A Report On The Internship Programme

at

Wipro Ltd, Bengaluru

Name:Nikhil Sudhan J

Description: GitOps-style continuous delivery with Cloud Build

Collage: Kalasalingam Academy of Research and Education

Intern between 07-June-2023 to 05-July-2023

CONTENT

Introduction

Deploying Python web application for predicting mosquito breeding sites based on environmental variables. We containerized the application using Docker and deployed it on Google Kubernetes Engine (GKE) cluster. We adopted the **GitOps methodology** to manage our Kubernetes configuration and deployment process. Any changes to the application, such as updating the code or modifying the Kubernetes configuration files, were done through Git version control. The changes were automatically deployed to the GKE cluster using a continuous integration and continuous deployment (CI/CD) pipeline.

GitOps is a popular way to manage Kubernetes deployments, including those on Google Kubernetes Engine (GKE). With GitOps on GKE, Kubernetes clusters and the desired state of the application are defined in the Git repository. Any changes made to the repository trigger automated processes that synchronize the actual state of the cluster with the desired state.

Difference between Standard Cluster, Autopilot Cluster and Cloud Run:

Cloud Run: Fully managed serverless platform for stateless container deployment with automatic scaling based on request traffic.

GKE: Managed Kubernetes service offering greater control and flexibility for complex deployments.

GKE Autopilot: Enhanced managed Kubernetes service simplifying cluster management by abstracting low-level tasks while providing some configuration options for scaling and resource allocation.

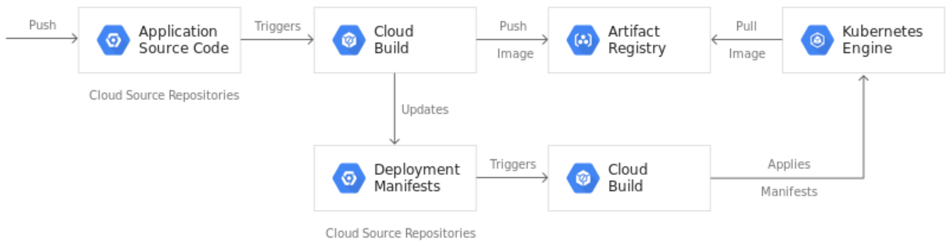
GCP resources involved:

Cloud Build is a cloud-based service provided by Google Cloud Platform (GCP) for automating and managing software build and deployment processes.

Artifact Registry in GKE is a managed service that allows you to store, manage, and deploy container images and other artifacts securely.

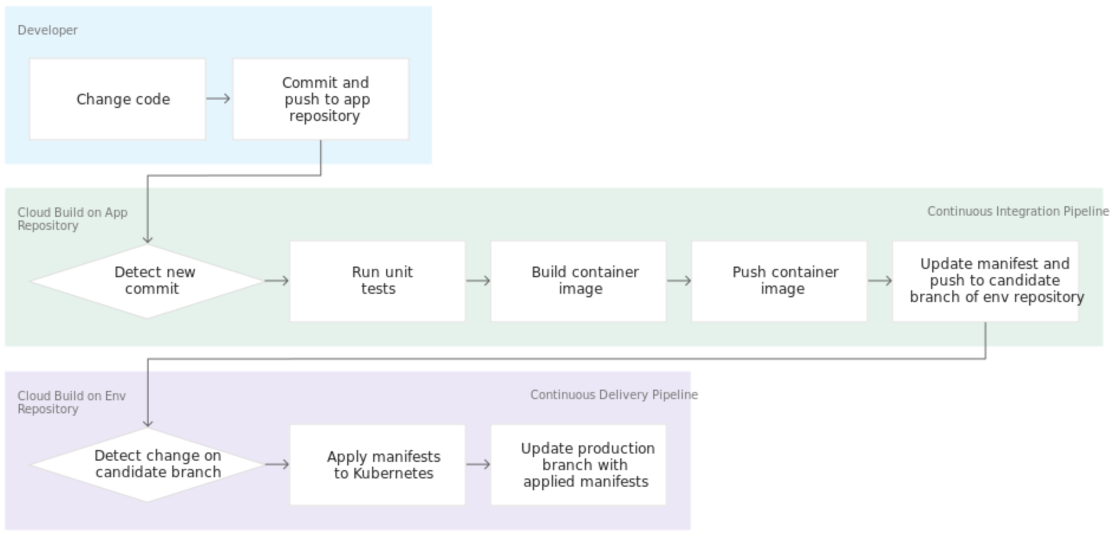
Kubernetes Engine enables users to deploy, manage, and scale containerized applications using Kubernetes orchestration.

