Multi-Region Expansion Strategy – “Blockchain Solutions Inc.”

Scott Buckel – SRE Case Study 5/20/2025

Problem Summary

- All infrastructure is located in a single region, eu-west-1
- Active and dormant service pairs, but failover is manual and error-prone
- All infrastructure is managed via a single terraform repository
    - Terraform apply is executed manually by engineers, often times without any peer review
    - Increases risk of errors and configuration drift
- There is minimal deployment automation

Objectives of Expansion

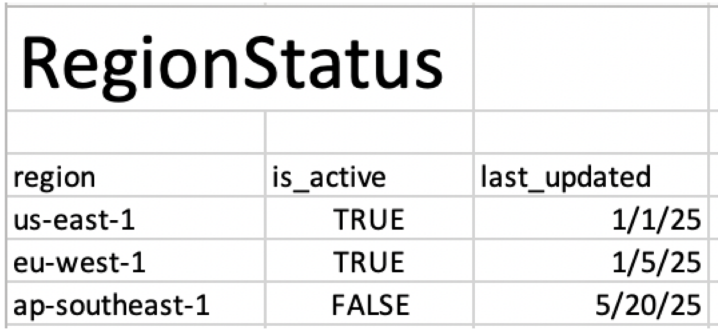
- Ensure services are globally available
- Support disaster recovery and failover
- Enable scalable and consistent infrastructure across multiple regions
- Improve automation, observability, and most importantly - reliability

High Level Architecture (text version)

- Multiple AWS regions: eu-west-1 and adding us-east-1 for now
- Route 53 latency-based routing between regional ALBs
- Stateless services – active/active
    - EKS clusters in both regions running stateless services
    - Microservice with DynamoDB (global table) backend to coordinate if region is active
- Stateful services – active/passive
    - Microservice with DynamoDB (global table) backend to coordinate which region should be active for the stateful service
- Terraform workspaces to manage multi-region consistency
- Monitoring via Datadog
- Anti-affinity enabled for all service to enforce pod distribution across availability zones

region-status microservice (stateless)

- Stateless services
  - Simple on/off switch for each stateless service in a region-agnostic format. Ideal for services in active/active mode but can also help with coordinating deployments. This can be used to shut down an entire region for both stateless and stateful services.
  - Table Name:
    - RegionStatus
  - Attributes:
    - region
    - is_active
    - last_updated
- Healthcheck Endpoint
  - GET /status/:region

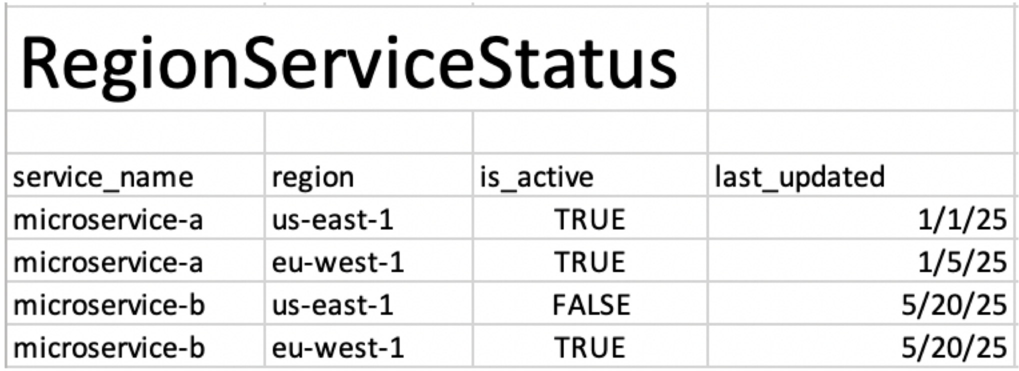


RegionStatus

region	is_active	last_updated
us-east-1	TRUE	1/1/25
eu-west-1	TRUE	1/5/25
ap-southeast-1	FALSE	5/20/25

region-status microservice (stateful)

