Global Active/Active Microservices Platform

Multi-Region Architecture for Continuous Availability and Global Resilience

Abstract

This paper presents a comprehensive architecture for a **multi-region active/active microservices platform** deployed on Amazon Web Services (AWS). The design enables **global low-latency access**, **zero-downtime resilience**, and **autonomous regional execution**, ensuring uninterrupted service delivery even in the event of a full regional outage.

The architecture integrates **Route 53 latency-based routing**, **EKS/ECS microservices clusters**, **DynamoDB Global Tables**, **EventBridge global event propagation**, and **S3 cross-region replication** to form a unified global data and event fabric. Each region operates as a fully self-contained execution plane with independent compute, storage, and observability layers, while remaining synchronized through sub-second replication and event-driven communication.

A dedicated section details the **regional microservices architecture**, including ingress, service mesh, asynchronous integration, data persistence, and security governance. The paper also outlines the **end-to-end data flow** from user request to global replication, and describes a **multi-region CI/CD pipeline** that ensures consistent, automated deployments across all regions. Collectively, this architecture provides a scalable, fault-tolerant foundation for enterprise-grade workloads requiring global reach, continuous availability, and operational excellence.

Table of Contents

Multi-Region Active/Active Microservices Platform Architecture Overview

Global Architecture Overview

- **Two or more AWS regions** (e.g., eu-central-1 + us-east-1) running **identical stacks**
- **EKS or ECS** clusters per region
- **DynamoDB Global Tables** for multi-master, low-latency, conflict-free replication
- **Route53 latency-based routing** for global traffic distribution
- **API Gateway / ALB** fronting microservices
- **Service Mesh** (AWS App Mesh or Istio) for cross-service observability and retries
- **S3 Cross-Region Replication** for static assets
- **CloudFront** optional for global CDN acceleration
- **EventBridge global bus** for asynchronous cross-region events
- **Active/Active**: both regions serve traffic simultaneously

Core Components

EKS/ECS Clusters

- Each region runs its own cluster
- Microservices deployed identically via GitOps or CI/CD
- Auto Scaling based on CPU, memory, or custom metrics

DynamoDB Global Tables

- Multi-master writes
- Replication latency typically < 1 second
- Eliminates need for cross-region database failover

Route53 Latency Routing

- Users routed to nearest healthy region
- Health checks automatically remove unhealthy endpoints

