Resilience lifecycle framework

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Resilience lifecycle framework

Table of Contents

Introduction **1**
Terms and definitions 2
Continuous resilience 3
Stage 1: Set objectives **4**
Mapping critical applications 4
Mapping user stories 5
Defining measurements 5
Creating additional measurements 6
Stage 2: Design and implement **8**
AWS Well-Architected Framework 8
Understanding dependencies 9
Disaster recovery strategies 9
Defining CI/CD strategies 10
Conducting ORRs 11
Understanding AWS fault isolation boundaries 11
Selecting responses 12
Resilience modeling 13
Failing safely 13
Stage 3: Evaluate and test **14**
Pre-deployment activities 14
Environment design 14
Integration testing 15
Automated deployment pipelines 15
Load testing 16
Post-deployment activities 16
Conducting resilience assessments 16
DR testing 17
Drift detection 17
Synthetic testing 17
Chaos engineering 18
Stage 4: Operate **19**
Observability 19
Event management 19
Continuous resilience 20

Stage 5: Respond and learn **21**
Creating incident analysis reports 21
Conducting operational reviews 22
Reviewing alarm performance 23
Alarm precision 23
False positives 23
False negatives 23
Duplicative alerts 24
Conducting metrics reviews 24
Providing training and enablement 24
Creating an incident knowledge base 24
Implementing resilience in depth 25
Conclusion and resources **26**
Contributors **27**
Document history **28**
Glossary **29**
..... 29
A 30
B 33
C 34
D 37
E 41
F 43
G 44
H 45
I 46
L 48
M 49
O 53
P 55
Q 57
R 57
S 60
T 63
U 64
V 65

W 65
Z 66

Resilience lifecycle framework: A continuous approach to resilience improvement

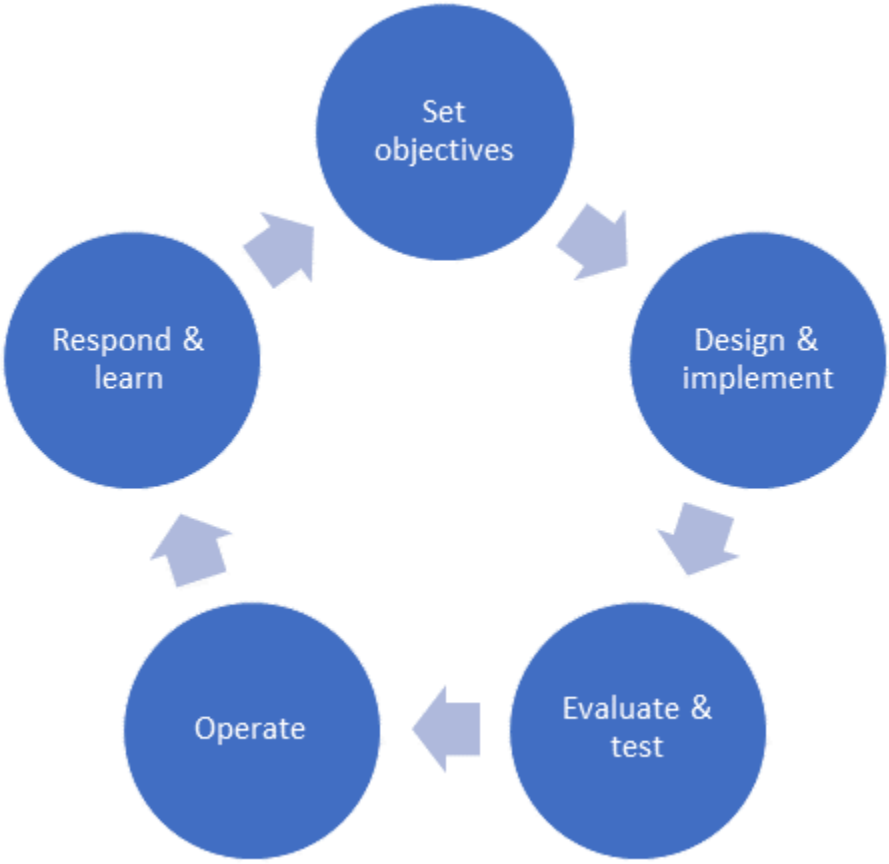
Amazon Web Services (AWS)

October 2023 (document history)

Modern organizations today face an ever-growing number of resilience-related challenges, especially as expectations from customers shift toward an *always on, always available* mindset. Remote teams and complex, distributed applications are coupled with an increasing need for frequent releases. As a result, an organization and its applications need to be more resilient than ever.

AWS defines resilience as the ability of an application to resist or recover from disruptions, including those related to infrastructure, dependent services, misconfigurations, and transient network issues. (See Resiliency, and the components of reliability in the AWS Well-Architected Framework Reliability Pillar documentation.) However, to achieve the desired level of resilience, trade-offs are often required. Operational complexity, engineering complexity, and cost will need to be assessed and adjusted accordingly.

Based on years of working with customers and internal teams, AWS has developed a resilience lifecycle framework that captures resilience learnings and best practices. The framework outlines five key stages that are illustrated in the following diagram. At each stage you can use strategies, services, and mechanisms to improve your resilience posture.



These stages are discussed in the following sections of this guide:

- Stage 1: Set objectives
- Stage 2: Design and implement
- Stage 3: Evaluate and test
- Stage 4: Operate
- Stage 5: Respond and learn

Terms and definitions

The resilience concepts of each stage are applied at different levels, ranging from individual components to entire systems. Implementing these concepts requires a clear definition of several terms:

- A *component* is an element that performs a function, and consists of software and technology resources. Examples of components include code configuration, infrastructure such as

networking, or even servers, data stores, and external dependencies such as multi-factor authentication (MFA) devices.

- An *application* is a collection of components that delivers business value, such as a customer-facing web storefront or the backend process that improves machine learning models. An application might consist of a subset of components in a single AWS account, or it might be a collection of multiple components that span multiple AWS accounts and Regions.
- A *system* is a collection of applications, people, and processes that are required to manage a given business function. It encompasses the application required to run a function; operational processes such as continuous integration and continuous delivery (CI/CD), observability, configuration management, incident response, and disaster recovery; and the operators who manage such tasks.
- A *disruption* is an event that prevents your application from delivering its business function properly.
- *Impairment* is the effect that a disruption has on an application if it isn't mitigated. Applications can be impaired if they suffer a set of disruptions.

