Reliability Pillar

AWS Well-Architected Framework

November 2016

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

The focus of this paper is the Reliability pillar of the AWS Well-Architected Framework. It provides guidance to help customers apply best practices in the design, delivery, and maintenance of Amazon Web Services (AWS) environments.

Introduction

At Amazon Web Services, we understand the value of educating our customers about architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud. As part of this effort, we developed the AWS Well-Architected Framework, which helps you understand the pros and cons of decisions you make while building systems on AWS. We believe well-architected systems greatly increase the likelihood of business success.

The AWS Well-Architected Framework is based on five pillars:

- Security
- Reliability
- Performance Efficiency
- Cost Optimization
- Operational Excellence

This paper focuses on the Reliability pillar and how to apply it to your solutions. Achieving reliability can be challenging in traditional on-premises environments due to single points of failure, lack of automation, and lack of elasticity. By adopting the practices in this paper you will build architectures that have strong foundations, consistent change management, and proven failure recovery processes.

This paper is intended for those in technology roles, such as chief technology officers (CTOs), architects, developers, and operations team members. After reading this paper, you will understand AWS best practices and strategies to use when designing cloud architectures for reliability. This paper does not provide implementation details or architectural patterns. However, it does include references to appropriate resources for this information.

Reliability

The Reliability pillar encompasses the ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or

transient network issues. This paper provides in-depth, best-practice guidance for architecting reliable systems on AWS.

Design Principles

In the cloud, there are a number of principles that can help you increase reliability:

- **Test recovery procedures:** In an on-premises environment, testing is often conducted to prove the system works in a particular scenario. Testing is not typically used to validate recovery strategies. In the cloud, you can test how your system fails, and you can validate your recovery procedures. You can use automation to simulate different failures or to recreate scenarios that led to earlier failures. This exposes failure pathways that you can test and rectify *before* a real failure scenario, reducing the risk of components failing that have not been tested before.
- **Automatically recover from failure:** By monitoring a system for key performance indicators (KPIs), you can trigger an automated response when a threshold is breached. This automation can include notification and tracking of failures, and recovery processes that work around or repair the failure. With more sophisticated automation, it's possible to anticipate and remediate failures before they occur.
- **Scale horizontally to increase aggregate system availability:** Replace one large resource with multiple small resources to reduce the impact of a single failure on the overall system. Distribute requests across multiple, smaller resources to ensure that they don't share a common point of failure.
- **Stop guessing capacity:** A common cause of failure in on-premises systems is resource saturation, when the demands placed on a system exceed the capacity of that system (this is often the objective of denial of service (DoS) attacks). In the cloud, you can monitor demand and system utilization, and automate the addition or removal of resources to maintain the optimal level to satisfy demand without over- or under-provisioning.
- **Manage change using automation**: Changes to your infrastructure should be made using automation. This way, the only changes that need to be managed are changes to the automation.

Definition

Reliability in the cloud is composed of three areas:

1. Foundations
2. Change management
3. Failure management

To achieve reliability, a system must have a well-planned foundation and monitoring in place, with mechanisms for handling changes in demand or requirements. The system should be designed to detect failure and automatically heal itself.

Foundations

Before architecting any system, foundational requirements that influence reliability should be in place. For example, you must have sufficient network bandwidth to your data center and to the Internet (if both are required). These requirements are sometimes neglected because they are beyond a single project's scope. This neglect can have a significant impact on the ability to deliver a reliable system. In an on-premises environment, these requirements can cause long lead times due to dependencies and therefore must be incorporated during initial planning.

AWS sets service limits (an upper limit on the number of each resource your team can request) to protect you from accidently over-provisioning resources, especially in response to a potential denial of service attack, or compromised access keys. Under normal business growth or increased adoption, you will need to have governance and processes in place to monitor and change these limits to meet your business needs. As you adopt the cloud, you may need to plan integration with existing on-premises resources (a hybrid approach). A hybrid model enables the gradual transition to an all-in, cloud approach over time, and therefore it's important to have a design for how your AWS and on-premises resources will interact as a network topology. Finally, you will want to ensure that your IT team receives training and updated processes to support your public cloud usage, and that you have partner or support agreements in place when appropriate.

In AWS, there are a number of different approaches to consider when addressing foundations. The following sections describe how to use these approaches:

- Limit management
- Planning network topology

Limit Management

When you architect systems you need to take into account physical limits and resource constraints. For example, the rate that you can push bits down a fiber optic cable, or the amount of storage on a physical disk. Understanding physical constraints is the first part to designing reliable systems. In addition, with service-based architecture, you need to consider service limits that act to protect the service from breaching Service Level Agreements (rate limits) or design constraints (hard limits). The final piece of limit management to keep in mind is alerting and reporting. It is important to know when you hit a limit or are about to hit a limit, and so you can react.

