

Advanced Capstone Project

Components:

- 1. Infrastructure Automation**
 - Use Terraform to provision a scalable Kubernetes-based architecture.
 - Introduce Load Balancers and HA configurations.
- 2. CI/CD Pipeline**
 - CI/CD workflows with automated testing, security checks, and rollback capabilities. Environments branching support.
- 3. Monitoring and Alerting**
 - Implement resource monitoring, application performance tracking, and automated alerts.
- 4. Secrets Management**
 - Primary Secret Manager: Use Cloud Secret Manager for all secrets related to infrastructure, database, and CI/CD.
- 5. Architecture diagram**

Source code: GitHub or GitLab

Cloud: GCP

Infrastructure automation tool: Terraform

Remote state storage for Terraform: Google Cloud Storage

Configuration management: Helm

CI/CD automation tool: GitLab CI/GitHub Actions/Jenkins

Build tool: Maven or Gradle

Artifacts: Docker images

Artifact storage: Google Container Registry/GitLab Registry

Persistent database for application: Cloud SQL in GCP

Scripts: Python and/or Bash

Requirements:

Infrastructure Automation

Create an infrastructure automation pipeline that prepares the environment for application deployment using the Infrastructure as Code approach. Infrastructure configuration should be in a separate repository from the application source code.

Some preparation steps can be done manually or by running automation scripts locally:

- If you're going to use Amazon S3, Azure Blob Storage, or Google Cloud Storage as a remote state storage for Terraform, it should be created in advance.

- If you're going to use GitLab SaaS (GitLab.com) or GitHub Actions, you need a runner for your first job. In the beginning, it can be installed on your local machine or terraform for runner creation can be run locally.
- If you're going to use Jenkins, then create a virtual machine, install Jenkins and take care of agents.

Infrastructure provisioning pipeline.

It should include the following jobs:

- configuration formatting,
- configuration validation and scanning (tflint, tfsec, fmt, validate)
- Plan,
- provisioning resources (manual job),
- destroying resources (manual job).

Use Terraform to create the following resources in the cloud:

- A managed Kubernetes cluster to host the application.
- Autoscaling based on CPU usage.
- A Load Balancer to distribute traffic across applications.
- A persistent database for application.
- Additional virtual machine with all the needed network-related resources to run additional software like Gitlab/GitHub Actions runners, Nexus (if used).
- Bucket for Terraform State (GCS with CMEK for Terraform remote state storage)

Configuration Management

Use Helm to deploy applications and any required middleware tools to K8S:

- Deploy applications.
- Install monitoring agents (e.g., Prometheus node exporter or OpenTelemetry collector, Grafana, Loki).
- For the Load Balancing, you can choose from the native cloud load-balancer (as a K8s service), ingress-nginx, or istio (advanced option).
- Manage sensitive data using a secret manager.

CI/CD Pipeline

A Continuous Integration and Continuous Delivery solution for the Java application [spring-petclinic](#).

The repository with the spring-petclinic application source code should additionally have configuration files for Maven or Gradle and a Dockerfile.

In the artifact storage of your choice prepare a registry.

CI/CD configuration (branches, pipelines, terraform code) should support several environments (ex., dev, qa, prod).

The pipeline for a merge request(MR) or a pull request(PR) should include:

- static code analysis,
- Security scans, ex. trivy for Docker image vulnerability scans;
- Tests,
- Build,
- creating an artifact (it can be tagged with a short commit hash),
- pushing the artifact to the artifact storage

The pipeline for the main branch should include:

- creating a Git tag in the repository using [Semantic Versioning](#) approach (a minor version increases on each commit). A python script with [semver · PyPI](#) can be used here.
- creating an artifact with Git tag representing the version
- pushing the artifact to the artifact storage
- a manual deployment job that:
 - connects to a k8s cluster,
 - checks if a previous version of the application is present and removes it,
 - gets the image from the container storage,
 - runs the application making sure it is connected to a MySQL database in the cloud,
 - prints the link to the application.

Secrets Management

- Primary Secret Manager: Use a cloud Secret Manager for all secrets related to infrastructure, database, and CI/CD.

Monitoring and Logging

- Use Prometheus and Grafana:
 - Export metrics from the application and k8s cluster.
 - Visualize metrics in Grafana dashboards.
- Optional: Use GCP's Cloud Monitoring and Logging:
 - Monitor k8s health, application logs, and performance metrics.
 - Create dashboards for:
 - Application uptime.
 - Resource usage (CPU, memory, disk).
 - Database queries and latency.
 - Set up alerts for critical metrics like:
 - High CPU/memory usage.
 - Application downtime.

Important

- Use a Service Account with minimal permissions for Terraform and CI/CD tools.
- All resources should **not** be open to the public Internet.

Architecture diagram

Create an architecture diagram for your solution. The following resources can be used as a reference:

- [AWS Reference Architecture Diagrams](#)
- [GCP Cloud Reference Architectures and Diagrams](#)
- [Azure Architectures](#)

Tools for diagrams:

- <https://draw.io>
- <https://excalidraw.com/>