Optimizing PostgreSQL query performance

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Optimizing PostgreSQL query performance

Table of Contents

Optimizing PostgreSQL query performance

Amazon Web Services (contributors)

April 2024 (document history)

PostgreSQL is an open source object-relational database system that is powerful, flexible, and reliable. There are many ways to optimize the performance of a PostgreSQL query. The process of optimizing the query depends on the use case. Knowing the current query plan can help you to identify and understand any issues and make the necessary changes. Sometimes, you might need to analyze the tables to keep the database statistics up to date. The PostgreSQL optimizer will use those statistics to run the query faster. This guide focuses on best practices for improving the performance of PostgreSQL queries.

This guide assumes that you have an existing Amazon Relational Database Service (Amazon RDS) for PostgreSQL or Amazon Aurora PostgreSQL-Compatible database instance.

Use cases for query performance tuning

This guide covers five use cases, with explanations and examples:

- Collations
- Data type mismatch
- Function call in the `SELECT` statement
- `IN` or `EXISTS`
- Subqueries or Common Table Expressions (CTEs)

Each use case provides details of the initial run plan, how to analyze the plan to identify the problem, and a solution. Implementing these use cases typically results in faster response times for queries, reduced load on the server, and overall enhanced system efficiency. Those improvements can lead to a better user experience and increased system reliability.

The EXPLAIN query plan

PostgreSQL provides the `EXPLAIN` and `EXPLAIN ANALYZE` options for returning query plans with details about how the query will be run.

The EXPLAIN statement

The `EXPLAIN` statement returns the query plan that the PostgreSQL planner generates for a given statement. The query plan shows the following:

- How tables involved in a statement will be scanned (for example, by index scan or sequential scan)
- How multiple tables will be joined (for example, hash join, merge join, or nested loop join)

Understanding the plan is critical when improving the performance of the query. After you understand the plan, you can focus on where the query is taking too long and take action to reduce the time.

Using EXPLAIN ANALYZE

In PostgreSQL, `EXPLAIN` will only generate a plan for the given statement. If you add the `ANALYZE` keyword, `EXPLAIN` will return the plan, run the query, and show the actual runtime and row count for each step. This is indispensable for analyzing the query performance.

> ⚠ **Important**
>
> When using `EXPLAIN ANALYZE`, be careful with `INSERT`, `UPDATE`, and `DELETE`.

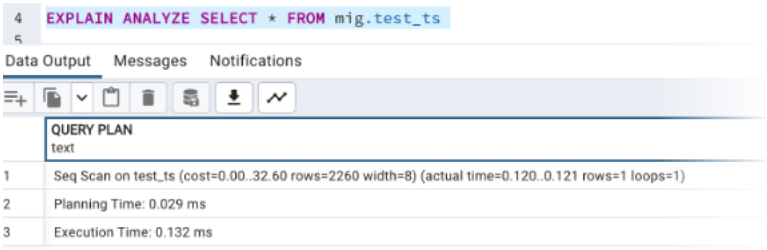
How to read the EXPLAIN query plan

A PostgreSQL query plan is a tree structure consisting of several nodes. The `EXPLAIN` query plan shows the steps that the database engine uses to run a query. The query plan provides the following information:

- The type of operations performed, such as sequential scans, index scans, or nested loop joins.

- A label, such as `Seq Scan`, `Index Scan`, or `Nested Loop`, to describe the operation being performed.
- The name of the table or index being processed by the query.
- Cost and row columns with information about the estimated cost in an arbitrary unit of computation and the number of rows processed.
- The filter condition of any filter applied on the operation, such as the `where` condition.
- A visual representation of the steps, with each operation shown as a node and arrows connecting the operations. The order of the operations is shown from left to right, with earlier operations feeding into later operations.

The following screenshot shows the query plan for a sequential scan.



The cost estimate (`cost=0.00..32.60 rows=2260 width=8`) means that PostgreSQL expects that the query will require 32.60 units of computation to return results.

The `0.00` value is the cost at which this node can begin working (in this case, startup time for the query). The `rows` value is the estimated number of rows that the sequential scan will return. The `width` value is the estimated size in bytes of the returned rows.