Service Mesh

- mTLS
- Traffic shaping
- Retries + circuit breaking

EventBridge Global Bus

- Cross-region event propagation
- Decoupled microservices

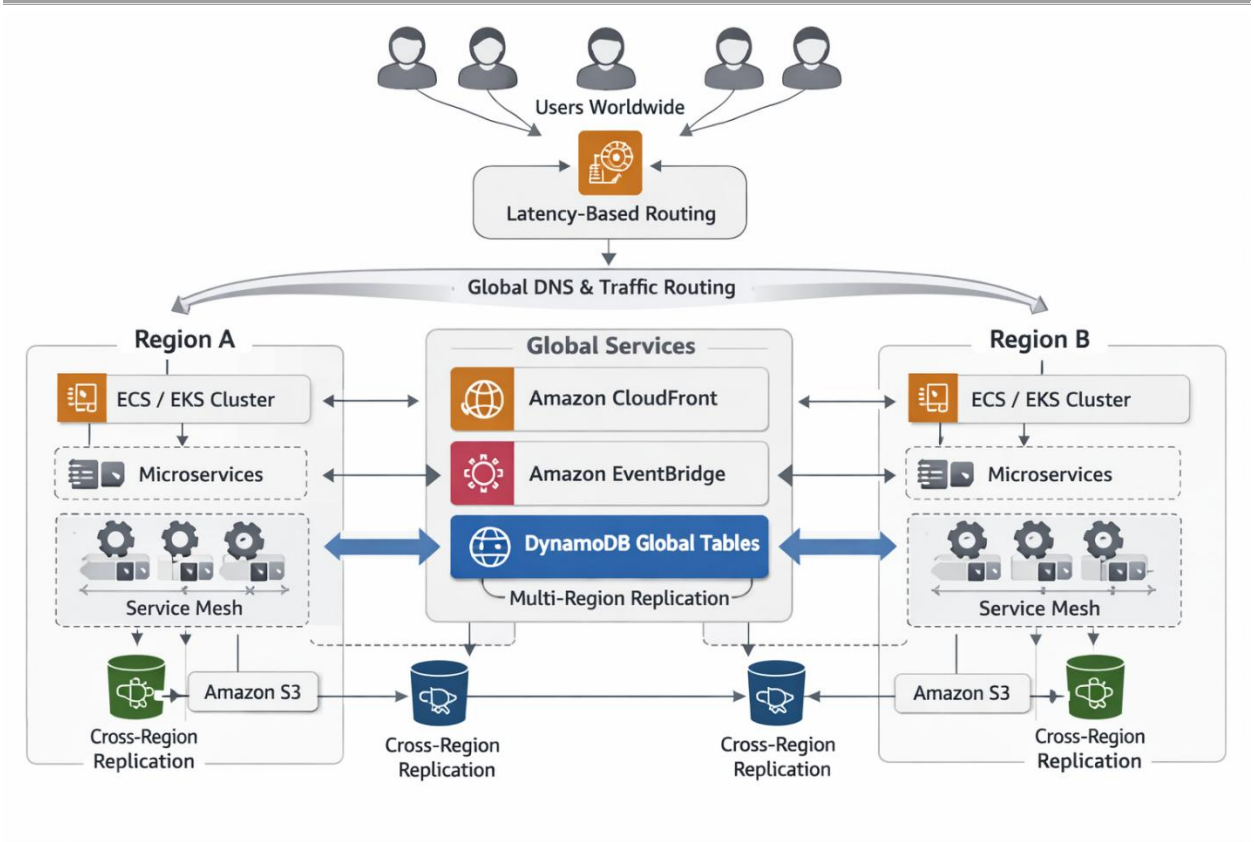
High Availability & Failover

Normal Operation

- Both regions active
- Writes accepted in both regions
- DynamoDB resolves conflicts automatically
- Route53 sends users to nearest region

Regional Failure

- Route53 health checks remove failed region
- All traffic flows to surviving region
- DynamoDB continues serving from remaining region
- When region recovers, replication catches up



1. Architectural Intent

This design establishes a **multi-region active/active cloud platform** on AWS, ensuring continuous availability, low latency, and global resilience. Each region operates as a fully autonomous deployment zone, yet remains synchronized through globally replicated data and event layers. The architecture supports **real-time failover** without manual intervention, enabling uninterrupted service delivery across continents.

2. Global Control Plane

At the top of the topology, **Amazon Route 53 latency-based routing** serves as the global traffic controller.

- It evaluates user request latency and directs each session to the nearest healthy region.
- Integrated health checks automatically remove degraded endpoints, maintaining seamless continuity.

- This routing layer forms the **global DNS and traffic orchestration plane**, abstracting regional complexity from end users.

3. Regional Execution Planes

Each AWS region hosts a **dedicated microservices cluster** deployed on **EKS or ECS**, encapsulating the full application stack.

- **Microservices** run within a **service mesh** (App Mesh or Istio) that provides mTLS, observability, and traffic management.
- **Auto Scaling Groups** dynamically adjust compute capacity based on workload metrics.
- **Amazon S3** stores static assets and configuration artifacts, synchronized through **cross-region replication** to maintain consistency.

4. Global Data and Event Fabric

At the heart of the system lies **DynamoDB Global Tables**, providing multi-master replication across all regions.

- Each region can perform read/write operations locally, with sub-second replication latency.
- Conflict resolution is handled natively by DynamoDB, ensuring eventual consistency without custom logic.
- **Amazon EventBridge** extends this layer by propagating domain events globally, enabling asynchronous communication between microservices in different regions.
- Together, these components form a **global data and event fabric**, ensuring both transactional integrity and event-driven responsiveness.

5. User Experience Layer

Amazon CloudFront accelerates content delivery through edge caching, minimizing latency for static and dynamic assets.

- Requests flow from CloudFront to Route 53, then to the nearest regional cluster.
- The combination of CloudFront and Route 53 provides a **dual-layer optimization** for both network and application performance.

6. Resilience and Recovery

The platform operates in **true active/active mode**:

- Both regions serve traffic concurrently.
- In the event of a regional outage, Route 53 automatically reroutes all traffic to the surviving region.
- DynamoDB continues serving from the remaining replica, and EventBridge ensures event continuity.
- When the failed region recovers, replication restores full synchronization automatically.

