Testing serverless applications on AWS

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Testing serverless applications on AWS

Table of Contents

Introduction **1**
Overview 1
Prerequisites 2
Definitions 2
Objectives 2
Increase software quality 2
Decrease time to implement or change features 5
Testing techniques for serverless applications **6**
Mock testing 6
Emulation testing 7
Testing in the cloud 8
Challenges when testing serverless applications **10**
Example: A Lambda function that creates an Amazon S3 bucket 10
Example: A Lambda function that processes messages from Amazon SQS 11
Best practices for testing serverless applications **12**
Prioritize testing in the cloud 12
Use mocks if necessary 12
Understand the tradeoffs of emulation testing 13
Scope tests through natural boundaries 14
Identify architecture boundaries 14
Separate Lambda code from business logic 14
Treat boundaries as contracts 14
Use test harnesses for asynchronous workflows 15
Organize cloud environments for developer isolation 15
Accelerate feedback loops 16
FAQ **17**
I have a Lambda function that performs calculations and returns a result without calling any other services. Do I really need to test this in the cloud? 17
How can testing in the cloud help with unit testing? If it's in the cloud and connects to other resources, isn't that an integration test? 17
Next steps and resources **19**
Sample implementations 19
Further reading 19
References 19

Tools 19
Contributors 20
Authoring 20
Reviewing 20
Technical writing 20
Document history 21
Glossary 22
..... 22
A 23
B 26
C 28
D 31
E 35
F 37
G 39
H 40
I 41
L 43
M 45
O 49
P 51
Q 54
R 54
S 57
T 61
U 62
V 63
W 63
Z 64

Testing serverless applications on AWS

Amazon Web Services (???contributors)

March 2026 (document history)

This guide discusses methodologies for testing serverless applications, describes the challenges that you might encounter during testing, and introduces best practices. These testing techniques are intended to help you iterate more quickly and release your code more confidently.

This guide is for developers who are looking to establish testing strategies for their serverless applications. You can use the guide as a starting point to learn about testing strategies, and then visit the Serverless Test Samples repository to see examples of tests that follow the patterns and best practices described in this guide. This guide describes serverless testing methodologies, describes the challenges customers encounter when testing serverless applications, and introduces best practices for testing serverless applications. These techniques are intended to help developers iterate more quickly and release more confidently.

Overview

Automated tests are critical investments that help ensure application quality and development speed. Testing also accelerates developer feedback. As a developer, you want to be able to iterate rapidly on your application and get feedback on the quality of your code. Many developers are used to writing applications that they deploy to an environment on their desktop, either directly to their operating system or within a container-based environment. When you work in desktop or container-based environments, you typically write tests against code that is hosted entirely on your desktop. However, in serverless applications, architecture components might not be deployable to a desktop environment but might instead exist only in the cloud. A cloud-based architecture might include persistence layers, messaging systems, security constructs, APIs, and other components. When you write application code that relies on these components, it might be difficult to determine the best way to design and run tests.

This guide helps you align to a testing strategy that reduces friction and confusion, and increases code quality.

Prerequisites

This guide assumes that you are familiar with the basics of automated tests, including how automated software tests are used to ensure software quality. The guide provides a high-level introduction to a serverless application testing strategy, and doesn't require any hands-on experience writing tests.

Definitions

This guide uses the following terms:

- *Unit tests* are tests that are run against code for a single architectural component in isolation.
- *Integration tests* are run against two or more architectural components, typically in a cloud environment.
- *End-to-end tests* verify behaviors across entire applications or workflows.
- *Emulators* are applications (often provided by a third party) that are designed to mimic a cloud service without provisioning or invoking any cloud resources.
- *Mocks* (also called *fakes*) are implementations in a testing application that replace a dependency with a simulation of that dependency.

Objectives

The best practices in this guide are intended to help you achieve two main objectives:

- Increase quality of serverless applications
  - Testing at architecture boundaries
  - Testing at code boundaries
- Decrease time to implement or change features

Increase software quality

An application's quality depends to a large extent on the ability of developers to test a variety of scenarios to verify functionality. When you don't implement automated tests, or, more typically, if your tests don't cover the required scenarios adequately, the quality of your application can't be determined or guaranteed.

In a server-based architecture, teams are able to easily define a scope for testing: Any code that runs on the application server has to be tested. Other components that call in to the server, or dependencies that the server calls, are often considered external and out of scope for testing by the team responsible for the application on the server.

