AWS Whitepaper

AWS Multi-Region Fundamentals

AWS Multi-Region Fundamentals: AWS Whitepaper

Table of Contents

Abstract and introduction ... **i**
Abstract ... 1
Are you Well-Architected? ... 1
Introduction ... 1
Engineering and operating for resilience in a single Region ... **3**
Multi-Region fundamental 1: Understanding the requirements ... **4**
Key guidance ... 6
Multi-Region fundamental 2: Understanding the data ... **7**
2a: Understanding data consistency requirements ... 7
2b: Understanding the data access patterns ... 8
Key guidance ... 9
Multi-Region fundamental 3: Understanding your workload dependencies ... **10**
3a: AWS services ... 10
3b: Internal and third-party dependencies ... 10
3c: Failover mechanism ... 11
3d: Configuration dependencies ... 12
Key guidance ... 12
Multi-Region fundamental 4: Operational readiness ... **13**
4a: AWS account management ... 13
4b: Deployment practices ... 13
4c: Observability ... 14
4d: Processes, procedures, and testing ... 14
4e: Cost and complexity ... 15
Key guidance ... 15
Conclusion ... **17**
Contributors ... **18**
Further reading ... **19**
Document revisions ... **20**
Notices ... **21**
AWS Glossary ... **22**

AWS Multi-Region Fundamentals

Publication date: **December 20, 2022** (*Document revisions*)

Abstract

This advanced, 300-level paper is intended for cloud architects and senior leaders building workloads on AWS who are interested in using a multi-Region architecture to improve resilience for their workloads. This paper assumes baseline knowledge of AWS infrastructure and services. It outlines common multi-Region use cases, shares fundamental multi-Region concepts and implications around design, development, and deployment, and provides prescriptive guidance to help you better determine whether a multi-Region architecture is right for your workloads.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

Each AWS Region consists of multiple independent and physically separate Availability Zones within a geographic area. Strict logical separation between the software services in each Region is maintained. This purposeful design ensures that an infrastructure or services failure in one Region will not result in a correlated failure in another Region.

Most AWS customers can achieve their resilience objectives for a workload in a single Region using multiple Availability Zones (AZs) or Regional AWS services. However, a subset of customers pursue multi-Region architectures for three reasons.

- They have high availability and continuity of operations requirements for their highest tier workloads that they believe cannot be met in a single Region.
- They need to satisfy data sovereignty requirements (such as adherence to local laws, regulations, and compliance) which require workloads to operate within a certain jurisdiction.
- They need to improve performance and customer experience for the workload by running the workloads in locations closest to end users.

This paper focuses on high availability and continuity of operations requirements, and helps you navigate the considerations for adopting a multi-Region architecture for a workload. We describe fundamental concepts that apply to design, development, and deployment of a multi-Region workload, along with a prescriptive framework to help you determine whether a multi-Region architecture is the right choice for a particular workload. You need to ensure a multi-Region architecture is the right choice for your workload, because these architectures are challenging, and it’s possible that, if not done correctly, the overall availability of the workload can decrease.

Engineering and operating for resilience in a single Region

Before diving into multi-Region concepts, start by confirming that your workload is already as resilient as possible in a single Region. To achieve this, evaluate your workload against the Reliability Pillar and Operational Excellence Pillar of the AWS Well-Architected Framework, and make any necessary changes to adopt the recommended best practices. The following concepts are covered in the AWS Well-Architected Framework:

- Workload segmentation based on domain boundaries
- Well-defined service contracts
- Dependency management and coupling
- Handling failures, retries and back-off strategies
- Idempotent operations and stateful vs stateless transactions
- Operational readiness and change management
- Understanding workload health
- Responding to events

To take single Region resilience further, review and apply concepts discussed in *Advanced Multi-AZ resilience patterns for handling gray failures*. This paper provides best practices around using replicas in each Availability Zone to contain failures, and expands on Multi-AZ concepts introduced in AWS Well Architected. Once you have fully applied the recommended concepts and best practices for achieving the highest resilience in a single Region, a specific workload can be evaluated against fundamentals for multi-Region architectures to determine if the workload's resilience can be increased using a multi-Region approach.