The default limits for cloud resources created by AWS services are documented by each service. These limits are tracked per account, so if you use multiple accounts, you need to know what the limits are in each account. Other limits may be based on your configuration.

The following are a few examples of resource limits:

- The number of Amazon EC2 instances in an Auto Scaling group
- Provisioned input/output operations per second (IOPS)
- Amazon Relational Database Service (RDS) storage allocated
- Amazon Elastic Block Store (EBS) volume allocations
- Network IO
- Available IP addresses in a subnet or a virtual private cloud (VPC)

Limits are enforced for each AWS Region and for each AWS account. If you are planning to deploy into multiple Regions or use multiple AWS accounts, then you should ensure that you increase limits in the Regions and accounts that you using. Additionally, you should ensure that you have sufficient buffer accounted for in Availability Zones in your Region. If, due to an event in an Availability Zone, you have to request additional resources, ensure that your buffer includes margin for any unhealthy resources that have not been terminated yet.

AWS provides a list of some service limits in the AWS Trusted Advisor and its FAQ, in the AWS documentation, and in the AWS Management Console. You can contact AWS Support to provide your current limits. For rate limits on throttled APIs, the SDKs provide mechanisms (retry, exponential back off) to handle throttled responses. You should evaluate your use cases to decide which scheme works best for you. If the throttling limits have an impact on your application's performance, then you should contact AWS Support to see if there are mitigations or if the limit can be increased.

The best practice is to automate your limit tracking. You can store what your current service limits are in a persistent data store like Amazon DynamoDB. If you integrate your Configuration Management Database (CMDB) or ticketing system with AWS Support APIs, you should be able to automate the tracking of limit increase requests and current limits. If you are integrating with a CMDB, then it is likely that you can store the service limits within that system.

Key AWS Services

The key AWS service that supports a way to identify current service limits is **AWS Trusted Advisor** which provides a list of what limits it returns. The following services are also important:

- **Amazon CloudWatch**: You can use CloudWatch metrics to set alarms that indicate when you are getting close to limits in network IO, provisioned IOPS, Amazon Elastic Block Store (EBS) volume capacity, and ephemeral volume capacity. You can also set alarms so you can be alerted when your Auto Scaling groups are approaching maximum capacity.
- **Amazon CloudWatch Logs**: Metric filters can be used to search and extract patterns in a log event. Log entries are converted to numeric metrics, and alarms can be applied.

Resources

Refer to the following resources to learn more about AWS best practices for identifying limits and managing limits.

Video

- How do I manage my AWS service limits?
- Massive Message Processing with Amazon SQS and Amazon DynamoDB (ARC301) | AWS re:Invent 2013

Tools

- AWS Limit Monitor

Documentation

- AWS Service Limits
- Service Limit Reports Blog Post

Planning Network Topology

When architecting systems using IP-address-based networks you need to plan network topology and addressing in anticipation of future growth and integration with other systems and their networks. You might find that your infrastructure is limited if you do not plan for growth, or you might have difficulties integrating incompatible addressing structures.

Amazon Virtual Private Cloud (VPC) lets you provision a private, isolated section of the AWS Cloud where you can launch AWS resources in a virtual network.

When you plan your network topology, the first step is to define the IP address space itself. Following RFC 1918 guidelines, Classless Inter-Domain Routing (CIDR) blocks should be allocated for each VPC. Consider doing the following things as part of this process:

- Allow IP address space for more than one VPC per Region.

- Consider cross-account connections. For example, each line of business might have a unique account and VPCs. These accounts should be able to connect back to shared services.
- Within a VPC, allow space for multiple subnets that span multiple Availability Zones.
- Always leave unused CIDR block space within a VPC.

The second step in planning your network topology is to ensure the resiliency of connectivity:

- How are you going to be resilient to failures in your topology?
- What happens if you misconfigure something and remove connectivity?
- Will you be able to handle an unexpected increase in traffic/use of your services?
- Will you be able to absorb an attempted denial of service (DoS) attack?

AWS has many features that will influence your design. How many VPCs do you envision you will be using? Will you be using Amazon VPC Peering between your VPCs? Will you be connecting virtual private networks (VPNs) to any of these VPCs? Are you going to use AWS Direct Connect or the Internet?

A best practice is to always use private address ranges as identified by RFC 1918 for your VPC Classless Inter-Domain Routing (CIDR) blocks. The range you pick should not overlap your existing use, or anything you plan on sharing address space with via VPC Peering or VPN. In general, you need to make sure your allocated range includes sufficient address space for the number of subnets you need to deploy, the potential size of Elastic Load Balancing (ELB) load balancers, and your servers deployed within your subnets. In general, you should plan on deploying large VPC CIDR blocks. Note that VPC CIDR blocks and subnet CIDR blocks cannot be changed after they are deployed. Keep in mind that deploying the largest VPC possible results in over 65,000 IP addresses. The base 10.x.x.x address space means that you can use over 16,000,000 IP addresses. You should err on the side of too large instead of too small for all these decisions.

