A phased approach for performance engineering in the AWS Cloud

AWS Prescriptive Guidance

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Prescriptive Guidance: A phased approach for performance engineering in the AWS Cloud

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

Introduction **1**
What is performance engineering? 1
Why use performance engineering? 1
Performance engineering pillars **2**
Test-data generation 3
Test-data generation tools 5
Test observability 5
Logging 7
Monitoring 11
Tracing 14
Test automation 17
Test-automation tools 18
Test reporting 19
Standardized recording 20
Performance pillars example **21**
Resources **23**
Contributors **25**
Document history **26**
Glossary **27**
..... 27
A 28
B 31
C 33
D 36
E 40
F 42
G 44
H 45
I 46
L 48
M 50
O 54
P 56
Q 59

R 59
S 62
T 66
U 67
V 68
W 68
Z 69

A phased approach for performance engineering in the AWS Cloud

Amazon Web Services (contributors)

April 2024 (document history)

This guide outlines the best practices for planning, building, and enabling performance engineering for application workloads running on Amazon Web Services (AWS). It lays out four pillars for performance engineering, and it suggests different approaches to meet applications' performance requirements. For each pillar, this guide lists tools and solutions for setting up performance tests and the testing environment.

What is performance engineering?

Performance Engineering encompasses the techniques applied during a system's development lifecycle to ensure the non-functional performance requirements (such as throughput, latency, or memory usage) are met.

Before performance testing starts, you need to set up the performance environment. A typical performance environment stands on the following pillars:

- Test-data generation
- Test observability
- Test automation
- Test reporting

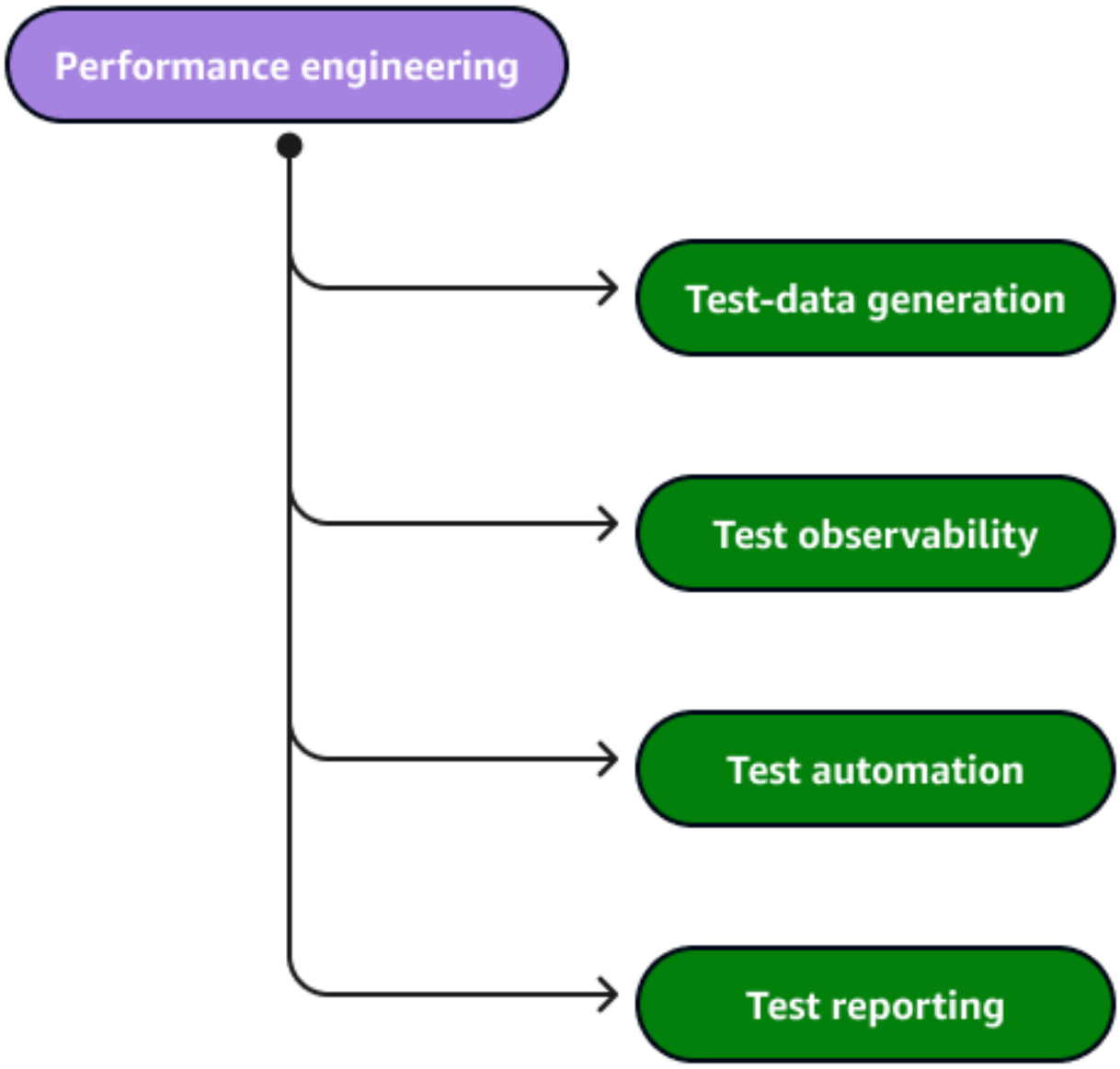
Why use performance engineering?

Performance engineering is the process of continuously optimizing the application performance from the start of the design phase. It brings great value to the business by avoiding rework and refactoring of code at a later stage in the development cycle. Beginning performance engineering at the design phase results in an application that performs better because performance can be factored into the design. Performance engineering requires the active participation of system architects, developers, DevOps, and Quality Assurance.