Serverless applications often consist of smaller units of work, such as AWS Lambda functions, that run in their own environment. Teams will likely be responsible for multiples of these smaller units within a single application. Some application functionality can be delegated entirely to managed services such as Amazon Simple Storage Service (Amazon S3) or Amazon Simple Queue Service (Amazon SQS) without using any internally developed code. Traditional server-based models for software testing might exclude managed services by considering them external to the application. This can lead to inadequate coverage, where critical scenarios might be limited to manual exploratory testing or to a few integration test cases where the outcome varies by environment. Therefore, adopting testing strategies that encompass managed service behaviors and cloud configurations can improve software quality.

Testing at architecture boundaries

As serverless applications grow, they naturally spread across multiple architectural components. While this uses AWS distributed capabilities, it can make end-to-end behavior difficult to understand.

Identifying natural boundaries

When designing your architecture following serverless best practices (one function = one job, decoupling), you'll notice natural boundaries around subsystems. These boundaries represent logical separation points in your application.

Boundaries as testing contracts

These architectural boundaries are excellent candidates for testing edges. Treat each boundary as a contract and validate that it behaves according to its defined specification. Think of these boundaries as *seams* in your application where you can insert test validation.

Key benefits

The following are key benefits of testing at architecture boundaries:

- **Focused testing scope** – Test subsystems independently without needing to understand the entire application.

- **Contract validation** – Ensure each boundary maintains its expected behavior as the system evolves
- **Dual-purpose instrumentation** – These same boundaries make excellent observability hooks in production
- **Test harness** – Allows you to test asynchronous serverless systems. It helps you test event-driven architectures by capturing and validating events as they flow through your subsystem.

Testing at code boundaries

Define clear code boundaries by separating infrastructure code, such as Lambda code, from your core business logic. This separation creates distinct testing scopes that simplify your test strategy.

The boundary pattern

Establish two clear code boundaries in your Lambda functions:

- **Outer boundary (Lambda handler)** – A slim adapter layer that handles concerns specific to AWS Lambda
- **Inner boundary (business logic)** – Pure business logic methods independent of Lambda runtime

Handler as adapter (outer scope)

Your Lambda function handler should be a thin layer that:

- Extracts data from the incoming `event` and `context` objects
- Validates the extracted data
- Passes only relevant details to business logic methods
- Returns results in the expected format for Lambda

Business logic (inner scope)

Your core business logic should:

- Operate independently of details specific to Lambda
- Accept simple, validated inputs
- Return predictable outputs

- Require minimal dependencies for initialization

Testing benefits by scope

- **Inner boundary tests** – Comprehensive unit tests around business logic without Lambda complexity or environment setup
- **Outer boundary tests** – Focused integration tests validating the adapter layer's event handling and data extraction
- **Minimal test overhead** – No complex environments or extensive dependencies needed for the majority of your tests

This boundary-based approach allows you to test most of your code as pure functions while keeping Lambda tests minimal and targeted.

Decrease time to implement or change features

You can minimize the effect of software bugs and configuration problems on costs and schedules by catching these issues during an iterative development cycle. When a developer fails to detect these issues, more people must invest additional effort to identify the problems.

A serverless architecture might include managed services that provide critical application functionality through API calls. For this reason, your development cycle should include tests that validate both the *happy path* (where interactions with these services behave as expected) and the *sad path* (where calls fail, return unexpected responses, or behave differently across environments). Without these tests in place, you may encounter issues that stem from differences between your local environment and the deployed environment. When that happens, you must spend additional time attempting to reproduce and verify a fix, because each iteration now requires validating changes against an environment that differs from your preferred setup.

A proper serverless testing strategy improves your iteration time by providing accurate results for tests that include calls to other services.

Testing techniques for serverless applications

There are three main approaches to running tests against serverless application code: mock testing, emulation testing, and testing in the cloud. You can generally find a mix of these approaches in use within the scope of a single project. For example, you might use mock testing when you're developing your code locally, emulation testing to more closely replicate service behavior before deployment, and cloud testing as part of a nightly continuous integration and continuous delivery (CI/CD) process.

Mock testing

Mock testing is a strategy where you create replacement objects in your code that simulate the behavior of cloud services. For example, you can write a test that uses a mock of the Amazon S3 service, and you can configure that mock to return a specific response object whenever the `PutObject` method is called. When the test runs, the mock returns the response without calling Amazon S3 or any other service endpoints.