7. Operational Governance

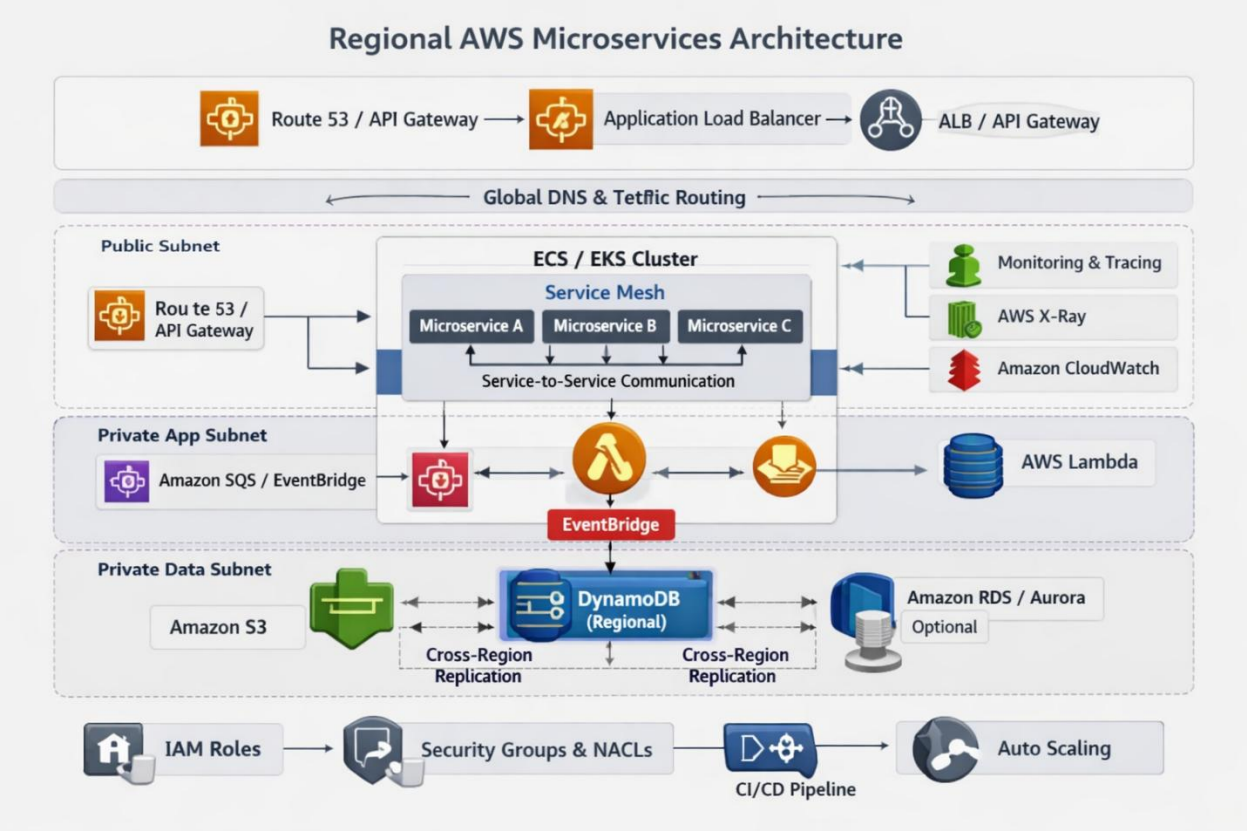
- **CI/CD pipelines** deploy identical configurations to each region using GitOps or AWS CodePipeline.
- **Monitoring and observability** are unified through CloudWatch and distributed tracing within the service mesh.
- **Security posture** is consistent across regions, leveraging IAM federation and centralized policy management.

8. Strategic Outcomes

This architecture delivers:

- **Global low-latency access** for users anywhere in the world.
- **Zero-downtime resilience** through active/active redundancy.
- **Simplified disaster recovery** via automated failover and replication.
- **Scalable foundation** for future multi-cloud or hybrid extensions.

Regional AWS Microservices Architecture



Regional architecture purpose

This regional architecture represents **one execution plane** within your global active/active platform. Each region is designed to be **fully self-sufficient**: it can independently receive traffic, execute microservices, persist data, and participate in global replication. The diagram is segmented into **Public Subnet**, **Private App Subnet**, and **Private Data Subnet**, reflecting a clear separation of concerns for security, scalability, and governance.

