

CP3061 – DevOps and Microservices

Mini Project

Infrastructure as Code Using Terraform and Kubernetes

Done by:

Gowtham A (2024207036)

M.E CSE

1. Abstract

In the rapidly evolving landscape of cloud computing and DevOps, the ability to automate infrastructure provisioning and application deployment has become a cornerstone of modern software development. This project exemplifies the principles of Infrastructure as Code (IaC) by leveraging Terraform, Kubernetes, and Docker to deploy a scalable, resilient, and fully automated full-stack web application. The system comprises a frontend built with HTML, CSS, and JavaScript, hosted on an Nginx server, and a backend REST API developed using Node.js and Express.js to manage a dynamic product catalog.

By codifying infrastructure requirements into declarative Terraform configurations, this project eliminates manual intervention, ensuring reproducibility across environments. Kubernetes orchestrates containerized workloads, enabling self-healing, load balancing, and horizontal scaling, while Docker guarantees environment consistency by packaging applications into portable containers. The integration of these tools demonstrates a seamless transition from development to production, mimicking real-world DevOps workflows.

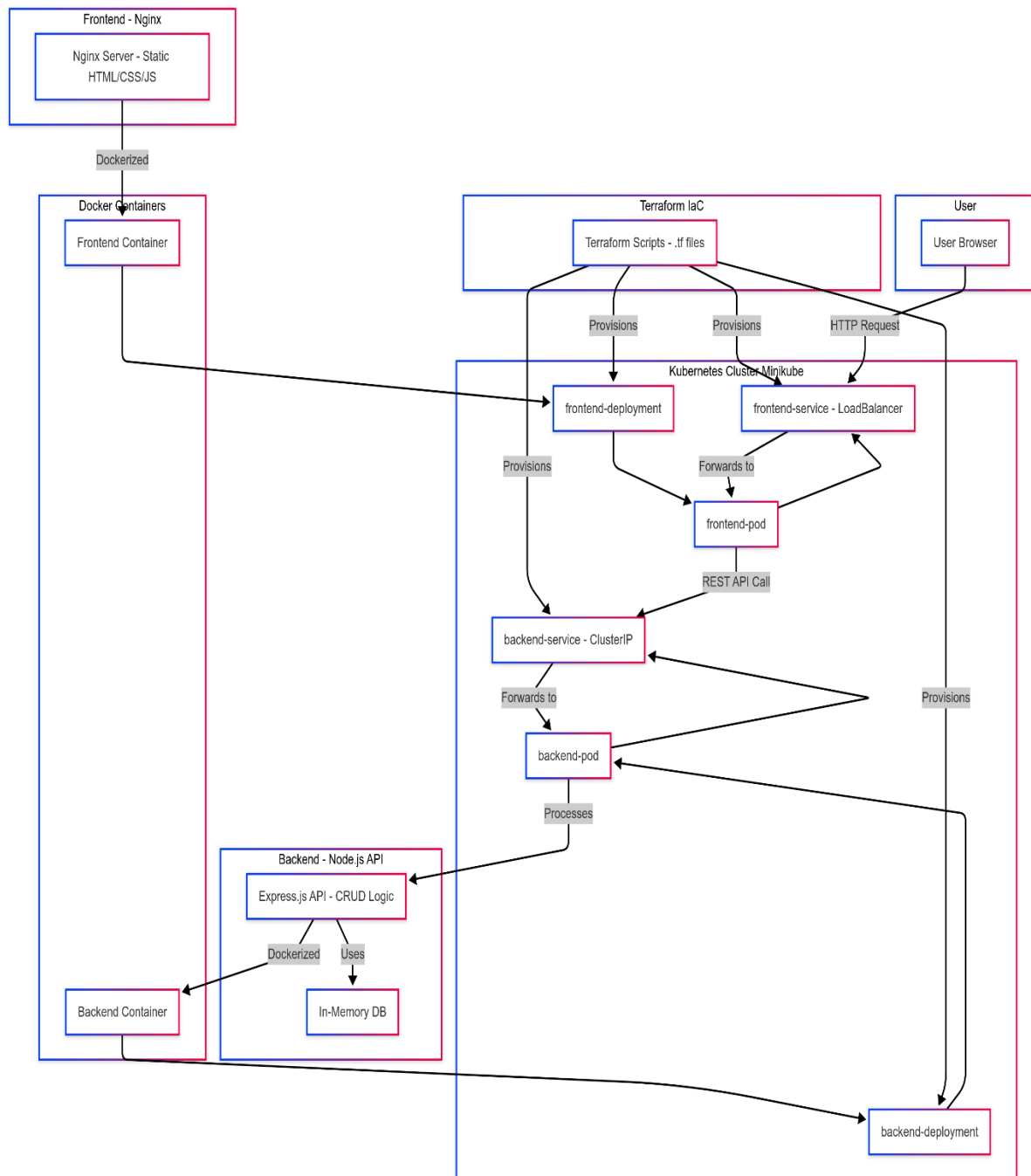
This work not only highlights the technical implementation of IaC but also underscores its broader implications: reducing human error, accelerating deployment cycles, and fostering collaboration between development and operations teams. By bridging the gap between theoretical concepts and practical application, the project serves as a pedagogical tool for understanding cloud-native architectures and the transformative potential of automation in modern software engineering.

