Software Project Management

Semester Project Report

'*DevOps Web App.*'

By:

Wania Shafqat

Muhammad Hassan Ahmed Khan

BS Computer Science 01

Institute of Space Technology, Islamabad

JANUARY 2024

Table of Contents

DevOps Operations

1. Introduction

This comprehensive documentation details the architecture and operational workflow of the DevOps Web App project. It covers the utilization of GitHub Actions for CI/CD, AWS services provisioned via Terraform, and the integration of monitoring and security tools to streamline the application development lifecycle.

2. Overview

The DevOps Web App is an educational tool designed to showcase various DevOps concepts. It's an interactive platform where users can explore and engage with different aspects of DevOps practices.

3. Environment Setup and Requirements

Prerequisites

- Python *3.12.1*
- Git
- MySQL

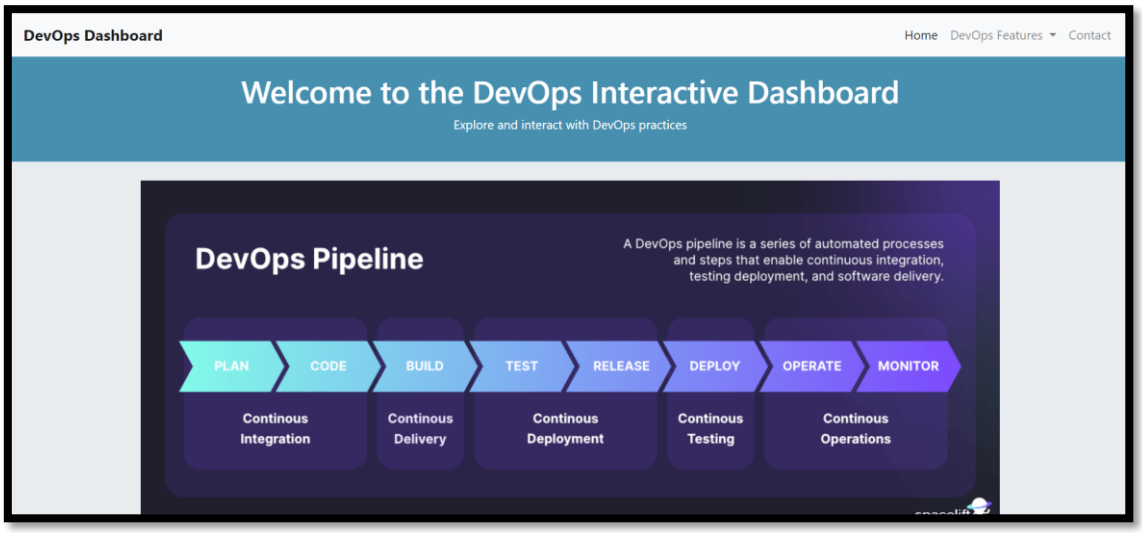
Installation and Configuration

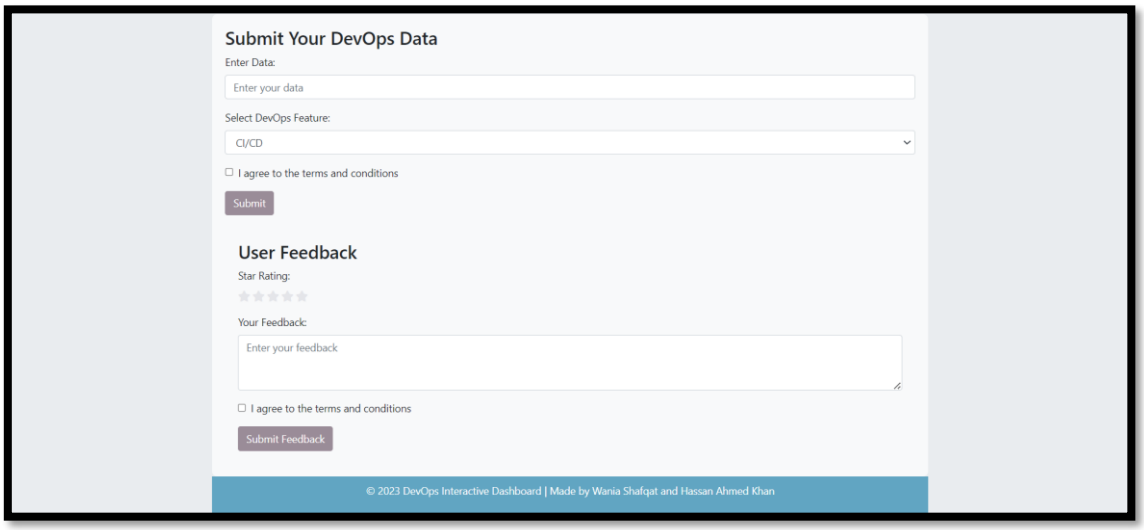
- Clone the repository: git clone https://github.com/waniashafqat/DevOps-Web-Application.git
- Change directory: ***cd DevOps-Web-Application***
- Install dependencies listed in ***requirements.txt*** using pip.
- Configure the environment variables for database connectivity and other services.
- Run the application: ***python app.py***

4. Project Architecture

Front-end Design

- **Technologies Used:** HTML, CSS, and JavaScript.
- **Functionality:** The front end provides an interactive user interface. It's designed to be intuitive, allowing users to navigate easily and submit data related to DevOps practices.
- **Responsiveness:** The design is responsive, ensuring that the application is accessible across various devices and screen sizes.





