**Project Setup**

* Create a GCP account
* After that, create a bucket under the project (Make sure to unclick the “Enforce public access prevention on this bucket” option. Otherwise, we cannot access this bucket folder)
* Then upload the csv file to that newly create bucket.
* Next step is to build the virtual environment
  + Before making any codes, we must create a virtual environment first
  + For that, we must create a folder in our local machine
  + Start the terminal of this folder and run below codes
    - python -m venv venv
    - source venv/bin/activate
    - Make below sub folders and python script
      * templates
      * src (Inside this folder, create a python file called “\_\_init\_\_.py”)
      * config (Inside this folder, create a python file called “\_\_init\_\_.py”)
      * utils (Inside this folder, create a python file called “\_\_init\_\_.py”)
      * pipeline (Inside this folder, create a python file called “\_\_init\_\_.py”)
      * artifacts
      * static
      * setup.py (Project management code will be here). Add below code to this setup.py file

A computer screen shot of a program code

AI-generated content may be incorrect.

* Run this code now: pip install -e .
* Then come to the “src” folder. Under that folder, create a python file called “logger.py” and paste below code

A computer screen shot of a program

AI-generated content may be incorrect.

* Then come to the “src” folder again. Under that folder, create a python file called “custom\_exception.py” and paste below code

A computer screen shot of a program

AI-generated content may be incorrect.

**Data Ingestion**

* In this stage, we are getting data from google bit bucket to vs code.
* First, we must create a service account. Before making the service account we must install google cloud CLI to our local machine. [Link](https://cloud.google.com/sdk/docs/install)
* After installing it, add “scikit-learn” and “google-cloud-storage” libraries into requirements file. Then run “pip install -e .” code
* Then navigate to GCP 🡪 IAM Admin 🡪 Service Accounts 🡪 create new service account 🡪 give a name to the service account 🡪 under role, select “storage admin” 🡪 under role, select “storage object viewer” 🡪 Done
* Go to buckets 🡪 select the bucket name 🡪 click three dots 🡪 Edit access 🡪 Add principal 🡪 Under new principal tab, add the service account 🡪 again give the same roles (storage admin and storage object viewer) and click save
* Again, navigate to the service account 🡪 Now we must create a key 🡪 Click three dots 🡪 manage keys 🡪 Add key 🡪 create new key 🡪 JSON 🡪 Create
* Come to VS code again 🡪 type below code in the terminal

export GOOGLE\_APPLICATION\_CREDENTIALS=/Users/thilina/Downloads/mlops-thilina-d275217160f5.json

* Create a file called “config.yaml” under config folder and paste below code

A screenshot of a computer

AI-generated content may be incorrect.

* Create a file called “paths\_config.py” under config folder and paste below code

A computer screen shot of a code

AI-generated content may be incorrect.

* Create a file called “common\_functions.py” under utils folder and paste below code. This python file will use to run yaml files

A screen shot of a computer program

AI-generated content may be incorrect.

* Create a file called “data\_ingestion.py” under src folder and paste below code.

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen with text and images

AI-generated content may be incorrect.

A computer screen with text on it

AI-generated content may be incorrect.

**Data Training**

**Save Processed Datasets:**

* Update the “paths\_config.py” file

A screen shot of a computer code

AI-generated content may be incorrect.

* Update the “config.yaml” file
* Create the “data\_preprocessing.py” file under src folder

**Experiment & Model Tracking**

* Add new lines to “config\_path.py” file

A screen shot of a computer

AI-generated content may be incorrect.

* Create a new file called “model\_params.py” under config folder

A screenshot of a computer program

AI-generated content may be incorrect.

* Create a new file called “model\_training.py” under src folder (it consist with mlflow as well)
* To run each src folder python file, we can use below code

cd /Users/thilina/Downloads/MLOps\ Projects/Project\_01

python -m src.model\_training

* If we want to see those results in mlflow dashboard, we can run “mlflow ui” code

**Data versioning and Code versioning**

* Create a new folder called “pipeline” and inside that make a file called “training\_pipeline.py”

A screen shot of a computer program

AI-generated content may be incorrect.

