

# Cloud Agnostic MLOps Pipeline Setup

Navneet  
Disha D Shetty  
Tejas Manu  
Gunasekar  
Darshan Balaji

## Contents

Objective .....	1
Proof of Concepts .....	1
1. MLOps .....	1
What is the use of MLOps? .....	1
Why do we need MLOps? .....	1
What are the benefits of MLOps? .....	1
What are the best practices for MLOps? .....	2
What is the difference between MLOps and DevOps? .....	2
2. Cloud agnostic.....	2
Benefits of being cloud agnostic.....	2
Challenges to Cloud Agnosticism.....	3
Should You Go Cloud Agnostic?.....	3
3. Docker .....	4
4. Kubernetes.....	4
What is Kubernetes?.....	4
Why use Kubernetes? .....	4
5. Kubeflow .....	4
Introduction .....	4
What is Kubeflow? .....	4
Introducing the ML workflow .....	4
Kubeflow interfaces .....	5
6. Git Hub Actions .....	5
Tech Stack .....	6
System requirement and Configuration .....	6
Microk8s →.....	6
Minikube → .....	6
Implementation .....	7
Challenges .....	13
Outcome .....	13
Future scope and improvements.....	14
Bibliography .....	14

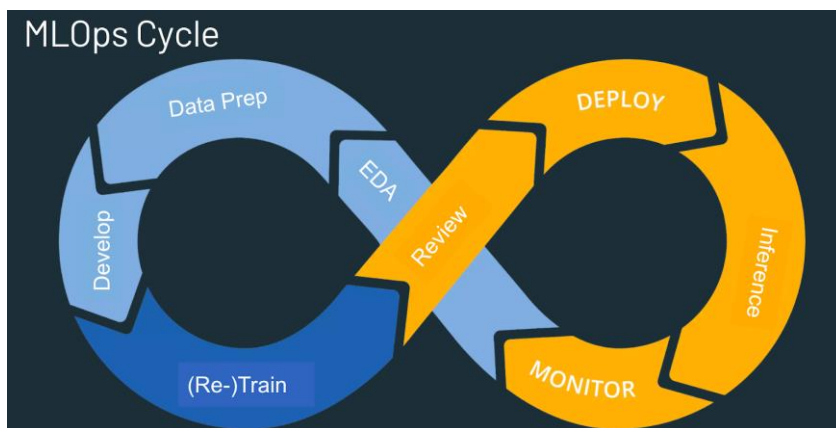
## Objective

MLOps is a core function of Machine Learning engineering, focused on streamlining the process of taking machine learning models to production, and then maintaining and monitoring them. This is usually done using a pipeline, but generally these pipelines are native to clouds. The main objective of this project is to setup a cloud Agnostic pipeline for machine learning models, which means our pipeline solution to this problem should work on every cloud platform.

## Proof of Concepts

### 1. MLOps

MLOps stands for Machine Learning Operations. MLOps is a core function of Machine Learning engineering, focused on streamlining the process of taking machine learning models to production, and then maintaining and monitoring them. MLOps is a collaborative function, often comprising data scientists, devops engineers, and IT.



### What is the use of MLOps?

MLOps is a useful approach for the creation and quality of machine learning and AI solutions. By adopting an MLOps approach, data scientists and machine learning engineers can collaborate and increase the pace of model development and production, by implementing continuous integration and deployment (CI/CD) practices with proper monitoring, validation, and governance of ML models.

### Why do we need MLOps?

Productionizing machine learning is difficult. The machine learning lifecycle consists of many complex components such as data ingest, data prep, model training, model tuning, model deployment, model monitoring, explain ability, and much more. It also requires collaboration and hand-offs across teams, from Data Engineering to Data Science to ML Engineering. Naturally, it requires stringent operational rigor to keep all these processes synchronous and working in tandem. MLOps encompasses the experimentation, iteration, and continuous improvement of the machine learning lifecycle.

### What are the benefits of MLOps?

The primary benefits of MLOps are efficiency, scalability, and risk reduction. Efficiency: MLOps allows data teams to achieve faster model development, deliver higher quality ML models, and faster deployment and production. Scalability: MLOps also enables vast scalability and management where thousands of models can be overseen, controlled, managed, and monitored for continuous integration, continuous delivery, and continuous deployment. Specifically, MLOps provides reproducibility of ML pipelines, enabling more tightly-coupled collaboration across data teams, reducing conflict with devops and IT, and accelerating release velocity. Risk reduction: Machine learning models often need regulatory scrutiny and drift-check, and MLOps enables greater transparency and faster response to such requests and ensures greater compliance with an organizations or industry's policies.

## What are the best practices for MLOps?

The best practices for MLOps can be delineated by the stage at which MLOps principles are being applied.

**Exploratory data analysis (EDA)** - Iteratively explore, share, and prep data for the machine learning lifecycle by creating reproducible, editable, and shareable datasets, tables, and visualizations.

**Data Prep and Feature Engineering**- Iteratively transform, aggregate, and de-duplicate data to create refined features. Most importantly, make the features visible and shareable across data teams, leveraging a feature store.

**Model training and tuning** - Use popular open source libraries such as scikit-learn and hyperopt to train and improve model performance. As a simpler alternative, use automated machine learning tools such as AutoML to automatically perform trial runs and create reviewable and deployable code.

**Model review and governance**- Track model lineage, model versions, and manage model artifacts and transitions through their lifecycle. Discover, share, and collaborate across ML models with the help of an open source MLOps platform such as MLflow.

**Model inference and serving** - Manage the frequency of model refresh, inference request times and similar production-specifics in testing and QA. Use CI/CD tools such as repos and orchestrators (borrowing devops principles) to automate the pre-production pipeline.

**Model deployment and monitoring** - Automate permissions and cluster creation to productionize registered models. Enable REST API model endpoints.

**Automated model retraining** - Create alerts and automation to take corrective action In case of model drift due to differences in training

## What is the difference between MLOps and DevOps?

MLOps is a set of engineering practices specific to machine learning projects that borrow from the more widely adopted DevOps principles in software engineering. While DevOps brings a rapid, continuously iterative approach to shipping applications, MLOps borrows the same principles to take machine learning models to production. In both cases, the outcome is higher software quality, faster patching and releases, and higher customer satisfaction. and inference data.