Back-end Development

- **Framework:** Flask, a lightweight WSGI web application framework in Python.
- **Functionality:** Handles HTTP requests, processes data, and manages interactions with the MySQL database.

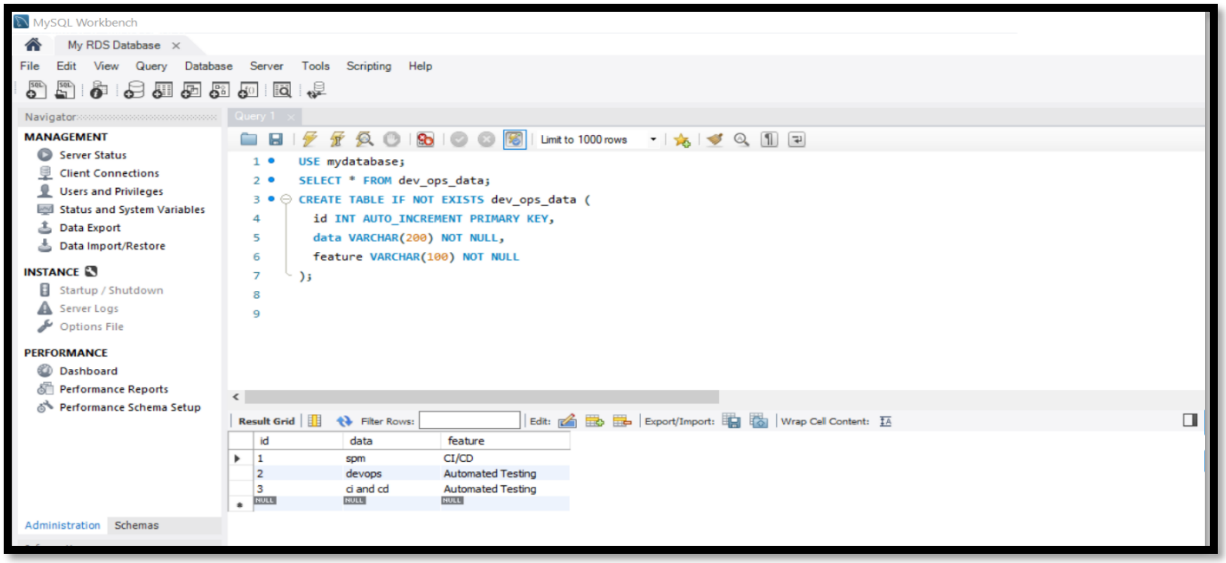
- **Modularity and Extensibility:** Flask's modular design allows developers to implement functionalities using various extensions, making it adaptable to specific project requirements. It supports integration with libraries for authentication, data validation, and more.
- **Routing and Views:** Flask uses routes to map URLs to functions, making it straightforward to handle different HTTP requests. Views render templates or return data in various formats (JSON, HTML, etc.) based on the request type.

API Development

- **RESTful API:** Flask facilitates the creation of RESTful APIs, allowing the front end to communicate with the back end through well-defined endpoints. It handles requests for data submission, retrieval, and other operations. These endpoints enable data exchange using HTTP methods (GET, POST, PUT, DELETE).

Database

- **Relational Data Model:** MySQL follows a relational database model, organizing data into tables with defined relationships. It supports the creation of complex queries and transactions for managing structured data.
- **Data Integrity and ACID Properties:** MySQL ensures data integrity by supporting ACID properties (Atomicity, Consistency, Isolation, Durability), guaranteeing reliable transactions, and maintaining database consistency.
- **Scalability and Performance:** MySQL's ability to handle large volumes of data and its optimization features (like indexing) contribute to its scalability and performance, making it suitable for various applications**.**



Communication Flow

- **Client-Server Interaction:**

  [1] The front end interacts with the back-end server via HTTP requests, typically using JSON or other data formats.

  [2] The server processes these requests, performs necessary operations (such as retrieving or modifying data), and sends back appropriate responses.

- **Server-Database Interaction:**

  [1] The back-end server communicates with the MySQL database using SQL queries, executing commands to retrieve, update, insert, or delete data as needed.

