Cloud Resiliency Guidance

Introduction

Cloud resiliency refers to the architecture design practices that contribute to applications, running in AWS, that are resilient to service disrupting events. Service disrupting events can include:

- Regional outages
- Service instability
- Platform instability
- Control plane instability
- Connectivity issues

Guiding Principles

Southwest Airlines critical business and operations functions depend on applications running in cloud environments hosted by AWS. In order to satisfy Southwest business function requirements and meet availability and recoverability targets, specific guidance and reviews must be in place to support positive resiliency outcomes.

Recently, there have been service disruptions, within the AWS platforms and infrastructure, that have negatively impacted technology services that support Southwest Airlines business and operations.

Analysis of these events has revealed underlying design flaws that should be addressed in existing designs and new designs.

Goals & Objectives

The objectives for this guidance on resiliency are to:

1. Evaluate and understand potential issues and capabilities that contribute or detract from the resiliency of applications running in AWS
2. Provide guidance for resiliency that maps to desired business outcomes
3. Establish architectural patterns that clearly describe how applications can be designed to avoid unnecessary outages
4. Describe best practices for implementing a service recovery and testing program
5. Recommended changes to architecture practices to enhance focus on application resiliency

Document Scope

Out of Scope

The following topics are considered out of scope for this Architecture Design Document:

- While health monitoring may be covered as a topic to manage component failures, general observability and dashboarding of application health is outside the scope of this document.
- The resiliency of data platforms, including operational, analytical, and streaming data platforms will be covered in Cloud Data Resiliency Guidance.

Resiliency Capabilities Overview

The following capabilities are relevant to resiliency in cloud. A brief description and discussion of relevance is included, but full documentation of these capabilities is beyond the scope of this document.

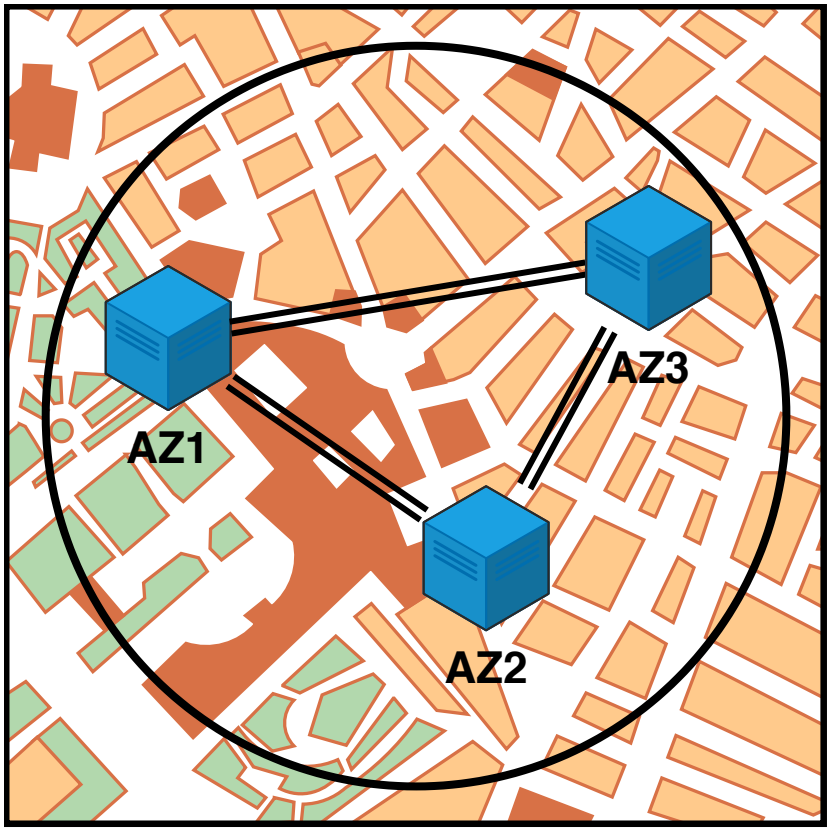
AWS Global Infrastructure Design

AWS provides globally available infrastructure for use by their customers. That infrastructure is comprised of data centers, Availability Zones (AZ)s, and Regions. The most up to date information on AZs and Regions can always be found on the AWS Website.

