Innovate Inc. Cloud Infrastructure Design (AWS)

Overview

This document outlines the cloud infrastructure design for Innovate Inc., a startup developing a web application with the following components:

- **Backend**: Python (Flask)
- **Frontend**: React (SPA)
- **Database**: PostgreSQL
- **Cloud Provider**: AWS
- **CI/CD**: Continuous Deployment pipeline with GitHub Actions
- **Security**: Handles sensitive user data with strong security requirements

1. Cloud Environment Structure

AWS Account Structure

Recommendation: Use a multi-account strategy following AWS best practices.

Account Name	Purpose
`org-root`	Management account (for consolidated billing, guardrails)
`dev`	Development environment, non-production workloads
`staging`	Staging environment, pre-production testing
`prod`	Production environment, runs user-facing services
`shared`	Shared services (logging, monitoring, IAM, security)

Justification:

- **Isolation**: Limits blast radius between environments
- **Security**: Separate IAM boundaries
- **Billing**: Enables cost tracking per environment
- **Governance**: Easier to implement SCPs, budgets, and guardrails

2. Network Design

VPC Architecture

Each environment (dev, staging, prod) will have its own VPC with the following setup:

- **3 Availability Zones** for high availability
- **Private Subnets** for backend applications and databases
- **Public Subnets** for load balancers and ingress traffic
- **NAT Gateways** to enable egress internet access for private subnets

Network Security

- Use **Security Groups** to tightly control traffic between resources
- Use **Network ACLs** for stateless subnet-level rules
- Use **VPC Endpoints** to securely access AWS services (e.g., S3, Secrets Manager)
- All internal traffic encrypted with TLS
- Use **AWS WAFv2** in front of the Application Load Balancer

3. Compute Platform

Kubernetes (Amazon EKS with Karpenter and EC2 Nodes)

Use **Amazon EKS** with EC2-based worker nodes and **Karpenter** for dynamic scaling.

Design:

- One EKS cluster per environment
- Use **Karpenter** to automatically provision EC2 instances based on workload requirements
- Support for **x86_64** and **Graviton (ARM64)** instance architectures
- Use **Spot Instances** for cost efficiency with fallback to On-Demand
- Minimal managed node group for core Kubernetes services

Benefits:

- Full control over compute types
- Automatic, intelligent scaling with Karpenter
- Cost optimization using Spot and Graviton

Containerization & Deployment

- Dockerfiles for frontend and backend
- Push container images to **Amazon ECR**
- CI/CD pipeline with **GitHub Actions**:
    - Build, test, and scan images
    - Push to ECR
    - Deploy to EKS using `kubectl` or GitOps tool like ArgoCD

Secrets Management:

- Store sensitive data in **AWS Secrets Manager**
- Sync secrets into Kubernetes using External Secrets Operator or IRSA

Public Access and WAF

- Frontend service exposed using **Application Load Balancer (ALB)**
- Attach **AWS WAFv2** to ALB for protection
- Kubernetes Ingress Controller configured for routing and TLS termination

ALB Annotations Example:

```
alb.ingress.kubernetes.io/scheme: internet-facing
alb.ingress.kubernetes.io/wafv2-acl-arn:
arn:aws:wafv2:<region>:<account-id>:regional/webacl/my-waf-acl
```

4. Database

Service Recommendation

Use **Amazon Aurora for PostgreSQL** for production environment and **RDS for PostgreSQL** for lower-environments with the following configuration:

- Multi-AZ deployment for fault tolerance
- Automated backups and point-in-time recovery
- Enable encryption at rest and in transit
- Store credentials in Secrets Manager

Backup and HA Strategy

- Daily backups with retention (7–30 days)
- Read replicas for offloading read traffic
- Cross-region snapshot replication (optional)

- Maintenance windows with automatic patching

5. Cost Optimization

- **Karpenter** auto-scales based on real-time pod requirements
- Use **Spot Instances** as primary with fallback to On-Demand
- Prefer **Graviton** instances for ARM-compatible workloads
- Dev/staging databases with auto-pause enabled
- Static assets optionally served via S3 + CloudFront

6. Monitoring & Logging

- Use **Amazon CloudWatch** for metrics and logging
- Deploy **Prometheus + Grafana** for in-cluster observability
- Enable **AWS X-Ray** for tracing backend services (optional)
- Set up alerts on cluster, WAF, and DB metrics

7. Security Best Practices

- Enable **IAM Roles for Service Accounts (IRSA)**
- Use **KMS encryption** for RDS, EBS, and S3
- Enforce **TLS encryption** in all communication paths
- Enable **GuardDuty**, **Security Hub**, and **AWS Config**
- Use **AWS WAF** with preconfigured rules for OWASP and rate limiting

High-Level Architecture Diagram

`In docs/README.md`

Next Steps

- Build Terraform modules for VPC, EKS, Karpenter, RDS, WAF, and ALB
- Create Kubernetes manifests or Helm charts for app deployment
- Configure GitHub Actions CI/CD pipelines
- Define policies for Karpenter and IAM roles (IRSA)
- Test Spot + Graviton scheduling via `nodeSelector` and taints/tolerations

References

- AWS EKS Best Practices Guide
- AWS Multi-Account Strategy
- Karpenter Autoscaler

- AWS WAF Documentation
- AWS Secrets Manager
- Amazon RDS for PostgreSQL