These replacement objects are often generated by a mock framework to reduce the amount of implementation necessary for a given test. Some mock frameworks are generic and others are designed specifically for AWS SDKs.

Mocks, along with stubs, fall into the broader category of *fakes*. Mocks differ from emulators (discussed later in this section) in that mocks are typically created or configured by a developer as part of the test code, whereas emulators are standalone applications that expose APIs (such as REST APIs) in the same manner as the systems they emulate.

The advantages of using mocks include the following:

- Mocks can simulate third-party services that are beyond the control of your application, such as APIs and software as a service (SaaS) providers, without needing direct access to those services.
- Mocks are also useful for testing failure conditions, especially when such conditions (such as service outages) are hard to simulate.
- Like emulators, mock frameworks can provide fast local development after they've been configured.
- Mocks can provide substitute behavior for virtually any kind of object, so mocking strategies can create coverage for a wider variety of services than emulators.

- When new features or behaviors become available, mock testing can react more quickly by using (or falling back to) a generic mock framework, which can simulate the new features as soon as the updated AWS SDK becomes available.

Mock testing has these disadvantages:

- Mocks generally require a non-trivial amount of setup and configuration effort, specifically when trying to determine return values from different services in order to properly mock responses.
- Because mocks are written or configured by the developers who write the tests, this is an added responsibility for developers.
- You might need to have access to the cloud in order to understand the APIs and return values of services.
- Mocks can also be difficult to maintain, because they require updates whenever the mocked cloud API signatures change, return value schemas evolve, or the tested application logic is extended to make calls to new APIs. These changes create additional test development workload for developers.
- Mock tests might pass locally but fail in the cloud because they simulate — rather than replicate — the behavior of real services, leaving environment-specific issues undetected.
- Mock frameworks, like emulators, cannot detect AWS Identity and Access Management (IAM) policy violations, service quota limits, or payload size constraints, nor do they trigger ancillary services such as Amazon CloudWatch, AWS CloudTrail, or Amazon GuardDuty that may affect application behavior in a deployed environment.
- Although mocks are better at simulating what an application will do when authorization fails or a quota is exceeded, mock testing can't determine which outcome will actually occur when the code is deployed to the production environment.

Emulation testing

Emulation testing is enabled by locally running applications known as *emulators* that resemble AWS services. Emulators have APIs that are similar to their cloud counterparts and provide similar return values. They can also simulate state changes that are initiated by API calls. For example, an Amazon S3 emulator might store an object on the local disk when the `PutObject` method is called. When `GetObject` is called with the same key, the emulator reads the same object from disk and returns it.

The advantages of emulation testing include the following:

- It provides the most familiar environment for developers who are used to developing code exclusively in a local environment. For example, if you're familiar with the development of an *n*-tier application, you might have a database engine and web server, similar to those running in production, running on your local machine to provide quick, local, isolated test capability.
- It doesn't require any changes to cloud infrastructure (such as developer cloud accounts), so it's easy to implement with existing testing patterns. Emulation testing provides the benefits of fast, local development iterations.

However, emulators have several drawbacks:

- They're often hard to set up and replicate, especially when they're used as a part of CI/CD pipelines. This might increase the workload and maintenance for IT staff or for developers who manage their own software installations.
- Emulated features and APIs typically lag behind services changes and might hinder the adoption of new features until support is added.
- Emulators might require support, updates, bug fixes, or feature parity enhancements, which are the responsibility of the emulator author (which is often a third-party company).
- Tests that rely on emulators can provide successful results locally, but they might fail in the cloud due to interactions with other aspects of AWS such as IAM policies and quotas, which are generally not emulated.
- Some AWS services don't have emulators available, so if you rely only on emulation, you might not have a satisfactory testing option for portions of your application.

Testing in the cloud

Testing in the cloud is the process of running tests against code that is deployed to a cloud environment instead of a desktop environment. The value of testing in the cloud increases with cloud-native application development. For example:

- You can run tests in the cloud against the most recent APIs and service features, ensuring your tests reflect the latest return values and behaviors as AWS continues to launch new services and capabilities.
- Your tests can cover IAM policies, service quotas, configurations, and all services.

- Your cloud test environment most closely resembles your production runtime environment, so tests that you run in the cloud are likely to achieve the most consistent results as they progress through environments.