Cloud Source Repositories in GKE refers to a version control system that allows you to store, manage, and collaborate on source code repositories for your GKE projects.



Starting with creating 2 cloud repository:

- **app repository**: contains the source code of the application itself
- **env repository**: contains the manifests for the Kubernetes Deployment


Developer
Change code
Commit and push to app repository
Cloud Build on App Repository
Continuous Integration Pipeline
Detect new commit
Run unit tests
Build container image
Push container image
Update manifest and push to candidate branch of env repository
Cloud Build on Env Repository
Continuous Delivery Pipeline
Detect change on candidate branch
Apply manifests to Kubernetes
Update production branch with applied manifests

Architecture of CI/CD pipeline

Kubernetes provides a powerful and flexible platform for deploying and managing containerized applications, offering scalability, resilience, automation, and a thriving ecosystem.

Kubernetes has significant advantages like:

Scalability: Kubernetes allows you to easily scale your applications up or down based on demand.

High availability: Kubernetes provides built-in mechanisms for ensuring high availability of applications.

Automation and Self-Healing: Kubernetes automates many aspects of application deployment and management, such as scheduling, health monitoring, and restarts.

Automatic rollouts and rollbacks: Kubernetes supports automated rollouts and rollbacks of application updates.

DevOps friendly: Kubernetes aligns well with DevOps practices by providing infrastructure-as-code capabilities. You can define your application infrastructure, including deployments, services, and networking, using declarative configuration files.

Infrastructure-as-Code: We define the desired state of your applications using YAML or JSON manifests, which makes it easier to manage and version control your deployments, promoting infrastructure-as-code practices.

Creating Clusters and Artifact Registry :

- Open Cloud Shell to execute the commands listed in this tutorial.
- If the `gcloud config get-value project` command does not return the ID of the project you selected, configure Cloud Shell to use your project

```
gcloud config set project [PROJECT_ID]
```

- In Cloud Shell, enable the required APIs.

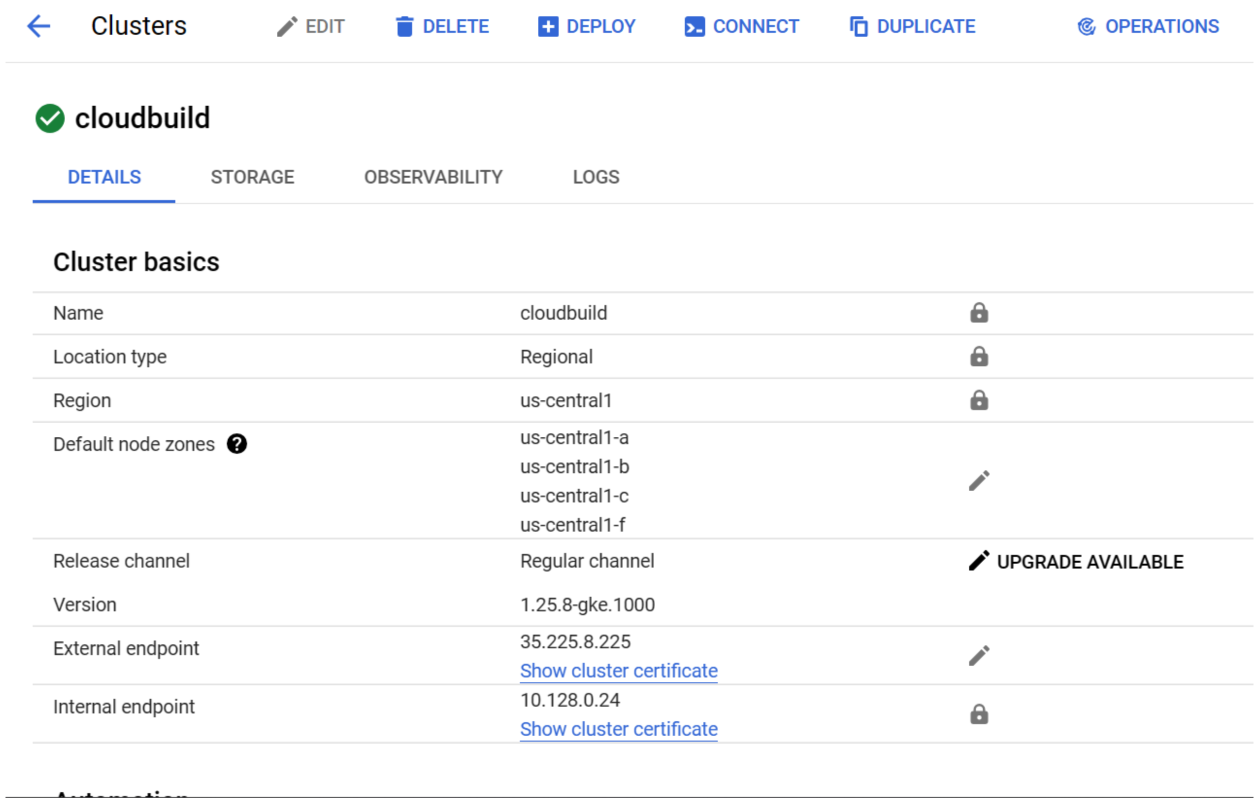
```
gcloud services enable container.googleapis.com \
  cloudbuild.googleapis.com \
  sourcerepo.googleapis.com \
  artifactregistry.googleapis.com
```

