



PostgreSQL 쿼리 성능 최적화

# AWS 규범적 지침



# AWS 규범적 지침: PostgreSQL 쿼리 성능 최적화

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

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

# Table of Contents

소개 .....	1
쿼리 성능 튜닝의 사용 사례 .....	1
플랜 설명 .....	2
EXPLAIN 명령문 .....	2
설명 분석 사용 .....	2
EXPLAIN 쿼리 계획을 읽는 방법 .....	2
.....	4
콜레이션 .....	13
데이터 유형 불일치 .....	16
선택트에서의 함수 호출 .....	18
현재 또는 존재 .....	19
서브쿼리 또는 CTE .....	22
FAQ .....	26
EXPLAIN이란 무엇입니까? .....	26
EXPLAIN ANALYZE는 무엇입니까? .....	26
PostgreSQL에서 데이터 정렬이란 무엇입니까? .....	27
CTE란 무엇입니까? .....	27
PostgreSQL의 함수 카테고리는 무엇입니까? .....	27
.....	29
기여자 .....	30
사용 설명서 기록 .....	31
용어집 .....	32
# .....	32
A .....	33
B .....	35
C .....	37
D .....	40
E .....	44
F .....	46
G .....	47
H .....	48
정보 .....	50
L .....	52
M .....	53

---

O .....	57
P .....	59
Q .....	62
R .....	62
S .....	65
T .....	68
U .....	70
V .....	70
W .....	71
Z .....	72
.....	lxxiii

# PostgreSQL 쿼리 성능 최적화

Amazon Web Services ([기고자](#))

2024년 4월 ([문서](#) 기록)

PostgreSQL은 강력하고 유연하며 신뢰할 수 있는 오픈 소스 객체 관계형 데이터베이스 시스템입니다. PostgreSQL 쿼리의 성능을 최적화하는 방법은 여러 가지가 있습니다. 쿼리를 최적화하는 프로세스는 사용 사례에 따라 달라집니다. 현재 쿼리 계획을 알면 문제를 식별 및 이해하고 필요한 사항을 변경하는 데 도움이 될 수 있습니다. 때로는 데이터베이스 통계를 최신 상태로 유지하기 위해 테이블을 분석해야 할 수도 있습니다. PostgreSQL 최적화 프로그램은 이러한 통계를 사용하여 쿼리를 더 빠르게 실행합니다. 이 가이드에서는 PostgreSQL 쿼리의 성능을 개선하기 위한 모범 사례를 중점적으로 다룹니다.

이 가이드에서는 PostgreSQL용 Amazon RDS (Amazon RDS) 또는 Amazon Aurora PostgreSQL과 호환되는 기존 데이터베이스 인스턴스가 있다고 가정합니다.

## 쿼리 성능 튜닝의 사용 사례

이 가이드에서는 설명과 예제와 함께 다섯 가지 사용 사례를 다룹니다.

- 콜레이션
- 데이터 유형 불일치
- 명령문의 함수 호출 SELECT
- IN 또는 EXISTS
- 하위 쿼리 또는 일반 테이블 표현식 (CTE)

각 사용 사례는 초기 실행 계획의 세부 정보, 계획을 분석하여 문제를 식별하는 방법, 해결 방법을 제공합니다. 이러한 사용 사례를 구현하면 일반적으로 쿼리 응답 시간이 빨라지고 서버 부하가 줄어들며 시스템 효율성이 전반적으로 향상됩니다. 이러한 개선을 통해 사용자 경험이 향상되고 시스템 안정성이 향상될 수 있습니다.

# EXPLAIN 쿼리 플랜

PostgreSQL은 쿼리 실행 방법에 대한 세부 정보가 포함된 쿼리 계획을 반환하기 위한 EXPLAIN ANALYZE 및 옵션을 제공합니다EXPLAIN.

## EXPLAIN 명령문

이 EXPLAIN 명령문은 PostgreSQL 플래너가 지정된 명령문에 대해 생성한 쿼리 계획을 반환합니다. 쿼리 계획에는 다음이 표시됩니다.

- 명령문과 관련된 테이블을 스캔하는 방법 (예: 인덱스 스캔 또는 순차 스캔)
- 여러 테이블을 조인하는 방법 (예: 해시 조인, 병합 조인 또는 중첩 루프 조인)

쿼리 성능을 개선하려면 계획을 이해하는 것이 중요합니다. 계획을 이해한 후에는 쿼리에 시간이 너무 오래 걸리는 부분에 집중하고 시간을 줄이기 위한 조치를 취할 수 있습니다.

## 설명 분석 사용

EXPLAIN PostgreSQL에서는 주어진 명령문에 대한 계획만 생성합니다. ANALYZE 키워드를 추가하면 계획이 EXPLAIN 반환되고 쿼리가 실행되며 각 단계의 실제 런타임과 행 수가 표시됩니다. 이는 쿼리 성능을 분석하는 데 필수적입니다.

### Important

사용할 때는 EXPLAIN ANALYZE INSERTUPDATE, 및 에 주의하십시오DELETE.

## EXPLAIN 쿼리 계획을 읽는 방법

PostgreSQL 쿼리 플랜은 여러 노드로 구성된 트리 구조입니다. EXPLAIN 쿼리 계획은 데이터베이스 엔진이 쿼리를 실행하는 데 사용하는 단계를 보여줍니다. 쿼리 계획은 다음 정보를 제공합니다.

- 수행된 작업 유형 (예: 순차 스캔, 인덱스 스캔 또는 중첩 루프 조인)
- 수행 중인 작업을 설명하는 레이블 (예: Seq Scan Index ScanNested Loop, 또는).
- 쿼리로 처리 중인 테이블 또는 인덱스의 이름.

- 임의의 계산 단위의 예상 비용 및 처리된 행 수에 대한 정보가 포함된 비용 및 행 열
- 작업에 적용된 모든 필터의 필터 조건 (예: 조건). where
- 단계를 시각적으로 표현한 것으로, 각 작업은 작업을 연결하는 노드와 화살표로 표시됩니다. 작업 순서는 왼쪽에서 오른쪽으로 표시되며 이전 작업은 이후 작업으로 이어집니다.

다음 스크린샷은 순차 스캔의 쿼리 계획을 보여줍니다.

EXPLAIN ANALYZE SELECT * FROM mig.test_ts	
1	Seq Scan on test_ts (cost=0.00..32.60 rows=2260 width=8) (actual time=0.120..0.121 rows=1 loops=1)
2	Planning Time: 0.029 ms
3	Execution Time: 0.132 ms

예상 비용은 PostgreSQL이 쿼리에서 결과를 반환하는 데 32.60단위의 계산이 필요할 것으로 예상한다는 (cost=0.00..32.60 rows=2260 width=8) 의미입니다.

0.00값은 이 노드가 작동을 시작할 수 있는 비용 (이 경우 쿼리 시작 시간)입니다. rows값은 순차 스캔이 반환할 예상 행 수입니다. width값은 반환된 행의 예상 크기 (바이트)입니다.

예제가 EXPLAIN ANALYZE 옵션과 함께 표시되기 때문에 쿼리가 실행되었고 타이밍 정보가 캡처되었습니다. 결과는 다음을 (actual time=0.120..0.121 rows=1 loops=1) 의미합니다.

- 순차 스캔을 한 번 실행했습니다 (loops값).
- 스캔 결과 한 행이 반환되었습니다.
- 실제 시간은 0.12밀리초였습니다.

## 쿼리 튜닝 사용 사례

이 가이드에서는 쿼리 성능 튜닝을 위한 다음과 같은 사용 사례를 다룹니다.

- 콜레이션
- 데이터 유형 불일치
- 명령문의 함수 호출 SELECT
- IN 또는 EXISTS
- 하위 쿼리 또는 일반 테이블 표현식 (CTE)

이러한 쿼리 성능 사용 사례에 대한 성능 튜닝을 테스트하려면 기존 데이터베이스와 이 가이드에서 제공하는 예제 데이터를 사용하십시오. 이 예제에서는 가상의 XX 항공사의 데이터를 사용합니다. 예제 데이터를 준비하려면 다음 예제 코드를 실행하십시오.

```
--Creating required tables along with data.

--Creating user and schema
create user perf_user;
create schema perf_user AUTHORIZATION perf_user;
set search_path to perf_user;

--Table1:

CREATE TABLE IF NOT EXISTS perf_user.rnr_expiry_date
(
    airline_iata_code character(2) COLLATE pg_catalog."default",
    pnr_number character varying(15) COLLATE pg_catalog."default" NOT NULL,
    calculated_pnr_expiry_date timestamp(0) without time zone,
    row_num bigint,
    arc_expiry_date timestamp(0) without time zone,
    status character varying(10) COLLATE pg_catalog."default"
);

insert into perf_user.rnr_expiry_date
select 'XX' , upper(substring(concat(md5(random()::text), md5(random()::text)), 0,
7)), '2023-01-01 00:00:00', generate_series(1,100000), '2023-02-02 00:00:00' ,null;
```

```

CREATE INDEX rnr_expiry_date_idx1 ON perf_user.rnr_expiry_date (row_num ASC NULLS
  LAST);

CREATE INDEX rnr_expiry_date_idx2  ON perf_user.rnr_expiry_date (airline_iata_code
  COLLATE pg_catalog."default" ASC NULLS LAST, pnr_number COLLATE pg_catalog."default"
  ASC NULLS LAST);

CREATE INDEX rnr_expiry_date_idx3  ON perf_user.rnr_expiry_date (pnr_number ASC NULLS
  LAST);

vacuum analyze perf_user.rnr_expiry_date;

-----
--Table2:

CREATE TABLE IF NOT EXISTS perf_user.rnr_segment_pax
(
  airline_iata_code character varying(6) COLLATE pg_catalog."default" NOT NULL,
  pnr_number character varying(15) COLLATE pg_catalog."default" NOT NULL,
  segment_pax_id numeric(25,0) NOT NULL,
  oandd_id numeric(25,0) NOT NULL,
  segment_id numeric(25,0) NOT NULL,
  cabin_class character varying(15) COLLATE pg_catalog."default",
  pax_id numeric(25,0) NOT NULL,
  ticket_number character varying(25) COLLATE pg_catalog."default",
  ticket_type character varying(10) COLLATE pg_catalog."default",
  archive_status smallint NOT NULL DEFAULT (0)::smallint,
  certificate_number character varying(100) COLLATE pg_catalog."default",
  loyalty_number character varying(25) COLLATE pg_catalog."default",
  arc_expiry_date timestamp(0) without time zone,
  CONSTRAINT rnr_segment_pax_pk PRIMARY KEY (airline_iata_code, pnr_number,
segment_id, pax_id),
  CONSTRAINT rnr_segment_pax_ck1 CHECK (ticket_type::text = ANY (ARRAY['E'::character
varying::text, 'A'::character varying::text, 'C'::character varying::text,
'M'::character varying::text, 'I'::character varying::text]))
);

insert into perf_user.rnr_segment_pax (airline_iata_code, pnr_number, segment_pax_id,
  oandd_id, segment_id, pax_id, ticket_type, arc_expiry_date )
select 'XX',upper(substring(concat(md5(random()::text), md5(random()::text)), 0, 7)),
generate_series(1,10000000),generate_series(1,10000000),
generate_series(1,10000000),generate_series(1,10000000),'A','2023-01-01 00:00:00';

```

```
insert into perf_user.rnr_segment_pax (airline_iata_code, pnr_number, segment_pax_id,
  oandd_id, segment_id, pax_id, ticket_type, arc_expiry_date )
select 'XX',upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(10000001,20000000),generate_series(10000001,20000000),
generate_series(10000001,20000000),generate_series(10000001,20000000),'I','2023-01-01
00:00:00';

insert into perf_user.rnr_segment_pax (airline_iata_code, pnr_number, segment_pax_id,
  oandd_id, segment_id, pax_id, ticket_type, arc_expiry_date)
select 'XX',upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(20000001,30000000),generate_series(20000001,30000000),
generate_series(20000001,30000000),generate_series(20000001,30000000),'E','2023-01-01
00:00:00';

insert into perf_user.rnr_segment_pax (airline_iata_code, pnr_number, segment_pax_id,
  oandd_id, segment_id, pax_id, ticket_type, arc_expiry_date)
select 'XX',upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(30000001,40000000),generate_series(30000001,40000000),
generate_series(30000001,40000000),generate_series(30000001,40000000),'M','2023-01-01
00:00:00';

CREATE INDEX rnr_segment_pax_idx1
  ON perf_user.rnr_segment_pax USING btree
  (loyalty_number COLLATE pg_catalog."default" ASC NULLS LAST, airline_iata_code
  COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS rnr_segment_pax_pn_idx1
  ON perf_user.rnr_segment_pax USING btree
  (pnr_number COLLATE pg_catalog."default" ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS rnr_segment_pax_seq_idx1
  ON perf_user.rnr_segment_pax USING btree
  (segment_id ASC NULLS LAST);

vacuum analyze perf_user.rnr_segment_pax;

-----

--Table3:

CREATE TABLE IF NOT EXISTS perf_user.rnr_segment
(
  airline_iata_code character varying(6) COLLATE pg_catalog."default" NOT NULL,
```

```
    pnr_number character varying(15) COLLATE pg_catalog."C" NOT NULL,
    segment_id numeric(25,0) NOT NULL,
    oandd_id numeric(25,0),
    price_id numeric(25,0),
    flight_carrier character varying(6) COLLATE pg_catalog."default" ,
    flight_number integer ,
    flight_suffix character varying(1) COLLATE pg_catalog."default" ,
    flight_date_ltc timestamp(0) without time zone ,
    airline_company_code character varying(6) COLLATE pg_catalog."default",
    bd_airport_code character varying(5) COLLATE pg_catalog."default" ,
    off_airport_code character varying(5) COLLATE pg_catalog."default" ,
    segment_status character varying(50) COLLATE pg_catalog."default" ,
    flight_status character varying(30) COLLATE pg_catalog."default",
    flight_type character varying(15) COLLATE pg_catalog."default",
    cabin_class character varying(15) COLLATE pg_catalog."default",
    arc_expiry_date timestamp(0) without time zone,
    oandd_dep_date_ltc timestamp(0) without time zone,
    added_time timestamp(6) without time zone,
    dep_date_ltc timestamp(0) without time zone ,
    arr_date_utc timestamp(0) without time zone,
    dep_date_utc timestamp(0) without time zone,
    origin character varying(5) COLLATE pg_catalog."default",
    destination character varying(5) COLLATE pg_catalog."default",
    CONSTRAINT rnr_segment_pk PRIMARY KEY (pnr_number, segment_id, airline_iata_code)
);
```

```
insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER,FLIGHT_NUMBER,FLIGHT_SUFFIX,FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(1,1000000),'XX',110,'*', '2023-01-01 00:00:00';
```

```
insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(10000001,20000000),'XX',120,'*', '2023-01-01 00:00:00';
```

```
insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER,FLIGHT_SUFFIX,FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(20000001,30000000),'XX',130,'*', '2023-01-01 00:00:00';
```

```
insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(30000001,40000000),'XX',140,'*', '2023-01-01 00:00:00';

insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(40000001,50000000),'XX',150,'*', '2023-01-01 00:00:00';

insert into perf_user.rnr_segment (airline_iata_code, pnr_number, segment_id,
    FLIGHT_CARRIER, FLIGHT_NUMBER, FLIGHT_SUFFIX, FLIGHT_DATE_LTC)
select 'XX',
upper(substring(concat(md5(random)::text), md5(random)::text)), 0, 7)),
generate_series(50000001,60000000),'XX',160,'*', '2023-01-01 00:00:00';

CREATE INDEX rnr_segment_idx1 ON perf_user.rnr_segment USING btree
    (flight_date_ltc ASC NULLS LAST, bd_airport_code COLLATE pg_catalog."default"
ASC NULLS LAST, off_airport_code COLLATE pg_catalog."default" ASC NULLS LAST,
flight_number ASC NULLS LAST, flight_carrier COLLATE pg_catalog."default" ASC NULLS
LAST, flight_suffix COLLATE pg_catalog."default" ASC NULLS LAST, airline_iata_code
COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC NULLS LAST);

CREATE INDEX rnr_segment_idx2
    ON perf_user.rnr_segment USING btree
    (dep_date_ltc ASC NULLS LAST, flight_number ASC NULLS LAST, bd_airport_code COLLATE
pg_catalog."default" ASC NULLS LAST, off_airport_code COLLATE pg_catalog."default" ASC
NULLS LAST, flight_carrier COLLATE pg_catalog."default" ASC NULLS LAST, flight_suffix
COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC NULLS LAST);

CREATE INDEX rnr_segment_idx3
    ON perf_user.rnr_segment USING btree
    (pnr_number COLLATE pg_catalog."default" ASC NULLS LAST, arr_date_utc ASC NULLS
LAST, airline_iata_code COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date
ASC NULLS LAST);

CREATE INDEX rnr_segment_idx4
    ON perf_user.rnr_segment USING btree
    (dep_date_utc ASC NULLS LAST, added_time ASC NULLS LAST, airline_iata_code COLLATE
pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC NULLS LAST);
```

```
CREATE INDEX rnr_segment_idx5
  ON perf_user.rnr_segment USING btree
  (origin COLLATE pg_catalog."default" ASC NULLS LAST, destination COLLATE
  pg_catalog."default" ASC NULLS LAST, oandd_dep_date_ltc ASC NULLS LAST,
  airline_iata_code COLLATE pg_catalog."default" ASC NULLS LAST, arc_expiry_date ASC
  NULLS LAST);

CREATE INDEX rnr_segment_idx6
  ON perf_user.rnr_segment USING btree
  (pnr_number COLLATE pg_catalog."default" ASC NULLS LAST, oandd_id ASC NULLS LAST,
  segment_id ASC NULLS LAST, airline_iata_code COLLATE pg_catalog."default" ASC NULLS
  LAST, arc_expiry_date ASC NULLS LAST);

vacuum analyze perf_user.rnr_segment;

-----

--Table4:

CREATE TABLE IF NOT EXISTS perf_user.rnr_seat_numbers
(
  airline_iata_code character varying(6) COLLATE pg_catalog."default" NOT NULL,
  pnr_number character varying(15) COLLATE pg_catalog."default" NOT NULL,
  segment_id numeric(25,0) NOT NULL,
  pax_id numeric(25,0) NOT NULL,
  seat_id numeric(25,0) NOT NULL,
  bd_airport_code character varying(5) COLLATE pg_catalog."default",
  off_airport_code character varying(5) COLLATE pg_catalog."default",
  seat_number character varying(5) COLLATE pg_catalog."default",
  seat_status character varying(20) COLLATE pg_catalog."default",
  ssr_id character varying(100) COLLATE pg_catalog."default",
  archive_status smallint DEFAULT (0)::smallint,
  seat_alloc_id numeric(25,0),
  archive_date timestamp(0) without time zone,
  seat_attribute_code character varying(201) COLLATE pg_catalog."default",
  channel_code character varying(20) COLLATE pg_catalog."default",
  arc_expiry_date timestamp(0) without time zone,
  CONSTRAINT rnr_seat_numbers_pk PRIMARY KEY (pnr_number, segment_id, pax_id,
  seat_id, airline_iata_code)
);
```

```

insert into perf_user.rnr_seat_numbers (pnr_number, segment_id, pax_id, seat_id,
  airline_iata_code)
select upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(1,10000000),generate_series(1,10000000),generate_series(1,10000000),'XX';

insert into perf_user.rnr_seat_numbers (pnr_number, segment_id, pax_id, seat_id,
  airline_iata_code)
select upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(10000001,20000000),generate_series(10000001,20000000),generate_series(10000001,

insert into perf_user.rnr_seat_numbers (pnr_number, segment_id, pax_id, seat_id,
  airline_iata_code)
select upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(20000001,30000000),generate_series(20000001,30000000),generate_series(20000001,

insert into perf_user.rnr_seat_numbers (pnr_number, segment_id, pax_id, seat_id,
  airline_iata_code)
select upper(substring(concat(md5(random())::text), md5(random())::text)), 0, 7)),
generate_series(30000001,40000000),generate_series(30000001,40000000),generate_series(30000001,

vacuum Analyze perf_user.rnr_seat_numbers;

--Table5:
CREATE TABLE IF NOT EXISTS perf_user.test_veh
(
  test_veh_id bigint NOT NULL,
  oiltype_id bigint,
  vehicle_id character varying(50) COLLATE pg_catalog."default",
  serviceprogram_id character varying(100) COLLATE pg_catalog."default",
  startdate timestamp without time zone,
  enddate timestamp without time zone,
  last_update_dt timestamp without time zone,
  CONSTRAINT test_veh_pkey PRIMARY KEY (test_veh_id),
  CONSTRAINT test_veh_oiltype_id_fkey FOREIGN KEY (oiltype_id)
    REFERENCES perf_user.oiltype (oiltype_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
  CONSTRAINT test_veh_oiltype_id_fkey1 FOREIGN KEY (oiltype_id)
    REFERENCES perf_user.oiltype (oiltype_id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
);

CREATE INDEX IF NOT EXISTS test_veh_enddate_ind

