Applying the AWS Well-Architected Framework for Amazon Neptune

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Applying the AWS Well-Architected Framework for Amazon Neptune

Table of Contents

Applying the AWS Well-Architected Framework for Amazon Neptune

Amazon Web Services (contributors)

January 2026 (document history)

You can build graph-based solutions on Amazon Web Services (AWS) by using Amazon Neptune. This guide provides prescriptive guidance for applying the AWS Well-Architected Framework principles when you plan your Neptune deployment.

The AWS Well-Architected Framework helps you build secure, high-performing, resilient, and efficient infrastructures for a variety of applications and workloads. It also provides a consistent approach for you to evaluate architectures and implement scalable designs.

The AWS Well-Architected Framework is built around the following six pillars:

- Operational excellence
- Security
- Reliability
- Performance efficiency
- Cost optimization
- Sustainability

This guide provides information from the AWS Well-Architected Framework design pillars and best practices, and considerations to keep in mind when deploying Neptune on AWS.

Intended audience

This guide is intended for data engineers, solution architects, and data analysts who design and implement solutions that use graphs on AWS.

Objectives

This guide can help you and your organization do the following:

- Choose from the supported deployment options and query languages, based on your use case and query patterns.
- Follow the AWS Well-Architected design patterns that will help in improving resiliency and security.
- Design your queries for optimal performance and cost savings.
- Learn how to be operationally efficient when managing your Neptune cluster in production.

Operational excellence pillar

The operational excellence pillar of the AWS Well-Architected Framework focuses on running and monitoring systems, and continually improving processes and procedures. It includes the ability to support development and run workloads effectively, gain insight into their operation, and continuously improve supporting processes and procedures to deliver business value. You can reduce operational complexity through self-healing workloads, which detect and remediate most issues without human intervention. You can work toward this goal by following the best practices described in this section. Use Amazon Neptune metrics, APIs, and mechanisms to properly respond when your workload deviates from expected behavior.

This discussion of the operational excellence pillar focuses on the following key areas:

- Infrastructure as code (IaC)
- Change management
- Resiliency strategies
- Incident management
- Audit reporting for compliance
- Logging and monitoring

Automate deployment using an IaC approach

Best practices for automating deployment on Neptune using IaC include the following:

- Apply infrastructure as- code (IaC) to deploy Neptune clusters whenever possible. For consistent environment configuration, use an AWS CloudFormation template, AWS Cloud Development Kit (AWS CDK), or HashiCorp Terraform to create all the required resources for your cluster.
- Automate Neptune operational procedures, such as resizing instances, adding or removing read replicas, or doing manual failovers on global tables, whenever possible.
- Store connection strings externally from your client. Use extract, transform, and load (ETL) processes to facilitate blue/green deployment strategies, disaster recovery (DR), and near-zero downtime migrations to new clusters. Connection strings can be stored in AWS Secrets Manager, Amazon DynamoDB, or any location where they can be changed dynamically.
- Use tags to add metadata to your Neptune resources, and track usage based on tags. For more information, see Tagging Amazon Neptune Resources.

Make frequent, small, reversible changes

The following recommendations focus on small, reversible changes to minimize complexity and reduce the likelihood of workload disruption:

- Store IaC templates and scripts in a source control service, such as GitHub or GitLab.

> **Important**
>
> Do not store AWS credentials in source control.

- Require IaC deployments to use a continuous integration and continuous delivery (CI/CD) service, such as AWS CodePipeline or AWS CodeBuild. These services compile, test, and deploy code in a non-production environment containing an ephemeral Neptune cluster before impacting your production Amazon Neptune cluster.
- Test infrastructure and application queries in a lower environment before you deploy them to production. This will minimize the likelihood of a disruption and help ensure they perform well with your workload and scale.

Anticipate failure

A self-healing infrastructure exemplifies operational excellence by anticipating failure and attempting to resolve any issues without intervention. The following recommendations help you achieve that maturity with Neptune:

- Create a monitoring plan that uses Amazon CloudWatch metrics to monitor your DB instance's CPU and memory usage, and understand the usage patterns. Create CloudWatch dashboards and alarms for key metrics and the Neptune client responses found in your application logs. For more information about indicators of high or low CPU utilization, see Using CloudWatch to monitor DB instance performance in Neptune in the Neptune documentation.

  If you frequently get out-of-memory exceptions on your queries, consider reducing the total number of nodes your query traverses or try using an instance from the X2 family, which has a higher RAM-to-CPU ratio.
- Set notifications to monitor the health of the Neptune cluster. For example, `BufferCacheHitRatio` should be constantly high (greater than 99.9 percent), whereas

`MainRequestQueuePendingRequests` should be constantly low (ideally 0 but dependent on your requirements and latency tolerance).

- Consider using read replicas to achieve high availability within Neptune. You should have at least two read replicas in different Availability Zones from the writer instance to ensure an instance is always available to serve read queries during a failover event.
- Automatically scale read replicas based on utilization metrics. For more information, see Auto-scaling the number of replicas in an Amazon Neptune DB cluster.
- Test failover for your DB instance to understand how long the process takes for your use case.
- If your application requires surviving a complete AWS Region outage, consider using global databases as part of your DR plans.