- Create an Artifact Registry repository named `my-repository` in the `us-central1` region to store your container images.

```
gcloud artifacts repositories create my-repository \
 --repository-format=docker \
 --location=us-central1
```

- In Cloud Shell, create a GKE cluster that you will use to deploy the application.

```
        gcloud container clusters create-auto hello-cloudbuild \
  --region us-central1
```



Clusters — EDIT · DELETE · DEPLOY · CONNECT · DUPLICATE · OPERATIONS

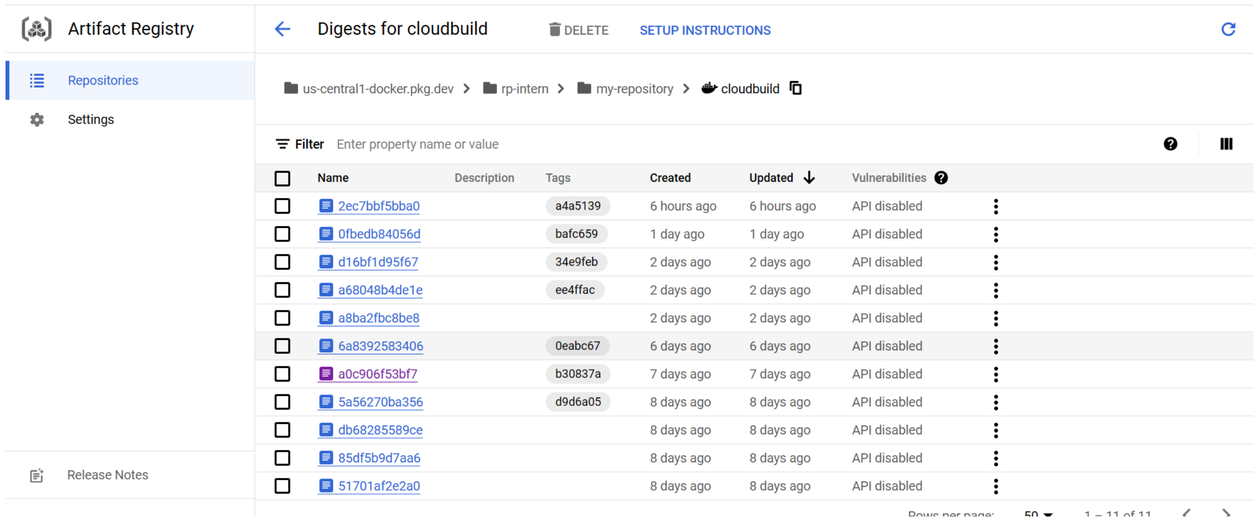
cloudbuild

DETAILS · STORAGE · OBSERVABILITY · LOGS

Cluster basics

Name	cloudbuild
Location type	Regional
Region	us-central1
Default node zones	us-central1-a us-central1-b us-central1-c us-central1-f
Release channel	Regular channel — UPGRADE AVAILABLE
Version	1.25.8-gke.1000
External endpoint	35.225.8.225 Show cluster certificate
Internal endpoint	10.128.0.24 Show cluster certificate