5. CI/CD Pipeline

Continuous Integration and Deployment

- **GitHub Actions:** Automated workflows are set up in GitHub Actions, which handle continuous integration and deployment tasks. These include automated building, testing, and deploying the application upon every commit to the repository.
- **Automated Testing:** Integral to the pipeline are automated unit and integration tests, ensuring application reliability and functionality before deployment.
- **Deployment:** Upon successful completion of tests, the application is automatically deployed to the Webhost server.

Deployment Strategy

Hosting Platform: Webhost

- **Platform Choice:** Due to compatibility issues with AWS Elastic Beanstalk and Python 3.12.1, the decision was made to utilize Webhost for hosting the application.
- **Benefits:** Webhost offers a simplified hosting solution with easy setup, making it ideal for hosting our Python-based web application.

Environment Management

- **Staging and Production:** The project includes separate environments for staging and production to ensure a controlled deployment pipeline. The staging environment is used for testing and validating changes before they are pushed to the live production environment.

Task Name	Description	Tools Used	Key Considerations
Code Commit	Developers commit code to the repository	GitHub	Ensure commit guidelines
Automated Testing	Run unit and integration tests	GitHub Actions	Test coverage, reliability
Build	Build application artifacts	GitHub Actions	Build consistency
Deploy to Staging	Deploy to a staging environment	Terraform, AWS	Validate changes

6. Infrastructure Configuration

Terraform: Role in Infrastructure Management

- **Infrastructure as Code:** Terraform is utilized for defining and managing the AWS infrastructure required for the project. This approach ensures consistency and version control for infrastructure provisioning.
- **Resource Provisioning:** Terraform scripts automate the creation and management of AWS resources such as RDS for the database, ensuring efficient and error-free deployment.

Key Features Utilized

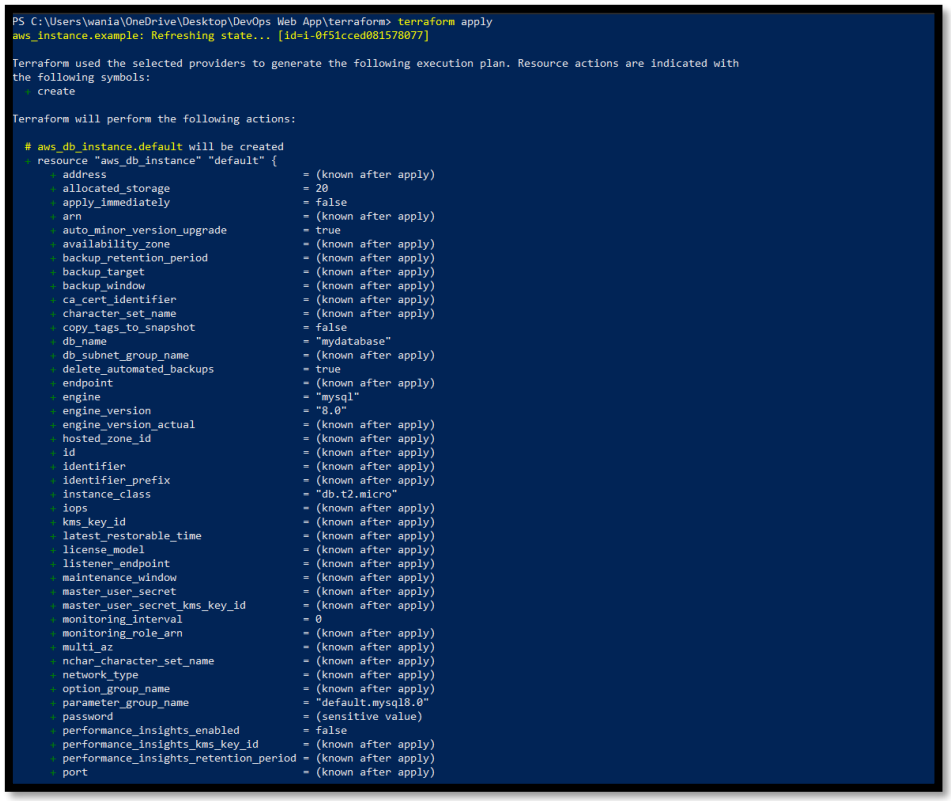
- **Modularity**: The project leverages Terraform modules for code organization and reusability, enhancing manageability and scalability.
- **State Management**: Terraform's state management capabilities enable tracking and maintenance of the AWS resources, crucial for applying updates and avoiding conflicts.
- **Resource Dependencies**: Understanding and managing resource dependencies is a key feature of Terraform, ensuring smooth provisioning and management of AWS resources.