## 2. Cloud agnostic

Being cloud agnostic refers to an organization's ability to migrate their primary production workloads to another cloud provider. This is more complicated than it sounds.

For instance, a high-throughput data processing application that's deployed to AWS and depends on Amazon Kinesis Data Streams (KDS) for its streaming data can't easily be migrated to another cloud provider, as that service is proprietary and unique to AWS. Significant portions of the architecture would have to be redesigned to become cloud agnostic. Conversely, an application that exists primarily in Docker containers could be deployed to multiple platforms as a cloud-agnostic solution without excessive boilerplate configuration, as most providers have multiple standardized options for container orchestration and deployment.

## Benefits of being cloud agnostic

Although a cloud-agnostic infrastructure requires a larger upfront investment in terms of design and planning, it yields several benefits that generally fall into two categories: choice and risk.

The benefit of choice empowers engineering organizations to make decisions about their infrastructure based on the functionality and offerings that best suit their needs. In the context of vendors, cloud-agnostic workloads mean an

organization possesses vendor independence and can migrate their workloads elsewhere in the face of price increase or other issues. The primary workloads can run on any platform with minimal overhead.

The benefit of risk refers to the way that cloud-agnostic architectures act as a kind of safety valve, allowing organizations to make a quick pivot to another platform if the need arises. If any one vendor experiences an outage, security compromise, or an overall degradation of performance, application workloads can be moved seamlessly.

Despite the obvious benefits, there are also challenges to consider before implementing a cloud-agnostic design.

### **Challenges to Cloud Agnosticism**

Some of the challenges discussed below might not be clear until you've started to implement a cloud-agnostic design. Of course, at that point, you'll have to switch to firefighting mode. Considering some of the primary challenges early on will allow for the development of a cloud-agnostic strategy that minimizes the impact of any issues that do arise.

#### *IAM*

Identity and Access Management (IAM) serves as both a general term and the exact name of the service offered by AWS. At a high level, IAM is a system meant to control access to cloud resources. It utilizes concepts like roles and policies to define specific access and authorization configurations, thus ensuring that users can only access the appropriate resources in the appropriate contexts.

#### *Operational Economy of Scale*

Going cloud agnostic could also potentially mean your organization misses out on the operational economies of scale offered by cloud providers. There are still a multitude of advantages to be gained by going with a cloud-native application versus sticking with your traditional, legacy infrastructure. But if you decide against a managed service, the provider won't be there to do the operational heavy lifting, removing the burden of SLAs and uptime from your in-house operations staff.

#### *Engineering Acumen*

Engineering acumen is a "people problem" that might not present itself until your engineering team has to scale their deployment—and headcount to match. If a cloud-agnostic design is a pivot from an existing single-provider architecture, suddenly your engineering staff only has, at best, 50% of the knowledge needed to effectively operate and develop your production software infrastructure. This has the potential to lead to additional costs in hiring, training, and potential turnover, as some engineers simply prefer to focus on one knowledge domain. In the meantime, operational excellence metrics like SLA could suffer.

### **Should You Go Cloud Agnostic?**

If your organization is willing to deal with the added complexity and upfront investment of developing a cloud-agnostic strategy, you're likely to see benefits over a longer time horizon. You will be uniquely positioned to quickly mitigate the operational impact of any single provider outage or security issue, weathering the storm while non-agnostic competitors lose valuable traffic and trust.

On the other hand, your team might be better served taking advantage of provider-specific services, eschewing long-term benefits for fast iteration and a quicker time-to-value cycle. If you have a smaller development team without a dedicated operations staff, you can still immediately capitalize on the reduced operational overhead offered by managed services.

Ultimately, the decision to go cloud agnostic depends on a balance of short-term and long-term goals, as well as your engineering resources and plans. Smaller teams would do well to utilize managed services, allowing them to go to market as quickly as possible. Conversely, larger enterprise organizations can bring their considerable resources to bear, crafting a cloud-agnostic design that will provide longer-term benefits.

### 3. Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production. Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.

### 4. Kubernetes

#### What is Kubernetes?

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

#### Why use Kubernetes?

Keeping containerized apps up and running can be complex because they often involve many containers deployed across different machines. Kubernetes provides a way to schedule and deploy those containers—plus scale them to your desired state and manage their lifecycles. Use Kubernetes to implement your container-based applications in a portable, scalable and extensible way.

- Make workloads portable
- Scale containers easily
- Build more extensible apps

### 5. Kubeflow

#### Introduction

The Kubeflow project is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable, and scalable. Our goal is not to recreate other services, but to provide a straightforward way to deploy best-of-breed open-source systems for ML to diverse infrastructures. Anywhere you are running Kubernetes, you should be able to run Kubeflow.

#### What is Kubeflow?

Kubeflow is the machine learning toolkit. It is a free and open-source machine learning platform designed to enable using machine learning pipelines to orchestrate complicated workflows running on Kubernetes

To use Kubeflow, the basic workflow is:

- Download and run the Kubeflow deployment binary.
- Customize the resulting configuration files.
- Run the specified script to deploy your containers to your specific environment.

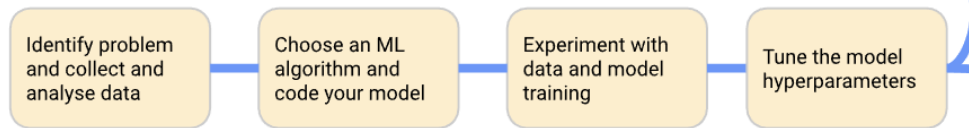
#### Introducing the ML workflow

When you develop and deploy an ML system, the ML workflow typically consists of several stages. Developing an ML system is an iterative process. You need to evaluate the output of various stages of the ML workflow and apply changes to the model and parameters when necessary to ensure the model keeps producing the results you need.

