Senior DevOps Engineer Hands-On Exercise **Title:** Deploy a Ruby on Rails API using Docker, Terraform, and GitHub Actions with an ALB in AWS

Objective Demonstrate the candidate’s ability to: - Containerize a Ruby on Rails (RoR) app - Provision AWS infrastructure via Terraform - Automate build & deployment via GitHub Actions CI/CD - Securely expose an API endpoint via an AWS Application Load Balancer (ALB)

---

1. Application Setup (Dockerization) - The application is a simple Rails API with one endpoint `/health` returning `{ "status": "ok" }`. - Containerize it using Docker.

Expected deliverables: - Dockerfile (multi-stage build preferred) - .dockerignore - Ability to run locally: docker build -t rails-api . docker run -p 8080:3000 rails-api - The `/health` endpoint should be reachable at http://localhost:8080/health.

---

2. Infrastructure Setup (Terraform) Use Terraform to provision: - AWS VPC with public subnets - ECS Cluster (Fargate or EC2 launch type) - ECR Repository for storing images - Application Load Balancer (ALB) exposing the Rails API - Security Group allowing inbound :80 and health check from ALB - IAM roles/policies for ECS Task Execution - Environment variables (DB_HOST, RAILS_ENV, etc.) via SSM Parameter Store

Bonus points: - Use Terraform modules for vpc, ecs, and alb - Use a remote backend with S3 + DynamoDB locking

Expected deliverables: infra/ ■■■ main.tf ■■■ variables.tf ■■■ outputs.tf ■■■ ecs.tf ■■■ alb.tf ■■■ ecr.tf ■■■ provider.tf

---

3. CI/CD (GitHub Actions) Automate deployment with .github/workflows/deploy.yml.

Trigger: - On push to main branch

Stages: 1. Build & Push Docker Image - Build image from Dockerfile - Tag with commit SHA - Push to ECR 2. Terraform Plan & Apply - Initialize Terraform - Run terraform plan - If successful, terraform apply -auto-approve 3. Deploy to ECS - Update ECS task definition with the new image - Force new deployment 4. Post-Deploy Verification - Curl the ALB endpoint /health - Fail workflow if not returning { "status": "ok" }

Expected deliverables: - .github/workflows/deploy.yml - Use GitHub OIDC for AWS authentication (no static keys) - Slack or GitHub Actions summary notification (bonus)

---

4. API Exposure via ALB - The deployed API should be accessible via ALB public DNS on port 80.

Example: curl http:///health => {"status":"ok"}

Bonus points: - Enable HTTPS using ACM + listener - Configure health check path /health - Apply sensible idle timeout & deregistration delay

---

5. Evaluation Criteria

| Area | Weight | Evaluation Focus | |------|--------|------------------| | Containerization | 20% | Multi-stage build, clean image, Docker best practices | | Terraform Design | 30% | Module use, variable organization, state mgmt, tagging, IAM least privilege | | CI/CD Automation | 25% | Workflow efficiency, reusability, OIDC auth, error handling | | Networking & ALB | 15% | Security groups, listeners, health checks | | Observability & Clean-up | 10% | Logging, metrics, teardown readiness |

---

6. Advanced Follow-up Questions 1. How would you handle secrets (Rails master key, DB password) securely in this setup? 2. How would you implement blue/green deployment or canary releases for ECS? 3. How would you enable auto-scaling based on request load or latency? 4. How would you integrate CloudWatch logs or Datadog metrics for the Rails container? 5. What are the trade-offs between ECS on EC2 vs ECS on Fargate?