Multi-Region fundamental 1: Understanding the requirements

As mentioned previously, *high availability* and *continuity of operations* are common reasons for pursuing multi-Region architectures. Availability metrics measure the percentage of time a workload is available for use over a defined period, while continuity of operations metrics measure recovery for large scale and typically longer duration events.

Measuring availability is a nearly continuous process. Specific measurements or metrics can vary, but typically coalesce around a target availability, most often referred to as *nines* (such as 99.99% availability). With availability goals, one size does not fit all. Availability goals need to be established at a workload level versus applying a single goal across all workloads, separating out the non-critical components from the critical.

For continuity of operations, the following point-in-time measurements are typically used:

- **Recovery Time Objective (RTO)** — RTO is the maximum acceptable delay between the interruption of service and restoration of service. This value determines an acceptable duration for which the service is impaired.
- **Recovery Point Objective (RPO)** — RPO is the maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable data loss between the latest recovery point and a service interruption.

Similar to setting availability goals, RTO and RPO should also be defined at a workload level. To achieve more aggressive continuity of operations or high availability requirements, increased investment is required. That said, not every application can demand or requires the same level of resilience. Creating a tiering mechanism can help establish the framework to align business and IT owners in identifying the most demanding applications based on business impact, and tier them accordingly. Examples of tiering can be found in the following tables.

Table 1 – Example resilience tiering for SLA

Availability Service-level Agreement (SLA)	Resilience tier	Acceptable downtime /year
99.99%	Platinum	52.60 minutes

Availability Service-level Agreement (SLA)	Resilience tier	Acceptable downtime /year
99.90%	Gold	8.77 hours
99.5%	Silver	1.83 days

Table 2 – Example resilience tiering for RTO and RPO

Tier	Max RTO	Max RPO	Criteria	Cost
Platinum	15 minutes	five minutes	Mission-critical workloads	$$$
Gold	15 minutes – six hours	two hours	Important, but not mission-critical workloads	$$
Silver	six hours – a few days	24 hours	Noncritical workloads	$

When designing workloads for resilience, an understanding of the relationship between high availability and continuity of operations is necessary. For example, if a workload requires 99.99% availability, no more than 53 minutes of downtime per year is tolerable. It can take at least five minutes to detect a failure and another ten minutes for an operator to engage, make decisions on recovery steps, and perform these steps. It's not unusual for a single issue to take 30 to 45 minutes to recover from. In this case, having a multi-Region strategy to provide an isolated instance that removes correlated impact can allow for continued operations by failing over within a bounded time, while triaging the initial impairment independently. This is where defining the appropriate RTO and RPO is necessary.

For mission-critical workloads that have extreme availability needs (for example, 99.99% availability or higher) or stringent continuity of operations requirements that can only be met by failing over into another Region, a multi-Region approach might be appropriate. However, these requirements are typically only applicable for a small subset of an enterprise's workload portfolio that has a bounded recovery time measured in minutes or hours. Unless an application needs a recovery time of minutes or a few hours, waiting for a Regional disruption to the application to be remediated within the affected Region might be a better approach, and is typically aligned with lower-tier workloads.

Before implementing a multi-Region architecture, business decision makers and technical teams should be aligned on cost implications, including operational and infrastructure cost drivers. A typical multi-Region architecture can incur a two times cost increase over a single-Region approach. While there are several multi-Region patterns for business continuity, such as running with a hot standby, warm standby, and pilot light, the pattern with the lowest risk of meeting recovery objectives will involve running hot standby, and will double the cost for your workload.

Key guidance

- Availability and continuity of operations goals such as RTO and RPO should be established per workload and aligned with business and IT stakeholders.
- Most availability and continuity of operations goals can be met within a single Region. For goals that cannot be met with a single Region, multi-Region should be considered, with a clear view on tradeoffs between cost, complexity, and benefits.

Multi-Region fundamental 2: Understanding the data

Managing data is a non-trivial problem with multi-Region architectures. The geographical distance between Regions imposes an unavoidable latency, which manifests as the time it takes to replicate data across Regions. Trade-offs between availability, data consistency, and introducing higher orders of magnitude of latency into a workload that uses a multi-Region architecture will be necessary. Whether using asynchronous or synchronous replication you'll need to modify your application to handle the behavioral changes the replication technology imposes. It's very difficult to take an existing application that was designed for a single-Region and make it multi-Region due to challenges around data consistency and latency. Understanding the data consistency requirements and data access patterns for particular workloads is critical to weighing the trade-offs.