The connectivity from a VPC is governed through route table entries. An Internet Gateway (IGW), NAT Gateway, Virtual Private Gateway (VGW), or VPC

Peering Gateway are all exposed to a subnet through an entry in its route table. When you plan your network consider the VGW and VPC Peering that you want.

Another option for setting up networking between VPCs is to use VPN appliances. Commonly used appliances are available on the AWS Marketplace.

You should consider the resiliency and bandwidth requirements that you need when you select the vendor and instance size on which you need to run the appliance. For example, if you choose to connect your VPC to your data center via an AWS Direct Connect connection, you should have a redundant connection fallback either through a second AWS Direct Connect connection from another provider, or through the Internet. If you use a VPN appliance that is not resilient in its implementation, then you should have a redundant connection through a second appliance. For all these scenarios, you need to define acceptable time to recovery (TTR), and test to ensure you can meet those requirements.

You should use existing standards for protecting your resources within this private address space. A subnet or set of subnets (one per Availability Zone) should be used as a barrier between the Internet and your applications. In an on-premises environment, you often have firewalls with features to deflect common web attacks, and you often have load balancers or edge routers to deflect denial of service (DoS) attacks, such as SYN floods. AWS provides many services that can provide this functionality, such as an integrated Web Application Firewall in Amazon CloudFront, Elastic Load Balancing, and features of AWS virtual networking like VPC Security Groups and Network Access Control Lists (ACLs). You can augment these features with virtual appliances from AWS Partners and the AWS Marketplace to meet your needs.

Key AWS Services

The key AWS service that supports your network planning is **Amazon Virtual Private Cloud (VPC)**, which allows you to allocate private IP address ranges to either provide non-Internet accessible resources or to extend your data center. The following services and features are also important:

- **AWS Direct Connect**: AWS Direct Connect can be used to give a private dedicated connection to AWS for possible lower latency and consistent performance to and from AWS.

- **Amazon Elastic Compute Cloud (EC2)**: If you choose to implement VPNs between your networks, you will be running VPN appliances on Amazon EC2.
- **Amazon Route 53:** Amazon Route 53 is a Domain Name System (DNS) service that is integrated directly with ELB, and can help provide a layer of defense in the event of a denial of service (DoS) attack.
- **Elastic Load Balancing:** ELB provides load balancing across Availability Zones, and it integrates with Auto Scaling to help create a self-healing infrastructure.

Resources

Refer to the following resources to learn more about AWS best practices for network planning.

Video

- AWS re:Invent 2015 | (ARC403) From One to Many: Evolving VPC Design
- AWS re:Invent 2015 | (SEC306) Defending Against DDoS Attacks

Documentation

- Amazon Virtual Private Cloud Product Page
- Amazon Virtual Private Cloud Documentation
- Transit VPC article on AWS Answers
- Amazon EC2 Instance Types Product Page
- Amazon EC2 Instance Types Documentation
- AWS Marketplace for Network Infrastructure
- AWS Best Practices for DDoS Resiliency Whitepaper
- Amazon VPC Connectivity Options Whitepaper

Change Management

When you are aware of how change affects a system you can plan proactively. By monitoring your system you can quickly identify trends that could lead to capacity issues or SLA breaches. In traditional environments, change-control

processes are often manual and must be carefully coordinated with auditing to effectively control who makes changes and when they are made.

Using AWS, you can monitor the behavior of a system and automate the response to KPIs. For example, you can automatically add more servers as a system gains additional users. You can control who has permission to make system changes and audit the history of these changes.

In AWS, there are a number of different approaches to consider when addressing change management:

- Changes in demand
- Monitoring
- Change execution

Changes in Demand

It is often when your product is at its most popular that any deficiencies in reliability are found. By ensuring that you have sufficient capacity to meet demand, you can cope with component failures. Testing for high demand to prove reliability is critical to success.

Elasticity refers to the virtually unlimited capacity of the cloud, where the vendor is responsible for capacity management and provisioning of resources. By taking advantage of APIs, you can programmatically vary the amount of cloud resources in your architecture dynamically. This allows you to horizontally scale components in your architecture, and automatically increase the number of resources during demand spikes to maintain availability and decrease capacity during lulls to reduce costs.

In AWS this is normally accomplished using Auto Scaling, which helps you to scale your Amazon EC2 capacity up or down automatically according to conditions you define. Auto Scaling is generally used with Elastic Load Balancing (ELB) to distribute incoming application traffic across multiple Amazon EC2 instances in an Auto Scaling group. Auto Scaling is triggered using scaling plans that include policies that define how to scale (manual, schedule, or demand) and the metrics and alarms to monitor in Amazon CloudWatch. CloudWatch metrics are used to trigger the scaling event. You can use standard

Amazon EC2 metrics such as CPU utilization, load balancer observed request/response latency, and even custom metrics which might originate from application code on your EC2 instances.