The pillars of performance engineering

To enable a performance engineering mindset, it's important to build a strong foundation while setting up performance engineering for the application. Performance engineering requires setting up four major pillars:

- **Test-data generation** – Performance engineers set up tools to generate the test data.
- **Test observability** – Performance engineers set up the observability environment to ensure that the performance run can be logged and traced, and that the resources handling the loads are monitored.
- **Test automation** – Performance engineers develop automated tests that simulate user traffic and system load using tools such as Apache JMeter or ghz.
- **Test reporting** – Data is gathered about the configuration of each test run along with the performance results. The data enables correlating configuration changes to performance and provides valuable insights.



Incorporating these pillars will encourage the performance mindset starting from the initial phases of the design. This will help avoid changes to the application or environment in later phases of development and testing.

Test-data generation

Test-data generation involves generating and maintaining a large amount of data for running the performance test case. This generated data acts as an input to the test cases so that the application can be tested on a diverse set of data.

Often, generating test data is a complex process. However, using poorly created dataset can lead to unpredictable application behavior in the production environment. Test-data generation for performance testing differs from traditional test-data generation approaches. It requires real-world scenarios, and most customers want to test their workloads with data that is similar to their actual production data. Generated test data also usually needs to be reset or refreshed into its original state after each test run, which adds to the time and effort.

Test-data generation includes the following major considerations:

- **Accuracy** – Accuracy of the data is important in all aspects of testing. Inaccurate data creates inaccurate results. For example, when a credit card transaction is generated, it should not be for a date in the future.
- **Validity** – The data should be valid for the use case. For example, while testing credit card transactions, it's not advisable to generate 10,000 transactions per user per day, because this deviates significantly from the valid use case scenario.
- **Automation** – Automation of test-data generation can bring time effort benefits. It also leads to effective test automation. Generating test-data manually can have consequences with respect to the quality and time effort requirements.

  There are different mechanism one can adopt based on the use cases as follows:

  - **API driven** – In this case, the developer provides a test-data generation API that the tester can consume to generate data. Using testing tools such as JMeter, testers can scale the data generation using a business API. For example, if you have an API to add a user, you can use the same API to create hundreds of user with different profiles. Similarly, you can delete the users by calling the delete API operation. For complex work flow applications, the developer can provide a composite API that can generate datasets across different components. Using this approach, testers can write automation to generate and delete the datasets based on their requirements.

    However, if the system is complex or the API response time per invocation is high, it might take a long time to set up and tear down the data.
  - **SQL statement driven** – An alternate approach is to use backend SQL statements to generate a large volume of data. The developer can provide template-based SQL statements for test-data generation. Testers can consume the statements to populate data, or they can create wrapper scripts on top of these statements for automating test-data generation. Using this approach, testers can populate and tear down data very quickly if the data needs to be reset after the test is completed. However, this approach requires direct access to the database

of the application, which might not be possible in typical secured environment. In addition, invalid queries might result in incorrect data population, which can produce skewed results. Developers must also continually update SQL statements in the application code to reflect changes made to the application over time.