Continuous resilience

The resilience lifecycle is an ongoing process. Even within the same organization, your application teams might perform at different levels of completeness within each stage, depending on the requirements of your application. However, the more complete each stage is, the higher level of resilience your application will have.

You should think of the resilience lifecycle as a standard process that your organization can operationalize. AWS has intentionally modeled the resilience lifecycle to be similar to the software development lifecycle (SDLC), with the goal of incorporating planning, testing, and learning throughout the operating processes while you develop and operate your applications. As with many agile development processes, the resilience lifecycle can be repeated with every iteration of the development process. We recommend that you deepen the practices within each stage of the lifecycle progressively over time.

Stage 1: Set objectives

Understanding what level of resilience is needed and how you will measure it is the basis for the *set objectives* stage. It's difficult to improve something if you don't have an objective and you can't measure it.

Not all applications need the same level of resilience. When you set objectives, consider the required level in order to make the correct investments and trade-offs. A good analogy for this is a car: It has four tires but carries only one spare tire. The chance of getting multiple flat tires during a ride is low, and having extra spares could take away from other features, such as cargo space or fuel efficiency, so this is a reasonable trade-off.

After you define objectives, you implement observability controls in later stages (Stage 2: design and implement and Stage 4: Operate) to understand if the objectives are being met.

Mapping critical applications

Defining resilience objectives shouldn't exclusively be a technical conversation. Instead, start with a business-oriented focus to understand what the application should deliver and the consequences of impairment. This understanding of business objectives then cascades to areas such as architecture, engineering, and operations. Any resilience objectives you define *might* be applied to all your applications, but the way the objectives are measured often vary depending on the function of the application. You might be running an application that's critical to the business, and if this application is impaired, your organization could lose significant revenue or suffer reputational harm. Alternately, you might have another application that isn't as critical and can tolerate some downtime without negatively impacting your organization's ability to do business.

As an example, think of an order management application for a retail company. If the components of the order management application are impaired and don't run properly, new sales won't go through. This retail company also has a coffee shop for its employees that's located in one of its buildings. The coffee shop has an online menu that employees can access on a static webpage. If this webpage becomes unavailable, some employees might complain, but it won't necessarily cause financial harm to the company. Based on this example, the business would likely choose to have more aggressive resilience goals for the order management application but won't make a significant investment to ensure the resilience of the web application.

Identifying the most critical applications, where to apply the most effort, and where to make trade-offs is as important as being able to measure an application's resilience in production. To

better understand the impact of impairment, you can perform a business impact analysis (BIA). A BIA provides a structured and systematic approach to identify and prioritize critical business applications, assess potential risks and impacts, and identify supporting dependencies. The BIA helps quantify the *cost of downtime* for your organization's most important applications. This metric helps outline how much it will cost if a specific application is impaired and unable to complete its function. In the previous example, if the order management application is impaired, the retail business could lose significant revenue.

Mapping user stories

During the BIA process, you might discover that an application is responsible for more than one business function, or that a business function requires multiple applications. Using the previous retail company example, the order management function might require separate applications for checkout, promotion, and pricing. If one application fails, the impact could be felt by the business and by users who interact with the company. For example, the company might not be able to add new orders, provide access to promotions and discounts, or update the price of their products. These different functions required by the order management function might rely on multiple applications. These functions might also have multiple external dependencies, which makes the process of achieving purely component-focused resilience too complex. A better way to handle this scenario is to focus on user stories, which outline the experience that users expect when interacting with one application or a set of applications.

Focusing on user stories helps you understand which pieces of the customer experience are most important, so you can build mechanisms to protect against specific threats. In the previous example, one user story could be checkout, which involves the checkout application and has a dependency on the pricing application. Another user story could be viewing promotions, which involves the promotion application. After you map the most critical applications and their user stories, you can begin to define the metrics you will use to measure resilience for these user stories. These metrics can be applied across an entire portfolio or to individual user stories.

Defining measurements

Recovery point objectives (RPOs), recovery time objectives (RTOs), and service-level objectives (SLOs) are standard industry measurements that are used to assess the resilience of a given system. RPO refers to how much data loss the business can tolerate in case of a failure, whereas RTO is a measure of how quickly an application must be available again after an outage. These two metrics are measured in time units: seconds, minutes, and hours. You can also measure the amount of time

during which the application is working properly; that is, it performs its functions as designed and is accessible to its users. These SLOs detail the expected level of service customers will receive and are measured by metrics such as the percentage (%) of requests that are serviced without error within a response time that's less than one second (for example, 99.99% of requests will receive a response each month). RPO and RTO are related to disaster recovery strategies, assuming that there will be interruptions in application operation and recovery processes that range from restoring backups to redirecting user traffic. SLOs are addressed by implementing high availability controls, which tend to reduce the downtime for an application.

SLO metrics are commonly used in the definition of service-level agreements (SLAs), which are contracts between service providers and end users. SLAs usually come with financial commitments and outline penalties that need to be paid by the provider if these agreements aren't met. However, an SLA isn't a measurement of your resilience posture, and increasing an SLA doesn't make your application more resilient.

You can start to set your objectives based on SLOs, RPOs, and RTOs. After you define your resilience objectives and gain a clear understanding of your RPO and RTO targets, you can use AWS Resilience Hub to run an assessment of your architecture to uncover potential resilience-related weaknesses. AWS Resilience Hub assesses an application architecture against AWS Well-Architected Framework best practices and shares remediation guidance in the context of what specifically needs to be improved to meet your defined RTO and RPO targets.

Creating additional measurements

RPO, RTO and SLOs are good indicators of resilience, but you can also think about goals from a business perspective and define objectives around your application's functions. For example, your objective could be: *Successful orders per minute will remain above 98% if latency between my frontend and backend increases by 40%.* Or: *Streams started per second will remain within a standard deviation from average even if a specific component is lost.* You can also create objectives to achieve a reduction on the mean time to recover (MTTR) across known failure types; for example: *Recovery times will be reduced by x% if any of these known issues happen*. Creating objectives that align with a business need helps you anticipate the types of failures that your application should tolerate. It also helps you identify approaches to reduce the likelihood of impairment to your application.