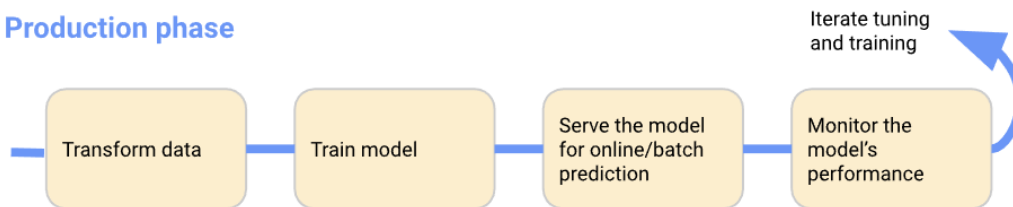


The arrow at the end of the workflow points back into the flow to indicate the iterative nature of the process:

### Experimental phase



### Production phase

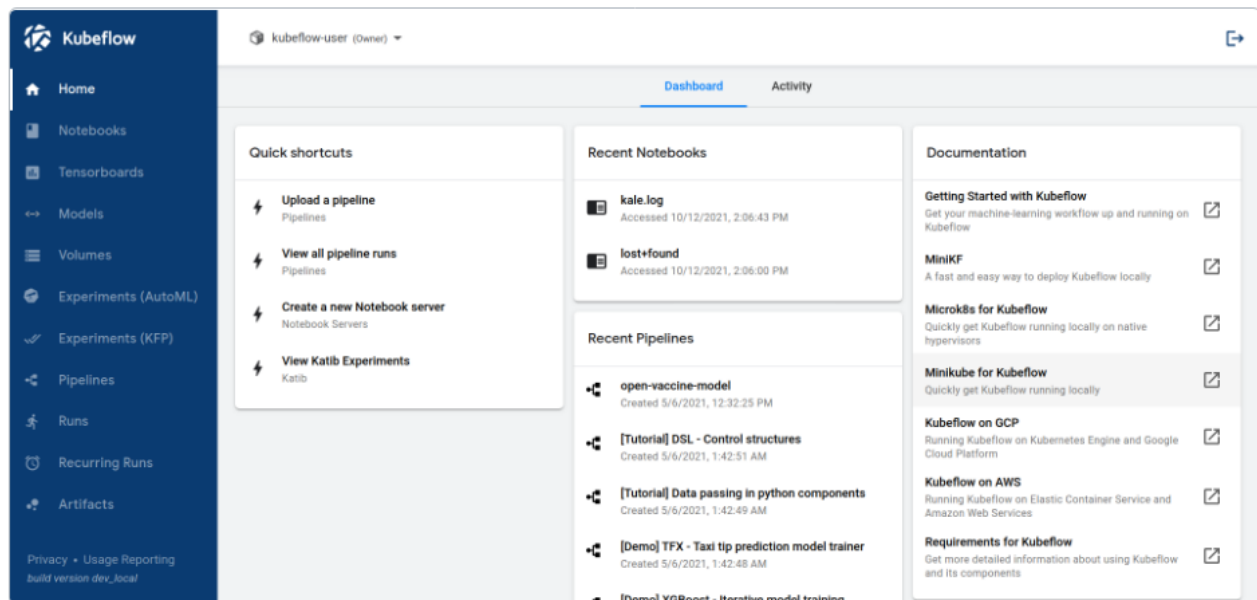


## Kubeflow interfaces

This section introduces the interfaces that you can use to interact with Kubeflow and to build and run your ML workflows on Kubeflow.

### Kubeflow user interface (UI)

The Kubeflow UI looks like this:



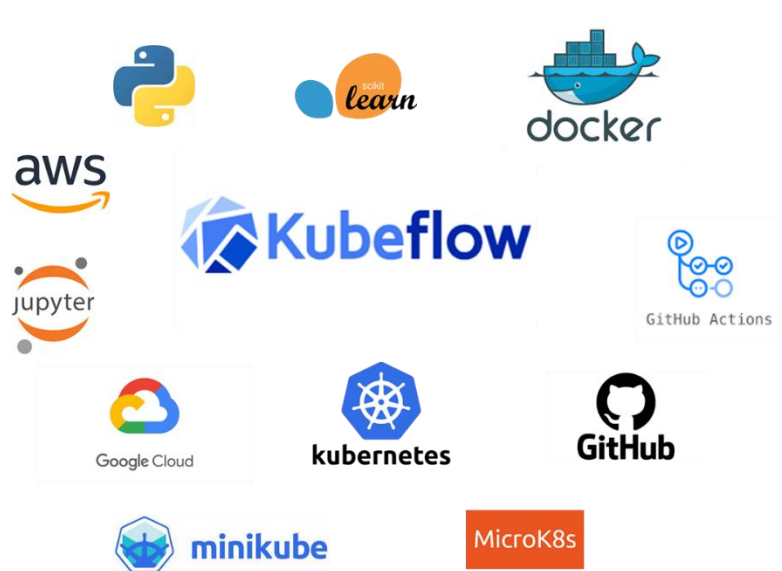
## 6. Git Hub Actions

GitHub Actions is a CI/CD (Continuous Integration/ Continuous Deployment) platform for automating the builds, test, and deployment process. Using GitHub actions, we can build and test every pull request in the repository using workflows or

push the merged pull requests to production with workflows. GitHub Actions allows you to conduct processes in response to the events in your repository.

When an event occurs in the repository, such as when a pull request is opened or an issue is raised, one can set up the GitHub Actions process to be triggered. The workflow includes one or more than one job that can be executed sequentially or concurrently.