Availability Zones

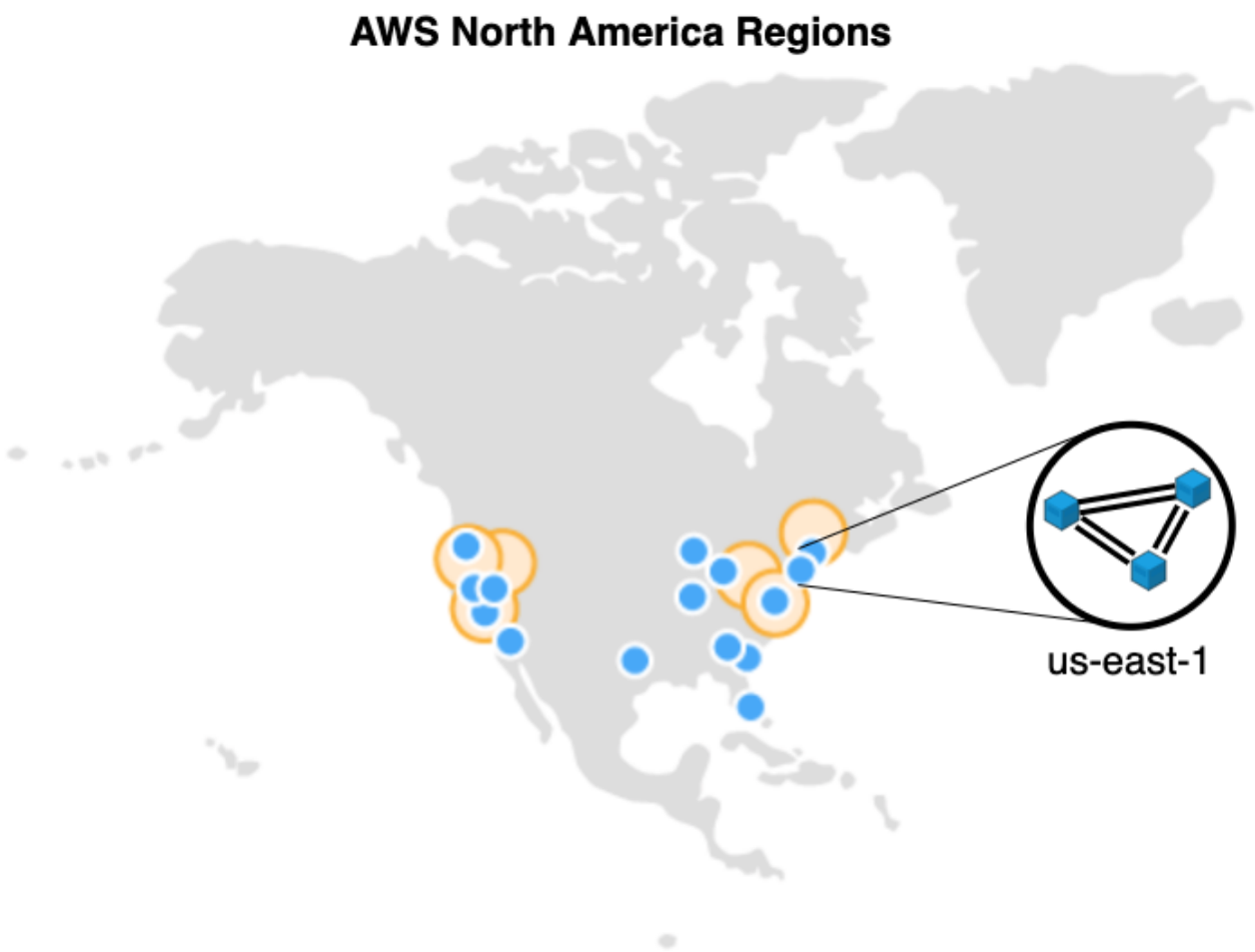
An Availability Zone (AZ) is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. For context, an AZ is the equivalent of the SWA SDC or WNDC data

center.



AWS Regions

AWS has the concept of a Region, which is a physical location around the world where they cluster data centers. Each group of logical data centers are referenced as an Availability Zone. Each AWS Region consists of multiple, isolated, and physically separate AZs within a geographic area.



AWS Service Fault Domains

Solution resiliency is a byproduct of individual service resiliency. The first aspect of any solution that should be examined is the resiliency of each component that the solution is comprised of. The table below describes patterns of component resiliency that are used throughout this guidance document.

Service Domain	Description	Examples
Global	These services are available even if a Region fails	Route 53, IAM, CloudFront
Region	These services are available even if an Availability Zone fails	S3, KMS, Lambda
Availability Zone	These services will fail if an Availability Zone fails. Additional steps should be taken to provide for regional resiliency	EC2, EBS, EFS

Service Domain	Description	Examples
Instance	An instance may fail if a platform within an Availability Zone fails. Additional steps should be taken to examine and understand the availability and recovery characteristics of the platform that the instance runs within.	EKS, Container

Component Scaling

Scaling is a key consideration when planning for application and component level resiliency. Scaling considerations need to balance the need to control costs with those of ensuring resiliency in the event of a failure. For services with critical business functions, (Tier 0, Tier 1), scaling events should be avoided during a failover event, as contention for Control Plane functionality and resources can cause scaling events to fail. For these reasons it is important to tailor the scaling of application components to the overall desired availability. The table below maps component level scaling to the component resiliency patterns above.

Component Resiliency	Scaling Description
Global Component	- Scaling managed by AWS
Active / Active Component	- Components scaled to support 100% of the application workload in a single region
Active / Passive with Automated Recovery	- Active region scaled to support 100% of the application workload - Passive region is also scaled to support 100% of the application workload. This prevents the passive region from failing to service requests adequately during a failover scenario.
Active / Passive with Manual Recovery	- Active region scaled to support 100% of the application workload - Passive region is also scaled to support 100% of the application workload. This prevents the passive region from failing to service requests adequately during a failover scenario.
Active / Standby	- Active region scaled to support 100% of the application workload
Single Region	- Single region scaled to support 100% of the application workload

Overview of Key Services

In addition to the AWS Infrastructure Design, there are key AWS services and capabilities that, when used correctly, can contribute to the resiliency of applications.

Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) service. Amazon Route 53 effectively connects user requests to infrastructure running in AWS and can also be used to route users to infrastructure outside of AWS. Below are features of Route53 that are relevant for resiliency and failover:

- DNS Health Checks – Allows you to monitor the health of your resources such as web servers and email servers. You can optionally configure Amazon CloudWatch alarms for your health checks, so that you receive notification when a resource becomes unavailable.
- DNS Failover - Along with DNS Health checks, you can configure DNS failover so that Route 53 will route your traffic from an unhealthy resource to a healthy resource. You can choose the **active-active** failover configuration when you want all of your resources to be available the majority of the time or **active-passive** when you want a primary resource or group of resources to be available the majority of the time and you want a secondary resource or group of resources to be on standby in case all the primary resources become unavailable.
- Traffic Flow - Makes it easy for you to manage traffic globally through a variety of routing policies, including Latency Based Routing, Geo DNS, Geoproximity, and Weighted Round Robin —all of which can be combined with DNS Failover in order to enable a variety of low-latency, fault-tolerant architectures. For more information on routing policies see AWS Routing Policies
- Using Amazon Route 53 Traffic Flow's simple visual editor, you can easily manage how your end-users are routed to your application's endpoints—whether in a single AWS region or distributed around the globe.
- Application Recover Controller - Gives you insights into whether your applications and resources are ready for recovery, and helps you manage and coordinate failover using readiness check and routing control features

Route 53 Control Plane vs. Data Plane

Understanding the different service capabilities of Route 53 and how those capabilities leverage the control plane vs the data plane can be crucial to understanding overall resiliency of Route 53 and how the it will respond in failure and recovery situations. For example, you **can** rely on the Amazon Route 53 data plane to reliably route DNS queries based on health checks. Updating Route 53 routing policies, DNS weights, and records, however, uses the control plane, and **cannot** be relied

upon for recovery. When implementing recovery or mitigation responses to potentially resiliency-impacting events, using control plane operations can lower the overall resiliency of your architecture.