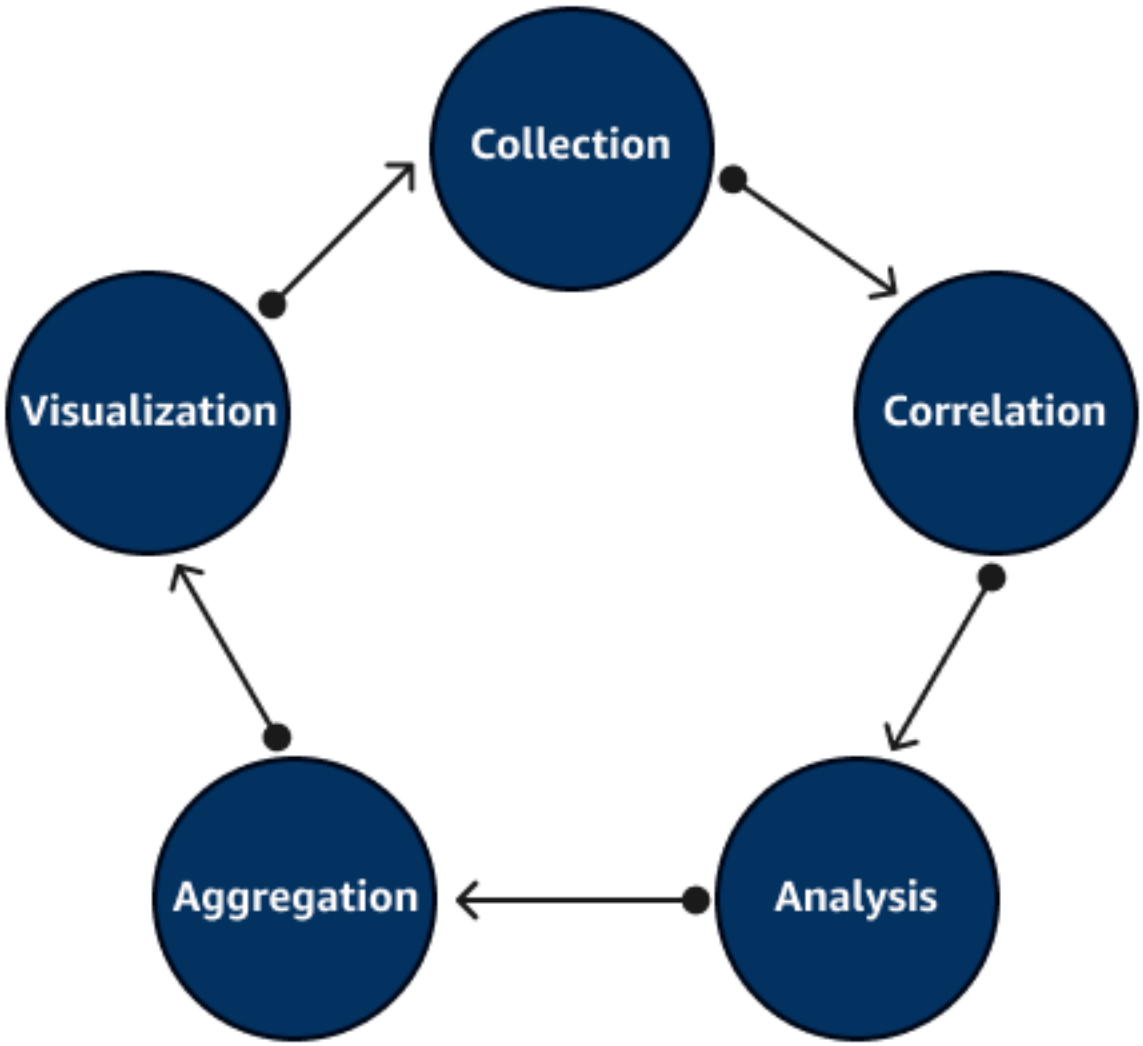
Test-data generation tools

AWS provides native custom tools that you can use for test-data generation:

- **Amazon Kinesis Data Generator** – The Amazon Kinesis Data Generator (KDG) simplifies the task of generating data and sending it to Amazon Kinesis. The tool provides a user-friendly UI that runs directly in your browser. For more information and a reference implementation, see the Test Your Streaming Data Solution with the New Amazon Kinesis Data Generator blog post.
- **AWS Glue Test Data Generator** – The AWS Glue Test Data Generator provides a configurable framework for test-data generation using AWS Glue PySpark serverless jobs. The required test-data description is fully configurable through a YAML configuration file. For more information and a reference implementation, see the AWS Glue Test Data Generator GitHub repository.

Test observability

Test observability supports collecting, correlating, aggregating, and analyzing telemetry in your network, infrastructure, and applications during the performance test runs. You gain full insights into the behavior, performance, and health of your system. These insights help you detect, investigate, and remediate problems faster. By adding artificial intelligence and machine learning, you can proactively react to, predict, and prevent problems.



Observability relies on logging , monitoring, and tracing. The responsibility of implementing these activities successfully spans the application and infrastructure teams.

At the start of the design phase, application teams should understand the current state of their observability stack, including logging, monitoring, and tracing. They can then choose tools integrate more smoothly into the observability stack.

Similarly, the infrastructure team is responsible for managing and scaling the observability infrastructure.

Consider the following aspects with respect to test observability:

- Availability of application logs and traces
- Correlation of logs and traces

- Availability of nodes, containers, and application metrics
- Automation to set up and update the observability infrastructure on demand
- Ability to visualize the telemetry
- Scaling of the observability infrastructure

Logging

Logging is the process of keeping data about events that occur in a system. The log can include problems, errors, or information about the current operation. Logs can be classified into different types, such as the following:

- Events log
- Server log
- System log
- Authorization and access logs
- Audit logs

A developer can search the logs for specific error codes or patterns, filter them based on specific fields, or archive them securely for future analysis. Logs help the developer to perform root cause analysis for performance issues and also to correlate between system components.

Building an effective logging solution involves close coordination between the application and infrastructure teams. Application logs are not useful unless there is a scalable logging infrastructure that supports use cases such as parsing, filtering, buffering, and correlation of logs. Common use cases, such as generating a correlation ID, logging run time for business-critical methods, and defining log patterns, can be simplified.

Application team

An application developer must ensure that the logs generated follow logging best practices. Best practices include the following:

- Generating correlation IDs to track unique requests
- Logging the time taken by business-critical methods
- Logging at an appropriate log level
- Sharing a common logging library

When you design applications that interact with different microservices, use these logging design principles to simplify filtering and log extraction on the backend.

Generating correlation IDs to track unique requests

When the application receives the request, it can check whether a correlation ID is already present in the header. If an ID isn't present, the application should generate an ID. For example, an Application Load Balancer adds a header called `X-Amzn-Trace-Id`. The application can use the header to correlate the request from the load balancer to the application. Similarly, the application should inject `traceId` if calling dependent microservices so that logs generated by different components in a request flow are correlated.

Logging the time taken by business-critical methods

When the application receives a request, it interacts with a different component. The application should log the time taken for business-critical methods in a defined pattern. This can make it easier to parse the logs in the backend. It can also help you to generate useful insights from the logs. You can use approaches such as aspect-oriented programming (AOP) to generate such logs so that you can separate logging concerns from your business logic.

Logging at an appropriate log level

The application should write logs that have a helpful amount of information. Use log levels to categorize events by their severity. For example, use `WARNING` and `ERROR` levels for important events that need investigating. Use `INFO` and `DEBUG` for detailed tracing and high-volume events. Set log handlers to capture only the levels that are necessary in production. Generating too much logging at the `INFO` level isn't helpful, and it adds pressure in the backend infrastructure. `DEBUG` logging can be useful, but it should be used cautiously. Using `DEBUG` logs can generate a large volume of data, so it isn't recommended in performance-testing environment.

Sharing a common logging library

The application teams should use a common logging library, such as AWS SDK for Java, with a predefined common logging pattern that developers can be use as dependencies in their project.

Infrastructure team

DevOps engineers can reduce effort by using the following logging design principles when filtering and extracting logs on the backend. The infrastructure team must set up and support the following resources.

Log agent

A log agent (log shipper) is a program that reads logs from one location and sends them to another location. Log agents are used to read log files stored on a computer and upload log events to the backend for centralization.

Logs are unstructured data that must be structured before you can make meaningful insights from them. Log agents use parsers to read log statements and extract relevant fields such as timestamp, log level, and service name, and they structure that data into a JSON format. Having a lightweight log agent at the edge is useful because it leads to less resource utilization. The log agent can directly push to the backend, or it can use an intermediary log forwarder that pushes the data to the backend. Using a log forwarder offloads the work from the log agents at the source.

Log parser

A log parser converts the unstructured logs into structured logs. Log agent parsers also enrich the logs by adding metadata. Data parsing of the data can be done at the source (application end) or it can be done centrally. The schema for storing the logs should be extensible so that you can add new fields. We recommend using standard log formats such as JSON. However, in some cases, the logs must be transformed to JSON formats for better searching. Writing the right parser expression enables efficient transformation.

