Configuration Manual

MSc Research Project
Cloud Computing

Shraddha Bhosle
Student ID: x21177252

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shraddha Bhosle		
Student ID:	X21177252		
Programme:	Cloud Computing	**Year:**	2023
Module:	MSc Research Project		
Lecturer:	Rejwanul Haque		
Submission Due Date:	14/12/2023		
Project Title:	Orchestration and CI/CD automation using MLOps for Cloud-native container deployments		
Word Count:	**2982**	**Page Count: 20**	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shraddha Bhosle

Date: 14/12/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shraddha Bhosle
Student ID: x21177252

1 Introduction

This configuration manual is designed to provide comprehensive steering on putting in and managing the device advanced in the studies on "Orchestration and CI/CD Automation using MLOps for Cloud-native container Deployments." The guide pursuits to help customers in replicating the studies environment, configuring the important components, and understanding the operational dynamics of the device. It covers the entire spectrum of the setup, from preliminary environment configuration to deployment and maintenance.

The system in awareness integrates machine learning Operations (MLOps) with continuous Integration and continuous Deployment (CI/CD) practices, specially tailored for cloud-local field deployments. It incorporates a system studying software for phishing link detection, leveraging logistic regression and multinomial Naive Bayes models. The utility is containerized the use of Docker and deployed on Amazon web services (AWS) and Elastic Kubernetes service (EKS), with Kubeflow pipelines orchestrating the Machine learning workflows. The CI/CD pipeline is managed via GitHub actions, automating the deployment and integration methods.

2 System requirements

2.1 Hardware requirements:

- A pc with a minimum of 8GB RAM (16GB advocated for the most useful overall performance).
- At least 20GB of free disk area.
- Excessive-pace internet connection for cloud services get entry to and information dealing with.

2.2 Software program conditions:

- Operating System: Windows 10, macOS, or a Linux distribution (e.g., Ubuntu 18.04 or later).
- Programming Language: Python (version 3.6 or later), with pip for package control.
- Libraries and Frameworks: Key Python libraries which includes Pandas, NumPy, Scikit-study, TensorFlow, and NLTK for system gaining knowledge of and statistics processing.
- Containerization device: Docker, for growing and dealing with utility containers.
- Cloud Platform: A lively AWS account with get admission to to services like EKS and IAM (identification and access management).
- Kubernetes control: Kubeflow, for orchestrating device getting to know workflows in Kubernetes.

- CI/CD tools: GitHub account for repository management and GitHub actions for CI/CD pipeline setup.
- Text Editor/IDE: Favoured textual content editor or integrated development environment (IDE) like visible Studio Code, PyCharm, or Jupyter notebook for code development.

Those requirements make certain that users have the essential hardware and software to efficiently configure and make use of the system. users need to ensure that every one software additives are up to date to their present-day versions to keep away from compatibility problems.

3 Environment Setup

3.1 Setting up the development environment:

- Operating system preparation: Ensure that your running machine (windows, macOS, or Linux) is up to date.
- Python installation: install Python (model 3.6 or later) from the official Python website. confirm the setup by way of running python --version inside the command line.
- IDE/textual content Editor: deploy a desired IDE or textual content editor, along with visible Studio Code, PyCharm, or Jupyter Notebook, for writing and executing code.
- Docker installation: download and install Docker from the professional Docker website. Affirm the setup with docker --version.

3.2 Installing Libraries and Tools:

- Python Libraries: Install necessary Python libraries the use of pip. Run the subsequent instructions:
    - pip set up pandas numpy scikit-study nltk tensorflow
- Kubeflow and Kubernetes equipment: follow the installation guides for Kubeflow and Kubernetes equipment likeminded with AWS EKS from their respective respectable documentation.