- **Control Plane** The control plane is used to configure services. For Route 53, the control plane has a lower availability design goal than the data plane. The control plane should not be relied upon for service recovery and application resiliency for highly available services in tiers 1 and 2.
- **Data Plane** The data plane is used to deliver services. The data plane maintains a much higher uptime commitment. The data plane should be the method by which services instrument high availability. Route 53 health checks are an example of a data plane capability that can aid in application services maintaining high availability.

Health Checks

Health checks are a way of asking a service whether or not it is capable of performing work successfully. Examples of health checks include, but are not limited to:

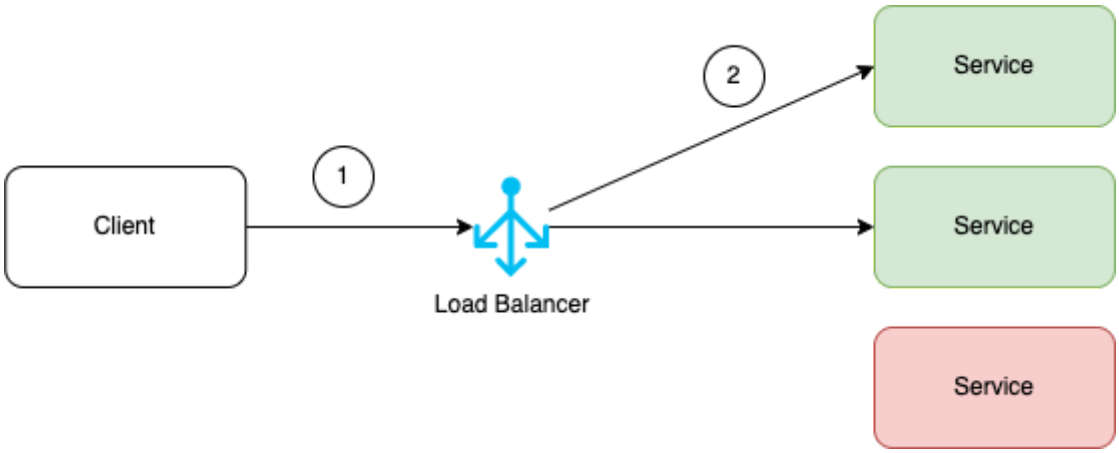
1. Health checks integrated into Route 53 DNS services, which validate underlying service health
2. Load balancers polling services periodically to determine which servers it is safe to direct traffic to.
3. A service that checks its own health before pulling messages off a queue.
4. Monitoring agents that determine the health of a service and raise events or remove a service from service discovery in the event of service degradation.

Health checks are a critical part of the design of resilient solutions, as they anticipate and isolate failures, thereby preserving service availability. Health checks need to be designed to mitigate component level failures, but must also take into account the design of the entire solution.

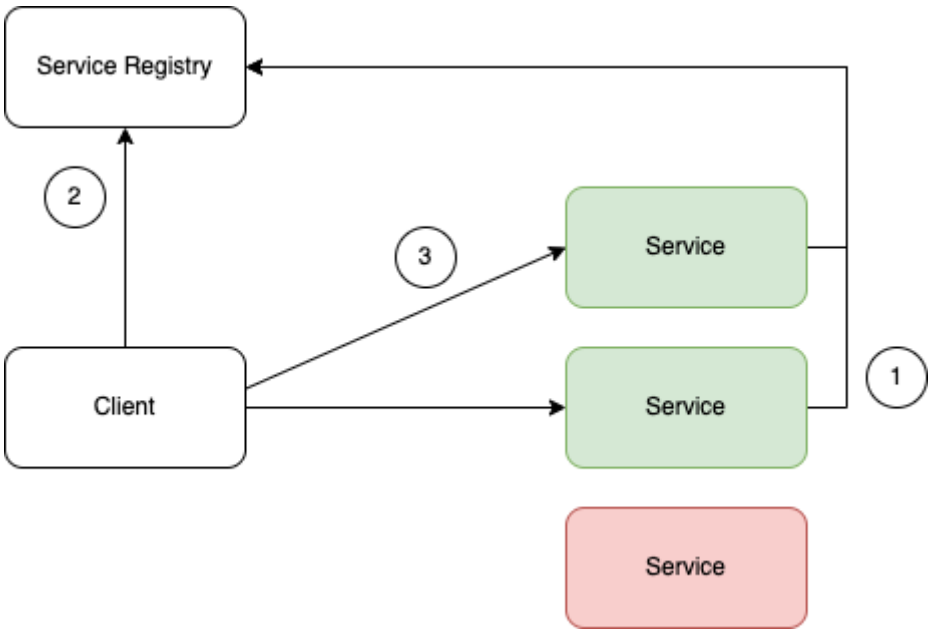
Service Discovery

Service Discovery is the process of dynamically detecting services available on a network. Resiliency of an application can be enhanced through the use of service discovery. Two models of service discovery exist, server-side and client-side service discovery.

In a server-side service discovery model, the server updates its availability with a load-balancer, DNS, or similar system to route traffic to available service endpoints.



In a client-side service discovery model, the client queries a service registry to determine available service endpoints and connects to the appropriate one.



A full discussion of service discovery is beyond the scope of this design document, and will require a separate design.

Application Load Balancing

Application Load Balancers allow for the distribution of incoming requests across service endpoints, while allowing for intelligent health checks, and load distribution algorithms. A limitation of AWS Application Load Balancers is that it they are a regional service, and can only control automated failover across services running in more than one Availability Zone.

Global Load Balancing

Global load balancing refers to the ability to distribute service requests across a global distribution of service endpoints. Global load balancing in AWS is managed by Route 53 and includes a variety of

routing policies, including Latency Based Routing, Geo DNS, Geoproximity, and Weighted Round Robin—all of which can be combined with DNS Failover in order to enable a variety of low-latency, fault-tolerant architectures.

Potential Recovery Limitations

In the event of a component or application recovery, certain conditions can occur that can limit the ability of the system to self-heal or limit manual recovery efforts. It is important to understand these limitations and design the application and component recovery accordingly.

