Implementing managed PostgreSQL for multi-tenant SaaS applications on AWS

AWS Prescriptive Guidance

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

AWS Prescriptive Guidance: Implementing managed PostgreSQL for multi-tenant SaaS applications on AWS

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

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Introduction **1**
Targeted business outcomes 1
Selecting a database for a SaaS application **3**
Choosing between Amazon RDS and Aurora 5
Multi-tenant SaaS partitioning models for PostgreSQL **7**
PostgreSQL silo model 8
PostgreSQL pool model 9
PostgreSQL bridge model 11
Decision matrix 12
Row-level security recommendations **25**
PostgreSQL availability for the pool model **27**
Best practices **29**
Compare AWS options for managed PostgreSQL 29
Select a multi-tenant SaaS partitioning model 29
Use row-level security for pool SaaS partitioning models 29
FAQ **30**
Which managed PostgreSQL options does AWS offer? 30
Which service is optimal for SaaS applications? 30
Which unique requirements should I consider if I decide to use a PostgreSQL database with a multi-tenant SaaS application? 30
Which models can I use to maintain tenant data isolation with PostgreSQL? 30
How do I maintain tenant data isolation with a single PostgreSQL database that is shared across multiple tenants? 31
Next steps **32**
Resources **33**
References 33
Partners 33
Document history **34**
Glossary **35**
..... 35
A 36
B 39
C 41
D 44

E 48
F 50
G 51
H 52
I 53
L 56
M 57
O 61
P 63
Q 66
R 66
S 69
T 73
U 74
V 75
W 75
Z 76

Implementing managed PostgreSQL for multi-tenant SaaS applications on AWS

Tabby Ward and Thomas Davis, Amazon Web Services (AWS)

April 2024 (document history)

When you select a database to store operational data, it is crucial to consider how the data should be structured, which queries it will answer, how fast it will provide answers, and the resiliency of the data platform itself. In addition to these general considerations are software as a service (SaaS) implications for operational data, such as performance isolation, tenant security, and unique characteristics and design patterns that are typical of data for multi-tenant SaaS applications. This guide discusses how these factors apply to using a PostgreSQL database on Amazon Web Services (AWS) as the primary operational data store for a multi-tenant SaaS application. Specifically, the guide focuses on two AWS managed PostgreSQL options: Amazon Aurora PostgreSQL-Compatible Edition and Amazon Relational Database Service (Amazon RDS) for PostgreSQL.

Targeted business outcomes

This guidance provides a detailed analysis of best practices for multi-tenant SaaS applications using Aurora PostgreSQL-Compatible and Amazon RDS for PostgreSQL. We recommend that you use the design patterns and concepts provided in this guide to inform and standardize your implementation of Aurora PostgreSQL-Compatible or Amazon RDS for PostgreSQL for your multi-tenant SaaS applications.

This prescriptive guidance helps achieve the following business outcomes:

- **Choosing the most optimal AWS managed PostgreSQL option for your use case** – This guidance compares relational and non-relational options for database usage with SaaS applications. It also discusses which use cases are most optimal for Aurora PostgreSQL-Compatible and Amazon RDS for PostgreSQL. This information will assist in selecting the best option for your SaaS application.
- **Enforcement of SaaS best practices through the adoption of a SaaS partitioning model** – This guide discusses and compares three broad SaaS partitioning models that are applicable to a PostgreSQL database management system (DBMS): pool, bridged, and silo models, and their variations. These approaches capture SaaS best practices and provide flexibility when designing

a SaaS application. The enforcement of a SaaS partitioning model is a crucial part of preserving best practices.

- **Effective use of RLS in pool SaaS partitioning models** – Row-level security (RLS) supports the enforcement of tenant data isolation within a single PostgreSQL table by restricting the rows that can be viewed based on the user or a context variable. When you use the pool partitioning model, RLS is required to prevent cross-tenant access.

Selecting a database for a SaaS application

For many multi-tenant SaaS applications, selecting an operational database can be distilled into a choice between relational and non-relational databases, or a combination of the two. To make your decision, consider these high-level application data requirements and characteristics:

- Data model of the application
- Access patterns for the data
- Database latency requirements
- Data integrity and transactional integrity requirements (atomicity, consistency, isolation, and durability, or ACID)
- Cross-Region availability and recovery requirements

The following table lists application data requirements and characteristics, and discusses them in the context of AWS database offerings: Aurora PostgreSQL-Compatible and Amazon RDS for PostgreSQL (relational), and Amazon DynamoDB (non-relational). You can reference this matrix when you’re trying to decide between relational and non-relational operational database offerings.

Databases	**SaaS application data requirements and characteristics**				
	Data model	**Access patterns**	**Latency requirements**	**Data and transactional integrity**	**Cross-Region availability and recovery**
Relational (Aurora PostgreSQL-Compatible and Amazon RDS for PostgreSQL)	Relational or highly normalized.	Doesn’t have to be thoroughly planned beforehand.	Preferably higher latency tolerance; can achieve lower latencies by default with Aurora and by implement	High data and transactional integrity maintained by default.	In Amazon RDS, you can create a read replica for cross-Region scaling and failover. Aurora mostly automates this process.

			ing read replicas, caching, and similar features.		For active-active configurations across multiple AWS Regions, you can use write forwarding in conjunction with Aurora global databases.
Non-relational (Amazon DynamoDB)	Usually denormalized. These databases take advantage of patterns for modeling many-to-many relationships, large items, and time series data.	All access patterns (queries) for data must be thoroughly understood before a data model is produced.	Very low latency with options such as Amazon DynamoDB Accelerator (DAX) able to improve performance even further.	Optional transactional integrity at the cost of performance. Data integrity concerns are shifted to the application.	Easy cross-Region recovery and active-active configuration with global tables. (ACID compliance is achievable only in a single AWS Region.)