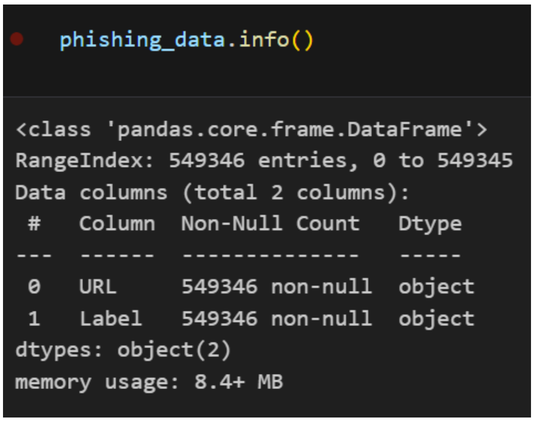
4 Data Collection and Preprocessing

4.1 Acquiring the Dataset:

- PhishTank Dataset: access the PhishTank dataset from PhishTank's professional internet site. download the dataset, generally available in CSV or JSON format.

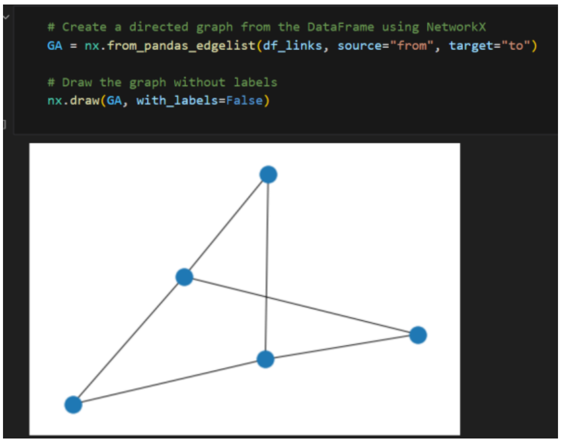
4.2 Preprocessing the data:

- Data cleansing: Load the dataset with the use of Pandas and perform preliminary cleansing, including removing duplicates and managing missing values.
- NLP Preprocessing: Utilize NLTK for text preprocessing. This consists of tokenization, getting rid of stopwords, and stemming. Use instructions like nltk.download('punkt') and nltk.download('stopwords') to get the necessary resources.



```
phishing_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 549346 entries, 0 to 549345
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   URL     549346 non-null  object
 1   Label   549346 non-null  object
dtypes: object(2)
memory usage: 8.4+ MB
```

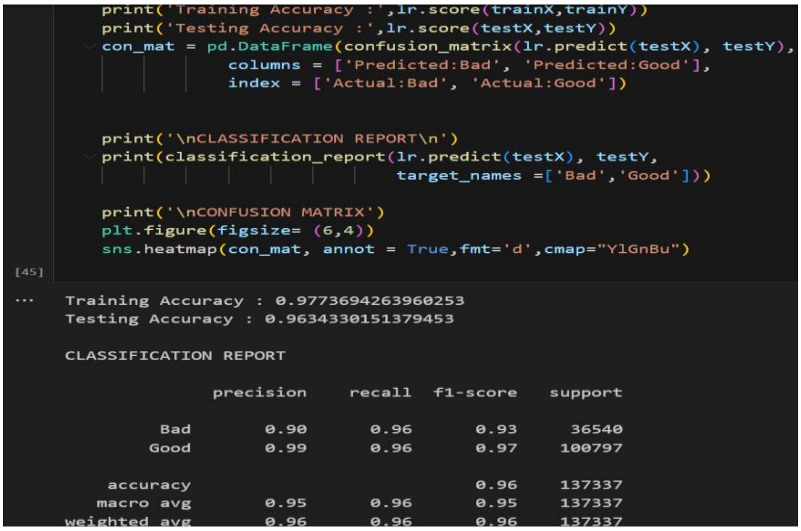


```
# Create a directed graph from the DataFrame using NetworkX
GA = nx.from_pandas_edgelist(df_links, source="from", target="to")

# Draw the graph without labels
nx.draw(GA, with_labels=False)
```