API Call Throttling

Especially in regional recovery scenarios, API throttling or contention, can limit the ability for even automated recovery functions to successfully recover services to a functioning state. This can be further exacerbated when attempting to use the Control Plane for recovery, where overall availability may not match SWA recovery requirements.

Availability of EC2 instance types

In rare circumstances, although increased potential exists during a regional outage, contention for specific EC2 instance types can result in the inability to rapidly provision new instances. Like API calls, this can be exacerbated by Control Plane issues that slow down or prevent instance provisioning events. For this reason, scaling and instance provisioning should be tailored to the availability expectations of the application. Please refer to the Component Scaling section of this guidance.

Service Quotas

Service quotas can impact application availability and should be managed using a process that balances cost implications of escalating services quotas yet mitigates potential impacts of restricting application scaling or performance. In particular, service quotas can be particularly impactful in failover events where there is increased potential for exceeding currently defined service limits or service quotas.

AWS Service Quotas best practices for reliability, from the AWS Well-Architected Framework can be found here, and include:

- Aware of service quotas and constraints: You are aware of your default quotas and quota increase requests for your workload architecture. You additionally know which resource

constraints, such as disk or network, are potentially impactful.
- Manage service quotas across accounts and regions: If you are using multiple AWS accounts or AWS Regions, ensure that you request the appropriate quotas in all environments in which your production workloads run.
- Accommodate fixed service quotas and constraints through architecture: Be aware of unchangeable service quotas and physical resources, and architect to prevent these from impacting reliability.
- Monitor and manage quotas: Evaluate your potential usage and increase your quotas appropriately allowing for planned growth in usage.
- Automate quota management: Implement tools to alert you when thresholds are being approached. By using AWS Service Quotas APIs, you can automate quota increase requests.
- Ensure that a sufficient gap exists between the current quotas and the maximum usage to accommodate failover: When a resource fails, it may still be counted against quotas until it is successfully terminated. Ensure that your quotas cover the overlap of all failed resources with replacements before the failed resources are terminated. You should consider an Availability Zone failure when calculating this gap.

State Management

A thorough understanding of the application's dependence on state is necessary to thoroughly understand the ability to successfully recover the application and meet the Recovery Point Objectives of the application. Recovery from failures, including the restoration of state is something that must be tested.

Service Recovery Testing

The process of full service recovery must be documented and tested regularly. Additionally, depending on business criticality, Teams should have the appropriate level of automated recovery in place. The table under Resiliency Requirements below describes service recovery and testing requirements. These tests should be run as part of the certification and deployment process, but also run under load, where real-world transactions will stress and expose unexpected situations.

Resiliency Requirements & Patterns

Component Resiliency

Components running only in a single Availability Zone can often times be designed to have a regional or even global resiliency through different configuration patterns. The table below describes patterns

of component resiliency that are used throughout this guidance document.

Option	Description
Global Component	- Regional resiliency managed by AWS. - Expected to be globally available, and should be used preferentially. - Care must be taken to understand what aspects of a global component, like Route 53, is actually globally available.
Active / Active Component	- Component service requests are sent to all regions and can be processed independent of other regions. - Care must be taken to configure these services correctly and to understand any required state management. (Either does not depend on state, can use an eventual consistency model, or uses a synchronous replication of state.) -Care must also be taken to understand event sequencing requirements, which can cause transactional or data integrity issues within the larger solution context.
Active / Passive Component with Automated Recovery	- Component service requests are sent to only one region at a time - Deep health checks are configured to assess regional component health, and recovery automated in the AWS data plane - Component recovery and failback from component recovery must be thoroughly tested to eliminate any issues with data integrity
Active / Passive Component with Manual Recovery	- Component service requests are sent to only one region - Health checks are configured to assess regional component health - Recovery mechanisms are in place that can be executed without use of the AWS control plane
Active / Standby Component	- Component service requests are sent to only one region - Health checks are in place to assess regional component health - Processes are in place and have been tested to active standby region components
Single Region	- Component service requests are sent to only one region - Regional outages will result in loss of service until the region is recovered

Resiliency Patterns

The following resiliency patterns describe the high level characteristics as well as supported Application recovery Hierarchy tiers. Further details can be found within the following documentation

of each resiliency pattern, including examples.

Resiliency Pattern	Characteristics	Supported ARH Tiers
Globally Resilient	- Application components run globally and are resilient to issues impacting more than one region - Service recovery is automatic and self-healing	Tier 0, Tier 1 (provided ltency requirements of the application can be met.)
Regionally Resilient	- Application components run active / active across 2 regions and are resilient to issues impacting a single region - Data is synchronous across regions - Health checks proactively monitor for failed components and isolate failures - Service recovery is automatic and self-healing	Tier 0, Tier 1
Regionally Resilient with Failover	- Application components run active / passive across 2 regions - Data replication may be asynchronous - Health checks identify failed components, but recovery may be manual in some or all cases	Tier 2
Single Region with Standby	- Application components run active / standby across 2 regions - Data replication may be asynchronous - Health checks identify failures - Service recovery requires starting the application in a standby region	Tier 3+
Single Region	- Application components run in a single region - Failures are detected by health checks - Services are recovered once the region has been restored to full operations	Tier 4

Globally Resilient

Globally resilient architectures extend the Regionally Resilient pattern to include global dispersion of workloads. This means that globally resilient applications are deployed and active in at least 3 regions, across 2 continents, and can withstand failures in more than one region.