## Tech Stack



## System requirement and Configuration

### Microk8s →

Operating system: Ubuntu 20.04

Kubernetes version: 1.20

system specification →

Ram: 32 GB

CPU: 8

### Minikube →

Operating system: Ubuntu 18.04

Kubernetes version: 1.21.24

system specification →

Ram: 32 GB

CPU: 8



## Implementation

Setting up of Kubeflow dashboard

Installing kubectl

```
sudo apt install curl
curl -LO https://dl.k8s.io/release/v1.21.14/bin/linux/amd64/kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
sudo kubectl version
```

Installing Docker driver

```
sudo apt install docker.io
sudo usermod -aG docker $USER && newgrp docker
```

Download and installing minikube

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Starting the minikube server

```
minikube start --cpus 6 --memory 10240 --disk-size=50g --kubernetes-version 1.21.14
```

Installing and configuring Kubeflow

```
set -e

KF_PATH="$HOME/.kubeflow"
rm -fr $KF_PATH
mkdir -p $KF_PATH
cd $KF_PATH
wget https://github.com/kubeflow/kfctl/releases/download/v1.2.0/kfctl_v1.2.0-0-gbc038f9_linux.tar.gz

tar -xvf kfctl_v1.2.0-0-gbc038f9_linux.tar.gz

export PATH=$PATH:$KF_PATH
export KF_NAME=my-kubeflow
export BASE_DIR=$KF_PATH
export KF_DIR=${BASE_DIR}/${KF_NAME}
export CONFIG_URI="https://raw.githubusercontent.com/kubeflow/manifests/v1.2-branch/kfdef/kfctl_k8s_istio.v1.2.0.yaml"
```

```
mkdir -p ${KF_DIR}
cd ${KF_DIR}
kfctl apply -V -f ${CONFIG_URI}
```

Checking if all the pods are up and running

```
watch kubectl get pod -n kubeflow
```

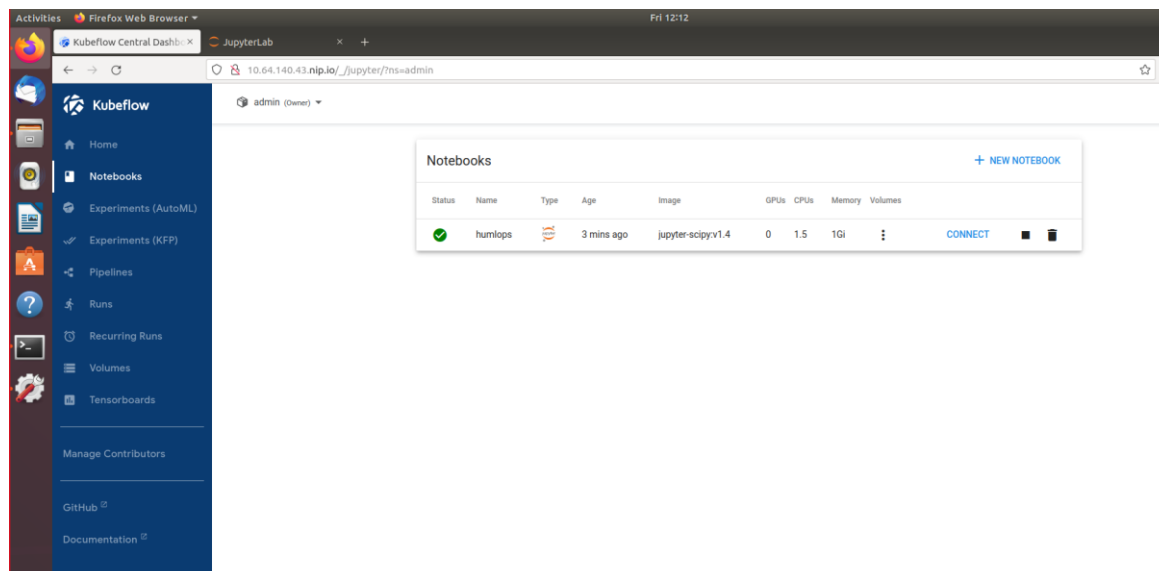
```
Every 2.0s: kubectl get pod -n kubeflow 1p-172-31-46-87: Mon Aug 1 16:50:07 2022
```

NAME	READY	STATUS	RESTARTS	AGE
admission-webhook-bootstrap-stateful-set-0	1/1	Running	0	11h
admission-webhook-deployment-5cd7dc96f5-hnjcz	1/1	Running	0	11h
application-controller-stateful-set-0	1/1	Running	0	11h
argo-ui-657cf69ff5-7s5s5	1/1	Running	0	11h
cache-deployer-deployment-5f4979f45-n86dp	2/2	Running	1	11h
cache-server-7859fd67f5-t9kxn	2/2	Running	0	11h
centraldashboard-86744cbb7b-6f8hf	1/1	Running	0	11h
jupyter-web-app-deployment-8486d5ffff-nbpfs	1/1	Running	0	11h
katib-controller-7fcc95676b-dflick	1/1	Running	1	11h
katib-db-manager-67867f5498-6csqr	1/1	Running	2	11h
katib-mysql-6b5d848bf5-svm4m	1/1	Running	0	11h
katib-ui-65dc4cf6f5-lznzb	1/1	Running	0	11h
kfserving-controller-manager-0	1/2	ImagePullBackOff	0	11h
kubeflow-pipelines-profile-controller-797fb44db9-vnjhc	1/1	Running	0	11h
metaccontroller-0	1/1	Running	0	11h
metadata-db-c65f4bc75-8gfsp	1/1	Running	0	11h
metadata-envoy-deployment-67bd5954c-j5tk4	1/1	Running	0	11h
metadata-grpc-deployment-577c67c96f-f8gtc	1/1	Running	0	11h
metadata-writer-756dbdd478-7w7sw	2/2	Running	0	11h
minio-54d995c97b-2mdlk	1/1	Running	0	11h
ml-pipeline-8d6749d9c-jl9wd	2/2	Running	2	11h
ml-pipeline-persistenceagent-d984c9585-sgzdj	2/2	Running	0	11h
ml-pipeline-scheduledworkflow-5ccf4c9fcc-7dqpj	2/2	Running	0	11h
ml-pipeline-ui-8ccbf585c-f8479	2/2	Running	0	11h
ml-pipeline-viewer-crd-56c68f6c85-gh5bc	2/2	Running	1	11h
ml-pipeline-visualizationserver-7446b96877-7d8wg	2/2	Running	0	11h
mpi-operator-d5bfb8489-4vmtt	0/1	CrashLoopBackOff	139	11h
mxnet-operator-7576d697d6-s6mwz	1/1	Running	0	11h
mysql-74f8f99bc8-dv2s9	2/2	Running	0	11h
notebook-controller-deployment-dd4c74b47-22w28	1/1	Running	0	11h
profiles-deployment-65f54cb5c4-pnnmg	2/2	Running	0	11h
pytorch-operator-847c8d55d8-7rh4j	1/1	Running	0	11h
seldon-controller-manager-6bf8b45656-1mz1f	1/1	Running	0	11h
spark-operators-sparkoperator-fdfbdf99-x9p6f	1/1	Running	0	11h
spartakus-volunteer-558f8bf4d7-sp996	1/1	Running	0	11h
tf-job-operator-58477797f8-p4tn8	1/1	Running	0	11h
workflow-controller-64fd7cffe5-ds8p2	1/1	Running	0	11h