Drawbacks to testing in the cloud include the following:

- Deployments to cloud environments typically take more time than deployments to desktop environments. Tools such as AWS Serverless Application Model (AWS SAM) Accelerate, AWS Cloud Development Kit (AWS CDK) watch mode, and SST help decrease the latency involved with cloud deployment iterations.
- Cloud testing can incur additional service costs, unlike local testing, which uses your local development environment.
- Testing in the cloud might be less feasible if you don't have high-speed internet access.
- In regulated industries, enterprise security policies may restrict developer access to cloud environments, making it difficult or impossible to run cloud tests as part of a local development workflow.
- Environment boundaries are often drawn at the stack level in shared accounts for developer environments, sometimes by using namespace type strategies such as using prefixes to identify ownership. For pre-production and production environments, boundaries are typically drawn at the account level to insulate workloads from noisy neighbor problems, to support least privilege security controls, and to protect sensitive data. The requirement to create isolated environments might place additional burdens on DevOps teams, especially if they're in an enterprise that has strict controls around accounts and infrastructure.

Challenges when testing serverless applications

When you use emulators and mocked calls to test your serverless application on your local desktop, you might experience testing inconsistencies as your code progresses from environment to environment in your CI/CD pipeline. The unit tests you create on your desktop to validate your application's business logic might not include or accurately represent critical aspects of cloud services. Complete tests can't be performed locally in isolation. They require verifying the permissions and configurations among multiple services.

The following sections outline the challenges you might experience when implementing a cloud testing strategy. The following sections outline the challenges customers experience when trying to implement a cloud testing strategy, and our guidance on best practices to achieve effective test coverage.

Example: A Lambda function that creates an Amazon S3 bucket

If a Lambda function's logic depends on creating an Amazon S3 bucket, a complete test should confirm that Amazon S3 was successfully called and the bucket was successfully created. In a mock testing setup, you might mock a success response and potentially add a test case to handle a failure response. In an emulation testing scenario, the `CreateBucket` API might be called, but the identity that makes the call will not originate from the Lambda service assuming a role, and a placeholder authentication will be used instead—this is often your more permissive role or user identity.

The mock and emulation setups discussed previously will most likely test *what the Lambda function will do* if it successfully (or unsuccessfully) calls Amazon S3. However, those tests will fail to capture *whether the Lambda function is capable* of successfully creating the bucket, given the function's configuration. This configuration is likely represented by infrastructure as code (IaC) for products and services such as AWS CloudFormation, AWS SAM, or HashiCorp Terraform. One possible issue is that the role that is assigned to the function doesn't have an attached policy that allows for the `s3:CreateBucket` action, and the function will therefore always fail when it is deployed to a cloud environment.

Example: A Lambda function that processes messages from Amazon SQS

If an Amazon Simple Queue Service (Amazon SQS) queue is the source of a Lambda function, a complete test should verify that the Lambda function is successfully invoked when a message is put in a queue. Emulation testing and mock testing are generally set up to run the Lambda function code directly, and to simulate the Amazon SQS integration by passing a JSON event payload (or a deserialized object) as the function handler's input.

Local testing that simulates the Amazon SQS integration will test *what the Lambda function will do* when it's called by Amazon SQS with a given payload, but it won't ensure that *Amazon SQS will successfully invoke the Lambda function* when it is deployed to a cloud environment.

Some examples of configuration problems you might encounter with Amazon SQS and Lambda include the following:

- Amazon SQS visibility timeout is too low, resulting in multiple invocations when only one was intended.
- The Lambda function's execution role doesn't allow reading messages from the queue (through `sqs:ReceiveMessage`, `sqs:DeleteMessage`, or `sqs:GetQueueAttributes`).
- The sample event that is passed to the Lambda function exceeds the Amazon SQS message size quota. Therefore, the test is invalid because Amazon SQS would never be able to send a message of that size.

As these examples show, tests that cover business logic but not the configurations between cloud services are likely to provide unreliable results.

Best practices for testing serverless applications

The following sections outline best practices for achieving effective coverage when testing serverless applications.

Prioritize testing in the cloud

For well-designed applications, you can employ a variety of testing techniques to satisfy a range of requirements and conditions. However, based on current tooling, we recommend that you focus on testing in the cloud as much as possible. Although testing in the cloud can create developer latency, increase costs, and sometimes require investments in additional DevOps controls, this technique provides the most reliable, accurate, and complete test coverage.

