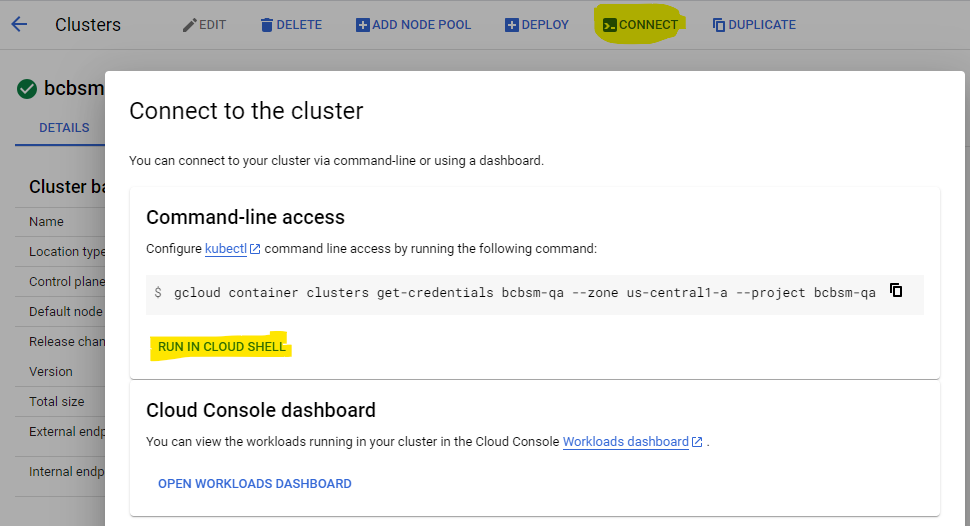
**CI/CD Deployment on Google Kubernetes Engine in a GitOps style**

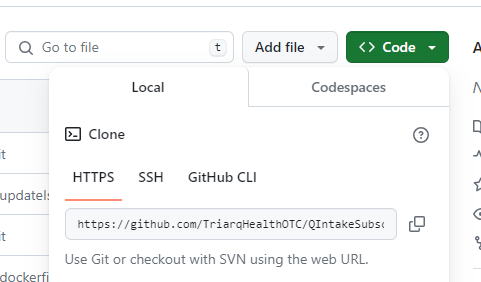
We are utilizing Google Kubernetes Engine.A screenshot of a computer

Description automatically generated

1. Connect to the cluster using Cloud Shell.
2. We are maintaining the source code on GitHub.

For deployment, we need to clone the GitHub repository into the cluster. Use the following command to clone the repository.

# git clone ${repository URL}



1. Change the directory to the cloned repository using the following command.

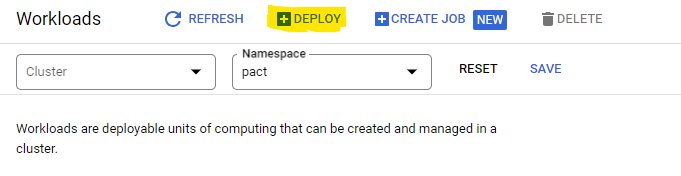
**# Cd ${repository name}**

1. Check whether the Dockerfile is present. If it is, create the image using the command.

# **docker build -t {image\_name}:{tag} .**

1. Push the image to the container registry using the following command.

# **docker push {image\_name}:{tag}**

1. Navigate to Workloads and select DEPLOY.

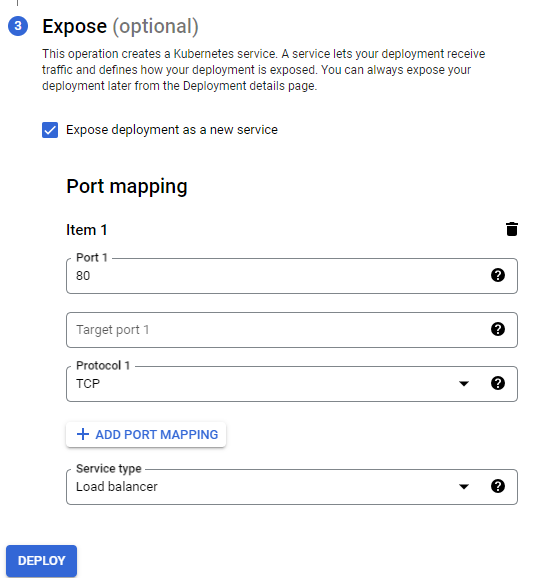
**Select the Existing container image and then choose the image path that we pushed.**A screenshot of a computer