Cluster Information



Artifact Registry — Repositories · Settings · Release Notes

Digests for cloudbuild — DELETE · SETUP INSTRUCTIONS

us-central1-docker.pkg.dev > rp-intern > my-repository > cloudbuild

Filter Enter property name or value

Name	Description	Tags	Created	Updated	Vulnerabilities
2ec7bbf5bba0		a4a5139	6 hours ago	6 hours ago	API disabled
0fbedb84056d		bafc659	1 day ago	1 day ago	API disabled
d16bf1d95f67		34e9feb	2 days ago	2 days ago	API disabled
a68048b4de1e		ee4ffac	2 days ago	2 days ago	API disabled
a8ba2fbc8be8			2 days ago	2 days ago	API disabled
6a8392583406		0eabc67	6 days ago	6 days ago	API disabled
a0c906f53bf7		b30837a	7 days ago	7 days ago	API disabled
5a56270ba356		d9d6a05	8 days ago	8 days ago	API disabled
db68285589ce			8 days ago	8 days ago	API disabled
85df5b9d7aa6			8 days ago	8 days ago	API disabled
51701af2e2a0			8 days ago	8 days ago	API disabled

Rows per page: 50 1 – 11 of 11

Artifact Registry

Creating the Git repositories in Cloud Source Repositories

In this section, you create the two Git repositories (*app* and *env*) used in this tutorial, and initialize the *app* one with some sample code.

1. In Cloud Shell, create the two Git repositories.

   ```
   gcloud source repos create hello-cloudbuild-app

   gcloud source repos create hello-cloudbuild-env
   ```

2. Clone the sample code from GitHub.

   ```
   cd ~

   git clone link
   ```

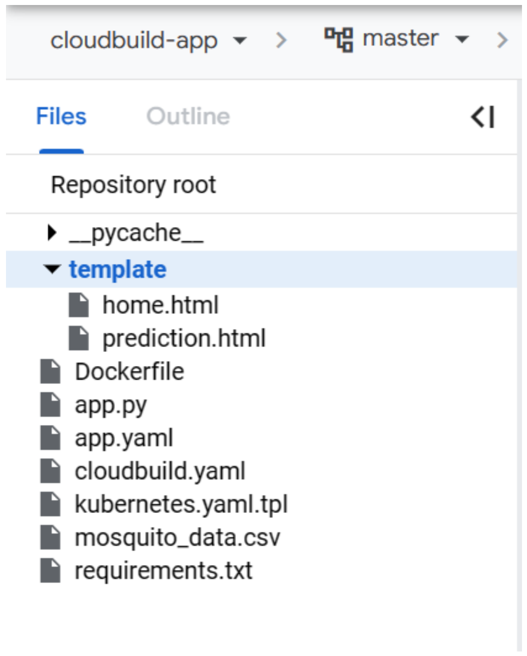
3. Configure Cloud Source Repositories as a remote.

   ```
   cd ~/folder

   PROJECT_ID=$(gcloud config get-value project)

   git remote add google \

     "https://source.developers.google.com/p/${PROJECT_ID}/r/name of the folder"
   ```

   Output:

   ```
   Your active configuration is: [cloudshell-11281]
   ```

The Project Folder Structure In Tree View



```
__pycache__
template
    home.html
    prediction.html
Dockerfile
app.py
app.yaml
cloudbuild.yaml
kubernetes.yaml.tpl
mosquito_data.csv
requirements.txt
```

Adding code to your repository

Provide your authentication credentials:

$gcloud init

#Clone this repository to a local Git repository:

$gcloud source repos clone hello-cloudbuildv --project=rp-intern

#Switch to your new local Git repository

$cd hello-cloudbuild

Granting Cloud Build access to GKE

To deploy the application in your Kubernetes cluster, Cloud Build needs the Kubernetes Engine Developer Identity and Access Management Role.