Learn from all operational failures

A self-healing infrastructure is a long-term effort that develops in iterations as rare problems occur or responses are not as effective as desired. Adopting the following practices drives focus toward that goal:

- Drive improvement by learning from all failures.
- Share what is learned across teams and the organization. If multiple teams within an organization use Neptune, create a common chatroom or user group to share learnings and best practices.

Use logging capabilities to monitor for unauthorized or anomalous activity

To observe anomalous performance and activity patterns, store logs in Amazon CloudWatch Logs. Consider the following best practices:

- Enable slow-query logging. Regularly review the log and diagnose why certain queries are slow. Use the Neptune explain and profile endpoints for Gremlin, SPARQL, or openCypher to gain insights into why these queries are slow.
- Enable Neptune audit logs, and regularly review the logs for unauthorized access or anomalies.
- If you are using slow-query logging or audit logging, enable publishing to CloudWatch Logs. This will help you to avoid running out of disk space on instances. Neptune instances have limited log

storage capacity and will overwrite older log files when log space is exceeded. CloudWatch Logs supports long-term retention of logs. The enhanced monitoring capabilities in CloudWatch Logs will improve your ability to query logs and diagnose issues.

- To facilitate better analysis tools for your audit logs, you can configure a Neptune DB cluster to publish audit log data to a log group in CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, use CloudWatch to create alarms and view metrics, and use CloudWatch Logs to store your log records in highly durable storage. For more information, see Publishing Neptune logs to Amazon CloudWatch Logs.
- Neptune supports logging of control plane actions using AWS CloudTrail. For more information, see Logging Amazon Neptune API Calls with AWS CloudTrail.

Security pillar

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of AWS security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon Neptune, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors, including the sensitivity of your data, your company's requirements, and applicable laws and regulations. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post.

The security pillar helps you understand how to apply the shared responsibility model when using Neptune. The following topics show you how to configure Neptune to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Neptune resources.

The security pillar includes the following key focus areas:

- Data security
- Network security
- Authentication and authorization

Implement data security

Data leakage and breaches put your customers at risk and can cause substantial negative impact on your company. The following best practices help protect your customer data from inadvertent and malicious exposure:

- Cluster names, tags, parameter groups, AWS Identity and Access Management (IAM) roles, and other metadata should not contain confidential or sensitive information because that data might appear in billing or diagnostic logs.
- URIs or links to external servers stored as data in Neptune should not contain credential information to validate requests.
- Neptune encrypted instances provide an additional layer of data protection by helping to secure your data from unauthorized access to the underlying storage. You can use Neptune encryption to increase data protection of your applications that are deployed in the cloud. You can also Neptune encryption to fulfill compliance requirements for data at rest.

  To enable encryption for a new Neptune DB instance, choose **Yes** in the **Enable encryption** section on the Neptune console (selected by default), or by setting the AWS::Neptune::DBCluster::StorageEncrypted property in CloudFormation. If encryption is enabled, Neptune will use the Amazon Relational Database Service (Amazon RDS) AWS managed key by default, or you can create a customer managed key. For information about creating a Neptune DB instance, see Creating a new Neptune DB cluster. For more details, see Encrypting Neptune Resources at Rest. Your automated and manual snapshots use the same encryption that you selected for your Neptune cluster.
- When using SPARQL and openCypher languages, practice proper input validation and parameterization techniques to prevent SQL injection and other forms of attacks. Avoid constructing queries that use string concatenation with user-supplied input. Use parameterized queries or prepared statements to safely pass input parameters to the graph database. For more information, see Examples of openCypher parameterized queries and SPARQL Injection Defence.
- For the Gremlin language, use Gremlin Language Variants instead of directly passing string-based Gremlin Scripts to avoid potential injection issues.

Secure your networks

An Amazon Neptune DB cluster can be created only in a virtual private cloud (VPC) on AWS. Until Neptune 1.4.6.0, the Neptune DB cluster's endpoints were accessible only within that VPC. As of Neptune 1.4.6.0 and later, Neptune instances can be configured to be publicly accessible over the internet. It is a best practice to use this feature only in non-production environments to enable simplified access to Neptune for your developers (though IAM authentication is always required on to enable public accessibility). If you have public accessibility enabled, consider setting inbound security group rules for your database port to only known IP address traffic. In production environments or with clusters containing sensitive data, secure your Neptune data by not allowing

public accessibility and limiting access to the VPC where your Neptune DB cluster is located. For more information, see Connecting to your Amazon Neptune graph.

To protect your data in transit, Neptune enforces SSL connections through HTTPS to any instance or cluster endpoint using secure protocols and ciphers. Neptune provides SSL certificates for your Neptune DB instances. Neptune SSL certificates support only cluster endpoint, reader endpoint, and instance endpoint hostnames.

If you are using a load balancer or a proxy server (such as HAProxy), you must use SSL termination and have your own SSL certificate on the proxy server. SSL passthrough doesn't work because the provided SSL certificates don't match the proxy server hostname. For more information about connecting to Neptune endpoints with SSL, see Using the HTTP REST endpoint to connect to a Neptune DB instance.

Implement authentication and authorization

To control who can perform Neptune management actions on Neptune DB clusters and DB instances, enable IAM database authentication and use IAM credentials. When you connect to AWS using IAM credentials, your IAM role must have IAM policies that grant the permissions required to perform Neptune management operations. Ensure that you follow principle of least privilege, granting only the permissions required to complete a task. For more information, see Using different kinds of IAM policies for controlling access to Neptune and IAM Authentication Using Temporary Credentials.