Task Name	Description	Tools Used	Key Considerations
Define Infrastructure	Define cloud resources as code	Terraform	Maintainability
Infrastructure Provisioning	Provisioning of defined infrastructure	Terraform, AWS	Idempotency of scripts
Version Control IaC	Version control for IaC scripts	GitHub	Change tracking
Update & Maintenance	Regular updates and maintenance	Terraform	Keeping up with changes

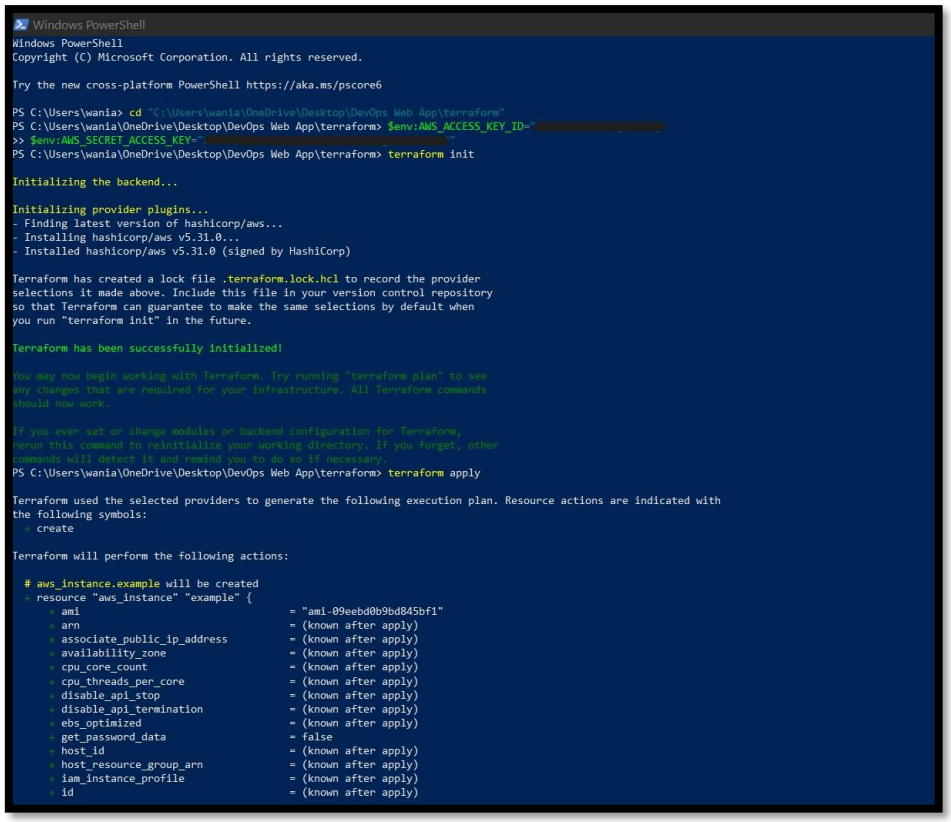
Implementation

- **Terraform Init:** The process starts with initializing the Terraform configuration to prepare for further actions.
- **Terraform Plan:** Before applying changes, the Terraform Plan assesses the necessary modifications required to achieve the desired infrastructure state.

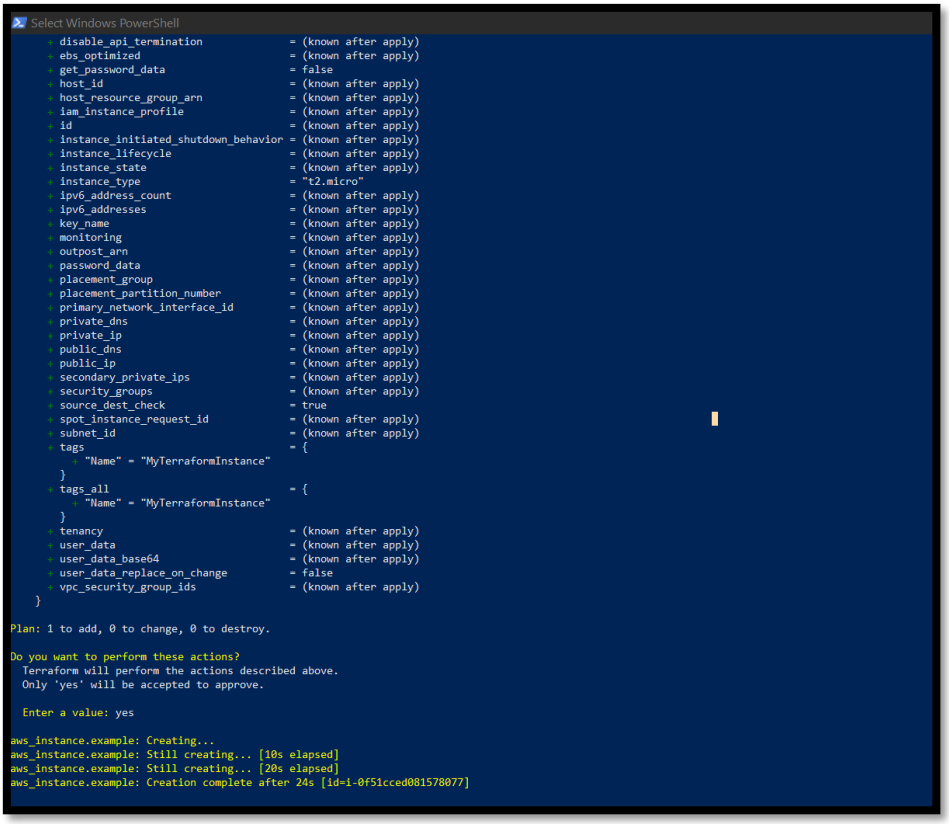
- **Terraform Apply:** This crucial step involves applying the changes to achieve the desired state, creating or modifying AWS resources as per the defined Terraform scripts.