5 Machine Learning Model Configuration

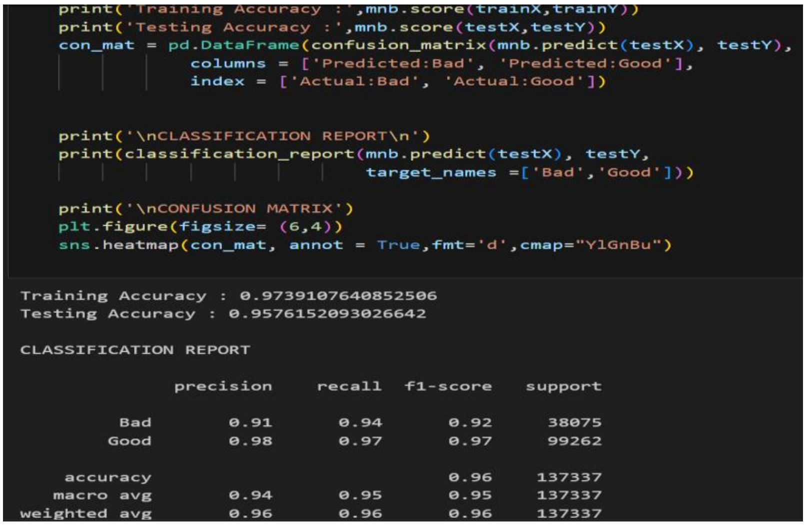
5.1 Logistic Regression and Multinomial Naive Bayes Setup:

- Model Initialization: Initialize the logistic regression and multinomial Naive Bayes models the usage of Scikit-analyze. as an instance, from sklearn.linear_model import LogisticRegression and from sklearn.naive_bayes import MultinomialNB.
- Feature Extraction: Convert text data into numerical capabilities the use of strategies like TF-IDF.



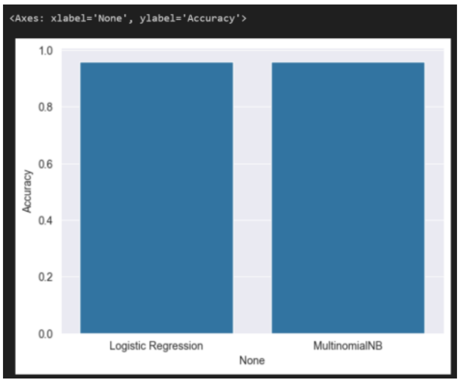
```
print('Training Accuracy :',lr.score(trainX,trainY))
print('Testing Accuracy :',lr.score(testX,testY))
con_mat = pd.DataFrame(confusion_matrix(lr.predict(testX), testY),
            columns = ['Predicted:Bad', 'Predicted:Good'],
            index = ['Actual:Bad', 'Actual:Good'])


print('\nCLASSIFICATION REPORT\n')
print(classification_report(lr.predict(testX), testY,
                            target_names =['Bad','Good']))

print('\nCONFUSION MATRIX')
plt.figure(figsize= (6,4))
sns.heatmap(con_mat, annot = True,fmt='d',cmap="YlGnBu")
```

```
Training Accuracy : 0.9773694263960253
Testing Accuracy : 0.9634330151379453

CLASSIFICATION REPORT

              precision    recall  f1-score   support

         Bad       0.90      0.96      0.93     36540
        Good       0.99      0.96      0.97    100797

    accuracy                           0.96    137337
   macro avg       0.95      0.96      0.95    137337
weighted avg       0.96      0.96      0.96    137337
```



```
    print('Training Accuracy :',mnb.score(trainX,trainY))
    print('Testing Accuracy :',mnb.score(testX,testY))
    con_mat = pd.DataFrame(confusion_matrix(mnb.predict(testX), testY),
                columns = ['Predicted:Bad', 'Predicted:Good'],
                index = ['Actual:Bad', 'Actual:Good'])


    print('\nCLASSIFICATION REPORT\n')
    print(classification_report(mnb.predict(testX), testY,
                                target_names =['Bad','Good']))

    print('\nCONFUSION MATRIX')
    plt.figure(figsize= (6,4))
    sns.heatmap(con_mat, annot = True,fmt='d',cmap="YlGnBu")
```

```
Training Accuracy : 0.9739107640852506
Testing Accuracy : 0.9576152093026642

CLASSIFICATION REPORT

              precision    recall  f1-score   support

         Bad       0.91      0.94      0.92     38075
        Good       0.98      0.97      0.97     99262

    accuracy                           0.96    137337
   macro avg       0.94      0.95      0.95    137337
weighted avg       0.96      0.96      0.96    137337
```



```
<Axes: xlabel='None', ylabel='Accuracy'>
```

5.2 Model Training and Parameter Tuning:

- Training the models: Training the models in the use of the feature-extracted dataset. break up the dataset into schooling and testing units the usage of train_test_split from Scikit-examine.
- Hyperparameter Tuning: Utilize grid seek (GridSearchCV from Scikit-learn) for tuning the hyperparameters of the models. cognizance on parameters like C and penalty for logistic regression, and alpha for multinomial Naive Bayes.
- Model evaluation: Evaluate the models using metrics along with accuracy, precision, consideration, and F1 rating.