Because the example shows `EXPLAIN` with the `ANALYZE` option, the query was run, and the timing information was captured. The result (`actual time=0.120..0.121 rows=1 loops=1`) means the following:

- The sequential scan was run one time (the `loops` value).
- The scan returned one row.
- The actual time was 0.12 milliseconds.

Use cases for tuning queries

This guide covers the following use cases for tuning query performance:

- Collations
- Data type mismatch
- Function call in the SELECT statement
- IN or EXISTS
- Subqueries or Common Table Expressions (CTEs)

To test performance tuning for these query-performance use cases, use your existing database and the example data provided by this guide. The example uses data for a fictitious XX airline. To prepare the example data, run the following example code:

```
--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(1,10000000),'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",
    progname character varying(150) COLLATE pg_catalog."default",
    CONSTRAINT serviceprogram_pkey PRIMARY KEY (serial)
);

CREATE INDEX IF NOT EXISTS progname_id_ind
    ON perf_user.serviceprogram USING btree
    (progname 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 vehicleservicehistory_pkey PRIMARY KEY (v_id)
);

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

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

CREATE INDEX IF NOT EXISTS vehicleservicehistory_v_id_ind
    ON perf_user.vehicleservicehistory 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)
);
```

Use case 1 – Collations

In a database, a collation is a set of rules for determining how data is sorted and compared. A collation is usually applied to how text data is sorted in different languages for indexing for making comparisons between text values. Different languages have different character sets and ordering. With a collation, you can sort character data for a given language by using rules that define the correct character sequence. You can also specify the following:

- Case-sensitivity
- Accent marks
- Kana character types
- Use of symbols or punctuation
- Character width
- Word sorting

There might be a performance impact if the join column uses a different collation. The following example query uses three tables, with a different collation for the join column.

Table name	Column name
`rnr_segment`	`pnr_number character varying(15) COLLATE pg_catalog."C" NOT NULL`
`rnr_segment_pax`	`pnr_number character varying(15) COLLATE pg_catalog."default" NOT NULL`

rnr_seat_numbers	pnr_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');
```

The query plan for the previous query uses a sequence scan on the `rnr_seat_numbers` table even though that table has a proper index on the joined columns. The planner isn't using an index scan because these joined columns are using different collations:

```
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
```

To change the table column collation from the "C" language to the default collation provided by PostgreSQL, run the following `alter` statement, and then analyze the table:

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

Analyze rnr_segment;
```

The query plan now uses an index scan, and the runtime is reduced.

```
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
```

Use case 2 – Data type mismatch

Choosing the proper data type based on the data helps to provide the optimum balance between storage size and performance.

The following example query uses the `pnr_number` column to join two tables. The `pnr_number` column has different data types in different tables.

Table name	Column name and data type
`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
```

When you run `EXPLAIN ANALYZE`, the planner uses a sequence scan on `rnr_segment_pax` instead of an index scan even though the columns used in the join have indexes. The planner isn't using an index scan because the columns used in the join are different lengths.

Alter the table columns to keep the data type the same for both tables involved in the join condition, and then analyze the table:

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

analyze perf_user.rnr_expiry_date;
```

Now the tables have the same length on both of the columns that are used in the join condition.

Run EXPLAIN ANALYZE again. The planner performs an index scan, which improves the query performance significantly.

```
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
```

Use case 3 – Function call in the SELECT statement

Calling a function in a where clause can reduce query performance when the function is VOLATILE and you don't use the select keyword while calling the function:

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

An index scan runs if the select statement is used while calling the function:

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

The pnr_number field has an index in the rnr_expiry_date table. The index is used when comparing the value in the where clause.

```
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"
```

A sequential scan is performed when a function is called without the `select` keyword even when an index is available on the field.

```
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"
```

An index scan is performed when the function is called with the `select` keyword.

```
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"
```

Use case 4 – IN or EXISTS

If the query has `IN` or `NOT IN` operators, we recommend checking the query plan to confirm that the proper index is being used. If the proper index isn't being used and query performance is taking more time than expected, try to rewrite the query using the `EXISTS` or `NOT EXISTS` conditions.

Consider the following example, which uses `NOT IN`:

```
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"
```

The query is taking more than 37 seconds 366 milliseconds to retrieve 4 million records.