```

```
ON perf_user.test_veh USING btree
(enddate ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS test_veh_oiltype_id_ind
ON perf_user.test_veh USING btree
(oiltype_id ASC NULLS LAST);

--Table6:
CREATE TABLE IF NOT EXISTS perf_user.oiltype
(
    oiltype_id bigint NOT NULL,
    descr character varying(50) COLLATE pg_catalog."default",
    CONSTRAINT oiltype_pkey PRIMARY KEY (oiltype_id)
);

CREATE INDEX IF NOT EXISTS oiltype_oiltyp_in
ON perf_user.oiltype USING btree
(oiltype_id ASC NULLS LAST);

--Table7:
CREATE TABLE IF NOT EXISTS perf_user.serviceprogram
(
    serial bigint NOT NULL,
    serviceprogram_id character varying(50) COLLATE pg_catalog."default",
    proname character varying(150) COLLATE pg_catalog."default",
    CONSTRAINT serviceprogram_pkey PRIMARY KEY (serial)
);

CREATE INDEX IF NOT EXISTS proname_id_ind
ON perf_user.serviceprogram USING btree
(proname COLLATE pg_catalog."default" ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS serviceprogram_id_ind
ON perf_user.serviceprogram USING btree
(serviceprogram_id COLLATE pg_catalog."default" ASC NULLS LAST);

--Table8:
CREATE TABLE IF NOT EXISTS perf_user.vehicleservicehistory
(
    v_id bigint NOT NULL,
    test_veh_id bigint,
```

```
desc_1 character varying(50) COLLATE pg_catalog."default",
start_date timestamp without time zone,
end_date timestamp without time zone,
CONSTRAINT vehiclesservicehistory_pkey PRIMARY KEY (v_id)
);

CREATE INDEX IF NOT EXISTS veh_end_date_id_ind
ON perf_user.vehiclesservicehistory USING btree
(end_date ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS veh_ser_ind
ON perf_user.vehiclesservicehistory USING btree
(test_veh_id ASC NULLS LAST);

CREATE INDEX IF NOT EXISTS vehiclesservicehistory_v_id_ind
ON perf_user.vehiclesservicehistory USING btree
(test_veh_id ASC NULLS LAST);

--Function creation
CREATE OR REPLACE FUNCTION perf_user.return_data()
RETURNS character varying
LANGUAGE 'plpgsql'
COST 100
VOLATILE PARALLEL UNSAFE
AS $BODY$
BEGIN
return 'EE9F41' ;
END;
$BODY$;

-----
CREATE TABLE IF NOT EXISTS ITEM_DETAILS
(
ITEMID INTEGER,
ORDID INTEGER,
ITEMNAME CHARACTER VARYING(200)
);

CREATE TABLE IF NOT EXISTS ORDER_DETAILS
(
ORDID INTEGER,
ORDNAME CHARACTER VARYING(200),
ORDEREDPLACE CHARACTER VARYING(55)
);
```

```
CREATE TABLE IF NOT EXISTS PAYMENT_DETAILS
(
    PAYID INTEGER,
    ORDID INTEGER,
    PAYPLACE CHARACTER VARYING(55)
);
```

## 사용 사례 1 — 데이터 정렬

데이터베이스에서 데이터 정렬은 데이터 정렬 및 비교 방법을 결정하는 일련의 규칙입니다. 데이터 정렬은 일반적으로 텍스트 값을 비교하기 위한 인덱싱을 위해 여러 언어로 텍스트 데이터가 정렬되는 방식에 적용됩니다. 언어마다 문자 집합과 순서가 다릅니다. 데이터 정렬을 사용하면 올바른 문자 순서를 정의하는 규칙을 사용하여 해당 언어의 문자 데이터를 정렬할 수 있습니다. 다음을 지정할 수도 있습니다.

- 대소문자 구분
- 액센트 마크
- 가나 문자 유형
- 기호 또는 문장 부호 사용
- 문자 너비
- 단어 정렬

조인 열이 다른 데이터 정렬을 사용하는 경우 성능에 영향을 미칠 수 있습니다. 다음 예제 쿼리는 조인 열에 데이터 정렬이 다른 테이블 3개를 사용합니다.

테이블 이름	열 이름
nrn_segment	pnr_number character varying(15) COLLATE pg_catalog."C" NOT NULL
nrn_segment_pax	pnr_number character varying(15) COLLATE pg_catalog."default" NOT NULL

`nr_seat_numbers`

`nr_number character varying(15)`  
`COLLATE pg_catalog."default" NOT`  
`NULL`

```
EXPLAIN ANALYZE SELECT
A.PNR_NUMBER,
A.PAX_ID,
A.SEGMENT_ID,
B.OANDD_ID,
C.SEAT_ID,
C.BD_AIRPORT_CODE,
C.OFF_AIRPORT_CODE,
C.SEAT_NUMBER ,
B.CABIN_CLASS ,
A.SEGMENT_PAX_ID,
C.SEAT_ALLOC_ID,
C.SSR_ID,
C.SEAT_ATTRIBUTE_CODE
from
RNR_SEGMENT_PAX A,
RNR_SEGMENT B,
RNR_SEAT_NUMBERS C
where
B.AIRLINE_IATA_CODE = 'XX'
and B.FLIGHT_CARRIER = 'XX'
and B.FLIGHT_NUMBER = 140
and B.FLIGHT_SUFFIX = '*'
and B.FLIGHT_DATE_LTC = TO_DATE('01-JAN-2023', 'DD-MON-YYYY')
and A.AIRLINE_IATA_CODE = B.AIRLINE_IATA_CODE
and A.PNR_NUMBER = B.PNR_NUMBER
and A.SEGMENT_ID = B.SEGMENT_ID
and C.AIRLINE_IATA_CODE = B.AIRLINE_IATA_CODE
and C.PNR_NUMBER = B.PNR_NUMBER
and C.SEGMENT_ID = B.SEGMENT_ID
and A.PAX_ID = C.PAX_ID
and B.PNR_NUMBER in ('9F1588', 'E37DE0', '04E82B', '813D11', 'BFF10F');
```

이전 쿼리의 쿼리 계획은 테이블의 조인된 열에 대한 적절한 인덱스가 있더라도 `nr_seat_numbers` 테이블에 대한 시퀀스 스캔을 사용합니다. 이렇게 조인된 열이 서로 다른 데이터 정렬을 사용하기 때문에 플래너는 인덱스 스캔을 사용하지 않습니다.

```

Nested Loop (cost=1112.14..927363.51 rows=1 width=833) (actual time=5395.367..5397.253
rows=0 loops=1)
  Join Filter: (((b.pnr_number)::text = (a.pnr_number)::text) AND (b.segment_id =
a.segment_id))
  -> Gather (cost=1111.58..670766.48 rows=1 width=843) (actual
time=5395.367..5397.251 rows=0 loops=1)
    Workers Planned: 2
    Workers Launched: 2
    -> Hash Join (cost=111.58..669766.38 rows=1 width=843) (actual
time=5388.992..5388.993 rows=0 loops=3)
      Hash Cond: (((c.pnr_number)::text = (b.pnr_number)::text) AND
(c.segment_id = b.segment_id))
      -> Parallel Seq Scan on rnr_seat_numbers c (cost=0.00..582154.96
rows=16666637 width=760) (actual time=0.008..2963.019 rows=13333333 loops=3)
        Filter: ((airline_iata_code)::text = 'XX'::text)
      -> Hash (cost=111.52..111.52 rows=4 width=86) (actual time=0.121..0.121
rows=2 loops=3)
        Buckets: 1024 Batches: 1 Memory Usage: 9kB
        -> Index Scan using rnr_segment_pk on rnr_segment b
(cost=0.56..111.52 rows=4 width=86) (actual time=0.082..0.116 rows=2 loops=3)
          Index Cond: (((pnr_number)::text = ANY
('{9F1588,E37DE0,04E82B,813D11,BFF10F}'::text[])) AND ((airline_iata_code)::text =
'XX'::text))
          Filter: (((flight_carrier)::text = 'XX'::text) AND
(flight_number = 140) AND ((flight_suffix)::text = '*'::text) AND (flight_date_ltc =
to_date('01-JAN-2023'::text, 'DD-MON-YYYY'::text)))
          Rows Removed by Filter: 20
        -> Index Scan using rnr_segment_pax_pk on rnr_segment_pax a (cost=0.56..256597.02
rows=1 width=28) (never executed)
          Index Cond: (((airline_iata_code)::text = 'XX'::text) AND (segment_id =
c.segment_id) AND (pax_id = c.pax_id))
          Filter: ((c.pnr_number)::text = (pnr_number)::text)
Planning Time: 0.982 ms
Execution Time: 5397.314 ms

```

테이블 열 정렬을 "C" 언어에서 PostgreSQL에서 제공하는 기본 데이터 정렬로 변경하려면 다음 명령문을 실행한 다음 alter 테이블을 분석하십시오.

```

alter table rnr_segment alter column pnr_number type character varying(15) COLLATE
pg_catalog."default";

Analyze rnr_segment;

```

이제 쿼리 계획이 인덱스 스캔을 사용하므로 런타임이 줄어듭니다.

```
Nested Loop (cost=1.69..146.63 rows=1 width=833) (actual time=0.155..0.155 rows=0
 loops=1)
  -> Nested Loop (cost=1.13..145.89 rows=1 width=111) (actual time=0.154..0.155
 rows=0 loops=1)
    -> Index Scan using rnr_segment_pk on rnr_segment b (cost=0.56..111.51 rows=4
 width=86) (actual time=0.048..0.097 rows=2 loops=1)
      Index Cond: (((pnr_number)::text = ANY
 ('{9F1588,E37DE0,04E82B,813D11,BFF10F}'::text[])) AND ((airline_iata_code)::text =
 'XX'::text))
      Filter: (((flight_carrier)::text = 'XX'::text) AND (flight_number =
 140) AND ((flight_suffix)::text = '*'::text) AND (flight_date_ltc = to_date('01-
JAN-2023'::text, 'DD-MON-YYYY'::text)))
      Rows Removed by Filter: 20
    -> Index Scan using rnr_segment_pax_pk on rnr_segment_pax a (cost=0.56..8.58
 rows=1 width=28) (actual time=0.027..0.027 rows=0 loops=2)
      Index Cond: (((airline_iata_code)::text = 'XX'::text) AND
 ((pnr_number)::text = (b.pnr_number)::text) AND (segment_id = b.segment_id))
    -> Index Scan using rnr_seat_numbers_pk on rnr_seat_numbers c (cost=0.56..0.72
 rows=1 width=760) (never executed)
      Index Cond: (((pnr_number)::text = (a.pnr_number)::text) AND (segment_id =
 a.segment_id) AND (pax_id = a.pax_id) AND ((airline_iata_code)::text = 'XX'::text))
Planning Time: 1.432 ms
Execution Time: 0.207 ms
```

## 사용 사례 2 — 데이터 유형 불일치

데이터를 기반으로 적절한 데이터 유형을 선택하면 스토리지 크기와 성능 간에 최적의 균형을 유지하는 데 도움이 됩니다.

다음 예제 쿼리는 pnr\_number 열을 사용하여 두 테이블을 조인합니다. 테이블마다 pnr\_number 열의 데이터 유형이 다릅니다.

테이블 이름	열 이름 및 데이터 유형
perf_user.rnr_segment_pax	pnr_number character varying(6)
perf_user.rnr_expiry_date	pnr_number character(2)

```
EXPLAIN ANALYZE UPDATE perf_user.RNR_SEGMENT_PAX x SET ARC_EXPIRY_DATE =
y.ARC_EXPIRY_DATE
  FROM (SELECT AIRLINE_IATA_CODE, PNR_NUMBER, ARC_EXPIRY_DATE, 0+row_num ROW_NUM
        FROM perf_user.RNR_EXPIRY_DATE
        WHERE airline_iata_code = 'XX'
        AND row_num BETWEEN (1*5000)+0 AND (1+1)*5000) y
 WHERE x.airline_iata_code = y.airline_iata_code
 AND x.PNR_NUMBER =y.PNR_NUMBER;
```

```
-----
Update on rnr_segment_pax x (cost=290.97..1104986.32 rows=15515 width=460) (actual
time=14574.118..14574.120 rows=0 loops=1)
-> Hash Join (cost=290.97..1104986.32 rows=15515 width=460) (actual
time=16.967..14101.983 rows=11953 loops=1)
  Hash Cond: ((x.pnr_number)::text = (rnr_expiry_date.pnr_number)::text)
  -> Seq Scan on rnr_segment_pax x (cost=0.00..954539.00 rows=40000320
width=446) (actual time=0.011..9702.989 rows=40000000 loops=1)
    Filter: ((airline_iata_code)::bpchar = 'XX'::bpchar)
  -> Hash (cost=225.37..225.37 rows=5248 width=24) (actual time=16.540..16.541
rows=5001 loops=1)
    Buckets: 8192 Batches: 1 Memory Usage: 338kB
    -> Index Scan using rnr_expiry_date_idx1 on rnr_expiry_date
(cost=0.29..225.37 rows=5248 width=24) (actual time=3.102..15.331 rows=5001 loops=1)
      Index Cond: ((row_num >= 5000) AND (row_num <= 10000))
      Filter: (airline_iata_code = 'XX'::bpchar)
Planning Time: 4.445 ms
Execution Time: 14574.322 ms
```

를 EXPLAIN ANALYZE 실행하면 플래너는 조인에 사용된 열에 인덱스가 있더라도 인덱스 스캔 rnr\_segment\_pax 대신 시퀀스 스캔을 사용합니다. 조인에 사용된 열의 길이가 다르기 때문에 플래너는 인덱스 스캔을 사용하지 않습니다.

조인 조건에 관련된 두 테이블의 데이터 유형을 동일하게 유지하도록 테이블 열을 변경한 다음 테이블을 분석하십시오.

```
alter table perf_user.rnr_expiry_date alter column airline_iata_code type character
varying(6) ;

analyze perf_user.rnr_expiry_date;
```

이제 조인 조건에 사용되는 두 열의 테이블 길이가 동일합니다.

EXPLAIN ANALYZE를 다시 실행합니다. 플래너는 인덱스 스캔을 수행하여 쿼리 성능을 크게 개선합니다.

```
Update on rnr_segment_pax x (cost=0.86..59733.09 rows=14637 width=460) (actual
time=416.653..416.654 rows=0 loops=1)
  -> Nested Loop (cost=0.86..59733.09 rows=14637 width=460) (actual
time=0.103..91.106 rows=11953 loops=1)
    -> Index Scan using rnr_expiry_date_idx1 on rnr_expiry_date
(cost=0.29..212.69 rows=4951 width=24) (actual time=0.025..3.023 rows=5001 loops=1)
      Index Cond: ((row_num >= 5000) AND (row_num <= 10000))
      Filter: ((airline_iata_code)::text = 'XX'::text)
    -> Index Scan using rnr_segment_pax_pk on rnr_segment_pax x (cost=0.56..11.99
rows=3 width=446) (actual time=0.014..0.016 rows=2 loops=5001)
      Index Cond: (((airline_iata_code)::text = 'XX'::text) AND
((pnr_number)::text = (rnr_expiry_date.pnr_number)::text))
Planning Time: 0.310 ms
Execution Time: 416.696 ms
```

## 사용 사례 3 — SELECT 문의 함수 호출

where절에서 함수를 호출하면 함수를 호출하는 동안 select 키워드를 사용하지 않고 함수를 호출할 때 쿼리 성능이 저하될 수 있습니다. VOLATILE

```
Select * from tab_name where fieldName = FunctionName(parameters);
```

함수를 호출하는 동안 해당 select 명령문을 사용하면 인덱스 스캔이 실행됩니다.

```
Select * from tab_name where fieldName = ( select FunctionName(parameters) );
```

pnr\_number필드의 rnr\_expiry\_date 테이블 색인이 있습니다. 인덱스는 where 조항의 값을 비교할 때 사용됩니다.

```
explain analyze select * from perf_user.rnr_expiry_date where pnr_number= 'EE9F41';

"Index Scan using rnr_expiry_date_idx3 on rnr_expiry_date (cost=0.29..8.31 rows=1
width=72) (actual time=0.020..0.021 rows=1 loops=1)"
" Index Cond: ((pnr_number)::text = 'EE9F41'::text)"
"Planning Time: 0.063 ms"
"Execution Time: 0.038 ms"
```

필드에 색인을 사용할 수 있는 경우에도 `select` 키워드 없이 함수를 호출하면 순차 스캔이 수행됩니다.

```
explain analyze select * from perf_user.rnr_expiry_date where pnr_number=
perf_user.return_data();

"Seq Scan on rnr_expiry_date (cost=0.00..27084.00 rows=1 width=72) (actual
time=0.112..135.917 rows=1 loops=1)"
"  Filter: ((pnr_number)::text = (perf_user.return_data())::text)"
"  Rows Removed by Filter: 99999"
"Planning Time: 0.053 ms"
"Execution Time: 136.803 ms"
```

`select` 키워드와 함께 함수를 호출하면 인덱스 스캔이 수행됩니다.

```
explain analyze select * from perf_user.rnr_expiry_date where pnr_number= (select
perf_user.return_data() );

"Index Scan using rnr_expiry_date_idx3 on rnr_expiry_date (cost=0.55..8.57 rows=1
width=72) (actual time=0.058..0.061 rows=1 loops=1)"
"  Index Cond: ((pnr_number)::text = ($0)::text)"
"  InitPlan 1 (returns $0)"
"    -> Result (cost=0.00..0.26 rows=1 width=32) (actual time=0.021..0.022 rows=1
loops=1)"
"Planning Time: 0.147 ms"
"Execution Time: 0.111 ms"
```

## 사용 사례 4 — IN 또는 EXISTS

쿼리에 `IN` 또는 `NOT IN` 연산자가 있는 경우 쿼리 계획을 확인하여 적절한 색인이 사용되고 있는지 확인하는 것이 좋습니다. 적절한 인덱스가 사용되지 않고 쿼리 성능이 예상보다 오래 걸리는 경우 `EXISTS` 또는 `NOT EXISTS` 조건을 사용하여 쿼리를 다시 작성해 보세요.

