Project Documentation

Title: Calculator Application Deployment on Azure using Terraform and Docker

Submitted by: Priyanshu Gupta

Student ID: 1576753

Course: Cloud Computing

Date: 14/07/2005

1. Introduction

The purpose of this project is to design, containerize, and deploy a simple calculator web application using cloud technologies.
This demonstrates modern DevOps practices including Infrastructure as Code (IaC) with Terraform, containerization with Docker, and deployment to **Microsoft Azure**.

The calculator performs basic arithmetic operations - addition, subtraction, multiplication, and division through a web interface built using **Python's Flask framework**.

2. Objectives

- Build a simple, functional calculator web app using Flask.
- Containerize the app using Docker.
- Deploy the Docker container to a Virtual Machine (VM) on Azure using Terraform.
- Automate infrastructure provisioning using Terraform scripts.
- Apply basic cloud security practices (e.g., managing secrets securely).

3. Technologies Used

Tool/Service:	Purpose
Python (Flask):	Web application backend
Docker:	Containerization of the application
Terraform:	Infrastructure as Code
Microsoft Azure:	Cloud hosting (VM deployment)
Jupyter Lab:	Code editor

4. Application Overview

The application has a user interface to input two numbers and select an arithmetic operation.
It sends the input to the Flask backend, which processes and returns the result.
The app is containerized using Docker and deployed to an **Ubuntu VM** in **Azure**.

5. Project Structure

```
calculator/
│
├── app.py
├── templates/
│   └── index.html
├── requirements.txt
├── Dockerfile
├── terraform/
│   ├── main.tf
│   ├── variables.tf
│   ├── terraform.tfvars
│   └── deploy.sh
└── docker-deploy.sh
```

6. Implementation Steps

6.1 Application Development

A simple Flask app was created in app.py.
HTML template (index.html) was designed for user input.
Required dependencies listed in requirements.txt.

6.2 Docker Containerization

Created a Dockerfile to define the environment and launch process.
The image was built and tested locally using Docker.

6.3 Terraform Configuration

Azure provider configured in main.tf.
Resources created include: Resource Group, Virtual Network, Subnet, Public IP, Network Interface, Ubuntu Virtual Machine.

6.4 Shell Scripts

deploy.sh: Automates Terraform initialization, planning, and deployment.
docker-deploy.sh: Used to build and run the Docker container locally for testing.

6.5 Cloud Deployment

VM is configured to install Docker at launch using a custom startup script.
The Docker image is pulled from **DockerHub** and the container is run on port 5000.

7. Scalability

Although this project currently deploys just a single virtual machine running a Docker container, it can be easily scaled based on future requirements. Terraform allows us to adjust the number of resources dynamically using the **count** parameter. For example, if we want to run three instances of the same VM, we can simply add **count = 3** in the VM resource block. This makes the infrastructure flexible and ready to grow when needed, aligning with the "rapid elasticity" principle defined by NIST. Additionally, if container-level scaling is required in the future, tools like Docker Compose or Kubernetes could be integrated to run multiple container replicas efficiently.

8. Security Considerations

Important: For production or real-world projects:
- Passwords & Secrets should not be hardcoded in Terraform or code.
- Use Terraform Variables or Azure Key Vault for secure management of sensitive data.
- Example:
  - Use **terraform.tfvars** to store values.
  - Mark variables as sensitive in variables.tf.

9. Output

After successful deployment, Terraform outputs the public IP address of the Azure VM. The calculator application becomes accessible through a web browser by navigating to **http://<public-ip>**.

10. Cost Management

A **Standard_B1s** VM was used — eligible under Azure Free Tier (with limitations).
Once the deployment is done, you can destroy resources using:
terraform destroy
Also verify by running:
az group list --output table

11. Conclusion

This project showcases a complete cloud deployment pipeline using Docker and Terraform. It provides hands-on experience with:
- Infrastructure automation
- Application containerization
- Public cloud platform (Azure)

- Security and cost-conscious deployment

This approach follows modern DevOps principles and introduces real-world cloud workflows suitable for scalable applications.

12. Future Improvements

- Add CI/CD using GitHub Actions or Azure DevOps.
- Use Azure Container Instances or Azure App Service instead of a VM.
- Implement TLS/HTTPS using Azure certificates.
- Add unit testing for backend logic.