DEVOPS PROJECT REPORT

CREATING DATASETS AND TABLES
USING TERRAFORM ON
GOOGLE CLOUD PLATFORM.

Submitted by

Sri venkatesh

Registration No: 12212297

Computer Science & Engineering

Section: 9S055

Course Code: CSES010

Discipline of CSE/IT

LOVELY SCHOOL OF COMPUTER SCIENCE & ENGINEERING.

DECLARATION

I, sri venkatesh student of Computer Science & Engineering under CSE/IT Discipline at Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own intensive work and is genuine.

Date: 20-07-2025 Signature

Registration No. 12212297 sri venkatesh.

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my teacher Gourav Yadav who gave me the golden opportunity to do this wonderful project of analysis of the data of a superstore namely “Creating datasets and tables using terraform on google cloud platform” which also helped me in doing a lot of research and I came to know about so many new things. I am thankful to them. Secondly, I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

SIGNATURE

Sri venkatesh.

ABSTRACT:

The project aims to facilitate the deployment and management of a web application using Docker containers and Terra form for infrastructure provisioning. The web application, built with HTML, CSS, and JavaScript, provides a platform for users to interact with various features such as browsing galleries, viewing services, and accessing team information. To ensure scalability and flexibility in managing the application infrastructure, Docker containers are employed to encapsulate the application's components and dependencies, enabling consistent deployment across different environments.

The infrastructure provisioning aspect of the project utilizes Terraform to define and manage resources on Google Cloud Platform (GCP). Specifically, Google Big Query datasets and tables are provisioned to store and manage structured data efficiently. The Terraform configuration automates the deployment of datasets and tables, ensuring consistency and reproducibility in the infrastructure setup.

Through the integration of Docker containers and Terra form, the project streamlines the deployment and management processes of the web application, enhancing scalability, reliability, and maintainability. This approach enables efficient resource utilization and simplifies the task of infrastructure management, empowering developers to focus on building and improving the application's features while ensuring a robust and scalable deployment environment.

Additionally, the use of Docker containers allows for improved portability and consistency across development, testing, and production environments. By encapsulating the application and its dependencies within containers, developers can ensure that the application behaves consistently across different platforms and deployment scenarios. This approach promotes a DevOps culture, where development and operations teams collaborate seamlessly to deliver high-quality software efficiently.

Furthermore, leveraging Terraform for infrastructure provisioning adds an extra layer of automation and repeatability to the deployment process. With Terraform, infrastructure can be defined as code, enabling version control, collaboration, and the ability to replicate infrastructure configurations across multiple environments. This approach enhances the agility of the development process, enabling teams to quickly adapt to changing requirements and scale resources as needed to meet growing demand. Overall, the combined use of Docker containers and Terra form for deployment and infrastructure provisioning offers a robust foundation for building and managing modern web applications with efficiency and agility.

1. INTRODUCTION:

1.1 Problem Definition

The project addresses the challenges associated with deploying and managing a web application in a scalable and efficient manner. Traditional deployment methods often involve manual setup and configuration, leading to inconsistencies across environments and increased deployment times. Additionally, managing infrastructure resources manually can be error-prone and time-consuming, hindering the development and delivery process.

The problem statement revolves around the need for a streamlined and automated approach to deployment and infrastructure provisioning. Specifically, the project aims to:

Simplify the deployment process: Traditional deployment methods often involve complex steps and manual configurations, leading to deployment errors and inconsistencies. There is a need for a solution that simplifies the deployment process, making it easier for developers to deploy the application consistently across different environments.

Ensure scalability and flexibility: As the application grows and evolves, it's essential to have a deployment infrastructure that can scale seamlessly to accommodate increased traffic and demand. Manual infrastructure management can be challenging to scale efficiently and may lead to resource wastage or performance bottlenecks.

Enhance maintainability and reliability: Manual management of infrastructure resources can result in configuration drift and inconsistencies, making it difficult to maintain and troubleshoot the deployment environment. There is a need for a solution that ensures consistency and reliability in the deployment process, reducing the risk of errors and downtime.

In summary, the problem statement revolves around the challenges associated with manual deployment and infrastructure management, highlighting the need for a streamlined and automated approach to ensure scalability, reliability, and efficiency in deploying and managing web applications.

1.2 Problem Overview:

The project aims to streamline the deployment and management of a web application using Docker containers and Terra form for infrastructure provisioning. The web application is built with HTML, CSS, and JavaScript, offering features such as gallery browsing, service viewing, and team information access.

Key Components:

1. Web Application: The core of the project is the web application, which provides an interactive platform for users. The application is designed with a user-friendly interface and features various functionalities to enhance user engagement.
2. Docker Containers: Docker containers are utilized to encapsulate the web application and its dependencies. This approach ensures consistency in deployment across different environments and enhances portability and scalability.
3. Terraform: Terraform is employed for infrastructure provisioning on Google Cloud Platform (GCP). Specifically, Google Big Query datasets and tables are provisioned to store and manage structured data efficiently.

Project Objectives:

1. Simplify Deployment: Implement a streamlined deployment process to simplify the deployment of the web application across different environments.
2. Ensure Scalability: Design an infrastructure setup that can scale seamlessly to accommodate increased traffic and demand as the application grows.
3. Enhance Reliability: Ensure consistency and reliability in the deployment process to minimize errors and downtime, promoting a robust and stable application environment.
4. Automate Infrastructure Management: Utilize Terraform to automate the provisioning and management of infrastructure resources, enabling efficient resource utilization and reproducibility.

Overall, the project aims to leverage Docker containers and Terraform to create an efficient and scalable deployment environment for the web application, enhancing developer productivity, and ensuring a reliable user experience.

1.2 **Hardware Specification**:

The hardware specifications primarily depend on the scale of the application, expected user traffic, and the complexity of the infrastructure required. Here's a general outline of hardware specifications you might consider:

Development Machine:

Processor: Multi-core processor (e.g., Intel Core i5 or higher, AMD Ryzen 5 or higher)

RAM: At least 8GB of RAM for development tasks, but preferably 16GB or more for smoother performance.

Storage: SSD storage for faster file access and compilation times.

Server(s) for Hosting:

CPU: Depending on the expected load, a multi-core CPU is recommended (e.g., Intel Xeon or AMD EPYC series).

RAM: Sufficient RAM to handle concurrent requests and application workload. At least 8GB to start, but scaling up to 16GB or more is common for production environments.

