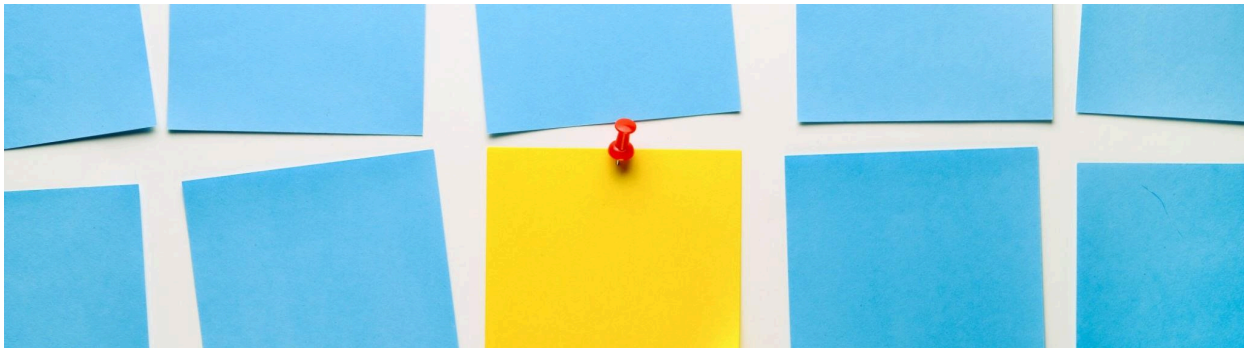


Bonmoja Take-home assignment

Status Completed

Timing May 23, 2025 to May 26, 2025

Owners Prince Gansie

Overview

This document outlines the completion of the assigned take-home project to **design, provision, and document an AWS-based infrastructure simulating a simple messaging system** using a containerised HTTP service (`hashicorp/http-echo`).

Objectives

Project objectives

- **Design and provision an AWS-based infrastructure** for a simple messaging system using Terraform or CloudFormation.
- **Containerise and deploy an HTTP service** (`hashicorp/http-echo`) on ECS Fargate.
- **Implement supporting AWS services**, including VPC, RDS (PostgreSQL), DynamoDB, SQS, and SNS with proper IAM and security groups.
- **Build a CI/CD pipeline** using GitHub Actions or CircleCI to automate Docker image deployment and infrastructure provisioning.
- **Create a health check script** to validate the deployed service post-deployment.
- **Configure monitoring and alerting** using CloudWatch logs and alarms.
- **Identify and document cost optimisation strategies** for AWS services used.

Strategy

✅ 1. Infrastructure as Code (IaC) – Completed

Provisioned using **Terraform**, with separation between environments:

- **VPC Design**:
    - Custom VPC with **2 public** and **2 private subnets** across multiple Availability Zones for high availability.
    - Configured **NAT Gateways** to allow outbound internet access from private subnets.

- **Routing & DNS**:
    - Application Load Balancer (ALB) targeting the ECS service.
    - Integrated with **Route 53 DNS records** for domain resolution.

- **Terraform Environments**:
    - `dev.tfvars`: Basic configuration aligning with task requirements.

    - `pro.tfvars`: Advanced implementation using production best practices.

- **Security Groups**:
    - Properly **segregated security group rules**, enforcing strict access between DMZ (public ECS tasks) and private datastores (RDS, DynamoDB).
    - IAM roles follow **least-privilege principles**.

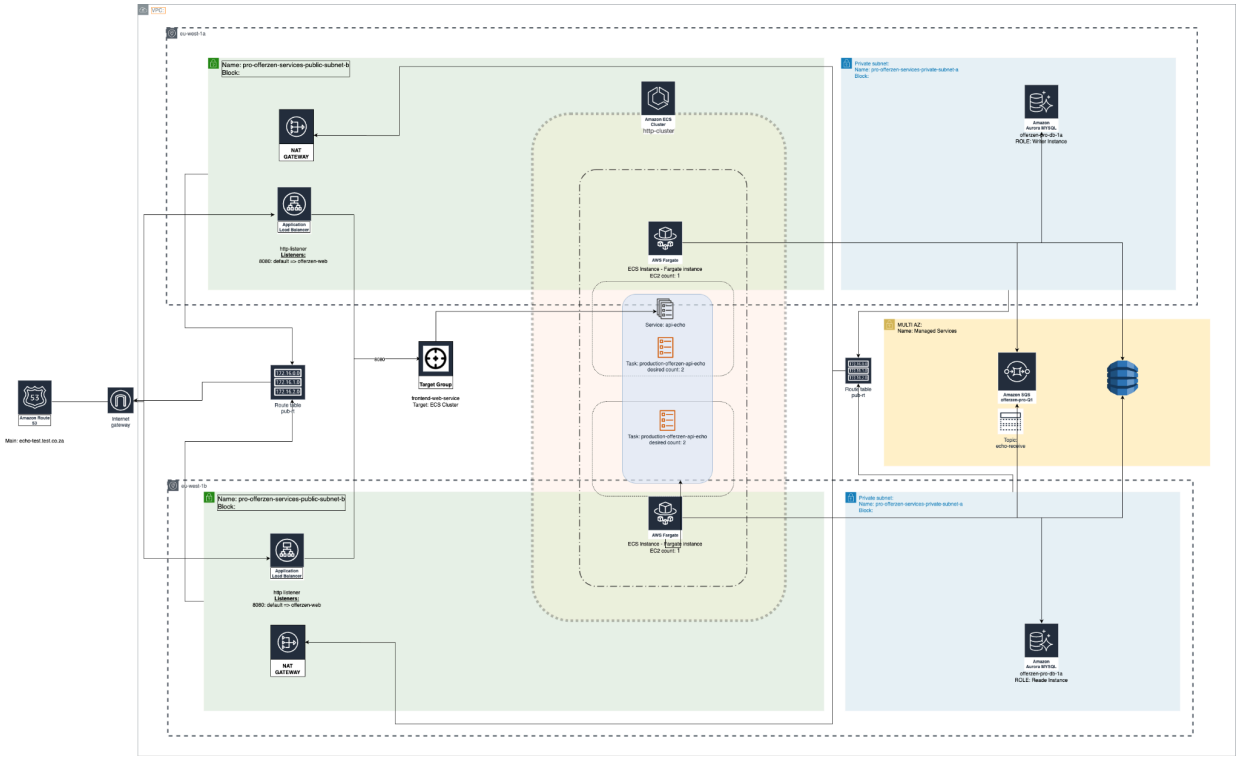
- **ECS (Fargate)**:
    - Deployed `hashicorp/http-echo` container via ECS Fargate using Terraform.
    - Proper roles and policies assigned to ECS tasks for interaction with other AWS services.