To control who can connect to a Neptune cluster and query the data, you can use IAM to authenticate to your Neptune DB instance or DB cluster. If you enable IAM authentication in a Neptune DB cluster, anyone accessing the DB cluster must first be authenticated. For more information, see Enabling IAM database authentication in Neptune for steps to enable IAM authentication.

When IAM database authentication is enabled, each request must be signed using AWS Signature Version 4. To understand how to send signed requests to all Neptune endpoints with IAM authentication enabled, see Connecting and Signing with AWS Signature Version 4. Many libraries and tools, such as awscurl, already support AWS Signature Version 4.

For interacting with other AWS services, Amazon Neptune uses IAM service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Neptune. Service-linked roles are predefined by Neptune and include all the permissions that the service requires to call other AWS services on your behalf. For more information, see Using Service-Linked Roles for Neptune.

Reliability pillar

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This includes the ability to operate and test the workload through its total lifecycle.

A reliable workload starts with upfront design decisions for both software and infrastructure. Your architecture choices will impact your workload behavior across all of the AWS Well-Architected pillars. For reliability, there are specific patterns you must follow.

The reliability pillar focuses on the following key areas:

- Workload architecture, including service quotas and deployment patterns
- Change management
- Failure management

Understand Neptune service quotas

A Neptune cluster volume can grow to a maximum size of 128 tebibytes (TiB) in all supported AWS Regions except China and GovCloud, where the quota is 64 TiB.

The 128 TiB quota is sufficient to store approximately 200-400 billion objects in the graph. In a labeled property graph (LPG), an object is a node, an edge, or a property on a node or edge. In a Resource Description Framework (RDF) graph, an object is a quad.

For any Neptune Serverless cluster, you set both the minimum and the maximum number of Neptune Capacity Units (NCUs). Each NCU consists of 2 gibibytes (GiB) of memory and the associated vCPU and networking. The minimum and maximum NCU values apply to any serverless instances in the cluster. The highest maximum NCU value you can set is 128.0 NCUs, and the lowest minimum is 1.0 NCUs. Optimize the NCU range that works best for your application by observing the Amazon CloudWatch metrics `ServerlessDatabaseCapacity` and `NCUUtilization` to capture the range you commonly run in and correlate undesired behavior or costs within that range. In many workloads, 1.0 NCU is too low of a starting point and results in unreliable behavior after periods of inactivity. If you find that your workload doesn't scale fast enough, increase the minimum NCUs to provide enough processing for the initial surge while it scales.

Each AWS account has quotas for each Region on the number of database resources that you can create. These resources include DB instances and DB clusters. After you reach a limit for a resource,

additional calls to create that resource fail with an exception. Some quotas are soft quotas that can be increased by request. For a list of quotas shared between Amazon Neptune and Amazon RDS, Amazon Aurora, and Amazon DocumentDB (with MongoDB compatibility), along with links to request quota increases when available, see Quotas in Amazon RDS.

Understand Neptune deployment patterns

In Neptune DB clusters, there is one primary DB instance and up to 15 Neptune replicas. The primary DB instance supports read and write operations, and it performs all of the data modifications to the cluster volume. Neptune replicas connect to the same storage volume as the primary DB instance, and they support only read operations. Neptune replicas can offload read workloads from the primary DB instance.

To achieve high availability, use read replicas. Having one or more read replica instances available in different Availability Zones can increase availability because read replicas serve as failover targets for the primary instance. If the writer instance fails, Neptune promotes a read replica instance to become the primary instance. When this happens, there is a brief interruption (generally less than 30 seconds) while the promoted instance is rebooted, during which read and write requests made to the primary instance fail with an exception. For highest reliability, consider two read replicas in different Availability Zones. If the primary instance in Availability Zone 1 goes offline, the instance in Availability Zone 2 is promoted to primary, but it cannot handle queries while that happens. So an instance in Availability Zone 3 is required to handle read queries during the transition.

If you are using Neptune Serverless, reader and writer instances in all Availability Zones will scale up and down, independently of each other, depending on their database load. You can set the promotion tier of a reader instance to 0 or 1 so that it scales up and down along with the capacity of the writer instance. This makes it ready to take over the current workload at any time.

If your application has a worldwide footprint or requires multi-Region failover, consider using a Neptune global database. An Amazon Neptune global database spans multiple AWS Regions, enabling low-latency global reads and providing fast recovery in the rare case where an outage affects an entire AWS Region. A Neptune global database consists of a primary DB cluster in one Region and up to five secondary DB clusters in different Regions.

Manage and scale Neptune clusters

You can use Neptune auto-scaling to automatically adjust the number of Neptune replicas in a DB cluster to meet your connectivity and workload requirements based on CPU utilization thresholds. With auto-scaling, your Neptune DB cluster can handle sudden increases in workload. When the workload decreases, auto-scaling removes unnecessary replicas so that you aren't paying for unused capacity. Be aware that new instance startup can take as long as 15 minutes, so auto-scaling alone is not a sufficient solution for rapid changes in demand.

You can use auto-scaling only with a Neptune DB cluster that already has one primary writer instance and at least one read-replica instance (see Amazon Neptune DB Clusters and Instances). Also, all read-replica instances in the cluster must be in an available state. If any read-replica is in a state other than available, Neptune auto-scaling does nothing until every read-replica in the cluster is available.

