Multi-tenancy guidance for ISVs running Amazon Neptune databases

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Multi-tenancy guidance for ISVs running Amazon Neptune databases

Table of Contents

Multi-tenancy guidance for ISVs running Amazon Neptune databases

Amazon Web Services (contributors)

August 2024 (document history)

Multi-tenancy is a computer systems architecture where a single instance of an application serves multiple customers. Each customer is referred to as a *tenant*. In a multi-tenant architecture, these instances of the application operate in a shared environment where each tenant is physically co-located on the same infrastructure but is logically separated.

As an independent software vendor (ISV), you can use Amazon Neptune to power applications that require navigation across highly connected data. You might be managing a cloud-based software as a service (SaaS) application in your account and providing tenants with subscriptions. Tenants can then access the service over the internet or privately over AWS PrivateLink. The economics of this model work for both parties, because the tenant gets access to software that's less expensive than it would be for them to purchase, build, and maintain. As the ISV, you can charge more for the subscription than it costs you to create and maintain the software. The question is how do you scale your business to multiple tenants.

Multi-tenancy provides ISVs with important economical and operational benefits. The multi-tenant architecture gives your organization a better return on investment (ROI). Multi-tenancy also simplifies operational requirements so that your organization can move more quickly and reduce the cost of delivering the software to your tenants.

This document provides guidance on effectively running a multi-tenant ISV application using Amazon Neptune. This guidance is based on best practices gained over years of supporting ISVs' successful delivery of SaaS solutions to their customers. Evaluating this guidance in the context of your organization's goals and architectural principles will help you find ways to optimize your solution.

> **Note**
>
> This document does not provide an exhaustive list of best practices. It supplements the document Applying the AWS Well-Architected Framework for Amazon Neptune by

providing additional specific guidance for multi-tenancy ISV workloads. We recommend reviewing the considerations in both documents when designing your solution.

SaaS data-partitioning models

One of the challenges for SaaS developers is to design architectural patterns for representing and organizing data in a multi-tenant environment. These multi-tenant storage mechanisms and patterns are typically referred to as data partitioning.

In a multi-tenant SaaS environment, it's important to distinguish between data partitioning and tenant isolation. These concepts, while related, are not synonymous. Data partitioning refers to the method of storing data for each tenant. However, partitioning alone does not guarantee tenant isolation. Additional measures are necessary to ensure that the data of one tenant remains inaccessible to another.

The three common data-partitioning models in multi-tenant SaaS systems are silo, pool, and hybrid. Your choice of any model depends on factors such as the following:

- Compliance
- Noisy neighbors
- Tiering strategy
- Operational requirements
- Tenant-isolation needs

Additionally, each database type available on AWS typically offers a unique collection of data partitioning and tenant-isolation models. When looking at how tenant graphs can be organized to support the various needs of your solution, consider the models that Amazon Neptune provides.

Many ISVs start their design on Neptune with one of the following assertions:

- The ISV solution requires physical separation of customers across separate clusters.
- The ISV solution requires constructs such as named databases or schemas found in traditional relational database management systems.

After consideration, ISVs realize that these assertions aren't true because, under almost all workloads, each of their customers has a disconnected graph in their database. Implementing the data modeling and access guidance discussed in this document prevents those data boundaries from being crossed and maintains customer data privacy.

This guide describes both the silo model and the pool model, but most ISVs choose the pool model for cost and operational efficiency. The guide briefly discusses a hybrid model that combines aspects of both silo and pool models. Some ISVs use a hybrid model for their largest customers to accommodate regulatory or compliance requirements of the size of graph.

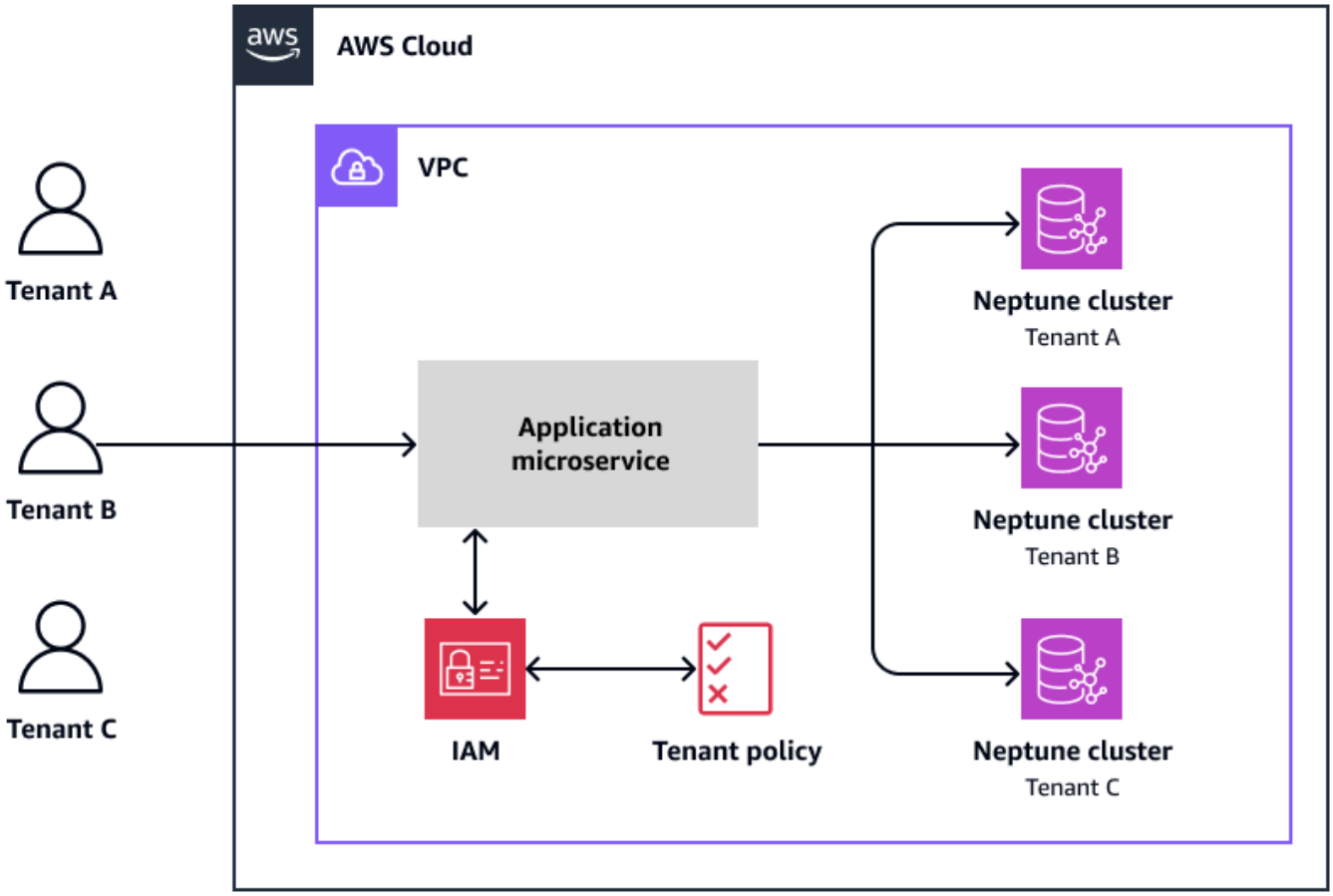
Silo-model multi-tenancy

Some multi-tenant SaaS environments might require tenants' data to be deployed on fully separated resources because of compliance and regulatory requirements. In some cases, large customers require dedicated clusters to reduce noisy neighbor impact. In those situations, you can apply the silo model.