These steps provide a comprehensive guide for putting in the surroundings, making ready the information, and configuring the machine learning models for the studies undertaken.

```
#acc = pd.DataFrame.from_dict(Scores_ml,orient = 'index',columns=['Accuracy'])
#sns.set_style('darkgrid')
#sns.barplot(acc.index,acc.Accuracy)

acc = pd.DataFrame.from_dict(Scores_ml, orient='index', columns=['Accuracy'])
sns.set_style('darkgrid')
sns.barplot(x=acc.index, y=acc['Accuracy'])
```

6 Containerization with Docker

6.1 Creating a Dockerfile

- Initialize Dockerfile: Create a file named Dockerfile in the root directory of your project.
- Base Image: Start by specifying a base image. For Python applications, use FROM python:3.8-slim.
- Set Working Directory: Set the working directory inside the container using WORKDIR /app.
- Copy Files: Copy the necessary files into the container. Use COPY . /app to copy all files in the current directory into /app in the container.
- Install Dependencies: Install required Python libraries. Include a line RUN pip install -r requirements.txt assuming you have a requirements.txt file listing all dependencies.
- Set Run Command: Specify the command to run the application, such as CMD ["python", "./your-script.py"].

```
# Use the official Ubuntu 20.04 image
FROM ubuntu:20.04

# Set the working directory inside the container
WORKDIR /app

# Create the output directory
RUN mkdir output

# Install system dependencies
RUN apt-get update && apt-get install -y python3 python3-pip curl && \
   curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl" && \
   chmod +x kubectl && \
   mv kubectl /usr/local/bin/

# Install Kubeflow Pipelines SDK
RUN pip3 install kfp==1.7.0  # Use the version that corresponds to your Kubeflow version

# Copy the requirements file into the container
COPY requirements.txt .

# Install Python dependencies
RUN pip3 install --no-cache-dir -r requirements.txt

# Copy the application files into the container
COPY . ./

# Expose the port that your application will run on
EXPOSE 8000

# Command to run your application
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000", "--root-path", "/"]
```

6.2 Building and Testing the Docker Container

- Build Container: In the command line, navigate to the directory containing the Dockerfile and run docker build -t your-app-name . to build the container.
- Run Container Locally: Test the container locally by running docker run -p 4000:80 your-app-name.
- Verify Operation: Ensure the application is running correctly in the container. Check logs or connect to the application if it has a user interface.
- To build the image –
  docker build -t kubeflow-fastapi .

7 Cloud Deployment on AWS EKS

7.1 Setting Up AWS EKS:

- Create EKS Cluster: Use the AWS Management Console or AWS CLI to create an EKS cluster. Follow AWS's documentation for detailed steps.
- Configure kubectl: Configure kubectl to interact with your EKS cluster. Run aws eks --region region update-kubeconfig --name cluster_name.
- Create a EKS cluster

```
export CLUSTER_NAME=MlOps-kube
export CLUSTER_REGION=us-east-1
eksctl create cluster \
--name ${CLUSTER_NAME} \
--version 1.25 \
--region ${CLUSTER_REGION} \
--nodegroup-name linux-nodes \
--node-type m5.xlarge \
--nodes 2 \
--nodes-min 2 \
--nodes-max 5 \
--managed \
--with-oidc


#Created addon for eks 1.25

eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster MlOps-kube \
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
    --approve
```

7.2 Deploying the Docker Container

- Push to container Registry: Push the Docker picture to a container registry, which includes Amazon ECR (Elastic container Registry).