If you experience rapid changes in demand, consider using serverless instances. The serverless instances can scale vertically over short periods while auto-scaling scales horizontally over longer periods. This configuration provides optimal scalability because the serverless instances scale vertically while auto-scaling instantiates new read replicas to handle the workload beyond the maximum capacity of a single serverless instance. For more information about capacity scaling of Amazon Neptune Serverless, see Capacity scaling in a Neptune Serverless DB cluster.

If your scaling needs change at predictable times, you can schedule changes to the minimum instances, maximum instances, and thresholds to better handle those shifting needs. Remember to schedule scale-out events at least 15 minutes in advance to allow for those instances to come online when needed.

You manage your database configuration in Amazon Neptune by using parameters in a parameter group. Parameter groups act as a *container* for engine configuration values that are applied to one or more DB instances. When modifying cluster parameters in parameter groups, understand the difference between static and dynamic parameters, and how and when they are applied. Use the status endpoint to see the current applied configuration.

Manage backups and failover events

Neptune backs up your cluster volume automatically, and it retains backed up data for the length of the *backup retention period*. Neptune backups are continuous and incremental so you can

quickly restore to any point within the backup retention period. You can specify a 1–35 day backup retention period when you create or modify a DB cluster.

To retain a backup beyond the backup retention period, you can also take a snapshot of the data in your cluster volume. Storing snapshots incurs the standard storage charges for Neptune.

When you create an Amazon Neptune snapshot of a DB cluster, Neptune creates a storage volume snapshot of the cluster, backing up all its data, not just individual instances. You can later create a new DB cluster by restoring from this DB cluster snapshot. When you restore the DB cluster, you provide the name of the DB cluster snapshot to restore from, and then you provide a name for the new DB cluster that is created by the restore.

Test how your system responds to failover events. Use the Neptune API to force a failover event. Reboot with failover is beneficial when you want to simulate a failure of a DB instance for testing or for restore operations to the original Availability Zone after a failover occurs. For more information, see Configuring and managing a Multi-AZ deployment. When you reboot a DB writer instance, it fails over to the standby replica. Rebooting a Neptune replica does not initiate a failover.

Design your clients for reliability. Test their behavior during failover events. Implement retry logic in your client with exponential backoff logic. Code examples that implement this logic can be found in the documentation under the AWS Lambda function examples for Amazon Neptune.

Consider using AWS Backup if you have a common set of backup requirements you apply across multiple database engines.

Performance efficiency pillar

The performance efficiency pillar of the AWS Well-Architected Framework focuses on how to optimize performance while ingesting or querying data. Performance optimization is an incremental and continual process of the following:

- Confirming business requirements
- Measuring the workload performance
- Identifying under-performing components
- Tuning the components to meet your business needs

The performance efficiency pillar provides use case–specific guidelines that can help in identifying the right graph data model and query languages to use. It also includes best practices to follow when ingesting data into and consuming data from Amazon Neptune.

The performance efficiency pillar focuses on the following key areas:

- Graph modeling
- Query optimization
- Cluster right-sizing
- Write optimization

Understand graph modeling

Understand the difference between Labeled Property Graph (LPG) and Resource Description Framework (RDF) models. In most cases, it is a matter of preference. There are several use cases, however, where one model is better suited than the other. If you require knowledge of the path connecting two nodes in your graph, choose LPG. If you want to federate data across Neptune clusters or other graph triple stores, choose RDF.

If you are building a software as a service (SaaS) application or an application that requires multi-tenancy, consider incorporating the logical separation of tenants in your data model instead of having one tenant for each cluster. To achieve that type of design, you can use SPARQL named graphs and labeling strategies, such as prepending customer identifiers to labels or

adding property key-value pairs representing tenant identifiers. Make sure your client layer injects these values to keep that logical separation. For more information about multi-tenancy recommendations, see Multi-tenancy guidance for ISVs running Amazon Neptune databases.

The performance of your queries depends on the number of graph objects (nodes, edges, properties) that need to be evaluated in the processing of your query. As such, the graph model can have significant impact on the performance of your application. Use granular labels when possible, and store only the properties you need to achieve path determination or filtering. To achieve higher performance, consider precalculating parts of your graph, such as creating summarization nodes or more direct edges connecting common paths.

Try to avoid navigating across nodes that have an abnormally high number of edges with the same label. Such nodes often have thousands of edges (where most nodes have edge counts in the tens). The result is much higher compute and data complexity. These nodes might not be problematic in some query patterns, but we recommend modeling your data differently to avoid it, especially if you will navigate across the node as an intermediate step. You can use slow-query logs to help identify queries that navigate across these nodes. You will likely observe much higher latency and data access metrics than your average query patterns, especially if you use debug mode.

Use deterministic node IDs for nodes and edges if your use case supports it instead of using Neptune to assign random GUID values for IDs. Accessing nodes by ID is the most efficient method.

Optimize queries

The openCypher and Gremlin languages can be used interchangeably on LPG models. If performance is a top concern, consider using the two languages interchangeably because one might perform better than the other for specific query patterns.

Neptune is in the process of converting to its alternative query engine (DFE). openCypher runs on the DFE only, but both Gremlin and SPARQL queries can be optionally set to run on the DFE by using query annotations. Consider testing your queries with the DFE activated and comparing performance of your query pattern when not using the DFE.