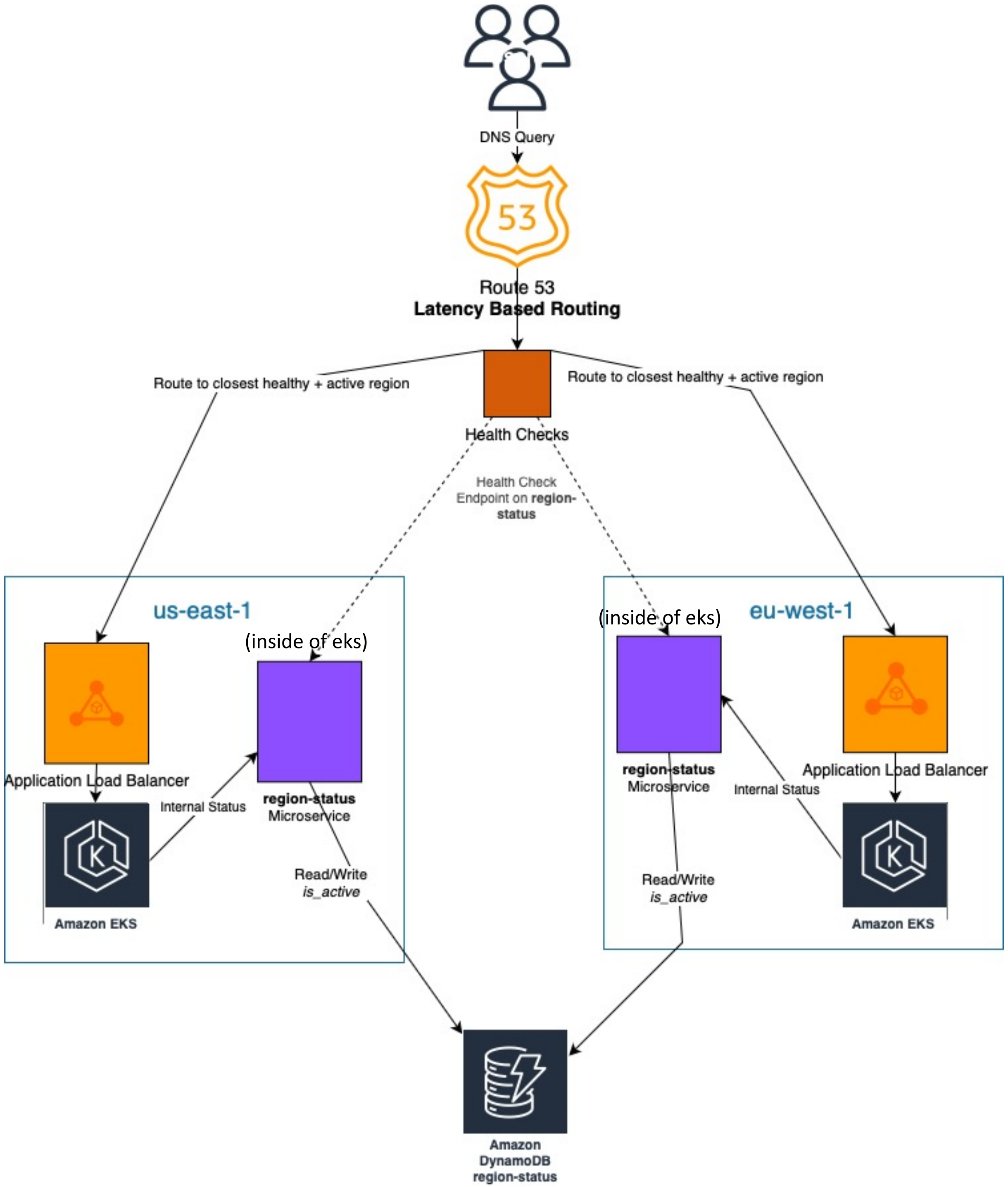
- Stateful services:
  - Tracks per-region status for individual services that are active/passive
  - Table Name:
    - RegionServiceStatus
  - Attributes:
    - service_name
    - region
    - is_active
    - last_updated
- Healthcheck Endpoint:
  - GET /status/:service/:region
- Healthcheck would also check the microservices healthcheck (eg: http://servicename:8080/actuator/health)
- Bonus: A service can be active in multiple regions if desired and not in others



RegionServiceStatus

service_name	region	is_active	last_updated
microservice-a	us-east-1	TRUE	1/1/25
microservice-a	eu-west-1	TRUE	1/5/25
microservice-b	us-east-1	FALSE	5/20/25
microservice-b	eu-west-1	TRUE	5/20/25