2. Introduction

The rise of cloud computing and microservices has revolutionized how applications are developed, deployed, and scaled. However, traditional infrastructure management—reliant on manual configuration and ad-hoc scripting—remains fraught with challenges, including inconsistency across environments, scalability bottlenecks, and deployment failures. These inefficiencies are exacerbated in complex systems where even minor misconfigurations can lead to downtime or security vulnerabilities.

Infrastructure as Code (IaC) emerges as a paradigm shift, enabling developers to define and manage infrastructure using machine-readable configuration files. This approach ensures version control, auditability, and repeatability, aligning infrastructure management with software development best practices. This project embodies these principles by automating the deployment of a full-stack application using Terraform for infrastructure provisioning, Kubernetes for orchestration, and Docker for containerization.

3. Architecture Diagram



4. Requirements and Supported Systems

4.1 Hardware Requirements

- CPU: 2+ cores (supports virtualization).
- RAM: 4+ GB.

- Storage: 20+ GB.

4.2 Software Requirements

- Terraform - Version 1.5+
- Kubernetes Cluster- Version 1.27+
- Docker -Version 24.0+
- Node.js -Version 18.0+
- Minikube -Version 1.35+

4.3 Supported Systems

- Operating Systems: Windows 10+, Linux (Ubuntu 22.04+), macOS Ventura+.
- Kubernetes Distributions: Minikube, Docker Desktop, AWS EKS, Google GKE.

5. Tools Used

5.1 Software Tools

Tool	Role
Terraform	IaC for Kubernetes resource provisioning.
Kubernetes	Orchestrates containerized workloads.
Docker	Builds and manages application containers.
Node.js	Backend API runtime.
Nginx	Frontend web server.
Minikube	Local Kubernetes cluster

5.2 Hardware

- Local Machine: Intel i5/Ryzen 5 or equivalent.
- Cloud VM (Optional): AWS EC2, Google Compute Engine.
- Storage (SSD) for optimal Minikube performance.