In Cloud Shell, execute the following command:

```
PROJECT_NUMBER="$(gcloud projects describe ${PROJECT_ID}
--format='get(projectNumber)')"
gcloud projects add-iam-policy-binding ${PROJECT_NUMBER} \

--member=serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.c
om \
    --role=roles/container.developer
```

Creating a CI/CD Pipeline

Dockerfile can create a container image with Cloud Build and store it in the Artifact Registry.Artifact Registry provides a single location for storing and managing your packages and Docker container images..

- After the build finishes, verify that your new container image is available in the Artifact Registry.

app/Dockerfile

```
FROM python:3.7-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . .
CMD ["python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

FROM python:3.7-slim: Specifies the base image for the Docker image. In this case, it uses the official Python 3.7 slim image as the base.

WORKDIR /app: Sets the working directory inside the Docker image to /app, where the application code and files will be placed.
COPY requirements.txt requirements.txt: Copies the requirements.txt file from the local directory to the /app directory in the Docker image.

RUN pip3 install -r requirements.txt: Runs the command pip3 install -r requirements.txt inside the Docker image, installing the Python dependencies specified in requirements.txt.

COPY . .: Copies all the files and directories from the local directory to the /app directory in the Docker image. This includes the Flask application code, templates, and any other necessary files.

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]: Specifies the default command to run when the Docker container starts.0.0.0.0 to allow external connections. This command starts the Flask application inside the container.

app/cloudbuild.yaml

```
# [START cloudbuild]
steps:
# This step runs the unit tests on the app
- name: 'python:3.7-slim'
id: Test
entrypoint: /bin/sh
args:
- -c
- 'pip install -r requirements.txt'
# This step builds the container image.
- name: 'gcr.io/cloud-builders/docker'
id: Build
args:
- 'build'
- '-t'
- 'us-central1-docker.pkg.dev/$PROJECT_ID/repo/container:$SHORT_SHA'
- '.'
# This step pushes the image to Artifact Registry
# The PROJECT_ID and SHORT_SHA variables are automatically
# replaced by Cloud Build.
- name: 'gcr.io/cloud-builders/docker'
id: Push
args:
- 'push'
-
'us-central1-docker.pkg.dev/$PROJECT_ID/repo/container:$SHORT_SHA'
# [END cloudbuild]
# [START cloudbuild-trigger-cd]
# This step clones the hello-cloudbuild-env repository
- name: 'gcr.io/cloud-builders/gcloud'
id: Clone env repository
entrypoint: /bin/sh
args:
- '-c'
- |
gcloud source repos clone hello-cloudbuild-env && \
cd hello-cloudbuild-env && \
git checkout candidate && \
git config user.email $(gcloud auth list --filter=status:ACTIVE
--format='value(account)')
# This step generates the new manifest
- name: 'gcr.io/cloud-builders/gcloud'
id: Generate manifest
entrypoint: /bin/sh
args:
- '-c'
- |
sed "s/GOOGLE_CLOUD_PROJECT/${PROJECT_ID}/g" kubernetes.yaml.tpl |
\
sed "s/COMMIT_SHA/${SHORT_SHA}/g" >
hello-cloudbuild-env/kubernetes.yaml
# This step pushes the manifest back to hello-cloudbuild-env
- name: 'gcr.io/cloud-builders/gcloud'
id: Push manifest
entrypoint: /bin/sh
args:
- '-c'
- |
set -x && \
cd hello-cloudbuild-env && \
git add kubernetes.yaml && \
git commit -m "Deploying image
us-central1-docker.pkg.dev/$PROJECT_ID/my-repository/cloudbuild:${
SHORT_SHA}
Built from commit ${COMMIT_SHA} of repository cloudbuild-app
Author: $(git log --format='%an <%ae>' -n 1 HEAD)" && \
git push origin candidate
```

Each step in the Cloud Build configuration performs specific actions such as testing, building the container image, pushing it to Artifact Registry, and updating the Kubernetes manifest file in a separate repository.

Step 1: Test

Uses a Python 3.7 slim image to run unit tests on the app.Installs the dependencies specified in requirements.txt. Executed only if testapp.py is available.

Step 2: Build

Uses the Docker builder image to build the container image.Tags the image with the format us-central1-docker.pkg.dev/$PROJECT_ID/repo/container:$SHORT_SHA.Builds the image using the current directory as the build context.

Step 3: Push

Uses the Docker builder image to push the container image to Artifact Registry.

Step 4: Clone env repository

Clones the hello-cloudbuild-env repository.Switches to the candidate branch.

Step 5: Push manifest

Pushes the updated manifest file back to the hello-cloudbuild-env repository.Adds the modified kubernetes.yaml file, commits with a message specifying the image and commit details, and pushes the changes to the candidate branch.

Process of Creating a trigger

1. Click **Create trigger**.
2. Fill out the following options:
    - In the **Name** field, type `hello-cloudbuild`.
    - Under **Event**, select **Push to a branch**.
    - Under **Source**, select `cloudbuild-app` as your **Repository** and `^master$` as your **Branch**.
    - Under **Build configuration**, select **Cloud Build configuration file**.
    - In the **Cloud Build configuration file location** field, type `cloudbuild.yaml` after the `/`.
3. Click **Create** to save your build trigger.

app/kubernetes.yaml.tpl

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: cluster
labels:
app: cluster
spec:
replicas: 1
selector:
matchLabels:
app: cluster
template:
metadata:
labels:
app: cluster
spec:
containers:
- name: cluster
image:
us-central1-docker.pkg.dev/rp-intern/repo/container:COMMIT_SHA
ports:
- containerPort: 5000
---
kind: Service
apiVersion: v1
metadata:
name: cluster
spec:
selector:
```

```
app: cluster
ports:
- protocol: TCP
port: 80
targetPort: 8080
type: LoadBalancer
```