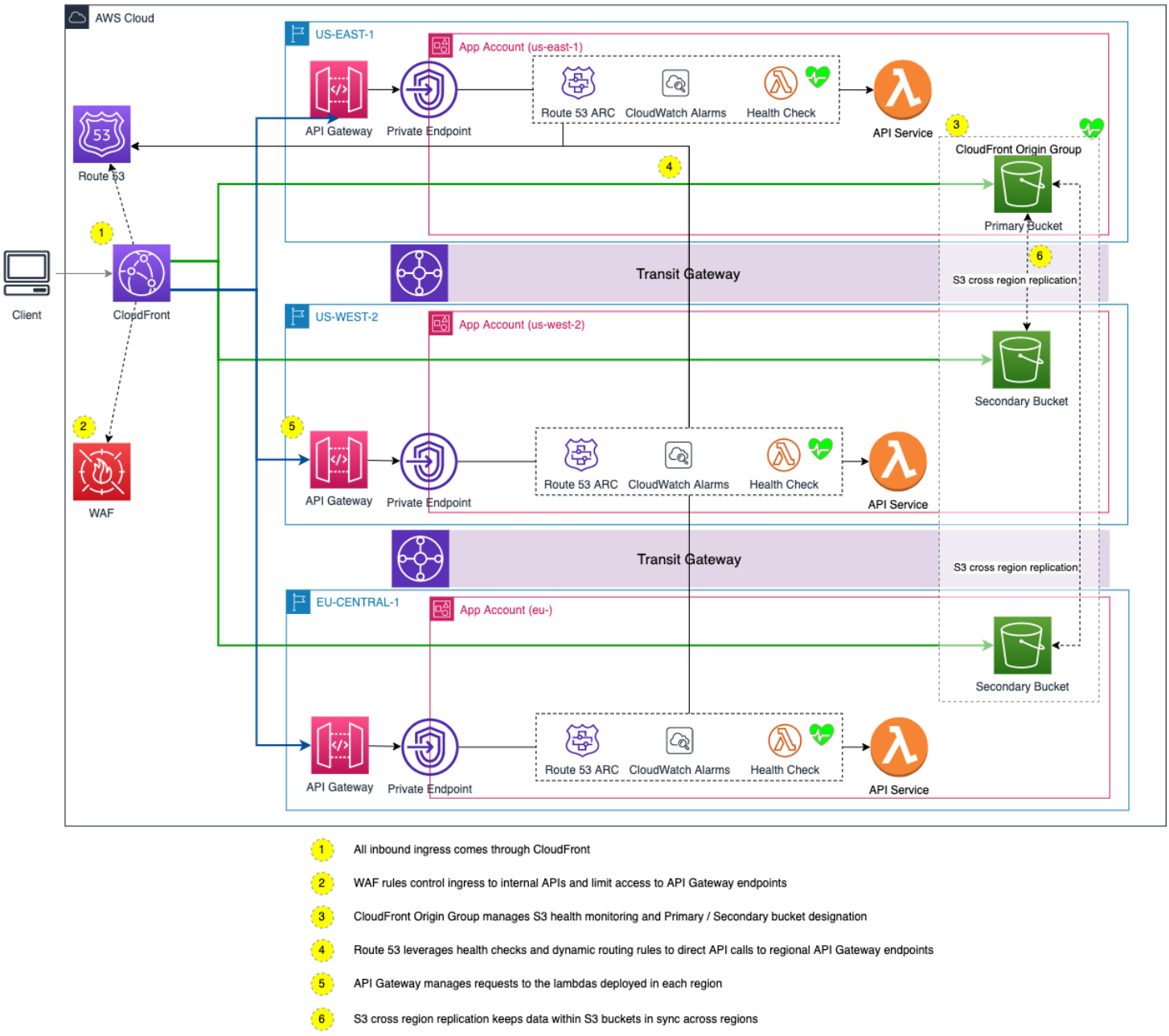
Component resiliency pattern	**Preference**
Global component	Allowed
Active / Active component	Allowed, with design review
Active / Passive with automated health check and recovery	Not Allowed
Active / Passive with manual recovery	Not Allowed
Active / Standby	Not Allowed
Single Region	Not Allowed

Key Requirements for Globally Resilient workloads

The following table shows key requirements for Globally Resilient workloads.

Criteria	**Requirement**
Regional Availability	All services are running in **at least 3 regions**
Availablility Zones	Where applicable, all services are running in **at least 3 AZs per region**
Compute	**Require serverless** compute, due to stateless nature and auto-scaling
Scaling	Application scaled to support **100%** of the workload in a single region
Health Checks	Health Checks have been implemented to detect failures and *must* leverage automated failover to redirect traffic away from the failed component.
Failover Control	Failover must leverage the data plane of the AWS service. **Failover cannot require the use of AWS Console or CICD Tools.**
Monitoring	All components have health monitoring instrumented in an approved oberservability platform

Example Globally Resilient Single Page App (SPA)



Integrations between components is a critical aspect of achieving true regional resilience.The diagram above depicts a basic pattern for Single Page Apps, with static content being served from S3, and dynamic content being served from one or more lambdas. CloudFront and Route 53 manage traffic routing avoiding any simgle point of failure. Traffic routing at each step is dynamic and can be routed to least latent and available services.

- The pattern above leverages CloudFront, Route 53, S3 replication and API Gateway to create a highly resilient solution.
- All services being used have health checks in place to dynamically route traffic to all regions.
- Services like lambda and S3 are managed by AWS and do require explicit scaling configurations.

Regionally Resilient

Regionally resilient architectures attempt to eliminate any single failure points within an application across at least 2 regions. Designs need to examine each component to ensure that regional resilience is accounted for. The following component resiliency patterns can be leveraged to design for regionally resilient solutions.