The query plan states that a sequence scan is performed on the table used in the subquery `vehicleservicehistory`. The sequence scan is producing a large number of records. For each of those records in the subquery, the query is performing a full table scan, which is causing the performance issue.

To avoid the sequence scan on the subquery, rewrite the subquery to use a correlated subquery with `NOT EXISTS`. The correlated subquery will use an index scan and a reduced number of table scans:

```
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"
```

After the modification, the query is taking less than 13 ms to process 4 million records

According to the query plan of the modified query, the table `vehicleservicehistory` can have an index scan. Using an index scan reduces the cost and the number of affected rows. This way, you can reduce the runtime of a query and increase its performance.

Use case 5 – Subqueries or CTEs

Common Table Expressions (CTEs) help break down large queries into smaller queries. This makes the whole query easier to maintain.

Subquery joins are replaced by CTE joins, which are more readable because the query is named and separated inside the CTE section. This is especially helpful when the size of the query grows and the query becomes harder to maintain. In addition, the CTE results in PostgreSQL are materialized. If you call the CTE in multiple places, the actual query definition will be run only one time. The result will be stored in memory. You can use this for any complex logic that must be used in

multiple places in the same query. Put that logic inside a CTE, and call the CTE any number of times.

For example, a customer was using inline application queries with many subqueries within queries. The subqueries were filtered by input parameter values sent from the applications.

```
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"
```

After modifying the subqueries by using a CTE and adding filters so that only required row sets are retrieved, the query performance improves.

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

```
"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"
```

These are the observations from the example data. When you run the query on a huge dataset, the difference in performance will be very high.

FAQ

Find answers to frequently raised questions about tuning query performance.

What is EXPLAIN?

`EXPLAIN` is a keyword that you prepend to a PostgreSQL query (`SELECT`, `UPDATE`, `INSERT`, `DELETE`) to generate a query plan. The PostgreSQL query plan details how the database intends to run the query. This plan includes information about the order of a table scan, index usage, and joins.

Use the query plan to identify potential bottlenecks, optimize queries, and improve overall performance. When reviewing the query plan, consider the following factors:

- Table access approaches
- Join approaches
- Filter conditions
- Sort operations
- Index usage
- Parallelism
- Statistics
- Cost estimations
- Rows retrieved from each step
- Data distribution

For more information about EXPLAIN, see the PostgreSQL documentation.

What is EXPLAIN ANALYZE?

When you prepend `EXPLAIN ANALYZE` to a query and run the query, PostgreSQL runs the query and returns both the query plan and runtime statistics. The actual runtime, rows processed from each step, and other relevant information are displayed along with the query plan. Using `EXPLAIN ANALYZE` on a production database should be done with caution, because running the query could impact database performance during the analysis.

For more information about EXPLAIN ANALYZE, see the PostgreSQL documentation.

What is collation in PostgreSQL?

In PostgreSQL, a collation is a set of rules for determining how strings are compared and sorted. The collation defines the order in which characters are considered in comparisons, considering language-specific rules and conversions.

For more information about collation, see the PostgreSQL documentation.

What is a CTE?

In a PostgreSQL database, a Common Table Expression (CTE) is a named temporary result set that you can reference. CTEs provide a way to create more readable and modular SQL queries by breaking down complex logic into smaller, named units.

For more information about CTEs, see the PostgreSQL documentation.

What are the categories of functions in PostgreSQL?

Every PostgreSQL function has a volatility classification, with the possibilities being `VOLATILE`, `STABLE`, or `IMMUTABLE`:

- **VOLATILE** – A `VOLATILE` function can do anything, including modify the database. It can return different results on successive calls with the same arguments. The optimizer makes no assumptions about the behavior of such functions. A query using a volatile function will re-evaluate the function at every row where its value is needed.
- **STABLE** – A `STABLE` function can't modify the database. It's guaranteed to return the same results given the same arguments for all rows within a single statement. When you use this classification, the optimizer can optimize multiple calls of the function to a single call. In particular, it's safe to use an expression that contains such a function in an index scan condition. (Because an index scan will evaluate the comparison value only one time, not one time at each row, it isn't valid to use a `VOLATILE` function in an index scan condition.)
- **IMMUTABLE** – An `IMMUTABLE` function can't modify the database and is guaranteed to return the same results given the same arguments forever. When you use this classification, the optimizer can pre-evaluate the function when a query calls it with constant arguments. For example, a query such as `SELECT ... WHERE x = 2 + 2` can be simplified on sight to

`SELECT ... WHERE x = 4`, because the function underlying the integer addition operator is marked `IMMUTABLE`.