6. Implementation

Step 1: Set Up Kubernetes Cluster

1. Start Minikube

- `minikube start --driver=docker`

2. Verify Cluster

- `kubectl get nodes`

Step 2: Build and deploy applications

1. Build Docker Images

- Use Minikube's Docker daemon

```
minikube docker-env | Invoke-Expression
```

- Frontend

```
docker build -t frontend:latest ./frontend
```

- Backend

```
docker build -t backend:latest ./backend
```

2. Apply Terraform Configurations

```
cd terraform
```

```
terraform init
```

```
terraform apply -auto-approve
```

Step 3: Configure Networking

1. Verify Services

```
kubectl get svc
```

Ensure frontend-service (LoadBalancer) and backend-service (ClusterIP) are running

2. Access Frontend

```
minikube service frontend-service --url
```

Step 4: Validate Deployment

1. Check Pods:

```
kubectl get pods -l app=frontend
```

```
kubectl get pods -l app=backend
```

2. Test API Connectivity:

```
# Forward backend port
```

```
kubectl port-forward svc/backend-service 3000:3000
```

7. Input and Output

7.1 Input

- Terraform Files:
 - main.tf (Kubernetes deployments/services definitions).
- Dockerfiles:
 - frontend/Dockerfile, backend/Dockerfile.
- Application Code:
 - Frontend: index.html, script.js, style.css.
 - Backend: index.js, package.json.

7.2 Output

- **Deployed Application:**

```
minikube service frontend-service --url
```

Output: <http://127.0.0.1:30734>

- **Backend API Test:**

```
curl http://localhost:3000/products
```

Output: [{"id":1,"name":"Laptop","price":999}]

- **Self-Healing Test:**

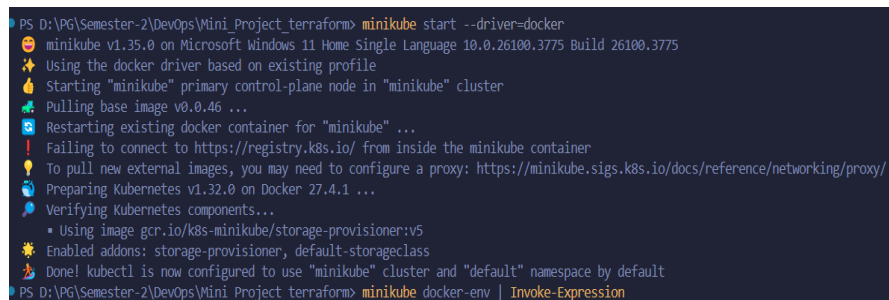
```
kubectl delete pod frontend-75849fb77b-rgkt9
```

```
kubectl get pods -watch
```

Deleting the frontend pod to see whether it is recreating the pod in case of a system crash or network error.

Screenshots:

- Starting and initializing the created MiniKube cluster:



```
PS D:\PG\Semester-2\DevOps\Mini_Project terraform> minikube start --driver=docker
minikube v1.35.0 on Microsoft Windows 11 Home Single Language 10.0.26100.3775 Build 26100.3775
Using the docker driver based on existing profile
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.46 ...
Restarting existing docker container for "minikube" ...
! Failing to connect to https://registry.k8s.io/ from inside the minikube container
To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
PS D:\PG\Semester-2\DevOps\Mini_Project terraform> minikube docker-env | Invoke-Expression
```


- Updation of recent code into the Dockerfile

```
ms (Ctrl+Shift+M) ter-2\DevOps\Mini_Project_terraform\frontend> docker build -t frontend:latest .
[+] Building 1.1s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 143B
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/3] FROM docker.io/library/nginx:alpine@sha256:65645c7bb6a0661892a8b03b89d0743208a18dd2f3f17a54ef4b76fb8e2f2a10
=> [internal] load build context
=> => transferring context: 149B
=> CACHED [2/3] COPY . /usr/share/nginx/html
=> CACHED [3/3] COPY nginx.conf /etc/nginx/conf.d/default.conf
=> exporting to image
=> => exporting layers
=> => writing image sha256:253cc12b95b813f6066e1ead6e66146fe3c4a2ff25d92cf056bcf39f121fc264
=> => naming to docker.io/library/frontend:latest

View build details: docker-desktop://dashboard/build/default/default/liht3b69r2dqdaovm7h1z56p3
PS D:\PG\Semester-2\DevOps\Mini_Project_terraform\frontend> cd ../backend
PS D:\PG\Semester-2\DevOps\Mini_Project_terraform\backend> docker build -t backend:latest .
[+] Building 2.3s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 159B
=> [internal] load metadata for docker.io/library/node:18-alpine
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e
=> [internal] load build context
=> => transferring context: 58.31kB
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY package*.json ./
=> CACHED [4/5] RUN npm install
=> CACHED [5/5] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:f4fa6f17af0ec5d1c6c112f29afec4f965ee1ccdd44a7ef6a908501b159c22de
=> => naming to docker.io/library/backend:latest

View build details: docker-desktop://dashboard/build/default/default/c7e6x4jm242dfbtcig9ht933a
```