Neptune is optimized for transactional type queries that start at a single node or set of nodes and fan out from there, rather than analytical queries that evaluate the entire graph. For your analytical query workloads, use Neptune Analytics. Neptune Analytics is an ideal choice for investigatory, exploratory, or data-science workloads that require fast iteration for data, analytical and algorithmic processing. It can also perform a vector search on graph data and can load data

directly from your Neptune database instance. If Neptune Analytics does not meet your needs, you can also consider AWS SDK for Pandas or using neptune-export combined with AWS Glue or Amazon EMR.

To identify inefficiencies and bottlenecks in your models and queries, use the `profile` and `explain` APIs for each query language to obtain detailed explanations of the query plan and query metrics. For more information, see Gremlin profile, openCypher explain, and SPARQL explain.

Understand your query patterns. If the number of distinct edges in a graph becomes large, the default Neptune access strategy can become inefficient. The following queries might become quite inefficient:

- Queries that navigate backward across edges when no edge labels are given.
- Clauses that use this same pattern internally, such as `.both()` in Gremlin, or clauses that drop nodes in any language (which requires dropping incoming edges without knowledge of labels).
- Queries that access property values without specifying property labels. These queries might become quite inefficient. If this matches your usage pattern, consider enabling the OSGP index (object, subject, graph, predicate).

Use slow-query logging to identify slow queries. Slow queries can be caused by unoptimized query plans or unnecessarily large numbers of index lookups, which can increase I/O costs. The Neptune explain and profile endpoints for Gremlin, SPARQL, or openCypher can help you understand why these queries are slow. Causes might include the following:

- Nodes with an abnormally high number of edges compared with the average node in the graph (for example, thousands compared with tens) can add computational complexity and therefore longer latency and greater resource consumption. Determine whether these nodes are correctly modeled, or whether the access patterns can be improved to reduce the number of edges that must be traversed.
- Unoptimized queries will contain a warning that specific steps are not optimized. Rewriting these queries to use optimized steps might improve performance.
- Redundant filters might cause unnecessary index lookups. Likewise, redundant patterns might cause duplicate index lookups that can be optimized by improving the query (see `Index Operations - Duplication ratio` in the profile output).
- Some languages such as Gremlin don't have strongly typed numerical values, and they use type promotion instead. For example, if the value is 55, Neptune looks for values that are integers,

longs, floats, and other numerical types equivalent to 55. This results in additional operations. If you know your types match in advance, you can avoid this by using a query hint.

- Your graph model can greatly impact performance. Consider reducing the number of objects that need to be evaluated by using more granular labels or by precalculating shortcuts to multiple-hop linear paths.

If query optimization alone does not allow you to reach your performance requirements, consider using a variety of caching techniques with Neptune to achieve those requirements.

Neptune performance is continuously improving with each version. Review the release notes to see details of the improvements with each release. Consider planning regular updates to your Neptune DB cluster to help achieve optimal performance. Newer versions also support newer instances. Consider upgrading to 1.4.5.0 or later to be able to utilize the `r8g` instances. For more information about how this can improve your workload's performance, see 4.7 times better write query price-performance with AWS Graviton4 R8g instances using Amazon Neptune v1.4.5.

Right-size clusters

Size your cluster for your concurrency and throughput requirements. The number of concurrent queries that can be handled by each instance in the cluster is equal to two times the number of virtual CPUs (vCPUs) on that instance. Additional queries that arrive while all worker threads are occupied are put into a server-side queue. Those queries are handled on a first-in-first-out (FIFO) basis when worker threads become available. The `MainRequestQueuePendingRequests` Amazon CloudWatch metric shows the current queue depth for each instance. If this value is frequently above zero, consider choosing an instance with more vCPUs. If the queue depth exceeds 8,192, Neptune will return a `ThrottlingException` error.

Approximately 65 percent of the RAM for each instance is reserved for buffer cache. The buffer cache holds the working data set of data (not the entire graph; just the data that is being queried). To determine what percentage of data is being fetched from the buffer cache instead of storage, monitor the CloudWatch metric `BufferCacheHitRatio`. If this metric often drops below 99.9 percent, consider trying an instance with more memory to determine if it decreases your latency and I/O costs.

Read replicas do not have to be the same size same as your writer instance. However, heavy write workloads can cause smaller replicas to fall behind and reboot because they cannot keep up

with the replication. Therefore, we recommend making replicas equal to or larger than the writer instance.

When using auto-scaling for your read replicas, remember that it might take up to 15 minutes to bring a new read replica online. When the client traffic increases quickly but predictably, consider using scheduled scaling to set the minimum number of read replicas higher to account for that initialization time.

Serverless instances support several different use cases and workloads. Consider serverless over provisioned instances for the following scenarios:

- Your workload fluctuates often throughout the day.
- You created a new application and you are unsure what the workload size will be.
- You're performing development and testing.

It's important to note that serverless instances are more expensive than equivalent provisioned instances on a dollar per GB of RAM basis. Each serverless instance consists of 2 GB of RAM along with associated vCPU and networking. Perform a cost analysis between your options to avoid surprise bills. In general, you will achieve cost savings with serverless only when your workload is very heavy for only a few hours each day and almost zero the rest of the day or if your workload fluctuates significantly throughout the day.

Utilize the Amazon Neptune Pricing Calculator to help assess the correct configuration for your cluster based upon factors like queries-per-second (QPS) requirements.