Auto Scaling can be used without a load balancer or CloudWatch metrics. For example, you can scale a set of workers that process a queue, or use custom code to trigger Auto Scaling in response to a business event or a time of day.

When you architect with elasticity keep in mind two key considerations: First, how quickly you can provision new resources? Second, what is the size of margin between supply and demand that you need to be aware of so that you can cope with the rate of change in demand and also with resource failures?

You can optimize the speed of provisioning by reducing the startup and configuration tasks your instances run at boot. This is often done using a pre-baked Amazon Machine Image (AMI) in a "bakery model." Note that this optimization will be at the expense of configurability of these instances, so you need to base this decision on your need for speed versus your need for flexible configurations. The margin between supply and demand can be very wide when you start development, and can be reduced through experimentation, load testing, and confidence building as you move into test and production.

In AWS, pre-warming is the mechanism you can use to deal with known rapid change in capacity. Outside of application functions like pre-populating a cache, or scaling out a cluster and/or group of instances, a number of AWS services can be pre-warmed or initialized programmatically:

- **Amazon Elastic Block Store (EBS):** Can be started from snapshots. You can reduce the latency of an IO operation on the block's first access by reading all blocks before accepting load.
- **Amazon DynamoDB**: IO can be programmatically changed through a table's provisioned throughput settings, and read/write capacity units.
- **Amazon Relational Database (RDS)**: Scales out read replicas for read-heavy database workloads.
- **Amazon API Gateway**: Although bounded by account limits, both stage- or method-level throttling limits can be set by using REST APIs.

- **AWS Lambda**: Depending on the language that you are using, AWS Lambda might need a way of "priming" its capacity before any expected sharp increase in capacity is required.

You should follow best practices for software development in your infrastructure as code implementation. All changes should be tested at multiple levels. You should always conduct load testing against your implementation in conditions that are as close to real conditions as possible before deploying your automation to production. For more information and best practices, see the Load Testing and Benchmarking sections of *The Performance Efficiency Pillar of the AWS Well-Architected Framework* whitepaper.

Key AWS Services

The key AWS service that supports automated demand management is **Auto Scaling**, which allows you to define Auto Scaling groups and integrate the CloudWatch alarms that trigger scaling actions. The following services and features are also important:

- **Amazon CloudWatch Events**: These events can be used to kick off complex scaling operations under very specific control.
- **AWS Elastic Beanstalk**: AWS Elastic Beanstalk configures Auto Scaling and Elastic Load Balancing for you. It also manages the operating system and the application container.
- **AWS OpsWorks**: AWS OpsWorks provides automation code to allow you to scale your application based on time or load.

Resources

Refer to the following resources to learn more about AWS best practices for automated demand management:

Video

- AWS re:Invent 2015 | (DVO304) AWS CloudFormation Best Practices
- AWS re:Invent 2014 | (ARC317) Maintaining a Resilient Front Door at Massive Scale (Scryer)

Documentation

- Auto Scaling Documentation

- Amazon CloudWatch Events Documentation
- AWS Elastic Beanstalk Documentation
- AWS OpsWorks Documentation
- Initializing EBS Volumes Documentation
- Managing Your AWS Infrastructure at Scale Whitepaper

Monitoring

Change management cannot be achieved without detailed monitoring and alarming. While monitoring from within an operating system is well understood, monitoring in the cloud offers new opportunities. Instead of using old de-facto standard methods like SNMP, cloud providers have developed customizable hooks and insights into everything from instance performance to network layers, down to request APIs themselves.

Monitoring at AWS consists of five distinct phases:

1. Generation
2. Aggregation
3. Real-time processing and alarming
4. Storage
5. Analytics

Generation

First determine which services and/or applications require monitoring, define important metrics and how to extract them from log entries if necessary, and finally create thresholds and corresponding alarm events. AWS makes an abundance of monitoring and log information available for consumption, which can be used to define change-in-demand processes. The following is just a partial list of services and features that generate log and metric data.

- Amazon EC2 Container Service (ECS), Amazon EC2, Classic Load Balancers, Application Load Balancers, Auto Scaling, and Amazon EMR publish metrics for CPU, network IO, and disk IO averages.

- Amazon CloudWatch Logs can be enabled for Amazon Simple Storage Service (S3), Classic Load Balancers, and Application Load Balancers.
- Amazon VPC Flow Logs can be enabled on any or all elastic network interfaces (ENIs) within a VPC.
- AWS CloudTrail logs all API events on an account-by-account basis.
- Amazon CloudWatch Events delivers a real-time stream of system events that describes changes in AWS services.
- AWS provides tooling to collect operating-system-level logs and stream them into Amazon CloudWatch Logs.
- Custom Amazon CloudWatch metrics can be used for metrics of any dimension.
- Amazon ECS and AWS Lambda stream log data to CloudWatch Logs.