2a: Understanding data consistency requirements

The CAP theorem provides a reference for reasoning about the tradeoffs between data consistency, availability, and network partitions, of which only two can be satisfied at the same time for a workload. Multi-Region by definition includes network partitions between Regions, so you have to choose between availability and consistency.

If you select availability of the data across Regions, you will not incur significant latency during transactional writes, because there is a reliance on asynchronous replication of committed data between Regions, resulting in reduced consistency across Regions until the replication completes. With asynchronous replication, when there is a failure in the primary Region, there is a high probability of writes pending replication from the primary Region. This leads to a scenario where the latest data is unavailable until replication resumes, and a reconciliation process is needed to handle in-flight transactions that didn't replicate from the Region that experienced the outage.

For workloads where asynchronous replication is favored, you can use services such as Amazon Aurora and Amazon DynamoDB, which provide asynchronous cross-Region replication. Both Amazon Aurora Global Database and Amazon DynamoDB global tables have default Amazon CloudWatch metrics to aid in monitoring replication lag.

Engineering the workload to take advantage of event-driven architectures is a benefit for a multi-Region strategy because it means the workload can embrace asynchronous replication of data, and enables the reconstruction of state by replaying events. Because streaming and messaging services buffer message payload data in a single Region, a Regional failover/failback process must include a mechanism to redirect client input data flows, as well as reconcile in-flight and/or undelivered payloads stored in the Region that experienced the outage.

If consistency is selected, you will incur significant latency as data are synchronously replicated during transactional writes. When writing to multiple Regions synchronously, if the write isn't successful in all Regions, availability is potentially lowered because the transaction will not commit, and will need to be retried. Retries that attempt to write the data to all Regions synchronously are done at the cost of latency with every attempt. At some point, when the retries have been exhausted, the decision will need to be made to either fail the transaction completely, thereby reducing availability, or commit the transaction to available Regions only, thereby leading to inconsistency. There are quorum forming technologies such as Paxos, which can help replicate and commit data synchronously, but which need significant developer investment.

When writes involve synchronous replication across multiple Regions to meet strong consistency requirements, write latency increases by an order of magnitude. A higher write latency is not something that can typically be retro-fitted into an application without significant changes. Ideally, it must be taken into consideration when the application is first being designed. For multi-Region workloads where synchronous replication is a priority, AWS Partner solutions can help.

2b: Understanding the data access patterns

Workload data access patterns fall into one of the following types: *read-intensive* or *write-intensive*. Understanding this characteristic for a particular workload will guide selection of an appropriate multi-Region architecture.

For read-intensive workloads such as static content that are completely read-only, an active/active multi-Region architecture can be achieved without significant complexity. Serving static content at the edge using a Content Distribution Network (CDN) ensures availability by caching content closest to the end user; using feature sets such as Origin failover within Amazon CloudFront can help achieve this. Another option is to deploy stateless compute in multiple Regions and use DNS to route users to the closest Region to read the content. Route 53 with geolocation routing policy can be used to achieve this.

For read-intensive workloads that have a larger percentage of reads than writes, a *read local, write global* strategy can be used. This entails all writes go to a database in a specific Region with asynchronous replication of data to all the other Regions, and reads can be done in any Region to achieve this. This approach requires a workload to embrace eventual consistency, because local reads may be stale because of increased latency for cross-Region replication of writes.

Aurora Global Database can help with provisioning of Read Replicas in a standby Region that can solely handle all read traffic locally, and a single primary datastore in a specific Region to handle writes. Data is asynchronously replicated from the primary to standby databases (Read Replicas) and the standby databases can be promoted to primary if you need to failover operations to the standby Region. If a workload is better suited for non-relational data models, DynamoDB can be used in this approach as well. Again, the workload needs to embrace eventual consistency, which may require it to be re-written if it wasn't designed for this from the start.

For write-intensive workloads, a primary Region should be selected and the capability to failover to a standby Region should be engineered into the workload. As compared to an active/active approach, a primary/standby approach is less complicated. This is because for an active/active architecture, the workload will need to be rewritten to handle intelligent routing to Regions, establish session affinity, ensure idempotent transactions, and handle potential conflicts.