Some multi-tenant SaaS applications might have unique data models or special circumstances that are better served by databases not included in the previous table. For example, time series datasets, highly connected datasets, or maintaining a centralized transaction ledger might necessitate using a different type of database. Analyzing all possibilities is beyond the scope of this guide. For a comprehensive list of AWS database offerings and how they can fulfill different use cases at a high level, see the Database section of the *Overview of Amazon Web Services* whitepaper.

The remainder of this guide focuses on AWS relational database services that support PostgreSQL: Amazon RDS and Aurora PostgreSQL-Compatible. DynamoDB requires a different approach to

optimize for SaaS applications, which is beyond the scope of this guide. For more information about DynamoDB, see the AWS blog post Partitioning Pooled Multi-Tenant SaaS Data with Amazon DynamoDB.

Choosing between Amazon RDS and Aurora

In most cases, we recommend using Aurora PostgreSQL-Compatible over Amazon RDS for PostgreSQL. The following table shows the factors that you should consider when deciding between these two options.

DBMS component	Amazon RDS for PostgreSQL	Aurora PostgreSQL-Compatible
Scalability	Replication lag of minutes, maximum of 5 read replicas	Replication lag under a minute (typically less than 1 second with global databases), maximum of 15 read replicas
Crash recovery	Checkpoints 5 minutes apart (by default), can slow database performance	Asynchronous recovery with parallel threads for rapid recovery
Failover	60-120 seconds in addition to crash recovery time	Usually about 30 seconds (including crash recovery)
Storage	Maximum IOPS of 256,000	IOPS constrained only by Aurora instance size and capacity
High availability and disaster recovery	Two Availability Zones with a standby instance, cross-Region failover to read replica or copied backups	Three Availability Zones by default, cross-Region failover with Aurora global databases, write forwarding across AWS Regions for active-active configurations

DBMS component	Amazon RDS for PostgreSQL	Aurora PostgreSQL-Compatible
Backup	During backup window, can impact performance	Automatic incremental backups, no performance impact
Database instance classes	See list of Amazon RDS instance classes	See list of Aurora instance classes

In all the categories described in the previous table, Aurora PostgreSQL-Compatible is usually the better option. However, Amazon RDS for PostgreSQL might still make sense for small to medium workloads, because it has a greater selection of instance classes that might provide a more cost-effective option at the expense of Aurora's more robust feature set.

Multi-tenant SaaS partitioning models for PostgreSQL

The best method for accomplishing multi-tenancy depends on the requirements for your SaaS application. The following sections demonstrate partitioning models for successfully implementing multi-tenancy in PostgreSQL.

> **Note**
>
> The models discussed in this section are applicable to both Amazon RDS for PostgreSQL and Aurora PostgreSQL-Compatible. References to *PostgreSQL* in this section apply to both services.

There are three high-level models that you can use in PostgreSQL for SaaS partitioning: silo, bridge, and pool. The following image summarizes the trade-offs between the silo and pool models. The bridge model is a hybrid of the silo and pool models.

Partitioning model	Advantages	Disadvantages
Silo	• Compliance alignment • No cross-tenant impact • Tenant-level tuning • Tenant-level availability	• Compromised agility • No centralized managemen t • Deployment complexity • Cost
Pool	• Agility • Cost optimization • Centralized management • Simplified deployment	• Cross-tenant impact • Compliance challenges • All or nothing availability
Bridge	• Some compliance alignment • Agility • Cost optimization	• Some compliance challenge s • All or nothing availability (mostly) • Cross-tenant impact

Partitioning model	Advantages	Disadvantages
	• Centralized management	• Deployment complexity

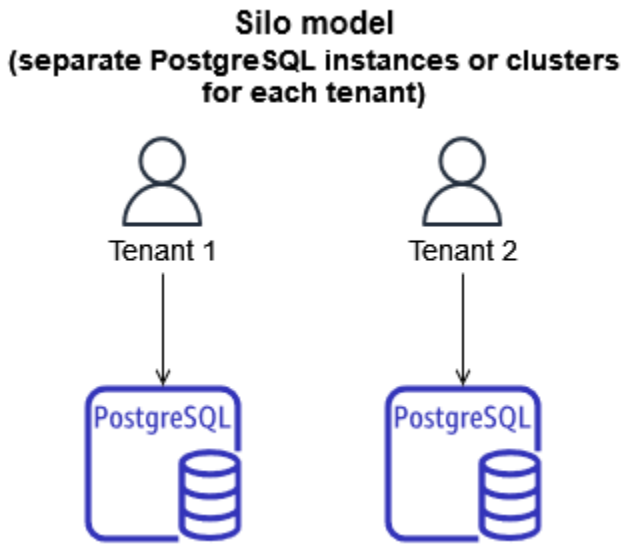
The following sections discuss each model in more detail.