Aggregation

Amazon CloudWatch and Amazon Simple Storage Service (S3) serve as the primary aggregation and storage layers. For some services, like Auto Scaling and ELB, default metrics are provided “out the box” for CPU load or average request latency across a cluster or instance. For streaming services, like Amazon VPC Flow Logs or AWS CloudTrail, event data is forwarded to Amazon CloudWatch Logs, and you need to define and apply filters to extract metrics from the event data. This gives you time series data, and you can define an array of CloudWatch alarms to trigger alerts.

Real-time Processing and Alarming

Alerts can trigger Auto Scaling events, so that clusters react to changes in demand. Alerts can also be sent to Amazon Simple Notification Service (SNS) topics, and then pushed to any number of subscribers. For example, Amazon SNS can forward alerts to an email alias, so that technical staff can respond. Alerts can be sent to Amazon Simple Queue Service (SQS), which can serve as an integration point for third-party ticket systems. Finally, AWS Lambda can also subscribe to alerts, providing users an asynchronous serverless model that reacts to change dynamically.

Storage and Analytics

Amazon CloudWatch Logs also supports subscriptions that allow data to flow seamlessly to Amazon S3. As CloudWatch logs and other access logs arrive in Amazon S3, you should consider using Amazon EMR to gain further insight and value from the data itself.

There are a number of tools provided by partners and third parties that allow for aggregation, processing, storage, and analytics. Some of these tools are New Relic, Splunk, Loggly, Logstash, CloudHealth, and Nagios. However, generation outside of system and application logs is unique to each cloud provider, and often unique to each service.

An often overlooked part of the monitoring process is data management. You need to determine retention requirements for monitoring data, and then apply lifecycle polices accordingly. Amazon S3 supports lifecycle management at the S3 bucket level. This lifecycle management can be applied differently to different paths in the bucket. Toward the end of the lifecycle you can transition data to AWS Glacier for long-term storage, and then expiration, after the end of the retention period is reached.

Key AWS Services

The key AWS service that supports monitoring is **Amazon CloudWatch**, which allows for easy creation of alarms that trigger scaling actions. The following services and features are also important:

- **Amazon S3**: Acts as the storage layer, and allows for lifecycle policies and data management.
- **Amazon EMR**: Use this service to gain further insight into log and metric data.

Resources

Refer to the following resources to learn more about AWS best practices for monitoring.

Video

- AWS re:Invent 2015 | (DVO315) Log, Monitor and Analyze your IT with Amazon CloudWatch
- AWS re:Invent 2015 | (BDT312) Application Monitoring: Why Data Context Is Critical

Documentation

- Amazon CloudWatch Documentation
- Amazon CloudWatch Logs Documentation
- View Amazon EMR Log Files Documentation

Change Execution

With infrastructure as code, change management takes on the form of software development. Changes in infrastructure can be described as *diffs* between running environments and objects that exist in source control. You can set up development, test, and production environments that allow you to effectively test your changes before deploying them. You can also test your complete deployment of any new applications or software in these environments. In non-cloud environments, it is extremely rare that your development and test environments would be the same as your production environment. Using the cloud, you can deploy the same networking, firewalls, and ingress and egress paths that you have in production on premises. The difference is often in the permissions needed to perform the deployments. You can have the same log aggregation services, monitoring, etc. All of your automation should be identical for all environments. This allows you to lower the risks involved in deploying to production because you have deployed successfully in the development and test environments.

There are multiple methods of deployment: deployment in place, blue-green (or red-black) deployments, and canary deployments. The deployment of new application versions using automation becomes a normal operational task instead of a change task. The changes that you need to manage are now changes to the automation, which can be performed according to your existing change management processes.

In AWS, the key service that enables infrastructure as code is AWS CloudFormation. You can deploy the various parts of your infrastructure and applications as separate AWS CloudFormation stacks. Your stacks should be loosely coupled and simple. You can automate these deployments using any of the AWS Software Development Kits (SDKs) in the language of your preference.

You can use other AWS services to assist in your deployments. AWS CodeDeploy allows you to automate the steps in the deployment of your

systems. AWS CodePipeline can be used to continuously deliver your systems. AWS OpsWorks uses Chef, so customers that heavily use Chef can use their existing Chef recipes. AWS Elastic Beanstalk allows easy deployment of code in a load balanced, auto scaling environment where you don't need to maintain the operating system or the language run time container. AWS Service Catalog allows you to create standard AWS CloudFormation templates and then publish them to internal users for use. AWS Config rules can determine and enforce that changes are within compliance.

You can implement blue-green or canary deployment using AWS SDKs. AWS has published a whitepaper that describes popular methods of blue-green deployments. This whitepaper is listed in the references section. Often multiple deployment methodologies are used, depending on the technologies used and systems deployed. You can consult with an AWS Solutions Architect for advice on how the technologies can best fit into your existing processes, or on how you could define new processes.