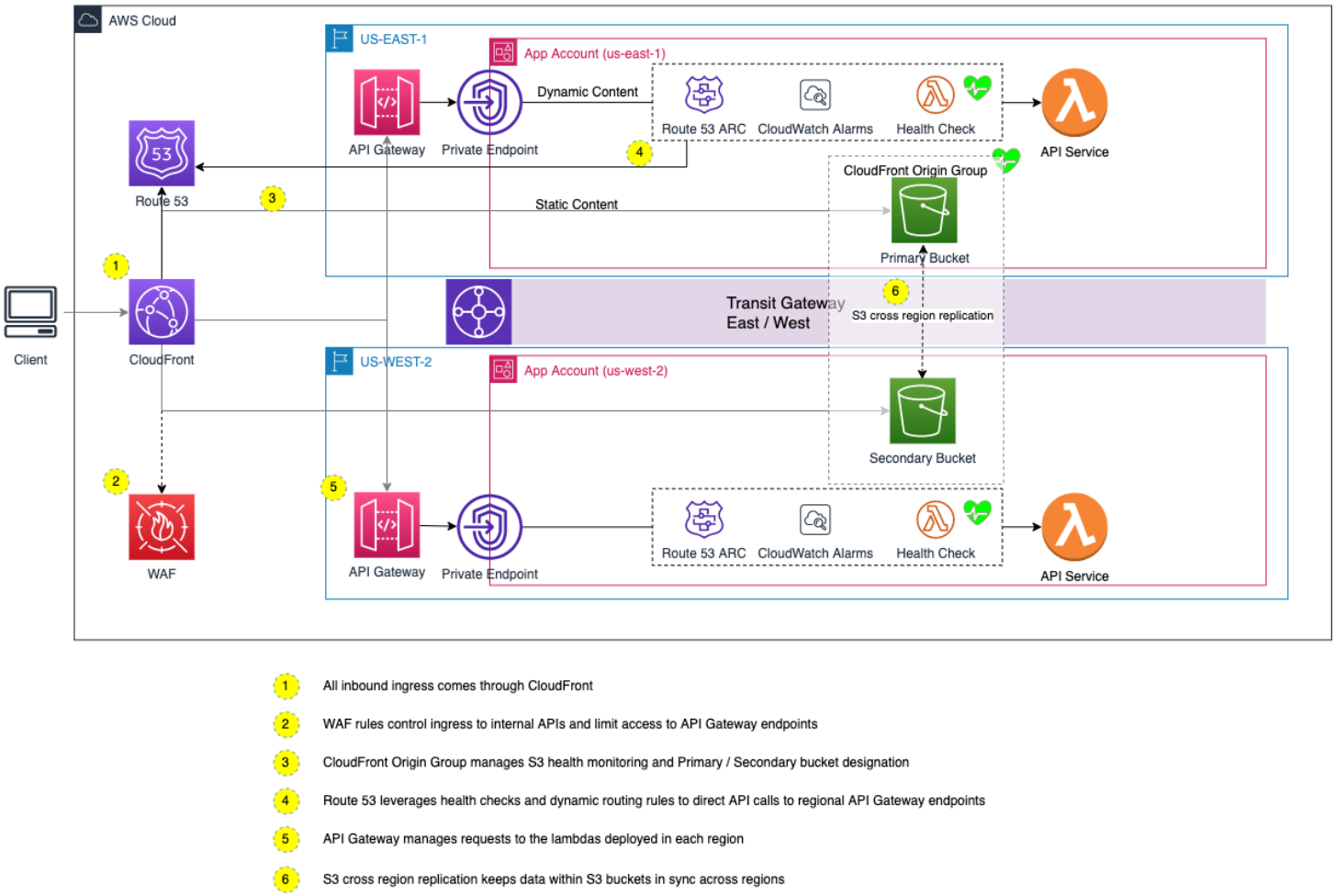
Component resiliency pattern	**Preference**
Global component	Preferred
Active / Active component	Preferred
Active / Passive with automated health check and recovery	Allowed
Active / Passive with manual recovery	Not Allowed
Active / Standby	Not Allowed
Single Region	Not Allowed

Key Requirements for Regionally Resilient workloads

The following table shows key requirements for Regionally Resilient workloads.

Criteria	**Requirement**
Regional Availability	All services are running in **at least 2 regions**
Availablility Zones	Where applicable, all services are running in **at least 3 AZs per region**
Compute	*Prefer serverless* compute, due to stateless nature and auto-scaling
Scaling	Application scaled to support **100%** of the workload in a single region
Health Checks	Health Checks have been implemented to detect failures and **automatically redirect traffic away from the failed component**.
Failover Control	Failover must leverage the data plane of the AWS service. **Failover cannot require the use of AWS Console or CICD Tools.**
Monitoring	All components have health monitoring instrumented in an approved oberservability platform

Example Single Page App (SPA)