- Terraform Apply

```
PS D:\PG\Semester-2\DevOps\Mini_Project_terraform\backend> cd ../terraform\
PS D:\PG\Semester-2\DevOps\Mini_Project_terraform\terraform> terraform apply
kubernetes_service.backend: Refreshing state... [id=default/backend-service]
kubernetes_service.frontend: Refreshing state... [id=default/frontend-service]
kubernetes_deployment.backend: Refreshing state... [id=default/backend]
kubernetes_deployment.frontend: Refreshing state... [id=default/frontend]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
PS D:\PG\Semester-2\DevOps\Mini_Project_terraform\terraform>
```

- Starting MiniKube tunnel

```
PS D:\PG\Semester-2\DevOps\Mini_Project_terraform> minikube tunnel
✓ Tunnel successfully started

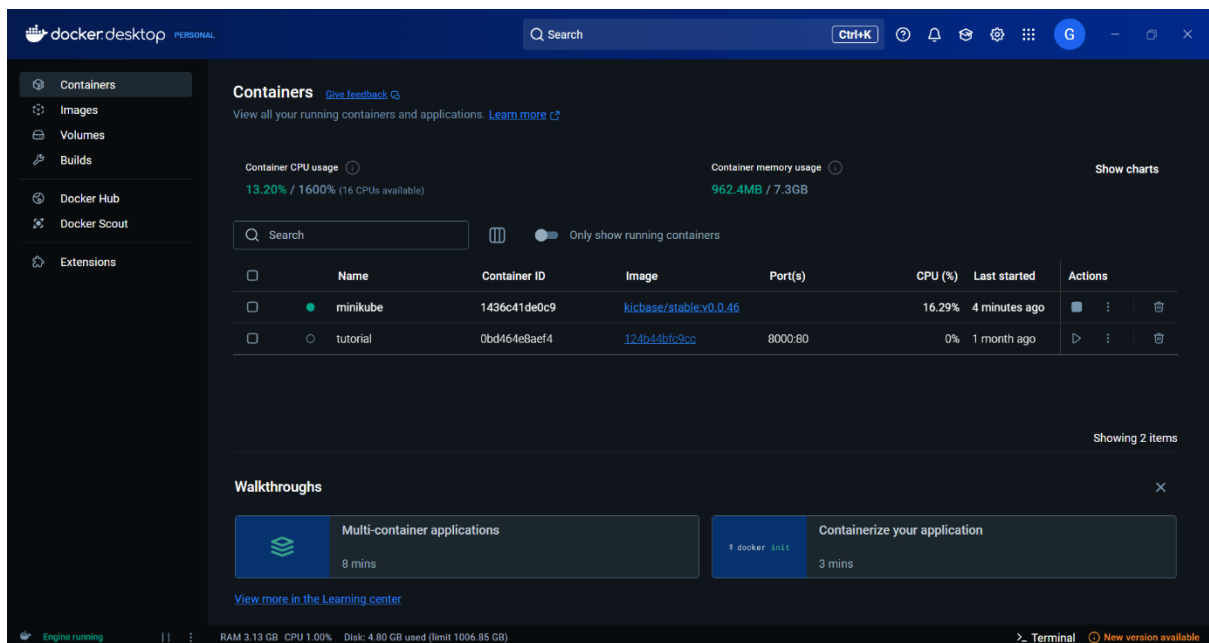
✦ NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be accessible ...

! Access to ports below 1024 may fail on Windows with OpenSSH clients older than v8.1. For more information, see: https://minikube.sigs.k8s.io/docs/handbook/accessing/#access-to-ports-1024-on-windows-requires-root-permission
✦ Starting tunnel for service frontend-service.
```