In the silo model, storage of tenant data is fully isolated from any other tenant data. All constructs that are used to represent the tenant's data are considered physically unique to that client, meaning that each tenant will generally have distinct storage, monitoring, and management. Each tenant will also have a separate AWS Key Management Service (AWS KMS) key for encryption. In Amazon Neptune, a silo is one cluster per tenant.

Cluster per tenant

You can implement a silo model with Neptune by having one tenant per cluster. The following diagram shows three tenants accessing an application microservice in a virtual private cloud (VPC), with a separate cluster for each tenant.



Each cluster has its individual endpoint to help ensure distinct access points for efficient data interaction and management. By placing each tenant in its own cluster, you create the well-defined boundary between tenants ensuring customers that their data is successfully isolated from other tenants' data. This isolation is also appealing for SaaS solutions that have strict regulatory and security constraints. Additionally, when each tenant having its own cluster you don't have to worry about noisy neighbor, where one tenant imposes a load that could adversely affect the experience of other tenants.

While the cluster-per-tenant silo model has advantages, it also introduces management and agility challenges. The distributed nature of this model makes it harder to aggregate and assess tenant activity and the operational health across all tenants. Deployment also becomes more challenging because setting up a new tenant now requires the provisioning of a separate cluster. Upgrading becomes more challenging in environments with a shared client layer when client upgrades and versions are tightly coupled to the database upgrade.

Neptune supports both serverless and provisioned clusters. Assess whether your application workload is better handled by serverless or provisioned instances. In general, if your workload has a

constant level of demand, provisioned instances will be more cost effective. Serverless is optimized for demanding, highly variable workloads with heavy database usage for short periods of time followed by long periods of light activity or no activity.

When using a Neptune provisioned cluster per tenant, you must select an instance size that approximates the maximum load of your tenant's demand. This dependence on a server also has a cascading impact on the scaling efficiency and cost of your SaaS environment. While a goal of SaaS is to size dynamically based on actual tenant load, a Neptune provisioned cluster requires you to over-provision to account for heavier periods of usage and spikes in loads. Over-provisioning increases the cost per tenant. Additionally, as tenant usage changes over time, scaling up or scaling down the cluster must be applied separately for each tenant.

The Neptune team generally advises against a silo model because of the higher cost incurred by idle resources and the additional operational complexities. However, for highly regulated or sensitive workloads require this additional isolation, customers might be willing to pay the additional cost.

Implementation guidance for the silo model

To implement a cluster-per-tenant silo-isolation model, create AWS Identity and Access Management (IAM) data-access policies. These policies control access to tenants' Neptune clusters by ensuring that tenants can access only the Neptune cluster containing their own data. Attach the IAM policy for each tenant to an IAM role. The application microservice then uses the IAM role to generate fine-grained temporary credentials using the `AssumeRole` method of AWS Security Token Service (AWS STS). These credentials, which have access only to the Neptune cluster for that tenant, are used to connect to the tenant's Neptune cluster.

The following code snippet shows a sample data-based IAM policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:ReadDataViaQuery",
        "neptune-db:WriteDataViaQuery"
      ],
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:tenant-1-cluster/*",
      "Condition": {
```