Partitioning models:

- PostgreSQL silo model
- PostgreSQL pool model
- PostgreSQL bridge model
- Decision matrix

PostgreSQL silo model

The silo model is implemented by provisioning a PostgreSQL instance for each tenant in an application. The silo model excels at tenant performance and security isolation, and completely eliminates the *noisy neighbor* phenomenon. The noisy neighbor phenomenon occurs when one tenant's usage of a system affects the performance of another tenant. The silo model lets you tailor performance specifically to each tenant and potentially limit outages to a specific tenant's silo. However, what generally drives adoption of a silo model is strict security and regulatory constraints. These constraints can be motivated by SaaS customers. For example, SaaS customers might demand that their data be isolated due to internal constraints, and SaaS providers might offer such a service for an additional fee.

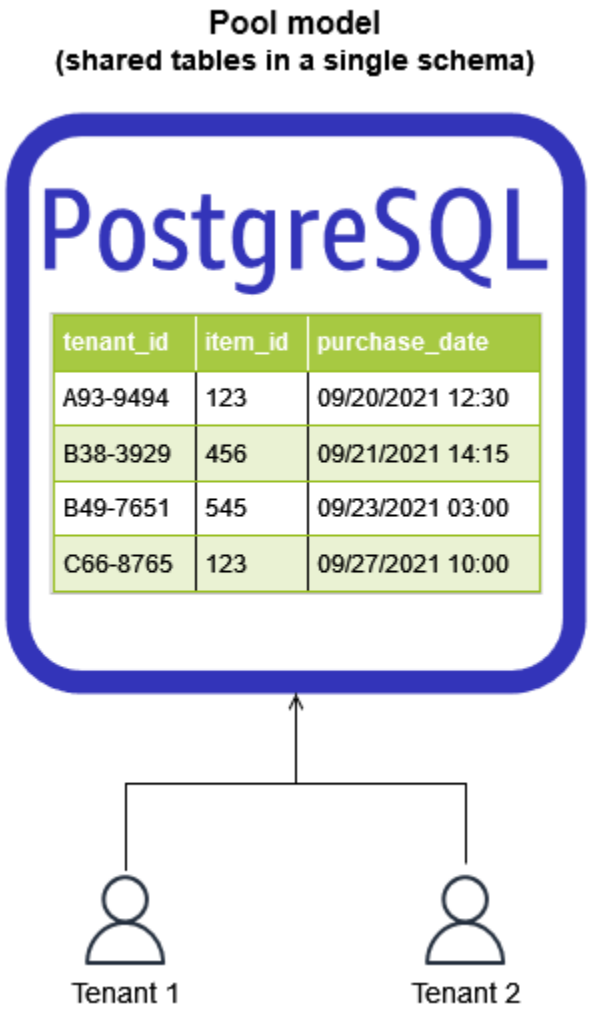


Although the silo model might be necessary in certain cases, it has many drawbacks. It is often difficult to use the silo model in a cost-effective manner, because managing resource consumption across multiple PostgreSQL instances can be complicated. Furthermore, the distributed nature of database workloads in this model makes it more difficult to maintain a centralized view of tenant activity. Managing so many independently operated workloads increases operational and administrative overhead. The silo model also makes tenant onboarding more complicated and time-consuming, because you have to provision tenant-specific resources. Furthermore, the entire SaaS system can be harder to scale, because the ever-increasing number of tenant-specific PostgreSQL instances will demand more operational time to administer. One last consideration is that an application or a data access layer will have to maintain a mapping of tenants to their associated PostgreSQL instances, which adds to the complexity of implementing this model.

PostgreSQL pool model

The pool model is implemented by provisioning a single PostgreSQL instance (Amazon RDS or Aurora) and using row-level security (RLS) to maintain tenant data isolation. RLS policies restrict which rows in a table are returned by `SELECT` queries or which rows are affected by `INSERT`, `UPDATE`, and `DELETE` commands. The pool model centralizes all tenant data in a single PostgreSQL schema, so it is significantly more cost-effective and requires less operational overhead to maintain. Monitoring this solution is also significantly simpler due to its centralization. However, monitoring tenant-specific impacts in the pool model usually requires some additional instrumentation in the application. This is because PostgreSQL by default isn't aware of which tenant is consuming resources. Tenant onboarding is simplified because no new infrastructure

is required. This agility makes it easier to accomplish rapid and automated tenant onboarding workflows.

