**Task 1 – Clone Git Repository**

**1-Create a folder for your projects**

mkdir -p ~/projects

cd ~/projects

**2-Clone the Microservices repository**

git clone https://github.com/sameh-Tawfiq/Microservices.git

**3-Navigate the repo**

cd Microservices

ls -la

**4-verify the remote link**

git remote -v

folder C:\Users\s.mostafa\projects\Microservices created

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Task 2 – Dockerize the Application**

**1-Clone the Microservice Repository**

**2.Open Docker Cloud.**

**3-Clone the repository again**

git clone https://github.com/sameh-Tawfiq/Microservices.git

cd Microservices

ls

**4-create the docker file**

FROM python:3.9-slim

WORKDIR /app

COPY . /app

RUN pip install --upgrade pip && pip install --no-cache-dir -r requirements.txt

EXPOSE 8080

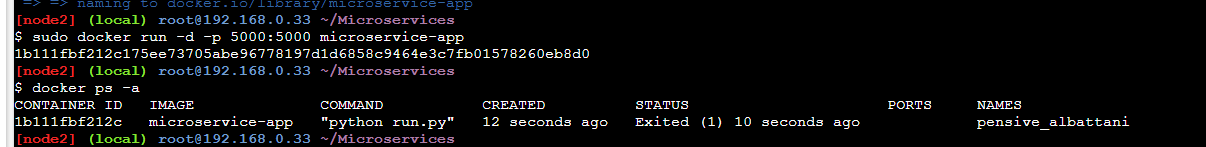
CMD ["python", "run.py"]

**A screenshot of a computer

AI-generated content may be incorrect.**

**A computer screen with blue text

AI-generated content may be incorrect.**

****

**5-Build & Run**

docker build -t microservice-app .

sudo docker run -d -p 8080:8080 microservice-app

**Task 3 – Terraform Setup for GKE**

**1-Initialize Terraform Environment**

mkdir terraform-gke-quickstart

cd terraform-gke-quickstart

gcloud auth application-default login

gcloud config set project gkepwc

**2-Define Terraform Files**

* provider.tf → Define provider and backend
* variables.tf → Input variables
* terraform.tfvars → Variable values
* main.tf → GKE cluster and node pool configuration

**3-Terraform Commands**

terraform init

terraform plan

terraform apply

**Step 1: Provision the GKE Cluster**

**terraform init**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Step 2: Review the Execution Plan**

**terraform plan**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Step 3: Apply the Configuration**

**terraform apply**

**A screen shot of a computer

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Task 4. Deploy the Microservice**

**1-Rebuild image with updated requirements.txt**

docker build -t us-west1-docker.pkg.dev/gkepwc/my-app-repo/my-app:v1.1 .

**2- Push to Artifact Registry**

docker push us-west1-docker.pkg.dev/gkepwc/my-app-repo/my-app:v1.1

**3-Apply Kubernetes deployment**

kubectl apply -f deployment.yaml

**4-Verify pod status**

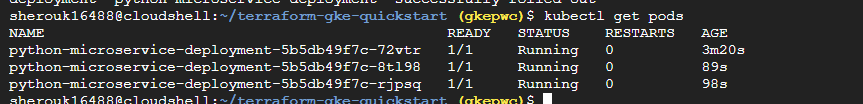
kubectl get pods

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A black screen with white text

AI-generated content may be incorrect.**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**Task 5 – Expose the Service Publicly**

**1-Create service.yaml**

A LoadBalancer service exposing the application externally

apiVersion: v1

kind: Service

metadata:

name: python-microservice-service

spec:

type: LoadBalancer

selector:

app: python-microservice

ports:

- port: 80

targetPort: 8080

**2-Apply and Verify**

kubectl apply -f service.yaml

kubectl get service python-microservice-service --watch

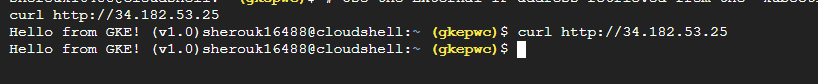
**External IP: 34.182.53.25**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**3-Test Connection:**

curl http://34.182.53.25

****

* **From console**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Task 6 – CI/CD Pipeline with Cloud Build**

**1-Create Cloud Build Configuration (cloudbuild.yaml)**

This pipeline automates build, push, and deploy steps:

1. Build Docker image
2. Push to Artifact Registry (my-app-repo)
3. Deploy to Cloud Run (python-web-app, region us-west1)

**2-Steps**

* Copy Dockerfile and cloudbuild.yaml into your local repo.
* Merge changes using GitHub Desktop.
* Fork repository to your GitHub account.
* in GCP → Cloud Build → Create Trigger, link to your repo path**.**