Logs backend

A logs backend service collects, ingests, and visualizes log data from various sources. The log agent can directly write to the backend or use an intermediary log forwarder. While performance testing, be sure to store the logs so that they can be searched at a later time. Store logs in the backend separately for each application. For example, use a dedicated index for an application, and use index pattern to search for logs that are spread across different related applications. We recommend saving at least 7 days of data for log searching. However, storing the data for a longer duration can result in unnecessary storage costs. Because a large volume of logs are generated during the performance test, it's important for the logging infrastructure to scale and right-size the logging backend.

Log visualization

To gain meaningful and actionable insights from application logs, use dedicated visualization tools to process and transform the raw log data into graphical representations. Visualizations such as charts, graphs, and dashboards can help uncover trends, patterns, and anomalies that might not be readily apparent when looking at the raw logs.

Key benefits of using visualization tools include the ability to correlate data across multiple systems and applications to identify dependencies and bottlenecks. Interactive dashboards support drilling down into the data at different levels of granularity to troubleshoot issues or spot usage trends. Specialized data visualization platforms provide capabilities such as analytics, alerting, and data sharing that can enhance monitoring and analysis.

By using the power of data visualization on application logs, development and operations teams can gain visibility into system and application performance. The insights derived can be used for a variety of purposes, including optimizing efficiency, improving user experience, enhancing security, and capacity planning. The end result is dashboards tailored to various stakeholders, providing at-a-glance views that summarize log data into actionable and insightful information.

Automating the logging infrastructure

Because different application have different requirements, it's important to automate the installation and operations of the logging infrastructure. Use infrastructure as code (IaC) tools to provision the logging infrastructure's backend. Then you can provision the logging infrastructure either as a shared service or as an independent bespoke deployment for a particular application.

We recommend that developers use continuous delivery (CD) pipelines to automate the following:

- Deploy the logging infrastructure on demand and tear it down when it isn't required.
- Deploy log agents across different targets.
- Deploy log parser and forwarder configurations.
- Deploy application dashboards.

Logging tools

AWS provides native logging, alarming, and dashboard services. The following are popular AWS services and resources for logging:

- Amazon OpenSearch Service helps organizations collect, ingest, and visualize log data from various sources. For more information, see Centralized Logging with OpenSearch.
- Amazon CloudWatch agent and AWS for Fluent Bit are the most popular log agents on AWS. For information about using the CloudWatch agent with Amazon CloudWatch Logs Insights, see the blog post Simplifying Apache server logs with Amazon CloudWatch Logs Insights. For AWS for Fluent Bit reference implementation, see the blog post Centralized Container Logging with Fluent Bit.

Monitoring

Monitoring is the process of collecting different metrics, such as CPU and memory, and storing them in a time-series database such as Amazon Managed Service for Prometheus. The monitoring system can be push based or pull based. In push-based systems, the source pushes metrics periodically to the time-series data base. In pull-based systems, the scraper scrapes metrics from various sources and stores them in the time-series database. Developers can analyze the metrics, filter the metrics, and plot them over time to visualize performance. Implementing monitoring successfully can be split into two broad areas: application and infrastructure.

For application developers, the following metrics are critical:

- **Latency** – The time taken to receive a response
- **Request throughput** – The total number of requests handled per second
- **Request error rate** – The total number of errors

Capture resource utilization, saturation, and error counts for each resource (such as the application container, the database) that's involved in the business transaction. For example, when monitoring CPU usage, you can track average CPU utilization, average load, and peak load during the performance-test run. When a resource reaches saturation during stress testing, but it might not reach saturation during a performance run for a shorter period of time.

Metrics

Applications can use different actuators, such as spring boot actuators, to monitor their applications. These production-grade libraries generally expose a REST endpoint for monitoring information about the running applications. The libraries can monitor the underlying infrastructure, application platforms, and other resources. If any of the default metrics don't meet the requirements, the developer must implement custom metrics. Custom metrics can help track business key performance indicators (KPIs) that can't be tracked through data from default implementations. For example, you might want to track a business operation such as third-party API integration latency or the total number of transactions completed.

Cardinality

Cardinality refers to number of unique time-series of a metric. Metrics are labeled to provide additional information. For example, a REST-based application that tracks the request count for a particular API indicates a cardinality of 1. If you add a user label to identify the request count per

user, the cardinality increases proportionally to the number of users. By adding labels that create cardinality, you can slice and dice metrics by various groups. It's important to use the right labels for the right use case because cardinality increases the number of metrics series in the backend monitoring time-series database.

Resolution

In a typical monitoring setup, the monitoring application is configured to scrape the metrics from the application periodically. The periodicity of scraping defines the granularity of the monitoring data. Metrics collected at shorter interval tends to provide a more accurate view of the performance because more data points are available. However, the load on the time-series database increases as more entries are stored. Typically a granularity of 60 seconds is standard resolution and 1 second is high resolution.

DevOps team

Application developers often ask DevOps engineers to set up a monitoring environment for visualizing metrics of the infrastructure and applications. The DevOps engineer must set up an environment that is scalable and supports the data-visualization tools used by the application developer. This involves scraping monitoring data from different sources and sending the data to a central time-series database such as Amazon Managed Service for Prometheus.

Monitoring backend

A monitoring backend service supports the collection, storage, querying, and visualization of metrics data. It's typically a time-series database such as Amazon Managed Service for Prometheus or InfluxData InfluxDB. Using a service-discovery mechanism, the monitoring collector can collect metrics from different sources and store them. While performance testing, it's important to store the metrics data so that it can be searched at a later time. We recommend saving at least 15 days of data for metrics. However, storing the metrics for a longer duration doesn't add significant benefits and leads to unnecessary storage costs. Because the performance test can generate a large volume of metrics, it's important for the metrics infrastructure to scale while providing fast query performance. The monitoring backend service provides a query language which can be used to view the metrics data.

