보가 추가되었습니다.	2017년 4월 6일

변경 사항	설명	변경 날짜
업데이트	"최대 입력 또는 결과 데이터 크기" 제한을 "작업, 상태 또는 실행에 대한 입력 또는 결과 데이터 크기 최대값"(32,768자)으로 변경했습니다. 자세한 정보는 작업 실행과 관련된 할당량을 참조하세요.	2017년 3월 31일
새 기능	• Step Functions는 Step Functions를 Amazon CloudWatch Events 타겟으로 설정하여 상태 머신을 실행할 수 있도록 지원합니다.	2017년 3월 21일
새 기능	• Step Functions에서 기본 오류 처리 방법으로 Lambda 함수 오류 처리를 허용합니다. • Step Functions 상태 시스템을 사용하여 오류 조건 처리 자습서 및 Step Functions에서 오류 처리 섹션을 업데이트했습니다.	2017년 3월 16일
새 기능	유럽(프랑크푸르트)에서 Step Functions를 사용할 수 있습니다.	2017년 3월 7일
업데이트	목차를 재구성하고 다음 자습서를 업데이트했습니다. • Lambda를 사용하는 Step Functions 상태 시스템 만들기 • Step Functions를 사용하여 Activity 상태 시스템 만들기 • Step Functions 상태 시스템을 사용하여 오류 조건 처리	2017년 23월 2일
새 기능	• Step Functions 콘솔의 상태 시스템 페이지에 새로 복사 및 삭제 버튼이 있습니다. • 콘솔 변경 사항에 맞게 스크린샷이 업데이트되었습니다.	2017년 23월 2일
새 기능	• Step Functions에서 API Gateway를 사용하여 API를 만들 수 있습니다. • API Gateway를 사용하여 Step Functions API 만들기 자습서가 추가되었습니다.	2017년 2월 14일

변경 사항	설명	변경 날짜
새 기능	• Step Functions는 와의 통합을 지원합니다 AWS CloudFormation. • AWS CloudFormation를 사용하여 Step Functions용 Lambda 상태 시스템 만들기 자습서가 추가되었습니다.	2017년 2월 10일
업데이트	`ResultPath` 및 `OutputPath` 필드의 현재 동작을 `Parallel` 상태와 비교하여 명확히 설명했습니다.	2017년 2월 6일
업데이트	• 자습서에 상태 머신 이름 지정 제한 사항을 명확히 설명했습니다. • 몇 가지 코드 예제를 수정했습니다.	2017년 1월 5일
업데이트	최신 프로그래밍 모델을 사용하도록 Lambda 함수 예제를 업데이트했습니다.	2016년 9월 12일
최초 릴리스	의 초기 릴리스 AWS Step Functions.	2016년 1월 12일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.