Public subnet: ingress and exposure

Route 53 / API Gateway → Application Load Balancer

- **Route 53 / API Gateway** at the top-left represents the regional entry point for HTTP(S) traffic once global DNS has routed the user to this region.
- Requests are forwarded to the **Application Load Balancer (ALB) / API Gateway**, which:
    - Terminates TLS.

    - Performs path-based or host-based routing.
    - Distributes traffic across microservices running in the ECS/EKS cluster.
- This layer is intentionally placed in the **Public Subnet**, exposing only the minimal surface area required to accept external traffic, while keeping application and data components private.

Private app subnet: microservices and service mesh

ECS / EKS Cluster with Service Mesh

- The **ECS / EKS Cluster** is the core compute fabric of the region.
- Within it, **Microservice A, Microservice B, and Microservice C** are deployed as containers (tasks or pods).
- These services are enclosed in a **Service Mesh** boundary, which provides:
    - **Service-to-service communication** via sidecar proxies.
    - **mTLS** for encrypted, authenticated traffic between services.
    - **Traffic policies** such as retries, timeouts, and circuit breaking.
    - **Observability hooks** for metrics and traces.

Monitoring & Tracing

- To the right of the microservices, **AWS X-Ray**, **Amazon CloudWatch**, and **Prometheus** represent the **observability stack**:
    - **X-Ray** traces requests across microservices and downstream dependencies.
    - **CloudWatch** aggregates logs, metrics, and alarms.
    - **Prometheus** (often paired with Grafana) collects fine-grained metrics from the service mesh and workloads.
- Together, they provide a unified view of **performance, reliability, and behavior** across the regional microservices.

Private app subnet: asynchronous integration and event processing

Amazon SQS / EventBridge and AWS Lambda

- **Amazon SQS / EventBridge** in the Private App Subnet represents the **asynchronous integration layer**:
    - **SQS** handles durable, point-to-point messaging and buffering.
    - **EventBridge** handles event routing, pub/sub patterns, and integration with other AWS services and external SaaS.
- **AWS Lambda** consumes messages and events:
    - Executes **lightweight, event-driven functions** such as data enrichment, notifications, or orchestration steps.
    - Decouples microservices from direct synchronous dependencies, improving resilience and scalability.
- The arrow from **EventBridge** to **Lambda** emphasizes that many domain events are processed out-of-band, reducing coupling and latency sensitivity in the core microservices.

Private data subnet: storage, replication, and security

Amazon S3

- **Amazon S3** in the Private Data Subnet stores:
    - Static assets (e.g., images, documents).
    - Backups, exports, and large data artifacts.
- The **cross-region replication** arrow indicates that S3 buckets are configured to replicate critical objects to peer regions, supporting **disaster recovery and data locality**.

DynamoDB (Regional) with Global Tables

- **DynamoDB (Regional)** is the primary operational data store for microservices in this region.
- The dashed arrow labeled **DynamoDB Global Tables** indicates that this table is part of a **multi-region, multi-master configuration**:
    - Local reads and writes occur in-region for low latency.
    - Changes are replicated to other regions automatically.

    - This enables the global active/active pattern without manual database failover.

Amazon RDS / Aurora (Optional)

- **Amazon RDS / Aurora** is shown as **optional**:
    - Used when relational semantics, complex joins, or transactional guarantees are required.
    - Typically configured with **read replicas** and possibly cross-region read replicas for DR.
- It complements DynamoDB rather than replacing it, supporting **polyglot persistence**.

AWS KMS

- **AWS KMS** provides **encryption key management**:
    - Encrypts data at rest in S3, DynamoDB, and RDS/Aurora.
    - Issues data keys for application-level encryption where needed.
- This ensures consistent **cryptographic controls** across all data services.

Security and governance layer

IAM Roles

- **IAM Roles** define **who can do what**:
    - Microservices assume roles to access DynamoDB, S3, EventBridge, etc.
    - CI/CD pipelines use roles to deploy workloads and manage infrastructure.
- This enforces **least privilege** and auditable access patterns.

Security Groups & NACLs

- **Security Groups** act as **stateful firewalls** around instances, containers, and databases.
- **Network ACLs** provide **stateless subnet-level controls**.
- Together, they:

  - Restrict inbound access to only the ALB/API Gateway from the internet.
  - Allow only necessary east–west traffic between microservices and data stores.
  - Block unauthorized lateral movement within the VPC.