`VOLATILE` is the default if the `CREATE FUNCTION` command doesn't specify a category. For more information about function types, see the PostgreSQL documentation.

Resources

References

- EXPLAIN
- Using EXPLAIN
- Collation Support
- WITH Queries (Common Table Expressions)

Guides

- Maintenance activities for PostgreSQL databases in Amazon RDS and Amazon Aurora to avoid performance issues
- Tuning PostgreSQL parameters in Amazon RDS and Amazon Aurora

Contributors

Contributors to this document include:

- Tirumala Dasari, Lead Consultant – Databases, AWS
- Veeranjaneyulu Grandhi, Lead Consultant – Databases, AWS
- Vamsikrishna Jammula, Consultant – Databases, AWS
- Srinivas Potlachervoo, Senior Lead Consultant – Databases, AWS
- Naga Srinivas Reddy Ravulapati, Consultant – Databases, AWS

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication	—	April 23, 2024

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than active-passive migration.

active-passive migration

A database migration method in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include `SUM` and `MAX`.

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see What is Artificial Intelligence?

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see ABAC for AWS in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A bot that is intended to disrupt or cause harm to individuals or organizations.

BCP

See business continuity planning.

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see Data in a behavior graph in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also endianness.

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of bots that are infected by malware and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see About branches (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the Implement break-glass procedures indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use AWS Fault Injection Service (AWS FIS) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See continuous integration and continuous delivery.

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to edge computing technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see Building your Cloud Operating Model.

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.

CMDB

See configuration management database.

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or Bitbucket Cloud. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of AI that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, Amazon SageMaker AI provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see Conformance packs in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see Benefits of continuous delivery. CD can also stand for *continuous deployment*. For more information, see Continuous Delivery vs. Continuous Deployment.

CV

See computer vision.

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see Data classification.

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see Building a data perimeter on AWS.

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See database definition language.

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see Services that work with AWS Organizations in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See environment.

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see Detective controls in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a star schema, a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a disaster. For more information, see Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

See database manipulation language.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

See disaster recovery.

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to detect drift in system resources, or you can use AWS Control Tower to detect changes in your landing zone that might affect compliance with governance requirements.

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with cloud computing, edge computing can reduce communication latency and improve response time.

electronic data interchange (EDI)

The automated exchange of business documents between organizations. For more information, see What is Electronic Data Interchange.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See service endpoint.

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more

information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, MES, and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see Envelope encryption in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a star schema. It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see AWS Fault Isolation Boundaries.

feature branch

See branch.

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with AWS.

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.

few-shot prompting

Providing an LLM with a small number of examples that demonstrate the task and desired output before asking it to perform a similar task. This technique is an application of in-context learning, where models learn from examples (*shots*) that are embedded in prompts. Few-shot prompting can be effective for tasks that require specific formatting, reasoning, or domain knowledge. See also zero-shot prompting.

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through change data capture to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

FM

See foundation model.

foundation model (FM)

A large deep-learning neural network that has been training on massive datasets of generalized and unlabeled data. FMs are capable of performing a wide variety of general tasks, such as understanding language, generating text and images, and conversing in natural language. For more information, see What are Foundation Models.

G

generative AI

A subset of AI models that have been trained on large amounts of data and that can use a simple text prompt to create new content and artifacts, such as images, videos, text, and audio. For more information, see What is Generative AI.

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see Restricting the geographic distribution of your content in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the trunk-based workflow is the modern, preferred approach.

golden image

A snapshot of a system or software that is used as a template to deploy new instances of that system or software. For example, in manufacturing, a golden image can be used to provision software on multiple devices and helps improve speed, scalability, and productivity in device manufacturing operations.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as brownfield. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries.