Storage: SSD storage for faster data access and better performance.

Network: Gigabit Ethernet or higher for fast data transfer rates and low latency.

Cloud Providers: If using cloud services like Google Cloud Platform, Amazon Web Services, or Microsoft Azure, ensure that the selected instance types meet the application's requirements for CPU, RAM, and storage.

Networking Equipment:

Router: A reliable router capable of handling network traffic efficiently.

Switch: Managed switches with sufficient ports for connecting servers and other network devices.

Firewall: Hardware or software-based firewall to protect the network from unauthorized access and attacks.

Monitoring and Management Tools:

Monitoring: Consider using monitoring tools like Prometheus, Grafana, or Datadog to monitor server performance, application health, and resource usage.

Deployment and Configuration Management: Tools like Kubernetes, Docker Swarm, or Ansible can help automate deployment and configuration tasks.

Backup and Disaster Recovery:

Regular backups: Implement a backup strategy to ensure data integrity and availability in case of hardware failures or data loss incidents.

Disaster Recovery Plan: Have a plan in place to recover from catastrophic events such as server failures, natural disasters, or cyberattacks.

These hardware specifications are general guidelines and may vary depending on the specific requirements and constraints of your project. It's essential to assess the scalability, performance, and security needs of your application to determine the most suitable hardware configuration. Additionally, consider factors like budget, maintenance, and future growth when planning your hardware infrastructure.

1.3 Software Specification:

Development Environment:

Code Editor: Choose a code editor or integrated development environment (IDE) suitable for web development. Popular options include Visual Studio Code, Sublime Text, Atom, or JetBrains IntelliJ IDEA.

Version Control: Utilize a version control system like Git for managing code changes, collaboration, and version history. Platforms like GitHub, GitLab, or Bitbucket offer hosting services for Git repositories.

Web Browser: Use modern web browsers like Google Chrome, Mozilla Firefox, or Microsoft Edge for testing and debugging your web application.

Frontend Technologies:

HTML, CSS, JavaScript: Standard web technologies for building the user interface and frontend functionality.

Frontend Frameworks: Consider using frontend frameworks/libraries like React.js, Angular, or Vue.js to simplify frontend development and enhance user experience.

CSS Preprocessors: Tools like Sass or Less can help streamline CSS development by adding features like variables, mixins, and nesting.

Backend Technologies:

Web Server: Choose a web server software to host your web application. Common options include Apache HTTP Server, Nginx, or Microsoft Internet Information Services (IIS).

Server-side Framework: Select a server-side programming language and framework to build the backend logic of your web application. Popular choices include Node.js with Express.js (JavaScript), Django (Python), Ruby on Rails (Ruby), or ASP.NET (C#).

Database: Use a relational database management system (RDBMS) like PostgreSQL, MySQL, or MariaDB for storing structured data. Alternatively, consider NoSQL databases like MongoDB or Firebase Firestore for flexible data storage.

Containerization and Orchestration:

Docker: Containerization platform for packaging, distributing, and running applications in isolated environments.

Container Orchestration: Tools like Kubernetes, Docker Swarm, or Amazon ECS (Elastic Container Service) for automating deployment, scaling, and management of containerized applications.

Infrastructure Provisioning:

Infrastructure as Code (IaC) Tools: Utilize tools like Terraform, AWS CloudFormation, or Google Cloud Deployment Manager for provisioning and managing infrastructure resources in a declarative manner.

Monitoring and Logging:

Application Monitoring: Implement monitoring solutions like Prometheus, Grafana, or Datadog to monitor application performance, resource usage, and user experience.

Logging: Use logging frameworks or services (e.g., Winston, Log4js, ELK stack) to collect, store, and analyze application logs for troubleshooting and debugging purposes.

2. LITERATURE SURVEY

The existing system incorporates the use of JSON files, Terraform, Google Cloud Platform (GCP), and Docker for infrastructure provisioning and deployment. Here's a breakdown of each component's role in the existing system:

JSON Files:

JSON (JavaScript Object Notation) files are used to define the schema of datasets and tables in Google BigQuery, a fully managed, serverless data warehouse provided by Google Cloud Platform.

These JSON files specify the structure of the data to be stored in BigQuery tables, including field names, data types, and other attributes such as mode (e.g., "REQUIRED").

Each JSON file corresponds to a dataset and contains configurations for one or more tables within that dataset.

Terraform:

Terraform is an open-source infrastructure as code (IaC) tool used for building, changing, and versioning infrastructure efficiently.

In the existing system, Terraform is employed to automate the provisioning of Google Cloud resources required for storing and managing structured data using BigQuery.

Terraform configuration files (*.tf) define the desired state of the infrastructure, including datasets, tables, and their associated schemas.

By utilizing Terraform, infrastructure resources can be provisioned and managed programmatically, allowing for reproducibility, consistency, and scalability.

Google Cloud Platform (GCP):

Google Cloud Platform offers a suite of cloud computing services, including infrastructure, storage, and data analytics solutions.

In the existing system, GCP is utilized as the cloud provider for hosting and managing the infrastructure required to store and process structured data.

Specifically, Google BigQuery is used as the data warehouse solution to store and analyze large datasets efficiently.

Other GCP services, such as Google Cloud Storage for object storage and Google Kubernetes Engine for container orchestration, may be integrated into the system depending on requirements.

Docker:

Docker is a platform for developing, shipping, and running applications in containers.

While not explicitly mentioned in the provided information, Docker containers may be used in conjunction with Terraform and GCP to containerize and deploy applications or services alongside the infrastructure provisioning process.

Docker containers offer advantages such as portability, consistency, and isolation, making them suitable for deploying applications across different environments, including development, testing, and production.

Overall, the existing system leverages JSON files to define data schemas, Terraform for infrastructure automation on Google Cloud Platform, and potentially Docker containers for application deployment. This combination of tools enables efficient, scalable, and reproducible infrastructure provisioning and deployment processes.

2.2 Proposed System:

The proposed system builds upon the existing infrastructure provisioning and deployment process by introducing enhancements and optimizations. Here's an outline of the proposed system**:**

Enhanced Infrastructure Automation:

Expand the use of Terraform to automate the provisioning of additional Google Cloud Platform (GCP) resources beyond Google BigQuery datasets and tables.

Define infrastructure-as-code (IaC) templates to provision compute instances, networking components, storage resources, and other GCP services required for the application stack.

Utilize Terraform modules to modularize and abstract infrastructure configurations, promoting reusability and maintainability across projects.