Description automatically generated

**Configuration🡪 Assign the name identical to the repository name for identification.**

**Cluster 🡪 Select the cluster where developers want to deploy their application.**

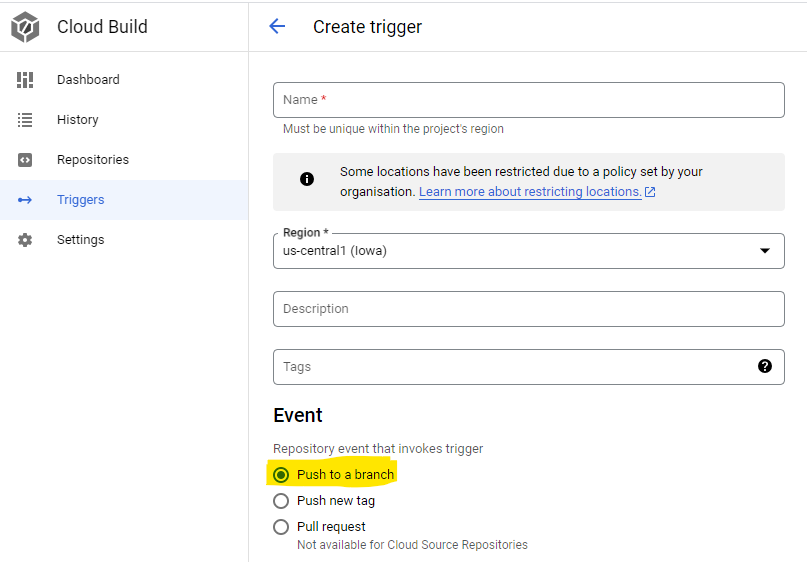
**Expose 🡪 Expose the service as a NodePort with the Service type.**

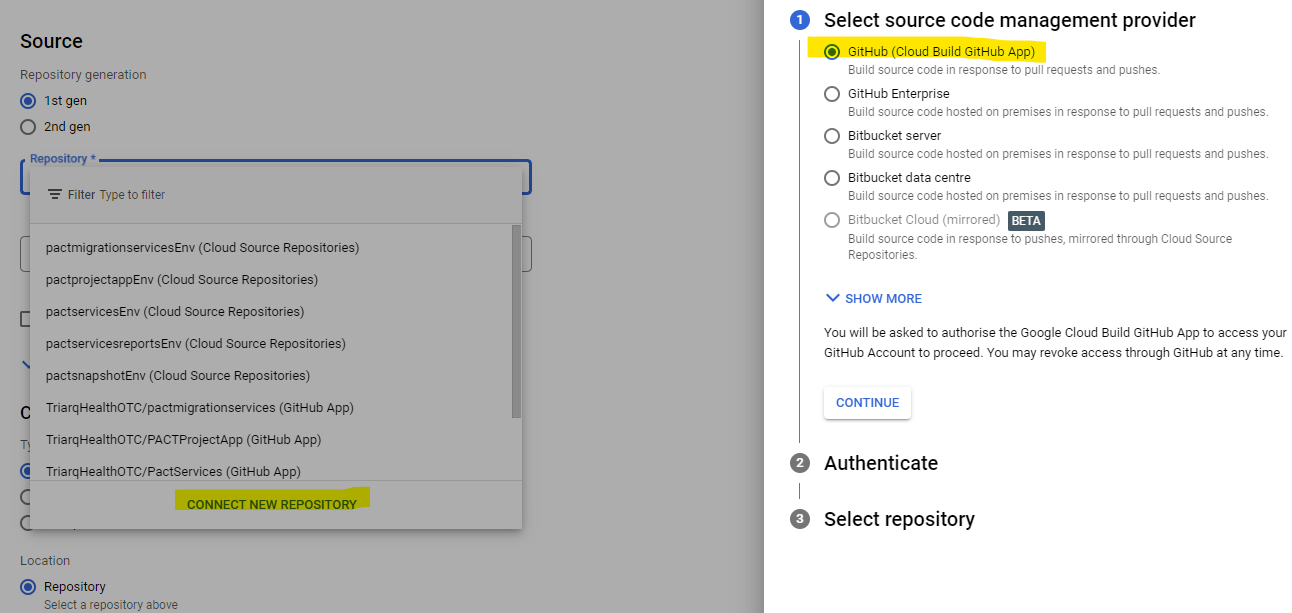
****

1. **After the deployment, a YAML file will be generated like this.**A screenshot of a computer

   Description automatically generated
2. We are using the **cloudbuild.yaml** file to automate the script.
3. We are using **kubernetes.yaml** files to store the deployment configurations.
4. In the GitHub repository, create three files: **kubernetes.yaml**, **kubernetes.yaml.tpl**, and **cloudbuild.yaml**.A screenshot of a computer