If you think about the objective to continue operating if you lose 5% of the instances that power your application, you might determine that your application should be prescaled or have the ability to scale fast enough to support the additional traffic caused during that event. Or, you might

determine that you should leverage different architectural patterns, as described in the Stage 2: Design and implement section.

You also should implement observability measures for your specific business objectives. For example, you can track *average order rate*, *average order price*, *average number of subscriptions*, or other metrics that can provide insights into the health of the business based on your application's behavior. By implementing observability capabilities for your application, you can create alarms and take action if these metrics exceed your defined boundaries. Observability is covered in more detail in the Stage 4: Operate section.

Stage 2: Design and implement

In the previous stage, you set your resilience objectives. Now at the *design and implement* stage, you try to anticipate failure modes and identify design choices, as guided by the objectives you set in the previous stage. You also define strategies for change management and develop software code and infrastructure configuration. The following sections highlight AWS best practices that you should consider while taking trade-offs such as cost, complexity, and operational overhead into account.

AWS Well-Architected Framework

When you architect your application based on your desired resilience objectives, you need to evaluate multiple factors and make trade-offs on the most optimal architecture. To build a highly resilient application you must consider aspects of design, building and deployment, security, and operations. The AWS Well-Architected Framework provides a set of best practices, design principles, and architectural patterns to help you design resilient applications on AWS. The six pillars of the AWS Well-Architected Framework provide best practices for designing and operating resilient, secure, efficient, cost-effective, and sustainable systems. The framework provides a way to consistently measure your architectures against best practices and identify areas for improvement.

The following are examples of how the AWS Well-Architected Framework can help you design and implement applications that meet your resilience objectives:

- **The reliability pillar**: The reliability pillar emphasizes the importance of building applications that can operate correctly and consistently, even during failures or disruptions. For example, the AWS Well-Architected Framework recommends that you use a microservices architecture to make your applications smaller and simpler, so you can differentiate between the availability needs of different components within your application. You can also find detailed descriptions of best practices for building applications by using throttling, retry with exponential back off, fail fast (load shedding), idempotency, constant work, circuit breakers, and static stability.
- **Comprehensive review**: The AWS Well-Architected Framework encourages a comprehensive review of your architecture against best practices and design principles. It provides a way to consistently measure your architectures and identify areas for improvement.
- **Risk management**: The AWS Well-Architected Framework helps you identify and manage risks that might impact the reliability of your application. By addressing potential failure scenarios proactively, you can reduce their likelihood or the resulting impairment.

- **Continuous improvement**: Resilience is an ongoing process, and the AWS Well-Architected Framework emphasizes continuous improvement. By regularly reviewing and refining your architecture and processes based on the AWS Well-Architected Framework's guidance, you can ensure that your systems stay resilient in the face of evolving challenges and requirements.

Understanding dependencies

Understanding a system's dependencies is key for resilience. Dependencies include the connections between components within an application, and connections to components outside the application, such as third-party APIs and business-owned shared services. Understanding these connections helps you isolate and manage disruptions, because an impairment in one component can affect other components. This knowledge helps engineers assess the impact of impairments and plan accordingly, and ensure that resources are used effectively. Understanding dependencies helps you create alternate strategies and coordinating recovery processes. It also helps you determine cases where you can replace a hard dependency with a soft dependency, so your application can continue to serve its business function when there's a dependency impairment. Dependencies also influence decisions on load balancing and application scaling. Understanding dependencies is vital when you make changes to your application, because it can help you determine potential risks and impacts. This knowledge helps you create stable, resilient applications, assisting in fault management, impact assessment, impairment recovery, load balancing, scaling, and change management. You can track dependencies manually or use tools and services such as AWS X-Ray to understand the dependencies of your distributed applications.

Disaster recovery strategies

A disaster recovery (DR) strategy plays a pivotal role in building and operating resilient applications, primarily by ensuring business continuity. It guarantees that crucial business operations can persist with the least possible impairment, even during catastrophic events, therefore minimizing downtime and potential loss of revenue. DR strategies are essential for data protection because they often incorporate regular data backups and data replication across multiple locations, which helps safeguard valuable business information and helps prevent total loss during a disaster. Furthermore, many industries are regulated by policies that require businesses to have a DR strategy in place to protect sensitive data and to ensure that services remain available during a disaster. By assuring minimal service impairment, a DR strategy also bolsters customer trust and satisfaction. A well-implemented and frequently practiced DR strategy reduces the recovery time after a disaster, and helps ensure that applications are quickly brought

back online. Moreover, disasters can incur substantial costs, not just from lost revenue due to downtime, but also from the expense of restoring applications and data. A well-designed DR strategy helps shield against these financial losses.

The strategy you choose depends on the specific needs of your application, your RTO and RPO, and your budget. AWS Elastic Disaster Recovery is a purpose-built resilience service that you can use to help implement your DR strategy for both on-premises and cloud-based applications.

For more information, see Disaster Recovery of Workloads on AWS and AWS Multi-Region Fundamentals on the AWS website.

Defining CI/CD strategies

One of the common causes of application impairments is code or other changes that alter the application from a previously known working state. If you don't address change management carefully, it can cause frequent impairments. The frequency of changes increases the opportunity for impact. However, making changes less frequently results in larger change sets, which are much more likely to result in impairment due to their high complexity. Continuous integration and continuous delivery (CI/CD) practices are designed to keep changes small and frequent (resulting in increased productivity) while subjecting each change to a high level of inspection through automation. Some of the foundational strategies are:

- **Full automation**: The fundamental concept of CI/CD is to automate the build and deployment processes as much as possible. This includes building, testing, deployment, and even monitoring. Automated pipelines help reduce the possibility of human error, ensure consistency, and make the process more reliable and efficient.
- **Test-driven development (TDD)**: Write tests before writing the application code. This practice ensures that all code has associated tests, which improves the reliability of the code and the quality of the automated inspection. These tests are run in the CI pipeline to validate changes.
- **Frequent commits and integrations**: Encourage developers to commit code frequently and perform integrations often. Small, frequent changes are easier to test and debug, which reduces the risk of significant problems. Automation reduces the cost of each commit and deployment, making frequent integrations possible.
- **Immutable infrastructure**: Treat your servers and other infrastructure components like static, immutable entities. Replace infrastructure instead of modifying it as much as possible, and build new infrastructure through code that is tested, and deployed through your pipeline.