를 사용하는 `NOT IN` 다음 예제를 고려해 보십시오.

```
EXPLAIN ANALYZE SELECT
  TEST_VEH.TEST_VEH_ID,
  TEST_VEH.VEHICLE_ID,
  TEST_VEH.SERVICEPROGRAM_ID,
  TEST_VEH.STARTDATE,
  TEST_VEH.ENDDATE,
```

```

TEST_VEH.OILTYPE_ID
FROM PERF_USER.TEST_VEH TEST_VEH
JOIN PERF_USER.OILTYPE OT ON OT.OILTYPE_ID =TEST_VEH.OILTYPE_ID
JOIN PERF_USER.SERVICEPROGRAM SP ON SP.SERVICEPROGRAM_ID = TEST_VEH.SERVICEPROGRAM_ID
WHERE SP.PROGNAME = '18FCE8FDAF365BB'
      AND OT.OILTYPE_ID =3
      AND TEST_VEH.ENDDATE IS NOT NULL
      AND TEST_VEH.TEST_VEH_ID NOT IN
          (SELECT TEST_VEH_ID
           FROM PERF_USER.VEHICLESERVICEHISTORY
           WHERE TEST_VEH_ID > 1
          );
-----
"Nested Loop (cost=1009.16..1188860356305.01 rows=1 width=76) (actual
time=37299.891..37347.853 rows=0 loops=1)"
"  -> Gather (cost=1009.16..1188860356303.88 rows=1 width=76) (actual
time=37299.890..37347.849 rows=0 loops=1)"
"      Workers Planned: 2"
"      Workers Launched: 2"
"      -> Hash Join (cost=9.16..1188860355303.78 rows=1 width=76) (actual
time=37286.742..37286.751 rows=0 loops=3)"
"          Hash Cond: ((test_veh.serviceprogram_id)::text =
(sp.serviceprogram_id)::text)"
"              -> Parallel Index Scan using test_veh_oiltype_id_ind on test_veh
(cost=0.56..1188860351273.04 rows=1072570 width=76) (actual time=37276.290..37276.292
rows=1 loops=3)"
"                  Index Cond: (oiltype_id = 3)"
"                  Filter: ((enddate IS NOT NULL) AND (NOT (SubPlan 1)))"
"                  Rows Removed by Filter: 0"
"                  SubPlan 1"
"                      -> Materialize (cost=0.00..1025071.31 rows=33333332 width=8)
(actual time=0.418..23201.432 rows=25001498 loops=4)"
"                          -> Seq Scan on vehicleservicehistory
(cost=0.00..728195.65 rows=33333332 width=8) (actual time=0.416..13249.975
rows=25001498 loops=4)"
"                              Filter: (test_veh_id > 1)"
"                                  -> Hash (cost=8.58..8.58 rows=1 width=11) (actual time=9.045..9.046
rows=0 loops=3)"
"                                      Buckets: 1024 Batches: 1 Memory Usage: 8kB"
"                                          -> Index Scan using progname_id_ind on serviceprogram sp
(cost=0.56..8.58 rows=1 width=11) (actual time=9.043..9.044 rows=0 loops=3)"
"                                              Index Cond: ((progname)::text = '18FCE8FDAF365BB'::text)"
"  -> Seq Scan on oiltype ot (cost=0.00..1.12 rows=1 width=8) (never executed)"
"      Filter: (oiltype_id = 3)"

```

```
"Planning Time: 37.696 ms"
"Execution Time: 37366.335 ms"
```

쿼리가 400만 개의 레코드를 검색하는 데 37초 (366밀리초) 이상 걸립니다.

쿼리 계획에는 하위 쿼리에 사용된 테이블에서 시퀀스 스캔이 수행된다고 명시되어 있습니다. `vehicleservicehistory` 시퀀스 스캔은 많은 수의 레코드를 생성합니다. 하위 쿼리의 각 레코드에 대해 쿼리가 전체 테이블 스캔을 수행하므로 성능 문제가 발생합니다.

하위 쿼리에서 시퀀스 스캔을 방지하려면 상호 관련된 하위 쿼리를 사용하도록 하위 쿼리를 다시 작성하십시오. `NOT EXISTS` 상관 관계가 있는 하위 쿼리는 인덱스 스캔을 사용하고 테이블 스캔 횟수를 줄였습니다.

```
EXPLAIN ANALYZE SELECT
  TEST_VEH.TEST_VEH_ID,
  TEST_VEH.VEHICLE_ID,
  TEST_VEH.SERVICEPROGRAM_ID,
  TEST_VEH.STARTDATE,
  TEST_VEH.ENDDATE,
  TEST_VEH.OILTYPE_ID
FROM PERF_USER.TEST_VEH TEST_VEH
JOIN PERF_USER.OILTYPE OT ON OT.OILTYPE_ID =TEST_VEH.OILTYPE_ID
JOIN PERF_USER.SERVICEPROGRAM SP ON SP.SERVICEPROGRAM_ID = TEST_VEH.SERVICEPROGRAM_ID
WHERE SP.PROGNAME = '18FCE8FDAF365BB'
      AND OT.OILTYPE_ID =3
      AND TEST_VEH.ENDDATE IS NOT NULL
      AND NOT EXISTS
          (SELECT TEST_VEH_ID
           FROM PERF_USER.VEHICLESERVICEHISTORY
           WHERE
TEST_VEH.TEST_VEH_ID=VEHICLESERVICEHISTORY.TEST_VEH_ID
           AND TEST_VEH_ID > 1
          );
```

```
-----
"Nested Loop Anti Join (cost=1009.03..936146.10 rows=1 width=76) (actual
time=12.693..12.810 rows=0 loops=1)"
"  -> Nested Loop (cost=1008.59..936141.78 rows=1 width=76) (actual
time=12.692..12.809 rows=0 loops=1)"
"    -> Gather (cost=1008.59..936140.64 rows=1 width=76) (actual
time=12.691..12.807 rows=0 loops=1)"
"      Workers Planned: 2"
"      Workers Launched: 2"
```

```

"          -> Hash Join (cost=8.59..935140.54 rows=1 width=76) (actual
time=0.773..0.774 rows=0 loops=3)"
"          Hash Cond: ((test_veh.serviceprogram_id)::text =
(sp.serviceprogram_id)::text)"
"          -> Parallel Seq Scan on test_veh (cost=0.00..927087.67
rows=2145139 width=76) (actual time=0.672..0.672 rows=1 loops=3)"
"          Filter: ((enddate IS NOT NULL) AND (oiltype_id = 3))"
"          Rows Removed by Filter: 7"
"          -> Hash (cost=8.58..8.58 rows=1 width=11) (actual
time=0.040..0.040 rows=0 loops=3)"
"          Buckets: 1024  Batches: 1  Memory Usage: 8kB"
"          -> Index Scan using progname_id_ind on serviceprogram sp
(cost=0.56..8.58 rows=1 width=11) (actual time=0.039..0.040 rows=0 loops=3)"
"          Index Cond: ((progname)::text =
'18FCE8FDAF365BB'::text)"
"          -> Seq Scan on oiltype ot (cost=0.00..1.12 rows=1 width=8) (never executed)"
"          Filter: (oiltype_id = 3)"
"          -> Index Only Scan using veh_ser_ind on vehicleservicehistory (cost=0.44..4.32
rows=1 width=8) (never executed)"
"          Index Cond: ((test_veh_id = test_veh.test_veh_id) AND (test_veh_id > 1))"
"          Heap Fetches: 0"
"Planning Time: 11.115 ms"
"Execution Time: 12.871 ms"

```

수정 후 쿼리에서 4백만 개의 레코드를 처리하는 데 걸리는 시간은 13ms 미만입니다.

수정된 쿼리의 쿼리 계획에 따르면 테이블에 인덱스 스캔이 있을 `vehicleservicehistory` 수 있습니다. 인덱스 스캔을 사용하면 비용이 절감되고 영향을 받는 행 수가 줄어듭니다. 이렇게 하면 쿼리 런타임을 줄이고 성능을 높일 수 있습니다.

## 사용 사례 5 — 하위 쿼리 또는 CTE

공용 테이블 표현식 (CTE) 은 큰 쿼리를 작은 쿼리로 분류하는 데 도움이 됩니다. 이렇게 하면 전체 쿼리를 더 쉽게 관리할 수 있습니다.

하위 쿼리 조인은 CTE 조인으로 대체되며, CTE 섹션 내에서 쿼리 이름이 지정되고 구분되므로 더 읽기 쉽습니다. 이는 쿼리 크기가 커져 쿼리를 유지 관리하기가 더 어려워질 때 특히 유용합니다. 또한 PostgreSQL의 CTE 결과도 구체화됩니다. 여러 위치에서 CTE를 호출하는 경우 실제 쿼리 정의는 한 번만 실행됩니다. 결과는 메모리에 저장됩니다. 동일한 쿼리의 여러 위치에서 사용해야 하는 복잡한 로직에 이 옵션을 사용할 수 있습니다. 해당 로직을 CTE에 넣고 원하는 횟수만큼 CTE를 호출하세요.

예를 들어, 한 고객이 쿼리 내에 많은 하위 쿼리가 있는 인라인 애플리케이션 쿼리를 사용하고 있었습니다. 애플리케이션에서 전송한 입력 매개 변수 값을 기준으로 하위 쿼리가 필터링되었습니다.

```
EXPLAIN ANALYZE
SELECT * FROM
ORDER_DETAILS A
WHERE A.ORDID IN (SELECT ORDID FROM PAYMENT_DETAILS)
AND A.ORDID IN (SELECT ORDID FROM ITEM_DETAILS )
AND A.ORDID = 1000000;
```

```
"Nested Loop Semi Join (cost=3000.00..194258.21 rows=5 width=74) (actual
time=201.605..747.945 rows=5 loops=1)"
"  -> Nested Loop Semi Join (cost=2000.00..135040.47 rows=5 width=74) (actual
time=146.016..666.779 rows=5 loops=1)"
"    -> Gather (cost=1000.00..78580.31 rows=5 width=74) (actual
time=58.893..463.570 rows=5 loops=1)"
"      Workers Planned: 2"
"      Workers Launched: 2"
"    -> Parallel Seq Scan on order_details a (cost=0.00..77579.81 rows=2
width=74) (actual time=165.627..549.702 rows=2 loops=3)"
"      Filter: (ordid = 1000000)"
"      Rows Removed by Filter: 1666665"
"    -> Materialize (cost=1000.00..56460.07 rows=3 width=4) (actual
time=17.424..40.638 rows=1 loops=5)"
"      -> Gather (cost=1000.00..56460.06 rows=3 width=4) (actual
time=87.113..203.178 rows=1 loops=1)"
"        Workers Planned: 2"
"        Workers Launched: 2"
"      -> Parallel Seq Scan on payment_details (cost=0.00..55459.76
rows=1 width=4) (actual time=174.431..423.792 rows=1 loops=3)"
"        Filter: (ordid = 1000000)"
"        Rows Removed by Filter: 1333002"
"      -> Materialize (cost=1000.00..59217.64 rows=4 width=4) (actual time=11.117..16.231
rows=1 loops=5)"
"        -> Gather (cost=1000.00..59217.62 rows=4 width=4) (actual
time=55.581..81.148 rows=1 loops=1)"
"          Workers Planned: 2"
"          Workers Launched: 2"
"        -> Parallel Seq Scan on item_details (cost=0.00..58217.22 rows=2
width=4) (actual time=287.030..411.004 rows=1 loops=3)"
"          Filter: (ordid = 1000000)"
"          Rows Removed by Filter: 1333080"
"Planning Time: 0.266 ms"
```

```
"Execution Time: 747.986 ms"
```

CTE를 사용하여 하위 쿼리를 수정하고 필수 행 집합만 검색되도록 필터를 추가하면 쿼리 성능이 향상됩니다.

```
EXPLAIN ANALYZE
WITH PAYMENT AS
(
  SELECT * FROM PAYMENT_DETAILS WHERE ORCID = 1000000
),
ITEM AS
(SELECT * FROM ITEM_DETAILS WHERE ORCID = 1000000)
SELECT * FROM
ORDER_DETAILS A JOIN PAYMENT B
ON A.ORCID=B.ORCID
JOIN ITEM C ON B.ORCID=C.ORCID
```

```
"Nested Loop (cost=3000.00..194258.91 rows=60 width=166) (actual time=586.410..732.918
rows=80 loops=1)"
"  -> Nested Loop (cost=2000.00..115677.83 rows=12 width=92) (actual
time=456.760..457.083 rows=16 loops=1)"
"    -> Gather (cost=1000.00..59217.62 rows=4 width=48) (actual
time=153.802..154.060 rows=4 loops=1)"
"      Workers Planned: 2"
"      Workers Launched: 2"
"    -> Parallel Seq Scan on item_details (cost=0.00..58217.22 rows=2
width=48) (actual time=85.417..249.045 rows=1 loops=3)"
"      Filter: (ordid = 1000000)"
"      Rows Removed by Filter: 1333332"
"    -> Materialize (cost=1000.00..56460.07 rows=3 width=44) (actual
time=75.738..75.753 rows=4 loops=4)"
"      -> Gather (cost=1000.00..56460.06 rows=3 width=44) (actual
time=302.947..303.005 rows=4 loops=1)"
"        Workers Planned: 2"
"        Workers Launched: 2"
"      -> Parallel Seq Scan on payment_details (cost=0.00..55459.76
rows=1 width=44) (actual time=184.609..294.784 rows=1 loops=3)"
"        Filter: (ordid = 1000000)"
"        Rows Removed by Filter: 1333332"
"      -> Materialize (cost=1000.00..78580.34 rows=5 width=74) (actual time=8.103..17.238
rows=5 loops=16)"
```

```
"      -> Gather (cost=1000.00..78580.31 rows=5 width=74) (actual
time=129.641..275.795 rows=5 loops=1)"
"      Workers Planned: 2"
"      Workers Launched: 2"
"      -> Parallel Seq Scan on order_details a (cost=0.00..77579.81 rows=2
width=74) (actual time=78.556..268.994 rows=2 loops=3)"
"      Filter: (ordid = 1000000)"
"      Rows Removed by Filter: 1666665"
"Planning Time: 0.108 ms"
"Execution Time: 732.953 ms"
```

다음은 예제 데이터에서 관찰한 내용입니다. 대규모 데이터셋에서 쿼리를 실행하면 성능 차이가 매우 커집니다.

## FAQ

쿼리 성능 조정과 관련하여 자주 제기되는 질문에 대한 답변을 찾아보세요.

### EXPLAIN이란 무엇입니까?

EXPLAIN 쿼리 계획을 생성하기 위해 PostgreSQL 쿼리 (SELECT, UPDATE, INSERT) 앞에 추가하는 키워드입니다. DELETE PostgreSQL 쿼리 계획은 데이터베이스에서 쿼리를 실행하려는 방법을 자세히 설명합니다. 이 계획에는 테이블 스캔 순서, 인덱스 사용 및 조인에 대한 정보가 포함됩니다.

쿼리 계획을 사용하여 잠재적 병목 현상을 식별하고, 쿼리를 최적화하고, 전반적인 성능을 개선할 수 있습니다. 쿼리 계획을 검토할 때는 다음 요소를 고려하세요.

- 테이블 액세스 접근 방식
- 조인 접근법
- 필터 조건
- 정렬 작업
- 인덱스 사용
- Parallelism
- Statistics
- 비용 추정
- 각 단계에서 검색된 행
- 데이터 분산

EXPLAIN에 대한 자세한 내용은 PostgreSQL [설명서](#)를 참조하십시오.

### EXPLAIN ANALYZE는 무엇입니까?

쿼리를 앞에 추가하고 EXPLAIN ANALYZE 쿼리를 실행하면 PostgreSQL은 쿼리를 실행하고 쿼리 계획과 런타임 통계를 모두 반환합니다. 실제 런타임, 각 단계에서 처리된 행 및 기타 관련 정보가 쿼리 계획과 함께 표시됩니다. 쿼리를 실행하면 분석 중에 데이터베이스 성능에 영향을 미칠 수 있으므로 프로덕션 데이터베이스에서 사용할 EXPLAIN ANALYZE 때는 주의해야 합니다.

EXPLAIN [ANALYZE](#)에 대한 자세한 내용은 PostgreSQL [설명서](#)를 참조하십시오.

## PostgreSQL에서 데이터 정렬이란 무엇입니까?

PostgreSQL에서 데이터 정렬은 문자열을 비교하고 정렬하는 방법을 결정하는 일련의 규칙입니다. 데이터 정렬은 언어별 규칙 및 변환을 고려하여 비교 시 문자를 고려하는 순서를 정의합니다.

[데이터 정렬에](#) 대한 자세한 내용은 PostgreSQL 설명서를 참조하십시오.

## CTE란 무엇입니까?

PostgreSQL 데이터베이스에서 CTE (공용 테이블 표현식) 는 참조할 수 있는 이름이 지정된 임시 결과 집합입니다. CTE를 사용하면 복잡한 논리를 더 작고 이름이 지정된 단위로 세분화하여 가독성이 뛰어난 모듈식 SQL 쿼리를 만들 수 있습니다.

[CTE에](#) 대한 자세한 내용은 PostgreSQL 설명서를 참조하십시오.