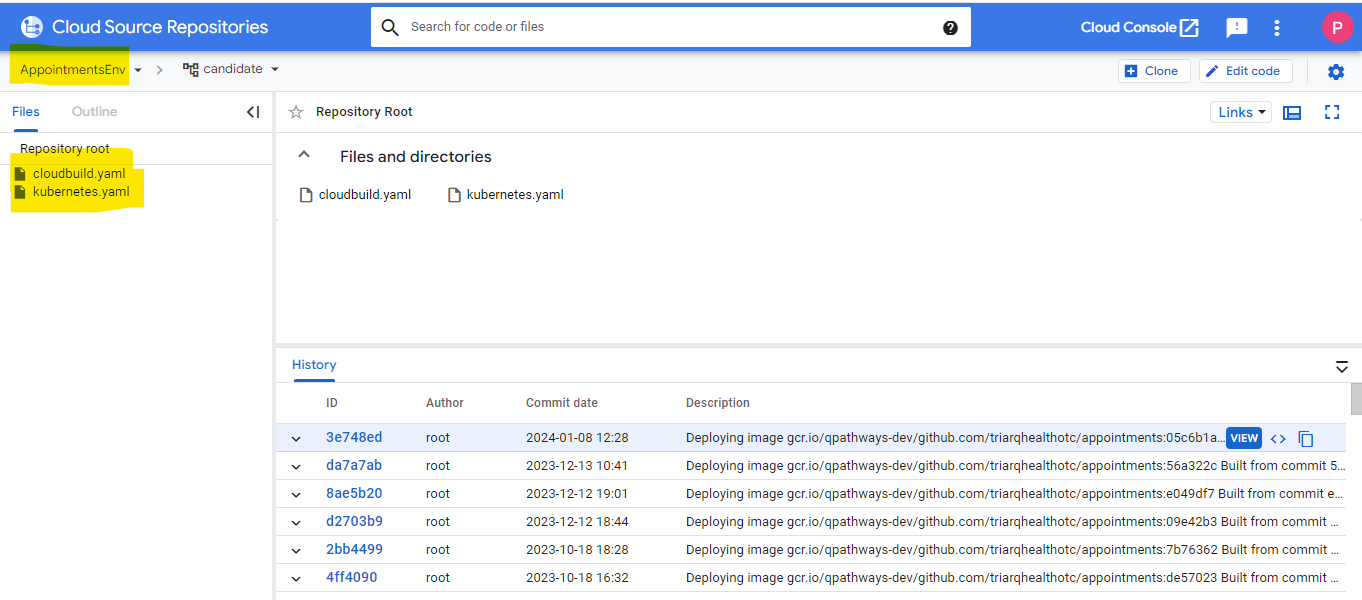
   Description automatically generated
5. In Google Cloud Platform (GCP), triggers are associated with Cloud Build and are used to initiate the execution of builds in response to specific events. The primary purpose of triggers is to automate the build and deployment process, ensuring that certain actions occur whenever a predefined event takes place. Triggers are part of the Cloud Build service and offer a way to automate continuous integration and continuous delivery (CI/CD) workflows.



**Select the repository**

Cloud Build configuration file location 🡪 cloudbuild.yaml

**Create**

**Cloud Source Repositories** seamlessly integrates with Google Cloud Build for continuous integration and continuous deployment (CI/CD) workflows. This allows automated builds and deployments triggered by code changes.  
  
  
****

1. We are using the **gitops.sh** automation script to create the repository on Cloud Source Repositories hosted on GCP.
2. We are uploading two files, **cloudbuild.yaml** and **kubernetes.yaml**, using the **gitops.sh** automation script.
3. These two files are used for continuous deployment.
4. execute **🡪 # sh gitops.sh ${repo\_name}Env**
5. ConfigMaps store key-value pairs or configuration files that applications need for their configuration.

**e.g.** Configuration settings, such as database connection strings, API keys, or environment-specific parameters, can be externalized from the application code.

For creating the ConfigMap, use the following command.

**# kubectl create configmap rpapayerpathbot2-config --from-literal Port="5432" --from-literal Host="10.143.128.8"**

For creating the secret, use the following command.

**# kubectl create secret generic rpapayerpathbot2-secret --from-literal Username=postgres --from-file=Password=rpapayerpathbot2.txt**

1. **Ingress**

The primary purpose of Ingress in GCP is to provide a way to route external HTTP and HTTPS traffic to different services running in a Kubernetes cluster based on specified rules. Ingress acts as an API gateway, allowing you to define how external traffic should be directed to various services.