To access Kubeflow dashboard we port forward ingressgateway to our desired port

```
kubectl port-forward --address 0.0.0.0 -n istio-system svc/istio-ingressgateway 8080:80
```

creating a Notebook server →



Implementation of ml pipeline components and their docker files →

pre-processing →

```

preprocess_data > preprocess.py > ...
1 from sklearn import datasets
2 from sklearn.model_selection import train_test_split
3 import numpy as np
4
5 def _preprocess_data():
6     X, y = datasets.load_boston(return_X_y=True)
7     X_train, X_test, y_train, y_test = train_test_split(X, y, test
8     np.save('x_train.npy', X_train)
9     np.save('x_test.npy', X_test)
10    np.save('y_train.npy', y_train)
11    np.save('y_test.npy', y_test)
12
13 if __name__ == '__main__':
14     print('Preprocessing data...')
15     _preprocess_data()

```

```

Dockerfile > FROM
1 FROM python:3.7-slim
2
3 WORKDIR /app
4
5 RUN pip install -U scikit-learn numpy
6
7 COPY preprocess.py ./preprocess.py
8
9 ENTRYPOINT [ "python", "preprocess.py" ]

```

training →

```

train > train.py > ...
1 import argparse
2 import joblib
3 import numpy as np
4 from sklearn.linear_model import SGDRegressor
5
6 def train_model(x_train, y_train):
7     x_train_data = np.load(x_train)
8     y_train_data = np.load(y_train)
9
10    model = SGDRegressor(verbose=1)
11    model.fit(x_train_data, y_train_data)
12
13    joblib.dump(model, 'model.pkl')
14
15 if __name__ == '__main__':
16     parser = argparse.ArgumentParser()
17     parser.add_argument('--x_train')
18     parser.add_argument('--y_train')
19     args = parser.parse_args()
20     train_model(args.x_train, args.y_train)

```

```

Dockerfile > FROM
1 FROM python:3.7-slim
2
3 WORKDIR /app
4
5 RUN pip install -U scikit-learn numpy
6
7 COPY train.py ./train.py
8
9 ENTRYPOINT [ "python", "train.py" ]

```

testing →

```

test > test.py > ...
1 import argparse
2 import joblib
3 import numpy as np
4 from sklearn.metrics import mean_squared_error
5
6 def test_model(x_test, y_test, model_path):
7     x_test_data = np.load(x_test)
8     y_test_data = np.load(y_test)
9
10    model = joblib.load(model_path)
11    y_pred = model.predict(x_test_data)
12
13    err = mean_squared_error(y_test_data, y_pred)
14
15    with open('output.txt', 'a') as f:
16        f.write(str(err))
17
18 if __name__ == '__main__':
19     parser = argparse.ArgumentParser()
20     parser.add_argument('--x_test')
21     parser.add_argument('--y_test')
22     parser.add_argument('--model')
23     args = parser.parse_args()
24     test_model(args.x_test, args.y_test, args.model)

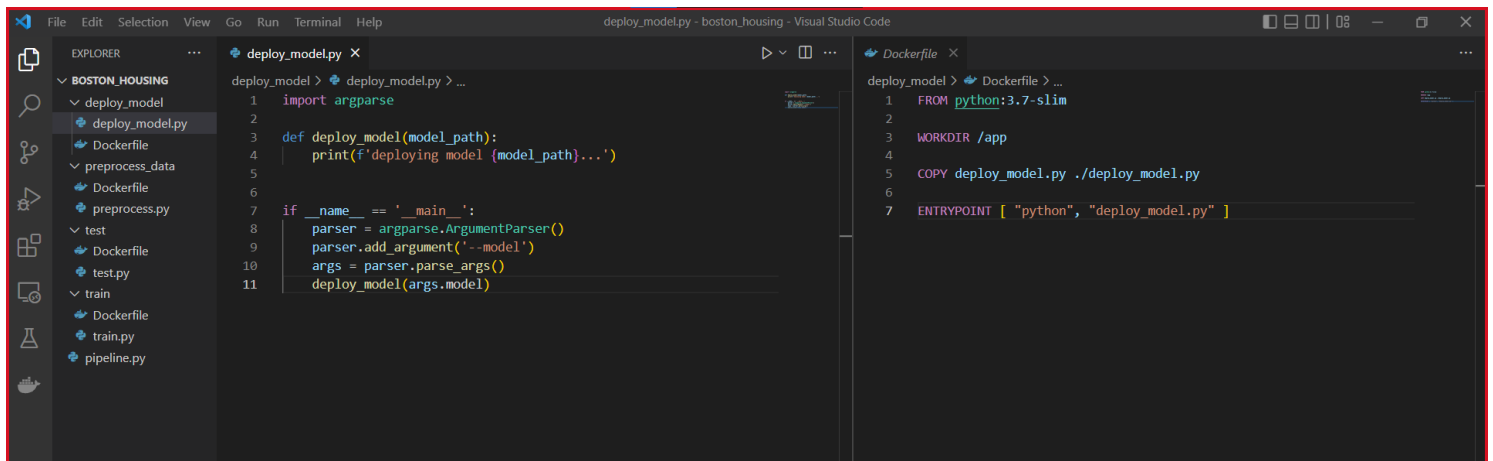
```

```

Dockerfile > FROM
1 FROM python:3.7-slim
2
3 WORKDIR /app
4
5 RUN pip install -U scikit-learn numpy
6
7 COPY test.py ./test.py
8
9 ENTRYPOINT [ "python", "test.py" ]