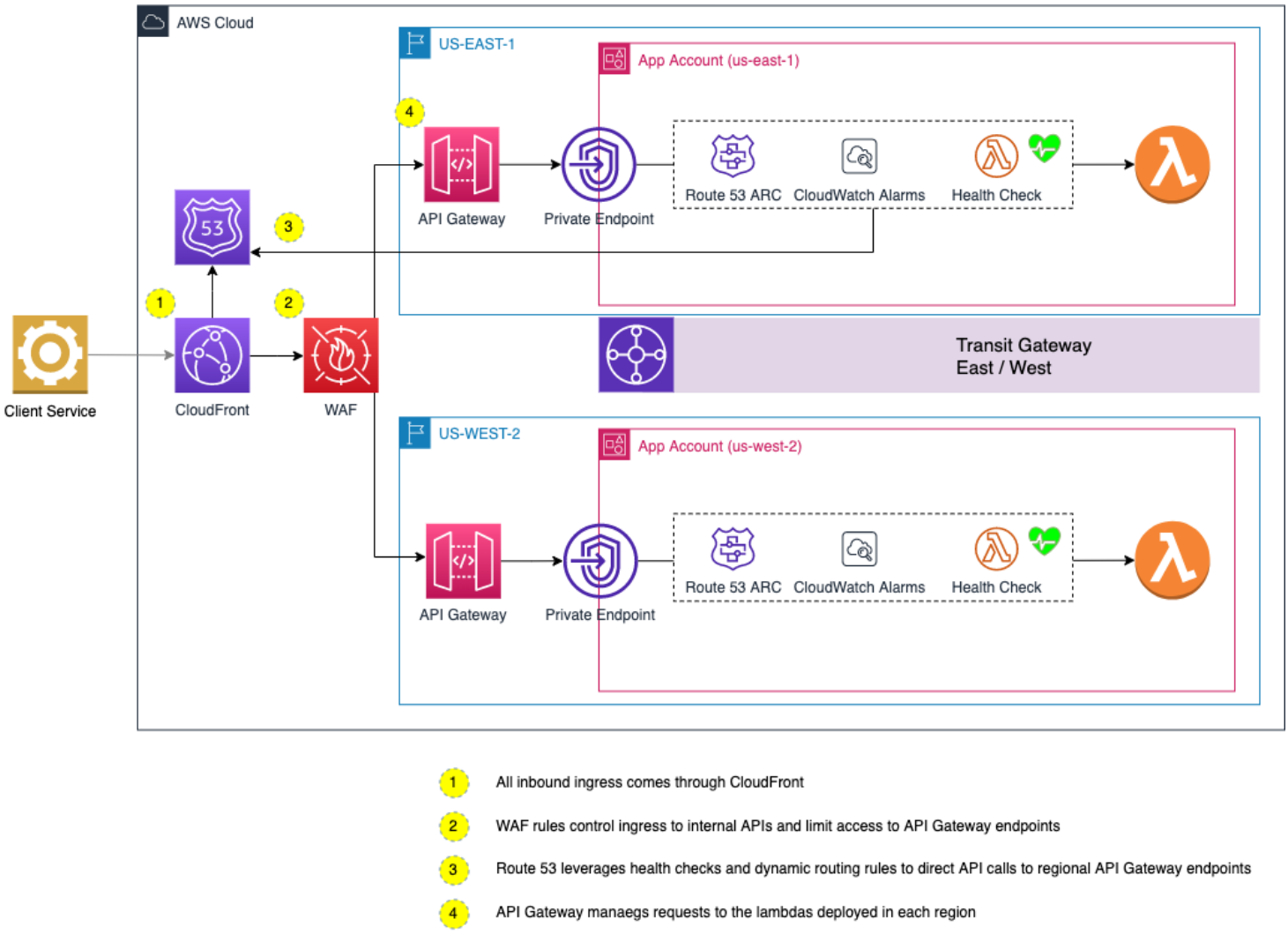


Integrations between components is a critical aspect of achieving true regional resilience.The diagram above depicts a basic pattern for Single Page Apps, with static content being served from S3, and dynamic content being served from one or more lambdas.

Key guidance for Regionally Resilient Single Page Apps:

- CloudFront and Route 53 manage traffic routing avoiding any simgle point of failure. Traffic routing at each step is dynamic and can be routed to least latent and available services.
- The pattern above leverages CloudFront, Route 53, S3 replication and API Gateway to create a highly resilient solution.
- All services being used have health checks in place to dynamically route traffic to both regions.
- Services like lambda and S3 are managed by AWS and do require explicit scaling configurations.

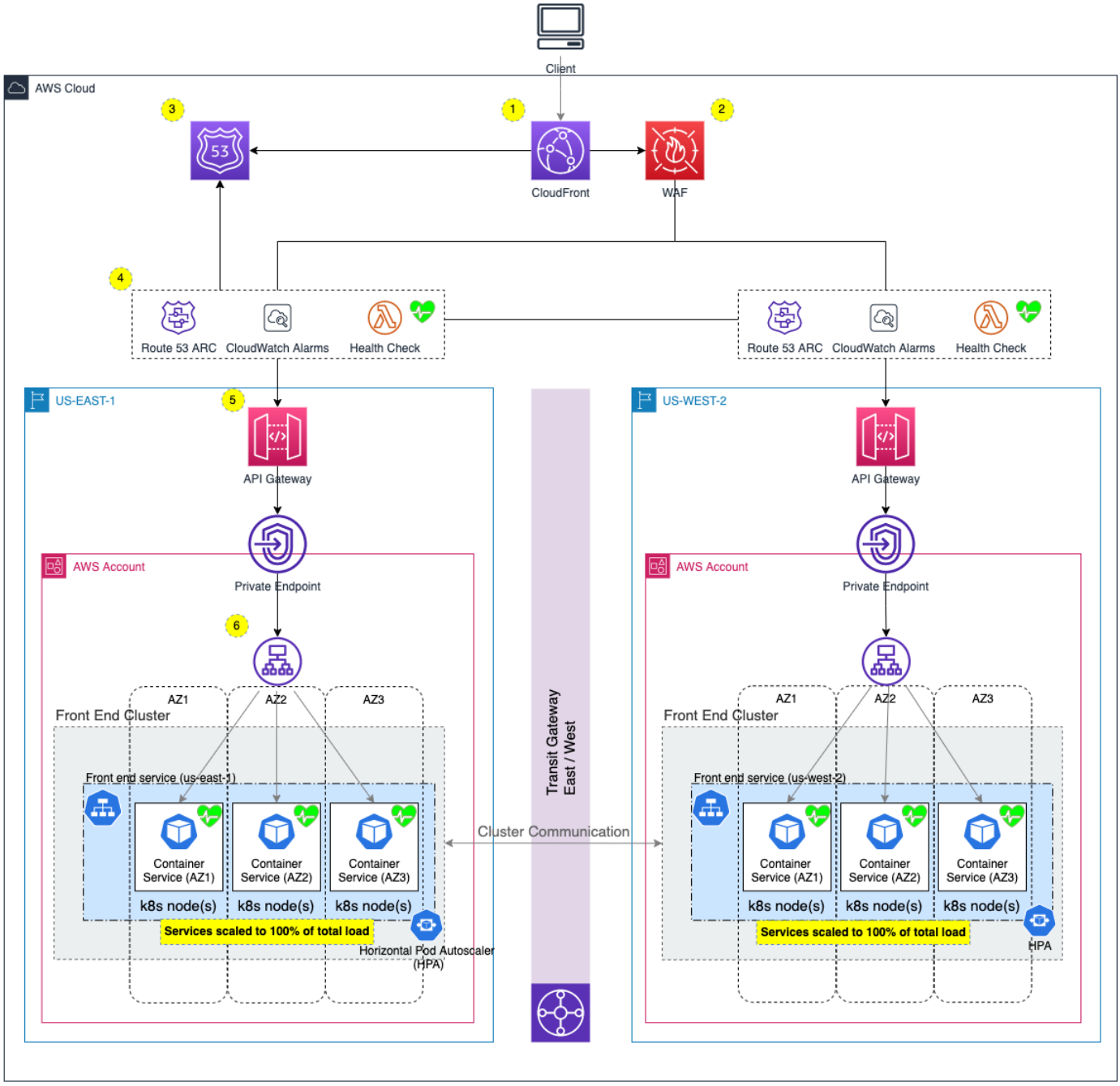
Example Service to Service



Key guidance for Regionally Resilient Service APIs:

- CloudFront and Route 53 manage traffic routing avoiding any simgle point of failure. Traffic routing at each step is dynamic and can be routed to least latent and available services.
- The pattern above leverages CloudFront, Route 53, and API Gateway to create a highly resilient solution.
- All services being used have health checks in place to dynamically route traffic to both regions.
- Services like lambda are managed by AWS and do require explicit scaling configurations.