## PostgreSQL의 함수 카테고리는 무엇입니까?

모든 PostgreSQL 함수에는 변동성 분류가 있으며, 가능성은 다음과 같습니다. VOLATILE STABLE IMMUTABLE

- **VOLATILE** — VOLATILE 함수는 데이터베이스 수정을 비롯한 모든 작업을 수행할 수 있습니다. 동일한 인수를 연속해서 호출하면 다른 결과를 반환할 수 있습니다. 옵티마이저는 이러한 함수의 동작을 가정하지 않습니다. 휘발성 함수를 사용하는 쿼리는 해당 값이 필요한 모든 행에서 함수를 재평가합니다.
- **STABLE** — STABLE 함수는 데이터베이스를 수정할 수 없습니다. 단일 명령문 내의 모든 행에 대해 동일한 인수를 지정하면 동일한 결과를 반환할 수 있습니다. 이 분류를 사용하면 최적화 프로그램이 함수의 여러 호출을 단일 호출로 최적화할 수 있습니다. 특히 인덱스 스캔 조건에서는 이러한 함수가 포함된 식을 사용하는 것이 안전합니다. (인덱스 스캔은 비교 값을 각 행에서 한 번만 평가하는 것이 아니라 한 번만 평가하므로 인덱스 스캔 조건에서 VOLATILE 함수를 사용하는 것은 유효하지 않습니다.)
- **불변성** — IMMUTABLE 함수는 데이터베이스를 수정할 수 없으며 동일한 인수가 주어지면 영원히 동일한 결과를 반환할 수 있습니다. 이 분류를 사용하면 쿼리가 상수 인수를 사용하여 함수를 호출할 때 최적화 프로그램이 함수를 미리 평가할 수 있습니다. 예를 들어 정수 더하기 `SELECT ... WHERE x = 4` 연산자의 기반이 되는 함수가 표시되므로 와 같은 쿼리는 즉시 단순화할 `SELECT ... WHERE x = 2 + 2` 수 있습니다. IMMUTABLE

VOLATILECREATE FUNCTION명령에서 범주를 지정하지 않는 경우 기본값입니다. [함수 유형에](#) 대한 자세한 내용은 PostgreSQL 설명서를 참조하십시오.

# 리소스

## 참조

- [설명](#)
- [설명 사용](#)
- [데이터 정렬 지원](#)
- [WITH 쿼리 \(일반 테이블 표현식\)](#)

## 가이드

- [성능 문제를 방지하기 위한 Amazon RDS 및 Amazon Aurora의 PostgreSQL 데이터베이스 유지 관리 활동](#)
- [아마존 RDS와 아마존 Aurora에서 PostgreSQL 파라미터 조정](#)

## 기여자

다음은 이 문서의 기여자입니다.

- 티루말라 다사리, 수석 컨설턴트 — 데이터베이스, AWS
- 베란자넬루 그란디, 수석 컨설턴트 — 데이터베이스, AWS
- 밤시크리슈나 잠물라, 컨설턴트 — 데이터베이스, AWS
- 스리니바스 포들라처보, 수석 수석 컨설턴트 — 데이터베이스, AWS
- 나가 스리니바스 레디 라불라파티, 컨설턴트 — 데이터베이스, AWS

## 문서 이력

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 [RSS 피드](#)를 구독하십시오.

변경 사항	설명	날짜
<a href="#">최초 게시</a>	—	2024년 4월 23일

# AWS 규범적 지침 용어집

다음은 AWS 규범적 지침에서 제공하는 전략, 가이드 및 패턴에서 일반적으로 사용되는 용어입니다. 용어집 항목을 제안하려면 용어집 끝에 있는 피드백 제공 링크를 사용하십시오.

## 숫자

### 7가지 전략

애플리케이션을 클라우드로 이전하기 위한 7가지 일반적인 마이그레이션 전략 이러한 전략은 Gartner가 2011년에 파악한 5가지 전략을 기반으로 하며 다음으로 구성됩니다.

- 리팩터링/리아키텍트 - 클라우드 네이티브 기능을 최대한 활용하여 애플리케이션을 이동하고 해당 아키텍처를 수정함으로써 민첩성, 성능 및 확장성을 개선합니다. 여기에는 일반적으로 운영 체제와 데이터베이스 이식이 포함됩니다. 예: 온프레미스 Oracle 데이터베이스를 Amazon Aurora PostgreSQL 호환 에디션으로 마이그레이션합니다.
- 리플랫폼(리프트 앤드 리세이프) - 애플리케이션을 클라우드로 이동하고 일정 수준의 최적화를 도입하여 클라우드 기능을 활용합니다. 예:에서 온프레미스 Oracle 데이터베이스를 Oracle용 Amazon Relational Database Service(AmazonRDS)로 마이그레이션합니다 AWS 클라우드.
- 재구매(드롭 앤드 쇼) - 일반적으로 기존 라이선스에서 SaaS 모델로 전환하여 다른 제품으로 전환합니다. 예: 고객 관계 관리(CRM) 시스템을 Salesforce.com 마이그레이션합니다.
- 리호스팅(리프트 앤드 시프트) - 애플리케이션을 변경하지 않고 클라우드로 이동하여 클라우드 기능을 활용합니다. 예:의 EC2 인스턴스에서 온프레미스 Oracle 데이터베이스를 Oracle로 마이그레이션합니다 AWS 클라우드.
- 재배포(하이퍼바이저 수준의 리프트 앤 시프트) - 새 하드웨어를 구매하거나, 애플리케이션을 다시 작성하거나, 기존 운영을 수정하지 않고도 인프라를 클라우드로 이동합니다. 온프레미스 플랫폼에서 동일한 플랫폼의 클라우드 서비스로 서버를 마이그레이션합니다. 예: 마이그레이션 Microsoft Hyper-V 에 대한 애플리케이션입니다 AWS.
- 유지(보관) - 소스 환경에 애플리케이션을 유지합니다. 대규모 리팩터링이 필요하고 해당 작업을 나중에 연기하려는 애플리케이션과 비즈니스 차원에서 마이그레이션할 이유가 없어 유지하려는 레거시 애플리케이션이 여기에 포함될 수 있습니다.
- 사용 중지 - 소스 환경에서 더 이상 필요하지 않은 애플리케이션을 폐기하거나 제거합니다.

# A

## ABAC

[속성 기반 액세스 제어를](#) 참조하세요.

## 추상화된 서비스

[관리형 서비스를](#) 참조하세요.

## ACID

[원자성, 일관성, 격리, 내구성](#)을 참조하세요.

## 능동-능동 마이그레이션

양방향 복제 도구 또는 이중 쓰기 작업을 사용하여 소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되고, 두 데이터베이스 모두 마이그레이션 중 연결 애플리케이션의 트랜잭션을 처리하는 데이터베이스 마이그레이션 방법입니다. 이 방법은 일회성 전환이 필요한 대신 소규모의 제어된 배치로 마이그레이션을 지원합니다. 유연성이 뛰어나지만 [액티브-패시브 마이그레이션](#)보다 더 많은 작업이 필요합니다.

## 능동-수동 마이그레이션

소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되지만 소스 데이터베이스만 연결 애플리케이션의 트랜잭션을 처리하고 데이터는 대상 데이터베이스로 복제되는 데이터베이스 마이그레이션 방법입니다. 대상 데이터베이스는 마이그레이션 중 어떤 트랜잭션도 허용하지 않습니다.

## 집계 함수

행 그룹에서 작동하고 그룹에 대한 단일 반환 값을 계산하는 SQL 함수입니다. 집계 함수의 예로는 SUM 및 MAX가 있습니다.

## AI

[인공 지능](#)을 참조하세요.

## AIOps

[인공 지능 작업을](#) 참조하세요.

## 익명화

데이터세트에서 개인 정보를 영구적으로 삭제하는 프로세스입니다. 익명화는 개인 정보 보호에 도움이 될 수 있습니다. 익명화된 데이터는 더 이상 개인 데이터로 간주되지 않습니다.

## 안티 패턴

솔루션이 다른 솔루션보다 비생산적이거나 비효율적이거나 덜 효과적이어서 반복되는 문제에 자주 사용되는 솔루션입니다.

## 애플리케이션 제어

맬웨어로부터 시스템을 보호하기 위해 승인된 애플리케이션만 사용할 수 있는 보안 접근 방식입니다.

## 애플리케이션 포트폴리오

애플리케이션 구축 및 유지 관리 비용과 애플리케이션의 비즈니스 가치를 비롯하여 조직에서 사용하는 각 애플리케이션에 대한 세부 정보 모음입니다. 이 정보는 [포트폴리오 검색 및 분석 프로세스](#)의 핵심이며 마이그레이션, 현대화 및 최적화할 애플리케이션을 식별하고 우선순위를 정하는 데 도움이 됩니다.

## 인공 지능

컴퓨터 기술을 사용하여 학습, 문제 해결, 패턴 인식 등 일반적으로 인간과 관련된 인지 기능을 수행하는 것을 전문으로 하는 컴퓨터 과학 분야입니다. 자세한 내용은 [What is Artificial Intelligence?](#)를 참조하십시오.

## 인공 지능 작업(AIOps)

기계 학습 기법을 사용하여 운영 문제를 해결하고, 운영 인시던트 및 사용자 개입을 줄이고, 서비스 품질을 높이는 프로세스입니다. AIOps가 마이그레이션 전략에 사용되는 AWS 방법에 대한 자세한 내용은 [운영 통합 가이드](#)를 참조하세요.

## 비대칭 암호화

한 쌍의 키, 즉 암호화를 위한 퍼블릭 키와 복호화를 위한 프라이빗 키를 사용하는 암호화 알고리즘입니다. 퍼블릭 키는 복호화에 사용되지 않으므로 공유할 수 있지만 프라이빗 키에 대한 액세스는 엄격히 제한되어야 합니다.

## 원자성, 일관성, 격리, 내구성(ACID)

오류, 정전 또는 기타 문제가 발생한 경우에도 데이터베이스의 데이터 유효성과 운영 신뢰성을 보장하는 소프트웨어 속성 세트입니다.

## 속성 기반 액세스 제어(ABAC)

부서, 직무, 팀 이름 등의 사용자 속성을 기반으로 세분화된 권한을 생성하는 방식입니다. 자세한 내용은 AWS Identity and Access Management (IAM) 설명서의 [ABAC에 대한 AWS](#) 섹션을 참조하세요.

## 신뢰할 수 있는 데이터 소스

가장 신뢰할 수 있는 정보 소스로 간주되는 기본 버전의 데이터를 저장하는 위치입니다. 익명화, 편집 또는 가명화와 같은 데이터 처리 또는 수정의 목적으로 신뢰할 수 있는 데이터 소스의 데이터를 다른 위치로 복사할 수 있습니다.

## 가용 영역

다른 가용 영역의 장애로부터 격리 AWS 리전 되고 동일한 리전의 다른 가용 영역에 저렴하고 지연 시간이 짧은 네트워크 연결을 제공하는 내 고유 위치입니다.

## AWS 클라우드 채택 프레임워크(AWS CAF)

조직이 클라우드로 성공적으로 전환 AWS 하기 위한 효율적이고 효과적인 계획을 개발하는 데 도움이 되는 지침 및 모범 사례 프레임워크입니다. AWS CAF는 지침을 비즈니스, 사람, 거버넌스, 플랫폼, 보안 및 운영이라는 6가지 중점 영역으로 구성합니다. 비즈니스, 사람 및 거버넌스 관점은 비즈니스 기술과 프로세스에 초점을 맞추고, 플랫폼, 보안 및 운영 관점은 전문 기술과 프로세스에 중점을 둡니다. 예를 들어, 사람 관점은 인사(HR), 직원 배치 기능 및 인력 관리를 담당하는 이해관계자를 대상으로 합니다. 이 관점에서 AWS CAF는 성공적인 클라우드 채택을 위해 조직을 준비하는 데 도움이 되는 인력 개발, 교육 및 커뮤니케이션에 대한 지침을 제공합니다. 자세한 내용은 [AWS CAF 웹 사이트](#) 및 [AWS CAF 백서](#)를 참조하세요.

## AWS 워크로드 검증 프레임워크(AWS WQF)

데이터베이스 마이그레이션 워크로드를 평가하고, 마이그레이션 전략을 권장하고, 작업 견적을 제공하는 도구입니다. AWS WQF는 AWS Schema Conversion Tool (AWS SCT)에 포함되어 있습니다. 데이터베이스 스키마 및 코드 객체, 애플리케이션 코드, 종속성 및 성능 특성을 분석하고 평가 보고서를 제공합니다.

## B

### 잘못된 봇

개인 또는 조직을 방해하거나 해를 입히기 위한 [봇](#)입니다.

### BCP

[비즈니스 연속성 계획](#)을 참조하세요.

## 동작 그래프

리소스 동작과 시간 경과에 따른 상호 작용에 대한 통합된 대화형 뷰입니다. Amazon Detective에서 동작 그래프를 사용하여 실패한 로그인 시도, 의심스러운 API 호출 및 유사한 작업을 검사할 수 있습니다. 자세한 내용은 Detective 설명서의 [Data in a behavior graph](#)를 참조하십시오.

## 빅 엔디안 시스템

가장 중요한 바이트를 먼저 저장하는 시스템입니다. [엔디안니스](#)도 참조하세요.

## 바이너리 분류

바이너리 결과(가능한 두 클래스 중 하나)를 예측하는 프로세스입니다. 예를 들어, ML 모델이 “이 이메일이 스팸인가요, 스팸이 아닌가요?”, ‘이 제품은 책인가요, 자동차인가요?’ 등의 문제를 예측해야 할 수 있습니다.

## 블룸 필터

요소가 세트의 멤버인지 여부를 테스트하는 데 사용되는 메모리 효율성이 높은 확률론적 데이터 구조입니다.

## 블루/그린(Blue/Green) 배포

별개의 동일한 두 환경을 생성하는 배포 전략입니다. 현재 애플리케이션 버전은 한 환경(파란색)에서 실행하고 새 애플리케이션 버전은 다른 환경(녹색)에서 실행합니다. 이 전략은 영향을 최소화하면서 빠르게 롤백하는 데 도움이 됩니다.

## bot

인터넷을 통해 자동화된 작업을 실행하고 인적 활동 또는 상호 작용을 시뮬레이션하는 소프트웨어 애플리케이션입니다. 인터넷에서 정보를 인덱싱하는 웹 크롤러와 같은 일부 봇은 유용하거나 유용합니다. 잘못된 봇이라고 하는 다른 봇은 개인 또는 조직에 방해가 되거나 피해를 입히기 위한 것입니다.

## 봇넷

[맬웨어](#)에 감염되고 [봇](#) 셰이더 또는 봇 운영자라고 하는 단일 당사자의 제어 하에 있는 봇 네트워크입니다. Botnet은 봇과 봇의 영향을 확장하는 가장 잘 알려진 메커니즘입니다.

## 브랜치

코드 리포지토리의 포함된 영역입니다. 리포지토리에 생성되는 첫 번째 브랜치가 기본 브랜치입니다. 기존 브랜치에서 새 브랜치를 생성한 다음 새 브랜치에서 기능을 개발하거나 버그를 수정할 수 있습니다. 기능을 구축하기 위해 생성하는 브랜치를 일반적으로 기능 브랜치라고 합니다. 기능을 출시할 준비가 되면 기능 브랜치를 기본 브랜치에 다시 병합합니다. 자세한 내용은 [브랜치 정보](#)(GitHub 문서)를 참조하세요.

## 브레이크 글래스 액세스

예외적인 상황에서 승인된 프로세스를 통해 사용자가 일반적으로 액세스할 권한이 없는데 액세스할 수 있는 빠른 방법입니다. 자세한 내용은 Well-Architected 지침의 [브로크 글래스 절차 구현](#) 표시기를 AWS 참조하세요.

## 브라운필드 전략

사용자 환경의 기존 인프라 시스템 아키텍처에 브라운필드 전략을 채택할 때는 현재 시스템 및 인프라의 제약 조건을 중심으로 아키텍처를 설계합니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 [그린필드](#) 전략을 혼합할 수 있습니다.

## 버퍼 캐시

가장 자주 액세스하는 데이터가 저장되는 메모리 영역입니다.

## 사업 역량

기업이 가치를 창출하기 위해 하는 일(예: 영업, 고객 서비스 또는 마케팅)입니다. 마이크로서비스 아키텍처 및 개발 결정은 비즈니스 역량에 따라 이루어질 수 있습니다. 자세한 내용은 백서의 [AWS에서 컨테이너화된 마이크로서비스 실행의 비즈니스 역량 중심의 구성화](#) 섹션을 참조하십시오.

## 비즈니스 연속성 계획(BCP)

대규모 마이그레이션과 같은 중단 이벤트가 운영에 미치는 잠재적 영향을 해결하고 비즈니스가 신속하게 운영을 재개할 수 있도록 지원하는 계획입니다.

# C

## CAF

[AWS 클라우드 채택 프레임워크](#)를 참조하세요.

## canary 배포

최종 사용자에게 버전의 느린 증분 릴리스입니다. 확신이 들면 새 버전을 배포하고 현재 버전을 완전히 교체합니다.

## CCoE

[Cloud Center of Excellence](#)를 참조하세요.

## CDC

[데이터 캡처 변경을](#) 참조하세요.

## 데이터 캡처 변경(CDC)

데이터베이스 테이블과 같은 데이터 소스의 변경 내용을 추적하고 변경 사항에 대한 메타데이터를 기록하는 프로세스입니다. 대상 시스템의 변경 사항을 감사하거나 복제하여 동기화를 유지하는 등 CDC 다양한 용도로 사용할 수 있습니다.

## 카오스 엔지니어링

시스템 복원력을 테스트하기 위해 의도적으로 장애 또는 중단 이벤트를 도입합니다. [AWS Fault Injection Service \(AWS FIS\)](#)를 사용하여 AWS 워크로드에 스트레스를 주고 응답을 평가하는 실험을 수행할 수 있습니다.

## CI/CD

[지속적 통합 및 지속적 제공](#)을 참조하세요.

## 분류

예측을 생성하는 데 도움이 되는 분류 프로세스입니다. 분류 문제에 대한 ML 모델은 이산 값을 예측합니다. 이산 값은 항상 서로 다릅니다. 예를 들어, 모델이 이미지에 자동차가 있는지 여부를 평가해야 할 수 있습니다.

## 클라이언트측 암호화

대상에서 데이터를 AWS 서비스 수신하기 전에 로컬에서 데이터를 암호화합니다.

## Cloud Center of Excellence(CCoE)

클라우드 모범 사례 개발, 리소스 동원, 마이그레이션 타임라인 설정, 대규모 혁신을 통한 조직 선도 등 조직 전체에서 클라우드 채택 노력을 추진하는 다분야 팀입니다. 자세한 내용은 AWS 클라우드 엔터프라이즈 전략 블로그의 [CCoE 게시물](#)을 참조하세요.

## 클라우드 컴퓨팅

원격 데이터 스토리지와 IoT 디바이스 관리에 일반적으로 사용되는 클라우드 기술 클라우드 컴퓨팅은 일반적으로 [엣지 컴퓨팅](#) 기술에 연결됩니다.

## 클라우드 운영 모델

IT 조직에서 하나 이상의 클라우드 환경을 구축, 성숙화 및 최적화하는 데 사용되는 운영 모델입니다. 자세한 내용은 [클라우드 운영 모델 구축](#)을 참조하십시오.