- **Rollback mechanism**: Always have an easy, reliable, and frequently tested way to roll back changes if something goes wrong. Being able to quickly return to the previous known good state quickly is essential to deployment safety. This can be a simple button to revert to the previous state, or it can be fully automated and initiated by alarms.
- **Version control**: Maintain all application code, configuration, and even infrastructure as code in a version-controlled repository. This practice helps ensure that you can easily track changes and revert them if needed.
- **Canary deployments and blue/green deployments**: Deploying new versions of your application to a subset of your infrastructure first, or maintaining two environments (blue/green), allows you to verify a change's behavior in production and quickly roll back if necessary.

CI/CD is not just about the tools but also about the culture. Creating a culture that values automation, testing, and learning from failures is just as important as implementing the right tools and processes. Rollbacks, if done very quickly with minimal impact, should not be considered a failure but a learning experience.

Conducting ORRs

An operational readiness review (ORR) helps identify operational and procedural gaps. At Amazon, we created ORRs to distill the learnings from decades of operating high-scale services into curated questions with best practice guidance. An ORR captures previous lessons learned and requires new teams to ensure that they have accounted for these lessons in their applications. ORRs can provide a list of failure modes or causes of failure that can be carried into the resilience modeling activity described in the resilience modeling section below. For more information, see Operational Readiness Reviews (ORRs) on the AWS Well-Architected Framework website.

Understanding AWS fault isolation boundaries

AWS provides multiple fault isolation boundaries to help you achieve your resilience objectives. You can use these boundaries to take advantage of the predictable scope of impact containment they provide. You should be familiar with how AWS services are designed by using these boundaries so that you can make intentional choices about the dependencies you select for your application. To understand how to use boundaries in your application, see AWS Fault Isolation Boundaries on the AWS website.

Selecting responses

A system can respond in a wide range of ways to an alarm. Some alarms might require a response from the operations team whereas others might trigger self-healing mechanisms within the application. You might decide to keep responses that could be automated as manual operations to control the costs of automation or to manage engineering constraints. The type of response to an alarm is likely to be selected as a function of the cost of implementing the response, the anticipated frequency of the alarm, the accuracy of the alarm, and the potential business loss of not responding to the alarm at all.

For example, when a server process crashes, the process might be restarted by the operating system, or a new server might be provisioned and the old one terminated, or an operator might be instructed to remotely connect to the server and restart it. These responses have the same result—restarting the application server process—but have varying levels of implementation and maintenance costs.

> **Note**
>
> You might select multiple responses in order to take an in-depth resilience approach. For example, in the previous scenario the application team might choose to implement all three responses with a time delay between each. If the Failed server process indicator is still in an alarmed state after 30 seconds, the team can assume that the operating system has failed to restart the application server. Therefore, they might create an auto scaling group to create a new virtual server and restore the application server process. If the indicator is still in an alarm state after 300 seconds, an alert might be sent to the operational staff to connect to the original server and attempt to restore the process.

The response that the application team and business select should reflect the appetite of the business to offset operational overhead with upfront investment in engineering time. You should choose a response—an architecture pattern such as static stability, a software pattern such as a circuit breaker, or an operational procedure—by carefully considering the constraints and the anticipated maintenance of each response option. Some standard responses might exist to guide application teams, so you can use the libraries and patterns that are managed by your centralized architecture function as an input to this consideration.

Resilience modeling

Resilience modeling documents how an application will respond to different anticipated disruptions. By anticipating disruptions, your team can implement observability, automated controls, and recovery processes to mitigate or prevent impairment despite disruptions. AWS has created guidance for developing a resilience model by using the resilience analysis framework. This framework can help you anticipate disruptions and their impact to your application. By anticipating disruptions, you can identify the mitigations needed to build a resilient, reliable application. We recommend that you use the resilience analysis framework to update your resilience model with every iteration of your application's lifecycle. Using this framework with each iteration helps reduce incidents by anticipating disruptions during the design phase and testing the application before and after production deployment. Developing a resilience model by using this framework helps you ensure that you meet your resilience objectives.

Failing safely

If you're unable to avoid disruptions, fail safely. Consider creating your application with a default fail-safe mode of operation, where no significant business loss can be incurred. An example of a fail-safe state for a database would be to default to read-only operations, where users aren't allowed to create or mutate any data. Depending on the sensitivity of the data, you might even want the application to default to a shutdown state and not even perform read-only queries. Consider what the fail-safe state for your application should be, and default to this mode of operation under extreme conditions.

Stage 3: Evaluate and test

During the *evaluate and test* stage of the lifecycle, the application, or changes to an existing application, have been designed but haven't yet been released to production. In this stage, you implement activities to test the practices that have been performed in previous stages and evaluate the results. The application might still be in active development, or primary development might be complete and the application might be undergoing testing before it's released to production. During this stage, you focus on developing and running tests that confirm or refute expectations that the application will meet the defined objectives for resilience. Additionally, you develop and test the system's operational procedures. The deployment procedures you developed in the Stage 2: Design and implement stage are put into practice and the results are evaluated. Although these testing and evaluation activities begin during this portion of the lifecycle, they do not end here. Testing and evaluation continue as you move into the Stage 4: Operate stage.

The *evaluate and test* stage is divided into two phases: pre-deployment activities and post-deployment activities. Pre-deployment activities consist of tasks that should be completed before you deploy the application into any environment, including deploying new versions of the software as well as the initial deployment into a testing environment. Post-deployment activities take place after the software has been deployed into a testing or production environment. The following sections discuss these phases in more detail.

Pre-deployment activities

Environment design

The environment in which you test and evaluate your application affects how thoroughly you can test it, and how much confidence you have that those results accurately reflect what will happen in production. You might be able to perform some integration testing locally on developer machines by using services such as Amazon DynamoDB (see Setting up DynamoDB local in the DynamoDB documentation). However, at some point you need to test in an environment that replicates your production environment in order to achieve the highest confidence in your results. This environment will incur cost, so we recommend that you take a staged, or pipelined, approach to your environments, where production-like environments appear later in the pipeline.

Integration testing

Integration testing is the process of testing that a well-defined component of an application performs its functions correctly when it operates with external dependencies. Those external dependencies could be other custom-developed components, AWS services that you use for your application, third-party dependencies, and on-premises dependencies. This guide focuses on integration tests that demonstrate the resilience of your application. It assumes that unit and integration tests already exist that demonstrate the functional accuracy of your software.