You should have access to isolated environments in which to perform testing. Ideally, each developer should have a dedicated AWS account to avoid any issues with resource naming that can occur when multiple developers who are working in the same code try to deploy or invoke API calls on resources that have identical names. These environments should be configured with the appropriate alerts and controls to avoid unnecessary spending. For example, you can limit the type, tier, or size of resources that can be created, and set up email alerts when estimated costs exceed a given threshold.

If you must share a single AWS account with other developers, automated test processes should name resources to be unique for each developer. For example, update scripts or TOML configuration files that cause AWS SAM CLI sam deploy or sam sync commands can automatically specify a stack name that includes the local developer's user name.

Testing in the cloud is valuable for all phases of testing, including unit tests, integration tests, and end-to-end tests.

Use mocks if necessary

Mock frameworks are a valuable tool for writing fast unit tests. They are especially valuable when tests need to cover complex internal business logic, such as mathematical or financial calculations or simulations. Look for unit tests that have a large number of test cases or input variations, where the inputs do not change the pattern or the content of calls to other cloud services. Creating mock tests for these scenarios can improve developer iteration times.

Code that is covered by unit tests that use mock testing should also be covered by testing in the cloud. This is necessary because the mocks are still running on a developer's laptop or build machine, and the environment might be configured differently than it will be in the cloud. For example, your code might include AWS Lambda functions that use more memory or take more time than Lambda is configured to allocate when it's run with certain input parameters. Or your code might include environment variables that aren't configured in the same way (or at all), and the differences might cause the code to behave differently or to fail.

Don't use mocks of cloud services to validate the proper implementation of those service integrations. Although it might be acceptable to mock a cloud service when you're testing other functionality, you should test cloud service calls in the cloud to validate the correct configuration and functional implementation.

Mocks can add value to unit testing, especially when you're testing a large number of cases frequently. This benefit is reduced for integration tests, because the level of effort to implement the necessary mocks increases with the number of connection points. End-to-end testing should not use mocks, because these tests generally deal with states and complex logic that cannot be easily simulated with mock frameworks.

Understand the tradeoffs of emulation testing

Emulators can be a practical choice for specific use cases. For example, a development team with limited, inconsistent, or slow internet access may find that emulation testing is the most reliable way to iterate on code before moving to a cloud environment.

For most other circumstances, use emulators selectively. When you rely heavily on an emulator, it can become difficult to incorporate new AWS service features into your testing until the emulation vendor releases an update to provide feature parity. Emulators also require upfront and ongoing investment for setup and configuration across development systems and build machines. Additionally, many cloud services don't have emulators available; selecting an emulation-first strategy may either preclude the use of those services or produce code and configurations that aren't well tested against real service behavior.

If you use emulation testing, complement it with cloud testing as much as possible to validate that proper cloud configurations are in place and to test interactions with services that can only be simulated or mocked in an emulated environment.

Emulation testing can provide fast feedback for unit tests and, depending on the features and behavioral parity of the emulation software, may support some integration and end-to-end tests as well.

Scope tests through natural boundaries

As serverless applications grow across more architectural components, natural boundaries emerge around subsystems—especially when following best practices like single-purpose functions and event-driven decoupling. These boundaries serve as effective testing edges where you can validate contracts between components.

Identify architecture boundaries

Look for natural seams in your application design:

- Between services, such as an Amazon EventBridge rule connecting a publisher to a consumer
- At API edges, such as Amazon API Gateway endpoints that front Lambda functions
- Around workflows, such as AWS Step Functions orchestrating multiple services
- At storage layers, such as Amazon DynamoDB streams triggering downstream processing

Separate Lambda code from business logic

Simplify your tests by isolating Lambda code from core business logic. Your Lambda handler should act as a thin adapter between the AWS runtime and your application logic. It should extract and validate event data and then delegate to a testable function that has no Lambda dependencies. This makes your business logic portable, easier to reason about, and straightforward to test without mocking Lambda objects or setting up complex environments.

Treat boundaries as contracts

Test at the boundary, not through it. Validate what crosses the edge without requiring the entire downstream system. These same boundaries also serve as observability hooks in production. The architectural seams where you test can be instrumented for monitoring using Amazon CloudWatch Logs, AWS X-Ray traces, and EventBridge events.

Use test harnesses for asynchronous workflows

Serverless applications often rely on asynchronous patterns, where events trigger processing, messages flow through queues, and workflows span multiple services without immediate responses. You can't simply invoke a function and inspect a return value. The result may appear later in a database, a log stream, or another service.