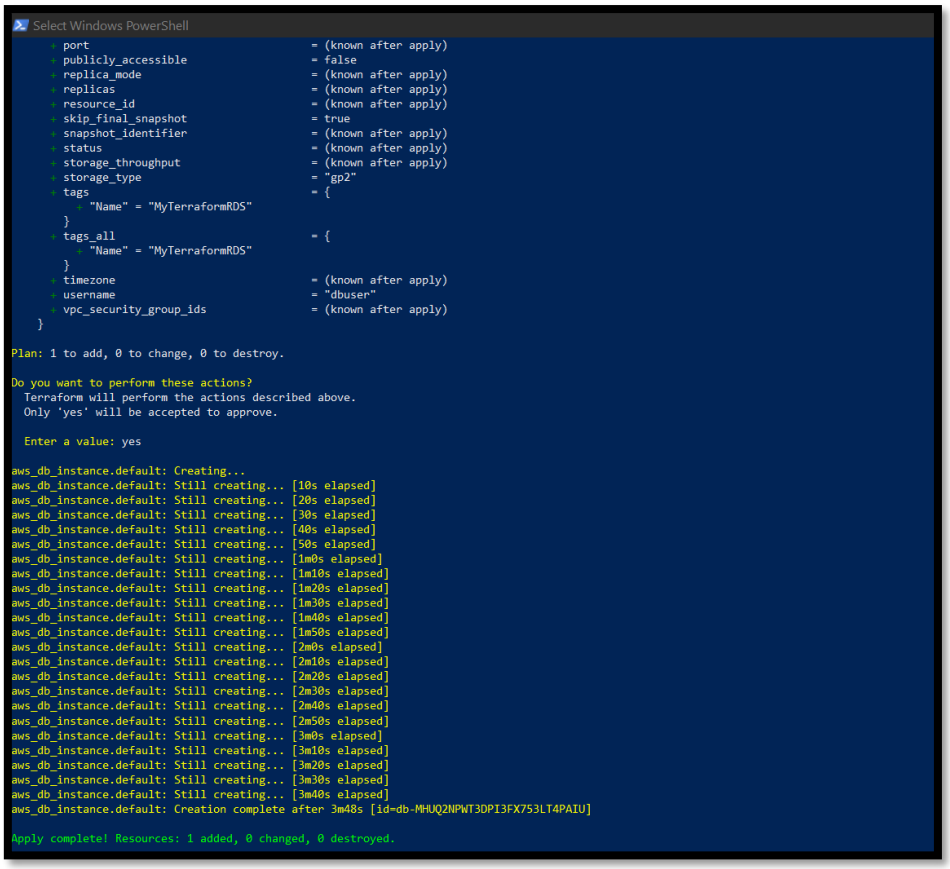


```
PS C:\Users\wania\OneDrive\Desktop\DevOps Web App\terraform> terraform apply
aws_instance.example: Refreshing state... [id=i-0f51cced081578077]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with
the following symbols:
  + create

Terraform will perform the following actions:

  # aws_db_instance.default will be created
  + resource "aws_db_instance" "default" {
      + address                               = (known after apply)
      + allocated_storage                     = 20
      + apply_immediately                     = false
      + arn                                   = (known after apply)
      + auto_minor_version_upgrade            = true
      + availability_zone                     = (known after apply)
      + backup_retention_period               = (known after apply)
      + backup_target                         = (known after apply)
      + backup_window                         = (known after apply)
      + ca_cert_identifier                    = (known after apply)
      + character_set_name                    = (known after apply)
      + copy_tags_to_snapshot                 = false
      + db_name                               = "mydatabase"
      + db_subnet_group_name                  = (known after apply)
      + delete_automated_backups              = true
      + endpoint                              = (known after apply)
      + engine                                = "mysql"
      + engine_version                        = "8.0"
      + engine_version_actual                 = (known after apply)
      + hosted_zone_id                        = (known after apply)
      + id                                    = (known after apply)
      + identifier                            = (known after apply)
      + identifier_prefix                     = (known after apply)
      + instance_class                        = "db.t2.micro"
      + iops                                  = (known after apply)
      + kms_key_id                            = (known after apply)
      + latest_restorable_time                = (known after apply)
      + license_model                         = (known after apply)
      + listener_endpoint                     = (known after apply)
      + maintenance_window                    = (known after apply)
      + master_user_secret                    = (known after apply)
      + master_user_secret_kms_key_id         = (known after apply)
      + monitoring_interval                   = 0
      + monitoring_role_arn                   = (known after apply)
      + multi_az                              = (known after apply)
      + nchar_character_set_name              = (known after apply)
      + network_type                          = (known after apply)
      + option_group_name                     = (known after apply)
      + parameter_group_name                  = "default.mysql8.0"
      + password                              = (sensitive value)
      + performance_insights_enabled          = false
      + performance_insights_kms_key_id       = (known after apply)
      + performance_insights_retention_period = (known after apply)
      + port                                  = (known after apply)
```



```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\wania> cd "C:\Users\wania\OneDrive\Desktop\DevOps Web App\terraform"
PS C:\Users\wania\OneDrive\Desktop\DevOps Web App\terraform> $env:AWS_ACCESS_KEY_ID="
>> $env:AWS_SECRET_ACCESS_KEY="
PS C:\Users\wania\OneDrive\Desktop\DevOps Web App\terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.31.0...
- Installed hashicorp/aws v5.31.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\wania\OneDrive\Desktop\DevOps Web App\terraform> terraform apply

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with
the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.example will be created
  + resource "aws_instance" "example" {
      + ami                                  = "ami-09eebd0b9bd845bf1"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
```