Dockerfile sets up the container environment, installs the required dependencies, copies the application code and files into the image, and defines the command to run the Flask application when the container is launched.

Enhancements in code

- Switched off the debug mode in flask application to ensure better security, performance, and stability in a production environment.
- Specifically mentioning port number in flask application for networking and service discovery process.

Creating the continuous deployment pipeline

cloudbuild.yaml ========> Build the container Image =====> push it to Artifact Registry
(pipeline's configuration)

GitOps workflow for deploying a new version of an application to a Kubernetes cluster and syncing the manifest changes to the production branch :

env/clould-delivery.yaml

```
steps:
# This step deploys the new version of our container image
# in the hello-cloudbuild Kubernetes Engine cluster.
- name: 'gcr.io/cloud-builders/kubectl'
  id: Deploy
  args:
  - 'apply'
  - '-f'
  - 'kubernetes.yaml'
  env:
```

```
  - 'CLOUDSDK_COMPUTE_REGION=us-central1'
  - 'CLOUDSDK_CONTAINER_CLUSTER=hello-cloudbuild'

# This step copies the applied manifest to the production branch
# The COMMIT_SHA variable is automatically
# replaced by Cloud Build.
- name: 'gcr.io/cloud-builders/git'
  id: Copy to production branch
  entrypoint: /bin/sh
  args:
  - '-c'
  - |
    set -x && \
    # Configure Git to create commits with Cloud Build's service account
    git config user.email $(gcloud auth list --filter=status:ACTIVE
--format='value(account)') && \
    # Switch to the production branch and copy the kubernetes.yaml file from the
candidate branch
    git fetch origin production && git checkout production && \
    git checkout $COMMIT_SHA kubernetes.yaml && \
    # Commit the kubernetes.yaml file with a descriptive commit message
    git commit -m "Manifest from commit $COMMIT_SHA
    $(git log --format=%B -n 1 $COMMIT_SHA)" && \
    # Push the changes back to Cloud Source Repository
    git push origin production
```

Step 1: Deploy the new container image to the Kubernetes Engine cluster:

Uses the kubectl command from the gcr.io/cloud-builders/kubectl container image.
Applies the Kubernetes manifest file (kubernetes.yaml) to the cluster using kubectl apply.