```
        "ArnEquals": {
          "aws:PrincipalArn": "arn:aws:iam::123456789012:role/tenant-role-1"
        }
      }
    }
  ]
}
```

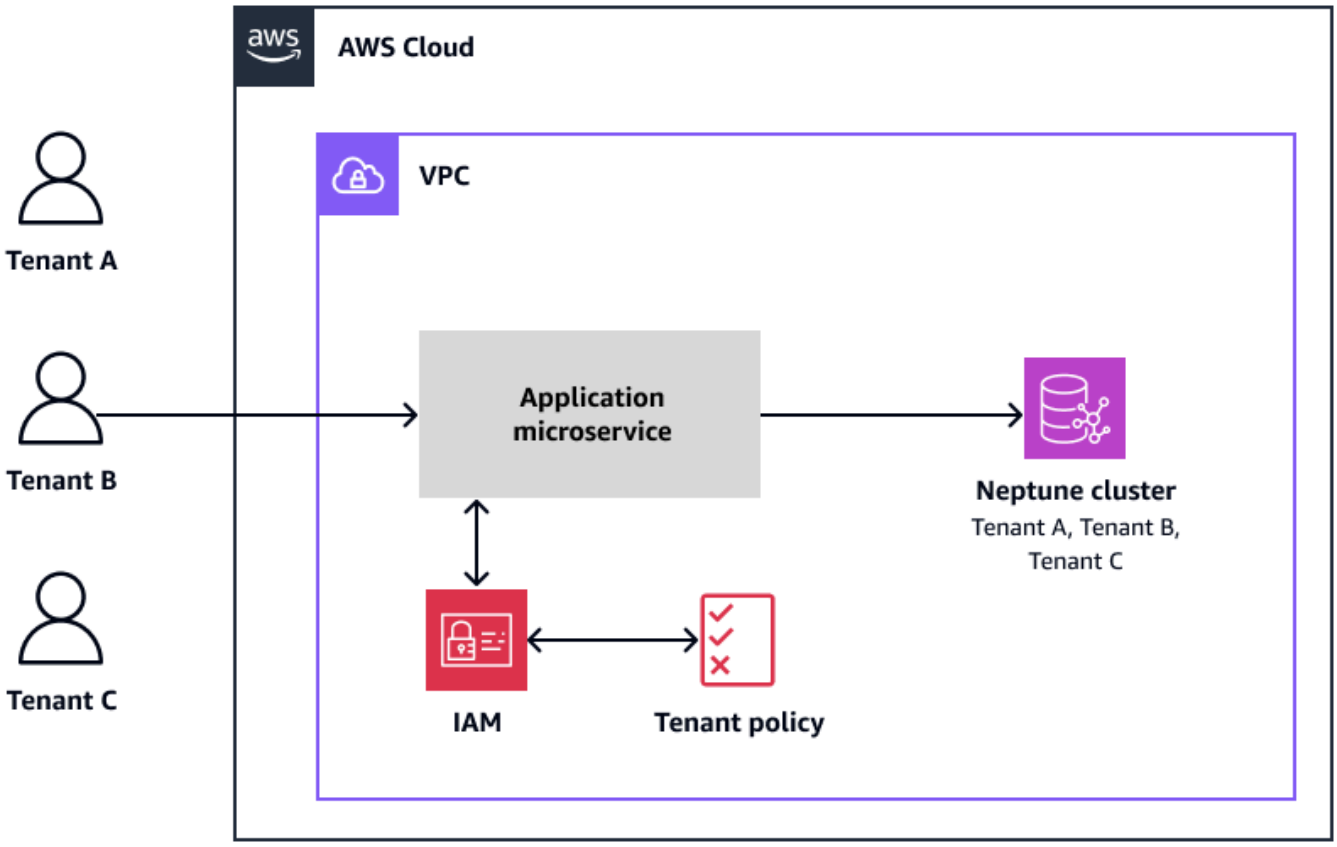
The code provides a sample tenant, `tenant-1`, with read and write query access to their respective Neptune cluster. The `Condition` element ensures that only the calling entity (the principal), which has assumed the `tentant-1` IAM role (`tenant-role-1`), is allowed to access `tenant-1`'s Neptune cluster.

Pool-model multi-tenancy

Sometimes it isn't necessary or feasible to implement the silo model because of cost or operational overhead:

- You might not have the resources to maintain an individual cluster per tenant.
- It might not be necessary to physically separate each tenant's data, and a logical separation is enough to meet their needs and compliance requirements.

The following diagram shows the pool model, with tenant data is placed in a single Amazon Neptune cluster, and all tenants share a common database.



This pool-isolation model reduces the management overhead and can improve the operational efficiency because there are fewer clusters to manage. Also, compute resources can be shared across multiple customers instead of remaining idle during customer inactive periods.

When you use the pool model, there are two ways to model data. Your approach depends on whether you're building a labeled property graph (LPG) or a graph with the Resource Description Framework (RDF).

Pool model for labeled property graphs

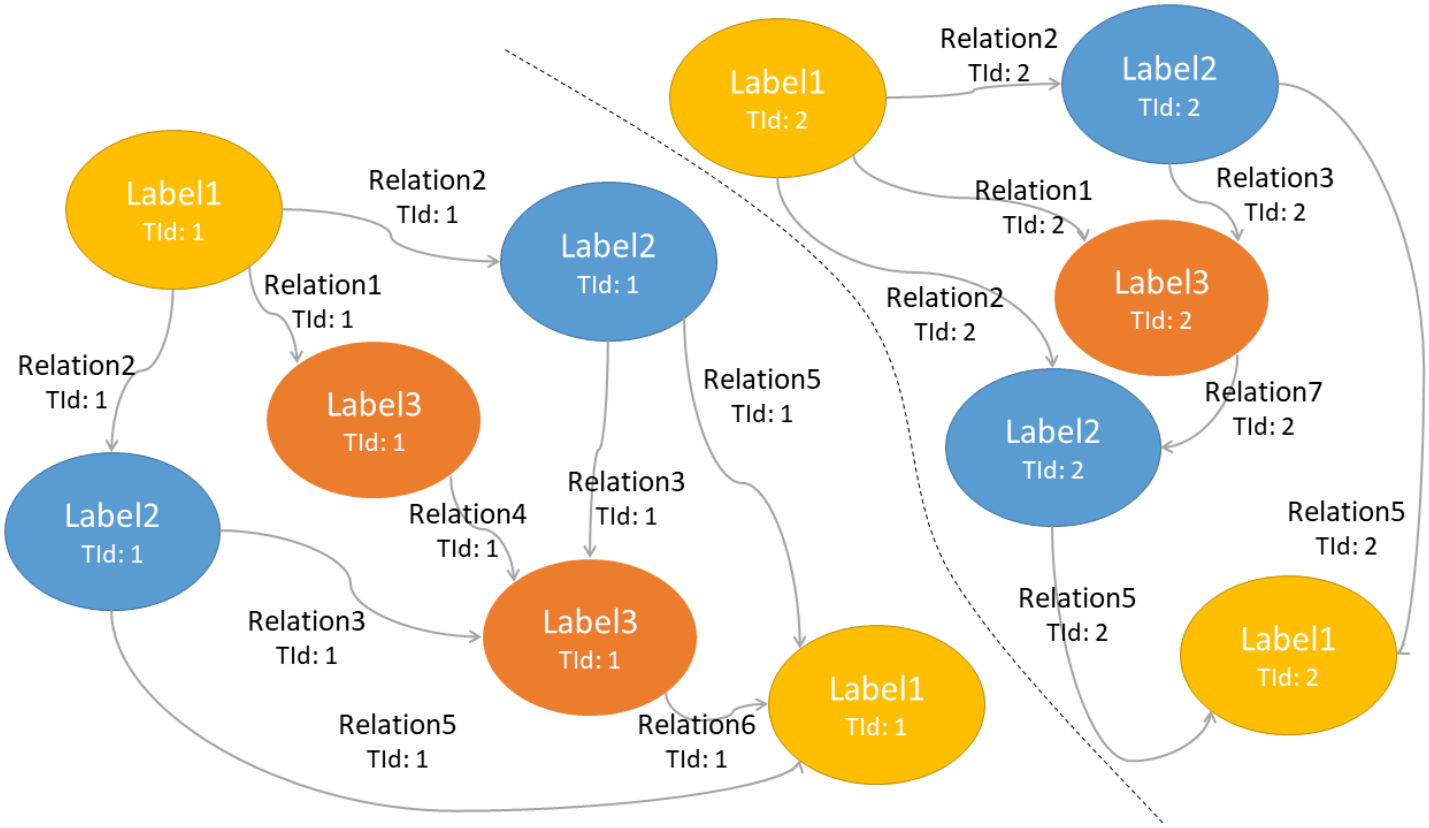
There are three different approaches to the pool model for LPGs on Amazon Neptune:

- **Property strategy** – Choose the property strategy when you need to prioritize use of established library constructs such as the Apache TinkerPop Gremlin language's PartitionStrategy over performance.
- **Prefix-label strategy** – We recommend the prefix-label strategy for most scenarios based on performance and limiting noisy neighbor effects.
- **Multiple-label strategy** – The multiple-label strategy has the improved performance of the prefix-label strategy. It also supports running queries that span all of the tenants on a cluster (for example, ISV queries for reporting or monitoring across all tenants).

Property strategy

With LPGs, users can add key-value pair properties to nodes, or vertices, and edges. To achieve logical separation, most customers intuitively model this as a unique property on every node and edge with a common *tenant property key*. The tenant property key represents the all the tenants that own the node. The *tenant identifier* is a unique value that identifies an individual tenant.

The following diagram shows this model. The two disconnected subgraphs have various labeled nodes and edges, with the tenant property key represented by `TId`. Every node and edge from one subgraph has a `TId` value of 1. In the other subgraph, every node and edge has a `TId` value of 2.



Within labeled property graphs, there are two ways to manage this. The Gremlin query language offers the [PartitionStrategy](#) traversal library to help manage data partitioning of the data. The code in the following example expects every node and edge to have a property called `TId`:

```
strategy1 = new PartitionStrategy(partitionKey: "TId", writePartition: "1",
 readPartitions: ["1"])
strategy2 = new PartitionStrategy(partitionKey: "TId", writePartition: "2",
 readPartitions: ["2"])
```

When new nodes or edges are written, the property `"TId"` is added with a value of `"1"` or `"2"`, depending on whether `strategy1` or `strategy2` was selected. For the customer with `"TId"` of `"1"`, you use `strategy1`. The following example shows writing data for that customer:

```
g.withStrategies(strategy1).addV("Label1").property("Value", "123456").property(id,
 "Item_1")
```

For read queries, a filter for `"TId == '1'"` or `"TId == '2'"` is added to every node or edge traversal by using `strategy1` or `strategy2`, respectively. These partition strategies simplify your code, but they aren't necessary. The benefit of using the strategy is that it can be injected at an

authorization level and passed to the lower-level code that forms the query. This separates the code that determines the customer identifier (`TId`) from the logic of the query.

The following example code shows a Gremlin query to read data:

```
g.withStrategies(strategy1).V().hasLabel("Label1")
```

The preceding code is equivalent to the following example:

```
g.V().hasLabel("Label1").has("TId", "1")
```

Likewise, when writing data by using Gremlin, you can use the following query:

```
g.withStrategies(strategy1).addV("Label1").property("Value").property(id, "Item_1")
```

The preceding code is equivalent to the following example, which does not use the partition strategy and therefore requires the `"TId"` property to be explicitly written:

```
g.addV("Label1").property("TId", "1").property("Value").property(id, "Item_1")
```

In openCypher, these libraries do not exist. You are responsible for writing and modifying your queries to add the tenant identifier as a property on nodes and edges. For example:

```
CREATE (n:Item {`~id`: 'Item_1', Value: '123456', TId: '1'})
CREATE (n:Item {`~id`: 'Item_2', Value: '123456', TId: '2'})
```

Note the similarity between the Gremlin code without the partition strategy. You can then read the node written from the first `CREATE` statement by using the following code:

```
MATCH (n:Item {TId: '1'})
RETURN n
--or
MATCH (n:Item)
WHERE n.TId == '1'
RETURN n
```

You might choose the property strategy when you want to use native TinkerPop Gremlin constructs such as PartitionStrategy. However, this model has performance drawbacks on Amazon Neptune

compared with the prefix-label strategy. For a discussion of these performance drawbacks, see the Performance implications for the LPG models section.

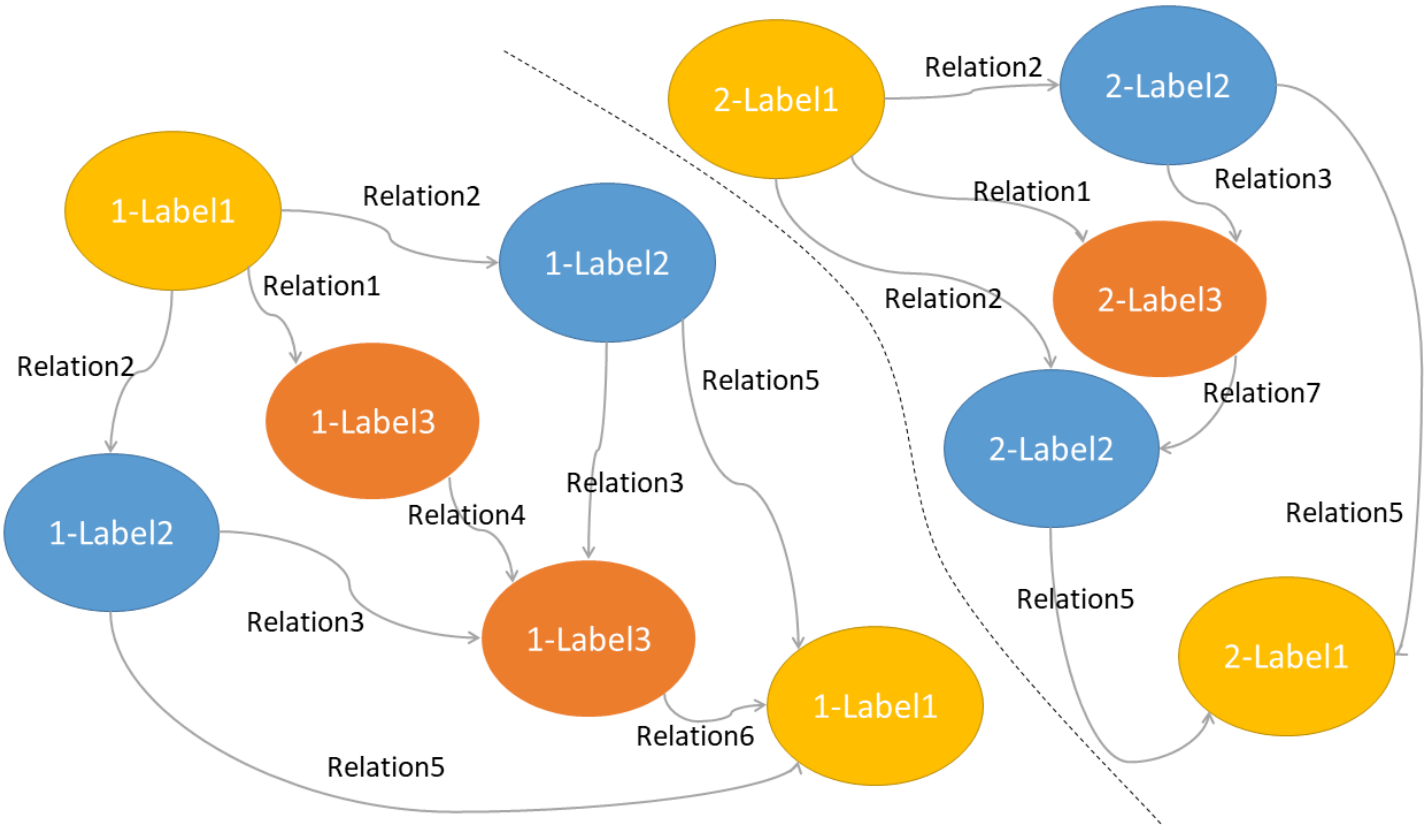
If the following conditions apply, consider modeling the property strategy only on nodes, not on edges:

- Your graph has significantly more edges than labels.
- Each tenant is a disconnected graph.
- You access the graph only by using nodes as a starting point, not labels.

Prefix-label strategy

If performance is a top concern, we highly recommend considering the prefix-label strategy over the property strategy.

In the prefix-label strategy, you label each node with a combination of tenant identifier and node label. For example, if the tenant has an identifier of `"1"` and the node label is `"Label1"`, you specify the node label as `"1-Label1"`. The following diagram shows two disconnected subgraphs that use this model.



When writing data in Gremlin, you can add an identifying number to any node's label:

```
g.addV("1-Label1")
g.addV("2-Label6")
```

When querying this graph, you can check for the existence of this prefix on a node:

```
g.V().hasLabel("1-Label1")
```

In openCypher you can write data by using a CREATE statement:

```
CREATE (n:`1-Label1` {`~id`: 'Item_1', Value: 'XYZ123456'})
```

To query the data that you wrote in openCypher, use the following code:

```
MATCH n= (:`1-Label1`)
RETURN n
```

The prefix-label strategy assumes that all nodes are assigned to one or more tenants and that permissions are not assigned at the edge scope. Avoid using this strategy on edge labels, because that will cause a large number of predicates and will negatively impact Neptune performance.

There are two primary drawbacks to the prefix label approach. First, it's difficult to run any queries that span across tenants. An example is a query that counts all nodes of a given label for reporting or monitoring. If this is your use case, consider combining this strategy with the multiple-label strategy. For more information about combining strategies, see the Hybrid model section.

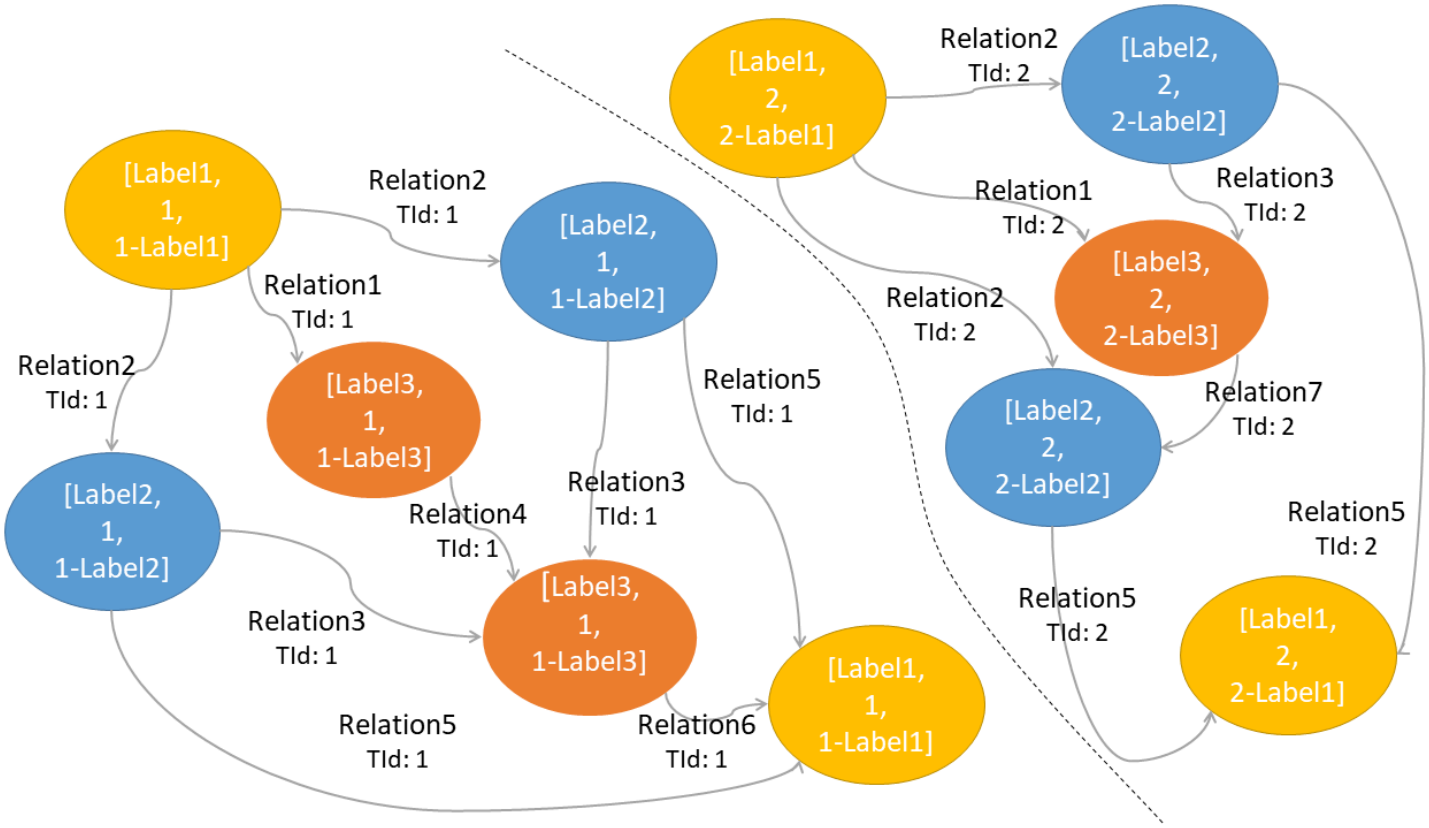
Second, the prefix-label strategy requires controls that enforce proper application of the appropriate prefix to every query to prevent data leakage. However, this strategy is the most efficient option for workloads that require low latency queries, and we highly recommend it. The Performance implications for LPG models section provides examples of why this is the most efficient strategy.

Multiple-label strategy

The third option is to use a multiple-label strategy. For this approach, you add extra labels to every node on the graph. For example, if you need to filter across all of the data for a given tenant, add

the tenant ID label. If you need to filter across all data for a given label regardless of tenant, add that label. The following diagram shows the multiple-label strategy applied by using three labels for each node.

You can now access the graph by using three different patterns:



- Filter on `Label1` to return all nodes with `Label1` across all tenants.
- Filter on `1` to return all nodes for tenant 1.
- Filter on `1-Label1` to return all nodes for only tenant 1 with label `Label1`.

For LPGs, there are two ways to implement this.

In Gremlin, you can use the traversal strategy called SubgraphStrategy to limit the scope of all queries to only vertices with a specific label, such as `"Label1"`:

```
g.withStrategies(
    new SubgraphStrategy(
        vertices=hasLabel("Label1")
    )
)
```

Unlike PartitionStrategy, SubgraphStrategy impacts reading data only, not writing data. To write the data, manually assign the labels in each query:

```
g.addV("Label1").property("Value","XYZ123456")
.addV("Label2").property("Value","XYZ123456")
```

When reading the data, you can use SubgraphStrategy to query all nodes with `"Label1"`:

```
g.withStrategies(
new SubgraphStrategy(vertices=.hasLabel("Label1"))
).
V().has("Value","XYZ123456")
```

Neptune returns only the first record, which has `"Label1"` and a value of `"XYZ123456"`. It's equivalent to the following query, which doesn't use SubgraphStrategy:

```
g.V().hasLabel("Label1").hasValue("XYZ123456")
```

In this basic query, it appears that SubgraphStrategy is more complex to use. Keep in mind that your libraries can provide an instance of g with the strategy already defined. Developers don't have to ensure that the proper filters are applied:

```
def getGraphTraversal():
return g.withStrategies(new SubgraphStrategy(vertices=.hasLabel("Label1"))

getGraphTraversal().has("Value","XYZ123456")
```

The openCypher libraries don't have these constructs, so you must create multiple labels for each node:

```
CREATE (n:`1`:`Label1`:`1-Label1` {`~id`: 'Item_1', Value: '12345'})
```

When you use these labels to filter for a subgraph, you can return nodes that have the customer label you are looking for or that share a relationship with another node that has that label:

```
MATCH n=(:Label1:`1`)
// or
MATCH n=(:`1-Label1`)
```

The multiple-label strategy gives you the most flexibility to query nodes by type (`Label1`) or tenant (1), or to use the more efficient prefix-label strategy when performance is of most importance (`1-Label1`).

The major drawback to this strategy is that each label is an extra object stored in your graph. An object is a node, edge, or a property on a node or edge in LPGs. Ingestion speed is measured and bound by objects per second, and storage costs depend on the number of gigabytes consumed. This means that extra objects might have a measurable impact at large scale.

Performance implications for the LPG models

The AWS Skill Builder course Data Modeling for Amazon Neptune describes in depth the Neptune data model internals and modeling implications, but we will summarize the important considerations for these designs here. Consider having three tenants (T1, T2, T3) on a single Neptune cluster. These tenants have the following attributes:

- Tenant 1 (T1) has 100 million nodes total, and 10 million are of type Item.
- Tenant 2 (T2) has 10 million nodes total, and 1 million are of type Item.
- Tenant 3 (T3) has 100 million nodes total, and 1 million are of type Item.

Run a query that will retrieve the items for Tenant 3 by using the property strategy. Neptune inspects the statistics for two index calls:

- Where `tenant property key=T3` has 100 million results
- Where `label = Item` has 12 million results (10 million from T1 + 1 million from T2 + 1 million from T3)

The Neptune query optimizer determines that the latter query is best applied first (12 million results) and then inspects each item for `tenant property key=T3`. You retrieve 12 million items to find the 1 million results.

Notice the noisy neighbor impact of this query. If you had 100 million Item nodes per tenant, the first query would have 300 million results instead of 12 million (This is overly simplified for illustrative purposes. The Neptune optimizer might have applied a different order of operations).

Next, consider the prefix-label strategy. Make a single index call where `label=T3-Item`, which returns 1 million results. This accomplishes the same result as the property strategy, but it

retrieves 11 million fewer records. In addition, you no longer have noisy neighbor concerns because the label doesn't overlap in the index.

The multiple-label strategy doesn't provide query performance improvement over the property strategy directly. Filtering by property value is comparable to filtering by label value when the search space is also comparable. Instead, the multiple-label strategy supports more flexibility. The multiple-label strategy provides performance equivalent to the prefix-label strategy for `label=T3` or the label `T3-Item`. The multiple-label strategy provides performance equivalent to the property strategy for `label=Item`. The benefit is to support a variety of access patterns.

Pool model for RDF

The Resource Description Framework (RDF) has a concept of named graphs, which provides a logical way of separating data. In Amazon Neptune, you have a default named graph and user-defined named graphs. You can create as many named graphs as you want. Collectively, they are called the RDF dataset. All named graphs, default or user-defined, are defined by an Internationalized Resource Identifier(IRI) within the RDF dataset. In Neptune, unless a user declared a named graph when writing data, all triples are considered part of the default named graph.

There are multiple use cases for named graphs:

- Data partitioning and data isolation
- Data provenance
- Versioning
- Inference

This guide focuses on the data-partitioning use case. We recommend creating one user-defined named graph for each tenant.

SPARQL query options using Graph Store HTTP Protocol

The following example queries use SPARQL Protocol and RDF Query Language (SPARQL) and Graph Store HTTP Protocol to query or create a named graph for a tenant.

- `HTTP GET` – To retrieve a specific graph of a tenant:

```
curl --request GET 'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//
www.example.com/named/tenant1'
```

- HTTP PUT – To create or replace a specific named graph with a payload specified in the request:

```
curl --request PUT -H "Content-Type: text/turtle" \  --data-raw "@prefix ex: http://
example.com/ . ex:subject ex:predicate ex:object ." \
'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//www.example.com/named/
tenant1'
```

  In RDF, an object is a triple.

- HTTP POST – To create a new named graph if one doesn't exist, or merge with an existing graph:

```
curl --request POST -H "Content-Type: text/turtle" \
--data-raw "@prefix ex: http://example.com/ . ex:subject ex:predicate ex:object ." \