Most workloads that are looking at multi-Region for resilience won't require an active/active approach. A sharding strategy can be used to provide increased resilience by limiting the blast radius of an impairment across the client base. If you can effectively shard a client base, different primary Regions can be selected for each shard. For example, if you can shard clients so half of the clients are aligned to Region One and half are aligned to Region Two, treating Regions as cells, a multi-Region cell approach can be created, which results in reducing the blast radius of impact for your workload.

The sharding approach can be combined with a primary/standby approach to provide failover capabilities for the shards. A tested failover process will need to be engineered into the workload and a process for data reconciliation will need to be engineered as well to ensure transactional consistency of the data stores after failover. These are covered in greater detail later in this paper.

Key guidance

- There is a high probability that writes pending for replication won't be committed to the standby Region when there is a failure. Data will be unavailable until replication resumes (assuming asynchronous replication).
- As part of failover, a data reconciliation process will be needed to ensure a transactionally consistent state is maintained for datastores using asynchronous replication.
- When strong consistency is required, workloads will need to be modified to tolerate required latency of datastore that synchronously replicates.

Multi-Region fundamental 3: Understanding your workload dependencies

A specific workload might have several dependencies in a Region, such as AWS services used, internal dependencies, third-party dependencies, network dependencies, certificates, keys, secrets, and parameters. To ensure operation of the workload during a failure scenario, there should be no dependencies between the primary Region and the standby Region; each should be able to operate independently of one another. To achieve this, all dependencies in the workload must be scrutinized to ensure they are available within each Region. This is required because a failure in the primary Region should not have an impact in the standby Region. In addition, knowledge of how the workload operates when a dependency is in a degraded state or completely unavailable is imperative, so that solutions can be engineered to handle this appropriately.

3a: AWS services

When designing a multi-Region architecture, an understanding of the specific AWS services that will be used is necessary. The first aspect is understanding what features the service has to enable multi-Region, and if a solution must be engineered to accomplish the multi-Region goals. For example, with Amazon Aurora and Amazon DynamoDB, there is a feature to asynchronously replicate data to a standby Region. Any AWS service dependencies will need to be available in all Regions that a workload is going to run from. To ensure the services that will be used are available in the desired Regions, review the AWS Regional Services List.

3b: Internal and third-party dependencies

For any internal dependencies that a workload has, ensure it's available from the Regions the workload will operate out of. For example, if the workload is composed of many microservices, be knowledgeable about all of the microservices that comprise a business capability. From there, ensure that all of those microservices are deployed in each Region the workload will operate out of.

Cross-Region calls between microservices within a workload is not advised, and Regional isolation should be maintained. This is because creating cross-Region dependencies adds risk of correlated failure, which negates the benefits you are trying to achieve with isolated Regional implementations of the workload. On-premises dependencies might be part of the workload as well, so understanding how characteristics of these integrations could change if the primary Region was to change is imperative. For example, if the standby Region is located farther from the on-premises environment, the increased latency will have a negative impact.

Understanding Software as a Service (SaaS) solutions, software development kits (SDKs), and other third-party product dependencies, and being able to exercise scenarios where these dependencies are either degraded or unavailable will provide more insight into how the chain of systems operates and behaves under different failure modes. These dependencies could be within an application code fromhow secrets are managed externally using AWS Secrets Manager, or a third-party vault solution (such as Hashicorp),to authentication systems having a dependency onIAM Identity Center for federated logins.

Having redundancy when it comes to dependencies can aid in increased resilience. There is also the possibility that a SaaS solution or third-party dependency is using the same primary AWS Region as the workload. If this is the case, you should work with the vendor to determine if their resilience posture matches requirements for the workload.

Additionally, be aware of shared fate between the workload and its dependencies, such as third-party applications. If the dependencies are not available in (or from) a secondary Region after a failover, the workload might not recover fully.