Key AWS Services

The key AWS service that supports an infrastructure as code methodology is **Amazon CloudFormation**, which allows you to manage your infrastructure and systems in the same manner that you manage the code that you deploy. The following services and features are also important:

- **AWS CodePipeline**: Allows you to continuously deliver systems.
- **AWS CodeDeploy**: Allows you to deploy code in place in existing systems.
- **AWS Elastic Beanstalk**: Allows you to easily deploy code in many languages without having to manage the operating system or language container.
- **AWS OpsWorks**: Allows you to use your existing Chef recipes to deploy your applications on AWS.
- **AWS Identity and Access Management (IAM)**: Allows you to manage access and permissions to the AWS APIs that you use in various environments.
- **AWS Service Catalog**: Builds on top of AWS CloudFormation to allow you to publish templates to your internal customers.

- **AWS Config**: Allows you to enforce compliance across change sets and resources.

Resources

Refer to the following resources to learn more about AWS best practices for infrastructure as code.

Video

- AWS re:Invent 2015 | (DVO304) AWS CloudFormation Best Practices
- Accelerating DevOps Pipelines with AWS

Documentation

- Blue/Green Deployments on AWS Whitepaper
- AWS CloudFormation Documentation
- AWS Tools and Software Development Kits Product Page
- AWS CodePipeline Documentation
- AWS CodeDeploy Documentation
- AWS Identity and Access Management Documentation
- AWS OpsWorks Documentation
- AWS Elastic Beanstalk Documentation
- AWS Service Catalog Documentation

Failure Management

In any complex system it is reasonable to anticipate that failures will occur, and it is important to know how to become aware of these failures, respond to them, and prevent them from happening again.

In AWS, you can take advantage of automation to react to monitoring data. For example, when a particular metric crosses a threshold, you can trigger an automated action to remedy the problem. Also, rather than trying to diagnose and fix a failed resource that is part of your production environment, you can replace it with a new one and carry out the analysis on the failed resource out of band. Since the cloud enables you to stand up temporary versions of a whole

system at low cost, you can use automated testing to verify full recovery processes.

In AWS, there are a number of different approaches to consider when you address failure management:

- Data durability
- Withstanding component failures
- Planning for recovery

Data Durability

Data loss is the worst failure mode for any system. For business continuity planning, a business will define an acceptable recovery point objective (RPO), which is defined as the time duration for which data will be lost in the event of an incident. They will also define a recovery time objective (RTO) that is defined as the time it takes to restore functionality. Durability includes the durability of your backups. These times are usually defined as an amount that is both achievable and affordable, and vary for different systems and data.

Best practice is to regularly test your backup procedure through restoring and validating the data. You should prove that you can continue normal operations with the restored data and have no dependencies on the affected context. Common issues are lack of access to documentation, software, tape machines, and encryption keys. Access permissions to the backed up data should be governed to the same level of security as access to the data itself.

Amazon S3 natively provides 99.999999999 percent ("11 9s") durability. Snapshots of AWS services are stored are stored in Amazon S3. When you use Amazon EBS we recommend that you take snapshots frequently to increase durability. This will allow you to restore to a recent snapshot, if a failure should occur. The initial Amazon EBS snapshot is the full image, and every snapshot thereafter is an incremental snapshot, which reduces cost. Amazon RDS takes regular snapshots, and you can take point-in-time snapshots upon demand. An initial Amazon RDS snapshot is taken upon creation, and all snapshots thereafter are incremental, allowing the snapshots to take minimal space.

Other AWS services that have persistent storage include Amazon Elastic File System (EFS), Amazon SQS, Amazon Kinesis, and Amazon DynamoDB. These services offer data durability as part of their service. AWS Key Management Service (KMS) helps manage the access to encryption keys, and also manages the lifecycle of those keys. You can deprecate the use of keys for encrypting data, but continue to use keys to decrypt data until the lifecycle of the data requires deletion of the data, which then removes the capability of using the key.

You can use software RAID on Amazon EBS volumes to get more durability without having to take regular snapshots. However, this requires more expense because you are using more Amazon EBS volumes. There is often a performance degradation because you are writing to more than one disk on each transaction. Provisioned IOPS can help (also at an expense), but often the highest durability requirements are on the highest performance systems. If you are using ephemeral storage on instances, you need to use the backup software that comes with the application to take the snapshots.

The key considerations for choosing your data store for durability are: durability requirements, ease of taking consistent snapshots, speed of taking consistent snapshots, and speed of restoring those snapshots.

Key AWS Services

The key AWS service that supports a durable backup strategy is **Amazon S3**, which provides extremely high durability of data. The following services and features are also important:

- **AWS KMS**: Allows you to manage encryptions keys that you can easily use with AWS services.
- **Amazon EBS**: Provides a managed service for block storage available to EC2 instances.

Resources

Refer to the following resources to learn more about AWS best practices for choosing which storage to use and how to use other services that have storage within them.