Multi Region Architecture


DNS Query
53
Route 53
Latency Based Routing
Route to closest healthy + active region
Route to closest healthy + active region
Health Checks
Health Check Endpoint on region-status
us-east-1
(inside of eks)
Application Load Balancer
Internal Status
region-status Microservice
Amazon EKS
Read/Write is_active
eu-west-1
(inside of eks)
region-status Microservice
Application Load Balancer
Internal Status
Amazon EKS
Read/Write is_active
Amazon DynamoDB region-status

Stateless Services Architecture

- Deployed to all regions, in active/active mode
- Route53 Latency based routing will route traffic to the closest healthy region
- Kubernetes HPA, cluster-autoscaler for load spikes

Stateful Services Architecture

- Optionally deployed to all regions
  - We always have the option to deploy service and keep service scaled down to 0 pods – this should give us faster reaction time in case of disaster
- Route53 Latency based routing will route traffic to the closest healthy region
- Kubernetes HPA, cluster-autoscaler for regional load spikes
- More research is needed: how do we stay in sync when a failover will need to happen? How important are these services? Do they need to stay in sync 24/7, or can there be a few minutes of downtime.

Deployment Process

- Repository changes
  - Individual AWS services will each have their own git repositories
    - tfe-eks for eks
    - tfe-storage for s3 buckets and databases (DynamoDB)
    - tfe-route53 for Route53 Setup
  - Changes require peer approvals
- Terraform changes
  - Terraform Enterprise self-hosted
    - Developers can no longer run `terraform apply` locally
  - Multiple terraform workspaces and .tfvars files for the unique environment
- Adopt GitOps
  - No manual CI/CD scripts to build deploy services. Services are built upon merge to release branch.
  - CD stages for dev->staging->prod with manual gates and quality checks in between. This supports gradual deployments.
- Centralize Helm Management
  - Shared standard helm charts across services
- Post deployment checks
  - Datadog for monitoring

Disaster Recovery

- If an entire region or microservice is down:
    - Healthcheck would fail, automatically moving traffic to the healthy region.
    - If Stateful services only have one region configured (this will be the most likely scenario when a stateful service goes down):
        - We update RegionServiceStatus table to move the traffic to the new region
- If entire region or microservice is misconfigured:
    - Update DynamoDB record in RegionStatus/RegionServiceStatus table to take entire region offline
    - Estimated time to resolve: 5 minutes (limited by TTL of the Route53 Record)
- Monitoring:
    - A Lambda could automatically trigger adjusting the active region for a passive service, including syncing the data if necessary

Observability and Monitoring

- Leverage Datadog for logging, monitoring, and observability, including APM
  - It can tell us if one region is acting much slower than another region
  - It can tell us if a new version of a service has higher response times, error rates, or resource usage. In the past I’ve actually seen this save money in the long run
  - We can correlate logs with stack traces when things go wrong
- SLO/SLI/SLA
  - Datadog will help us define and monitor these metrics – potential options are latency, error rates, availability, etc. For example: 99.99% availability, 200ms latency for 95% of requests
  - Different regions could have different SLO’s (in developing countries, requests could take longer?)

Tradeoffs

- **COST**
  - Spinning up a new active/active region doubles the cost of EKS
  - Datadog monitoring and alerting is great, but comes with cost
  - Explanation to management: “Increase in cost is justified by the significant improvement in reliability, DR capabilities, and global user experience which directly impacts business continuity”
- **COMPLEXITY**
  - Managing the new infrastructure is much more complicated than managing one cluster
  - Adding a new microservice to maintain
- **SPEED**
  - These changes could initially slow down developer velocity, as a tradeoff for reliability and consistency
  - Developers won’t be able to \`terraform apply\` or \`helm install\` or \`helm upgrade\` anymore. This will be handled in CI/CD Pipelines.
  - **Eventually, with robust documentation, clear guidelines, automation for common deployment patterns, new developers will be able to dive right in.**

Timeline and Roadmap

- Phase 1
  - Terraform changes for Infrastructure – TFE, separate repositories, workspaces
  - CI/CD change for Services, including a centralized helm management
  - In parallel, devs will build region-status microservice
- Phase 2
  - Deploy Route53 and point it to ALB->K8s
  - Deploy DynamoDB
  - Deploy second cluster, deploy microservices to it
- Phase 3
  - Update Route53 to point to both regions with region-status as the healthcheck

Questions?

- From me:
  - How will stateful services stay in sync? It wasn’t mentioned. I’m not sure what stateful services will be.
  - Will stateful<-->stateless services need to communicate?