3c: Failover mechanism

The Domain Name System (DNS) is commonly used as a failover mechanism to shift traffic away from the primary Region to a standby Region. Critically review and scrutinize all dependencies the failover mechanism takes. For example, if your workload is using Amazon Route 53, understanding that the control plane is hosted in US-East-1 means you are taking a dependency on the control plane in that specific Region. This is not recommended as part of a failover mechanism if the primary Region is US-East-1 as well. If another failover mechanism is being used, a deep understanding of any scenario in which it wouldn’t operate as expected is necessary. Once this understanding is established, plan for contingency or develop a new mechanism if required. Review Creating Disaster Recovery Mechanisms Using Amazon Route 53 to learn about approaches you can use to failover successfully.

As discussed in the internal dependency section, all microservices that are part of a business capability need to be available in each Region in which the workload is deployed. As part of the failover strategy, the business capability needs to failover together to remove the chance of cross-Region calls. Alternatively, if microservices failover independently, this introduces the potential for undesirable behavior where microservices potentially make cross-Region calls, which introduces latency and could lead to the workload being unavailable in the event of client timeouts.

3d: Configuration dependencies

Certificates, keys, secrets, and parameters are part of the dependency analysis needed when designing for multi-Region. Whenever possible, it’s best to localize these components within each Region so they do not have shared fate between Regions for these dependencies. For certificates, expiration should vary among them, and if possible, in each Region, to avoid a scenario when an expiring certificate (with alarms set to notify in advance) impacts multiple Regions.

Encryption keys and secrets should be Region-specific as well. That way, if there is an error in rotation of a key or secret, the impact is limited to a specific Region.

Lastly, any workload parameters should be stored locally for the workload to retrieve in the specific Region.

Key guidance

- A multi-Region architecture benefits from physical and logical separation between Regions. Introducing cross-Region dependencies at the application layer breaks this benefit. Avoid such dependencies.
- Failover controls should work with no dependencies on the primary Region.
- Coordinating failover at the business capability needs to be done to remove the possibility of increased latency and dependency of cross-Region calls.

Multi-Region fundamental 4: Operational readiness

Operating a multi-Region workload is a complex task that comes with operational challenges that are specific to multi-Region. These include AWS account management, retooled deployment processes, creating a multi-Region observability strategy, creating and testing failover and failback runbook, and then managing the cost. An Operational Readiness Review (ORR) can help teams prepare a workload for production, whether running in a single Region or across multiple Regions.

4a: AWS account management

To deploy a workload across AWS Regions, ensure that all AWS service quotas within an account are in parity across Regions. First, learn all of the AWS services that are part of the architecture, look at the planned usage in the standby Regions, then compare them to the current usage. In some cases, if the standby Region hasn't been used before, you can reference the default service quotas to understand the starting point. Then, across all the services that will be used, request a quota increase using the Service Quotas console (login required) or APIs.

AWS Identity and Access Management (IAM) roles need to be configured in each Region to ensure operators, automation tooling, and AWS services have the appropriate permissions to resources within the standby Region. Regionally isolating roles achieves the Regional isolation we are after for multi-Region architectures. Ensure these permissions are in place before going live with a standby Region.

4b: Deployment practices

With multi-Region capabilities, the deployment of the workload to multiple Regions can be complex. AWS CloudFormation helps with deploying infrastructure to a single or multiple Regions, and can be tailored according to your needs. AWS CodePipeline helps with providing a nearly continuous integration/continuous delivery (CI/CD) pipeline, which has cross-Region actions that allow deployment to Regions that are different from the Region the pipeline is in. This, combined with robust deployment strategies such as blue/green, allows for a minimum to zero downtime deployment.

However, the deployment of stateful capabilities can be more complex when the state of the application or data is not externalized to a persistent store. In these situations, carefully tailor the deployment process to suit your needs. Design the deployment pipeline and process to deploy at one Region at a time, versus multiple Regions simultaneously. This reduces the chance of correlated failures between the Regions. To learn about techniques Amazon uses to automate software deployments, read the Builder Library article Automating safe, hands-off deployments.

4c: Observability

When designing for multi-Region, consider how the health of all components in each Region will be monitored to get a holistic view of Regional health. This could include monitoring metrics for replication lag, which is not a consideration for a single Region workload.