CI/CD and elasticity

CI/CD Pipeline

- The **CI/CD Pipeline** icon represents automated delivery using tools like **CodePipeline, CodeBuild, Argo CD, or GitHub Actions**:
  - Builds container images.
  - Runs tests and security scans.
  - Deploys to ECS/EKS using declarative manifests or blue/green strategies.
- This ensures **consistent, repeatable deployments** across all regions.

Auto Scaling

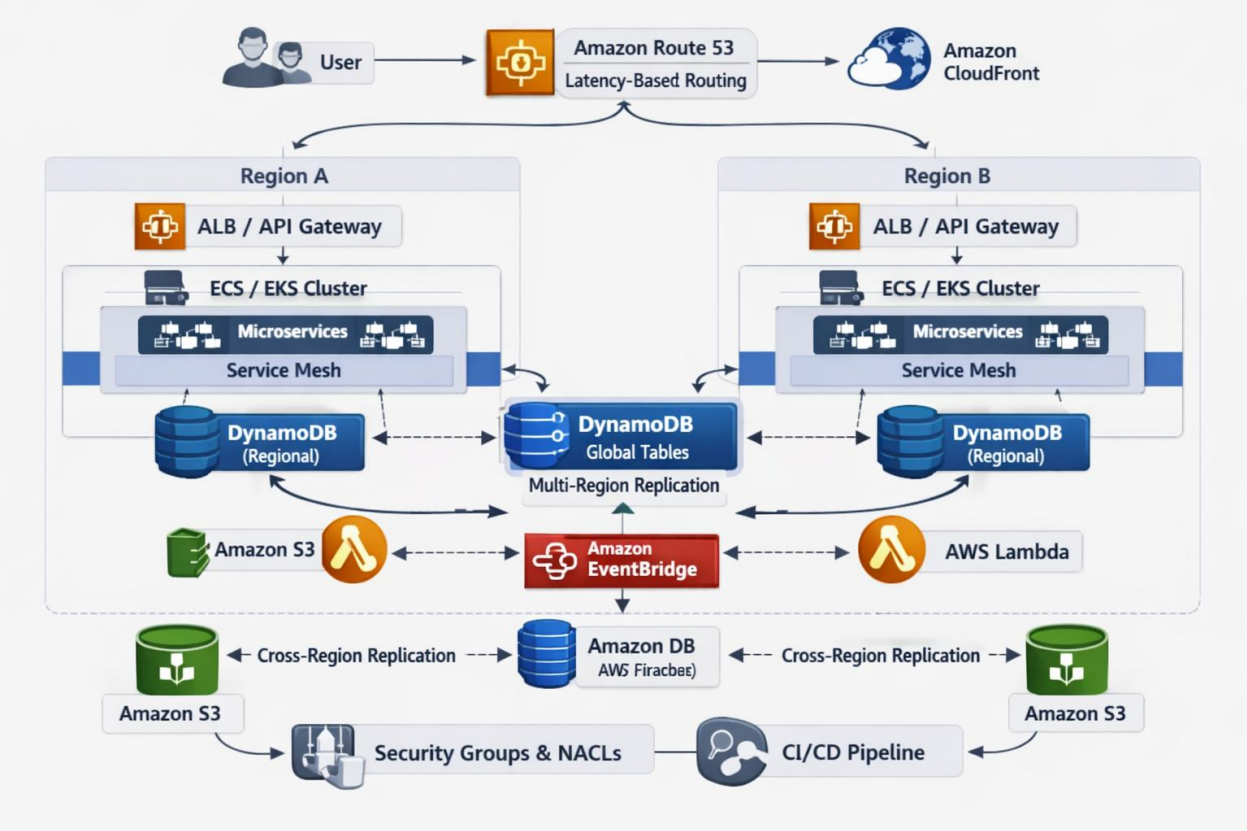
- **Auto Scaling** is connected to the CI/CD and cluster:
  - Scales microservice replicas based on CPU, memory, or custom metrics.
  - Scales underlying nodes (EC2 or Fargate capacity) as demand changes.
- This provides **elastic capacity**, aligning cost with actual usage while maintaining performance.

How this region fits into the global picture

Within the broader **multi-region active/active design**:

- Each region implements this **same regional architecture**.
- **DynamoDB Global Tables** and **S3 cross-region replication** tie data together across regions.
- **Route 53** and global routing decide which region receives each user's traffic.
- If another region fails, this region can **absorb additional load** without architectural changes, because it already runs a complete, autonomous stack.