Optimize writes

To optimize writes, consider the following:

- The Neptune Bulk Loader is the optimal way to initially load your database or append to existing data. The Neptune loader is not transactional and cannot delete data, so do not use it if these are your requirements.
- Transactional updates can be made by using the supported query languages. To optimize write I/O operations, write data in batches of 50-100 objects per commit. An object is a node, an edge, or a property on a node or edge in LPG, or a triple store or a quad in RDF..
- All transactional Neptune write operations are single threaded for each connection. When sending a large amount of data to Neptune, consider having multiple parallel connections that

are each writing data. When you choose a Neptune provisioned instance, the instance size is associated with a number of vCPUs. Neptune creates two database threads for each vCPU on the instance, so start at twice the number of vCPUs when testing for optimal parallelization. Serverless instances scale the number of vCPUs at a rate of approximately one for each 4 NCUs.

> **Note**
>
> This does not apply to the bulk loading API, only direct connections.

- Plan for and efficiently handle ConcurrentModificationExceptions during all write processes, even if only a single connection is writing data at any time. Design your clients for reliability when `ConcurrentModificationExceptions` occur.
- If you want to delete all of your data, consider using the fast reset API instead of issuing concurrent delete queries. The latter will take much longer and incur substantial I/O cost compared with the former.
- If you want to delete most of your data, consider exporting the data you that you want to keep by using neptune-export to load the data into a new cluster. Then delete the original cluster.

Cost optimization pillar

The cost optimization pillar of the AWS Well-Architected Framework focuses on avoiding unnecessary costs. The following recommendations can help you meet the cost optimization design principles and architectural best practices for Amazon Neptune.

The cost optimization pillar focuses on the following key areas:

- Understanding spending over time and controlling fund allocation
- Selecting resources of the right type and quantity
- Scaling to meet business needs without overspending

Understand usage patterns and services needed

Neptune is a good fit for your workload if your data model has a discernible graph structure, and your queries need to explore relationships and traverse multiple hops. A graph database isn't a good fit for the following patterns:

- Mainly single-hop queries (consider whether your data might be better represented as attributes of an object)
- JSON or BLOB data stored as properties
- Queries that aggregate across a dataset, such as calculating the sum of a numeric property across a large number of nodes

Consider whether using several purpose-built databases together for specific access patterns might address all of your needs. For example:

- An API that requires less frequent complex graph navigations alongside highly concurrent retrieval of properties for a single node might be best presented by using one or more of Neptune, DynamoDB, or Amazon DocumentDB.
- Relational databases can co-exist with Neptune to maintain your existing functionality, but use Neptune only for multiple-hop traversals that do not perform and scale well in relational databases.

Understand the costs associated with services that interact with and complement Neptune, including the following:

- Amazon Simple Storage Service (Amazon S3) storage costs for data files being bulk loaded into Neptune
- Lambda functions used for insert or upsert queries, read queries, and Neptune streams processing
- The API layer built on Neptune to interact with the client application (instead of having direct connections to the database) in Amazon API Gateway or AWS AppSync
- AWS Glue jobs used to transfer data to and from Neptune
- Amazon Kinesis or Amazon Managed Streaming for Apache Kafka (Amazon MSK) instances receiving streaming data for near real-time ingestion into Neptune.
- AWS Database Migration Service for migration of relational data to Neptune
- Amazon SageMaker Runtime costs for Jupyter notebooks and deep graph library machine learning models

Select resources with attention to cost

Neptune pricing is based on hourly instance cost (or Neptune Compute Units consumed for serverless), data I/O, and storage usage. Instances make up, on average, 85 percent of the overall cost, so right-sizing can have significant cost implications. The best way to right-size instances is to test application performance on a variety of instances and compare following factors:

- Does the `MainRequestQueuePendingRequests` CloudWatch metric stay at a consistently low number near zero?
- Does the `BufferCacheHitRatio` CloudWatch metric stay at or above 99.9 percent a majority of the time?
- What are the cost and performance curves for instance costs and for associated data I/O costs? Data read costs might increase significantly with an undersized instance that requires frequent buffer cache swapping with storage. `BufferCacheHitRatio` will be dropping frequently in these scenarios.

Instance costs scale linearly with size within the same instance family. The hourly cost of the `db.r6i.2xlarge` instance is twice that of the `db.r6i.xlarge` instance and also has twice

the resource allocation. The `db.r6i.24xlarge` instance is 24 times the hourly cost of the `db.r6i.xlarge` instance.

Estimate the number of concurrent queries you must support. You can have between zero and fifteen read replicas for processing read-only queries. If your requirements vary by the time of day, week, or month, you can use multiple smaller instances to scale on a schedule. Each vCPU on an instance provides two threads for handling concurrent queries. Three `db.r6i.xlarge` read replicas, with 4 vCPU each, can handle 24 concurrent queries..

If your traffic volume is instead measured in queries per second (QPS), you must experiment to determine the average latency of your queries. The number of queries per second a Neptune cluster can support is equal to `vCPU × 2 × (1 second/average query latency)`. For example, if you have 4 vCPU and query latency of 100 milliseconds (0.1 second), `QPS = 4 × 2 × (1s/0.1s) = 80 queries per second`.