A *test harness* is testing infrastructure you deploy alongside your application to observe and validate this asynchronous behavior. Test harnesses typically include:

- Event listeners that subscribe to the same events your application produces
- Storage mechanisms (such as DynamoDB tables or Amazon S3 buckets) where test results can be captured
- Polling logic in your test code that waits for expected outcomes to appear

Your test code initiates an event, waits for the workflow to complete, then queries the test harness to verify the expected outcome occurred.

The following are best practices:

- **Define clear SLAs for asynchronous operations** – Establish how long workflows should take and use these as polling timeouts in your tests
- **Use unique identifiers for test isolation** – Generate unique filenames, message IDs, or correlation tokens per test run to prevent interference between tests
- **Deploy test infrastructure alongside your application** – Include test harness resources in your infrastructure-as-code templates so they stay in sync as your application evolves
- **Clean up test data after test runs** – This prevents accumulating test artifacts in your cloud environment

Test harnesses are most valuable for integration tests that validate workflows across multiple services, end-to-end tests that verify complete user journeys, and event-driven architectures where services communicate through EventBridge, Amazon SNS, Amazon SQS, or Amazon Kinesis.

Organize cloud environments for developer isolation

Testing in the cloud requires environments that are isolated from one another. When developers share a single AWS account, such as a team development account, consider creating a separate

application stack for each developer or feature branch. This isolates resources, prevents naming collisions, and avoids quota contention or noisy neighbor issues during testing.

Use AWS Systems Manager Parameter Store or similar tooling to manage stack-specific configurations, such as API endpoints and queue names. For cost efficiency, share expensive resources like Amazon Relational Database Service (Amazon RDS) clusters across developer stacks while keeping lightweight serverless resources (such as Lambda functions, API Gateway stages, and DynamoDB tables) isolated per stack.

In regulated industries, enterprise security policies may restrict developer access to cloud environments, making it difficult to run cloud tests as part of a local development workflow. In these cases, emulation testing can fill the gap between local mock testing and full cloud validation, though it should be complemented with cloud testing whenever access permits.

Accelerate feedback loops

When you test in the cloud, use tools and techniques to accelerate development feedback loops. For example, use AWS SAM Accelerate and AWS CDK watch mode to decrease the time it takes to push code modifications to a cloud environment. The samples in the GitHub Serverless Test Samples repository explore some of these techniques.

We also recommend that you create and test cloud resources from your local machine as early as possible during development—not only after a check-in to source control. This practice enables quicker exploration and experimentation when developing solutions. In addition, the ability to automate deployment from a development machine helps you discover cloud configuration problems more quickly and reduces wasted effort from updating and approving modifications to source control.

FAQ

I have a Lambda function that performs calculations and returns a result without calling any other services. Do I really need to test this in the cloud?

Yes. AWS Lambda functions have configuration parameters that could change the outcome of the test. All Lambda function code has a dependency on timeout and memory settings, which could cause the function to fail if they aren't set properly. Lambda policies also enable standard output logging to Amazon CloudWatch. Even if your code doesn't call CloudWatch directly, a permission is required in order to enable logging, and that permission cannot be accurately mocked or emulated.

How can testing in the cloud help with unit testing? If it's in the cloud and connects to other resources, isn't that an integration test?

We define unit tests as tests that operate on architectural components in isolation. This definition doesn't necessarily preclude the use of service calls or other network communications.

Many serverless applications do have architectural components that can be tested in isolation, even in the cloud. A basic example is a Lambda function that takes some input, interprets it, and sends a message to an Amazon Simple Queue Service (Amazon SQS) queue. A unit test of such a function would likely test whether input values result in certain values being present in the queued message. Consider a test that is written by using the *arrange, act, assert* pattern:

- **Arrange** – Allocate resources (a queue to receive messages, and the function under test).
- **Act** – Call the function under test.
- **Assert** – Retrieve the message sent by the function, and validate the output.

A mock testing approach would involve mocking the queue with an in-process mock object, and creating an in-process instance of the class or module that contains the Lambda function code. During the assert phase, the queued message would be retrieved from the mocked object.

In a cloud-based approach, the test would create an Amazon SQS queue for the purposes of the test, and would deploy the Lambda function with environment variables that are configured to use the isolated Amazon SQS queue as the output destination. After running the Lambda function, the test would retrieve the message from the Amazon SQS queue.