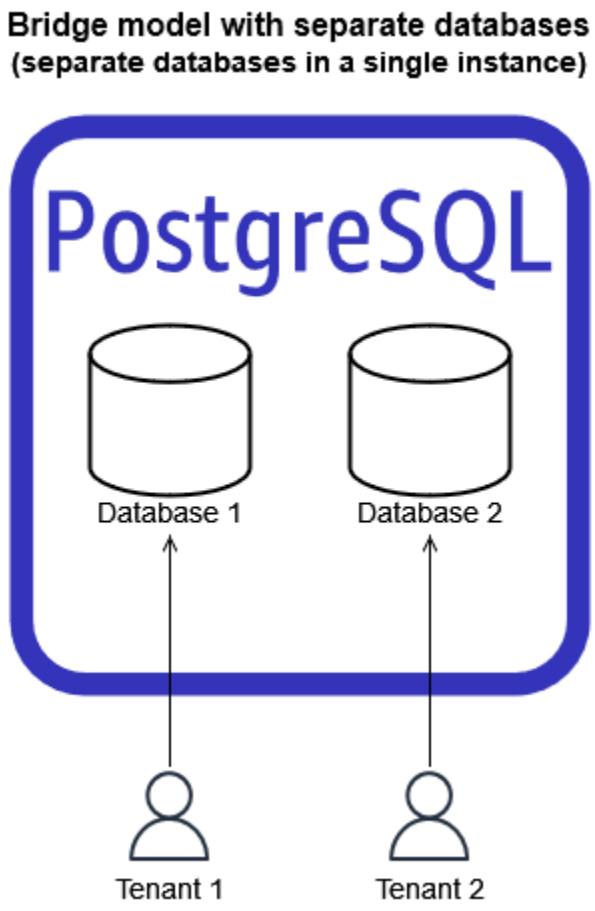


Although the pool model is generally more cost-effective and simpler to administer, it does have some disadvantages. The noisy neighbor phenomenon cannot be completely eliminated in a pool model. However, it can be mitigated by ensuring that appropriate resources are available on the PostgreSQL instance and by using strategies to reduce the load in PostgreSQL, such as offloading queries to read replicas or to Amazon ElastiCache. Effective monitoring also plays a role in responding to tenant performance isolation concerns, because application instrumentation can log and monitor tenant-specific activity. Lastly, some SaaS customers might not find the logical separation provided by RLS to be sufficient and might ask for additional isolation measures.

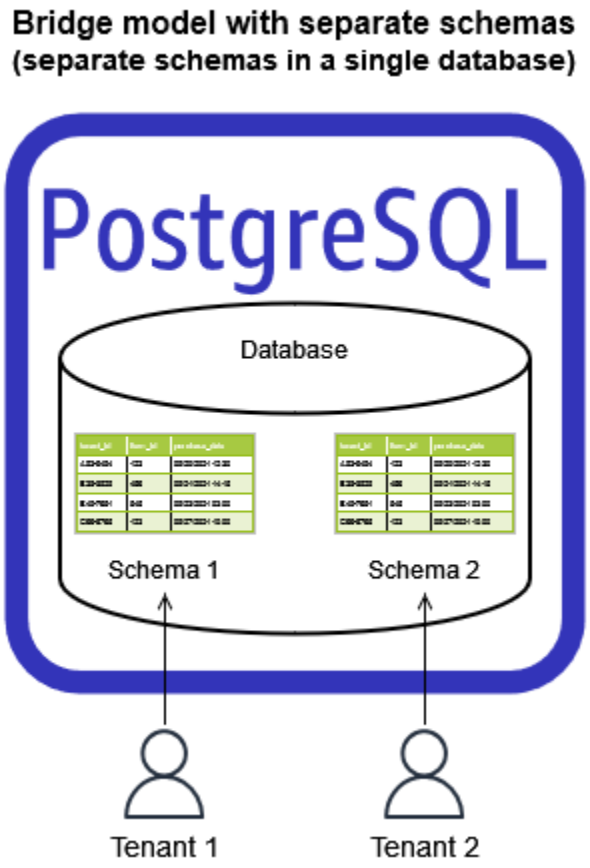
PostgreSQL bridge model

The PostgreSQL bridge model is a combination of the pooled and siloed approaches. Like the pooled model, you provision a single PostgreSQL instance for each tenant. To maintain tenant data isolation, you use PostgreSQL logical constructs. In the following diagram, PostgreSQL databases are used to logically separate data.

> **Note**
>
> A PostgreSQL database doesn't refer to a separate Amazon RDS for PostgreSQL or Aurora PostgreSQL-Compatible DB instance. Instead, it refers to a logical construct of the PostgreSQL database management system to separate data.



You can also implement the bridge model by using a single PostgreSQL database, with tenant-specific schemas in each database, as illustrated in the following diagram.



The bridge model suffers from the same noisy neighbor and tenant performance isolation concerns as the pool model. It also incurs some additional operational and provisioning overhead by requiring either separate databases or schemas to be provisioned on a per-tenant basis. It requires effective monitoring to respond quickly to tenant performance concerns. It also requires application instrumentation to monitor tenant-specific usage. Overall, the bridge model can be viewed as an alternative to RLS that slightly augments the tenant onboarding effort by requiring new PostgreSQL databases or schemas. As with the silo model, an application or a data access layer will have to maintain a mapping of tenants to their associated PostgreSQL databases or schemas.

Decision matrix

To decide which multi-tenant SaaS partitioning model you should use with PostgreSQL, consult the following decision matrix. The matrix analyzes these four partitioning options:

- Silo – A separate PostgreSQL instance or cluster for each tenant.

- Bridge with separate databases – A separate database for each tenant in a single PostgreSQL instance or cluster.
- Bridge with separate schemas – A separate schema for each tenant in a single PostgreSQL database, in a single PostgreSQL instance or cluster.
- Pool – Shared tables for tenants in a single instance and schema.