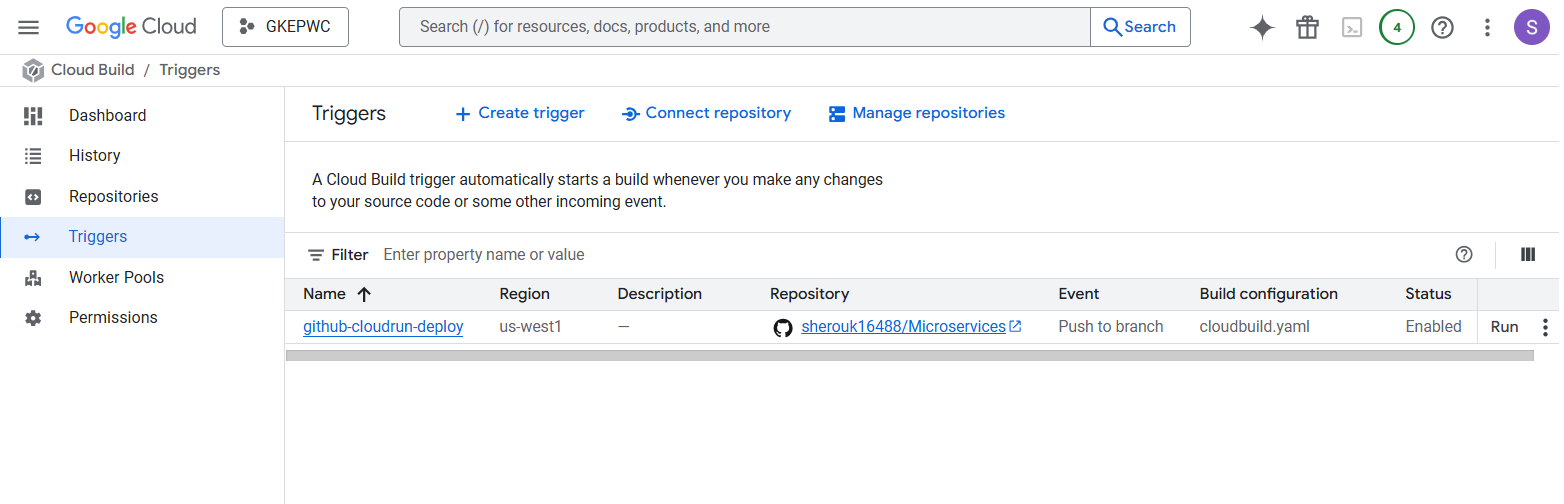
**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**3-Start to create the pipelines from gcp from cloud trigger and provide it with my path for git**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

**Task 7 – Monitoring Stack Implementation**

**1 – Clone the kube-prometheus Repository**

git clone https://github.com/prometheus-operator/kube-prometheus.git

cd kube-prometheus

A screen shot of a computer

AI-generated content may be incorrect.

**2 – Apply the Setup Manifests**

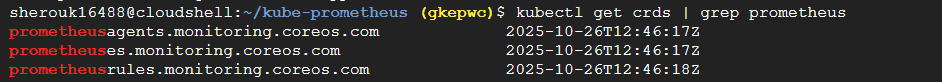
kubectl apply --server-side -f manifests/setup/

**A screen shot of a computer code

AI-generated content may be incorrect.**

* **Wait a few seconds for the CRDs to be created, then confirm**

kubectl get crds | grep Prometheus



**3 – Deploy the Monitoring Stack**

* **After the CRDs are installed, apply the main manifests to deploy all components.**

kubectl apply -f manifests/

A screen shot of a computer program

AI-generated content may be incorrect.

this deploys:

* Prometheus Operator
* Prometheus
* Alertmanager
* Grafana
* Kube State Metrics
* Node Exporter

**4 - Verify deployment**

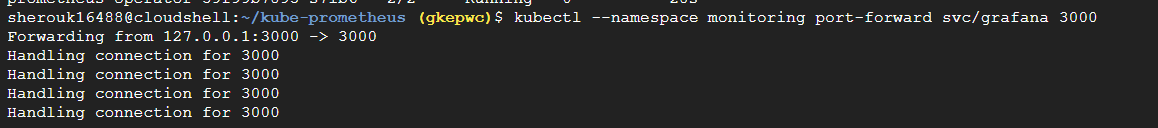
kubectl get pods -n monitoring

A screen shot of a computer

AI-generated content may be incorrect.

**5 -Access Grafana Dashboard**

kubectl --namespace monitoring port-forward svc/grafana 3000



A screenshot of a computer

AI-generated content may be incorrect.