Visualization

Provide visualization tools that can display the application data to provide meaningful insights. The DevOps engineer and the application developer should learn the query language for the monitoring backend and work closely to generate a dashboard template that can be reused. On

the dashboards, include latency, and errors while also displaying resource utilization and saturation across the infrastructure and the application resources.

Automating the monitoring infrastructure

Similar to logging, it's important to automate installation and operation of the monitoring infrastructure so that you can accommodate the different requirements of different applications. Use IaC tools to provision the monitoring infrastructure's backend. Then you can provision the monitoring infrastructure either as a shared service or as an independent bespoke deployment for a particular application.

Use CD pipelines to automate the following:

- Deploy the monitoring infrastructure on demand and tear it down when it isn't required.
- Update the monitoring configuration to filter or aggregate metrics.
- Deploy application dashboards.

Monitoring tools

Amazon Managed Service for Prometheus is a Prometheus-compatible monitoring service for container infrastructure and application metrics for containers that you can use to securely monitor container environments at scale. For more information, see the blog post Getting Started with Amazon Managed Service for Prometheus.

Amazon CloudWatch provides full-stack monitoring on AWS. CloudWatch supports both AWS native and open source solutions so that you can understand what is happening across your technology stack at any time.

Native AWS tools include the following:

- Amazon CloudWatch dashboards
- CloudWatch Container Insights
- CloudWatch metrics
- CloudWatch alarms

Amazon CloudWatch offers purpose-built features that address specific use cases such as container monitoring through CloudWatch Container Insights. These features are built-into CloudWatch so that you can set up logs, metrics collection, and monitoring.

For your containerized applications and microservices, use Container Insights to collect, aggregate, and summarize metrics and logs. Container Insights is available for Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and Kubernetes platforms on Amazon Elastic Compute Cloud (Amazon EC2). Container Insights collects data as performance log events in the embedded metric format. These performance log event entries use a structured JSON schema that supports high-cardinality data ingestion and storage at scale.

For information about implementing Container Insights with Amazon EKS, see the blog post Introducing Amazon CloudWatch Container Insights for Amazon EKS Fargate using AWS Distro for OpenTelemetry.

Tracing

Tracing involves the specialized use of logging information about a program's processes. Insights from the logs can help engineers debug individual transactions and identify bottlenecks. Tracing can be enabled automatically or by using manual instrumentation.

Because an application integrates with different services, it's important to identify how the application and its underlying services are performing. Tracing works with traces and spans. A *trace* is the complete request process, and each trace is made up of spans. A *span* is a tagged time interval and is the activity within a system's individual components or services. Traces provide the big picture of what happens when a request is made to an application.

Application team

Application developers instrument their applications by sending trace data for inbound and outbound requests and other events within the application, along with metadata about each request. To generate traces, an application must be instrumented to generate traces. Instrumentation can be automatic or manual.

Automatic instrumentation

You can collect telemetry from an application by using automatic instrumentation without having to modify the source code. Automatic instrumentation agents can generate application traces of an application or service. Typically, you use configuration changes to add the agent or another mechanism.

Library instrumentation involves making minimal application code changes to add prebuilt instrumentation. The instrumentation targets specific libraries or frameworks, such as the AWS SDK, Apache HTTP clients, or SQL clients.

Manual instrumentation

In this approach, application developers add instrumentation code to the application at each location where they want to collect trace information. For example, use aspect-oriented programming (AOP) to collect AWS X-Ray tracing data. Developers can use SDKs to instrument their applications.

Sampling

Trace data is often generated in large volumes. It's important to have a mechanism to determine whether the trace data should be exported or not. Sampling is the process of determining what data should be exported. This is generally done to save cost. By customizing sampling rules, you can control the amount of data that you record. You can also change sampling behavior without changing and redeploying your code. It's important to control the sampling rate to generate the right amount of traces.

Application developers can annotate the traces by adding metadata as key-value pairs. The annotations enrich the traces and help to refine filtering in the backend.

DevOps team

DevOps engineers are often asked to set up a tracing environment for the application developer to visualize traces for infrastructure and applications. Tracing environment setup involves collecting trace data from different sources and sending it to a central store for visualizing.

Tracing backend

A tracing backend is a service such as AWS X-Ray that collects data about requests that your application serves. It provides tools that you can use to view, filter, and gain insights into that data to identify issues and opportunities for optimization. For any traced request to your application, you can see detailed information about the request and response, and about other calls that your application makes to downstream AWS resources, microservices, databases, and web APIs.

Automating tracing

Because different applications have different tracing requirements, it's important to automate the configuration and operation of the tracing infrastructure. Use IaC tools to provision the tracing infrastructure's backend.

Use CD pipelines to automate the following:

- Deploy the tracing infrastructure on demand and tear down it when it isn't required.
- Deploy the tracing configuration across applications.

Tracing tools

AWS provides the following services for tracing and its associated visualization:

- AWS X-Ray receives traces from your application, in addition to traces from AWS services your application uses that are already integrated with X-Ray. There are several SDKs, agents, and tools that can be used to instrument your application for X-Ray tracing. For more information, see the AWS X-Ray documentation.