## 클라우드 채택 단계

조직이 로 마이그레이션할 때 일반적으로 거치는 4단계: AWS 클라우드

- 프로젝트 - 개념 증명 및 학습 목적으로 몇 가지 클라우드 관련 프로젝트 실행
- 파운데이션 - 클라우드 채택을 확장하기 위한 기본 투자(예: 랜딩 영역 생성, 정의CCoE, 운영 모델 설정)
- 마이그레이션 - 개별 애플리케이션 마이그레이션
- Re-invention - 제품 및 서비스 최적화와 클라우드 혁신

이러한 단계는 Stephen Orban이 블로그 게시물 [The Journey Toward Cloud-First 및 Enterprise Strategy 블로그의 채택 단계에](#) 정의했습니다. AWS 클라우드 AWS 마이그레이션 전략과 어떤 관련이 있는지에 대한 자세한 내용은 [마이그레이션 준비 가이드](#)를 참조하세요.

## CMDB

[구성 관리 데이터베이스](#)를 참조하세요.

## 코드 리포지토리

소스 코드와 설명서, 샘플, 스크립트 등의 기타 자산이 버전 관리 프로세스를 통해 저장되고 업데이트되는 위치입니다. 일반적인 클라우드 리포지토리에는 다음이 포함됩니다. GitHub or Bitbucket Cloud. 코드의 각 버전을 브랜치라고 합니다. 마이크로서비스 구조에서 각 리포지토리는 단일 기능 전용입니다. 단일 CI/CD 파이프라인은 여러 리포지토리를 사용할 수 있습니다.

## 콜드 캐시

비어 있거나, 제대로 채워지지 않았거나, 오래되었거나 관련 없는 데이터를 포함하는 버퍼 캐시입니다. 주 메모리나 디스크에서 데이터베이스 인스턴스를 읽어야 하기 때문에 성능에 영향을 미치며, 이는 버퍼 캐시에서 읽는 것보다 느립니다.

## 콜드 데이터

거의 액세스되지 않고 일반적으로 과거 데이터인 데이터. 이런 종류의 데이터를 쿼리할 때는 일반적으로 느린 쿼리가 허용됩니다. 이 데이터를 성능이 낮고 비용이 저렴한 스토리지 계층 또는 클래스로 옮기면 비용을 절감할 수 있습니다.

## 컴퓨터 비전(CV)

기계 학습을 사용하여 디지털 이미지 및 비디오와 같은 시각적 형식에서 정보를 분석하고 추출하는 [AI](#) 필드입니다. 예를 들어, 온프레미스 카메라 네트워크에 CV를 추가하는 디바이스를 AWS Panorama 제공하고 Amazon SageMaker AI는 CV에 대한 이미지 처리 알고리즘을 제공합니다.

## 구성 드리프트

워크로드의 경우 구성이 예상 상태에서 변경됩니다. 이로 인해 워크로드가 규정 미준수가 될 수 있으며 일반적으로 점진적이고 의도하지 않습니다.

## 구성 관리 데이터베이스(CMDB)

하드웨어 및 소프트웨어 구성 요소와 해당 구성을 포함하여 데이터베이스와 해당 IT 환경에 대한 정보를 저장하고 관리하는 리포지토리입니다. 일반적으로 CMDB 포트폴리오 검색 및 마이그레이션 분석 단계에서의 데이터를 사용합니다.

## 규정 준수 팩

규정 준수 및 보안 검사를 사용자 지정하기 위해 조합할 수 있는 AWS Config 규칙 및 문제 해결 작업의 모음입니다. YAML 템플릿을 사용하여 적합성 팩을 AWS 계정 및 리전 또는 조직 전체에 단일 개체로 배포할 수 있습니다. 자세한 내용은 AWS Config 설명서의 [적합성 팩](#)을 참조하세요.

## 지속적 통합 및 지속적 전달(CI/CD)

소프트웨어 릴리스 프로세스의 소스, 빌드, 테스트, 스테이징 및 프로덕션 단계를 자동화하는 프로세스입니다.는 프로세스를 자동화하고, 생산성을 개선하고, 코드 품질을 개선하고, 더 빠르게 제공할 CI/CD is commonly described as a pipeline. CI/CD 수 있도록 지원합니다. 자세한 내용은 [지속적 전달의 이점](#)을 참조하십시오. CD는 지속적 배포를 의미하기도 합니다. 자세한 내용은 [지속적 전달\(Continuous Delivery\)](#)과 [지속적인 개발](#)을 참조하십시오.

## CV

[컴퓨터 비전을](#) 참조하세요.

## D

### 저장 데이터

스토리지에 있는 데이터와 같이 네트워크에 고정되어 있는 데이터입니다.

### 데이터 분류

중요도와 민감도를 기준으로 네트워크의 데이터를 식별하고 분류하는 프로세스입니다. 이 프로세스는 데이터에 대한 적절한 보호 및 보존 제어를 결정하는 데 도움이 되므로 사이버 보안 위험 관리 전략의 중요한 구성 요소입니다. 데이터 분류는 AWS Well-Architected Framework의 보안 원칙 구성 요소입니다. 자세한 내용은 [데이터 분류](#)를 참조하십시오.

### 데이터 드리프트

프로덕션 데이터와 ML 모델 학습에 사용된 데이터 간의 상당한 차이 또는 시간 경과에 따른 입력 데이터의 의미 있는 변화. 데이터 드리프트는 ML 모델 예측의 전반적인 품질, 정확성 및 공정성을 저하시킬 수 있습니다.

## 전송 중 데이터

네트워크를 통과하고 있는 데이터입니다. 네트워크 리소스 사이를 이동 중인 데이터를 예로 들 수 있습니다.

## 데이터 메시

중앙 집중식 관리 및 거버넌스를 통해 분산되고 분산된 데이터 소유권을 제공하는 아키텍처 프레임워크입니다.

## 데이터 최소화

꼭 필요한 데이터만 수집하고 처리하는 원칙입니다. 에서 데이터를 최소화하면 프라이버시 위험, 비용 및 분석 탄소 발자국을 줄일 AWS 클라우드 수 있습니다.

## 데이터 경계

신뢰할 수 있는 자격 증명만 예상 네트워크에서 신뢰할 수 있는 리소스에 액세스할 수 있도록 하는 AWS 환경의 예방 가드레일 세트입니다. 자세한 내용은 [데이터 경계 구축을 참조하세요 AWS](#).

## 데이터 사전 처리

원시 데이터를 ML 모델이 쉽게 구문 분석할 수 있는 형식으로 변환하는 것입니다. 데이터를 사전 처리한다는 것은 특정 열이나 행을 제거하고 누락된 값, 일관성이 없는 값 또는 중복 값을 처리함을 의미할 수 있습니다.

## 데이터 출처

라이프사이클 전반에 걸쳐 데이터의 출처와 기록을 추적하는 프로세스(예: 데이터 생성, 전송, 저장 방법).

## 데이터 주체

데이터를 수집 및 처리하는 개인입니다.

## 데이터 웨어하우스

분석과 같은 비즈니스 인텔리전스를 지원하는 데이터 관리 시스템입니다. 데이터 웨어하우스에는 일반적으로 많은 양의 기록 데이터가 포함되어 있으며 일반적으로 쿼리 및 분석에 사용됩니다.

## 데이터베이스 정의 언어(DDL)

데이터베이스에서 테이블 및 객체의 구조를 만들거나 수정하기 위한 명령문 또는 명령입니다.

## 데이터베이스 조작 언어(DML)

데이터베이스에서 정보를 수정(삽입, 업데이트 및 삭제)하기 위한 명령문 또는 명령입니다.

## DDL

[데이터베이스 정의 언어](#)를 참조하세요.

### 딥 앙상블

예측을 위해 여러 딥 러닝 모델을 결합하는 것입니다. 딥 앙상블을 사용하여 더 정확한 예측을 얻거나 예측의 불확실성을 추정할 수 있습니다.

### 딥 러닝

여러 계층의 인공 신경망을 사용하여 입력 데이터와 관심 대상 변수 간의 매핑을 식별하는 ML 하위 분야입니다.

### defense-in-depth

네트워크와 그 안의 데이터 기밀성, 무결성 및 가용성을 보호하기 위해 컴퓨터 네트워크 전체에 일련의 보안 메커니즘과 제어를 신중하게 계층화하는 정보 보안 접근 방식입니다. 이 전략을 채택하면 AWS Organizations 구조의 여러 계층에 여러 제어를 AWS 추가하여 리소스를 보호할 수 있습니다. 예를 들어 접근 방식은 다중 인증, 네트워크 세분화 및 암호화를 defense-in-depth 결합할 수 있습니다.

### 위임된 관리자

에서 AWS Organizations 호환되는 서비스는 AWS 멤버 계정을 등록하여 조직의 계정을 관리하고 해당 서비스에 대한 권한을 관리할 수 있습니다. 이러한 계정을 해당 서비스의 위임된 관리자라고 합니다. 자세한 내용과 호환되는 서비스 목록은 AWS Organizations 설명서의 [AWS Organizations 와 함께 사용할 수 있는 AWS 서비스](#)를 참조하십시오.

### 배포

대상 환경에서 애플리케이션, 새 기능 또는 코드 수정 사항을 사용할 수 있도록 하는 프로세스입니다. 배포에는 코드 베이스의 변경 사항을 구현한 다음 애플리케이션 환경에서 해당 코드베이스를 구축하고 실행하는 작업이 포함됩니다.

### 개발 환경

[환경](#)을 참조하세요.

### 탐지 제어

이벤트 발생 후 탐지, 기록 및 알림을 수행하도록 설계된 보안 제어입니다. 이러한 제어는 기존의 예방적 제어를 우회한 보안 이벤트를 알리는 2차 방어선입니다. 자세한 내용은 Implementing security controls on AWS의 [Detective controls](#)를 참조하십시오.

## 개발 값 스트림 매핑(DVSM)

소프트웨어 개발 수명 주기에서 속도와 품질에 부정적인 영향을 미치는 제약 조건을 식별하고 우선 순위를 지정하는 데 사용되는 프로세스입니다.는 원래 린 제조 관행을 위해 설계된 가치 흐름 매핑 프로세스를 DVSM 확장합니다. 소프트웨어 개발 프로세스를 통해 가치를 창출하고 이동하는 데 필요한 단계와 팀에 중점을 둡니다.

## 디지털 트윈

건물, 공장, 산업 장비 또는 생산 라인과 같은 실제 시스템을 가상으로 표현한 것입니다. 디지털 트윈은 예측 유지 보수, 원격 모니터링, 생산 최적화를 지원합니다.

## 차원 테이블

[스타 스키마](#)에서는 팩트 테이블의 정량적 데이터에 대한 데이터 속성을 포함하는 더 작은 테이블입니다. 차원 테이블 속성은 일반적으로 텍스트 필드 또는 텍스트처럼 동작하는 개별 숫자입니다. 이러한 속성은 일반적으로 쿼리 제약, 필터링 및 결과 세트 레이블 지정에 사용됩니다.

## 재해

워크로드 또는 시스템이 기본 배포 위치에서 비즈니스 목표를 달성하지 못하게 방해하는 이벤트입니다. 이러한 이벤트는 자연재해, 기술적 오류, 의도하지 않은 구성 오류 또는 멀웨어 공격과 같은 사람의 행동으로 인한 결과일 수 있습니다.

## 재해 복구(DR)

[재해](#)로 인한 가동 중지 시간과 데이터 손실을 최소화하는 데 사용하는 전략 및 프로세스입니다. 자세한 내용은 AWS Well-Architected Framework의 [재해 워크로드 복구 AWS: 클라우드에서 복구를 참조하세요](#).

## DML

[데이터베이스 조작 언어](#)를 참조하세요.

## 도메인 기반 설계

구성 요소를 각 구성 요소가 제공하는 진화하는 도메인 또는 핵심 비즈니스 목표에 연결하여 복잡한 소프트웨어 시스템을 개발하는 접근 방식입니다. 이 개념은 에릭 에반스에 의해 그의 저서인 도메인 기반 디자인: 소프트웨어 중심의 복잡성 해결(Boston: Addison-Wesley Professional, 2003)에서 소개되었습니다. 스트랭글러 무화과 패턴으로 도메인 기반 설계를 사용하는 방법에 대한 자세한 내용은 [컨테이너 및 Amazon API Gateway를 사용하여 레거시 Microsoft ASP.NET \(ASMX\) 웹 서비스 증분 현대화를 참조하세요](#).

## DR

[재해 복구](#)를 참조하세요.

## 드리프트 감지

기존 구성과의 편차 추적. 예를 들어 AWS CloudFormation 를 사용하여 [시스템 리소스의 드리프트를 감지](#)하거나 사용하여 AWS Control Tower 거버넌스 요구 사항 준수에 영향을 미칠 수 있는 [랜딩 영역의 변경 사항을 감지](#)할 수 있습니다.

## DVSM

[개발 값 스트림 매핑](#)을 참조하세요.

## E

### EDA

[탐색 데이터 분석](#)을 참조하세요.

### EDI

[전자 데이터 교환](#)을 참조하세요.

### 엣지 컴퓨팅

IoT 네트워크의 엣지에서 스마트 디바이스의 컴퓨팅 성능을 개선하는 기술 [클라우드 컴퓨팅](#)과 비교할 때 엣지 컴퓨팅은 통신 지연 시간을 줄이고 응답 시간을 개선할 수 있습니다.

### 전자 데이터 교환(EDI)

조직 간의 비즈니스 문서 자동 교환. 자세한 내용은 [전자 데이터 교환이란 무엇입니까?](#)를 참조하세요.

### 암호화

사람이 읽을 수 있는 일반 텍스트 데이터를 암호 텍스트로 변환하는 컴퓨팅 프로세스입니다.

### 암호화 키

암호화 알고리즘에 의해 생성되는 무작위 비트의 암호화 문자열입니다. 키의 길이는 다양할 수 있으며 각 키는 예측할 수 없고 고유하게 설계되었습니다.

### 엔디안

컴퓨터 메모리에 바이트가 저장되는 순서입니다. 빅 엔디안 시스템은 가장 중요한 바이트를 먼저 저장합니다. 리틀 엔디안 시스템은 가장 덜 중요한 바이트를 먼저 저장합니다.

### 엔드포인트

[서비스 엔드포인트](#)를 참조하세요.

## 엔드포인트 서비스

가상 프라이빗 클라우드(VPC)에서 호스팅하여 다른 사용자와 공유할 수 있는 서비스입니다. 이를 사용하여 엔드포인트 서비스를 생성하고 다른 AWS 계정 또는 AWS Identity and Access Management (IAM) 보안 주체에 권한을 AWS PrivateLink 부여할 수 있습니다. 이러한 계정 또는 보안 주체는 인터페이스 엔드포인트를 생성하여 VPC 엔드포인트 서비스에 비공개로 연결할 수 있습니다. 자세한 내용은 Amazon Virtual Private Cloud(AmazonVPC) 설명서의 [엔드포인트 서비스 생성을 참조하세요](#).

## 엔터프라이즈 리소스 계획(ERP)

엔터프라이즈의 주요 비즈니스 프로세스(예: 회계, 및 프로젝트 관리)를 자동화 [MES](#) 하고 관리하는 시스템입니다.

## 봉투 암호화

암호화 키를 다른 암호화 키로 암호화하는 프로세스입니다. 자세한 내용은 AWS Key Management Service (AWS KMS) 설명서의 [봉투 암호화](#)를 참조하세요.

## 환경

실행 중인 애플리케이션의 인스턴스입니다. 다음은 클라우드 컴퓨팅의 일반적인 환경 유형입니다.

- 개발 환경 - 애플리케이션 유지 관리를 담당하는 핵심 팀만 사용할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. 개발 환경은 변경 사항을 상위 환경으로 승격하기 전에 테스트하는 데 사용됩니다. 이러한 유형의 환경을 테스트 환경이라고도 합니다.
- 하위 환경 - 초기 빌드 및 테스트에 사용되는 환경을 비롯한 애플리케이션의 모든 개발 환경입니다.
- 프로덕션 환경 - 최종 사용자가 액세스할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. CI/CD 파이프라인에서 프로덕션 환경이 마지막 배포 환경입니다.
- 상위 환경 - 핵심 개발 팀 이외의 사용자가 액세스할 수 있는 모든 환경입니다. 프로덕션 환경, 프로덕션 이전 환경 및 사용자 수용 테스트를 위한 환경이 여기에 포함될 수 있습니다.

## 에픽

애자일 방법론에서 작업을 구성하고 우선순위를 정하는 데 도움이 되는 기능적 범주입니다. 에픽은 요구 사항 및 구현 작업에 대한 개괄적인 설명을 제공합니다. 예를 들어 보안 AWS CAF 에픽에는 자격 증명 및 액세스 관리, 탐지 제어, 인프라 보안, 데이터 보호 및 인시던트 대응이 포함됩니다. AWS 마이그레이션 전략의 에픽에 대한 자세한 내용은 [프로그램 구현 가이드](#)를 참조하십시오.

## ERP

[엔터프라이즈 리소스 계획을 참조하세요](#).

## 탐색 데이터 분석(EDA)

데이터 세트를 분석하여 주요 특성을 파악하는 프로세스입니다. 데이터를 수집하거나 집계한 다음 초기 조사를 수행하여 패턴을 찾고, 이상을 감지하고, 가정을 확인합니다. EDA는 요약 통계를 계산하고 데이터 시각화를 생성하여 수행됩니다.

## F

### 팩트 테이블

[별표 스키마](#)의 중앙 테이블입니다. 비즈니스 운영에 대한 정량적 데이터를 저장합니다. 일반적으로 팩트 테이블에는 측정값이 포함된 열과 차원 테이블에 대한 외래 키가 포함된 열의 두 가지 유형이 있습니다.

### 빠른 실패

자주 증분 테스트를 사용하여 개발 수명 주기를 줄이는 철학입니다. 애자일 접근 방식의 중요한 부분입니다.

### 장애 격리 경계

에서 장애의 영향을 제한하고 워크로드의 복원력을 개선하는 데 도움이 되는 가용 영역, AWS 리전 제어 영역 또는 데이터 영역과 같은 AWS 클라우드경계입니다. 자세한 내용은 [AWS 장애 격리 경계를 참조하세요](#).

### 기능 브랜치

[브랜치를 참조하세요](#).

### 기능

예측에 사용하는 입력 데이터입니다. 예를 들어, 제조 환경에서 기능은 제조 라인에서 주기적으로 캡처되는 이미지일 수 있습니다.

### 기능 중요도

모델의 예측에 특성이 얼마나 중요한지를 나타냅니다. 이는 일반적으로 Shapley Additive Explanations(SHAP) 및 통합 그라데이션과 같은 다양한 기술을 통해 계산할 수 있는 숫자 점수로 표현됩니다. 자세한 내용은 [기계 학습 모델 해석 가능성을 참조하세요 AWS](#).