The cloud-based test would run the same code, assert the same behavior, and validate the application's functional correctness. However, it would have the added advantage of being able to validate the following settings of the Lambda function: the AWS Identity and Access Management (IAM) role, IAM policies, and the function's timeout and memory settings.

Next steps and resources

Use the following resources for further reading and additional concrete examples.

Sample implementations

The Serverless Test Samples repository on GitHub contains concrete examples of tests that follow the patterns and best practices described in this guide. The repository contains sample code and guided walkthroughs of the mock, emulation, and cloud testing processes described in previous sections. Use this repository to get up to speed on the latest serverless testing guidance from AWS.

Further reading

Visit Serverless Land to access the latest blogs, videos, and training for AWS serverless technologies.

References

- Accelerating serverless development with AWS SAM Accelerate (AWS blog post)
- Increasing development speed with CDK Watch (AWS blog post)
- Mocking service integrations with AWS Step Functions Local (AWS blog post)
- Getting started with testing serverless applications (AWS blog post)
- Accelerate serverless testing with LocalStack integration in VS Code IDE (AWS blog post)
- Locally debug functions with AWS SAM (AWS documentation)

Tools

- AWS SAM – Testing and debugging serverless applications
- AWS SAM – Integrating with automated tests
- AWS Lambda – Testing Lambda functions in the console
- LocalStack Cloud Emulator – Enhance the local testing experience for serverless applications with LocalStack

Contributors

Authoring

- Dan Fox, AWS
- Leslie Raj, AWS
- Rohan Mehta, AWS
- Rob Hill, AWS

Reviewing

- Brian Krygsman, AWS

Technical writing

- Lilly AbouHarb, 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
Updates	We added guidance on testing at architecture and code boundaries, test harnesses for asynchronous workflows , and developer environme nt isolation. We also updated emulation testing recommend ations.	March 18, 2026
Initial publication	—	December 9, 2022

AWS Prescriptive Guidance glossary

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

Numbers

7 Rs

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

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

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

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

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

active-passive migration

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

aggregate function

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

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

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

anti-pattern

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

application control

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

application portfolio

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

artificial intelligence (AI)

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

artificial intelligence operations (AIOps)

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

asymmetric encryption

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

atomicity, consistency, isolation, durability (ACID)

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

attribute-based access control (ABAC)

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

authoritative data source

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

Availability Zone

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

AWS Cloud Adoption Framework (AWS CAF)

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

AWS Workload Qualification Framework (AWS WQF)

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

B

bad bot

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

BCP

See business continuity planning.

behavior graph

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

big-endian system

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

binary classification

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

bloom filter

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

blue/green deployment

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

bot

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

botnet

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

branch

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

break-glass access

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

brownfield strategy

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

buffer cache

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

business capability

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

business continuity planning (BCP)

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

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

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

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

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

chaos engineering

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

CI/CD

See continuous integration and continuous delivery.

classification

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

client-side encryption

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

Cloud Center of Excellence (CCoE)

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

cloud computing

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

cloud operating model

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

cloud stages of adoption

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

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

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

CMDB

See configuration management database.

code repository

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

cold cache

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

cold data

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

computer vision (CV)

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

configuration drift

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

configuration management database (CMDB)

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

conformance pack

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

continuous integration and continuous delivery (CI/CD)

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

CV

See computer vision.

D

data at rest

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

data classification

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

data drift

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

data in transit

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

data mesh

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

data minimization

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

data perimeter

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

data preprocessing

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

data provenance

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

data subject

An individual whose data is being collected and processed.

data warehouse

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

database definition language (DDL)

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

database manipulation language (DML)

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

DDL

See database definition language.

deep ensemble

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

deep learning

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

defense-in-depth

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

delegated administrator

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

deployment

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

development environment

See environment.

detective control

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

development value stream mapping (DVSM)

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

digital twin

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

dimension table

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

disaster

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

disaster recovery (DR)

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

DML

See database manipulation language.

domain-driven design

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

DR

See disaster recovery.

drift detection

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

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

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

electronic data interchange (EDI)

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

encryption

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

encryption key

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

endianness

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

endpoint

See service endpoint.

endpoint service

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

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

enterprise resource planning (ERP)

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

envelope encryption

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

environment

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

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

epic

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

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

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

F

fact table

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

fail fast

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

fault isolation boundary

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

feature branch

See branch.