We recommend that you design integration tests that specifically test the resilience patterns you have implemented, such as circuit breaker patterns or load shedding (see Stage 2: Design and implement). Resilience-oriented integration tests often involve applying a specific load to the application or intentionally introducing disruptions into the environment by using capabilities such as AWS Fault Injection Service (AWS FIS). Ideally, you should run all integration tests as part of your CI/CD pipeline and ensure that you run tests every time code is committed. This helps you quickly detect and react to any changes to code or configurations that result in violations of your resilience objectives. Large-scale distributed applications are complex, and even minor changes can significantly impact the resilience of seemingly unrelated portions of your application. Try to run your tests on every commit. AWS provides an excellent set of tools for operating your CI/CD pipeline and other DevOps tools. For more information, see Introduction to DevOps on AWS on the AWS website.

Automated deployment pipelines

Deployment to, and testing in, your pre-production environments is a repetitive and complex task that is best left to automation. Automation of this process frees up human resources and reduces the opportunity for error. The mechanism for automating this process is often referred to as a *pipeline*. When you create your pipeline, we recommend that you set up a series of testing environments that get increasingly closer to your production configuration. You use this series of environments to repeatedly test your application. The first environment provides a more limited set of capabilities than the production environment but incurs a significantly lower cost. Subsequent environments should add services and scale to more closely mirror the production environment.

Start by testing in the first environment. After your deployments pass all your tests in the first test environment, let the application run under some amount of load for a period of time to see whether any issues occur over time. Confirm that you have configured observability correctly (see *Alarm precision* later in this guide) so that you can detect any issues that arise.

When this observation period has completed successfully, deploy your application to your next testing environment and repeat the process, adding additional tests or load as supported by the environment. After you have sufficiently tested your application in this way, you can use the deployment methods that you previously set up to deploy the application into production (see *Define CI/CD strategies* earlier in this guide). The article Automating safe, hands-off deployments in the Amazon Builders' Library is an excellent resource that describes how Amazon automates code deployment. The number of environments that precede your production deployment will vary, depending on the complexity of your application and the types of dependencies it has.

Load testing

On the surface, load testing resembles integration testing. You test a discrete function of your application and its external dependencies to verify that it operates as expected. Load testing then goes beyond integration testing to focus on how the application functions under well-defined loads. Load testing requires verification of correct functionality, so it must occur after a successful integration test. It is important to understand how well the application responds under expected loads as well as how it behaves when the load exceeds expectations. This helps you verify that you have implemented the necessary mechanisms to ensure that your application remains resilient under extreme load. For a comprehensive guide to load testing on AWS, see Distributed Load Testing on AWS in the AWS Solutions Library.

Post-deployment activities

Resilience is an ongoing process and the evaluation of your application's resilience must continue after the application has been deployed. The results of your post-deployment activities, such as ongoing resilience assessments, might require that you re-evaluate and update some of the resilience activities you performed earlier in the resilience lifecycle.

Conducting resilience assessments

Assessing resilience doesn't stop after you deploy your application into production. Even if you have well-defined and automated deployment pipelines, changes can sometimes occur directly in a production environment. Additionally, there might be factors that you have not yet taken into consideration in your pre-deployment resilience verification. AWS Resilience Hub provides a central place where you can assess whether your deployed architecture meets your defined RPO and RTO needs. You can use this service to run on-demand assessments of your application's resilience, automate assessments, and even integrate them into your CI/CD tools, as discussed in

the AWS blog post Continually assessing application resilience with AWS Resilience Hub and AWS CodePipeline. Automating these assessments is a best practice because it helps ensure that you are continuously evaluating your resilience posture in production.

DR testing

In Stage 2: Design and implement, you developed disaster recovery (DR) strategies as part of your system. During Stage 4, you should test your DR procedures to ensure that your team is fully prepared for an incident and your procedures work as expected. You should test all your DR procedures, including failover and failback, on a regular basis and review the results of each exercise to determine if and how your system's procedures should be updated for the best possible outcome. When you initially develop your DR test, schedule the test well in advance and ensure that the entire team understands what to expect, how the outcomes will be measured, and what feedback mechanism will be used to update procedures based on the outcome. After you become proficient in running scheduled DR tests, consider running unannounced DR tests. Real disasters don't occur on a schedule, so you need to be prepared to exercise your plan at any time. However, unannounced doesn't mean unplanned. Key stakeholders still need to plan the event to ensure that proper monitoring is in place and that customers and critical applications are not adversely impacted.

Drift detection

Unanticipated changes to configuration in production applications can occur even when automation and well-defined procedures are in place. To detect changes to your application's configuration, you should have mechanisms for detecting *drift*, which refers to deviations from a baselined configuration. To learn how to detect drift in your AWS CloudFormation stacks, see Detecting unmanaged configuration changes to stacks and resources in the AWS CloudFormation documentation. To detect drift in your application's AWS environment, see Detect and resolve drift in AWS Control Tower in the AWS Control Tower documentation.

Synthetic testing

Synthetic testing is the process of creating configurable software that runs in production, on a scheduled basis, to test your application's APIs in a way that simulates the end-user experience. These tests are sometimes referred to as *canaries*, in reference to the term's original use in coal mining. Synthetic tests can often provide early warnings when an application suffers from a disruption, even if the impairment is partial or intermittent, as is often the case with gray failures.

Chaos engineering

Chaos engineering is a systematic process that involves deliberately subjecting an application to disruptive events in a risk-mitigated way, closely monitoring its response, and implementing necessary improvements. Its purpose is to validate or challenge assumptions about the application's ability to handle such disruptions. Instead of leaving these events to chance, chaos engineering empowers engineers to orchestrate experiments in a controlled environment, typically during periods of low traffic and with readily available engineering support for effective mitigation.

Chaos engineering begins with understanding the normal operating conditions, known as the *steady state*, of the application under consideration. From there, you formulate a hypothesis that details the successful behavior of the application in the presence of disruption. You run the experiment, which involves deliberate injection of disruptions, including, but not limited to, network latency, server failures, hard drive errors, and impairment of external dependencies. You then analyze the results of the experiment and enhance the application's resilience based on your learnings. The experiment serves as a valuable tool for improving various facets of the application, including its performance, and uncovers latent issues that might have remained hidden otherwise. Additionally, chaos engineering helps reveal deficiencies in observability and alarming tools, and helps you refine them. It also contributes to reducing recovery time and enhancing operational skills. Chaos engineering accelerates the adoption of best practices and cultivates a mindset of continuous improvement. Ultimately, it enables teams to build and hone their operational skills through regular practice and repetition.