- **RDS (PostgreSQL)**:
    - RDS cluster setup with **reader/writer endpoints** and **failover support**.
    - Configured security and IAM access from ECS tasks only.

- **DynamoDB**:
    - Provisioned with access control defined via IAM policies scoped to ECS tasks.

- **SQS & SNS**:
    - Configured **SQS queue** and **SNS topic** with at least one subscription.
    - IAM roles for ECS tasks to interact with both services securely.

Infrastructure MAP:



🔁 2. CI/CD Pipeline – Completed

Implemented using **GitHub Actions**:

- **Docker Image Build & Push**:
  - CI workflow to build and push Docker images to **Amazon ECR**.
- **Terraform Workflow**:
  - Linting and validation of Terraform code (`terraform fmt`, `terraform validate`).
  - Infrastructure provisioning to a **staging environment**.
- **Deployment**:
  - Automated deployment of ECS services with updated ECR image tags.
  - Integrated **post-deploy health check** step to ensure application readiness.

🛠️ 3. Automation and Scripting – Completed

Health check automation:

- **Script (Bash)**:
    - Sends an HTTP request to the deployed `http-echo` service.
    - Logs results and **fails the pipeline** if the HTTP response is not `200 OK`.
    - Ensures stability by maintaining the previous deployment state on failure.

📊 4. Monitoring and Alerting – Completed

Provisioned and/or documented via Terraform:

- **CloudWatch Log Groups**:
    - Logs configured for ECS service to support observability.
- **CloudWatch Alarms**:
    - RDS CPU Utilisation > 80% for 5 minutes.
    - SQS Queue Depth > 100 messages for 10 minutes.
    - **Note**: Implemented **flapping prevention** on the CPU alarm by requiring **two consecutive breaches** before alarm state triggers.
- **SNS Topic Integration**:
    - Alarms notify via **SNS topic subscription** for real-time alerting.

💰 5. Cost Optimisation – Completed

Outlined cost-saving strategies with trade-offs:

- **ECS Optimisation**:
    - Suggest transitioning from **Fargate to EC2-backed ECS** in production for better cost control.
    - Enables options like:
        - **Spot instances** for savings.
        - **Instance pools** with auto scaling.

        - **Low-cost compute during non-peak hours**.

- **Other Strategies**:

    - Use **Savings Plans or Reserved Instances** for predictable workloads (e.g., RDS).
    - Employ **DynamoDB On-Demand** for low/irregular workloads.

📦 Repository Structure Overview

- `terraform/`: All Terraform modules and environment files.
- `.github/workflows/`: GitHub Actions CI/CD configuration.
- `scripts/health_check.sh`: Health check script.
- `README.md`: Setup instructions, architecture diagram, usage guide.
- `SOLUTION.md`: Architecture decisions, trade-offs, security and cost optimisation insights.

🏁 Final Summary

Thank you for the opportunity to work through this take-home assignment. It was both a challenging and rewarding exercise that allowed me to demonstrate my hands-on experience in designing scalable, secure, and automated cloud-native infrastructure on AWS.

The solution I delivered reflects my focus on **best practices**, such as high availability, least-privilege access, production-aware CI/CD pipelines, and cost-conscious architecture decisions. I took the liberty of including both a standard implementation (`dev.tfvars`) and production-ready variant (`pro.tfvars`) to illustrate how I approach real-world infrastructure challenges.

Please feel free to explore the repository for full code, configuration, diagrams, and documentation.