### 기능 변환

추가 소스로 데이터를 보강하거나, 값을 조정하거나, 단일 데이터 필드에서 여러 정보 세트를 추출하는 등 ML 프로세스를 위해 데이터를 최적화하는 것입니다. 이를 통해 ML 모델이 데이터를 활용

할 수 있습니다. 예를 들어, 날짜 '2021-05-27 00:15:37'을 '2021년', '5월', '목', '15일'로 분류하면 학습 알고리즘이 다양한 데이터 구성 요소와 관련된 미묘한 패턴을 학습하는 데 도움이 됩니다.

## 몇 번의 샷 프롬프트

[LLM](#)에 유사한 작업을 수행하도록 요청하기 전에 작업과 원하는 출력을 보여주는 몇 가지 예제를 제공합니다. 이 기법은 컨텍스트 내 학습을 적용하여 모델이 프롬프트에 포함된 예제(샷)에서 학습합니다. 소수의 샷 프롬프트는 특정 형식 지정, 추론 또는 도메인 지식이 필요한 작업에 효과적일 수 있습니다. [제로샷 프롬프트도 참조하세요.](#)

## FGAC

[세분화된 액세스 제어를 참조하세요.](#)

### 세분화된 액세스 제어(FGAC)

여러 조건을 사용하여 액세스 요청을 허용하거나 거부합니다.

### 플래시컷 마이그레이션

단계적 접근 방식을 사용하는 대신 [변경 데이터 캡처](#)를 통해 지속적인 데이터 복제를 사용하여 가능한 최단 시간 내에 데이터를 마이그레이션하는 데이터베이스 마이그레이션 방법입니다. 목표는 가동 중지 시간을 최소화하는 것입니다.

## FM

[파운데이션 모델을 참조하세요.](#)

### 파운데이션 모델(FM)

일반화된 데이터와 레이블이 지정되지 않은 데이터의 대규모 데이터 세트에 대해 훈련해 온 대규모 딥 러닝 신경망입니다. FMs는 언어 이해, 텍스트 및 이미지 생성, 자연어 대화와 같은 다양한 일반 작업을 수행할 수 있습니다. 자세한 내용은 [파운데이션 모델이란 무엇입니까?](#)를 참조하세요.

## G

### 생성형 AI

대량의 데이터에 대해 훈련되었으며 간단한 텍스트 프롬프트를 사용하여 이미지, 비디오, 텍스트 및 오디오와 같은 새로운 콘텐츠 및 아티팩트를 생성할 수 있는 [AI](#) 모델의 하위 집합입니다. 자세한 내용은 [생성형 AI란 무엇입니까?](#)를 참조하세요.

### 지리적 차단

[지리적 제한을 참조하세요.](#)

## 지리적 제한(지리적 차단)

Amazon에서는 특정 국가의 사용자가 콘텐츠 배포에 액세스하지 못하도록 하는 CloudFront 옵션입니다. 허용 목록 또는 차단 목록을 사용하여 승인된 국가와 차단된 국가를 지정할 수 있습니다. 자세한 내용은 CloudFront 설명서 [의 콘텐츠의 지리적 배포 제한을 참조하세요](#).

## Gitflow 워크플로

하위 환경과 상위 환경이 소스 코드 리포지토리의 서로 다른 브랜치를 사용하는 방식입니다. Gitflow 워크플로는 레거시로 간주되며 [트렁크 기반 워크플로](#)는 현대적이고 선호하는 접근 방식입니다.

## 골든 이미지

시스템 또는 소프트웨어의 새 인스턴스를 배포하기 위한 템플릿으로 사용되는 시스템 또는 소프트웨어의 스냅샷입니다. 예를 들어 제조에서 황금 이미지를 사용하여 여러 디바이스에 소프트웨어를 프로비저닝할 수 있으며 디바이스 제조 작업의 속도, 확장성 및 생산성을 개선할 수 있습니다.

## 브라운필드 전략

새로운 환경에서 기존 인프라의 부재 시스템 아키텍처에 대한 그린필드 전략을 채택할 때 [브라운필드](#)라고도 하는 기존 인프라와의 호환성 제한 없이 모든 새로운 기술을 선택할 수 있습니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 그린필드 전략을 혼합할 수 있습니다.

## 가드레일

조직 단위() 전반의 리소스, 정책 및 규정 준수를 관리하는 데 도움이 되는 상위 수준 규칙입니다 OUs. 예방 가드레일은 규정 준수 표준에 부합하도록 정책을 시행하며, 서비스 제어 정책 및 IAM 권한 경계를 사용하여 구현됩니다. 탐지 가드레일은 정책 위반 및 규정 준수 문제를 감지하고 해결을 위한 알림을 생성하며, 이는 AWS Config, Amazon AWS Security Hub, GuardDuty, Amazon AWS Trusted Advisor Amazon Inspector 및 사용자 지정 AWS Lambda 검사를 사용하여 구현됩니다.

# H

## HA

[고가용성을 참조하세요](#).

## 이기종 데이터베이스 마이그레이션

다른 데이터베이스 엔진을 사용하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Oracle에서 Amazon Aurora로) 이기종 마이그레이션은 일반적으로 리아키텍트 작업의 일부이며 스

키마를 변환하는 것은 복잡한 작업일 수 있습니다. AWS 는 스키마 변환에 도움이 되는 [AWS SCT를](#) [제공](#)합니다.

## 높은 가용성(HA)

문제나 재해 발생 시 개입 없이 지속적으로 운영할 수 있는 워크로드의 능력. HA 시스템은 자동으로 장애 조치되고, 지속적으로 고품질 성능을 제공하고, 성능에 미치는 영향을 최소화하면서 다양한 부하와 장애를 처리하도록 설계되었습니다.

## 히스토리언 현대화

제조 산업의 요구 사항을 더 잘 충족하도록 운영 기술(OT) 시스템을 현대화하고 업그레이드하는 데 사용되는 접근 방식입니다. 히스토리언은 공장의 다양한 출처에서 데이터를 수집하고 저장하는 데 사용되는 일종의 데이터베이스입니다.

## 홀드아웃 데이터

[기계 학습](#) 모델을 훈련하는 데 사용되는 데이터 세트에서 보류된 레이블이 지정된 기록 데이터의 일부입니다. 홀드아웃 데이터를 사용하여 모델 예측을 홀드아웃 데이터와 비교하여 모델 성능을 평가할 수 있습니다.

## 동종 데이터베이스 마이그레이션

소스 데이터베이스를 동일한 데이터베이스 엔진을 공유하는 대상 데이터베이스로 마이그레이션 (예: Microsoft SQL Server에서 Amazon RDS for SQL Server로). 동종 마이그레이션은 일반적으로 리호스팅 또는 리플랫폼 작업의 일부입니다. 네이티브 데이터베이스 유틸리티를 사용하여 스키마를 마이그레이션할 수 있습니다.

## 핫 데이터

자주 액세스하는 데이터(예: 실시간 데이터 또는 최근 번역 데이터). 일반적으로 이 데이터에는 빠른 쿼리 응답을 제공하기 위한 고성능 스토리지 계층 또는 클래스가 필요합니다.

## 핫픽스

프로덕션 환경의 중요한 문제를 해결하기 위한 긴급 수정입니다. 긴급성으로 인해 핫픽스는 일반적으로 일반적인 DevOps 릴리스 워크플로 외부에서 이루어집니다.

## 하이퍼케어 기간

전환 직후 마이그레이션 팀이 문제를 해결하기 위해 클라우드에서 마이그레이션된 애플리케이션을 관리하고 모니터링하는 기간입니다. 일반적으로 이 기간은 1~4일입니다. 하이퍼케어 기간이 끝나면 마이그레이션 팀은 일반적으로 애플리케이션에 대한 책임을 클라우드 운영 팀에 넘깁니다.

## 정보

### laC

[인프라를 코드로](#) 참조하세요.

#### 자격 증명 기반 정책

AWS 클라우드 환경 내에서 권한을 정의하는 하나 이상의 IAM 보안 주체에 연결된 정책입니다.

#### 유휴 애플리케이션

90일 동안 평균 CPU 및 메모리 사용량이 5~20%인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하거나 온프레미스에 유지하는 것이 일반적입니다.

### IIoT

[산업용 사물 인터넷을](#) 참조하세요.

#### 변경할 수 없는 인프라

기존 인프라를 업데이트, 패치 또는 수정하는 대신 프로덕션 워크로드를 위한 새 인프라를 배포하는 모델입니다. 변경 가능한 인프라는 본질적으로 [변경 가능한 인프라](#)보다 더 일관되고 안정적이며 예측 가능합니다. 자세한 내용은 AWS Well-Architected Framework의 [변경할 수 없는 인프라를 사용한 배포](#) 모범 사례를 참조하세요.

#### 인바운드(수신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 외부에서 네트워크 연결을 수락, 검사 및 라우팅VPC 하는입니다. [AWS 보안 참조 아키텍처](#)는 애플리케이션과 더 광범위한 인터넷 간의 양방향 인터페이스를 보호하기 VPCs 위해 인바운드, 아웃바운드 및 검사로 네트워크 계정을 설정하는 것이 좋습니다.

#### 증분 마이그레이션

한 번에 전체 전환을 수행하는 대신 애플리케이션을 조금씩 마이그레이션하는 전환 전략입니다. 예를 들어, 처음에는 소수의 마이크로서비스나 사용자만 새 시스템으로 이동할 수 있습니다. 모든 것이 제대로 작동하는지 확인한 후에는 레거시 시스템을 폐기할 수 있을 때까지 추가 마이크로서비스 또는 사용자를 점진적으로 이동할 수 있습니다. 이 전략을 사용하면 대규모 마이그레이션과 관련된 위험을 줄일 수 있습니다.

#### Industry 4.0

연결, 실시간 데이터, 자동화, 분석 및 AI/ML의 발전을 통해 제조 프로세스의 현대화를 언급하기 위해 2016년에 [Klaus Schwab](#)에서 도입한 용어입니다.

## 인프라

애플리케이션의 환경 내에 포함된 모든 리소스와 자산입니다.

### 코드형 인프라(IaC)

구성 파일 세트를 통해 애플리케이션의 인프라를 프로비저닝하고 관리하는 프로세스입니다. IaC는 새로운 환경의 반복 가능성, 신뢰성 및 일관성을 위해 인프라 관리를 중앙 집중화하고, 리소스를 표준화하고, 빠르게 확장할 수 있도록 설계되었습니다.

### 산업용 사물 인터넷(IIoT)

제조, 에너지, 자동차, 의료, 생명과학, 농업 등의 산업 부문에서 인터넷에 연결된 센서 및 디바이스의 사용 자세한 내용은 [산업 사물 인터넷 구축\(IIoT\) 디지털 혁신 전략을 참조하세요](#).

### 검사 VPC

AWS 다중 계정 아키텍처에서는 VPCs (동일하거나 다른 AWS 리전), 인터넷 및 온프레미스 네트워크 간의 네트워크 트래픽 검사를 VPC 관리하는 중앙 집중식 아키텍처입니다. [AWS 보안 참조 아키텍처](#)는 애플리케이션과 광범위한 인터넷 간의 양방향 인터페이스를 보호하기 VPCs 위해 인바운드, 아웃바운드 및 검사로 네트워크 계정을 설정하는 것이 좋습니다.

### 사물 인터넷(IoT)

인터넷이나 로컬 통신 네트워크를 통해 다른 디바이스 및 시스템과 통신하는 센서 또는 프로세서가 내장된 연결된 물리적 객체의 네트워크 자세한 내용은 [IoT란?](#)을 참조하십시오.

### 해석력

모델의 예측이 입력에 따라 어떻게 달라지는지를 사람이 이해할 수 있는 정도를 설명하는 기계 학습 모델의 특성입니다. 자세한 내용은 [를 사용한 기계 학습 모델 해석 가능성을 참조하세요](#)AWS.

### IoT

[사물 인터넷](#)을 참조하세요.

### IT 정보 라이브러리(ITIL)

IT 서비스를 제공하고 이러한 서비스를 비즈니스 요구 사항에 맞추기 위한 모범 사례 세트입니다. ITIL는의 기반을 제공합니다ITSM.

### IT 서비스 관리(ITSM)

조직의 IT 서비스 설계, 구현, 관리 및 지원과 관련된 활동 클라우드 작업을 ITSM 도구와 통합하는 방법에 대한 자세한 내용은 [작업 통합 가이드](#)를 참조하세요.

## ITIL

[IT 정보 라이브러리](#)를 참조하세요.

## ITSM

[IT 서비스 관리](#)를 참조하세요.

## L

### 레이블 기반 액세스 제어(LBAC)

사용자와 데이터 자체에 각각 보안 레이블 값이 명시적으로 할당되는 필수 액세스 제어(MAC)의 구현입니다. 사용자 보안 레이블과 데이터 보안 레이블 간의 교차 부분에 따라 사용자가 볼 수 있는 행과 열이 결정됩니다.

### 랜딩 존

랜딩 존은 확장 가능하고 안전한 잘 설계된 다중 계정 AWS 환경입니다. 조직은 여기에서부터 보안 및 인프라 환경에 대한 확신을 가지고 워크로드와 애플리케이션을 신속하게 시작하고 배포할 수 있습니다. 랜딩 존에 대한 자세한 내용은 [안전하고 확장 가능한 다중 계정 AWS 환경 설정](#)을 참조하십시오.

### 대형 언어 모델(LLM)

방대한 양의 데이터에 대해 사전 훈련된 딥 러닝 [AI](#) 모델입니다. 는 질문에 답변, 문서 요약, 텍스트를 다른 언어로 번역, 문장 완성과 같은 여러 작업을 수행할 LLM 수 있습니다. 자세한 내용은 [정의 섹션을 참조하세요LLMs](#).

### 대규모 마이그레이션

300대 이상의 서버 마이그레이션입니다.

### LBAC

[레이블 기반 액세스 제어를](#) 참조하세요.

### 최소 권한

작업을 수행하는 데 필요한 최소 권한을 부여하는 보안 모범 사례입니다. 자세한 내용은 IAM 설명서의 [최소 권한 적용](#)을 참조하세요.

### 리프트 앤드 시프트

[7R](#)을 참조하세요.

## 리틀 엔디안 시스템

가장 덜 중요한 바이트를 먼저 저장하는 시스템입니다. [엔디안니스](#)도 참조하세요.

## LLM

[대형 언어 모델을](#) 참조하세요.

## 하위 환경

[환경을](#) 참조하세요.

## M

### 기계 학습(ML)

패턴 인식 및 학습에 알고리즘과 기법을 사용하는 인공지능의 한 유형입니다. ML은 사물 인터넷 (IoT) 데이터와 같은 기록된 데이터를 분석하고 학습하여 패턴을 기반으로 통계 모델을 생성합니다. 자세한 내용은 [기계 학습](#)을 참조하십시오.

### 기본 브랜치

[브랜치를](#) 참조하세요.

### 맬웨어

컴퓨터 보안 또는 개인 정보 보호를 손상하도록 설계된 소프트웨어입니다. 맬웨어는 컴퓨터 시스템을 중단하거나, 민감한 정보를 유출하거나, 무단 액세스를 가져올 수 있습니다. 맬웨어의 예로는 바이러스, 웜, 랜섬웨어, 트로이 목마, 스파이웨어 및 키로거가 있습니다.

### 관리형 서비스

AWS 서비스가 인프라 계층, 운영 체제 및 플랫폼을 AWS 작동하고 엔드포인트에 액세스하여 데이터를 저장하고 검색합니다. Amazon Simple Storage Service(Amazon S3) 및 Amazon DynamoDB는 관리형 서비스의 예입니다. 이를 추상화된 서비스라고도 합니다.

### 제조 실행 시스템(MES)

원재료를 작업 현장의 완성 제품으로 변환하는 생산 프로세스를 추적, 모니터링, 문서화 및 제어하기 위한 소프트웨어 시스템입니다.

## MAP

[마이그레이션 가속화 프로그램을](#) 참조하세요.

## 메커니즘

도구를 생성하고 도구 채택을 유도한 다음 결과를 검사하여 조정하는 전체 프로세스입니다. 메커니즘은 작동 시 자체를 강화하고 개선하는 주기입니다. 자세한 내용은 AWS Well-Architected Framework의 [메커니즘 구축](#)을 참조하세요.

## 멤버 계정

조직의 일부인 관리 계정을 AWS 계정 제외한 모든 계정. AWS Organizations 하나의 계정은 한 번에 하나의 조직 멤버만 될 수 있습니다.

## MES

[제조 실행 시스템](#)을 참조하세요.

## 메시지 대기열 원격 측정 전송(MQTT)

리소스가 제한된 IoT 디바이스에 대한 [게시/구독](#) 패턴을 기반으로 하는 경량 machine-to-machine(M2M) 통신 프로토콜입니다.

## 마이크로서비스

잘 정의된를 통해 통신APIs하고 일반적으로 소규모 독립 팀이 소유하는 소규모 독립 서비스입니다. 예를 들어, 보험 시스템에는 영업, 마케팅 등의 비즈니스 역량이나 구매, 청구, 분석 등의 하위 영역에 매핑되는 마이크로 서비스가 포함될 수 있습니다. 마이크로서비스의 이점으로 민첩성, 유연한 확장, 손쉬운 배포, 재사용 가능한 코드, 복원력 등이 있습니다. 자세한 내용은 [AWS 서버리스 서비스를 사용하여 마이크로서비스 통합을 참조하세요](#).

## 마이크로서비스 아키텍처

각 애플리케이션 프로세스를 마이크로서비스로 실행하는 독립 구성 요소를 사용하여 애플리케이션을 구축하는 접근 방식입니다. 이러한 마이크로서비스는 경량를 사용하여 잘 정의된 인터페이스를 통해 통신합니다APIs. 애플리케이션의 특정 기능에 대한 수요에 맞게 이 아키텍처의 각 마이크로서비스를 업데이트, 배포 및 조정할 수 있습니다. 자세한 내용은 [에서 마이크로서비스 구현을 참조하세요 AWS](#).

## 마이그레이션 가속화 프로그램(MAP)

조직이 클라우드로 전환하기 위한 강력한 운영 기반을 구축하고 초기 마이그레이션 비용을 상쇄하는 데 도움이 되는 컨설팅 지원, 교육 및 서비스를 제공하는 AWS 프로그램입니다. 이는 체계적인 방식으로 레거시 마이그레이션을 실행하기 위한 마이그레이션 방법론과 일반적인 마이그레이션 시나리오를 자동화하고 가속화하기 위한 도구 세트가 MAP 포함되어 있습니다.

## 대규모 마이그레이션

애플리케이션 포트폴리오의 대다수를 웨이브를 통해 클라우드로 이동하는 프로세스로, 각 웨이브에서 더 많은 애플리케이션이 더 빠른 속도로 이동합니다. 이 단계에서는 이전 단계에서 배운 모범 사례와 교훈을 사용하여 팀, 도구 및 프로세스의 마이그레이션 팩토리를 구현하여 자동화 및 민첩한 제공을 통해 워크로드 마이그레이션을 간소화합니다. 이것은 [AWS 마이그레이션 전략](#)의 세 번째 단계입니다.