Provisioned instances are cheaper than serverless for continuous, stable, and predictable workloads. Serverless provides opportunities for optimizing costs when you have a workload that requires very high usage for just a few hours per day (for example, `db.r6i.4xlarge`) and then almost no traffic for the remainder of the day (for example, 1 Neptune Compute Unit). A serverless instance that scales up for a few hours and then back down will be less expensive than using a provisioned `db.r6i.4xlarge` instance all day.

Consider upgrading to Neptune 1.4.5.0 or later and utilizing `r8g` instances to achieve better read and write throughput at a lower cost than older generation instances, such as `r7g` or `r6g`. For more information, see 4.7 times better write query price-performance with AWS Graviton4 R8g instances using Amazon Neptune v1.4.5 (AWS blog post).

Neptune clusters are created by default with standard storage (if you create using the console, it will default to selecting I/O-optimized storage). With I/O-optimized storage, you pay a slightly higher cost for storage and instances, but there are no I/O costs. This leads to more predictable recurring costs, but if your I/O usage is generally low, it may be more cost efficient to utilize standard storage. If you intend to load a lot of data initially, you can optimize cost by choosing I/O-optimized storage, performing your initial data load, and then switch to standard storage. The storage type affects the billing model only and has no technical difference in the Neptune DB cluster or instance configuration. You can change the storage type once per 30 days. After 30 days, check your detailed Neptune costs and use the Neptune pricing page to calculate whether your costs would have been higher using I/O-optimized storage. If they would have been, continue to use standard storage, otherwise switch back to I/O-optimized.

Choose the best Neptune instance configuration for your workload

If you created your AWS account before July 15th, 2025, you can use the AWS Free Tier for entry-level experimentation with Neptune. The 750 free hours of `db.t3.medium` and `db.t4g.medium` instance usage are enough for you to get a good understanding of Neptune at low scale. Your cluster will remain after the free trial period ends, although you will be charged for usage going forward from that point.

The `db.t3.medium` and `db.t4g.medium` instances are good for low-cost development environments where you are not using openCypher, Graph Explorer, or various generative AI integrations. These instances have a smaller RAM-to-vCPU ratio (2:1) than the R family instances (8:1) or X family instances (16:1). This reduces ratio prevents the use of DFE engine statistics that enable openCypher performance, GenAI integrations (to inform the LLM of the graph schema), and Graph Explorer. Performance profiles might differ significantly when using T family instances, especially for the previously mentioned workloads. These instances can also increase the occurrence of `OutOfMemoryExceptions` when queries navigate across a significant portion of the graph. To determine whether the latter condition might be affected, check the `BufferCacheHitRatio` CloudWatch metric.

We strongly advise against doing any performance or load testing with T family instances because you might experience inconsistent results that are not indicative of a production environment.

Provisioned instances give you the best cost and performance combination when your workload is fairly stable and predictable. Choose the instance size based on the request concurrency required and the query complexity. Higher concurrency requires more vCPUs. Higher query complexity requires more RAM. Use the `MainRequestQueuePendingRequests` CloudWatch metric to determine the impact of the former (greater than zero represents more concurrent requests than can be handled). Use the `BufferCacheHitRatio` CloudWatch metric to determine the impact of the latter. A ratio that is frequently falling lower than 99.9 percent suggests that there isn't enough RAM to contain the working portion of the graph being evaluated, which results in more frequent cache swapping. If the R family of instances provides sufficient concurrency but not enough RAM, consider trying the X family of instances.

Ideal use cases for serverless instances are described in the Neptune documentation. If you are unsure whether provisioned or serverless is best for you, and cost is your primary concern, test your workload in serverless to determine the number of NCUs used and compare the cost of provisioned

(`N hours × hourly provisioned cost`) with serverless (`sum of NCUs × hourly cost per NCU`). If you are unsure about the equivalent sized provision instance, one NCU is equivalent to approximately 2 GB of RAM and associated vCPU and networking. If your provisioned instance is from the `r6i` family, the ratio is 1 vCPU per 8 GB of RAM, or 4 NCUs, along with associated networking. The Amazon Neptune Pricing Calculator also provides a comparison to help you decide your optimal cost configuration.

When using serverless for primary and replica instances, remember that read replicas in promotion tiers 0 and 1 will scale their NCUs in line with the writer instance so that they are properly scaled if a failover event occurs. Set your NCU limits for these instances based on which of your instances—writer or readers—receive the most traffic.

In environments where the cluster is not needed 24 hours per day, 7 days a week, consider writing scripts that will turn off the Neptune instances when not in use and start them again before they are used. Neptune instances will automatically restart every 7 days to ensure required maintenance updates are applied. If you intend to leave the instances off for long durations, use a weekly script to shut them down again.

Right-size data storage and transfer

More efficient queries (for example, queries that need to touch fewer nodes, edges, and properties in the graph) require less I/O transfer and potentially can make use of smaller instances because less buffer cache is required. Use the profile or explain endpoints for your query language to optimize your query, and consider optimizing your graph model for your query performance.

Neptune uses dictionary encoding on large strings, and that dictionary is optimized for performance, not efficiency. If you have large BLOBs, JSON, or frequently changing strings for properties, consider storing them outside Neptune in Amazon S3, Amazon DynamoDB, or Amazon DocumentDB, and store only a reference within the Neptune node.

In some cases, choosing a larger instance size can be cheaper. If your I/O costs are very high because of a low `BufferCacheHitRatio`, it's possible that the larger buffer cache would significantly reduce that cost. That's because all of the data would fit in the cache instead of being frequently swapped from storage and incurring the I/O transfer rate.