Data Flow: From User to Route 53 to Region to DynamoDB Global Tables



Data Flow Narrative

Diagram Overview

The visual shows:

- Bidirectional arrows for **data replication** between regions.
- Sequential flow from **User → Route 53 → Regional Ingress → Microservices → Data Layer → Response**.
- EventBridge and DynamoDB forming the **global synchronization backbone**.
- Observability and security layers wrapping the entire flow.

1. User Entry

- **Global DNS (Route 53)** receives the user's request and performs **latency-based routing**.
- The request is directed to the nearest healthy region (e.g., *Region A* or *Region B*).

2. Regional Ingress

- Within the selected region, traffic flows through **API Gateway / ALB**, which terminates TLS and routes requests to the appropriate microservice in the **EKS/ECS cluster**.
- The **Service Mesh** handles internal routing, retries, and observability.

3. Microservice Processing

- The target microservice executes business logic and may:
  - Query or update **DynamoDB (regional replica)**.
  - Publish domain events to **EventBridge** for asynchronous propagation.
  - Store or retrieve static assets from **S3**.

4. Data Persistence

- **DynamoDB Global Tables** replicate changes across regions in near real-time.
- **S3 Cross-Region Replication** ensures object consistency for shared assets.
- Optional **RDS/Aurora** instances handle relational transactions and replicate read replicas cross-region.

5. Event Propagation

- **EventBridge** broadcasts events to peer regions, triggering **Lambda** functions or microservices subscribed to those events.
- This enables **event-driven synchronization** and cross-region orchestration.

6. Response Path

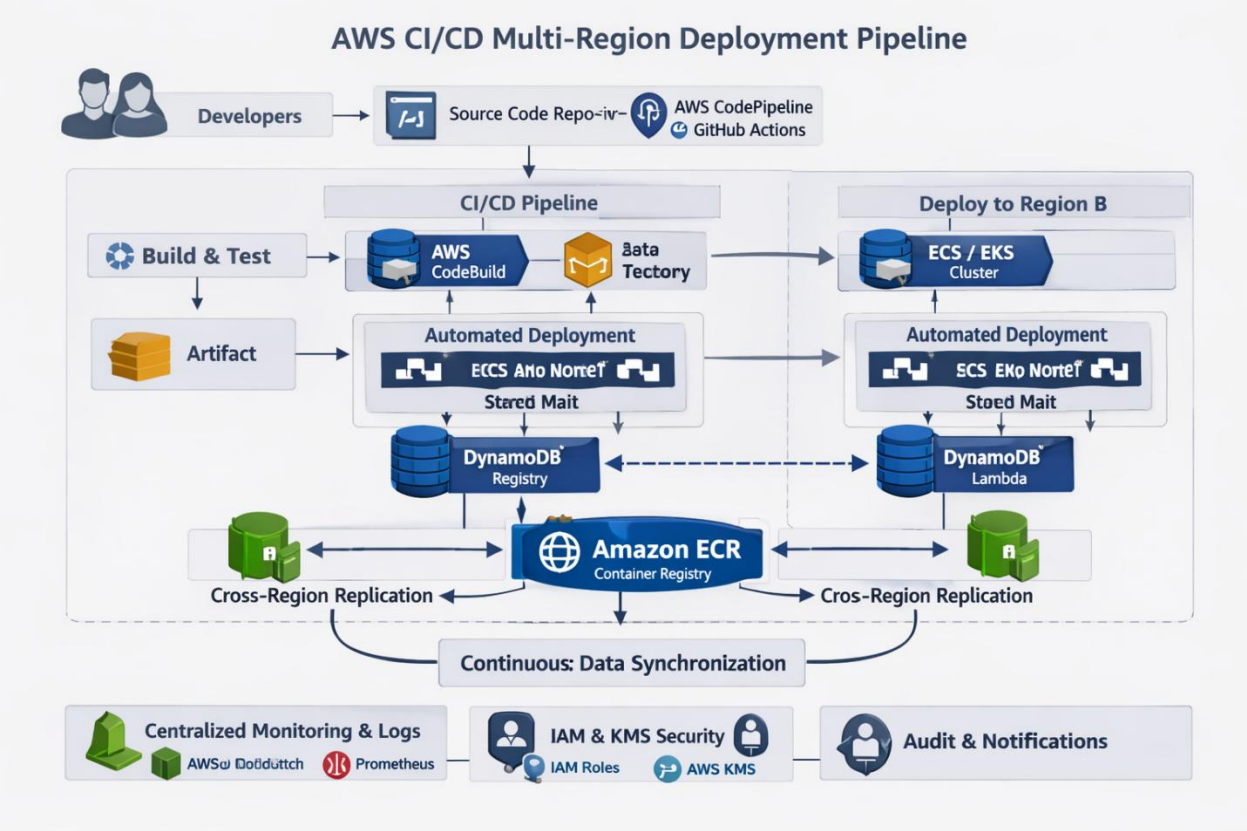
- The microservice returns the processed result to the **API Gateway / ALB**, which sends it back through **Route 53** to the end user.