	Silo	**Bridge with separate databases**	**Bridge with separate schemas**	**Pool**
Use case	Isolation of data with full control of resource usage is a key requirement, or you have very large and very performan ce-sensitive tenants.	Isolation of data is a key requirement, and limited or no cross-reference of tenants' data is required.	Moderate number of tenants with a moderate amount of data. This is the preferred model if you have to cross-reference tenants' data.	Large number of tenants with less data per tenant.
New tenant onboarding agility	Very slow. (A new instance or cluster is required for each tenant.)	Moderately slow. (Requires creating a new database for each tenant to store schema objects.)	Moderately slow. (Requires creating a new schema for each tenant to store objects.)	Fastest option. (Minimal setup is required.)
Database connection pool configura tion effort and efficiency	Significant effort required. (One connection pool per tenant.) Less efficient . (No database	Significant effort required. (One connection pool configura tion per tenant unless you use	Less effort required. (One connection pool configuration for all tenants.)	Least effort required. Most efficient. (One connectio n pool for all tenants

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
	connection sharing between tenants.)	Amazon RDS Proxy.) Less efficient. (No database connection sharing between tenants and total number of connections. Usage across all tenants is limited based on the DB instance class.)	Moderately efficient. (Connection reuse through the `SET ROLE` or `SET SCHEMA` command in session pool mode only. `SET` commands also cause session pinning when using Amazon RDS Proxy, but the client connection pools can be eliminated and direct connections can be made for each request for efficiency.)	and efficient connection reuse across all tenants. Database connection limits are based on the DB instance class.)

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Database maintenance (vacuum management) and resource usage	Simpler management.	Medium complexity. (Might lead to high resource consumption, because a vacuum worker has to be started for each database after `vacuum_naptime`, which leads to high autovacuum launcher CPU usage. There might also be additional overhead associated with vacuuming the PostgreSQL system catalog tables for each database.)	Large PostgreSQL system catalog tables. (Total `pg_catalog` size in tens of GBs, depending on number of tenants and relations. Likely to require modifications to vacuuming-related parameters to control table bloat.)	Tables might be large, depending on the number of tenants and data per tenant. (Likely to require modifications to vacuuming-related parameters to control table bloat.)
Extensions management effort	Significant effort (for each database in separate instances).	Significant effort (at each database level).	Minimal effort (one time in the common database).	Minimal effort (one time in the common database).

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Change deployment effort	Significant effort. (Connect to each separate instance and roll out changes.)	Significant effort. (Connect to each database and schema, and roll out changes.)	Moderate effort. (Connect to common database and roll out changes for each schema.)	Minimal effort. (Connect to common database and roll out changes.)
Change deployment – scope of impact	Minimal. (Single tenant affected.)	Minimal. (Single tenant affected.)	Minimal. (Single tenant affected.)	Very large. (All tenants affected.)
Query performance management and effort	Manageabl e query performance.	Manageabl e query performance.	Manageabl e query performance.	Significant effort likely required to maintain query performance. (Over time, queries might run more slowly due to the increased size of tables. You can use table partitioning and database sharding to maintain performance.)

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Cross-tenant resource impact	No impact. (No resource sharing among tenants.)	Moderate impact. (Tenants share common resources such as instance CPU and memory.)	Moderate impact. (Tenants share common resources such as instance CPU and memory.)	Heavy impact. (Tenants affect one another in terms of resources, lock conflicts, and so on.)
Tenant-level tuning (for example, creation of additional indexes per tenant or DB parameter tweaking for a particular tenant)	Possible.	Somewhat possible. (Schema-level changes can be made for each tenant, but database parameters are global across all tenants.)	Somewhat possible. (Schema-level changes can be made for each tenant, but database parameters are global across all tenants.)	Not possible. (Tables are shared by all tenants.)

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Rebalance effort for performance-sensitive tenants	Minimal. (No need to rebalance. Scale server and I/O resources to handle this scenario.)	Moderate. (Use logical replication or `pg_dump` to export the database, but downtime might be lengthy depending on data size. You can use the transportable database feature in Amazon RDS for PostgreSQL to copy databases between instances faster.)	Moderate but likely involves lengthy downtime. (Use logical replication or `pg_dump` to export the schema, but downtime might be lengthy depending on data size.)	Significant, because all tenants share the same tables. (Sharding the database requires copying everything to another instance and an additional step to clean up tenant data.) Most likely requires a change in application logic.
Database downtime for major version upgrades	Standard downtime. (Depends on PostgreSQL system catalog size.)	Longer downtime likely. (Depending on system catalog size, the time will vary. PostgreSQL system catalog tables are also duplicated across databases)	Longer downtime likely. (Depending on PostgreSQL system catalog size, the time will vary.)	Standard downtime. (Depends on PostgreSQL system catalog size.)

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Administration overhead (for example, for database log analysis or backup job monitoring)	Significant effort	Minimal effort.	Minimal effort.	Minimal effort.
Tenant-level availability	Highest. (Each tenant fails and recovers independently.)	Higher scope of impact. (All tenants fail and recover together in case of hardware or resource issues.)	Higher scope of impact. (All tenants fail and recover together in case of hardware or resource issues.)	Higher scope of impact. (All tenants fail and recover together in case of hardware or resource issues.)
Tenant-level backup and recovery effort	Least effort. (Each tenant can be backed up and restored independently.)	Moderate effort. (Use logical export and import for each tenant. Some coding and automation are required.)	Moderate effort. (Use logical export and import for each tenant. Some coding and automation are required.)	Significant effort. (All tenants share the same tables.)
Tenant-level point-in-time recovery effort	Minimal effort. (Use point-in time recovery by using snapshots, or use backtrack ing in Amazon Aurora.)	Moderate effort. (Use snapshot restore, followed by export/im port. However, this will be a slow operation.)	Moderate effort. (Use snapshot restore, followed by export/im port. However, this will be a slow operation.)	Significant effort and complexity.