Video

- AWS re:Invent 2014 | (SOV203) Understanding AWS Storage Options

- AWS re:Invent 2015 | (SEC301) Strategies for Protecting Data Using Encryption in AWS

Documentation

- Amazon S3 Documentation
- AWS KMS Documentation
- Amazon EBS Documentation
- Amazon RDS Documentation
- Amazon DynamoDB Documentation
- Amazon SQS Documentation
- AWS Kinesis Documentation

Withstanding Component Failures

When you architect for reliability, understanding and removing single points of failure is key. This is often done through *load sharing*, where additional resources share load to mitigate a single component failure. This load sharing applies at all levels, from servers to data centers.

In the cloud, the best practice is to define a mean time to recovery (MTTR) instead of defining mean time between failures (MTBF) of components. The cloud offers the capability to perform this using automation, and to notify personnel about the incident while it is being repaired.

Systems that do not have state can be scaled by bringing up additional identical resources (*horizontal scaling*). By combining horizontal scaling with load balancing you can ensure that a similar load is distributed to each horizontal implementation. When you build systems and software that are resistant to service failures you will increase your system's ability to withstand failure during recovery. You should decouple your systems so that you can avoid direct cascading failures. Systems that are decoupled are isolated from the failure of dependent systems. They can continue to operate while recovery is being performed.

The cloud enables easy deployment and automation of horizontally scaled systems. AWS offers services for load balancing, queuing, sending notifications, automating recovery, and using multiple locations.

AWS offers features and services such as multiple Availability Zones in multiple AWS Regions, Classic Elastic Load Balancers, Application Load Balancers, Amazon SQS, Amazon SNS, and Auto Scaling groups. Some AWS services are inherently highly tolerant to failure like Amazon S3, Amazon DynamoDB, AWS Lambda, and ELB. In addition, AWS services are available through application programming interfaces (APIs), allowing you to automate all aspects of your fault tolerance.

ELB offers health checks that allow you to report the status of your resource, so that the service can replace unhealthy resources. These health checks should use the same mechanisms that you use in normal operation, for example a database query to Amazon RDS (data plane), rather than an Amazon RDS API call (control plane). Health checks should report issues local to the resource, where replacing the resource could fix the issue. If the issue is caused by a downstream dependency, then your health check should not report a problem because this will cause the unnecessary termination of your resource rather than fixing the dependency.

Key AWS Services

The key AWS tools that support testing to withstand component failure are the **AWS SDKs,** which enable you to automate recovery from failure and to inject simulated failures. The following services and features are also important:

- **ELB**: Load balancing across multiple instances allows you to continue to provide your service in the event of instance failures, monitor the health of your instance, and automatically replace failed instances.
- **Amazon SQS**: Allows you to decouple actions from requests and scale your processing independently from your request generation.
- **Amazon SNS**: Allows you to use an event-driven notification model to perform actions.
- **Auto Scaling**: Allows you to add and remove resources based on demand.

Resources

Refer to the following resources to learn more about AWS best practices for building fault tolerant systems.

Video

- Resiliency Through Failure – Netflix's Approach to Extreme Availability in the Cloud
- AWS re:Invent 2014 | (ARC309) Building and Scaling Cloud Drive to Millions of Users
- AWS re:Invent 2014 (PFC305) Embracing Failure: Fault-Injection and Service Reliability
- AWS re:Invent 2015 (ARC401) Cloud First: New Architecture for New Infrastructure

Documentation

- AWS Developer Tools and SDKs
- AWS Global Infrastructure
- Amazon SQS Documentation
- Auto Scaling Documentation
- Elastic Load Balancing Documentation
- Amazon SNS Documentation
- Benchmarking Availability and Reliability in the Cloud

Planning for Recovery

When components fail, or when a large-scale disruption affects your ability to operate your systems, it is critical that you know what happens next. You need to have a disaster recovery plan that is well understood, proven, and current. The resiliency of your architecture must be thoroughly tested.

Testing Resiliency: Your normal development processes should involve component testing and how they recover from failure. In production, failures are often caused by the interaction between multiple components. Therefore you should test your architecture holistically, at production scale. Your tests

should examine how the architecture only fails, and also how it recovers. Historically, recovery pathways have often been the least tested part of a system. You should have a playbook that defines how you handle failures. Use failure injection to simulate failures. When failures occur in production be ready with a root cause analysis (RCA) process that allows you to learn from the failure, and mitigate against it happening in the future. Finally you should plan for *game days*, where your organization executes large scale tests that prove not just your architecture but also your playbook and RCA process.

Disaster Recovery

You need to test your disaster recovery processes and procedures to ensure that they work and that the team knows how to use them. Regular practice ensures that when an incident actually occurs, the processes and procedures will be familiar and followed. Any problems that you identify should be well documented to indicate both the root cause and the steps taken to ensure that the error is corrected. If possible, this information should be widely shared in a knowledge base in order to prevent similar errors from happening in other systems.