```

deploy →

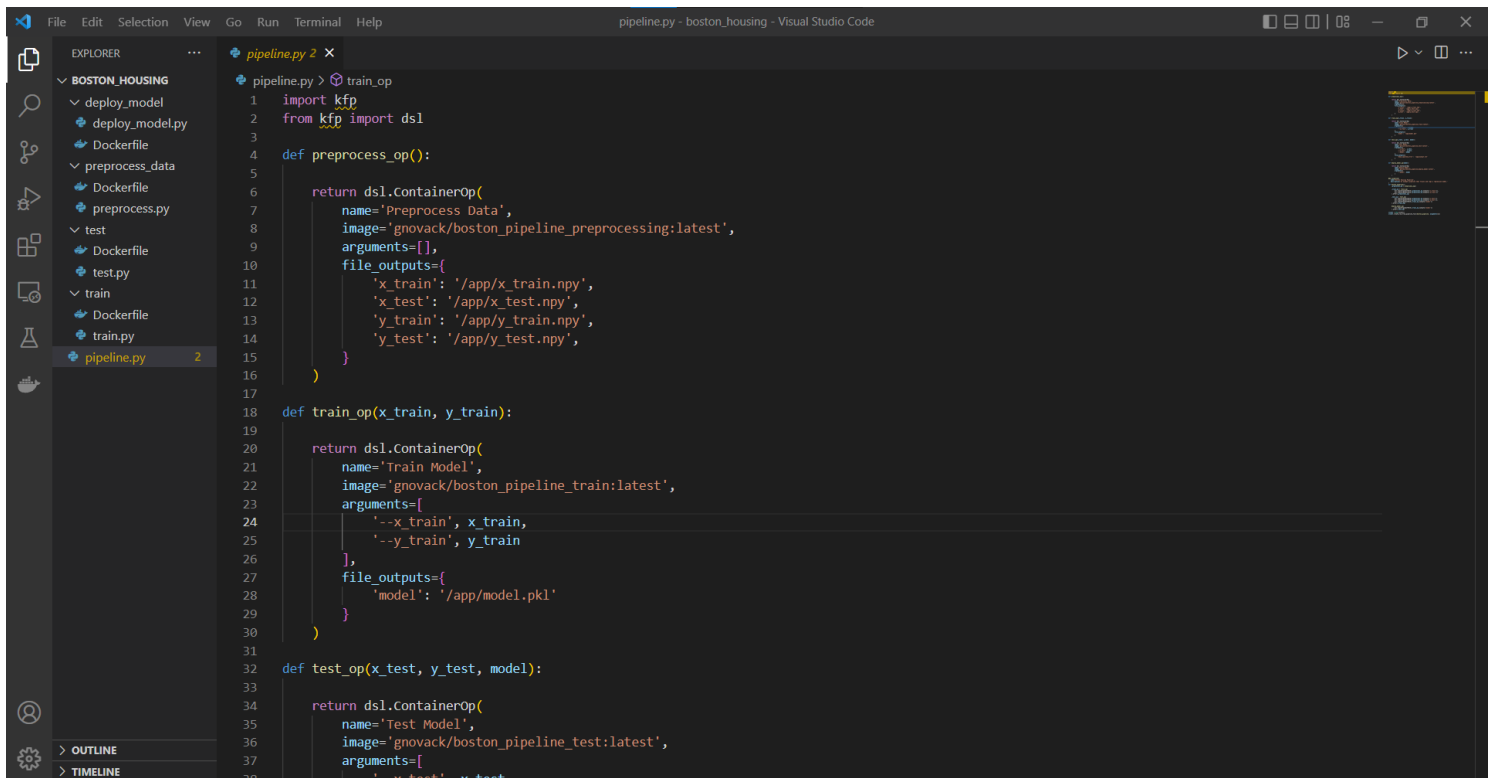


The screenshot shows the Visual Studio Code interface with two files open: `deploy_model.py` and `Dockerfile`. The `deploy_model.py` file contains a script that uses `argparse` to handle command-line arguments and a `deploy_model` function that prints the model path. The `Dockerfile` file defines a container image based on `python:3.7-slim`, sets the working directory to `/app`, copies the `deploy_model.py` file, and sets the entrypoint to run `python deploy_model.py`.

```
deploy_model.py
1 import argparse
2
3 def deploy_model(model_path):
4     print(f'deploying model {model_path}...')
5
6
7 if __name__ == '__main__':
8     parser = argparse.ArgumentParser()
9     parser.add_argument('--model')
10    args = parser.parse_args()
11    deploy_model(args.model)

Dockerfile
1 FROM python:3.7-slim
2
3 WORKDIR /app
4
5 COPY deploy_model.py ./deploy_model.py
6
7 ENTRYPOINT [ "python", "deploy_model.py" ]
```

code for pipeline →



The screenshot shows the Visual Studio Code interface with the `pipeline.py` file open. The script defines three functions: `preprocess_op`, `train_op`, and `test_op`, each using `dsl.ContainerOp` to define a container operation. The `preprocess_op` function takes no arguments and outputs training and testing data files. The `train_op` function takes training and testing data files as arguments and outputs a trained model file. The `test_op` function takes a testing data file and a trained model file as arguments.