Example Regionally Resilient Kubernetes Application



1. All inbound ingress comes through CloudFront
2. WAF rules control ingress to internal APIs and limit access to API Gateway endpoints
3. Route 53 leverages health checks and dynamic routing rules to direct API calls to regional API Gateway endpoints
4. Route 53 available endpoints are updated by health checks
5. API Gateway access must come through CloudFront and manages requests to the kubernetes clusters deployed in each region
6. Application Load Balancer distributes load to healthy services running in the kubernetes cluster

Key points for Regionally Resilient EKS container based applications:

- EKS clusters exist within a single region and cannot span regions. Each region will have a separate EKS cluster.
- EKS cluster nodes should be distributed evenly across at least 3 AZs

- Application services, running as containers within EKS, should be evenly distributed across EKS cluster nodes and at least 3 AZs
- Route 53 health checks determine health of each region and distribute traffic accordingly
- Health monitoring within EKS isolates and restarts unhealth nodes and container workloads, maintaining a minimum of 3 healthy AZs at all times
- For Regionally Resilient workloads, capacity of each regional EKS cluster should be scaled to support 100% of the application workload at all times. Scaling of workloads should not be part of the recovery design for the Regionally Resilient pattern, as resource contention and control plane issues can hinder scaling events during regional recovery activities.

Regionally Resilient with Failover

Description

Component resiliency pattern	**Preference**
Global component	Preferred
Active / Active component	Preferred
Active / Passive with automated health check and recovery	Preferred
Active / Passive with manual recovery	Allowed
Active / Standby	Not Allowed
Single Region	Not Allowed

Key Requirements for Regionally Resilient (Manual Failover) workloads

The following table shows key requirements for Regionally Resilient workloads.

Criteria	**Requirement**
Regional Availability	All services are running in **at least 2 regions**
Availablility Zones	Where applicable, all services are running in **at least 3 AZs per region**
Compute	*Prefer serverless* compute, due to stateless nature and auto-scaling
Scaling	Application scaled to support **100%** of the workload in a single region

Criteria	Requirement
Health Checks	Health Checks have been implemented to detect failures and *may* leverage automated failover to redirect traffic away from the failed component.
Failover Control	Failover must leverage the data plane of the AWS service. **Failover cannot require the use of AWS Console or CICD Tools.**
Monitoring	All components have health monitoring instrumented in an approved oberservability platform

Single Region with Standby

Description

Component resiliency pattern	Preference
Global component	Preferred
Active / Active component	Allowed
Active / Passive with automated health check and recovery	Allowed
Active / Passive with manual recovery	Preferred
Active / Standby	Allowed
Single Region	Not Allowed

Key Requirements for Single Region with Standby workloads

The following table shows key requirements for Regionally Resilient workloads.

Criteria	Requirement
Regional Availability	All services are running in **at least 2 regions**
Availablility Zones	Where applicable, all services are running in **at least 3 AZs per region**
Compute	*Prefer serverless* compute, due to stateless nature and auto-scaling
Scaling	Application scaled to support **100%** of the workload in a single region

Criteria	Requirement
Health Checks	Health Checks have been implemented to detect failures and **automatically redirect traffic away from the failed component**.
Failover Control	Failover must leverage the data plane of the AWS service. **Failover cannot require the use of AWS Console or CICD Tools.**
Monitoring	All components have health monitoring instrumented in an approved oberservability platform

Single Region

Description

Component resiliency pattern	Preference
Global component	Allowed
Active / Active component	Not Allowed
Active / Passive with automated health check and recovery	Not Allowed
Active / Passive with manual recovery	Not Allowed
Active / Standby	Not Allowed
Single Region	Preferred

Key Requirements for Single Region workloads

The following table shows key requirements for Regionally Resilient workloads.

Criteria	Requirement
Regional Availability	All services are running in **at least 2 regions**
Availablility Zones	Where applicable, all services are running in **at least 3 AZs per region**
Compute	*Prefer serverless* compute, due to stateless nature and auto-scaling
Scaling	Application scaled to support **100%** of the workload in a single region

Criteria	Requirement
Health Checks	Health Checks have been implemented to detect failures and **automatically redirect traffic away from the failed component**.
Failover Control	Failover must leverage the data plane of the AWS service. **Failover cannot require the use of AWS Console or CICD Tools.**
Monitoring	All components have health monitoring instrumented in an approved oberservability platform

Sound Recovery Practices

Best practices for service recovery

1. **Service preference:** When designing applications, prefer AWS Global and AWS Regional services to minimize recovery effort and control scaling costs.
2. **Prefer serverless:** This applies to compute workloads as well. Service design should prefer serverless designs over alternatives, as serverless designs require externalizing state, eliminate scaling issues, and
3. **Start with components:** Start by understanding how each component will behave in a failure scenario, and understand and document recovery processes.
4. **Understand data resiliency:** Data resiliency is a key part of understanding the ability to effectively recover any service. Key data dependencies, like the data consistency model, must be thoroughly understood. Data replication, backup, and recovery processes play a critical role in overall system recovery. Separate Data Resiliency guidance will address the specific aspects of data resiliency.
5. **Prefer automation:** Service recovery automaton is a key non-functional requirement for services and should be part of the architecture design process.
   a. For known use cases, automate intelligent and deep health checks that uncover service issues and isolate failures.
   b. To raise overall service availability, extend automation to self-healing activities that remove human interaction. Required for some resiliency patterns.
6. **Test often:** Test failure scenarios frequently. Build automated integration tests that validate heath checks and recovery solutions. Perform thought experiments to uncover new scenarios that may lead to failures.
7. **Assume the worst:** Assume everything will fail eventually and be sure that those failure scenarios have been tested and behave as expected.

Recommended corresponding Architecture Practices

Design Reviews

Design Approval

Architecture Exceptions