* Run

cd /Users/thilina/Downloads/MLOps\ Projects/Project\_01

python -m pipeline.training\_pipeline

* Make a GitHub repo and push the code, after creating “.gitignore” file (it will fill with files and folders that do not need to push to GitHub repo)

**User App Building**

**CI/CD Deployment**

1. **Setup Jenkins Container**

* Start the docker desktop engine
* Create a new folder called “custom\_jenkins” in the root directory. Under that, make a file called “Dockerfile” update it with the code
* Then run below codes,

>> cd custom\_jenkins/

>> docker login

>> docker build -t jenkins-dind .

>> docker images

>> docker run -d --name jenkins-dind --privileged -p 8080:8080 -p 50000:50000 -v //var/run/docker.sock:/var/run/docker.sock -v jenkins\_home:/var/jenkins\_home jenkins-dind

>> docker ps

>> docker logs jenkins-dind

* Copy the password in the log file
* Go to the chrome tab and type “localhost:8080” and paste the password in there
* Paste the password in the space 🡪 install suggested plugins 🡪 Give admin user (thilina97) and give any password (thilina@123) 🡪 save and continue
* Now we must install some dependencies in the Jenkins env
* Back to the VS code again. Then run below codes,

>> docker exec -u root -it jenkins-dind bash

>> apt update -y (update all the packages)

>> apt install -y python3 (it will install python3 on the Jenkins container)

>> python3 --version

>> ln -s /usr/bin/python3 /usr/bin/python (it will install python on the Jenkins container)

>> python --version

>> apt install -y python3-pip

>> apt install -y python3-venv (it will create the virtual env in Jenkins container)

>> exit

>> docker restart jenkins-dind (restart the container again)

1. **GitHub Integration**

* Navigate to GitHub account 🡪 Settings 🡪 Developer Settings 🡪 Personal access tokens 🡪 Tokens (Classic) 🡪 Generate new token (Classic) 🡪 Give a name 🡪 make to give repo, admin repo\_hook permissions 🡪 generate
* Navigate to Jenkins Dashboard 🡪 Manage Jenkins 🡪 Credentials 🡪 Click global 🡪 Add credentials 🡪 Add username (GitHub account name) 🡪 Password (the GitHub token) 🡪 give any ID 🡪 give same id for the description
* Navigate to Jenkins Dashboard again 🡪 click “New Item” 🡪 give a name 🡪 select **pipeline** 🡪 click “OK” 🡪 Under the Pipeline Section, select definition as “Pipeline script from SCM” 🡪 SCM (Source Code Management) should be git 🡪 Add the GitHub repo URL 🡪 Select the GitHub credentials 🡪 select the branch 🡪 Apply and Save
* Then go to pipeline syntax 🡪 under sample step select “Checkout from version control” 🡪 Give the URL of the GitHub repo and other necessary details 🡪 Generate pipeline Script 🡪 Copy the script code

1. **Dockerization of the Project / Create a venv in the Jenkins / Push image to GCR**

* Navigate to VS code again 🡪 Create a file called “Jenkinsfile” 🡪 Paste the Code
* Before running the stage related to “Building and pushing image to GCR”, we must install google cloud CLI in “custom\_jenkins” folder
* Run below code inside “custom\_jenkins” folder

>> docker exec -u root -it jenkins-dind bash

>> apt-get update

>> apt-get install -y curl apt-transport-https ca-certificates gnupg

>> curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

>> echo "deb https://packages.cloud.google.com/apt cloud-sdk main" | tee -a /etc/apt/sources.list.d/google-cloud-sdk.list