Containerization and Microservices Architecture:

Embrace containerization using Docker to package applications and services into lightweight, portable containers.

Adopt a microservices architecture to decompose the application into smaller, independently deployable services.

Containerize each microservice using Docker containers, allowing for easy deployment, scaling, and management across different environments.

Container Orchestration with Kubernetes:

Implement container orchestration using Kubernetes to automate deployment, scaling, and management of containerized microservices.

Deploy Kubernetes clusters on Google Kubernetes Engine (GKE), a managed Kubernetes service provided by GCP.

Define Kubernetes deployment manifests (YAML files) to specify the desired state of application deployments, including pod specifications, service definitions, and ingress configurations.

Monitoring, Logging, and Observability:

Integrate monitoring and logging solutions such as Prometheus, Grafana, and ELK Stack (Elasticsearch, Logstash, Kibana) for real-time visibility into application performance and health.

Configure alerts and notifications to proactively identify and address potential issues or anomalies in the system.

Implement distributed tracing and logging to facilitate troubleshooting and debugging of microservices-based architectures**.**

Security and Compliance:

Implement security best practices and compliance standards to safeguard application data and infrastructure resources.

Utilize GCP security features such as Identity and Access Management (IAM), VPC Service Controls, and Security Command Center for access control, network security, and threat detection.

Implement encryption mechanisms (e.g., TLS/SSL) to secure data in transit and at rest, adhering to regulatory requirements and industry standards.

Scalability and High Availability:

Design the system for scalability and high availability to accommodate varying workloads and ensure uptime and reliability.

2. PROBLEM FORMULATION:

The problem formulation for this Terraform project revolves around the need to efficiently manage and automate the provisioning of infrastructure resources on Google Cloud Platform (GCP) for storing and managing structured data using Google Big Query. Here's a breakdown of the key components of the problem formulation:

Infrastructure Provisioning: The project aims to automate the provisioning of GCP resources required to set up and manage datasets and tables in Google Big Query. This includes defining the infrastructure components such as datasets, tables, and associated schemas in a declarative manner using Terraform.

Infrastructure as Code (IaC): The challenge involves defining the infrastructure configurations as code using Terraform to ensure consistency, repeatability, and version control. By treating infrastructure as code, changes can be made programmatically and applied consistently across different environments, eliminating manual configuration and reducing the risk of human error.

Data Schema Management: The project involves defining and managing the schema of datasets and tables in Google Big Query using JSON files. Each JSON file specifies the structure of the data to be stored, including field names, data types, and mode (e.g., "REQUIRED"). The challenge is to integrate these JSON schema definitions into the Terraform configuration to automate the creation and management of datasets and tables.

Automation and Orchestration: The goal is to automate the entire process of infrastructure provisioning and configuration using Terraform, from creating datasets and tables to defining their schemas and dependencies. Additionally, the project may involve orchestrating the deployment of infrastructure resources in a coordinated and efficient manner to ensure proper sequencing and dependency resolution.

Scalability and Efficiency: The project needs to address scalability and efficiency considerations, such as optimizing resource utilization, minimizing costs, and accommodating future growth and expansion. This may involve implementing best practices for resource allocation, performance tuning, and cost optimization within the Terraform configurations.

Integration with GCP Services: The project may require integration with other GCP services and features to enhance functionality and meet specific requirements. This could include integrating with Google Cloud Storage for data storage, Google Cloud IAM for access control, and Google Cloud Monitoring for resource monitoring and management.

Overall, the problem formulation for this Terraform project revolves around automating the provisioning and management of infrastructure resources on Google Cloud Platform for storing and managing structured data using Google Big Query, while ensuring scalability, efficiency, and integration with other GCP services.

3. OBJECTIVES:

The objectives of the project encompass various aspects related to infrastructure automation, deployment efficiency, and scalability on Google Cloud Platform (GCP). Here are the key objectives:

Automate Infrastructure Provisioning: Develop Terraform configurations to automate the provisioning of infrastructure resources on GCP, including datasets and tables in Google Big Query, networking components, compute instances, and other required services.

Implement Infrastructure as Code (IaC): Utilize Terraform to define infrastructure configurations as code, enabling version-controlled, repeatable, and consistent deployments across different environments, such as development, staging, and production.

Streamline Data Schema Management: Integrate JSON schema definitions with Terraform configurations to automate the creation and management of datasets and tables in Google BigQuery, ensuring proper data structure and consistency.

Enhance Deployment Efficiency: Optimize the deployment process to improve efficiency, reduce manual intervention, and minimize deployment time by leveraging Terraform's declarative approach and infrastructure automation capabilities.

Ensure Scalability and Flexibility: Design the infrastructure to be scalable and flexible, allowing for seamless scaling of resources based on demand, efficient resource utilization, and support for future growth and expansion of the application.

Ensure Reliability and High Availability: Implement best practices for reliability and high availability, including redundancy, fault tolerance, and disaster recovery measures, to ensure continuous operation and minimal downtime of the deployed services.

Integrate with GCP Services: Integrate with other GCP services and features as needed, such as Google Cloud Storage for data storage, Google Cloud IAM for access control, and Google Cloud Monitoring for resource monitoring and management.

Implement Security and Compliance: Ensure adherence to security best practices and compliance standards by implementing robust security controls, encryption mechanisms, access controls, and monitoring solutions within the GCP environment.

Enable Continuous Integration and Continuous Deployment (CI/CD): Establish CI/CD pipelines to automate the build, test, and deployment processes, enabling rapid and reliable delivery of application updates and enhancements while maintaining quality and stability.

Provide Monitoring and Observability: Implement monitoring, logging, and observability solutions to gain insights into the performance, health, and behavior of the deployed infrastructure and applications, enabling proactive monitoring, troubleshooting, and optimization.

Facilitate DevOps Practices: Foster collaboration and communication between development and operations teams by adopting DevOps practices, automation tools, and standardized processes for infrastructure management and deployment.

Optimize Cost Management: Implement cost optimization strategies to maximize cost efficiency and minimize infrastructure expenses, including rightsizing of resources, utilization analysis, and billing monitoring.

By achieving these objectives, the project aims to modernize infrastructure management practices, enhance deployment efficiency, ensure scalability and reliability, and enable seamless operation of cloud-native applications on Google Cloud Platform.

5. METHODOLOGY:

Requirement Analysis: Gather and analyze the requirements for infrastructure provisioning, including datasets, tables, schemas, networking, security, and compliance considerations.