AWS recommends that you start your chaos engineering efforts in a non-production environment. You can use AWS Fault Injection Service (AWS FIS) to run chaos engineering experiments with general-purpose faults as well as faults that are unique to AWS. This fully managed service includes stop-condition alarms and full permission controls so you can easily adopt chaos engineering with safety and confidence.

Stage 4: Operate

After you've completed the Stage 3: Evaluate and test, you're ready to deploy the application to production. In the *Operate* stage, you deploy your application to production and manage your customers' experience. The design and implementation of your application determine many of its resilience outcomes, but this stage focuses on the operational practices your system uses to maintain and improve resilience. Building a culture of operational excellence helps create standards and consistency in these practices.

Observability

The most important part of understanding the customer experience is through monitoring and alarming. You need to instrument your application to understand its state, and you need diverse perspectives, which means that you need to measure from both the server side and the client side, typically with canaries. Your metrics should include data about your application's interactions with its dependencies and dimensions that align to your fault isolation boundaries. You should also produce logs that provide additional details about every unit of work performed by your application. You might consider combining metrics and logs by using a solution such as the Amazon CloudWatch embedded metric format. You'll likely find that you always want more observability, so consider the cost, effort, and complexity trade-offs required to implement your desired level of instrumentation.

The following links provide best practices for instrumenting your application and creating alarms:

- Monitoring production services at Amazon (AWS re:Invent 2020 presentation)
- Amazon Builders' Library: Operational Excellence at Amazon (AWS re:Invent 2021 presentation)
- Observability best practices at Amazon (AWS re:Invent 2022 presentation)
- Instrumenting distributed systems for operational visibility (Amazon Builders' Library article)
- Building dashboards for operational visibility (Amazon Builders' Library article)

Event management

You should have an event management process in place to handle impairments when your alarms (or worse, your customers) tell you that something is going wrong. This process should include engaging an on-call operator, escalating problems, and establishing runbooks for consistent approaches to troubleshooting that help remove human errors. However, impairments typically

don't happen in isolation; a single application could impact multiple other applications that depend on it. You can rapidly address issues by understanding all applications that are impacted and bringing operators from multiple teams together on a single conference call. However, depending on your organization's size and structure, this process might require a centralized operations team.

In addition to setting up an event management process, you should regularly review your metrics through dashboards. Regular reviews help you understand the customer experience and longer-term trends in the performance of your application. This helps you identify issues and bottlenecks before they cause significant production impact. Reviewing metrics in a consistent, standardized way provides significant benefits but requires top-down buy-in and an investment of time.

The following links provide best practices on building dashboards and operational metrics reviews:

- Building dashboards for operational visibility (Amazon Builders' Library article)
- Amazon's approach to failing successfully (AWS re:Invent 2019 presentation)

Continuous resilience

During Stage 2: Design and implement and Stage 3: Evaluate and test, you initiated review and test activities before deploying your application to production. During the *operate* stage, you should continue iterating on those activities in production. You should periodically review the resilience posture of your application through AWS Well-Architected Framework reviews, Operational Readiness Reviews (ORRs), and the resilience analysis framework. This helps ensure that your application hasn't drifted from established baselines and standards and keeps you up to date with new or updated guidance. These continuous resilience activities help you discover previously unanticipated disruptions and help you come up with new mitigations.

You might also want to consider running game days and chaos engineering experiments in production after you've successfully run them in pre-production environments. Game days simulate known events that you have built resilience mechanisms to mitigate. For example, a game day might simulate an AWS Regional service impairment and implement a multi-Region failover. Although implementing these activities can require a significant level of effort, both practices help you build confidence that your system is resilient to the failure modes that you've designed it to withstand.

By operating your applications, encountering operational events, reviewing metrics, and testing your application, you'll encounter numerous opportunities to respond and learn.

Stage 5: Respond and learn

How your application responds to disruptive events influences its reliability. Learning from experience and how your application has responded to disruption in the past is also critical to improving its reliability.

The *Respond and learn* stage focuses on practices that you can implement to better respond to disruptive events in your applications. It also includes practices to help you distill the maximum amount of learning from the experiences of your operations teams and engineers.

Creating incident analysis reports

When an incident occurs, the first action is to prevent further harm to customers and the business as quickly as possible. After the application has recovered, the next step is to understand what happened and to identify steps to prevent reoccurrence. This post-incident analysis is usually captured as a report that documents the set of events that led to an impairment of the application, and the effects of the disruption on the application, customers, and the business. Such reports become valuable learning artifacts and should be shared widely across the business.

> **Note**
>
> It's critical to perform incident analysis without assigning any blame. Assume that all operators took the best and most appropriate course of action given the information they had. Do not use the names of operators or engineers in a report. Citing human error as a reason for impairment might cause team members to be guarded in order to protect themselves, resulting in the capture of incorrect or incomplete information.

A good incident analysis report, like that documented in the Amazon Correction of Error (COE) process, follows a standardized format and tries to capture, in as much detail as possible, the conditions that led to an impairment of the application. The report details a time-stamped series of events and captures quantitative data (often metrics and screenshots from monitoring dashboards) that describe the measurable state of the application over the timeline. The report should capture the thought processes of operators and engineers who took action, and the information that led them to their conclusions. The report should also detail the performance of different indicators—for example, which alarms were raised, whether those alarms accurately reflected the state of the application, the time lag between events and the resulting alarms, and

the time to resolve the incident. The timeline also captures the runbooks or automations that were initiated and how they helped the application regain a useful state. These elements of the timeline help your team understand the effectiveness of automated and operator responses, including how quickly they addressed the problem and how effective they were in mitigating the disruption.

This detailed picture of a historical event is a powerful educational tool. Teams should store these reports in a central repository that is available to the entire business so that others can review the events and learn from them. This can improve your teams' intuition about what can go wrong in production.