Select Windows PowerShell
+ disable_api_termination = (known after apply)
+ ebs_optimized = (known after apply)
+ get_password_data = false
+ host_id = (known after apply)
+ host_resource_group_arn = (known after apply)
+ iam_instance_profile = (known after apply)
+ id = (known after apply)
+ instance_initiated_shutdown_behavior = (known after apply)
+ instance_lifecycle = (known after apply)
+ instance_state = (known after apply)
+ instance_type = "t2.micro"
+ ipv6_address_count = (known after apply)
+ ipv6_addresses = (known after apply)
+ key_name = (known after apply)
+ monitoring = (known after apply)
+ outpost_arn = (known after apply)
+ password_data = (known after apply)
+ placement_group = (known after apply)
+ placement_partition_number = (known after apply)
+ primary_network_interface_id = (known after apply)
+ private_dns = (known after apply)
+ private_ip = (known after apply)
+ public_dns = (known after apply)
+ public_ip = (known after apply)
+ secondary_private_ips = (known after apply)
+ security_groups = (known after apply)
+ source_dest_check = true
+ spot_instance_request_id = (known after apply)
+ subnet_id = (known after apply)
+ tags = {
+ "Name" = "MyTerraformInstance"
}
+ tags_all = {
+ "Name" = "MyTerraformInstance"
}
+ tenancy = (known after apply)
+ user_data = (known after apply)
+ user_data_base64 = (known after apply)
+ user_data_replace_on_change = false
+ vpc_security_group_ids = (known after apply)
}
Plan: 1 to add, 0 to change, 0 to destroy.
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
Enter a value: yes
aws_instance.example: Creating...
aws_instance.example: Still creating... [10s elapsed]
aws_instance.example: Still creating... [20s elapsed]
aws_instance.example: Creation complete after 24s [id=i-0f51cced081578077]