Architecture Design: Design the architecture of the infrastructure, defining the layout of datasets, tables, and associated resources in Google Big Query.

Determine the networking setup, security policies, and access controls required to ensure a secure and compliant environment.

Terraform Configuration: Develop Terraform configurations to define the infrastructure as code, including datasets, tables, schemas, networking components, IAM policies, and other resources. Implement best practices for Terraform code structure, variable management, and resource dependencies.

JSON Schema Management: Create JSON schema files to specify the structure of datasets and tables in Google Big Query, including field names, data types, and modes. Integrate the JSON schema files into the Terraform configurations to automate the creation and management of datasets and tables.

Testing and Validation:

Conduct thorough testing of the Terraform configurations to ensure correctness, completeness, and consistency of the infrastructure deployment.

Validate the functionality of datasets, tables, and associated resources in Google Big Query to confirm that they meet the specified requirements.

Deployment and Orchestration: Deploy the infrastructure resources on GCP using Terraform, following a well-defined deployment process and workflow. Orchestrate the deployment sequence to ensure proper sequencing and dependency resolution of resources, minimizing downtime and ensuring reliability.

Monitoring and Optimization: Implement monitoring and observability solutions to track the performance, health, and usage of the deployed infrastructure and applications.Continuously monitor resource utilization, cost metrics, and security posture, and optimize the infrastructure configuration as needed to improve efficiency and reduce costs.

6. EXPERIMENTAL SETUP:

Google Cloud Platform Account: Create a Google Cloud Platform account or use an existing one with appropriate permissions to create and manage resources.

Terraform Installation: Install Terraform on the local development machine or a dedicated server. Ensure the Terraform binary is added to the system PATH for easy access.

Project Directory Structure: Organize the project files into a structured directory layout.Create separate directories for Terraform configurations, JSON schema files, documentation, and other project-related assets.

Terraform Configuration Files: Write Terraform configuration files (*.tf) to define the infrastructure resources, including datasets, tables, networking components, IAM policies, and other GCP resources.

Use Terraform modules to encapsulate reusable components and promote code modularity and maintainability.

JSON Schema Files: Create JSON schema files (*.json) to specify the structure of datasets and tables in Google Big Query. Define field names, data types, modes, and other schema properties according to the requirements.

Service Account and Credentials: Create a service account on GCP with appropriate roles and permissions to manage resources. Generate JSON key credentials for the service account and securely store them locally.

Variable Configuration: Define Terraform variables to parameterize the configuration and allow for customization based on environment-specific requirements. Configure input variables for project IDs, dataset names, table configurations, networking details, and other parameters.

Backend Configuration: Configure Terraform backend to store state files remotely for collaboration and version control. Choose an appropriate backend storage solution such as Google Cloud Storage (GCS) or Terraform Cloud.

6. ADVANTAGES:

Infrastructure as Code (IaC): Gain hands-on experience with Infrastructure as Code (IaC) principles and practices. Learn how to define, provision, and manage infrastructure resources using declarative configuration files.

Terraform Proficiency: Develop proficiency in Terraform, a leading IaC tool used for automating infrastructure provisioning across various cloud providers. Understand Terraform's syntax, resource types, modules, and best practices for writing maintainable and scalable infrastructure code.

Google Cloud Platform (GCP) Skills: Acquire knowledge and skills in Google Cloud Platform (GCP) services, particularly Google Big Query for data analytics and warehousing. Learn how to create and manage datasets, tables, schemas, and other resources on GCP using Terraform.

JSON Schema Management: Understand the importance of JSON schema files for defining the structure and schema of datasets and tables in Google BigQuery. Learn how to integrate JSON schema files into Terraform configurations to automate dataset and table creation. Infrastructure Automation: Explore the benefits of infrastructure automation, including consistency, repeatability, and scalability of deployments. Learn how to automate the deployment and management of infrastructure resources using Terraform, reducing manual effort and minimizing errors. Modular and Reusable Components: Gain experience in designing modular and reusable Terraform configurations to promote code reuse and maintainability. Understand how to structure Terraform codebase into modules for encapsulating common infrastructure patterns and configurations.

Version Control and Collaboration: Learn how to use version control systems like Git to manage Terraform codebase, track changes, and collaborate with team members. Understand Terraform's backend configuration options for storing state files remotely, enabling collaboration and version control across distributed teams.

Cloud-native Development: Embrace cloud-native development practices by leveraging cloud services and technologies for building scalable and resilient applications.

Gain insights into designing cloud-native architectures and integrating with managed services on GCP.

Continuous Integration and Deployment (CI/CD): Explore CI/CD pipelines for automating the testing, validation, and deployment of infrastructure changes. Learn how to integrate Terraform with CI/CD tools like Jenkins, GitLab CI/CD, or GitHub Actions to automate the infrastructure lifecycle.

Real-world Project Experience: Apply theoretical knowledge to real-world scenarios and projects, gaining practical experience in infrastructure automation and cloud technologies. Develop problem-solving skills by addressing challenges and complexities encountered during the project lifecycle.

7. REQUIREMENT ANALYSIS:

The requirement analysis for the project involves understanding the objectives, constraints, and stakeholders' needs to define the scope and functionality of the infrastructure automation solution. Here's a breakdown of the requirement analysis process:

Stakeholder Identification: Identify stakeholders involved in the project, including developers, operations teams, data analysts, and business stakeholders.

Understand their roles, responsibilities, and expectations regarding the infrastructure automation solution.

Objective Definition: Define the primary objectives and goals of the project, such as automating the provisioning of infrastructure resources on Google Cloud Platform (GCP) using Terraform. Determine the specific outcomes expected from the project, such as improving efficiency, reducing manual effort, and ensuring consistency in infrastructure deployments.

Functional Requirements: Identify the functional requirements of the infrastructure automation solution, including: Provisioning datasets and tables in Google Big Query based on JSON schema files. Configuring networking components, security policies, and access controls. Integrating with other GCP services for logging, monitoring, and alerting. Supporting scalability and high availability requirements for the deployed infrastructure.

Non-Functional Requirements: Define non-functional requirements that describe the quality attributes of the solution, such as:

Performance: Ensure efficient resource provisioning and minimal downtime.

Reliability: Guarantee the stability and resilience of the infrastructure.

Security: Implement robust security measures to protect sensitive data and resources.

Scalability: Support the ability to scale infrastructure resources based on demand.