- Create Kubernetes Deployment: Write a Kubernetes deployment YAML report to define the deployment. include the Docker image URL from ECR.
- Deploy to EKS: Practice the deployment to your EKS cluster using kubectl observe -f deployment.yaml.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin
871927043079.dkr.ecr.us-east-1.amazonaws.com

docker build -t kubeflow .

docker tag kubeflow:latest 871927043079.dkr.ecr.us-east-1.amazonaws.com/kubeflow:latest

docker push 871927043079.dkr.ecr.us-east-1.amazonaws.com/kubeflow:latest
```

8 Kubeflow Integration

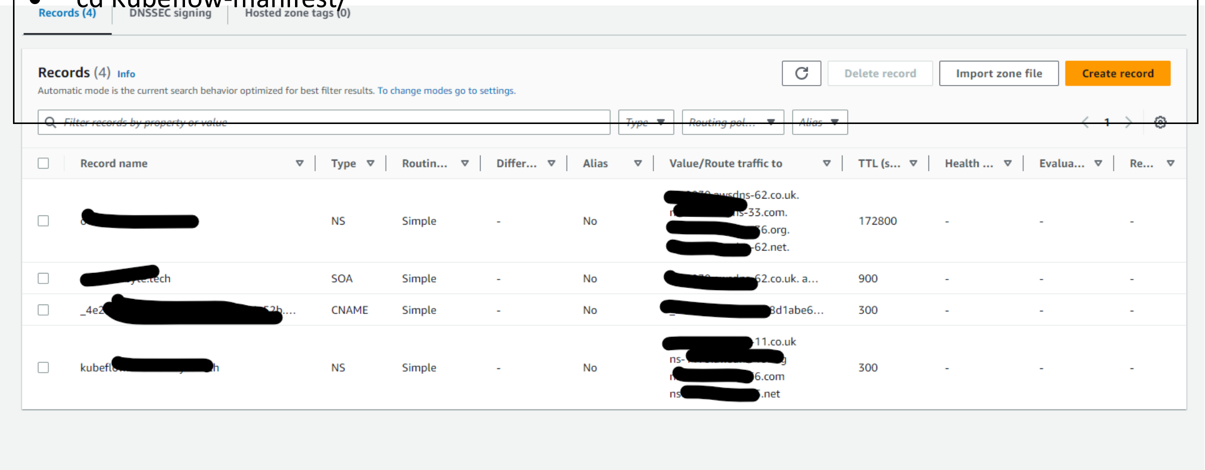
8.1 Integrating Kubeflow in AWS EKS

- Installation Kubeflow: Observe the reputable Kubeflow documentation to put in Kubeflow on your EKS cluster. This usually includes jogging a set of kubectl instructions.
- Configure Kubeflow Pipelines: installation Kubeflow pipelines in your ML workflows. define pipeline additives and obligations in a pipeline definition document.
- Deploy Pipelines: Deploy your pipelines to the Kubeflow Pipelines surroundings the usage of the Kubeflow Pipelines UI or CLI equipment.
- Test and monitor: Test the pipeline to make sure it is processing as predicted. screen pipeline runs and overall performance via the Kubeflow dashboard.
- Installing Kubeflow steps below:

```
export KUBEFLOW_RELEASE_VERSION=v1.7.0
export AWS_RELEASE_VERSION=v1.7.0-aws-b1.0.3

git clone https://github.com/awslabs/kubeflow-manifests.git && cd
kubeflow-manifests

git checkout ${AWS_RELEASE_VERSION}

git clone --branch ${KUBEFLOW_RELEASE_VERSION}
https://github.com/kubeflow/manifests.git upstream

cd Kubeflow-manifest/

# make the install-tools, which will install all necessary tools

make install-tools

# Then make sure the default is set to python3.8

alias python=python3.8

aws configure --profile=kubeflow
# AWS Access Key ID [None]: <enter access key id>
# AWS Secret Access Key [None]: <enter secret access key>
# Default region name [None]: <AWS region>
# Default output format [None]: json

# Set the AWS_PROFILE variable with the profile above
export AWS_PROFILE=Kubeflow

eksctl utils associate-iam-oidc-provider --cluster ${CLUSTER_NAME} \
--region ${CLUSTER_REGION} –approve

#  installing Kubeflow with vanilla option

make deploy-kubeflow INSTALLATION_OPTION=kustomize
DEPLOYMENT_OPTION=vanilla

# to check the status of Kubeflow deployment

kubectl get pods -n cert-manager
kubectl get pods -n istio-system
kubectl get pods -n auth
kubectl get pods -n knative-eventing
kubectl get pods -n knative-serving
kubectl get pods -n kubeflow
kubectl get pods -n kubeflow-user-example-com