'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//www.example.com/named/
tenant1'
```

Tenant isolation for RDF

For logical isolation of data with necessary guardrails in place at the application layer, create a mapping between the tenant and user-defined named graphs. When you design multi-tenancy for an RDF dataset, be aware of the following aspects of RDF and SPARQL:

- In Neptune, when you query without specifying a named graph, it retrieves all triples that match the pattern across all named graphs in the database.
- In RDF, there are no constraints around connections between nodes of different named graphs. For instance, in the previous diagram, a node in :G1 can be connected to a node in :G2 through an edge.

For example, if an end user of a particular tenant submits a query to the API, the API should validate the following requirements before it submits the query to the Neptune database:

- Any query scoped at a single tenant must specify a named graph. Otherwise, you risk leaking data across tenants.
- Update or Delete queries should always specify a named graph.
- Nodes on either side of an edge or relationship should always belong to the correct named graph.

For additional information about best practices, see the Neptune documentation.

Prepare for growth

When you use the pool model successfully, you eventually outgrow the size of a single Neptune cluster. Tenants grow, or the number of tenants grows, and the ingestion rate of data needed across all of your customers exceeds the cluster's capability. When this occurs, you will need to split your customers across multiple clusters. Design for this configuration upfront instead of trying to retrofit for it later. Even if your initial scale is to use only a single cluster, mock up the components you that will need to route tenants across multiple clusters in the future when you reach that scale.

If your solution requires more resources based on your tenant's size, prepare for their growth as well. If several of the customers on a single cluster grow significantly, that cluster might no longer support your requirements. Design a strategy to either move tenants to another cluster or split an existing cluster into two by using the Amazon Neptune DB cloning feature.

Be familiar with the Neptune Copy-on-Write Protocol, which can save you money when you implement DB cloning., If you split a cluster because of ingestion bottlenecks, it might be more efficient not to delete data from the clusters, providing your policies allow that. The two clusters will share a data page if it is unchanged but not if the data page was modified (because some data on it was deleted).

> **Note**
>
> This guidance applies to the most recent Neptune version at the time of this writing, which is Neptune version 1.3.1. This guidance might change in future versions as the Neptune storage layer evolves.

Limitations for multi-tenancy scenarios

Be aware that some Neptune features are not built for multi-tenancy scenarios. Tenants should not be given direct access to Neptune endpoints in a pool model because these multi-tenancy strategies aren't enforced at the database level. Always keep some kind of proxy between your customers and the Neptune endpoint that enforces the designs described in this document. Examples of such a proxy include the following:

- Appending the label filters in your client layer

- Having an API that maps the authentication token to a tenant ID and injects this filter into the query

This guidance also applies to giving customers direct access to features such as Neptune graph notebooks, Neptune graph-explorer, or Neptune Streams.

Hybrid model multi-tenancy

SaaS solutions often use a blend of silo and pool models. Various factors influence the decision of when and how to employ both silo and pool models within the same environment.

One such factor is tiering, where a SaaS solution offers unique experiences to each tier of tenants. For example, if your tiers are Free, Standard, and Premium, your Free tier tenant data could be stored in a shared Neptune cluster using a pool model. For your Standard and Premium tier tenants, you could use a cluster-per-tenant silo model.

Additionally, some SaaS providers have the capability to build their pool solution on a shared Amazon Neptune cluster as their foundation. Subsequently, they can create a separate Neptune cluster for tenants that require siloed storage, often because of compliance and regulatory mandates.

Although this can add a level of complexity to your data-access layer and management profile, it can also offer your business a way to tier your offering to fulfill customer requirements.

Operational best practices for ISVs

Many of the guidelines in this section are best practices for all customers, but they have added significance for ISVs.

Update your Neptune cluster with the newest versions

In the Amazon Neptune release notes, you can see that every version brings a number of bug fixes, performance improvements, and new features. Keep your Neptune clusters on the latest version as much as possible.

If you find a previously undiscovered bug in your workload and your cluster is on the latest version, Neptune engineers can create a private patch for your cluster (if warranted and you want it). The patch can bridge until the next release when that fix will be generally available. To help with updating your clusters to the newest version, use the Neptune Blue/Green solution.

Use deltas instead of delete and replace for data ingestion

You can use several techniques to ingest, or write, data to Neptune. Many customers try to simplify their data ingestion by deleting and reinserting their graph every time a change is received in the feed. They might add a `last-modified` property to each node and periodically scan for nodes that haven't been modified since some specified date and delete them. While these techniques simplify the data ingestion process, they have long-term health and scalability implications for your Neptune cluster.

First, Neptune uses dictionary encoding of strings. Unless you explicitly specify the IDs of nodes and edges, Neptune generates a GUID represented as a string for the ID and stores that string in the dictionary. If you are constantly deleting and adding objects, the automatically generated IDs will cause bloat in the dictionary.

Second, Neptune scales up to ingest about 120 K objects per second at the maximum. If you continuously delete and add objects, you consume a lot of that bandwidth on objects that essentially are not changing. This limits the number of tenants that you can host on a cluster, requires larger writer instances in the clusters, and requires more I/O operations. All of these factors increase your costs.

We highly recommend that you develop a way to calculate the true delta of what has changed instead of using the delete and add methods. However, some data sources aren't conducive

to this (for example, API calls that return the current state, or events that do not track exactly what changed). If your raw data source is not conducive to identifying changes, use your extract, transform, and load (ETL) processes to calculate the delta. For example, you can maintain snapshots from each previous data capture in Parquet format, use AWS Glue to calculate the differences between those snapshots, and push only the differences to Neptune.

Model how Neptune costs will evolve with your tenants

Whether you use a silo, pool, or hybrid model, your cloud costs will scale with your tenants' size. Tenants that require more concurrent connections need larger instances or more read replicas than those with fewer concurrent connections. The same applies to tenants that require more rapid data ingestion.

The three components of Neptune cluster cost are instance size (and number), data size (GB-months), and I/O operations (per million). While these costs are generally workload specific, they scale with size and data volume, they can be measured by using AWS tools. Track and understand the economies of scale against key indicators of your tenants' sizes, including how their sizes vary over time. If the unpredictability of your I/O charges impacts your margins, consider choosing Neptune I/O-Optimized storage for a more predictable cost.

Scale your clusters for customer demand

There is no tried or true formula for right-sizing your Neptune instance size. The Neptune documentation provides guidance, but there are too many variables to recommend a direct mapping. These variables include but aren't limited to the following:

- Data model
- Data shape
- Query concurrency
- Query complexity.

Plan testing to determine the optimal size for your workloads and tenant profiles. In general, we recommend using provisioned instances for cost efficiency and predictability. If your customer-experience goals prioritize optimal scaling over costs, consider using Neptune Serverless instances to ensure a more consistent experience regardless of workload fluctuations.

If your tenant read workloads have significant variability in their peaks and troughs, combine Neptune Serverless instances with Neptune auto-scaling. It usually takes 10-15 minutes for a new read replica to come online after it's initialized. This means that auto-scaling alone can handle prolonged changes in traffic, but it isn't sufficient for rapidly changing spikes in activity. By combining Neptune Serverless and Neptune auto-scaling, you can both scale instances up or down and scale the number of read replicas in and out.

If your tenants have significantly different workload profiles or service level agreements (SLAs), consider using custom endpoints and dedicated read replicas to direct traffic to instances that are optimized for that traffic. Optimization can include a different sizing of the instance, specific query patterns, or pre-warming the buffer cache.

Next steps

If you are just starting on your journey implementing Amazon Neptune for your multi-tenancy ISV application, put extra thought into your desired model. Changing the model will be more costly later in your journey.

If you are early in your journey, verify that you are using the best model for your needs and that you are following the guidance for that model.

Plan ahead. When you are early in your journey, it's tempting to defer work on sharding customers across clusters or optimizing your ETL processes to provide the delta of changes instead of deleting and re-adding vertices and edges. As you scale, those decisions might negatively impact performance and cost.

Finally, if you are already well into your journey, this guidance might reassure you that your architecture is optimal, or it might provide changes to improve your architecture.

If you have questions on this guidance or need further assistance, please reach out to your AWS account team and ask for a session with a Neptune specialist.

Resources

- Amazon Neptune documentation
- Data Modeling for Amazon Neptune (course)
- Applying the AWS Well-Architected Framework for Amazon Neptune
- SaaS Lens Well-Architected Framework
- Guidance for Multi-Tenant Architectures on AWS
- SaaS Tenant Isolation Strategies: Isolating Resources in a Multi-Tenant Environment
- Apache TinkerPop documentation
- SPARQL

Contributors

Contributors to this guide include:

- Brian O'Keefe, Principal WW SSA Neptune, AWS
- Veeresham Gande, Senior Technical Account Manager, AWS
- Dana Owens, Startup Solutions Architect, AWS
- Nima Seifi, Startup Solutions Architect, AWS

Document history

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

Change	Description	Date
Initial publication	—	September 3, 2024

AWS Prescriptive Guidance glossary

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

Numbers

7 Rs

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

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

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

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

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

active-passive migration

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

aggregate function

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

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

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

anti-pattern

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

application control

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

application portfolio

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

artificial intelligence (AI)

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

artificial intelligence operations (AIOps)

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

asymmetric encryption

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

atomicity, consistency, isolation, durability (ACID)

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

attribute-based access control (ABAC)

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

authoritative data source

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

Availability Zone

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

AWS Cloud Adoption Framework (AWS CAF)

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

AWS Workload Qualification Framework (AWS WQF)

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

B

bad bot

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

BCP

See business continuity planning.

behavior graph

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

big-endian system

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

binary classification

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

bloom filter

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

blue/green deployment

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

bot

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

botnet

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

branch

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

break-glass access

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

brownfield strategy

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

buffer cache

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

business capability

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

business continuity planning (BCP)

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

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

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

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

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

chaos engineering

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

CI/CD

See continuous integration and continuous delivery.

classification

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

client-side encryption

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

Cloud Center of Excellence (CCoE)

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

cloud computing

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

cloud operating model

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

cloud stages of adoption

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

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

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

CMDB

See configuration management database.

code repository

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

cold cache

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

cold data

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

computer vision (CV)

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

configuration drift

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

configuration management database (CMDB)

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

conformance pack

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

continuous integration and continuous delivery (CI/CD)

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

CV

See computer vision.

D

data at rest

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

data classification

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

data drift

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

data in transit

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

data mesh

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

data minimization

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

data perimeter

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

data preprocessing