Maintainability: Facilitate ease of maintenance, updates, and modifications to the infrastructure codebase.

Constraints and Assumptions: Identify any constraints or limitations that may impact the design and implementation of the solution, such as budgetary constraints, compliance requirements, or technical limitations. Document any assumptions made during the requirement analysis phase to clarify expectations and mitigate risks.

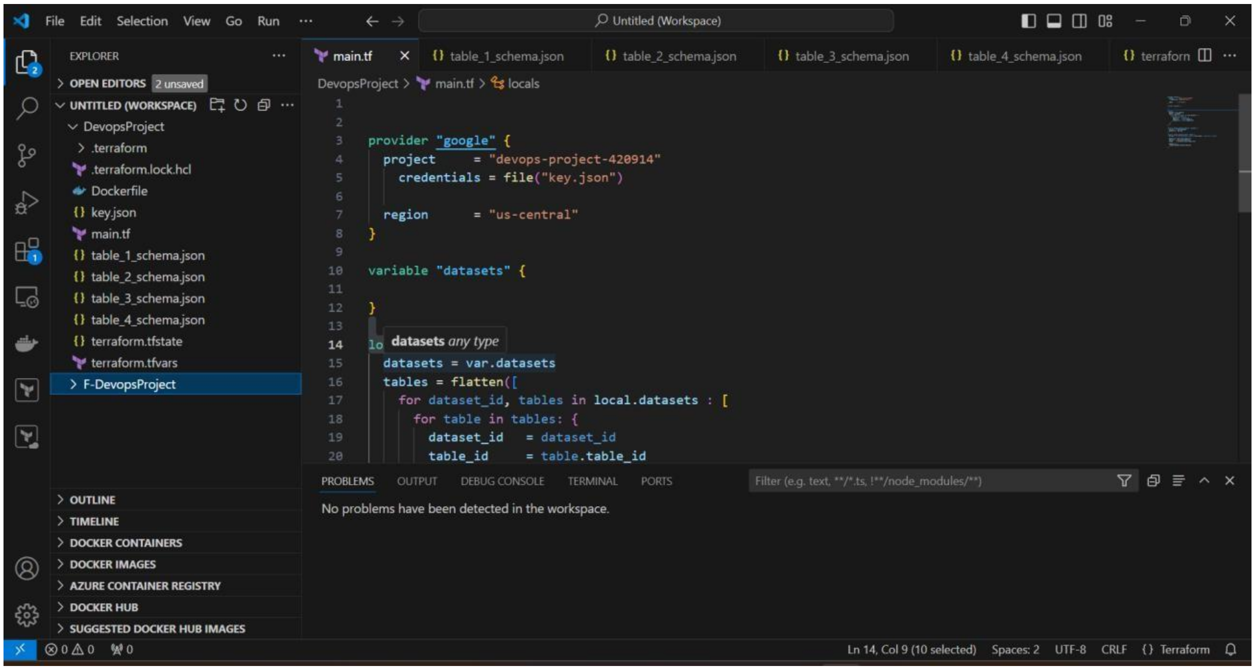
Use Cases and User Stories: Define use cases and user stories to capture specific scenarios and interactions with the infrastructure automation solution.Detail the steps, actors, and outcomes associated with each use case to guide the design and development process.

Risk Assessment: Conduct a risk assessment to identify potential risks and uncertainties associated with the project. Evaluate the impact and likelihood of each risk and develop mitigation strategies to address them proactively.

Feedback and Validation: Gather feedback from stakeholders to validate the requirements and ensure alignment with their needs and expectations. Iterate on the requirement analysis process based on feedback and incorporate any changes or adjustments as necessary.

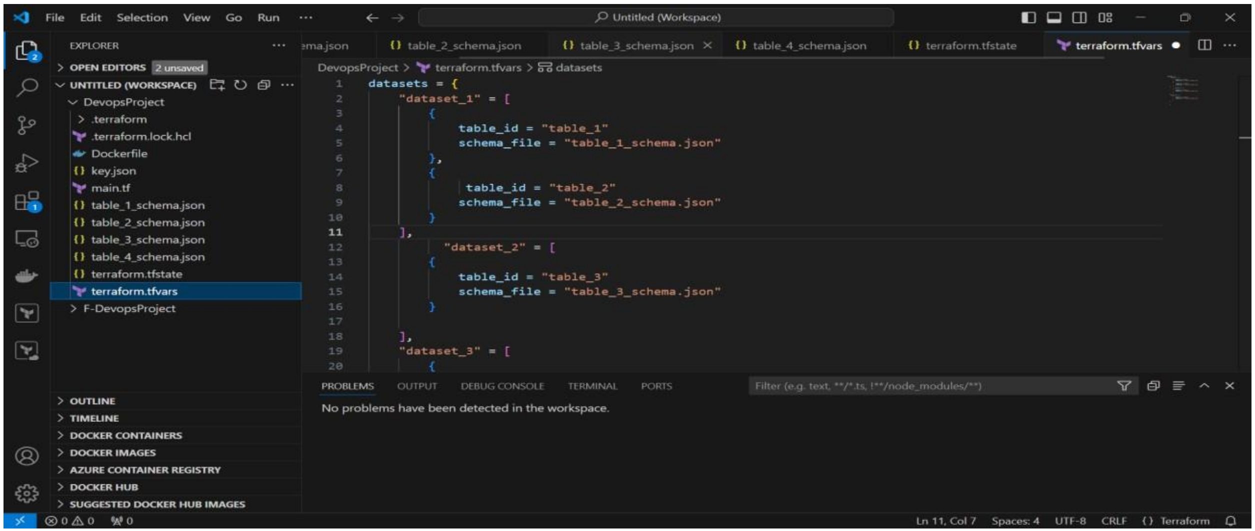
By conducting a thorough requirement analysis, the project team can establish a clear understanding of the project goals, scope, and deliverables, laying the foundation for a successful infrastructure automation solution on Google Cloud Platform using Terraform.