>> apt-get update && apt-get install -y google-cloud-sdk

>> gcloud –version

>> groupadd docker

>> usermod -aG docker jenkins

>> usermod -aG root jenkins

>> exit

>> docker restart jenkins-dind

* Refresh the Jenkins dashboard again 🡪 Manage Jenkins 🡪 Credentials 🡪 Under stores scoped to Jenkins 🡪 Global 🡪 Add credentials 🡪 Under the Kind, select secrete file 🡪 Upload the service key JSON file downloaded from google 🡪 Given any ID 🡪 Create
* Navigate to VS code again 🡪 Create a file called “Dockerfile” (This will use to docker our project) 🡪 Paste the Code
* Navigate to GCP 🡪 API Service 🡪 Select service as Google Container Registry API, Artifact Registry API, cloud Resource manager API 🡪 Enable
* Go to IAM Admin 🡪 select the service principal 🡪 hit edit icon 🡪 Add role as “Owner” 🡪 Save
* Back to VS code now (Important Note)
* environment {
* VENV\_DIR = 'venv'
* GCP\_PROJECT = "mlops-thilina"
* GCLOUD\_PATH = "/var/jenkins\_home/google-cloud-sdk/bin"
* }

This **GCP\_PROJECT** should be the ID of the GCP project

stage('Building and Pushing Docker Image to GCR'){

steps{

withCredentials([file(credentialsId: 'gcp-Jenkins-auth' , variable : 'GOOGLE\_APPLICATION\_CREDENTIALS')]){

script{

echo 'Building and Pushing Docker Image to GCR.............'

sh '''

This **credentialsId** should be the ID which store as Jenkins credentials

1. **Push GCR image to Google Run**

* Update the Jenkins file
* Push all the files to GitHub
* Navigate to GCP and paste below two lines

gcloud projects add-iam-policy-binding mlops-thilina \

--member="serviceAccount:mlops-thilina@mlops-thilina.iam.gserviceaccount.com" \

--role="roles/run.admin"

gcloud projects add-iam-policy-binding mlops-thilina \

--member="serviceAccount:mlops-thilina@mlops-thilina.iam.gserviceaccount.com" \

--role="roles/iam.serviceAccountUser"

* Start Jenkins workflow using build button

**Important**

If the jenkins environment full with space, we can run below two lines to make some space

>> docker system prune -af

>> docker volume prune -f

**Grafana & Prometheus**

**Create a GKE Cluster**

gcloud container clusters create ml-monitoring-cluster \

--zone us-central1-a \

--num-nodes 2 \

--machine-type e2-medium

gcloud container clusters get-credentials ml-monitoring-cluster --zone us-central1-a

kubectl get nodes

**Deploy Prometheus using Helm**

helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>

helm repo update

kubectl create namespace monitoring

**Deploy Grafana using Helm**

helm install prometheus prometheus-community/prometheus \

--namespace monitoring

kubectl get pods -n monitoring

helm install grafana prometheus-community/grafana \

--namespace monitoring \

--set adminUser=admin \

--set adminPassword=admin

kubectl get svc -n monitoring

kubectl patch svc grafana -n monitoring -p '{"spec": {"type": "LoadBalancer"}}'

**Configure Prometheus to scrape Cloud Run /metrics**

gcloud run services add-iam-policy-binding YOUR\_SERVICE\_NAME \

--member="allUsers" \

--role="roles/run.invoker"

* Replace “YOUR\_SERVICE\_NAME” by cloud run name

**Create a file prometheus-cloudrun-config.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: prometheus-config

namespace: monitoring

data:

prometheus.yml: |

global:

scrape\_interval: 15s

scrape\_configs:

- job\_name: 'cloudrun-ml-app'

metrics\_path: /metrics

scheme: https

static\_configs:

- targets: ['YOUR\_CLOUD\_RUN\_URL.run.app']

kubectl apply -f prometheus-cloudrun-config.yaml

**Add Prometheus as a Grafana Data Source**

1. Login to Grafana → **Configuration → Data Sources → Add Prometheus**.
2. URL: http://prometheus-server.monitoring.svc.cluster.local:80
3. Save & test connection.