  Developers can also use AWS X-Ray SDKs to send traces to X-Ray. AWS X-Ray provides SDKs for Go, Java, Node.js, Python, .NET, and Ruby. Each X-Ray SDK provides the following:

  - Interceptors to add to your code to trace incoming HTTP requests
  - Client handlers to instrument AWS SDK clients that your application uses to call other AWS services
  - An HTTP client to instrument calls to other internal and external HTTP web services

  X-Ray SDKs also support instrumenting calls to SQL databases, automatic AWS SDK client instrumentation, and other features. Instead of sending trace data directly to X-Ray, the SDK sends JSON segment documents to a daemon process listening for UDP traffic. The X-Ray daemon buffers segments in a queue and uploads them to X-Ray in batches. For more information about instrumenting your application by using an X-Ray SDK, see the X-Ray documentation.
- Amazon OpenSearch Service is an AWS managed service for running and scaling OpenSearch clusters, which can be used to centrally store logs, metrics, and traces. The Observability plugin provides a unified experience for collecting and monitoring metrics, logs, and traces from common data sources. Data collection and monitoring in one place provides full-stack, end-to-end observability of your entire infrastructure. For implementation information, see the OpenSearch Service documentation.
- AWS Distro for OpenTelemetry (ADOT) is an AWS distribution based on the Cloud Native Computing Foundation (CNCF) OpenTelemetry project. ADOT currently includes automatic-instrumentation support for Java and Python. In addition, ADOT supports automatic instrumentation of AWS Lambda functions and their downstream requests using Java, Node.js, and Python runtimes, through ADOT Managed Lambda Layers. Developers can use the ADOT

collector to send traces to different backends, including AWS X-Ray and Amazon OpenSearch Service.

For a reference example of how to instrument your application by using the ADOT SDK, see the documentation. For a reference example of how to use the ADOT SDK to send data to Amazon OpenSearch Service, see the OpenSearch Service documentation.

For a reference example of how to instrument your application running on Amazon EKS, see the blog post Metrics and traces collection using Amazon EKS add-ons for AWS Distro for OpenTelemetry.

Test automation

Automated testing with a specialized framework and tools can reduce human intervention and maximize quality. Automated performance testing is no different from automation tests such as unit testing and integration testing.

Use DevOps pipelines in the different stages for performance testing.

The five stages for the test automation pipeline are:

1. **Set up** – Use the test-data approaches described in the Test-data generation section for this stage. Generating realistic test data is critical to obtain valid test results. You must carefully create diverse test data that covers a wide range of use cases and closely matches live production data. Before running full-scale performance tests, you might need to run initial trial tests to validate the test scripts, environments, and monitoring tools.
2. **Test tool** – To conduct the performance testing, select an appropriate load-testing tool, such as JMeter or ghz. Consider the best fit for your business needs in terms of simulating real-world user loads.
3. **Test run** – With the test tools and environments established, run end-to-end performance tests across a range of expected user loads and durations. Throughout the test, closely monitor the health of the system being tested. This is typically a long-running stage. Monitor error rates for automatic test invalidation, and stop the test if there are too many errors.

The load-testing tool provides insights into resource utilization, response times, and potential bottlenecks.

4. **Test reporting** – Collect the test results along with application and test configuration. Automate collection of application configuration, test configuration, and results, which helps with recording the performance test–related data and storing it centrally. Maintaining performance data centrally helps with providing good insights and supports defining success criteria programmatically for your business.
5. **Clean up** – After you complete a performance test run, reset the test environment and data to prepare for subsequent runs. First, you revert any changes made to the test data during the run. You must restore the databases and other data stores to their original state, reverting any new, updated, or deleted records generated during the test.

You can reuse the pipeline to repeat the test multiple times until the results reflect the performance that you want. You can also use the pipeline to validate that code changes don't break performance. You can run code-validation tests in off-business hours and use the test and observability data available for troubleshooting.

Best practices include the following:

- Record the start and end time, and automatically generate URLs for logging, This helps you to filter observability data in that appropriate time window. monitoring, and tracing systems.
- Inject test identifiers in the header while invoking the tests. Application developers can enrich their logging, monitoring, and tracing data by using the identifier as a filter in the backend.
- Limit the pipeline to only one run at a time. Running concurrent tests generates noise that can cause confusion during troubleshooting. It's also important to run the test in a dedicated performance environment.

Test-automation tools

Testing tools play an important part in any test automation. Popular choices for open source testing tools include the following:

- Apache JMeter is the seasoned power horse. Over the years, Apache JMeter has become more reliable and has added features. With the graphical interface, you can create complex tests without knowing a programming language. Companies such as BlazeMeter support Apache JMeter.

- K6 is a free tool that offers support, hosting of the load source, and an integrated web interface to organize, run, and analyze load tests.
- The Vegeta load test follows a different concept. Instead of defining concurrency or throwing load at your system, you define a certain rate. The tool then creates that load independent of your system's response times.
- Hey and ab, the Apache HTTP server bench marking tool, are basic tools that you can use from the command line to run the specified load on a single endpoint. This is the fastest way to generate load if you have a server to run the tools on. Even a local laptop will perform, although it might be not powerful enough to produce high load.
- ghz is a command line utility and Go package for load testing and bench marking gRPC services.

AWS provides the Distributed Load Testing on AWS solution. The solution creates and simulates thousands of connected users generating transactional records at a constant pace without the need to provision servers. For more information, see the AWS Solutions Library.

You can use AWS CodePipeline to automate the performance testing pipeline. For more information about automating your API testing by using CodePipeline, see the AWS DevOps Blog and the AWS documentation.

Test reporting

Test reporting refers to the collection, analysis, and presentation of data related to the performance of systems, applications, services, or processes. It involves measuring various metrics and indicators to assess the efficiency, responsiveness, reliability, and overall effectiveness of a particular system or component.

Performance-test reporting involves choosing relevant metrics based on the context and goals of the analysis. Common performance metrics include response times, throughput, error rates, resource utilization (CPU, memory, disk), and network latency.

After the performance-related data has been collected, it needs to be stored in a central repository. These test results could come from different environments, applications, and testing tools. When you have multiple workloads running in different environments, it's difficult to gather performance-related data and correlate between these data points to draw informed conclusions. We recommend defining a standard method for collecting performance metrics data using a central repository for data storage and visualization.