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Uniform schema name	Same schema name for each tenant.	Same schema name for each tenant.	Different schema for each tenant.	Common schema.
Per-tenant customization (for example, additional table columns for a specific tenant)	Possible.	Possible.	Possible.	Complicated (because all tenants share the same tables).
Catalog management efficiency at object-relational mapping (ORM) layer (for example, Ruby)	Efficient (because the client connection is specific for a tenant).	Efficient (because the client connection is specific to a database).	Moderately efficient. (Depending on the ORM used, user/role security model, and `search_path` configuration, the client sometimes caches the metadata for all tenants, leading to high DB connection memory usage.)	Efficient (because all tenants share the same tables).

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Consolidated tenant reporting effort	Significant effort. (You have to use foreign data wrappers [FDWs] to consolidate data in all tenants or extract, transform, and load [ETL] to another reporting database.)	Significant effort. (You have to use FDWs to consolidate data in all tenants or ETL to another reporting database.)	Moderate effort. (You can aggregate data in all schemas by using unions.)	Minimal effort. (All tenant data is in the same tables, so reporting is simple.)
Tenant-specific read-only instance for reporting (for example, based on subscription)	Least effort. (Create a read replica.)	Moderate effort. (You can use logical replication or AWS Database Migration Service [AWS DMS] to configure.)	Moderate effort. (You can use logical replication or AWS DMS to configure.)	Complicated (because all tenants share the same tables).

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Data isolation	Best.	Better. (You can manage database-level permissions by using PostgreSQL roles.)	Better. (You can manage schema-level permissions by using PostgreSQL roles.)	Worse. (Because all tenants share the same tables, you have to implement features such as row-level security [RLS] for tenant isolation.)
Tenant-specific storage encryption key	Possible. (Each PostgreSQL cluster can have its own AWS Key Management Service [AWS KMS] key for storage encryption.)	Not possible. (All tenants share the same KMS key for storage encryption.)	Not possible. (All tenants share the same KMS key for storage encryption.)	Not possible. (All tenants share the same KMS key for storage encryption.)
Using AWS Identity and Access Management (IAM) for database authentication for each tenant	Possible.	Possible.	Possible (by having separate PostgreSQL users for each schema).	Not possible (because tables are shared by all tenants).
Infrastructure cost	Highest (because nothing is shared).	Moderate.	Moderate.	Lowest.

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Data duplication and storage usage	Highest aggregate across all tenants. (PostgreSQL system catalog tables and the application's static and common data are duplicated across all tenants.)	Highest aggregate across all tenants. (PostgreSQL system catalog tables and the application's static and common data are duplicated across all tenants.)	Moderate. (The application's static and common data can be in a common schema and accessed by other tenants.)	Minimal. (No duplication of data. The application's static and common data can be in the same schema.)
Tenant-centric monitoring (quickly find out which tenant is causing issues)	Least effort. (Because each tenant is monitored separately, it's easy to check the activity of a specific tenant.)	Moderate effort. (Because all tenants share the same physical resource, you have to apply additional filtering to check the activity of a specific tenant.)	Moderate effort. (Because all tenants share the same physical resource, you have to apply additional filtering to check the activity of a specific tenant.)	Significant effort. (Because all tenants share all resources, including tables, you have to use bind variable capture to check which tenant a specific SQL query belongs to.)

	Silo	Bridge with separate databases	Bridge with separate schemas	Pool
Centralized management and health/activity monitoring	Significant effort (to set up central monitoring and a central command center).	Moderate effort (because all tenants share the same instance).	Moderate effort (because all tenants share the same instance).	Minimal effort (because all tenants share the same resources, including the schema).
Chances of object identifier (OID) and transaction ID (XID) wraparound	Minimal.	High. (Because OID,XID is a single PostgreSQL clusterwide counter and there can be issues vacuuming effectively across physical databases).	Moderate. (Because OID,XID is a single PostgreSQL clusterwide counter).	High. (For example, a single table can reach the TOAST OID limit of 4 billion, depending on the number of out-of-line columns.)

Row-level security recommendations

Row-level security (RLS) is required to maintain tenant data isolation in a pooled model with PostgreSQL. RLS centralizes the enforcement of isolation policies at the database level and removes the burden of maintaining this isolation from software developers. The most common way to implement RLS is to enable this feature in the PostgreSQL DBMS. RLS involves filtering access to rows of data based on a value in a specified column. You can use two methods to filter access to data:

- A specified column of data in a table is compared to the value of the current PostgreSQL user. Values in the column that are equivalent to the logged-in PostgreSQL user are accessible to that user.
- A specified column of data in a table is compared to the value of a runtime variable set by the application. Values in the column that are equivalent to the runtime variable are accessible during that session.

The second option is preferred, because the first option requires the creation of a new PostgreSQL user for each tenant. Instead, a SaaS application that uses PostgreSQL should be responsible for setting a tenant-specific context at runtime when it queries PostgreSQL. This will have the effect of enforcing RLS. You can also enable RLS on a table-by-table basis. As a best practice, you should enable RLS on all tables that contain tenant data.