A repository of detailed incident reports also becomes a source of training material for operators. Teams can use an incident report to inspire a table-top or live game day, where teams are given information that plays back the timeline that's captured in the report. Operators can walk through the scenario with partial information from the timeline and describe what actions they would take. The moderator for the game day can then provide guidance on how the application responded based on the operator's actions. This develops the troubleshooting skills of operators, so they can more easily anticipate and troubleshoot issues.

A centralized team that's responsible for application reliability should maintain these reports in a centralized library that the entire organization can access. This team should also be responsible for maintaining the report template and training teams on how to complete the incident analysis report. The reliability team should periodically review the reports to detect trends across the business that can be addressed through software libraries, architecture patterns, or changes to team processes.

Conducting operational reviews

As discussed in Stage 4: Operate, operational reviews are an opportunity to review recent feature releases, incidents, and operational metrics. The operational review is also an opportunity to share learnings from feature releases and incidents with the wider engineering community in your organization. During the operational review, the teams review feature deployments that were rolled back, incidents that occurred, and how they were handled. This gives engineers across the organization an opportunity to learn from the experiences of others and to ask questions.

Open your operational reviews to the engineering community in your company so they can learn more about the IT applications that run the business and the types of issues they can encounter. They will carry this knowledge with them as they design, implement, and deploy other applications for the business.

Reviewing alarm performance

Alarms, as discussed in the *operate* stage, might result in dashboard alerts, tickets being created, emails being sent, or operators being paged. An application will have numerous alarms configured to monitor various aspects of its operation. Over time, the accuracy and effectiveness of these alarms should be reviewed to increase alarm precision, reduce false positives, and consolidate duplicate alerts.

Alarm precision

Alarms should be as specific as possible to reduce the amount of time that you have to spend interpreting or diagnosing the specific disruption that caused the alarm. When an alarm is raised in response to an application impairment, the operators who receive and respond to the alarm must first interpret the information that the alarm conveys. The information might be a simple error code that maps to a course of action such as a recovery procedure, or it might include lines from application logs that you have to review to understand why the alarm was raised. As your team learns to operate an application more effectively, they should refine these alarms to make them as clear and concise as possible.

You can't anticipate all possible disruptions to an application, so there will always be general alarms that require an operator to analyze and diagnose. Your team should work to reduce the number of general alarms in order to improve response times and decrease the mean time to repair (MTTR). Ideally, there should be a one-to-one relationship between an alarm and an automated or human-performed response.

False positives

Alarms that require no action from operators but produce alerts as emails, pages, or tickets will be ignored by operators over time. Periodically, or as part of an incident analysis, review alarms to identify those that are often ignored or require no action from operators (*false positives*). You should work to either remove the alarm, or improve the alarm so that it issues an actionable alert to operators.

False negatives

During an incident, alarms that are configured to alert during the incident might fail, perhaps because of an event that impacts the application in an unexpected way. As part of an incident analysis, you should review the alarms that should have been raised but weren't. You should work to improve these alarms so they better reflect the conditions that might arise from an event.

Alternatively, you might have to create additional alarms that map to the same disruption but are raised by a different symptom of the disruption.

Duplicative alerts

A disruption that impairs your application is likely to cause multiple symptoms and might result in multiple alarms. Periodically, or as part of an incident analysis, you should review the alarms and alerts that were issued. If operators received duplicate alerts, create aggregate alarms to consolidate them into a single alert message.

Conducting metrics reviews

Your team should collect operational metrics about your application, such as the number of incidents by severity per month, the time to detect the incident, the time to identify the cause, the time to remediate, and the number of tickets created, alerts sent, and pages raised. Review these metrics at least monthly to understand the burden on operational staff, the signal-to-noise ratio they deal with (for example, informational versus actionable alerts), and whether the team is improving its ability to operate the applications under their control. Use this review to understand trends in the measurable aspects of the operations team. Solicit ideas from the team on how to improve these metrics.

Providing training and enablement

It's difficult to capture a detailed description of an application and its environment that led to an incident or unexpected behavior. Furthermore, modeling the resilience of your application to anticipate such scenarios isn't always straightforward. Your organization should invest in training and enablement materials for your operations teams and developers to participate in activities such as resilience modeling, incident analysis, game days, and chaos engineering experiments. This will improve the fidelity of the reports that your teams produce and the knowledge that they capture. The teams will also become better equipped to anticipate failures without relying on a smaller, more experienced group of engineers who have to lend their insight through scheduled reviews.

Creating an incident knowledge base

An incident report is a standard output from an incident analysis. You should use the same or a similar report to document scenarios where you detected anomalous application behavior,

even if the application didn't become impaired. Use the same standardized report structure to capture the outcome of chaos experiments and game days. The report represents a snapshot of the application and its environment that led to an incident or otherwise unexpected behavior. You should store these standardized reports in a central repository that all engineers within the business can access.

Operations teams and developers can then search this knowledge base to understand what has disrupted applications in the past, what types of scenarios could have caused disruption, and what prevented application impairment. This knowledge base becomes an accelerator for improving the skills of your operations teams and your developers, and enables them to share their knowledge and experiences. Additionally, you can use the reports as training material or scenarios for game days or chaos experiments to improve the operational team's intuition and ability to troubleshoot disruptions.

> **Note**
>
> A standardized report format also provides readers with a sense of familiarity and helps them find the information they are looking for more quickly.

Implementing resilience in depth

As discussed earlier, an advanced organization will implement multiple responses to an alarm. There is no guarantee that a response will be effective, so by layering responses an application will be better equipped to fail gracefully. We recommend that you implement at least two responses for each indicator to ensure that an individual response doesn't become a single point of failure that might lead to a DR scenario. These layers should be created in serial order, so that a successive response is performed only if the previous response was ineffective. You shouldn't run multiple layered responses to a single alarm. Instead, use an alarm that indicates whether a response has been unsuccessful, and, if so, initiates the next layered response.

Conclusion and resources

This guide presents a lifecycle that helps you continuously improve the resilience of your applications by implementing best practices across five stages: *Set objectives*, *Design and implement*, *Evaluate and test*, *Operate*, and *Respond and learn*.

For more information about the services and concepts discussed in this guide, see the following resources.