# to change the password create a hash first

python3 -c 'from passlib.hash import bcrypt; import getpass;
print(bcrypt.using(rounds=12, ident="2y").hash(getpass.getpass()))'

# edit the config-map.yaml file

nano upstream/common/dex/base/config-map.yaml

# chage the value of
...
  staticPasswords:
  - email: user@example.com
    hash: <enter the generated hash here>
```

Access Kubeflow on the public domain for API access purpose and with the SSL:-

- kubectl config current-context
- aws eks describe-cluster --name $CLUSTER_NAME --region $CLUSTER_REGION
- cd Kubeflow-manifest/

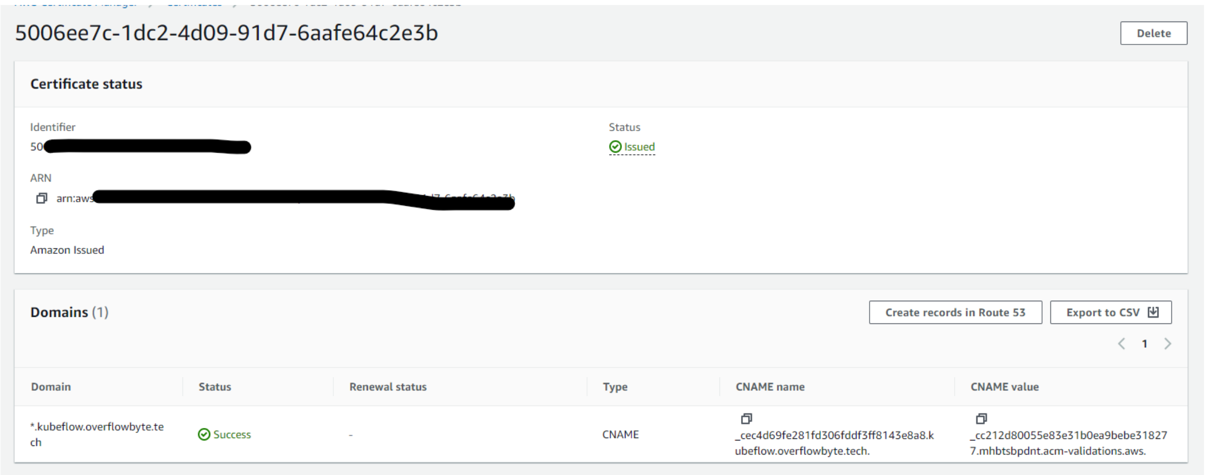


Records (4) Info

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

□	Record name	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...	Health ...	Evalua...	Re...
□	[redacted]	NS	Simple	-	No	[redacted]	172800	-	-	-
□	[redacted]	SOA	Simple	-	No	[redacted]	900	-	-	-
□	[redacted]	CNAME	Simple	-	No	[redacted]	300	-	-	-
□	[redacted]	NS	Simple	-	No	[redacted]	300	-	-	-

Create a route 53 domain and added a subdomain for that.
Issued for subdomain



5006ee7c-1dc2-4d09-91d7-6aafe64c2e3b

Certificate status

Identifier
50[redacted]

Status
Issued

ARN
arn:aws[redacted]

Type
Amazon Issued

Domains (1)

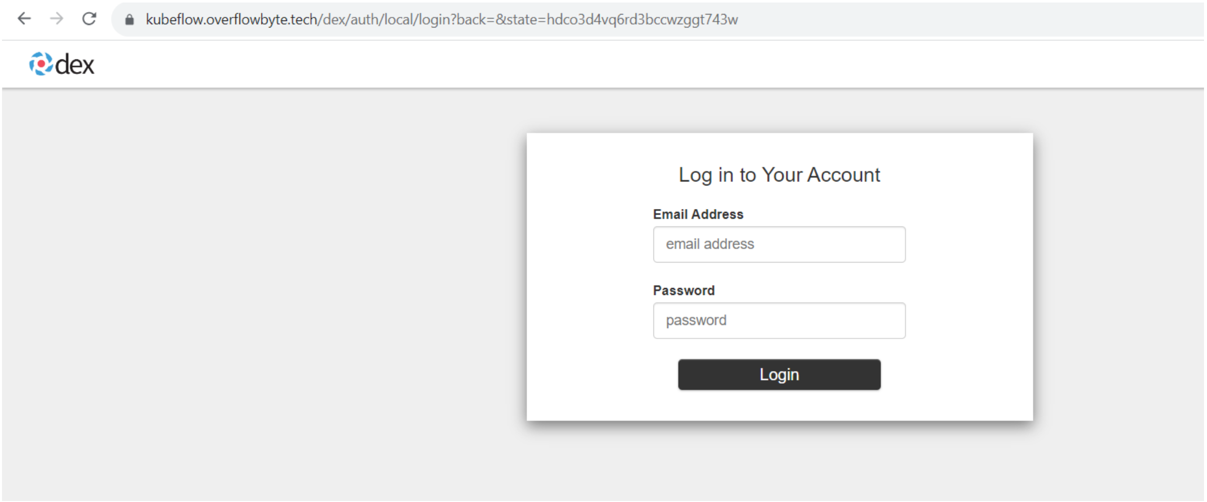
Domain	Status	Renewal status	Type	CNAME name	CNAME value
*.kubeflow.overflowbyte.tech	Success	-	CNAME	_cec4d69fe281fd306fddf3ff8143e8a8.kubeflow.overflowbyte.tech.	_cc212d80055e83e31b0ea9bebe318277.mhbtsbpdnt.acm-validations.aws.

