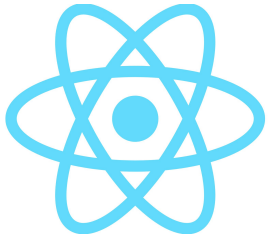
Hybrid Infrastructure Deployment & Application Orchestration

Introduction

Brief overview of the project: transitioning to a hybrid infrastructure (80% on-premises, 20% AWS) to deploy a simple web application.

Key components:

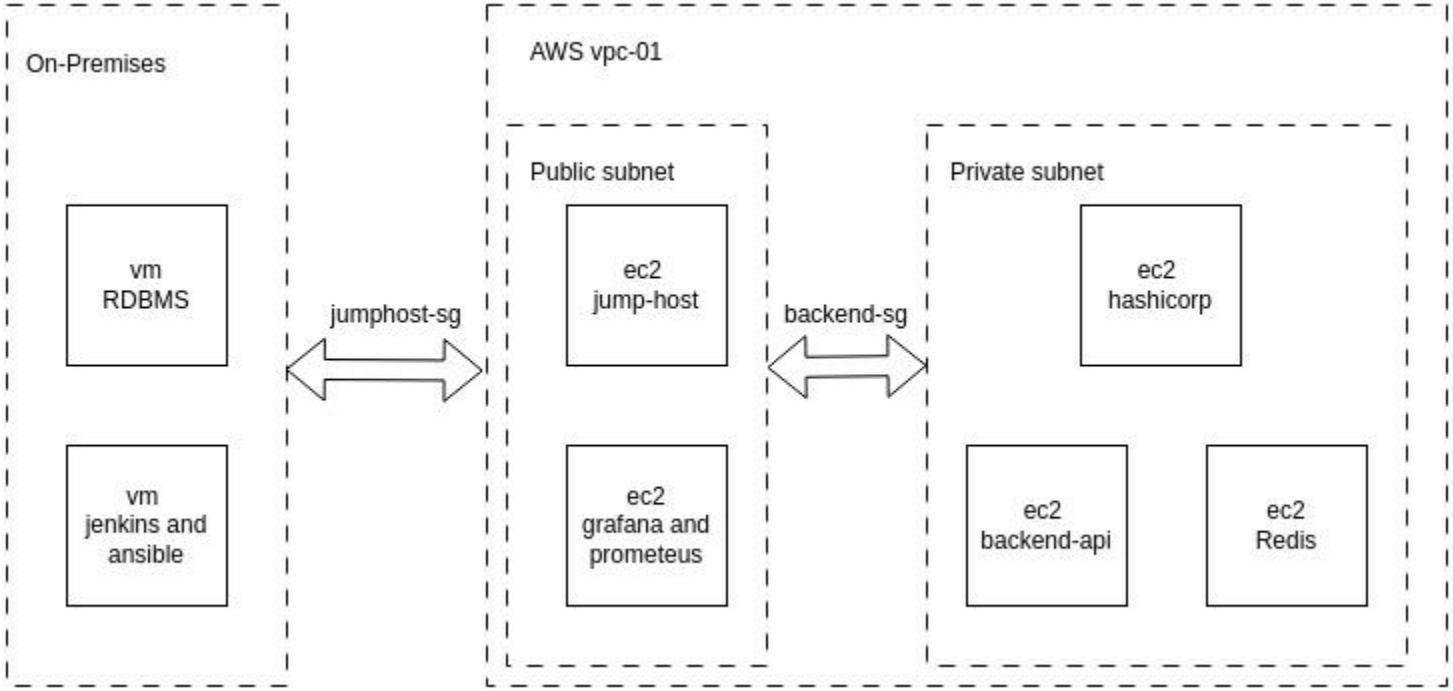
- Frontend (React)
- Backend API (Flask)
- RDBMS (PostgreSQL)
- Redis,
- Terraform
- Ansible
- Nomad
- Consul
- Vault
- Jenkins
- Docker
- Prometheus
- Grafana.



Infrastructure Provisioning

Terraform to provision a VPC with public and private subnets, EC2 instances, RDS, and Elasticache.

Ansible for On-Premises like Jenkins, Nomad, Consul, and Vault.