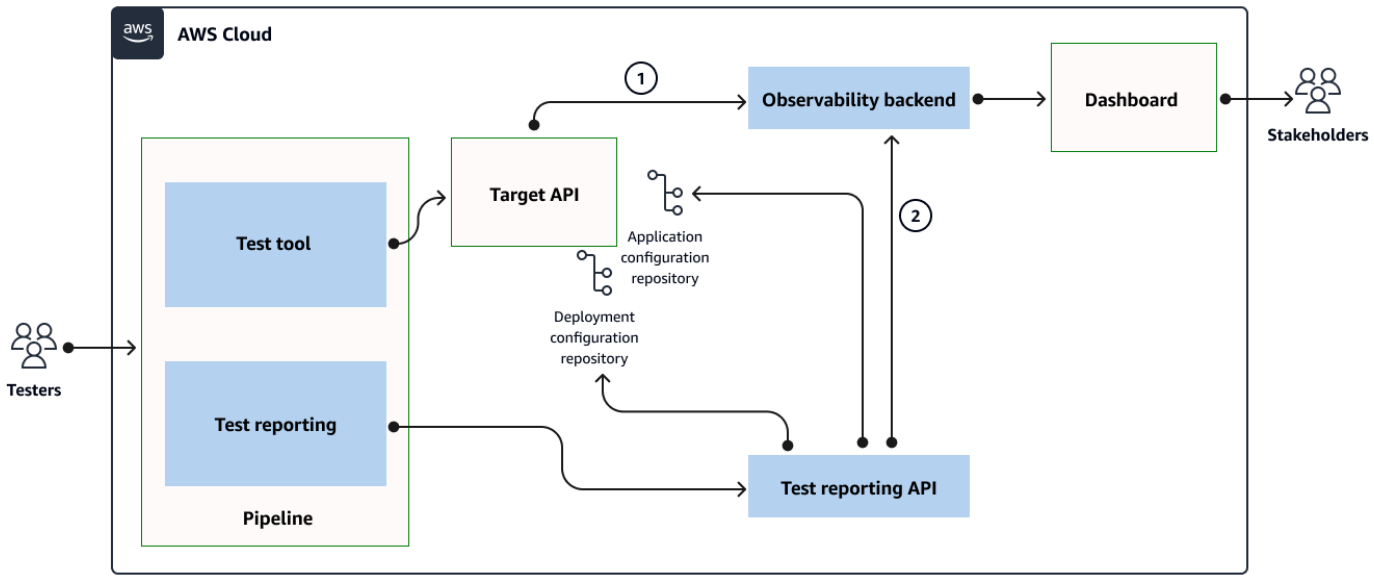
Standardized recording

We recommend standardizing the way that different stakeholders perform the performance tests and write the resulting data to a central repository. For example, this could take the form of an API accepting the results and storing them into a persistent storage solution. In situations where data needs to be fetched from sources such as GitOps or Amazon Managed Service for Prometheus, the API can directly pull those details from the specified sources based on schema files that describe how to extract the fields from deployment specifications and Kubernetes specifications. The schema files can use `JSONPath` expressions or Prometheus Query Language (PromQL). As mentioned previously, the metrics that are collected should be relevant to the context and goals of the performance analysis.

The data passed to the API can include details and tags related to the application and the environment for which the test has been performed. This helps with performing analytics on the performance testing data.

Performance engineering pillars in action

The following reference architecture demonstrates performance engineering pillars for testing a specific API.



1. Logging, monitoring, and tracing data is sent from the target API to the backend.
2. When invoked, the test reporting API sends results and configuration information to the backend.

The core component is the target API or application under test. The target API syncs with the application configuration repository and deployment configuration repository in GitOps fashion to obtain the latest application and infrastructure configurations. This synchronization allows the automated tests to run against the current desired state of the application and its supporting infrastructure as defined in the Git repositories.

The test-automation pipeline automates generating the test data, running the test, and reporting the test results for the target API.

The target API generates performance insights (metrics, logs, and traces), using observability best practices, and it streams metrics data to the observability backend.

The test reporting API collects all test-related reporting data (configuration and test results), and it stores them in the observability backend.

Aggregation of performance insights and reporting data (configuration, test results) helps you to query performance related data for the target API. For example, you might ask the following:

- What are the top ten slowest transactions?
- What is the P99, P90, average number of each test?
- How do the configurations of the two test runs compare?

Correlating tests cases with results, configurations, and metrics over a period time helps with identifying the best configuration and the performance results.

Using these test results, you can make more precise, data-driven decisions for the API and have confidence when taking the API to production.

Resources

AWS services

- Amazon CloudWatch
- AWS CodePipeline
- AWS Distro for OpenTelemetry
- Amazon OpenSearch Service
- AWS X-Ray

Implementations

- amazon-kinesis-data-generator
- AWS Glue Test Data Generator
- Distributed Load Testing on AWS

Blog posts

- Centralized Container Logging with Fluent Bit
- Test Your Streaming Data Solution with the New Amazon Kinesis Data Generator
- Introducing Amazon CloudWatch Container Insights for Amazon EKS Fargate using AWS Distro for OpenTelemetry
- Application Tracing on Kubernetes with AWS X-Ray
- Metrics and traces collection using Amazon EKS add-ons for AWS Distro for OpenTelemetry
- Getting Started with Amazon Managed Service for Prometheus

Workshop

- Introduction to AWS Observability

AWS Prescriptive Guidance

- Load testing applications (guide)

Third-party applications

- Apache JMeter
- K6
- Vegeta
- Hey and ab
- ghz

Contributors

Contributors to this document include:

- Varun Sharma, Sr. Lead Consultant, AWS
- Akash Kumar, Sr. Lead Consultant, AWS
- Archana Bhatnagar, Practice Manager, AWS
- Pratik Sharma, Professional Services II, AWS

Document history

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

Change	Description	Date
Initial publication	—	April 24, 2024

AWS Prescriptive Guidance glossary

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

Numbers

7 Rs

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

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

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

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

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

active-passive migration

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

aggregate function

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

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

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

anti-pattern

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

application control

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

application portfolio

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

artificial intelligence (AI)

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

artificial intelligence operations (AIOps)

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

asymmetric encryption

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

atomicity, consistency, isolation, durability (ACID)

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

attribute-based access control (ABAC)

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

authoritative data source

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

Availability Zone

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

AWS Cloud Adoption Framework (AWS CAF)