SNAPSHOTS OF THE OUTPUT AND THE COMMANDS :



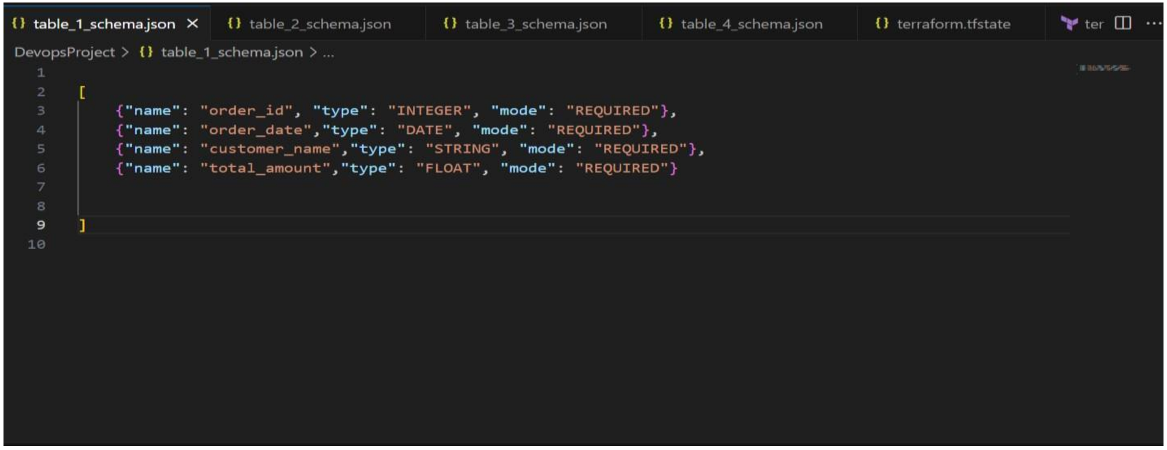
```
provider "google" {
  project     = "devops-project-420914"
    credentials = file("key.json")

  region      = "us-central"
}

variable "datasets" {

}

lo
  datasets = var.datasets
  tables = flatten([
    for dataset_id, tables in local.datasets : [
      for table in tables: {
        dataset_id   = dataset_id
        table_id     = table.table_id
```

No problems have been detected in the workspace.

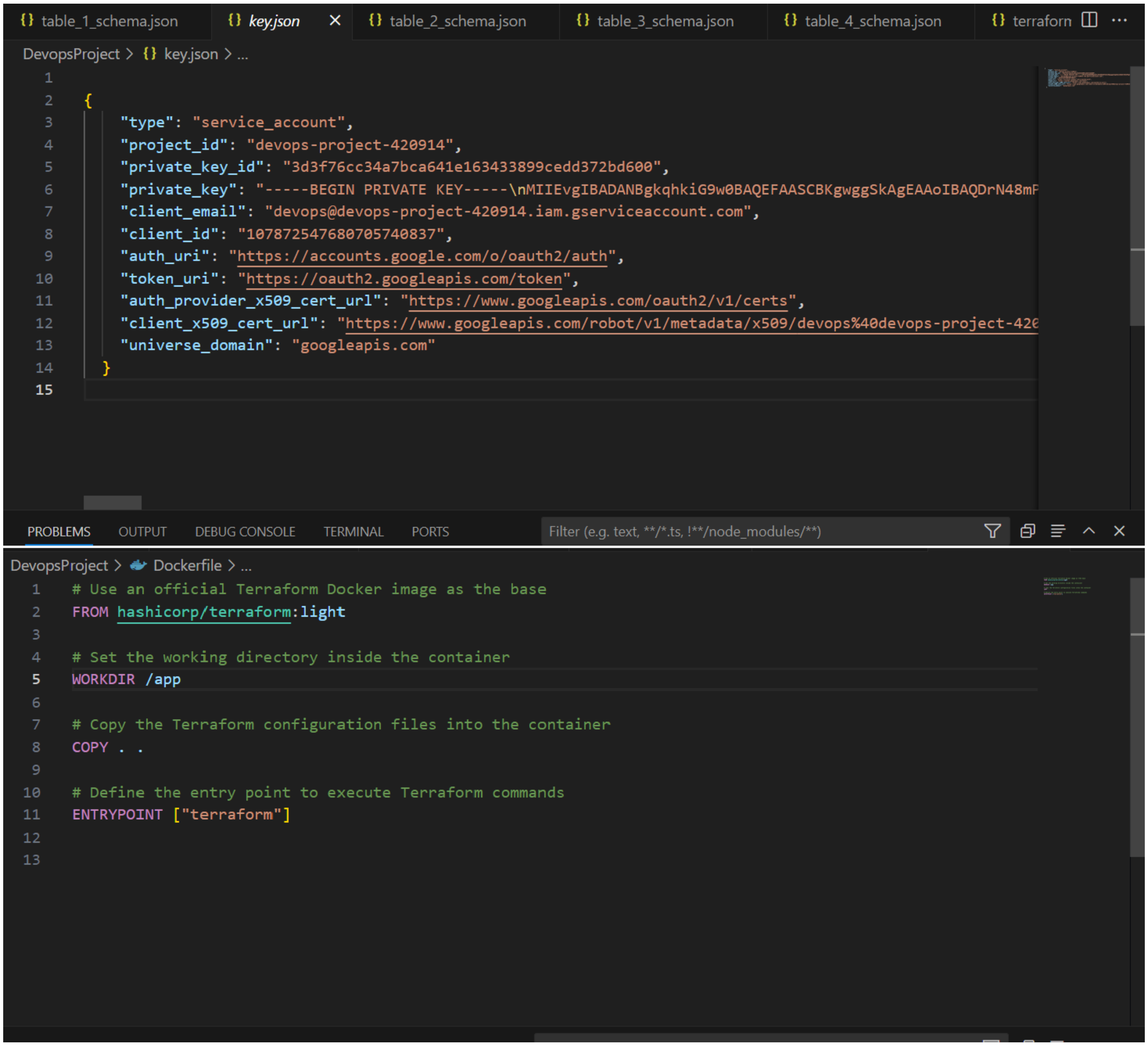


```
datasets = {
    "dataset_1" = [
        {
            table_id = "table_1"
            schema_file = "table_1_schema.json"
        },
        {
             table_id = "table_2"
            schema_file = "table_2_schema.json"
        }
    ],
          "dataset_2" = [
        {
            table_id = "table_3"
            schema_file = "table_3_schema.json"
        }

    ],
    "dataset_3" = [
        {
```

No problems have been detected in the workspace.