Detective guardrails detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

holdout data

A portion of historical, labeled data that is withheld from a dataset that is used to train a machine learning model. You can use holdout data to evaluate the model performance by comparing the model predictions against the holdout data.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See infrastructure as code.

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See industrial Internet of Things.

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than mutable infrastructure. For more information, see the Deploy using immutable infrastructure best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by Klaus Schwab in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see Building an industrial Internet of Things (IIoT) digital transformation strategy.

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see What is IoT?

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.

IoT

See Internet of Things.

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.

large language model (LLM)

A deep learning AI model that is pretrained on a vast amount of data. An LLM can perform multiple tasks, such as answering questions, summarizing documents, translating text into other languages, and completing sentences. For more information, see What are LLMs.

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see Apply least-privilege permissions in the IAM documentation.

lift and shift

See 7 Rs.

little-endian system

A system that stores the least significant byte first. See also endianness.

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see Machine Learning.

main branch

See branch.

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See Migration Acceleration Program.

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see Building mechanisms in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the publish/subscribe pattern, for resource-constrained IoT devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners,

migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the Cloud Migration Factory guide in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.

ML

See machine learning.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of immutable infrastructure as a best practice.

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See operations integration.

OLA

See operational-level agreement.

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see Operational Readiness Reviews (ORR) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for Industry 4.0 transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see Creating a trail for an organization in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic `PUT` and `DELETE` requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also OAC, which provides more granular and enhanced access control.

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see Permissions boundaries in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See personally identifiable information.

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

An object that can define permissions (see identity-based policy), specify access conditions (see resource-based policy), or define the maximum permissions for all accounts in an organization in AWS Organizations (see service control policy).

polyglot persistence

Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.

predicate

A query condition that returns `true` or `false`, commonly located in a `WHERE` clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see Preventative controls in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in Roles terms and concepts in the IAM documentation.

privacy by design

A system engineering approach that takes privacy into account through the whole development process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

proactive control

A security control designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the Controls reference guide in the AWS Control Tower documentation and see Proactive controls in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See environment.

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

prompt chaining

Using the output of one LLM prompt as the input for the next prompt to generate better responses. This technique is used to break down a complex task into subtasks, or to iteratively refine or expand a preliminary response. It helps improve the accuracy and relevance of a model's responses and allows for more granular, personalized results.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based MES, a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See responsible, accountable, consulted, informed (RACI).

RAG

See Retrieval Augmented Generation.

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See responsible, accountable, consulted, informed (RACI).

RCAC

See row and column access control.

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See 7 Rs.

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Specify which AWS Regions your account can use.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See 7 Rs.

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

An application's ability to resist or recover from disruptions. High availability and disaster recovery are common considerations when planning for resiliency in the AWS Cloud. For more information, see AWS Cloud Resilience.

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see Responsive controls in *Implementing security controls on AWS*.

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

A generative AI technology in which an LLM references an authoritative data source that is outside of its training data sources before generating a response. For example, a RAG model might perform a semantic search of an organization's knowledge base or custom data. For more information, see What is RAG.

rotation

The process of periodically updating a secret to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see About SAML 2.0-based federation in the IAM documentation.

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata.

The secret value can be binary, a single string, or multiple strings. For more information, see What's in a Secrets Manager secret? in the Secrets Manager documentation.

security by design

A system engineering approach that takes security into account through the whole development process.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: preventative, detective, responsive, and proactive.

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as detective or responsive security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see Service control policies in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a service-level indicator.

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see Shared responsibility model.

SIEM

See security information and event management system.

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your

organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a data warehouse or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use Amazon CloudWatch Synthetics to create these tests.

system prompt

A technique for providing context, instructions, or guidelines to an LLM to direct its behavior. System prompts help set context and establish rules for interactions with users.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see Tagging your AWS resources.

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see Using AWS Organizations with other AWS services in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn't provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See environment.

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered immutable.

Z

zero-day exploit

An attack, typically malware, that takes advantage of a zero-day vulnerability.

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zero-shot prompting

Providing an LLM with instructions for performing a task but no examples (*shots*) that can help guide it. The LLM must use its pre-trained knowledge to handle the task. The effectiveness of zero-shot prompting depends on the complexity of the task and the quality of the prompt. See also few-shot prompting.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.