Neptune uses copy-on-write cloning. When cloning to split a graph into multiple shards, it might be more efficient not to delete the unwanted data on the cloned cluster because that will involve the creation of new data pages, resulting in increased storage costs. Data that is unchanged from

before the cloning event will exist in a single data page shared across the two clusters and will be charged only for that single copy.

Do not enable the OSGP index or use R5d instances unless you have tested to confirm that they make a substantial difference in your workload. Both are designed for rarely occurring scenarios, and they might increase your costs for minimal or no gains.

Sustainability pillar

The sustainability pillar focuses on minimizing the environmental impacts of running cloud workloads. Key topics include a shared responsibility model for sustainability, understanding impact, and maximizing use to minimize required resources and reduce downstream impacts.

The sustainability pillar contains the following key focus areas:

- Your impact
- Sustainability goals
- Maximized usage
- Anticipating and adopting new, more efficient hardware and software offerings
- Use of managed services
- Downstream impact reduction

This guide focuses on your impact. For more information about the other sustainability design principles, see the AWS Well-Architected Framework.

Your choices and requirements have an impact on the environment. If you can choose AWS Regions that have lower carbon intensity, and if your requirements reflect actual workload needs instead of just maximizing uptime and durability, the sustainability of the workload increases. The next sections discuss best practices and thoughtful considerations that will have a positive environmental impact if adopted in your workload design and ongoing operations.

AWS Region selection

Some AWS Regions are near Amazon renewable energy projects or located where the grid has a published carbon intensity that is lower than others. Consider the sustainability impact for Regions that might be viable for your workload, and cross-reference your list with the Regions where Neptune is available.

Consumption based on user-behavior patterns

Right-sizing your consumption to match the traffic and behavior of your users helps AWS minimize the impact of services on the environment. Consider the following best practices when designing your solution:

- Monitor Amazon CloudWatch metrics such as `CPUUtilization`, `MainRequestQueuePendingRequests`, and `TotalRequestsPerSec` to determine when your demand is highest and lowest, and ensure that your cluster resources are right-sized during those times.
- Automate the stopping of non-production environments during hours when they are not being used. For more information, see the blog post Automate the stopping and starting of Amazon Neptune environment resources using resource tags.
- If your traffic patterns vary frequently and unpredictably, consider using Neptune Serverless instances that will scale up and down with demand instead of using an instance provisioned for peak traffic.
- Consider aligning your service-level agreements with sustainability goals in addition to business continuity goals. Easing requirements such as multi-Region disaster recovery, high availability, or long-term backup retention, especially for non-production environments or non-mission critical workloads, can reduce the amount of resources required to meet those goals.

Optimize software development and architecture patterns

To prevent waste, optimize your models and queries, and share compute resources so that you use all the resources available in Neptune instances and clusters. Specific best practices include:

- Have developers share Neptune instances and Jupyter Notebook application instances instead of each creating their own. Give each developer their own logical partition in a single Neptune cluster through the use of multi-tenancy partitioning strategies, and create separate notebook folders for each developer on a single Jupyter instance.
- Implement patterns that maximize the use of resources and minimize idle time, such as parallel threads for loading data and batching records together into a larger transaction.
- Optimize your queries and graph model to minimize the resources required to compute the results.
- For Gremlin query results, use the results cache feature to minimize the resources spent recalculating paginated or frequently recurring queries.
- Keep your Neptune environments up to date. The newest versions of Neptune support the latest Amazon EC2 instances, such as Graviton, that are more efficient. They also have query optimization improvements and bug fixes that reduce the amount of resources needed to calculate your queries.

Resources

References

- AWS Well-Architected
- AWS Well-Architected Framework documentation
- Neptune latest updates
- Best practices: getting the most out of Neptune
- Amazon Neptune Pricing Calculator

Blog posts

- Automated testing of Amazon Neptune data access with Apache TinkerPop Gremlin
- Automate the stopping and starting of Amazon Neptune environment resources using resource tags
- Fine Grained Access Control for Amazon Neptune data plane actions
- 4.7 times better write query price-performance with AWS Graviton4 R8g instances using Amazon Neptune v1.4.5
- How Orca Security optimized their Amazon Neptune database performance
- Build graph applications faster with Amazon Neptune public endpoints
- New Amazon Neptune engine version delivers up to 9 times faster and 10 times higher throughput for openCypher query performance

Free AWS Skill Builder courses

- Getting Started with Amazon Neptune
- Building Applications on Amazon Neptune
- Data Modeling for Amazon Neptune

Contributors

Contributors to this guide include:

- Brian O'Keefe, Principal Neptune Solutions Architect, AWS
- Abhishek Mishra, Senior Neptune Solutions Architect, AWS
- Ganesh Sawhney, Team Lead - Strategic Partner Success Solutions Architect, AWS
- Michael Havey, Senior Neptune Solutions Architect, AWS
- Kevin Phillips, Neptune Solutions Architect, AWS
- Melissa Kwok, Neptune Solutions Architect, AWS
- Sakti Mishra, Principal Solutions Architect AWS
- Javed Ali, Senior 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
Neptune release updates	We updated the documentation to include information about Amazon Neptune 1.4.6.0 and later.	January 2, 2026
Initial publication	—	September 27, 2023

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.

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.