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

AWS Workload Qualification Framework (AWS WQF)

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

B

bad bot

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

BCP

See business continuity planning.

behavior graph

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

big-endian system

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

binary classification

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

bloom filter

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

blue/green deployment

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

bot

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

botnet

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

branch

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

break-glass access

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

brownfield strategy

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

buffer cache

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

business capability

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

business continuity planning (BCP)

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

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

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

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

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

chaos engineering

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

CI/CD

See continuous integration and continuous delivery.

classification

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

client-side encryption

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

Cloud Center of Excellence (CCoE)

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

cloud computing

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

cloud operating model

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

cloud stages of adoption

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

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

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

CMDB

See configuration management database.

code repository

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

cold cache

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

cold data

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

computer vision (CV)

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

configuration drift

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

configuration management database (CMDB)

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

conformance pack

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

continuous integration and continuous delivery (CI/CD)

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

CV

See computer vision.

D

data at rest

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

data classification

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

data drift

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

data in transit

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

data mesh

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

data minimization

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

data perimeter

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

data preprocessing

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

data provenance

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

data subject

An individual whose data is being collected and processed.

data warehouse

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

database definition language (DDL)

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

database manipulation language (DML)

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

DDL

See database definition language.

deep ensemble

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

deep learning

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

defense-in-depth

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

delegated administrator

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

deployment

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

development environment

See environment.

detective control

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

development value stream mapping (DVSM)

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

digital twin

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

dimension table

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

disaster

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

disaster recovery (DR)

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

DML

See database manipulation language.

domain-driven design

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

DR

See disaster recovery.

drift detection

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

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

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

electronic data interchange (EDI)

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

encryption

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

encryption key

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

endianness

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

endpoint

See service endpoint.

endpoint service

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

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

enterprise resource planning (ERP)

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

envelope encryption

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

environment

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

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

epic

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

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

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

F

fact table

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

fail fast

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

fault isolation boundary

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

feature branch

See branch.

features

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

feature importance

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

feature transformation

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

few-shot prompting

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

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

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

flash-cut migration

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

FM

See foundation model.

foundation model (FM)

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

G

generative AI

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

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

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

Gitflow workflow

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

golden image

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

greenfield strategy

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

guardrail

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

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

H

HA

See high availability.

heterogeneous database migration

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

high availability (HA)

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

historian modernization

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

holdout data

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

homogeneous database migration

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

hot data

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

hotfix

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

hypercare period

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

I

IaC

See infrastructure as code.

identity-based policy

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

idle application

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

IIoT

See industrial Internet of Things.

immutable infrastructure

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

inbound (ingress) VPC

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

incremental migration

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

Industry 4.0

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

infrastructure

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

infrastructure as code (IaC)

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

industrial Internet of Things (IIoT)

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

inspection VPC

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

Internet of Things (IoT)

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

interpretability

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

IoT

See Internet of Things.

IT information library (ITIL)

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

IT service management (ITSM)

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

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

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

landing zone

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

large language model (LLM)

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

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

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

lift and shift

See 7 Rs.

little-endian system

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

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

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

main branch

See branch.

malware

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

managed services

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

manufacturing execution system (MES)

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

MAP

See Migration Acceleration Program.

mechanism

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

member account

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

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

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

microservice

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

microservices architecture

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

Migration Acceleration Program (MAP)

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

migration at scale

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

migration factory

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

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

migration metadata

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

migration pattern

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

Migration Portfolio Assessment (MPA)

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

Migration Readiness Assessment (MRA)

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

migration strategy

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

ML

See machine learning.

modernization

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

modernization readiness assessment

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

monolithic applications (monoliths)

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

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

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

mutable infrastructure

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

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

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

OI

See operations integration.

OLA

See operational-level agreement.

online migration

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

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

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

operational-level agreement (OLA)

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

operational readiness review (ORR)

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

operational technology (OT)

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

operations integration (OI)

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

organization trail

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

organizational change management (OCM)

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

origin access control (OAC)

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

origin access identity (OAI)

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

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

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

P

permissions boundary

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

personally identifiable information (PII)

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

PII

See personally identifiable information.

playbook

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

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

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

polyglot persistence

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

portfolio assessment

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

predicate

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

predicate pushdown

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

preventative control

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

principal

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

privacy by design

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

private hosted zones

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

proactive control

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

product lifecycle management (PLM)

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

production environment

See environment.

programmable logic controller (PLC)

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

prompt chaining

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

pseudonymization

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

publish/subscribe (pub/sub)

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

Q

query plan

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

query plan regression

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

R

RACI matrix

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

RAG

See Retrieval Augmented Generation.

ransomware

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

RASCI matrix

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

RCAC

See row and column access control.

read replica

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

re-architect

See 7 Rs.

recovery point objective (RPO)

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

recovery time objective (RTO)

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

refactor

See 7 Rs.

Region

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

regression

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

rehost

See 7 Rs.

release

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

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

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

resource-based policy

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

responsible, accountable, consulted, informed (RACI) matrix

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

responsive control

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

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

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

rotation

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

row and column access control (RCAC)

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

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

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

S

SAML 2.0

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

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

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

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

security by design

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

security control

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

security hardening

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

security information and event management (SIEM) system

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

security response automation

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

server-side encryption

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

service control policy (SCP)

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

service endpoint

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

service-level agreement (SLA)

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

service-level indicator (SLI)

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

service-level objective (SLO)

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

shared responsibility model

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

SIEM

See security information and event management system.

single point of failure (SPOF)

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

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

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

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

SPOF

See single point of failure.

star schema

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

strangler fig pattern

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

subnet

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

supervisory control and data acquisition (SCADA)

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

symmetric encryption

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

synthetic testing

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

system prompt

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

T

tags

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

target variable

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

task list

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

test environment

See environment.

training

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

transit gateway

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

trunk-based workflow

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

trusted access

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

tuning

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

two-pizza team

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

U

uncertainty

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

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.