DEVOPS  
  
In this project, I was tasked with setting up a comprehensive DevOps pipeline for a cloud-based application, encompassing server management, Docker, container orchestration, CI/CD, and monitoring. The following is an overview of the approach and tools used to achieve the project requirements.

**Server Management**

The first procedure was to allocate a Virtual Machine (VM) from Google Cloud through the application of Terraform. I used description to define the VM instance in a Terraform configuration file and deliberately chose e2-standard as the machine type instead of the described e2-medium. The e2-standard was opted for because it provided better performance than its colleague, the e2-medium, which was slightly weaker for the given project. Ubuntu 20. The choice of the operating system was made and the option of 04 LTS was chosen. Firewall policies were set to permit the required traffic; HTTP, HTTPS, and Database ports. In particular, the VM was created and provisioned with the help of applying Terraform’s configuration script.   
After that, I provisioned basic databases, which are required for the VM, and message queues by Terraform, and also by Docker. What was done PostgreSQL and RabbitMQ were declared in the Terraform configuration files as Docker containers running the corresponding services. Correct user IDs and permissions were assigned for storing and executing the database and message queue to maintain the confidentiality of data access.

**Docker**

For containerization of the applications, I developed Dockerfiles for Frontend service and Backend service. The frontend application Dockerfile was based on node:14-alpine, the actual application code was pasted into Docker image, the dependencies were installed with npm, the steps of the production build were described. The backend application was created with the help of . NET had Dockerfile with the base image of mcr. microsoft. com/dotnet/aspnet:6. 0. The application code and required files were copied, the dependencies were met or installed and the application was deployed.

**Container Orchestration**

For container orchestration I used Google Kubernetes Engine (GKE) in creating a Kubernetes cluster with the help of Terraform which is an IaC tool. This approach enabled me to describe and control the cluster’s constitution in a clear and repeatable way. The Terraform configuration created a non-autopilot GKE cluster to give me more control over the cluster and its resources. The cluster was set to have a node pool that ranged from 1 to 2 e2-medium nodes, which was quite effective in terms of resource utilization while at the same time would not be too costly. This node pool size was selected to guarantee the cluster had adequate resources to execute the project’s tasks as well as respect the project’s financial plan. Thus, thanks to the use of e2-medium nodes, I was able to save money on the project by choosing Google Cloud’s rather favorable pricing. The above cluster was also set to be public in order to be accessed through the internet. This was made necessary to allow the cluster to interact with other services outside the cluster like load balancers and ingress controllers. But, to secure the cluster I applied standard network policies and standard firewall rules for incoming traffic.

During the setup process I made sure that the cluster’s settings were properly done to allow the best utilization of resources. This included setting up autoscaling policies to scale up and down the number of nodes in the cluster depending on the traffic load, setting up of requests and limits to ensure that the availability of resources is controlled to avoid over subscribing the resources. Thus, the approach towards resource management was proactive, and I was able to avoid wastage and make sure that the cluster was operating with the best efficiency possible. In conclusion, the Kubernetes cluster deployment with the help of GKE through the tool Terraform was seen yielding a solution that was progressive, defendable, and economical for the orchestration of containers. I also used the infrastructure-as-code concept of Terraform to declare and provision the cluster allowing for its infrastructure to always be consistent and correct throughout the lifecycle of the project.

**CI/CD Pipelines**

To overcome this, I wrote build, push, and deployment YAML files in Azure DevOps for CI/CD pipeline. The frontend CI/CD pipeline was about creating Docker image, storing the image at Docker Hub and deploying it at the Kubernetes cluster. It consisted of forming several Kubernetes objects such as Deployment, Service, and Ingress also, Startup, Readiness, Liveness probes. As for secrets, they were utilized to store the configurations of the environments, for example, the connection strings to the database.   
Likewise, the CI/CD pipeline for the backend comprised the stages to build and push the Docker image and to deploy the image to the Kubernetes. Kubernetes Secrets were designed for storing such information, and the deployment was set up to provide the connection between the frontend, backend, and the database/message queue.   
Autoresponder pipeline triggers were configured to be triggered on code change, and the environments were Development and Production. For deployment stages, approval gates were used in order to achieve controlled release.

**Monitoring**

For health and performance of the system, I configured Prometheus, Grafana, and Alertmanager for VM instance and databases. These tools were set in a manner to monitor important parameters like the CPU usage, RAM usage, disk space, and the response rate of databases. There were also some alarms that were created to make the team aware of any critical that may occur so as to respond to them.

**Deliverables**

The project deliverables included Terraform scripts for provisioning the Google Cloud VM, databases, and GKE cluster; YAML files for Azure DevOps pipelines; shell scripts for setting up monitoring and alerting; and Kubernetes manifests for deploying the frontend and backend applications. These components were organized and stored in a git repository, providing a comprehensive and reusable infrastructure setup for future projects.