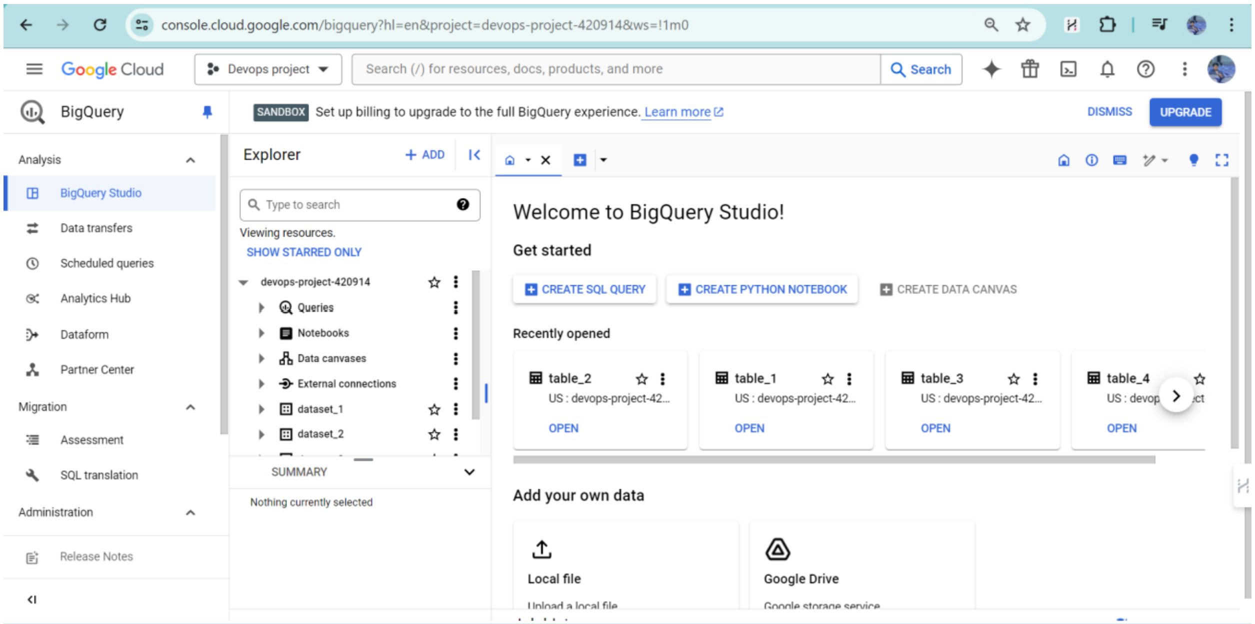
```

[
    {"name": "order_id", "type": "INTEGER", "mode": "REQUIRED"},
    {"name": "order_date","type": "DATE", "mode": "REQUIRED"},
    {"name": "customer_name","type": "STRING", "mode": "REQUIRED"},
    {"name": "total_amount","type": "FLOAT", "mode": "REQUIRED"}


]
```



```json

{
    "type": "service_account",
    "project_id": "devops-project-420914",
    "private_key_id": "3d3f76cc34a7bca641e163433899cedd372bd600",
    "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDrN48mP
    "client_email": "devops@devops-project-420914.iam.gserviceaccount.com",
    "client_id": "107872547680705740837",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/devops%40devops-project-420
    "universe_domain": "googleapis.com"
  }

```

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*.ts, !**/node_modules/**)
DevopsProject > Dockerfile > ...

```dockerfile
# Use an official Terraform Docker image as the base
FROM hashicorp/terraform:light

# Set the working directory inside the container
WORKDIR /app

# Copy the Terraform configuration files into the container
COPY . .

# Define the entry point to execute Terraform commands
ENTRYPOINT ["terraform"]


```



Set up billing to upgrade to the full BigQuery experience. Learn more

Viewing resources.

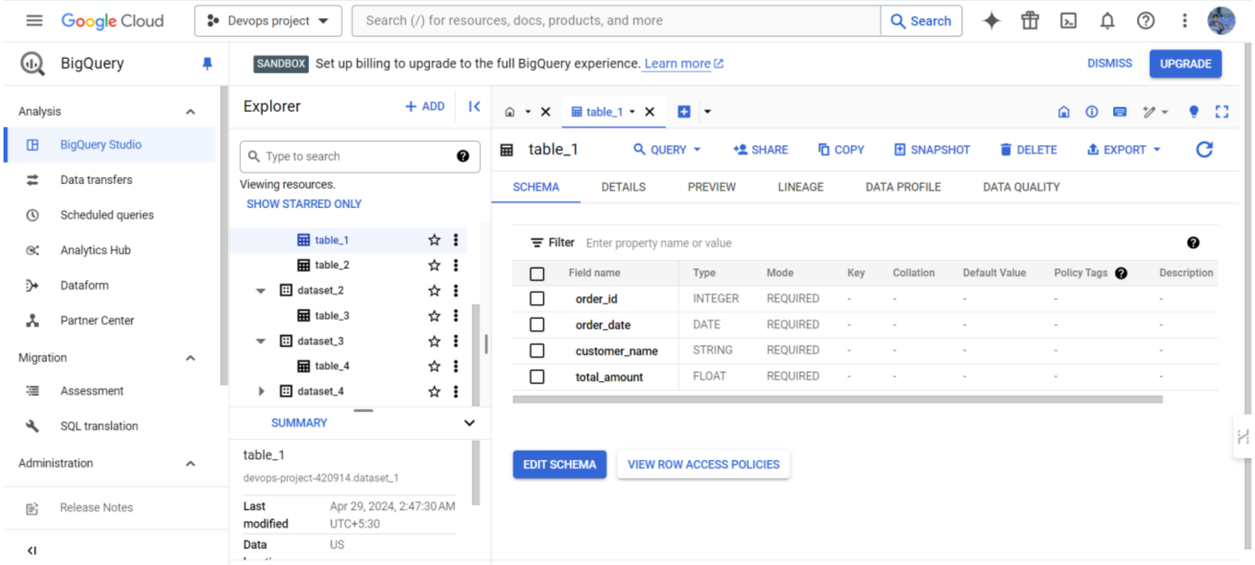
Nothing currently selected

Welcome to BigQuery Studio!