It is critical that the versions in your disaster recovery site are the same as the versions in your primary site. If the two sites become out of sync this is known as *drift*. A good way to prevent drift is to ensure that your delivery automation makes the new versions of everything are available in all locations. Your disaster recovery process should include the plan for protecting the business while it operates in recovery mode. This plan should enumerate the steps that are required to bring your primary site back in sync with your recovery data.

The AWS Cloud is built around Regions and Availability Zones (AZs). A Region is a physical location in the world where we have multiple Availability Zones. Availability Zones consist of one or more discrete data centers, each with redundant power, networking, and connectivity, housed in separate facilities. Our Availability Zones offer you the ability to operate production applications and databases that are more highly available, fault tolerant, and scalable than those operated from a single data center. The AWS Disaster Recovery whitepaper, listed in the resources section, summarizes the approaches for disaster recovery ranging from on-premises systems to the AWS Cloud.

When storing critical files in Amazon S3 you should enable versioning, disable delete version permissions in your IAM entities, and use object lifecycle policies

to govern the deletion of data. This ensures that accidental or malicious deletion of data doesn't occur. You should use a second AWS account for your disaster recovery and write cross-account permissions so that the primary AWS account can only add content, not delete it. For your second account, you need to ensure that all limit increases that you request for your primary account are also requested for your disaster recovery account. AWS provides APIs to allow sharing of resources such as objects, Amazon Machine Images (AMIs), Amazon EBS snapshots, Amazon RDS snapshots, and Amazon DynamoDB Streams.

Using AWS services, you can automate the delivery of all your systems to another AWS Region or account. You can then ensure that the new versions are used. Automated testing will enable you to perform game day exercises for disaster recovery procedures and load testing of your disaster recovery implementation. If you choose to implement an active-passive failover configuration, you can use Amazon Route 53 health checks to govern when you will enable the disaster recovery Region or account to accept traffic, and when you will fail back to your primary Region or account.

You should ensure that your automation contains sufficient abstraction such that it runs in both AWS Regions and accounts. You also need to ensure your data access and retention policies are validated.

Consider the following when you choose your disaster recovery scheme:

- Complexity of cross account access management for common functionality like deployment, investigations, and data upload, etc.
- Impact of Region-based keys in AWS Key Management Service. Snapshots alone will not allow you to restore a snapshot of encrypted data based on a key that is unavailable in that Region.
- Effectiveness of health checks. You will want to ensure that you are not spuriously failing over to your disaster recovery implementation on a transient error in a single subsystem.
- Latency of data replication if using a second Region. The speed with which you can replicate data to another Region will affect your RPO.

Key AWS Services

The key AWS service that supports a Disaster Recovery method is **AWS Identity and Access Management**, which allows you to govern access to

data needed in the event of a disaster. The following services and features are also important:

- **Amazon S3**: This service's versioning, lifecycle management, cross region replication, and highly durable storage makes it a key service for storing data.
- **Amazon Glacier**: Glacier is a key service related to Amazon S3 that is used to archive data.
- **AWS Key Management Service**: Understanding the regionality of AWS KMS keys is essential to your ability to decrypt snapshots encrypted with AWS KMS.
- **Amazon Route 53**: Provides health checks that you can use to detect failure and assist in your disaster recovery procedures.

Resources

Refer to the following resources to learn more about AWS best practices for disaster recovery on AWS.

Video

- AWS re:Invent 2014 | (BAC404) Deploying High Availability & Disaster Recovery Architectures with AWS
- AWS re:Invent 2014 (SDD424) Simplifying Scalable Distributed Applications Using DynamoDB Streams

Documentation

- Amazon S3 Object Versioning Documentation
- Amazon S3 Object Lifecycle Management Documentation
- Amazon S3 Cross Region Replication Documentation
- AWS KMS Documentation
- Amazon Route 53 Health Check Documentation
- Backup, Archive, and Restore Approaches Using AWS Whitepaper
- AWS Disaster Recovery Whitepaper

Conclusion

Reliability is an ongoing effort. When incidents occur, treat them as opportunities to improve the reliability of your architecture. By regularly testing failure pathways, you will be more likely to catch errors before they reach production. If an error does occur in production, your well-tested architecture will be ready to mitigate failures. Your disaster recovery effort can be easier thanks to the AWS services and their programmatic functions.

AWS strives to help you build value by adding systems that are highly resilient, responsive, and adaptive. To make your architectures truly reliable, you should use the best practices discussed in this paper.

Contributors

The following individuals and organizations contributed to this document:

- Philip Fitzsimons, Sr Manager Well-Architected, Amazon Web Services
- Rodney Lester, Principal Professional Services Consultant, Amazon Web Services
- Michael Wallman, Sr. Professional Services Consultant, Amazon Web Services

Further Reading

For additional help, please consult the following sources:

- AWS Well-Architected Framework