When building a multi-Region architecture, consider observing the performance of the workload from the standby Regions as well. This includes having health checking and canaries (synthetic testing) running from the standby Region, providing an outside view of the health of the primary. In addition, you can use Amazon CloudWatch Internet Monitor to understand the state of the external network and performance of your workloads from an end user perspective. Similarly, the primary Region should have the same observability in place to monitor the standby Region. These canaries should be monitoring customer experience metrics to get an overall health of the workload. This is required because if there was problem in the primary Region, the observability in the primary could be impaired and would impact ability to assess the health of the workload.

In that case, observing outside that Region can provide insight. These metrics should be rolled up into dashboards available in each Region, and alarms created in each Region. Because Amazon CloudWatch is a Regional service, having these in both Regions is a requirement. This monitoring data will be used to make the call to failover from a primary to a standby Region.

4d: Processes, procedures, and testing

The best time to answer the question, ‘When should I failover?’ is long before you need to. Business continuity plans inclusive of people, processes, and technology should all be defined well in advance of an issue, and be tested regularly. Decide on a recovery decision framework. If there is a well-practiced recovery process and the time to recovery is well understood, the point in time to start the recovery process that meets the RTO target through a failover can be chosen. This point in time could be immediately after an issue with the application in the primary Region is identified, or it could be further into an event where recovery options within the application in the Region have been exhausted, and should now start a failover to meet the RTO.

While the failover action itself should be 100% automated, the decision to activate the failover should be made by a human (usually a small number of pre-determined individuals in the organization). Also, the criteria to decide on a failover needs to be clearly defined and globally understood with the organization. These processes can be defined and completed using AWS System Manager runbooks, which allows for complete end-to-end automation and ensures consistency of process running during testing and failover.

These runbooks should be available in the primary and standby Region to start the failover or failback processes. Once this automation is in place, a regular testing cadence should be defined and followed. This ensures that when there is an actual event, the response is run on a well-defined, practiced process that the organization has confidence in. It’s also important to keep in mind the established tolerances for data reconciliation processes. Confirm that the established RPO/RTO requirements are met with the proposed process.

4e: Cost and complexity

Cost implications of a multi-Region architecture are driven by higher infrastructure usage, operational overhead, and resource time. As mentioned previously, the infrastructure cost in a standby Region is similar to the infrastructure cost in a Primary Region when pre-provisioning, resulting in two times the cost. Provision capacity so that it is sufficient for daily operations, but still reserves enough buffer capacity to tolerate spikes in demand – and configure the same limits in each Region.

Additionally, application-level changes might be required to successfully run in a multi-Region architecture if you are adopting an active-active architecture, which can be time and resource intensive to design and operate. At minimum, organizations would need to spend time understanding technical and business dependencies in each Region, and designing failover and failback processes.

Teams should also go through normal failover and failback exercises to feel comfortable with runbooks which will be used during an event. Though incredibly important and crucial to getting the expected outcome from a multi-Region investment, these exercises represent an opportunity cost, and take time and resources away from other activities.

Key guidance

- AWS service quotas need to be reviewed and in parity in all Regions in which the workload will operate.
- The deployment process should target one Region at a time, rather than multiple Regions simultaneously.

- Additional metrics such as replication lag need to be monitored, and are specific to multi-Region scenarios.
- Extend monitoring for the workload beyond the primary Region. Customer experience metrics should be monitored per Region, and measured from outside each Region in which a workload is running.
- Failover and failback need to be tested regularly. Ensure implementation of a single runbook for failover and failback processes that is used during both testing and a live event. Runbooks for testing and live events cannot be different.

Conclusion

This whitepaper discussed common use cases for multi-Region, fundamentals on how to implement a multi-Region architecture, and implications of this approach. These fundamentals can be applied to any workload and used as a framework to aid in decision making about whether or not a multi-Region architecture is the right approach for a particular business.

Contributors

Contributors to this document include:

Technical contributor:

- John Formento, Jr., Principal Solutions Architect, AWS Multi-Region Team

Editorial contributor:

- Lisi Lewis, Sr. Manager, Product Marketing

Further reading

For additional information, refer to:

- *Advanced Multi-AZ Resilience Patterns* (AWS whitepaper)
- Reliability Pillar - AWS Well-Architected Framework
- *Availability and Beyond: Understanding and Improving the Resilience of Distributed Systems on AWS* (AWS whitepaper)
- *AWS Fault Isolation Boundaries* (AWS whitepaper)

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Document published	First publication.	December 20, 2022

Notices

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.