Get started

Recently opened

Add your own data



Set up billing to upgrade to the full BigQuery experience. Learn more

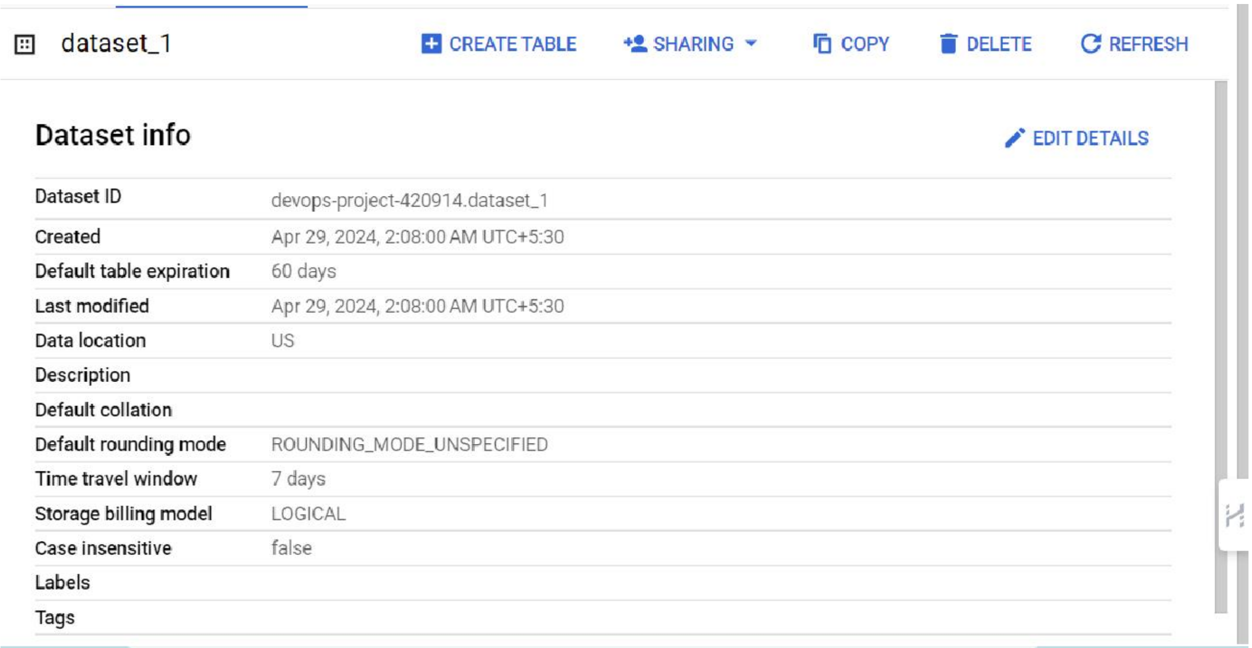
Viewing resources.

table_1

devops-project-420914.dataset_1

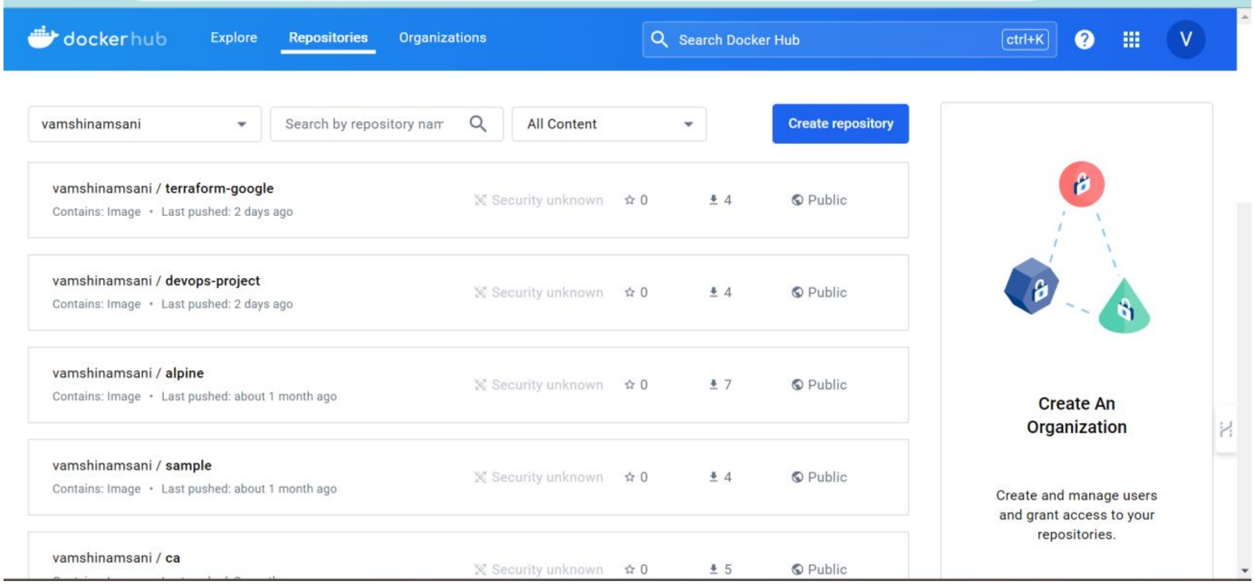
Last modified	Apr 29, 2024, 2:47:30 AM UTC+5:30
Data	US

☐	Field name	Type	Mode	Key	Collation	Default Value	Policy Tags	Description
☐	order_id	INTEGER	REQUIRED	-	-	-	-	-
☐	order_date	DATE	REQUIRED	-	-	-	-	-
☐	customer_name	STRING	REQUIRED	-	-	-	-	-
☐	total_amount	FLOAT	REQUIRED	-	-	-	-	-



Dataset info

Dataset ID	devops-project-420914.dataset_1
Created	Apr 29, 2024, 2:08:00 AM UTC+5:30
Default table expiration	60 days
Last modified	Apr 29, 2024, 2:08:00 AM UTC+5:30
Data location	US
Description	
Default collation	
Default rounding mode	ROUNDING_MODE_UNSPECIFIED
Time travel window	7 days
Storage billing model	LOGICAL
Case insensitive	false
Labels	
Tags	



vamshinamsani / terraform-google Contains: Image • Last pushed: 2 days ago	Security unknown	☆ 0	4	Public
vamshinamsani / devops-project Contains: Image • Last pushed: 2 days ago	Security unknown	☆ 0	4	Public
vamshinamsani / alpine Contains: Image • Last pushed: about 1 month ago	Security unknown	☆ 0	7	Public
vamshinamsani / sample Contains: Image • Last pushed: about 1 month ago	Security unknown	☆ 0	4	Public
vamshinamsani / ca	Security unknown	☆ 0	5	Public

Create and manage users and grant access to your repositories.

CONCLUSION:

The conclusion of the project serves as a summary of the key findings, achievements, and implications of the infrastructure automation solution implemented on Google Cloud Platform (GCP) using Terraform.