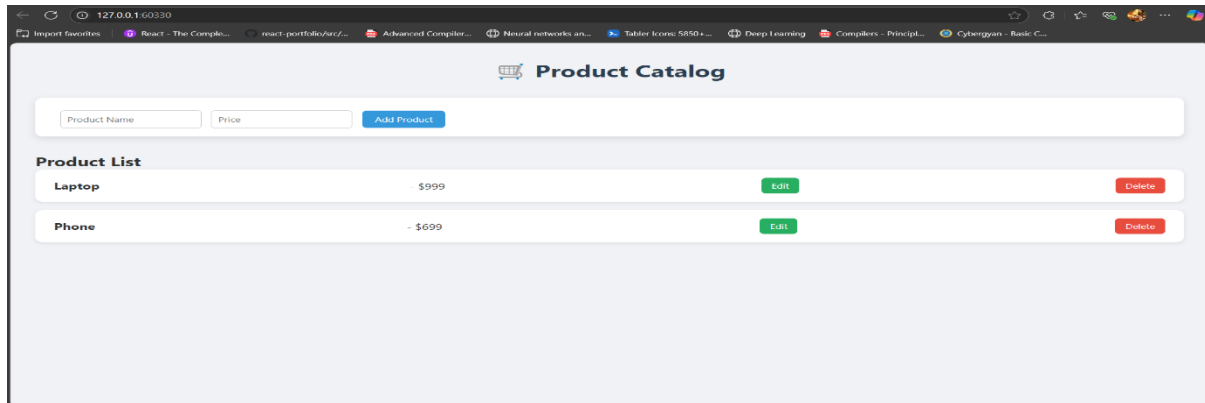
- The web app service

```
PS D:\PG\Semester-2\DevOps\Mini_Project_terraform\terraform> minikube service frontend-service --url http://127.0.0.1:60330
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

- Minikube instance running on the Docker Desktop



- Web App – Product Catalog -Performs CRUD



8. Applications

1. Automated Deployment of Scalable Applications

- Use case: Companies can deploy applications across multiple environments (dev, staging, production) with just a few commands.
- Example: E-commerce platforms can auto-scale their services during high traffic (like festive sales) without manual intervention.

2. Multi-Cloud and Hybrid Cloud Management

- Use case: Organizations using AWS, Azure, or GCP can manage resources from all providers using a single Terraform codebase.
- Example: A company can deploy its frontend on Azure and backend on AWS, all orchestrated via Kubernetes.

3. Disaster Recovery and Infrastructure Replication

- Use case: Quickly replicate infrastructure in a different region or cloud in case of failure or for backup.
- Example: Banking and financial services can restore full infrastructure in a different region to maintain business continuity.

4. Resource Efficiency and Cost Optimization

- Use case: Kubernetes autoscaling ensures resources are used only when needed, and Terraform helps destroy unused resources to cut cloud bills.
- Example: Streaming platforms scale up during peak hours and scale down during off-hours to save cloud costs.

9. References

1. HashiCorp. (2023). *Terraform Documentation*.
<https://developer.hashicorp.com/terraform/docs>
2. Kubernetes. (2023). *Kubernetes Documentation*.
<https://kubernetes.io/docs>
3. Docker. (2023). *Docker Documentation*.
<https://docs.docker.com>
4. Minikube. (2023). *Getting Started*.
<https://minikube.sigs.k8s.io/docs/start/>