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

data provenance

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

data subject

An individual whose data is being collected and processed.

data warehouse

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

database definition language (DDL)

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

database manipulation language (DML)

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

DDL

See database definition language.

deep ensemble

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

deep learning

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

defense-in-depth

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

delegated administrator

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

deployment

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

development environment

See environment.

detective control

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

development value stream mapping (DVSM)

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

digital twin

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

dimension table

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

disaster

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

disaster recovery (DR)

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

DML

See database manipulation language.

domain-driven design

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

DR

See disaster recovery.

drift detection

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

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

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

electronic data interchange (EDI)

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

encryption

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

encryption key

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

endianness

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

endpoint

See service endpoint.

endpoint service

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

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

enterprise resource planning (ERP)

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

envelope encryption

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

environment

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

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

epic

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

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

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

F

fact table

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

fail fast

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

fault isolation boundary

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

feature branch

See branch.

features

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

feature importance

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

feature transformation

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

few-shot prompting

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

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

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

flash-cut migration

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

FM

See foundation model.

foundation model (FM)

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

G

generative AI

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

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

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

Gitflow workflow

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

golden image

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

greenfield strategy

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

guardrail

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

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

H

HA

See high availability.

heterogeneous database migration

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

high availability (HA)

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

historian modernization

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

holdout data

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

homogeneous database migration

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

hot data

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

hotfix

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

hypercare period

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

I

IaC

See infrastructure as code.

identity-based policy

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

idle application

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

IIoT

See industrial Internet of Things.

immutable infrastructure

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

inbound (ingress) VPC

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

incremental migration

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

Industry 4.0

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

infrastructure

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

infrastructure as code (IaC)

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

industrial Internet of Things (IIoT)

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

inspection VPC

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

Internet of Things (IoT)

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

interpretability

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

IoT

See Internet of Things.

IT information library (ITIL)

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

IT service management (ITSM)

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

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

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

landing zone

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

large language model (LLM)

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

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

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

lift and shift

See 7 Rs.

little-endian system

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

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

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

main branch

See branch.

malware

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

managed services

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

manufacturing execution system (MES)

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

MAP

See Migration Acceleration Program.

mechanism

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

member account

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

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

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

microservice

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

microservices architecture

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

Migration Acceleration Program (MAP)

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

migration at scale

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

migration factory

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

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

migration metadata

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

migration pattern

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

Migration Portfolio Assessment (MPA)

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

Migration Readiness Assessment (MRA)

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

migration strategy

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

ML

See machine learning.

modernization

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

modernization readiness assessment

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

monolithic applications (monoliths)

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

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

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

mutable infrastructure

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

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

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

OI

See operations integration.

OLA

See operational-level agreement.

online migration

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

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

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

operational-level agreement (OLA)

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

operational readiness review (ORR)

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

operational technology (OT)

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

operations integration (OI)

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

organization trail

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

organizational change management (OCM)

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

origin access control (OAC)

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

origin access identity (OAI)

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

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

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

P

permissions boundary

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

personally identifiable information (PII)

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

PII

See personally identifiable information.

playbook

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

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

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

polyglot persistence

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

portfolio assessment

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

predicate

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

predicate pushdown

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

preventative control

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

principal

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

privacy by design

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

private hosted zones

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

proactive control

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

product lifecycle management (PLM)

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

production environment

See environment.

programmable logic controller (PLC)

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

prompt chaining

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

pseudonymization

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

publish/subscribe (pub/sub)

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

Q

query plan

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

query plan regression

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

R

RACI matrix

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

RAG

See Retrieval Augmented Generation.

ransomware

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

RASCI matrix

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

RCAC

See row and column access control.

read replica

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

re-architect

See 7 Rs.

recovery point objective (RPO)

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

recovery time objective (RTO)

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

refactor

See 7 Rs.

Region

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

regression

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

rehost

See 7 Rs.

release

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

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

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

resource-based policy

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

responsible, accountable, consulted, informed (RACI) matrix

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

responsive control

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

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

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

rotation

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

row and column access control (RCAC)

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

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

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

S

SAML 2.0

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

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

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

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

security by design

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

security control

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

security hardening

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

security information and event management (SIEM) system

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

security response automation

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

server-side encryption

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

service control policy (SCP)

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

service endpoint

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

service-level agreement (SLA)

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

service-level indicator (SLI)

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

service-level objective (SLO)

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

shared responsibility model

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

SIEM

See security information and event management system.

single point of failure (SPOF)

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

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

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

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

SPOF

See single point of failure.

star schema

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

strangler fig pattern

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

subnet

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

supervisory control and data acquisition (SCADA)

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

symmetric encryption

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

synthetic testing

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

system prompt

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

T

tags

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

target variable

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

task list

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

test environment

See environment.

training

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

transit gateway

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

trunk-based workflow

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

trusted access

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

tuning

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

two-pizza team

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

U

uncertainty

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

undifferentiated tasks

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

upper environments

See environment.

V

vacuuming

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

version control

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

VPC peering

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

vulnerability

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

W

warm cache

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

warm data

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

window function

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

workload

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

workstream

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

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

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

Z

zero-day exploit

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

zero-day vulnerability

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

zero-shot prompting

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

zombie application

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