### 마이그레이션 팩토리

자동화되고 민첩한 접근 방식을 통해 워크로드 마이그레이션을 간소화하는 다기능 팀입니다. 마이그레이션 팩토리 팀에는 일반적으로 스프린트에서 작업하는 운영, 비즈니스 분석가 및 소유자, 마이그레이션 엔지니어, 개발자 및 DevOps 전문가가 포함됩니다. 엔터프라이즈 애플리케이션 포트폴리오의 20~50%는 공장 접근 방식으로 최적화할 수 있는 반복되는 패턴으로 구성되어 있습니다. 자세한 내용은 이 콘텐츠 세트의 [클라우드 마이그레이션 팩토리 가이드](#)와 [마이그레이션 팩토리에 대한 설명](#)을 참조하십시오.

### 마이그레이션 메타데이터

마이그레이션을 완료하는 데 필요한 애플리케이션 및 서버에 대한 정보 각 마이그레이션 패턴에는 서로 다른 마이그레이션 메타데이터 세트가 필요합니다. 마이그레이션 메타데이터의 예로는 대상 서브넷, 보안 그룹 및 AWS 계정이 있습니다.

### 마이그레이션 패턴

사용되는 마이그레이션 전략, 마이그레이션 대상, 마이그레이션 애플리케이션 또는 서비스를 자세히 설명하는 반복 가능한 마이그레이션 작업입니다. 예: AWS Application Migration Service를 EC2 사용하여 Amazon으로 마이그레이션을 다시 호스팅합니다.

### 마이그레이션 포트폴리오 평가(MPA)

로 마이그레이션하기 위한 비즈니스 사례를 검증하기 위한 정보를 제공하는 온라인 도구입니다 AWS 클라우드.는 자세한 포트폴리오 평가(서버 적정 규모, 요금, TCO 비교, 마이그레이션 비용 분석)와 마이그레이션 계획(애플리케이션 데이터 분석 및 데이터 수집, 애플리케이션 그룹화, 마이그레이션 우선 순위 지정 및 웨이브 계획)을 MPA 제공합니다. 이 [MPA 도구](#)(로그인 필요)는 모든 AWS 컨설턴트 및 APN 파트너 컨설턴트에게 무료로 제공됩니다.

### 마이그레이션 준비 평가(MRA)

를 사용하여 조직의 클라우드 준비 상태에 대한 인사이트를 얻고, 강점과 약점을 식별하고, 식별된 격차를 줄이기 위한 실행 계획을 수립하는 프로세스입니다 AWS CAF. 자세한 내용은 [마이그레이션 준비 가이드](#)를 참조하세요. MRA는 [AWS 마이그레이션 전략](#)의 첫 번째 단계입니다.

## 마이그레이션 전략

워크로드를 로 마이그레이션하는 데 사용되는 접근 방식입니다 AWS 클라우드. 자세한 내용은 이 용어집의 [7 Rs](#) 항목을 참조하고 [대규모 마이그레이션을 가속화하기 위해 조직 동원을 참조하세요.](#)

## ML

[기계 학습](#)을 참조하세요.

## 현대화

비용을 절감하고 효율성을 높이고 혁신을 활용하기 위해 구식(레거시 또는 모놀리식) 애플리케이션과 해당 인프라를 클라우드의 민첩하고 탄력적이고 가용성이 높은 시스템으로 전환하는 것입니다. 자세한 내용은 [의 애플리케이션 현대화 전략을 참조하세요 AWS 클라우드.](#)

## 현대화 준비 상태 평가

조직 애플리케이션의 현대화 준비 상태를 파악하고, 이점, 위험 및 종속성을 식별하고, 조직이 해당 애플리케이션의 향후 상태를 얼마나 잘 지원할 수 있는지를 확인하는 데 도움이 되는 평가입니다. 평가 결과는 대상 아키텍처의 청사진, 현대화 프로세스의 개발 단계와 마일스톤을 자세히 설명하는 로드맵 및 파악된 격차를 해소하기 위한 실행 계획입니다. 자세한 내용은 [의 애플리케이션에 대한 현대화 준비 상태 평가를 참조하세요 AWS 클라우드.](#)

## 모놀리식 애플리케이션(모놀리식 유형)

긴밀하게 연결된 프로세스를 사용하여 단일 서비스로 실행되는 애플리케이션입니다. 모놀리식 애플리케이션에는 몇 가지 단점이 있습니다. 한 애플리케이션 기능에 대한 수요가 급증하면 전체 아키텍처 규모를 조정해야 합니다. 코드 베이스가 커지면 모놀리식 애플리케이션의 기능을 추가하거나 개선하는 것도 더 복잡해집니다. 이러한 문제를 해결하기 위해 마이크로서비스 아키텍처를 사용할 수 있습니다. 자세한 내용은 [마이크로서비스로 모놀리식 유형 분해](#)를 참조하십시오.

## MPA

[마이그레이션 포트폴리오 평가](#)를 참조하세요.

## MQTT

[메시지 대기열 원격 측정 전송](#)을 참조하세요.

## 멀티클래스 분류

여러 클래스에 대한 예측(2개 이상의 결과 중 하나 예측)을 생성하는 데 도움이 되는 프로세스입니다. 예를 들어, ML 모델이 '이 제품은 책인가요, 자동차인가요, 휴대폰인가요?' 또는 '이 고객이 가장 관심을 갖는 제품 범주는 무엇인가요?'라고 물을 수 있습니다.

## 변경 가능한 인프라

프로덕션 워크로드에 대한 기존 인프라를 업데이트하고 수정하는 모델입니다. 일관성, 신뢰성 및 예측 가능성을 높이기 위해 AWS Well-Architected Framework는 [변경할 수 없는 인프라](#)를 모범 사례로 사용할 것을 권장합니다.

## O

### OAC

[오리진 액세스 제어를](#) 참조하세요.

### OAI

[오리진 액세스 자격 증명을](#) 참조하세요.

### OCM

[조직 변경 관리를](#) 참조하세요.

### 오프라인 마이그레이션

마이그레이션 프로세스 중 소스 워크로드가 중단되는 마이그레이션 방법입니다. 이 방법은 가동 중지 증가를 수반하며 일반적으로 작고 중요하지 않은 워크로드에 사용됩니다.

## OI

[작업 통합](#)을 참조하세요.

### OLA

[운영 수준 계약을](#) 참조하세요.

### 온라인 마이그레이션

소스 워크로드를 오프라인 상태로 전환하지 않고 대상 시스템에 복사하는 마이그레이션 방법입니다. 워크로드에 연결된 애플리케이션은 마이그레이션 중에도 계속 작동할 수 있습니다. 이 방법은 가동 중지 차단 또는 최소화를 수반하며 일반적으로 중요한 프로덕션 워크로드에 사용됩니다.

### OPC-UA

[Open Process Communications - Unified Architecture](#)를 참조하세요.

### Open Process Communications - 통합 아키텍처(OPC-UA)

산업 자동화를 위한 machine-to-machine (M2M) 통신 프로토콜입니다. OPC-UA는 데이터 암호화, 인증 및 권한 부여 체계와 상호 운용성 표준을 제공합니다.

## 운영 수준 계약(OLA)

서비스 수준 계약(SLA)을 지원하기 위해 기능 IT 그룹이 서로에게 제공할 것을 명확히 하는 계약입니다.

## 운영 준비 검토(ORR)

인시던트 및 가능한 장애의 범위를 이해, 평가, 방지 또는 줄이는 데 도움이 되는 질문 및 관련 모범 사례 체크리스트입니다. 자세한 내용은 AWS Well-Architected 프레임워크의 [운영 준비 검토\(ORR\)](#)를 참조하세요.

## 운영 기술(OT)

물리적 환경과 협력하여 산업 운영, 장비 및 인프라를 제어하는 하드웨어 및 소프트웨어 시스템입니다. 제조에서 OT 및 정보 기술(IT) 시스템의 통합은 [Industry 4.0](#) 혁신의 주요 초점입니다.

## 운영 통합(OI)

클라우드에서 운영을 현대화하는 프로세스로 준비 계획, 자동화 및 통합을 수반합니다. 자세한 내용은 [운영 통합 가이드](#)를 참조하십시오.

## 조직 트레일

에서 생성한 추적은 조직 AWS 계정 내 모든에 대한 모든 이벤트를 AWS CloudTrail 기록합니다 AWS Organizations. 이 트레일은 조직에 속한 각 AWS 계정에 생성되고 각 계정의 활동을 추적합니다. 자세한 내용은 설명서의 [조직에 대한 추적 생성](#)을 참조하세요 CloudTrail.

## 조직 변경 관리(OCM)

사람, 문화 및 리더십 관점에서 주요하고 파괴적인 비즈니스 혁신을 관리하기 위한 프레임워크입니다. OCM는 변화 채택을 가속화하고, 전환 문제를 해결하고, 문화 및 조직 변화를 주도하여 조직이 새로운 시스템 및 전략을 준비하고 전환할 수 있도록 지원합니다. AWS 마이그레이션 전략에서는 클라우드 채택 프로젝트에 필요한 변경 속도 때문에이 프레임워크를 인력 가속화라고 합니다. 자세한 내용은 [OCM 안내서](#)를 참조하세요.

## 오리진 액세스 제어(OAC)

에서는 Amazon Simple Storage Service(Amazon S3) 콘텐츠의 보안을 위해 액세스를 제한하는 CloudFront 향상된 옵션입니다. 모든 AWS 리전 서버 측 암호화 AWS KMS (-SSEKMS) 및 동적 PUT 및 S3 버킷에 대한 DELETE 요청의 모든 S3 버킷을 OAC 지원합니다.

## 오리진 액세스 자격 증명(OAI)

에서 Amazon S3 콘텐츠를 보호하기 위해 액세스를 제한하는 CloudFront 옵션입니다. OAI를 사용하면 Amazon S3가 인증할 수 있는 보안 주체가 CloudFront 생성됩니다. 인증된 보안 주체는 특정

CloudFront 배포를 통해서만 S3 버킷의 콘텐츠에 액세스할 수 있습니다. 보다 세분화되고 향상된 액세스 제어를 제공하는 도 참조[OAC](#)하세요.

## ORR

[운영 준비 상태 검토](#)를 참조하세요.

## OT

[운영 기술을](#) 참조하세요.

## 아웃바운드(송신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 내에서 시작된 네트워크 연결을 VPC 처리하는 입니다. [AWS 보안 참조 아키텍처](#)는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 VPCs 위해 인바운드, 아웃바운드 및 검사로 네트워크 계정을 설정하는 것이 좋습니다.

## P

### 권한 경계

IAM 보안 주체에 연결되어 사용자 또는 역할이 가질 수 있는 최대 권한을 설정하는 IAM 관리 정책입니다. 자세한 내용은 IAM 설명서의 [권한 경계](#)를 참조하세요.

### 개인 식별 정보(PII)

직접 보거나 다른 관련 데이터와 함께 짝을 지을 때 개인의 신원을 합리적으로 추론하는 데 사용할 수 있는 정보입니다. 의 예로는 이름, 주소 및 연락처 정보가 PII 있습니다.

## PII

[개인 식별 정보를](#) 참조하세요.

### 플레이북

클라우드에서 핵심 운영 기능을 제공하는 등 마이그레이션과 관련된 작업을 캡처하는 일련의 사전 정의된 단계입니다. 플레이북은 스크립트, 자동화된 런북 또는 현대화된 환경을 운영하는 데 필요한 프로세스나 단계 요약의 형태를 취할 수 있습니다.

## PLC

[프로그래밍 가능한 로직 컨트롤러](#)를 참조하세요.

## PLM

[제품 수명 주기 관리](#)를 참조하세요.

## 정책

권한을 정의하거나([자격 증명 기반 정책](#) 참조), 액세스 조건을 지정하거나([리소스 기반 정책](#) 참조), 조직의 모든 계정에 대한 최대 권한을 정의할 수 있는 객체 AWS Organizations 입니다([서비스 제어 정책](#) 참조).

## 다국어 지속성

데이터 액세스 패턴 및 기타 요구 사항을 기반으로 독립적으로 마이크로서비스의 데이터 스토리지 기술 선택. 마이크로서비스가 동일한 데이터 스토리지 기술을 사용하는 경우 구현 문제가 발생하거나 성능이 저하될 수 있습니다. 요구 사항에 가장 적합한 데이터 스토어를 사용하면 마이크로서비스를 더 쉽게 구현하고 성능과 확장성을 높일 수 있습니다. 자세한 내용은 [마이크로서비스에서 데이터 지속성 활성화](#)를 참조하십시오.

## 포트폴리오 평가

마이그레이션을 계획하기 위해 애플리케이션 포트폴리오를 검색 및 분석하고 우선순위를 정하는 프로세스입니다. 자세한 내용은 [마이그레이션 준비 상태 평가](#)를 참조하십시오.

## 조건자

WHERE 절에서 false 일반적으로 위치한 true 또는를 반환하는 쿼리 조건입니다.

## 조건자 푸시다운

전송 전에 쿼리의 데이터를 필터링하는 데이터베이스 쿼리 최적화 기법입니다. 이렇게 하면 관계형 데이터베이스에서 검색하고 처리해야 하는 데이터의 양이 줄어들고 쿼리 성능이 향상됩니다.

## 예방적 제어

이벤트 발생을 방지하도록 설계된 보안 제어입니다. 이 제어는 네트워크에 대한 무단 액세스나 원치 않는 변경을 방지하는 데 도움이 되는 1차 방어선입니다. 자세한 내용은 Implementing security controls on AWS의 [Preventative controls](#)를 참조하십시오.

## 보안 주체

작업을 수행하고 리소스에 액세스할 수 있는 AWS 있는의 개체입니다. 이 엔터티는 일반적으로 , AWS 계정 IAM 역할 또는 사용자의 루트 사용자입니다. 자세한 내용은 IAM 설명서의 역할의 보안 주체 용어 및 개념을 참조하세요. [https://docs.aws.amazon.com/IAM/latest/UserGuide/id\\_roles.html#id\\_roles\\_terms-and-concepts](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html#id_roles_terms-and-concepts)

## 프라이버시 중심 설계

전체 개발 프로세스를 통해 프라이버시를 고려하는 시스템 엔지니어링 접근 방식입니다.

## 프라이빗 호스팅 영역

Amazon Route 53가 하나 이상의 내에서 도메인 및 하위 도메인에 대한 DNS 쿼리에 응답하는 방법에 대한 정보를 포함하는 컨테이너입니다VPCs. 자세한 내용은 Route 53 설명서의 [프라이빗 호스팅 영역 작업](#)을 참조하십시오.

## 사전 예방적 제어

규정 미준수 리소스의 배포를 방지하도록 설계된 [보안 제어](#)입니다. 이러한 제어는 리소스가 프로비저닝되기 전에 스캔 리소스를 제어합니다. 리소스가 컨트롤을 준수하지 않으면 프로비저닝되지 않습니다. 자세한 내용은 AWS Control Tower 설명서의 [제어 참조 가이드](#)를 참조하고의 보안 [제어 구현의 사전](#) 예방적 제어를 참조하세요. AWS

## 제품 수명 주기 관리(PLM)

설계, 개발 및 출시부터 성장 및 성숙도, 거부 및 제거에 이르기까지 전체 수명 주기 동안 제품의 데이터 및 프로세스 관리.

## 프로덕션 환경

[환경](#)을 참조하세요.

## 프로그래밍 가능한 로직 컨트롤러(PLC)

제조 분야에서는 기계를 모니터링하고 제조 프로세스를 자동화하는 매우 안정적이고 적응력이 뛰어난 컴퓨터입니다.

## 프롬프트 체인

한 [LLM](#) 프롬프트의 출력을 다음 프롬프트의 입력으로 사용하여 더 나은 응답을 생성합니다. 이 기법은 복잡한 작업을 하위 작업으로 나누거나 예비 응답을 반복적으로 구체화하거나 확장하는 데 사용됩니다. 이는 모델 응답의 정확성과 관련성을 개선하는 데 도움이 되며 보다 세분화되고 개인화된 결과를 제공합니다.

## 가명화

데이터세트의 개인 식별자를 자리 표시자 값으로 바꾸는 프로세스입니다. 가명화는 개인 정보를 보호하는 데 도움이 될 수 있습니다. 가명화된 데이터는 여전히 개인 데이터로 간주됩니다.

## publish/subscribe (pub/sub)

마이크로서비스 간의 비동기 통신을 지원하여 확장성과 응답성을 개선하는 패턴입니다. 예를 들어 마이크로서비스 기반에서 [MES](#)마이크로서비스는 다른 마이크로서비스가 구독할 수 있는 채널에 이벤트 메시지를 게시할 수 있습니다. 시스템은 게시 서비스를 변경하지 않고도 새 마이크로서비스를 추가할 수 있습니다.

## Q

### 쿼리 계획

지침처럼 SQL 관계형 데이터베이스 시스템의 데이터에 액세스하는 데 사용되는 일련의 단계입니다.

### 쿼리 계획 회귀

데이터베이스 서비스 최적화 프로그램이 데이터베이스 환경을 변경하기 전보다 덜 최적의 계획을 선택하는 경우입니다. 통계, 제한 사항, 환경 설정, 쿼리 파라미터 바인딩 및 데이터베이스 엔진 업데이트의 변경으로 인해 발생할 수 있습니다.

## R

### RACI 매트릭스

[책임, 책임, 상담, 정보 제공\(RACI\)을 참조하세요.](#)

### RAG

[Augmented Generation 검색을 참조하세요.](#)

### 랜섬웨어

결제가 완료될 때까지 컴퓨터 시스템이나 데이터에 대한 액세스를 차단하도록 설계된 악성 소프트웨어입니다.

### RASCI 매트릭스

[책임, 책임, 상담, 정보 제공\(RACI\)을 참조하세요.](#)

### RCAC

[행 및 열 액세스 제어를 참조하세요.](#)

### 읽기 전용 복제본

읽기 전용 용도로 사용되는 데이터베이스의 사본입니다. 쿼리를 읽기 전용 복제본으로 라우팅하여 기본 데이터베이스의 로드를 줄일 수 있습니다.

### 재설계

[7R을 참조하세요.](#)

## 복구 시점 목표(RPO)

마지막 데이터 복구 시점 이후 허용되는 최대 시간입니다. 이에 따라 마지막 복구 시점과 서비스 중단 사이에 허용되는 데이터 손실로 간주되는 범위가 결정됩니다.

## 복구 시간 목표(RTO)

서비스 중단과 서비스 복원 사이의 허용 가능한 지연 시간입니다.

## 리팩터링

[7 Rs](#)를 참조하세요.

## 리전

지리적 영역의 AWS 리소스 모음입니다. 각 AWS 리전은 내결함성, 안정성 및 복원력을 제공하기 위해 서로 격리되고 독립적입니다. 자세한 내용은 [계정에서 사용할 수 있는 항목 지정을 참조 AWS 리전 하세요.](#)

## 회귀

숫자 값을 예측하는 ML 기법입니다. 예를 들어, '이 집은 얼마에 팔릴까?'라는 문제를 풀기 위해 ML 모델은 선형 회귀 모델을 사용하여 주택에 대해 알려진 사실(예: 면적)을 기반으로 주택의 매매 가격을 예측할 수 있습니다.