features

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

feature importance

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

feature transformation

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

few-shot prompting

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

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

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

flash-cut migration

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

FM

See foundation model.

foundation model (FM)

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

G

generative AI

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

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

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

Gitflow workflow

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

golden image

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

greenfield strategy

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

guardrail

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

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

H

HA

See high availability.

heterogeneous database migration

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

high availability (HA)

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

historian modernization

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

holdout data

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

homogeneous database migration

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

hot data

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

hotfix

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

hypercare period

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

I

IaC

See infrastructure as code.

identity-based policy

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

idle application

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

IIoT

See industrial Internet of Things.

immutable infrastructure

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

inbound (ingress) VPC

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

incremental migration

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

Industry 4.0

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

infrastructure

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

infrastructure as code (IaC)

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

industrial Internet of Things (IIoT)

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

inspection VPC

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

Internet of Things (IoT)

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

interpretability

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

IoT

See Internet of Things.

IT information library (ITIL)

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

IT service management (ITSM)

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

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

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

landing zone

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

large language model (LLM)

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

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

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

lift and shift

See 7 Rs.

little-endian system

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

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

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

main branch

See branch.

malware

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

managed services

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

manufacturing execution system (MES)

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

MAP

See Migration Acceleration Program.

mechanism

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

member account

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

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

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

microservice

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

microservices architecture

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

Migration Acceleration Program (MAP)

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

migration at scale

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

migration factory

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

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

migration metadata

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

migration pattern

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

Migration Portfolio Assessment (MPA)

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

Migration Readiness Assessment (MRA)

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

migration strategy

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

ML

See machine learning.

modernization

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

modernization readiness assessment

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

monolithic applications (monoliths)

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

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

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

mutable infrastructure

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

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

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

OI

See operations integration.

OLA

See operational-level agreement.

online migration

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

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

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

operational-level agreement (OLA)

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

operational readiness review (ORR)

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

operational technology (OT)

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

operations integration (OI)

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

organization trail

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

organizational change management (OCM)

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

origin access control (OAC)

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

origin access identity (OAI)

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

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

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

P

permissions boundary

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

personally identifiable information (PII)

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

PII

See personally identifiable information.

playbook

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

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

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

polyglot persistence

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

portfolio assessment

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

predicate

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

predicate pushdown

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

preventative control

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

principal

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

privacy by design

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

private hosted zones

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

proactive control

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

product lifecycle management (PLM)

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

production environment

See environment.

programmable logic controller (PLC)

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

prompt chaining

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

pseudonymization

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

publish/subscribe (pub/sub)

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

Q

query plan

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

query plan regression

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

R

RACI matrix

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

RAG

See Retrieval Augmented Generation.

ransomware

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

RASCI matrix

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

RCAC

See row and column access control.

read replica

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

re-architect

See 7 Rs.

recovery point objective (RPO)

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

recovery time objective (RTO)

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

refactor

See 7 Rs.

Region

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

regression

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

rehost

See 7 Rs.

release

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

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

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

resource-based policy

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

responsible, accountable, consulted, informed (RACI) matrix

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

responsive control

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

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

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

rotation

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

row and column access control (RCAC)

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

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

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

S

SAML 2.0

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

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

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

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

security by design

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

security control

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

security hardening

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

security information and event management (SIEM) system

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

security response automation

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

server-side encryption

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

service control policy (SCP)

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

service endpoint

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

service-level agreement (SLA)

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

service-level indicator (SLI)

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

service-level objective (SLO)

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

shared responsibility model

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

SIEM

See security information and event management system.

single point of failure (SPOF)

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

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

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

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

SPOF

See single point of failure.

star schema

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

strangler fig pattern

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

subnet

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

supervisory control and data acquisition (SCADA)

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

symmetric encryption

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

synthetic testing

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

system prompt

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

T

tags

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

target variable

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

task list

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

test environment

See environment.

training

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

transit gateway

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

trunk-based workflow

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

trusted access

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

tuning

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

two-pizza team

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

U

uncertainty

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

undifferentiated tasks

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

upper environments

See environment.

V

vacuuming

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

version control

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

VPC peering

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

vulnerability

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

W

warm cache

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

warm data

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

window function

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

workload

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

workstream

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

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

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

Z

zero-day exploit

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

zero-day vulnerability

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

zero-shot prompting

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

zombie application

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