Select Windows PowerShell
+ port = (known after apply)
+ publicly_accessible = false
+ replica_mode = (known after apply)
+ replicas = (known after apply)
+ resource_id = (known after apply)
+ skip_final_snapshot = true
+ snapshot_identifier = (known after apply)
+ status = (known after apply)
+ storage_throughput = (known after apply)
+ storage_type = "gp2"
+ tags = {
+ "Name" = "MyTerraformRDS"
}
+ tags_all = {
+ "Name" = "MyTerraformRDS"
}
+ timezone = (known after apply)
+ username = "dbuser"
+ vpc_security_group_ids = (known after apply)
}
Plan: 1 to add, 0 to change, 0 to destroy.
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
Enter a value: yes
aws_db_instance.default: Creating...
aws_db_instance.default: Still creating... [10s elapsed]
aws_db_instance.default: Still creating... [20s elapsed]
aws_db_instance.default: Still creating... [30s elapsed]
aws_db_instance.default: Still creating... [40s elapsed]
aws_db_instance.default: Still creating... [50s elapsed]
aws_db_instance.default: Still creating... [1m0s elapsed]
aws_db_instance.default: Still creating... [1m10s elapsed]
aws_db_instance.default: Still creating... [1m20s elapsed]
aws_db_instance.default: Still creating... [1m30s elapsed]
aws_db_instance.default: Still creating... [1m40s elapsed]
aws_db_instance.default: Still creating... [1m50s elapsed]
aws_db_instance.default: Still creating... [2m0s elapsed]
aws_db_instance.default: Still creating... [2m10s elapsed]
aws_db_instance.default: Still creating... [2m20s elapsed]
aws_db_instance.default: Still creating... [2m30s elapsed]
aws_db_instance.default: Still creating... [2m40s elapsed]
aws_db_instance.default: Still creating... [2m50s elapsed]
aws_db_instance.default: Still creating... [3m0s elapsed]
aws_db_instance.default: Still creating... [3m10s elapsed]
aws_db_instance.default: Still creating... [3m20s elapsed]
aws_db_instance.default: Still creating... [3m30s elapsed]
aws_db_instance.default: Still creating... [3m40s elapsed]
aws_db_instance.default: Creation complete after 3m48s [id=db-MHUQ2NPWT3DPI3FX753LT4PAIU]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

7. Monitoring and Logging

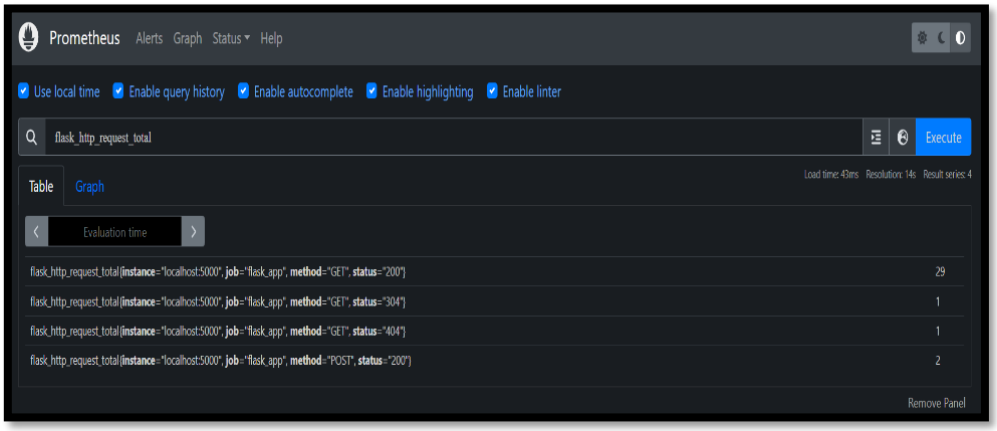
Prometheus: Implementation

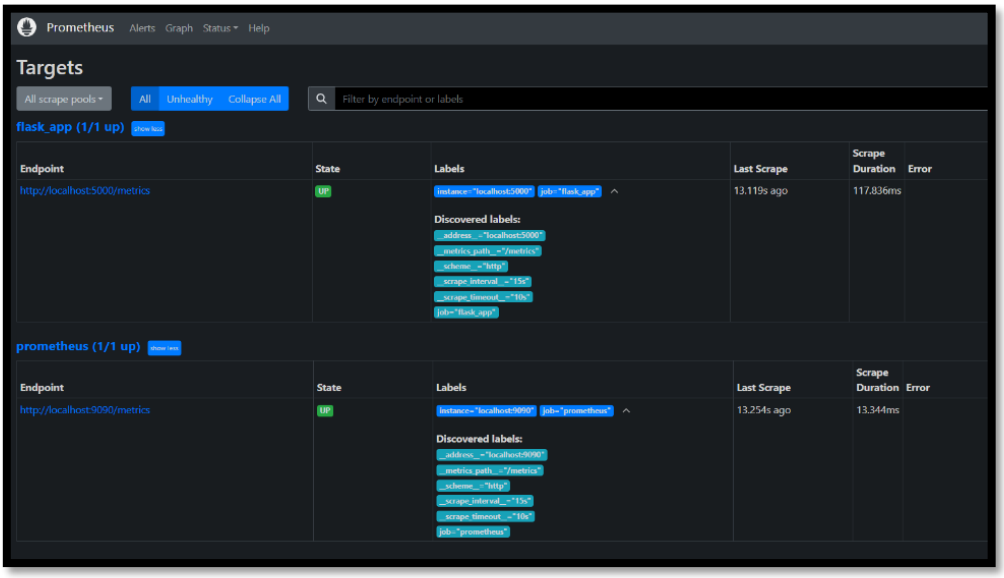
- **Real-time Monitoring:** The application integrates Prometheus for continuous monitoring. This tool collects and stores metrics, providing real-time insights into the application's performance and health.
- **Metrics Collection:** Prometheus gathers crucial metrics such as CPU usage, memory utilization, response times, and more. This data is crucial for proactive monitoring and performance tuning.
- **Alerting System:** Prometheus's alerting framework sends notifications based on defined alert rules, ensuring immediate attention to potential issues.