- **CloudFront** may cache responses globally for performance optimization.

7. Observability & Governance

- **CloudWatch**, **X-Ray**, and **Prometheus** capture metrics and traces across the entire flow.
- **IAM**, **KMS**, and **Security Groups/NACLs** enforce security and compliance throughout the data path.

CI/CD Multi-Region Deployment Pipeline



1. Developer Source and Version Control

At the top of the diagram, **Developers** commit code to a **Source Code Repository** such as **GitHub**, **AWS CodeCommit**, or another Git-based system.

- This repository serves as the **single source of truth** for all application code, infrastructure definitions, and configuration files.

- Every commit or pull request triggers the **CI/CD Pipeline**, ensuring automated build, test, and deployment across all regions.

2. Continuous Integration (CI) Pipeline

The **CI/CD Pipeline** begins with **AWS CodePipeline** or **GitHub Actions**, orchestrating the end-to-end workflow.

- **Build & Test Stage:**
    - **AWS CodeBuild** compiles source code, runs unit and integration tests, and packages artifacts.
    - Test results are automatically reported back to developers, enforcing quality gates.
- **Artifact Creation:**
    - Successful builds produce deployable artifacts (container images, Lambda packages, or static assets).
    - These artifacts are stored in **Amazon ECR (Container Registry)** for versioned, immutable storage.

3. Automated Multi-Region Deployment

Once artifacts are ready, the pipeline triggers **automated deployments** to multiple AWS regions — shown as **Region A** and **Region B** in the diagram.

- Each region hosts its own **ECS/EKS Cluster**, ensuring isolation and resilience.
- Deployment automation uses **CodeDeploy**, **Argo CD**, or **CloudFormation Stacks** to roll out updates consistently.
- **AWS Lambda** functions may handle post-deployment tasks such as configuration validation or event propagation.
- **Amazon S3** stores static assets and configuration files, replicated across regions for consistency.
- **Amazon DynamoDB** (regional tables) synchronize through **Global Tables**, maintaining data integrity across deployments.

4. Cross-Region Synchronization

The diagram highlights **Cross-Region Replication** and **Continuous Data Synchronization** between regions:

- **Amazon ECR** ensures container images are available in all regions to minimize deployment latency.
- **S3 Cross-Region Replication** keeps static assets identical across regions.
- **DynamoDB Global Tables** replicate state and configuration data automatically. This guarantees that both regions operate in **active/active mode**, capable of serving traffic independently.

5. Centralized Monitoring & Logs

At the bottom left, **Amazon CloudWatch**, **AWS X-Ray**, and **Prometheus** provide unified observability:

- **CloudWatch** aggregates metrics, logs, and alarms from all regions.
- **X-Ray** traces requests through microservices for performance analysis.
- **Prometheus** collects container-level metrics for real-time monitoring. Together, they enable **global visibility** into build health, deployment status, and runtime performance.

6. IAM & KMS Security

In the center bottom section, **IAM Roles** and **AWS KMS** enforce security and compliance:

- **IAM Roles** define least-privilege access for pipeline components, build agents, and deployment services.
- **KMS** encrypts artifacts, secrets, and environment variables at rest and in transit. This ensures that every stage of the pipeline adheres to enterprise-grade security standards.

7. Audit & Notifications

On the bottom right, **AWS CloudTrail** and **Amazon SNS** handle governance and communication:

- **CloudTrail** logs all API calls and pipeline actions for auditability.

- **SNS** sends notifications on build success, failure, or deployment completion to teams or chat systems. These components close the feedback loop, ensuring transparency and rapid incident response.

8. Strategic Outcome

This CI/CD architecture delivers:

- **Automated, repeatable deployments** across multiple regions.
- **Consistent artifact management** and data synchronization.
- **Unified monitoring and security governance** across the global footprint.
- **Resilient, low-latency operations** supporting active/active regional workloads.