```
export certArn=arn:aws:acm:us-east-1:1232454:certificate/4003wea3-3434ddf-dfdf

# Configure the parameters for ingress with the certificate ARN of the subdomain

printf 'certArn='$certArn'' > awsconfigs/common/istio-ingress/overlays/https/params.env

export TAG_VALUE=kubernetes.io/cluster/MlOps-kube # this is the tag value for our cluster

export CLUSTER_SUBNET_IDS=$(aws ec2 describe-subnets --region $CLUSTER_REGION --filters
Name=tag:alpha.eksctl.io/cluster-name,Values=$CLUSTER_NAME --output json | jq -r
'.Subnets[].SubnetId')
for i in "${CLUSTER_SUBNET_IDS[@]}"
do
   aws ec2 create-tags --resources ${i} --tags
Key=kubernetes.io/cluster/${CLUSTER_NAME},Value=${TAG_VALUE}
done

# The Load balancer controller uses IAM roles for service accounts(IRSA) to access AWS services

eksctl utils associate-iam-oidc-provider --cluster ${CLUSTER_NAME} --region ${CLUSTER_REGION} –
approve

# Create an IAM role with the necessary permissions for the Load Balancer controller to use via a
service account to access AWS services.

export LBC_POLICY_NAME=alb_ingress_controller_${CLUSTER_REGION}_${CLUSTER_NAME}

export LBC_POLICY_ARN=$(aws iam create-policy --policy-name $LBC_POLICY_NAME --policy-
document file://awsconfigs/infra_configs/iam_alb_ingress_policy.json --output text --query
'Policy.Arn')

eksctl create iamserviceaccount --name aws-load-balancer-controller --namespace kube-system --
cluster ${CLUSTER_NAME} --region ${CLUSTER_REGION} --attach-policy-arn ${LBC_POLICY_ARN} --
```

```
override-existing-serviceaccounts --approve

# Configure the parameters for load balancer controller with the cluster name.

printf 'clusterName='$CLUSTER_NAME'' > awsconfigs/common/aws-alb-ingress-controller/base/params.env

# build and install the Load Balancer controller kustomize file

kustomize build awsconfigs/common/aws-alb-ingress-controller/base | kubectl apply -f -
kubectl wait --for condition=established crd/ingressclassparams.elbv2.k8s.aws
kustomize build awsconfigs/common/aws-alb-ingress-controller/base | kubectl apply -f –

# Create an ingress that will use the certifcate we specified in certArn

kustomize build awsconfigs/common/istio-ingress/overlays/https | kubectl apply -f –