Task Name	Description	Tools Used	Key Considerations
Set Up Monitoring	Configure monitoring tools	Prometheus	Coverage of key metrics
Log Aggregation	Aggregate logs for analysis	Prometheus	Log retention policy
Performance Monitoring	Monitor app performance	Prometheus	Alerts for anomalies
Health Checks	Regular services health checks	Prometheus	Frequency of checks

Logging Strategy

- **Flask Logging:** The backend, built with Flask, employs its built-in logging mechanism. This allows for capturing and recording critical information about the application's operation, including errors and warnings.
- **Log Aggregation:** For efficient log management, the application incorporates log aggregation tools, ensuring that logs from various components are centralized and easily accessible for analysis.





8. Security Scanning

SonarQube: Integration into CI/CD Pipeline

- **Automated Scanning:** SonarQube is integrated into the GitHub Actions pipeline. This ensures automated scanning of the codebase for vulnerabilities and code quality issues with each commit.
- **Continuous Feedback:** SonarQube provides continuous feedback on code security, highlighting vulnerabilities and suggesting improvements to enhance code quality and security.

SonarQube: Key Benefits

- **Early Detection of Vulnerabilities:** SonarQube aids in identifying security weaknesses early in the development process, reducing the risk of security breaches post-deployment.
- **Code Quality Assurance:** Apart from security, SonarQube also focuses on maintaining high standards of code quality, enforcing best practices, and ensuring maintainability.

9. Troubleshooting Steps

Common Issues and Resolutions

- **Database Connectivity Issues:** Steps to diagnose include checking network configurations, validating credentials, and ensuring the database service is running.
- **Application Deployment Errors:** In case of deployment failures, reviewing deployment logs, checking for compatibility issues, and verifying configuration settings are primary steps.
- **Performance Bottlenecks:** When facing performance issues, analyzing Prometheus metrics to identify resource-intensive operations or bottlenecks is crucial.

Debugging Strategies

- **Log Analysis:** Regularly reviewing logs to spot anomalies or recurring errors.
- **Monitoring Alerts:** Responding promptly to alerts from Prometheus to address potential issues before they escalate.
- **Testing and Rollbacks:** Employing thorough testing in the staging environment and being prepared to roll back changes if they introduce critical issues in the production environment.

Continuous Improvement

- **Feedback Loop:** Incorporating user and team feedback into development to continually refine and improve the application.
- **Iterative Approach:** Regularly updating and improving the application based on monitoring data, security scan results, and user feedback.

10. Feedback Loop Implementation

User Feedback Integration

- **Interactive User Interface:** The web application features a dedicated section for user feedback, including a rating system and a comments section. This allows users to provide valuable insights into their experience with the application.
- **Real-time Analysis:** Feedback submitted by users is analyzed in real-time, providing the development team with immediate insights into user satisfaction and areas for improvement.

Utilizing Monitoring Data

- **Performance Metrics:** Metrics collected via Prometheus are regularly reviewed to identify trends, potential issues, and opportunities for optimization.
- **Data-Driven Decisions:** Decisions regarding feature updates, bug fixes, and performance enhancements are driven by the collected monitoring data, ensuring that changes align with actual user needs and application performance.

Continuous Improvement Process

- **Iterative Development:** The development process is iterative, with regular updates based on user feedback and monitoring insights.
- **Enhancement and Refinement:** Continuous refinement of features and performance, aiming to enhance user experience and application efficiency.

11. Hosting

Hosting Choice

- **Compatibility Consideration:** Due to compatibility issues with the latest Python version (Python 3.12.1) on AWS Elastic Beanstalk, the decision was made to host the application on Webhost.
- **Accessibility and Reliability:** The Webhost platform provides reliable hosting with easy access, ensuring the application is available for real-time access and testing.

Deployment Process

- **File Transfer:** The application codebase, along with all dependencies and configurations, was transferred to Webhost.
- **Configuration and Setup:** Necessary configurations were made on Webhost to align with the application's requirements, ensuring smooth operation.

12. Conclusion

The DevOps Web App project stands as a classic example of integrating modern software development and operational practices. It effectively demonstrates the seamless fusion of continuous integration, deployment, monitoring, and security within a software project lifecycle. Leveraging innovative tools like Terraform for infrastructure and SonarQube for security, the project not only adheres to industry best practices but also showcases adaptability and problem-solving skills in hosting solutions. This endeavor not only fulfills the academic objectives of the Software Project Management course but also serves as a practical blueprint for future software development projects aiming to embody the essence of DevOps principles.