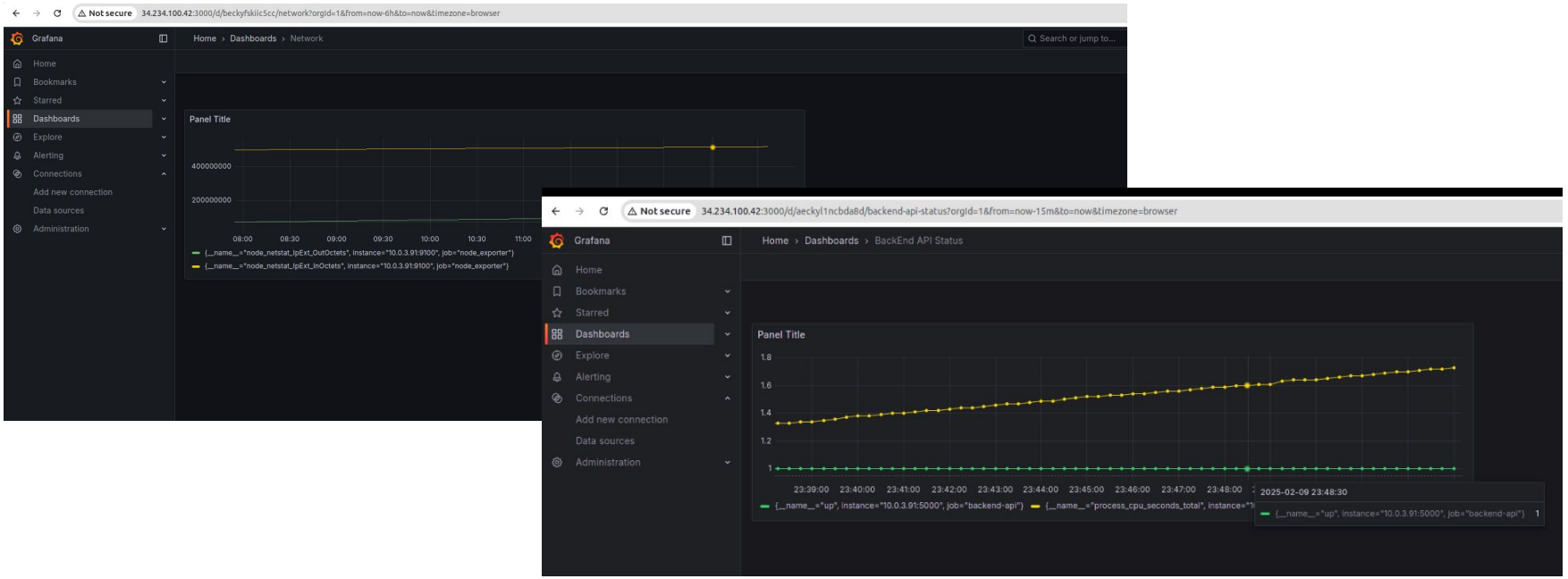


Containerization

- Docker is used to package the backend API.
- CI/CD Pipeline:
- Jenkins Pipeline builds and pushes the Docker image (tagged with BUILD_NUMBER), and then triggers a Nomad job deployment.
- Orchestration & Service Discovery:
- Nomad deploys the backend API container.
- Consul monitors service health and enables dynamic discovery.
- Secrets Management:
- Vault manages sensitive information (e.g., database credentials)

Monitoring & Alerting

- Prometheus & Grafana:
- Prometheus scrapes metrics from the backend API and Node Exporter.
- Grafana is used to visualize these metrics.
- Alertmanager:
- Configured to send alerts for issues like high CPU usage or application downtime.



Key Challenges & Solutions

- The Terraform block for the RDS instance was disabled because my IAM user did not have sufficient permissions to provision that resource in AWS. Moreover, the assignment instructions were somewhat contradictory—one part specified that the RDBMS should be on-premises, while another suggested deploying it on AWS. Given the permissions constraints and this ambiguity, I opted to deploy the RDBMS on-premises.
- DNS Resolution in Docker Builds - Resolved by using the --network=host flag in the Jenkins Pipeline.
- Hybrid Environment Integration - Separated cloud provisioning (Terraform) from on-premises configuration (Ansible) to maintain clarity.
- Sensitive File Management - Excluded Terraform state and lock files from version control to prevent accidental exposure.
- Secure Communication - Implemented TLS and strict IAM roles to ensure secure data exchange between services.

Conclusion & Future Enhancements

- The implemented solution provides a **robust, scalable, and secure hybrid infrastructure**, ensuring seamless integration between on-premises and cloud environments.
- Future improvements could include **enhanced multi-environment support** (Dev, QA, Staging, Production) to streamline deployments across different stages.
- Further **automation in deployment and monitoring** can optimize operational efficiency and reduce manual intervention.
- The integration of **industry-standard DevOps tools** ensures a **reliable, maintainable, and efficient** CI/CD pipeline, supporting long-term growth and scalability.