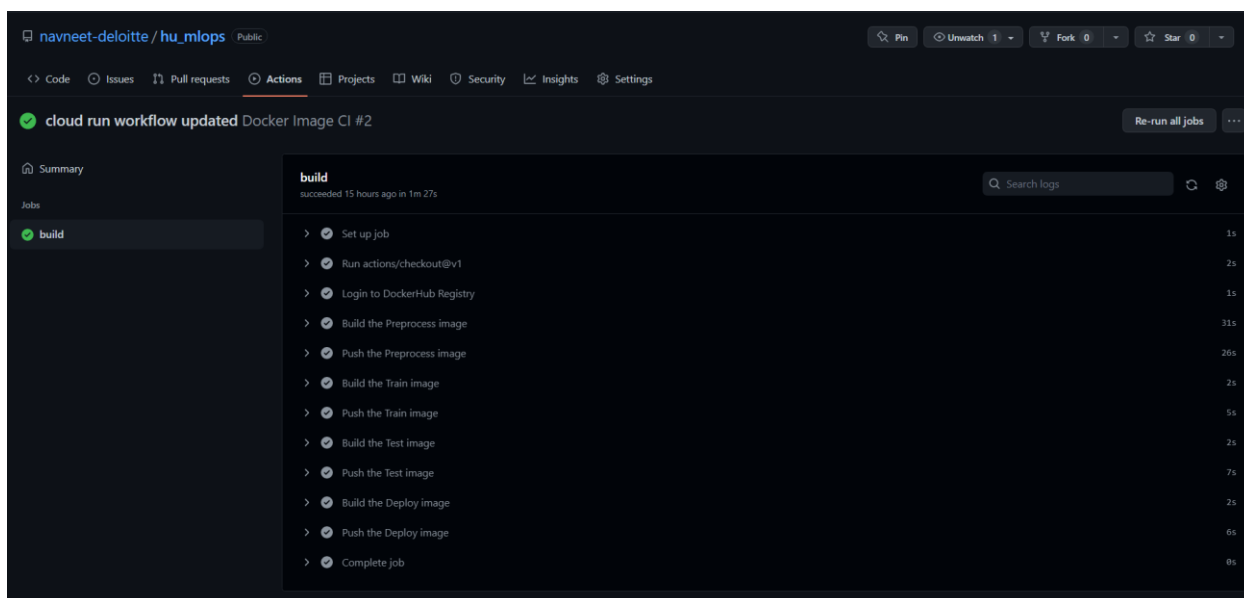
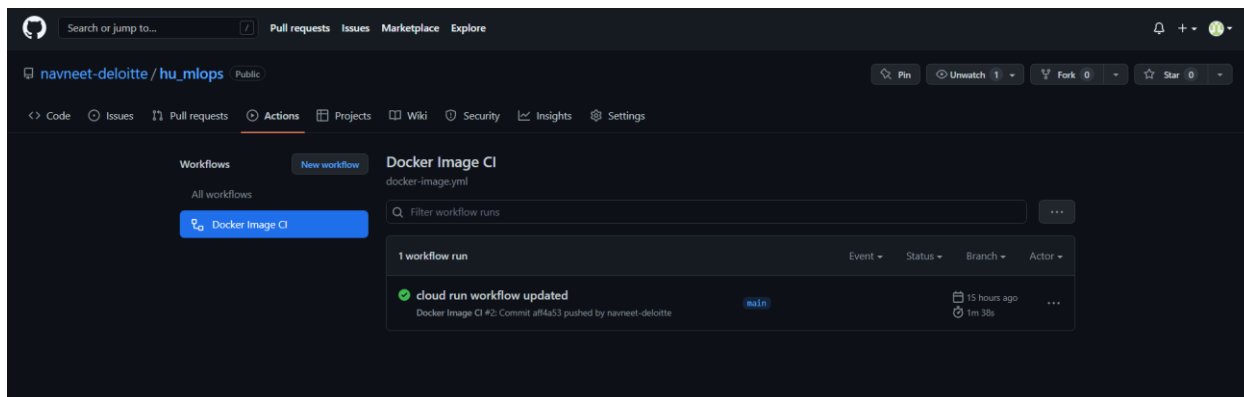
```
pipeline.py
1 import kfp
2 from kfp import dsl
3
4 def preprocess_op():
5
6     return dsl.ContainerOp(
7         name='Preprocess Data',
8         image='gnovack/boston_pipeline_preprocessing:latest',
9         arguments=[],
10        file_outputs={
11            'x_train': '/app/x_train.npy',
12            'x_test': '/app/x_test.npy',
13            'y_train': '/app/y_train.npy',
14            'y_test': '/app/y_test.npy',
15        }
16    )
17
18 def train_op(x_train, y_train):
19
20     return dsl.ContainerOp(
21         name='Train Model',
22         image='gnovack/boston_pipeline_train:latest',
23         arguments=[
24             '--x_train', x_train,
25             '--y_train', y_train
26         ],
27         file_outputs={
28             'model': '/app/model.pkl'
29         }
30     )
31
32 def test_op(x_test, y_test, model):
33
34     return dsl.ContainerOp(
35         name='Test Model',
36         image='gnovack/boston_pipeline_test:latest',
37         arguments=[
38             '--x_test', x_test,
```