The following example creates two tables and enables RLS. This example compares a column of data to the value of the runtime variable `app.current_tenant`.

```
-- Create a table for our tenants with indexes on the primary key and the tenant’s name
CREATE TABLE tenant (
    tenant_id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
    name VARCHAR(255) UNIQUE,
    status VARCHAR(64) CHECK (status IN ('active', 'suspended', 'disabled')),
    tier VARCHAR(64) CHECK (tier IN ('gold', 'silver', 'bronze'))
);

-- Create a table for users of a tenant
CREATE TABLE tenant_user (
    user_id UUID DEFAULT uuid_generate_v4() PRIMARY KEY,
    tenant_id UUID NOT NULL REFERENCES tenant (tenant_id) ON DELETE RESTRICT,
    email VARCHAR(255) NOT NULL UNIQUE,
```

```
    given_name VARCHAR(255) NOT NULL CHECK (given_name <> ''),
    family_name VARCHAR(255) NOT NULL CHECK (family_name <> '')
);

-- Turn on RLS
ALTER TABLE tenant ENABLE ROW LEVEL SECURITY;

-- Restrict read and write actions so tenants can only see their rows
-- Cast the UUID value in tenant_id to match the type current_setting
-- This policy implies a WITH CHECK that matches the USING clause
CREATE POLICY tenant_isolation_policy ON tenant
USING (tenant_id = current_setting('app.current_tenant')::UUID);

-- And do the same for the tenant users
ALTER TABLE tenant_user ENABLE ROW LEVEL SECURITY;

CREATE POLICY tenant_user_isolation_policy ON tenant_user
USING (tenant_id = current_setting('app.current_tenant')::UUID);
```

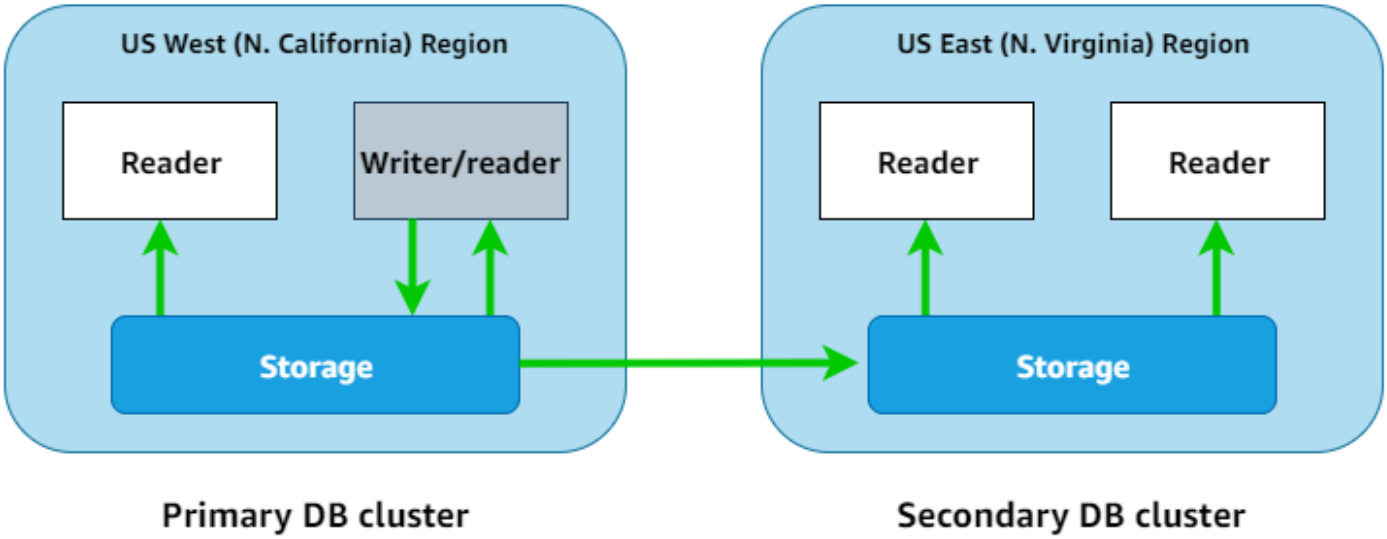
For more information, see the blog post Multi-tenant data isolation with PostgreSQL Row Level Security. The AWS SaaS Factory team also has some examples in GitHub to assist in implementing RLS.

PostgreSQL availability for the pool model

Pool models by their nature have only a single PostgreSQL instance. Therefore, designing your application for high availability is crucial. A failure or outage of a pooled database results in your application being degraded or becoming inaccessible for all your tenants.

Amazon RDS for PostgreSQL DB instances can be made redundant across two Availability Zones by enabling the high-availability feature. For more information, see High availability (Multi-AZ) for Amazon RDS in the Amazon RDS documentation. For cross-Region failover, you can create a read replica in a different AWS Region. (This read replica has to be promoted as part of a failover process.) In addition, you can replicate backups replicated across AWS Regions for recovery. For more information, see Replicating automated backups to another AWS Region in the Amazon RDS documentation.

Aurora PostgreSQL-Compatible automatically backs up data in a way that can sustain the failure of multiple Availability Zones. (See High availability for Amazon Aurora in the Aurora documentation.) To make Aurora more resilient and recover faster, you can create Aurora read replicas in other Availability Zones. You can use Aurora global databases to replicate data into five additional AWS Regions for cross-Region recovery and automatic failover. (See Using Amazon Aurora global databases in the Aurora documentation.) Additionally, you can enable write forwarding with Aurora global databases to achieve high availability across multiple AWS Regions.



Regardless of whether you're using Amazon RDS for PostgreSQL or Aurora PostgreSQL-Compatible, we recommend that you implement high availability features to mitigate the impact of any outages for all multi-tenant SaaS applications that use a pool model.

Best practices

This section lists some of the high-level takeaways from this guide. For detailed discussions on each point, follow the links to the corresponding sections.