Step 2: Copy the applied manifest to the production branch:

Fetches the production branch and switches to it.
Checks out the kubernetes.yaml file from the candidate branch at the specified COMMIT_SHA.
Commits the kubernetes.yaml file with a descriptive message that includes the commit information.Pushes the changes back to the Cloud Source Repository in the production branch. Make sure the production branch is created.

Creating trigger for Environment file

1. Click **Create trigger**.
2. Fill out the following options:
   - In the **Name** field, type `hello-cloudbuild-deploy`.
   - Under **Event**, select **Push to a branch**.
   - Under **Source**, select `cloudbuild-env` as your **Repository** and `^candidate$` as your **Branch**.
   - Under **Configuration**, select **Cloud Build configuration file (yaml or json)**.
   - In the **Cloud Build configuration file location** field, type `cloudbuild.yaml` after the `/`.
3. Click **Create**

Grant the Source Repository Writer IAM role to the Cloud Build service account for the *hello-cloudbuild-env* repository.

```
PROJECT_NUMBER="$(gcloud projects describe ${PROJECT_ID} \

    --format='get(projectNumber)')"

cat >/tmp/hello-cloudbuild-env-policy.yaml <<EOF
bindings:
- members:
  - serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com
  role: roles/source.writer
EOF
gcloud source repos set-iam-policy \
    hello-cloudbuild-env /tmp/hello-cloudbuild-env-policy.yaml
```

Roll back a target to an earlier release:

Determine the commit hash or tag of the desired previous version in your Git repository.

```
git log #To see a list of all commits and their corresponding commit hashes
```

Checkout the specific commit or tag locally on your development machine by running:

```
git checkout <commit-hash-or-tag>
```

Commit and push the changes to the Git repository, ensuring you're on the desired branch

```
git commit --amend --reword "Roll back to previous version”

git push origin master
```

By following these steps, you can use the git command with the --reword flag to modify the commit message and roll back to a previous version in GKE without creating a new branch.

Kubectl basic commands:

```
kubectl get pods # List all pods in the namespace

kubectl describe pod pod-name #Describe pod name

kubectl logs pod-name. #Logs of process inside the pod

kubectl get events #logs of running event

kubectl get svc #list the load balancer and cluster info

Kubectl exec -it podname —- /bin/sh #To access the container
```

Troubleshooting References

Troubleshooting #1 - Pod is not running

Check the cloudbuild.yaml file and kubernetes.yaml file location of artifact registry is the same.

Troubleshooting #2 - Port configuration

```
kubectl get svc #list the load balancer and cluster info

Curl *external IP*

HTML OUTPUT OF THE PROJECT

kubectl exec -it pod-name -- /bin/sh

ACCESS TO THE CONTAINER

Curl *external IP*
```

Above process ping the external ip. If nothing is happening, check with the port number from application code as well as dockerfile. External IP should be the same.

Troubleshooting #3 - Permission is denied

Permission is denied by the Source Repository Writer IAM role to the Cloud Build service account for the hello-cloudbuild-env repository. To grant an IAM role to the Cloud Build service run this command in cloud-build:

```
PROJECT_NUMBER="$(gcloud projects describe ${PROJECT_ID} \

    --format='get(projectNumber)')"

cat >/tmp/hello-cloudbuild-env-policy.yaml <<EOF

bindings:

- members:

  - serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com

  role: roles/source.writer

EOF

gcloud source repos set-iam-policy \

    hello-cloudbuild-env /tmp/hello-cloudbuild-env-policy.yaml
```

Important keywords to understand the kubernetes.yaml file

apiVersion: Specifies the API version being used. In this case, apps/v1 and v1 are used for the Deployment and Service, respectively.

containers: Defines the containers running in the Pod. It specifies the name, image, and any associated ports for the container.

image: Specifies the container image to use for the deployment. In this case, it references an image in the Google Artifact Registry, using the COMMIT_SHA variable to refer to a specific version.

ports: Specifies the ports to expose on the container.

type: Specifies the type of Service. In this case, LoadBalancer is used to expose the Service externally.

Reference Link:

https://cloud.google.com/kubernetes-engine/docs/tutorials/gitops-cloud-build

https://cloud.google.com/kubernetes-engine/docs/resources/autopilot-standard-feature-comparison

https://devopsnet.com/2022/01/04/cloud-run-vs-gke-vs-gke-autopilot/