building and pushing docker images using docker run →

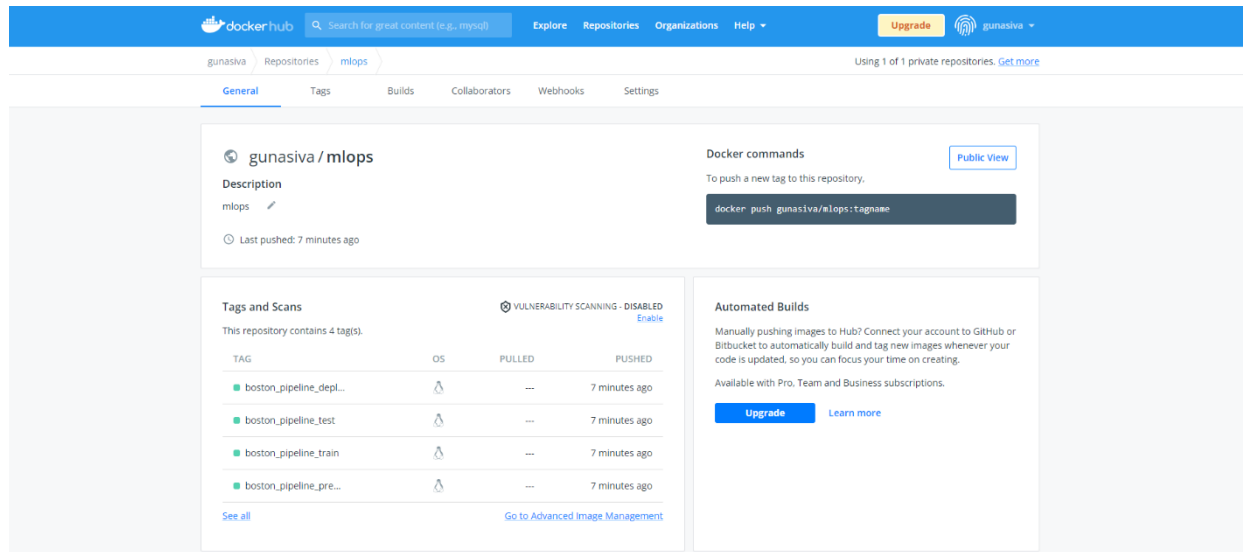
```

34 lines (29 sloc) | 1.24 KB
1  name: Docker Image CI
2
3  on:
4    push:
5      branches: [ "main" ]
6    pull_request:
7      branches: [ "main" ]
8
9  jobs:
10
11    build:
12
13      runs-on: ubuntu-latest
14
15      steps:
16      - uses: actions/checkout@v1
17      - name: Login to DockerHub Registry
18        run: echo ${ secrets.DOCKER_HUB_PASSWORD } | docker login -u ${ secrets.DOCKER_HUB_USERNAME } --password-stdin
19      - name: Build the Preprocess image
20        run: docker build ./boston_housing/preprocess_data --tag gunasiva/mlops:boston_pipeline_preprocessing
21      - name: Push the Preprocess image
22        run: docker push gunasiva/mlops:boston_pipeline_preprocessing
23      - name: Build the Train image
24        run: docker build ./boston_housing/train --tag gunasiva/mlops:boston_pipeline_train
25      - name: Push the Train image
26        run: docker push gunasiva/mlops:boston_pipeline_train
27      - name: Build the Test image
28        run: docker build ./boston_housing/test --tag gunasiva/mlops:boston_pipeline_test
29      - name: Push the Test image
30        run: docker push gunasiva/mlops:boston_pipeline_test
31      - name: Build the Deploy image
32        run: docker build ./boston_housing/deploy_model --tag gunasiva/mlops:boston_pipeline_deploy_model
33      - name: Push the Deploy image
34        run: docker push gunasiva/mlops:boston_pipeline_deploy_model

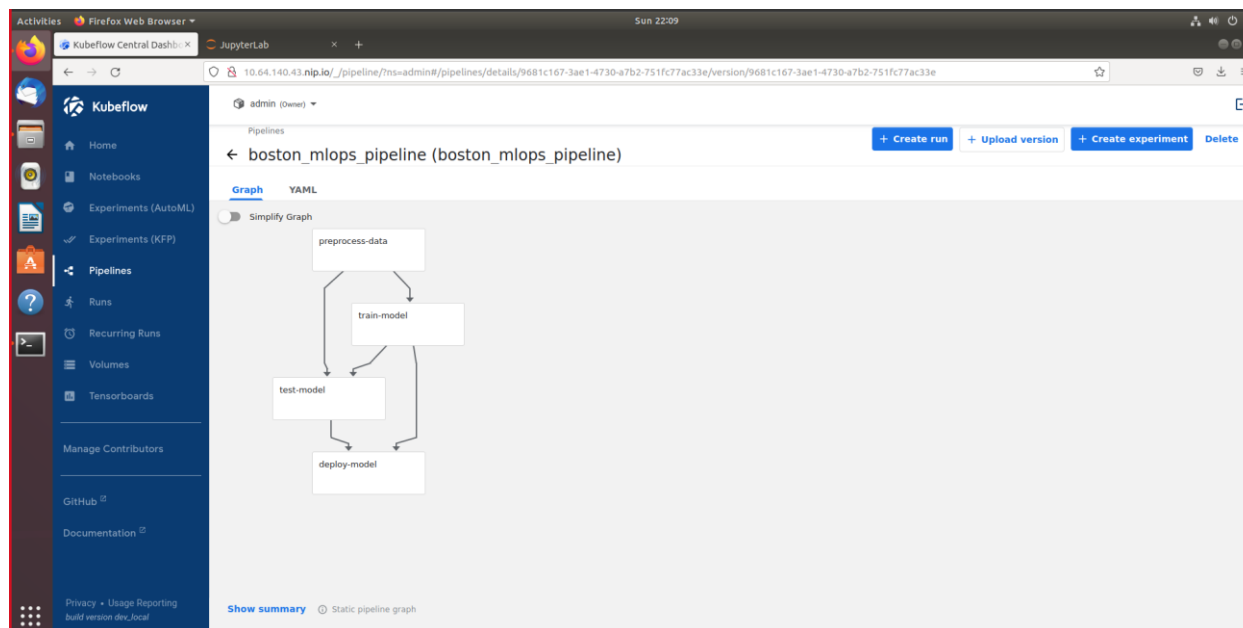
```



as soon as anything is pushed to the git hub repo, or anything is pulled a new image is build and pushed to the docker hub



creating a pipeline →







## Future scope and improvements

Here in our solution, we didn't use any cloud platform to deploy our model which can be configured with respect to a particular cloud platform.

complete automation of the pipeline, though we have automated the containerizing model but even the ml pipeline could be automated to run whenever a new version of images are available.

Only the automation of containerizing the model has been introduced in this project and can be further improvised by automating the pipeline run when a new version of the image is pushed into the repository.

The pipeline can be further used for deploying the model to any cloud platform by specifying its configuration required.

The model's performance can be monitored with the Prometheus and other platforms.

## Bibliography

<https://minikube.sigs.k8s.io/docs/start/>

<https://docs.docker.com/engine/swarm/secrets/>

<https://charmed-kubeflow.io/docs/install>

<https://microk8s.io/docs/getting-started>

<https://www.kubeflow.org/docs/components/pipelines/sdk/sdk-overview/>