Compare AWS options for managed PostgreSQL

AWS offers two primary ways to run PostgreSQL in a managed environment. (In this context, managed means that the PostgreSQL infrastructure and DBMS are partially or completely supported by an AWS service.) Managed PostgreSQL options on AWS have the benefit of automating backups, failover, optimization, and some administration of PostgreSQL. As managed options, AWS offers Amazon Aurora PostgreSQL-Compatible Edition and Amazon Relational Database Service (Amazon RDS) for PostgreSQL. You can select the best choice from these two models by analyzing your PostgreSQL use case. For more information, see the section Choosing between Amazon RDS and Aurora in this guide.

Select a multi-tenant SaaS partitioning model

You can choose from three SaaS partitioning models that are applicable to PostgreSQL: silo, bridge, and pool. Each model has advantages and disadvantages, and you should choose the most optimal model depending on your use case. Amazon RDS for PostgreSQL and Aurora PostgreSQL-Compatible support all three models. Choosing a model is critical to maintaining tenant data isolation in your SaaS applications. For a detailed discussion of these models, see the section Multi-tenant SaaS partitioning models for PostgreSQL in this guide.

Use row-level security for pool SaaS partitioning models

Row-level security (RLS) is required to maintain tenant data isolation in a pool model with PostgreSQL. This is because there is no logical separation between infrastructure, PostgreSQL databases, or schemas on a per-tenant basis in a pool model. RLS centralizes the enforcement of isolation policies at the database level and removes the burden of maintaining this isolation from software developers. You can use RLS to limit database operations to a specific tenant. For more information and an example, see the section Row-level security recommendations in this guide.

FAQ

This section provides answers to commonly raised questions about implementing managed PostgreSQL in multi-tenant SaaS applications.

Which managed PostgreSQL options does AWS offer?

AWS offers Amazon Aurora PostgreSQL-Compatible and Amazon Relational Database Service (Amazon RDS) for PostgreSQL. AWS also has a broad catalog of managed database offerings.

Which service is optimal for SaaS applications?

You can use both Aurora PostgreSQL-Compatible and Amazon RDS for PostgreSQL for SaaS applications and all the SaaS partitioning models discussed in this guide. These two services have differences in scalability, crash recovery, failover, storage options, high availability, disaster recovery, backup, and the instance classes available for each option. The optimal choice will depend on your specific use case. Use the decision matrix in this guide to choose the best option for your use case.

Which unique requirements should I consider if I decide to use a PostgreSQL database with a multi-tenant SaaS application?

As with any data store used with a SaaS application, the most important consideration is the method for maintaining tenant data isolation. As discussed in this guide, there are multiple ways you can achieve tenant data isolation with AWS managed PostgreSQL offerings. Additionally, you should consider performance isolation on a per-tenant basis for any PostgreSQL implementations.

Which models can I use to maintain tenant data isolation with PostgreSQL?

You can use the silo, bridge, and pool models as SaaS partitioning strategies to maintain tenant data isolation. For a discussion of these models and how they can be applied to PostgreSQL, see the section Multi-tenant SaaS partitioning models for PostgreSQL in this guide.

How do I maintain tenant data isolation with a single PostgreSQL database that is shared across multiple tenants?

PostgreSQL supports a row-level security (RLS) feature that you can use to enforce tenant data isolation in a single PostgreSQL database or instance. Additionally, you can provision separate PostgreSQL databases per tenant in a single instance, or create schemas on a per-tenant basis to achieve this goal. For the advantages and disadvantages of these approaches, see the section Row-level security recommendations in this guide.

Next steps

AWS offers two options for operating managed PostgreSQL: Aurora PostgreSQL-Compatible and Amazon RDS for PostgreSQL. We recommend that you evaluate the two services and choose the option that best supports your specific use case for your multi-tenant SaaS applications. Conforming to a SaaS partitioning model can ensure that a SaaS application that uses PostgreSQL adheres strictly to best practices to maintain tenancy. The SaaS silo, bridge, and pool partitioning models support many SaaS use cases. These models provide varying advantages among factors such as performance isolation, operational overhead, and tenant security.

Next steps:

- Evaluate Aurora PostgreSQL-Compatible and Amazon RDS for PostgreSQL, and pick the best option for your SaaS application.
- Select a SaaS partitioning model that meets the requirements for your application: silo, bridge, or pool.
- Implement PostgreSQL in accordance with your selected SaaS partitioning model.

Resources

References

- SaaS Storage Strategies: Building a Multitenant Storage Model on AWS (AWS whitepaper)
- Cross-Region disaster recovery using Amazon Aurora Global Database for Amazon Aurora PostgreSQL (AWS blog post)
- Multitenant data isolation with PostgreSQL Row Level Security (AWS blog post)
- Working with Amazon Aurora PostgreSQL (Aurora documentation)
- PostgreSQL on Amazon RDS (Amazon RDS documentation)

Partners

- Amazon Aurora for PostgreSQL Partners
- Amazon RDS for PostgreSQL Partners

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
Update	Updates to reflect the availability of write forwardin g in Aurora.	April 29, 2024
Update	Updated the Amazon RDS and Aurora comparison table.	October 21, 2022
—	Initial publication	September 30, 2021

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 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 AWS CodeCommit. 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, AWS Panorama offers devices that add CV to on-premises camera networks, and Amazon SageMaker 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.

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.

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.

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.

G

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.

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.

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 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.

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

An approach in system engineering that takes privacy into account throughout the whole engineering 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.

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).

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.

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 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.

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.

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.