**Running the Project with GKE, Node.js App, ArgoCD, and CI/CD Pipeline**

This guide outlines the organized steps to run our project, which involves creating GKE clusters, deploying a Node.js application, integrating with ArgoCD, and setting up a CI/CD pipeline using GitHub Actions.

**Prerequisites:**

* **GCP Project:** A Google Cloud Platform project with billing enabled.
* **Terraform:** Installed and configured with our GCP credentials.
* **Helm:** Installed on our local machine.
* **Docker Hub account** If you plan to push your Node.js application image to a private registry.
* **Git repository:** A Git repository to store our Terraform configurations, Helm chart, and Node.js application code.

**Steps:**

1. **Project Structure:**
   * Create a root directory for our project.
   * Organize subdirectories for:
     + charts: Our Helm chart for the Node.js application.
     + clusters (dev, test, prod): Terraform configurations for each GKE cluster environment.
     + deployments: Deployment manifests for ArgoCD and Prometheus Operator (optional).
     + exporters : Exporter configurations for monitoring (e.g., Node Exporter).
     + grafana : Terraform configurations for Grafana deployment.
     + modules : Reusable modules for Terraform configurations.
     + ci.yml: Our GitHub Actions workflow file for CI/CD automation.
     + main.tf: The main Terraform configuration file.
     + outputs.tf: Outputs for cluster information from Terraform.
     + roles : RBAC role definitions for our clusters.
     + terraform.tfvars or terraform.vars (optional): Variable definition files for Terraform.
2. **Configure Terraform:**
   * Create a terraform.tfvars or terraform.vars file (or separate files per environment) to define variables for your project:
     + Project ID
     + Region
     + Cluster names (dev, test, prod)
     + Initial node count for each cluster
     + Cluster admin passwords (stored securely)
     + VPC network details
     + Firewall rule name
     + RBAC configuration
     + Node.js Helm chart details
     + ArgoCD project name
     + ArgoCD Git repository URL
     + Prometheus and Grafana configurations (optional)
   * Define the main Terraform configuration in main.tf to:
     + Initialize the GCP provider
     + Reference variable definitions
     + Create the VPC network (optional)
     + Define firewall rules (optional)
     + Call a reusable module (modules/gke\_cluster.tf) for creating GKE clusters with RBAC bindings (optional)
     + Deploy ArgoCD and Prometheus Operator (optional)
     + Deploy Grafana (optional)
     + Output cluster information (IP addresses, endpoints)
3. **Develop Node.js Application:**
   * Develop your Node.js application code.
   * Package your application into a Docker image using a Dockerfile within the charts directory.
   * Create a Helm chart in the charts directory to package your application and deployment configurations.
4. **CI/CD Pipeline with GitHub Actions:**
   * Create a ci.yml file in your Git repository to define the GitHub Actions workflow.
   * The workflow should:
     + Check out the code from your Git repository.
     + If using a private Docker registry, log in using Docker credentials stored as GitHub Actions secrets.
     + Build and push your Node.js application image to the container registry.
     + Use Terraform to apply configurations for each cluster environment (dev, test, prod). This might involve iterating through a list of cluster names and using environment variables for specific configurations.
     + Deploy your application using ArgoCD with the Helm chart, referencing the built image and Git commit SHA for version control.
     + Optionally, run tests for your Node.js application.
     + Optionally, update Grafana dashboards with relevant metrics.
5. **Run the Project:**
   * Initialize Terraform in each cluster directory (clusters/dev, clusters/test, clusters/prod):

Bash

cd clusters/dev

terraform init

Apply Terraform configurations for each environment (replace with your actual environment names):

Bash

terraform apply -auto-approve (for dev environment)

cd ../test

terraform apply -auto-approve (for test environment)

cd ../prod

terraform apply -auto-approve (for prod environment)