AWS services:

- AWS Backup
- AWS Elastic Disaster Recovery
- AWS Fault Injection Service (AWS FIS)
- AWS Resilience Hub
- Amazon Route 53 Application Recovery Controller
- AWS X-Ray

Blog posts and articles:

- Availability and Beyond: Understanding and Improving the Resilience of Distributed Systems on AWS
- AWS Fault Isolation Boundaries
- AWS Multi-Region Fundamentals
- Chaos Engineering in the cloud
- Continually assessing application resilience with AWS Resilience Hub and AWS CodePipeline
- Disaster Recovery of On-Premises Applications to AWS
- Reliability Pillar – AWS Well-Architected Framework
- Resilience analysis framework

Contributors

Contributors to this guide include:

- Bruno Emer, Principal Solutions Architect, Amazon Web Services
- Clark Richey, Principal Solutions Architect, Amazon Web Services
- Elaine Harvey, General Manager, Reliability Services, Amazon Web Services
- Jason Barto, Principal Solutions Architect, Amazon Web Services
- John Formento, Principal Solutions Architect, Amazon Web Services
- Lisi Lewis, Sr. Product Marketing Manager, Amazon Web Services
- Michael Haken, Principal Solutions Architect, Amazon Web Services
- Neeraj Kumar, Principal Solutions Architect, Amazon Web Services
- Wangechi Doble, Principal Solutions Architect, Amazon Web Services

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	—	October 6, 2023

AWS Prescriptive Guidance glossary

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

Numbers

7 Rs

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

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. This migration scenario is specific to VMware Cloud on AWS, which supports virtual machine (VM) compatibility and workload portability between your on-premises environment and AWS. You can use the VMware Cloud Foundation technologies from your on-premises data centers when you migrate your infrastructure to VMware Cloud on AWS. Example: Relocate the hypervisor hosting your Oracle database to VMware Cloud on AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later

time, and legacy applications that you want to retain, because there's no business justification for migrating them.

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

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

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

active-passive migration

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

aggregate function

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

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

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

anti-pattern

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

application control

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

application portfolio

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

artificial intelligence (AI)

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

artificial intelligence operations (AIOps)

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

asymmetric encryption

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

atomicity, consistency, isolation, durability (ACID)

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

attribute-based access control (ABAC)

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

authoritative data source

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

Availability Zone

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

AWS Cloud Adoption Framework (AWS CAF)

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

AWS Workload Qualification Framework (AWS WQF)

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

B

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.

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.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

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

chaos engineering

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

CI/CD

See continuous integration and continuous delivery.

classification

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

client-side encryption

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

Cloud Center of Excellence (CCoE)

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

cloud computing

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

cloud operating model

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

cloud stages of adoption

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

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

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

CMDB

See configuration management database.

code repository

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

cold cache

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

cold data

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

computer vision

A field of AI used by machines to identify people, places, and things in images with accuracy at or above human levels. Often built with deep learning models, it automates extraction, analysis, classification, and understanding of useful information from a single image or a sequence of images.

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.

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 minimization

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

data perimeter

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

data preprocessing

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

data provenance

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

data subject

An individual whose data is being collected and processed.

data warehouse

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

database definition language (DDL)

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

database manipulation language (DML)

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

DDL

See database definition language.

deep ensemble

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

deep learning

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

defense-in-depth

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

delegated administrator

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

deployment

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

development environment

See environment.

detective control

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

development value stream mapping (DVSM)

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

digital twin

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

dimension table

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

disaster

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

disaster recovery (DR)

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

DML

See database manipulation language.

domain-driven design

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

DR

See disaster recovery.

drift detection

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

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

edge computing

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

encryption

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

encryption key

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

endianness

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

endpoint

See service endpoint.

endpoint service

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

can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

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.

exploratory data analysis (EDA)

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

F

fact table

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

fail fast

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

fault isolation boundary

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

feature branch

See branch.

features

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

feature importance

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

feature transformation

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

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

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

flash-cut migration

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

G

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

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

Gitflow workflow

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

greenfield strategy

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

guardrail

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

for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

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

high availability (HA)

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

historian modernization

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

homogeneous database migration

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

hot data

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

hotfix

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

hypercare period

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

I

IaC

See infrastructure as code.

identity-based policy

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

idle application

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

IIoT

See industrial Internet of Things.

immutable infrastructure

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

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends

setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

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

infrastructure

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

infrastructure as code (IaC)

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

industrial Internet of Things (IIoT)

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

inspection VPC

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

Internet of Things (IoT)

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

interpretability

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

IoT

See Internet of Things.

IT information library (ITIL)

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

IT service management (ITSM)

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

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

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

landing zone

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

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

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

lift and shift

See 7 Rs.

little-endian system

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

lower environments

See environment.

M

machine learning (ML)

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

main branch

See branch.

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*.

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.

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.

MPA

See Migration Portfolio Assessment.

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.

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.

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.

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.

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.

policy

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

polyglot persistence

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

portfolio assessment

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

predicate

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

predicate pushdown

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

preventative control

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

principal

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

Privacy by Design

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

private hosted zones

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

proactive control

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

production environment

See environment.

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.

Q

query plan

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

query plan regression

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

R

RACI matrix

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

ransomware

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

RASCI matrix

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

RCAC

See row and column access control.

read replica

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

re-architect

See 7 Rs.

recovery point objective (RPO)

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

recovery time objective (RTO)

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

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Managing AWS Regions in *AWS General Reference*.

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.

resource-based policy

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

responsible, accountable, consulted, informed (RACI) matrix

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

responsive control

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

retain

See 7 Rs.

retire

See 7 Rs.

rotation

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

row and column access control (RCAC)

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

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

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

S

SAML 2.0

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

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 Secret in the Secrets Manager documentation.

security control

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

security hardening

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

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers,

networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

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

server-side encryption

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

service control policy (SCP)

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

service endpoint

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

service-level agreement (SLA)

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

service-level indicator (SLI)

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

service-level objective (SLO)

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

shared responsibility model

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

SIEM

See security information and event management system.

single point of failure (SPOF)

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

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

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

strangler fig pattern

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

Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

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

symmetric encryption

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

synthetic testing

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

T

tags

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

target variable

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

task list

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

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that

captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

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

trunk-based workflow

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

trusted access

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

tuning

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

two-pizza team

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

U

uncertainty

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

undifferentiated tasks

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

upper environments

See environment.

V

vacuuming

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

version control

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

VPC peering

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

vulnerability

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

W

warm cache

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

warm data

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

window function

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

workload

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

workstream

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

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

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

Z

zero-day exploit

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

zero-day vulnerability

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

zombie application

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