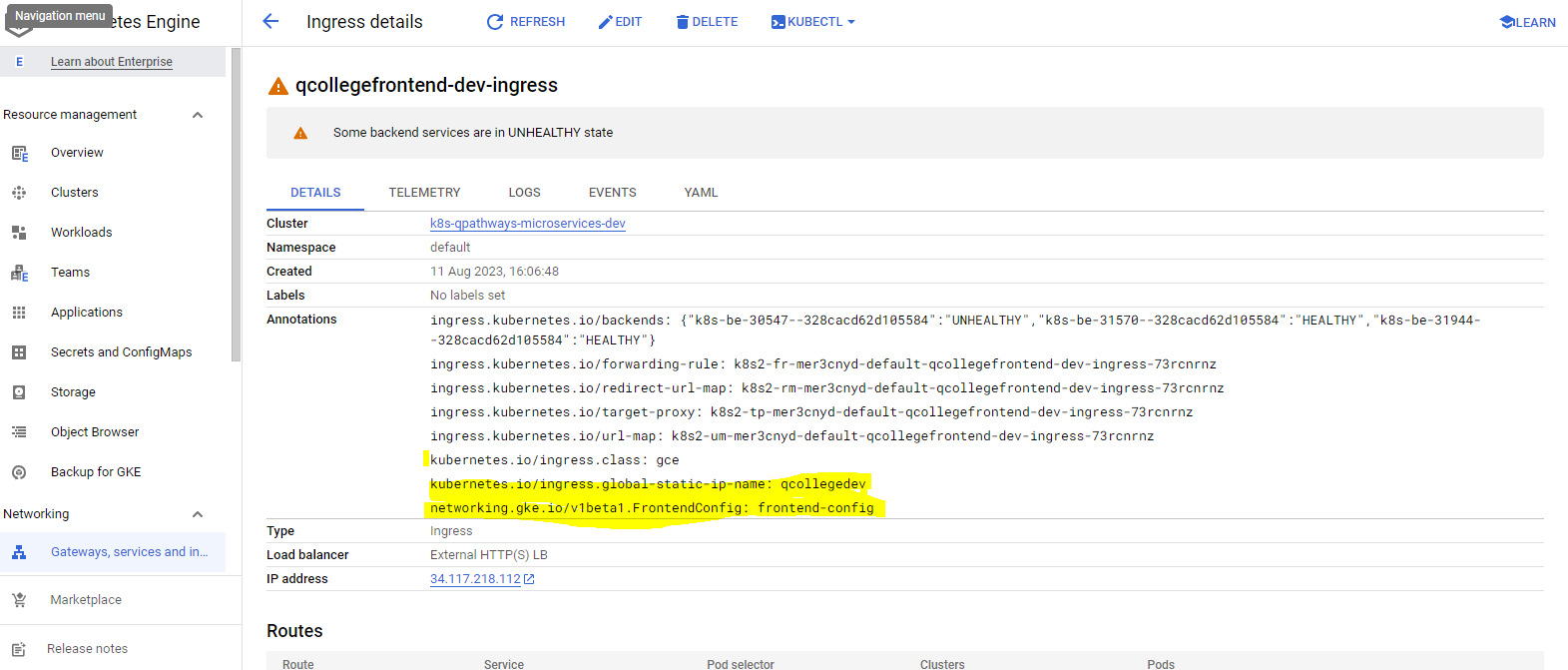
* In Kubernetes, the number of services you can host on an Ingress is practically unlimited and depends on your specific requirements, the capacity of your cluster, and the resources allocated to your Ingress controller.
* An Ingress resource defines how external HTTP/S traffic should be directed to services within your Kubernetes cluster based on rules. Each rule can have multiple paths, and each path can route traffic to a different Kubernetes service.

A screenshot of a computer

Description automatically generated

* We promoted the ingress IP to static and provided a name for that static IP.

We are implementing permanent redirection from HTTP to HTTPS by adding the following annotation in the Ingress YAML file.



Example domain request form fallowing.

|  |  |
| --- | --- |
| 1. Root Domain | Myqone.com |
| 1. Provide Environment   (Development, Stagging, QA, Production) | Development |
| 1. Type (Backend /Frontend) | Frontend |
| 1. Service name / Ingress Name | ehiexport-ingress |
| 1. Domain URL | ehi-dev.myqone.com |
| 1. IP address | 35.190.126.221 |
| 1. Project | <https://github.com/TriarqHealthOTC/EHIExport> |
| 1. Project owner | Ojeswini Jawale / Nilesh Yeole |

We are deploying the frontend configuration using the following YAML file for permanent redirection.

**apiVersion: networking.gke.io/v1beta1**

**kind: FrontendConfig**

**metadata:**

**name: frontend-config**

**spec:**

**redirectToHttps:**

**enabled: true**

**responseCodeName: PERMANENT\_REDIRECT**

* We are creating an external load balancer with the same static Ingress IP, and we are generating an SSL certificate and attaching it to the domain name.
* We are mapping the domain name to the static IP and registering the domain name with GoDaddy.
* Users can access the system using only this domain URL.