## 리호스팅

[7 Rs](#)를 참조하세요.

## release

배포 프로세스에서 변경 사항을 프로덕션 환경으로 승격시키는 행위입니다.

## 재배치

[7 Rs](#)를 참조하세요.

## 리플랫폼

[7 Rs](#)를 참조하세요.

## 재구매

[7R](#)을 참조하세요.

## 복원력

중단에 저항하거나 복구할 수 있는 애플리케이션의 기능입니다. 에서 복원력을 계획할 때 [고가용성](#) 및 [재해 복구](#)가 일반적인 고려 사항입니다 AWS 클라우드. 자세한 내용은 [AWS 클라우드 복원력을 참조하세요.](#)

## 리소스 기반 정책

Amazon S3 버킷, 엔드포인트, 암호화 키 등의 리소스에 연결된 정책입니다. 이 유형의 정책은 액세스가 허용된 보안 주체, 지원되는 작업 및 충족해야 하는 기타 조건을 지정합니다.

## 책임, 책임, 상담, 정보 제공(RACI) 매트릭스

마이그레이션 활동 및 클라우드 운영에 참여하는 모든 당사자의 역할과 책임을 정의하는 매트릭스입니다. 매트릭스 이름은 매트릭스에 정의된 책임 유형에서 파생됩니다. 실무 담당자 (R), 의사 결정권자 (A), 업무 수행 조연자 (C), 결과 통보 대상자 (I). 지원자는 (S) 선택사항입니다. 지원을 포함하는 경우 매트릭스를 RASCI 매트릭스라고 하고, 매트릭스를 제외하면 RACI 매트릭스라고 합니다.

## 대응 제어

보안 기준에서 벗어나거나 부정적인 이벤트를 해결하도록 설계된 보안 제어입니다. 자세한 내용은 [Implementing security controls on AWS의 Responsive controls](#)를 참조하십시오.

## retain

[7R을 참조하세요.](#)

## 사용 중지

[7R을 참조하세요.](#)

## Augmented Generation 검색(RAG)

응답을 생성하기 전제가 훈련 데이터 소스 외부에 있는 신뢰할 수 있는 데이터 소스를 [LLM](#) 참조하는 [생성형 AI](#) 기술입니다. 예를 들어 모델은 조직의 지식 기반 또는 사용자 지정 데이터에 대한 의미 검색을 수행할 RAG 수 있습니다. 자세한 내용은 [RAG란 무엇인가요?](#)를 참조하세요.

## 교체

공격자가 보안 인증 정보에 액세스하는 것을 더 어렵게 만들기 위해 [보안 암호](#)를 주기적으로 업데이트하는 프로세스입니다.

## 행 및 열 액세스 제어(RCAC)

액세스 규칙을 정의한 기본적이고 유연한 SQL 표현식 사용.는 행 권한과 열 마스크로 RCAC 구성됩니다.

## RPO

[복구 시점 목표를 참조하세요.](#)

## RTO

[복구 시간 목표를](#) 참조하세요.

## 런북

특정 작업을 수행하는 데 필요한 일련의 수동 또는 자동 절차입니다. 일반적으로 오류율이 높은 반복 작업이나 절차를 간소화하기 위해 런북을 만듭니다.

## S

### SAML 2.0

많은 자격 증명 공급자(IdPs)가 사용하는 개방형 표준입니다. 이 기능을 사용하면 페더레이션 Single Sign-On(SSO)을 사용할 수 있으므로 사용자는 조직 내 모든 사용자에게 AWS API IAM 대해서 사용자를 생성할 필요 없이 AWS Management Console 에 로그인하거나 작업을 호출할 수 있습니다. SAML 2.0 기반 페더레이션에 대한 자세한 내용은 IAM 설명서의 [SAML 2.0 기반 페더레이션 정보를](#) 참조하세요.

### SCADA

[감독 제어 및 데이터 수집](#)을 참조하세요.

### SCP

[서비스 제어 정책을](#) 참조하세요.

### secret

에는 암호 또는 사용자 자격 증명과 같이 암호화된 형식으로 저장하는 AWS Secrets Manager 기밀 또는 제한된 정보가 있습니다. 보안 암호 값과 메타데이터로 구성됩니다. 보안 암호 값은 바이너리, 단일 문자열 또는 여러 문자열일 수 있습니다. 자세한 내용은 [Secrets Manager 설명서의 Secrets Manager 보안 암호의 내용을](#) 참조하세요.

### 설계별 보안

전체 개발 프로세스를 통해 보안을 고려하는 시스템 엔지니어링 접근 방식입니다.

### 보안 제어

위험 행위자가 보안 취약성을 악용하는 능력을 방지, 탐지 또는 감소시키는 기술적 또는 관리적 가드레일입니다. 보안 제어에는 [예방](#), [탐지](#), [대응](#) 및 [사전](#) 예방의 네 가지 기본 유형이 있습니다.

## 보안 강화

공격 표면을 줄여 공격에 대한 저항력을 높이는 프로세스입니다. 더 이상 필요하지 않은 리소스 제거, 최소 권한 부여의 보안 모범 사례 구현, 구성 파일의 불필요한 기능 비활성화 등의 작업이 여기에 포함될 수 있습니다.

### 보안 정보 및 이벤트 관리(SIEM) 시스템

보안 정보 관리(SIM) 및 보안 이벤트 관리(SEM) 시스템을 결합하는 도구 및 서비스입니다. SIEM 시스템은 서버, 네트워크, 디바이스 및 기타 소스에서 데이터를 수집, 모니터링 및 분석하여 위협 및 보안 위반을 감지하고 알림을 생성합니다.

### 보안 응답 자동화

보안 이벤트에 자동으로 응답하거나 이를 해결하도록 설계된 사전 정의되고 프로그래밍된 작업입니다. 이러한 자동화는 보안 모범 사례를 구현하는 데 도움이 되는 [탐지적](#) 또는 [대응적](#) AWS 보안 제어 역할을 합니다. 자동 응답 작업의 예로는 VPC 보안 그룹 수정, Amazon EC2 인스턴스 패치 적용 또는 자격 증명 교체 등이 있습니다.

### 서버 측 암호화

데이터를 AWS 서비스 수신하는가 대상에서 데이터를 암호화합니다.

### 서비스 제어 정책(SCP)

조직의 모든 계정에 대한 권한을 중앙 집중식으로 제어하는 정책입니다 AWS Organizations. 가드 레일을 SCPs 정의하거나 관리자가 사용자 또는 역할에 위임할 수 있는 작업에 대한 제한을 설정합니다. 허용 목록 또는 거부 목록 SCPs으로 사용하여 허용되거나 금지된 서비스 또는 작업을 지정할 수 있습니다. 자세한 내용은 AWS Organizations 설명서의 [서비스 제어 정책을](#) 참조하세요.

### 서비스 엔드포인트

에 대한 진입점 URL의 입니다 AWS 서비스. 엔드포인트를 사용하여 대상 서비스에 프로그래밍 방식으로 연결할 수 있습니다. 자세한 내용은 AWS 일반 참조의 [AWS 서비스 엔드포인트](#)를 참조하십시오.

### 서비스 수준 계약(SLA)

IT 팀이 고객에게 제공하기로 약속한 내용(예: 서비스 가동 시간 및 성능)을 명시한 계약입니다.

### 서비스 수준 표시기(SLI)

오류율, 가용성 또는 처리량과 같은 서비스의 성능 측면 측정.

### 서비스 수준 목표(SLO)

서비스 [수준 지표](#)로 측정되는 서비스의 상태를 나타내는 대상 지표입니다.

## 공동 책임 모델

클라우드 보안 및 규정 준수에 AWS 대해 사용자가 공유하는 책임을 설명하는 모델입니다. AWS 는 클라우드의 보안을 책임지고, 사용자는 클라우드의 보안을 책임집니다. 자세한 내용은 [공동 책임 모델](#)을 참조하십시오.

## SIEM

[보안 정보 및 이벤트 관리 시스템을](#) 참조하세요.

## 단일 장애 지점(SPOF)

시스템을 중단시킬 수 있는 애플리케이션의 중요한 단일 구성 요소의 장애입니다.

## SLA

[서비스 수준 계약을](#) 참조하세요.

## SLI

[서비스 수준 표시기를](#) 참조하세요.

## SLO

[서비스 수준 목표를](#) 참조하세요.

## split-and-seed 모델

현대화 프로젝트를 확장하고 가속화하기 위한 패턴입니다. 새로운 기능과 제품 릴리스가 정의되면 핵심 팀이 분할되어 새로운 제품 팀이 만들어집니다. 이를 통해 조직의 역량과 서비스 규모를 조정하고, 개발자 생산성을 개선하고, 신속한 혁신을 지원할 수 있습니다. 자세한 내용은 [에서 애플리케이션 현대화에 대한 단계별 접근 방식을 참조하세요 AWS 클라우드](#).

## SPOF

[단일 장애 지점을](#) 참조하세요.

## 스타 스키마

하나의 큰 팩트 테이블을 사용하여 트랜잭션 또는 측정된 데이터를 저장하고 하나 이상의 작은 차원 테이블을 사용하여 데이터 속성을 저장하는 데이터베이스 조직 구조입니다. 이 구조는 [데이터 웨어하우스](#) 또는 비즈니스 인텔리전스용으로 설계되었습니다.

## Strangler Fig 패턴

레거시 시스템을 폐기할 수 있을 때까지 시스템 기능을 점진적으로 다시 작성하고 교체하여 모놀리식 시스템을 현대화하기 위한 접근 방식. 이 패턴은 무화과 덩굴이 나무로 자라 결국 숙주를 압도하고 대체하는 것과 비슷합니다. [Martin Fowler](#)가 모놀리식 시스템을 다시 작성할 때 위험을 관리

하는 방법으로 이 패턴을 도입했습니다. 이 패턴을 적용하는 방법의 예는 [컨테이너 및 Amazon API Gateway를 사용하여 레거시 Microsoft ASP.NET \(ASMX\) 웹 서비스 증분 현대화를 참조하세요.](#)

## 서브넷

의 IP 주소 범위입니다VPC. 서브넷은 단일 가용 영역에 상주해야 합니다.

## 감독 제어 및 데이터 수집(SCADA)

제조에서 하드웨어와 소프트웨어를 사용하여 물리적 자산과 프로덕션 작업을 모니터링하는 시스템입니다.

## 대칭 암호화

동일한 키를 사용하여 데이터를 암호화하고 복호화하는 암호화 알고리즘입니다.

## 합성 테스트

잠재적 문제를 감지하거나 성능을 모니터링하기 위해 사용자 상호 작용을 시뮬레이션하는 방식으로 시스템을 테스트합니다. [Amazon CloudWatch Synthetics](#)를 사용하여 이러한 테스트를 생성할 수 있습니다.

## 시스템 프롬프트

동작을 지시하기 위해 [LLM](#)에 컨텍스트, 지침 또는 지침을 제공하는 기법입니다. 시스템 프롬프트는 컨텍스트를 설정하고 사용자와의 상호 작용을 위한 규칙을 설정하는 데 도움이 됩니다.

# T

## tags

AWS 리소스를 구성하기 위한 메타데이터 역할을 하는 키-값 페어입니다. 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색 및 필터링할 수 있습니다. 자세한 내용은 [AWS 리소스에 태그 지정](#)을 참조하십시오.

## 대상 변수

지도 ML에서 예측하려는 값으로, 결과 변수라고도 합니다. 예를 들어, 제조 설정에서 대상 변수는 제품 결함일 수 있습니다.

## 작업 목록

런북을 통해 진행 상황을 추적하는 데 사용되는 도구입니다. 작업 목록에는 런북의 개요와 완료해야 할 일반 작업 목록이 포함되어 있습니다. 각 일반 작업에 대한 예상 소요 시간, 소유자 및 진행 상황이 작업 목록에 포함됩니다.

## 테스트 환경

[환경을](#) 참조하세요.

### 훈련

ML 모델이 학습할 수 있는 데이터를 제공하는 것입니다. 훈련 데이터에는 정답이 포함되어야 합니다. 학습 알고리즘은 훈련 데이터에서 대상(예측하려는 답)에 입력 데이터 속성을 매핑하는 패턴을 찾고, 이러한 패턴을 캡처하는 ML 모델을 출력합니다. 그런 다음 ML 모델을 사용하여 대상을 모르는 새 데이터에 대한 예측을 할 수 있습니다.

### 전송 게이트웨이

VPCs 및 온프레미스 네트워크를 상호 연결하는 데 사용할 수 있는 네트워크 전송 허브입니다. 자세한 내용은 AWS Transit Gateway 설명서의 [전송 게이트웨이란 무엇입니까?](#)를 참조하세요.

### 트렁크 기반 워크플로

개발자가 기능 브랜치에서 로컬로 기능을 구축하고 테스트한 다음 해당 변경 사항을 기본 브랜치에 병합하는 접근 방식입니다. 이후 기본 브랜치는 개발, 프로덕션 이전 및 프로덕션 환경에 순차적으로 구축됩니다.

### 신뢰할 수 있는 액세스

사용자를 대신하여 AWS Organizations 및 해당 계정에서 조직에서 작업을 수행하도록 지정한 서비스에 대한 권한 부여. 신뢰할 수 있는 서비스는 필요할 때 각 계정에 서비스 연결 역할을 생성하여 관리 작업을 수행합니다. 자세한 내용은 설명서의 [다른 AWS 서비스와 AWS Organizations 함께 사용을](#) 참조하세요 AWS Organizations .

### 튜닝

ML 모델의 정확도를 높이기 위해 훈련 프로세스의 측면을 여러 변경하는 것입니다. 예를 들어, 레이블링 세트를 생성하고 레이블을 추가한 다음 다양한 설정에서 이러한 단계를 여러 번 반복하여 모델을 최적화하는 방식으로 ML 모델을 훈련할 수 있습니다.

### 피자 두 판 팀

두 개의 피자로 먹을 수 있는 작은 DevOps 팀입니다. 피자 두 판 팀 규모는 소프트웨어 개발에 있어 가능한 최상의 공동 작업 기회를 보장합니다.

## U

### 불확실성

예측 ML 모델의 신뢰성을 저해할 수 있는 부정확하거나 불완전하거나 알려지지 않은 정보를 나타내는 개념입니다. 불확실성에는 두 가지 유형이 있습니다. 인식론적 불확실성은 제한적이고 불완전한 데이터에 의해 발생하는 반면, 우연한 불확실성은 데이터에 내재된 노이즈와 무작위성에 의해 발생합니다. 자세한 내용은 [Quantifying uncertainty in deep learning systems](#) 가이드를 참조하십시오.

### 차별화되지 않은 작업

애플리케이션을 만들고 운영하는 데 필요하지만 최종 사용자에게 직접적인 가치를 제공하거나 경쟁 우위를 제공하지 못하는 작업을 헤비 리프팅이라고도 합니다. 차별화되지 않은 작업의 예로는 조달, 유지보수, 용량 계획 등이 있습니다.

### 상위 환경

[환경](#)을 참조하세요.

## V

### 정리

스토리지를 회수하고 성능을 향상시키기 위해 증분 업데이트 후 정리 작업을 수행하는 데이터베이스 유지 관리 작업입니다.

### 버전 제어

리포지토리의 소스 코드 변경과 같은 변경 사항을 추적하는 프로세스 및 도구입니다.

### VPC 피어링

프라이빗 IP 주소를 사용하여 트래픽을 라우팅할 수 VPCs 있는 두 개의 연결입니다. 자세한 내용은 Amazon VPC 설명서의 [VPC 피어링이란 무엇입니까?](#)를 참조하세요.

### 취약성

시스템 보안을 손상시키는 소프트웨어 또는 하드웨어 결함입니다.

# W

## 웜 캐시

자주 액세스하는 최신 관련 데이터를 포함하는 버퍼 캐시입니다. 버퍼 캐시에서 데이터베이스 인스턴스를 읽을 수 있기 때문에 주 메모리나 디스크에서 읽는 것보다 빠릅니다.

## 웜 데이터

자주 액세스하지 않는 데이터입니다. 이런 종류의 데이터를 쿼리할 때는 일반적으로 적절히 느린 쿼리가 허용됩니다.

## 창 함수

현재 레코드와 어떤 식으로든 관련된 행 그룹에 대해 계산을 수행하는 SQL 함수입니다. 창 함수는 이동 평균을 계산하거나 현재 행의 상대 위치를 기반으로 행 값에 액세스하는 등의 작업을 처리하는 데 유용합니다.

## 워크로드

고객 대면 애플리케이션이나 백엔드 프로세스 같이 비즈니스 가치를 창출하는 리소스 및 코드 모음입니다.

## 워크스트림

마이그레이션 프로젝트에서 특정 작업 세트를 담당하는 직무 그룹입니다. 각 워크스트림은 독립적이지만 프로젝트의 다른 워크스트림을 지원합니다. 예를 들어, 포트폴리오 워크스트림은 애플리케이션 우선순위 지정, 웨이브 계획, 마이그레이션 메타데이터 수집을 담당합니다. 포트폴리오 워크스트림은 이러한 자산을 마이그레이션 워크스트림에 전달하고, 마이그레이션 워크스트림은 서버와 애플리케이션을 마이그레이션합니다.

## WORM

[쓰기를 한 번 보고 많이 읽습니다.](#)

## WQF

[AWS 워크로드 검증 프레임워크](#)를 참조하세요.

## 한 번 쓰기, 많이 읽기(WORM)

데이터를 한 번에 쓰고 데이터가 삭제되거나 수정되지 않도록 하는 스토리지 모델입니다. 권한 있는 사용자는 필요한 만큼 데이터를 읽을 수 있지만 변경할 수는 없습니다. 이 데이터 스토리지 인프라는 [변경할 수 없는](#) 것으로 간주됩니다.

## Z

### 제로데이 익스플로잇

[제로데이 취약성](#)을 활용하는 공격, 일반적으로 맬웨어입니다.

### 제로데이 취약성

프로덕션 시스템의 명백한 결함 또는 취약성입니다. 위협 행위자는 이러한 유형의 취약성을 사용하여 시스템을 공격할 수 있습니다. 개발자는 공격의 결과로 취약성을 인지하는 경우가 많습니다.

### 제로샷 프롬프트

에 작업 수행에 대한 [LLM](#) 지침을 제공하지만, 작업 가이드에 도움이 될 수 있는 예제(샷)는 제공하지 않습니다. 는 사전 훈련된 지식을 사용하여 작업을 처리해야 LLM 합니다. 제로샷 프롬프트의 효과는 작업의 복잡성과 프롬프트의 품질에 따라 달라집니다. [또한 몇 번의 샷 프롬프트를 참조하세요.](#)

### 좀비 애플리케이션

평균 CPU 및 메모리 사용량이 5% 미만인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하는 것이 일반적입니다.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.