# To check the ALB provisioned or not

kubectl get ingress -n istio-system istio-ingress
```

Setup Kubeflow Pipeline –

Login to dashboard


kubeflow.overflowbyte.tech/dex/auth/local/login?back=&state=hdco3d4vq6rd3bccwzggt743w
dex

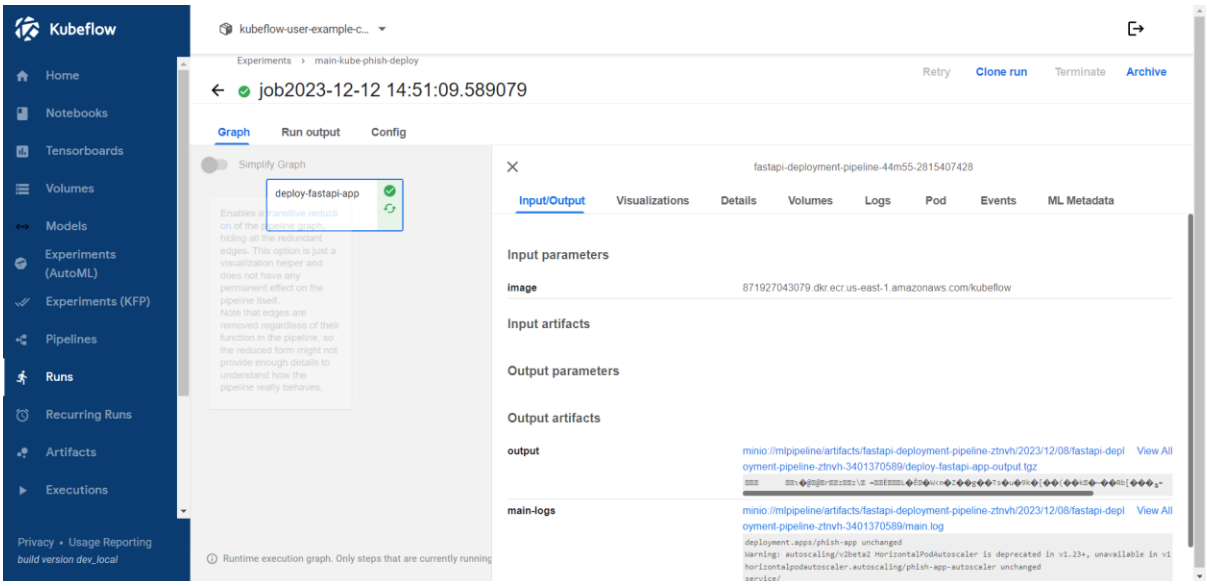
Log in to Your Account

Email Address

email address

Password

password



job2023-12-12 14:51:09.589079

Enables a transitive reduction of the pipeline graph, hiding all the redundant edges. This option is just a visualization helper and does not have any permanent effect on the pipeline itself.
Note that edges are removed regardless of their function in the pipeline, so the reduced form might not provide enough details to understand how the pipeline really behaves.

Runtime execution graph. Only steps that are currently running

fastapi-deployment-pipeline-44m55-2815407428

Input parameters

image 871927043079.dkr.ecr.us-east-1.amazonaws.com/kubeflow

Input artifacts

Output parameters

Output artifacts

output minio://mlpipeline/artifacts/fastapi-deployment-pipeline-ztnvh/2023/12/08/fastapi-deployment-pipeline-ztnvh-3401370589/deploy-fastapi-app-output.tgz

main-logs minio://mlpipeline/artifacts/fastapi-deployment-pipeline-ztnvh/2023/12/08/fastapi-deployment-pipeline-ztnvh-3401370589/main.log

```
deployment.apps/phish-app unchanged
Warning: autoscaling/v2beta2 HorizontalPodAutoscaler is deprecated in v1.23+, unavailable in v1
horizontalpodautoscaler.autoscaling/phish-app-autoscaler unchanged
service/
```

Making Kubeflow pipeline

```
import kfp
from kfp.dsl import ContainerOp

def deploy_fastapi_app(image: str):
    return ContainerOp(
        name='deploy-fastapi-app',
        image=image,
        command=[
            '/bin/sh', '-c',
            'kubectl apply -f phish-app-deployment.yaml -f phish-app-ingress.yaml && tar -czvf
/tmp/output.txt.tgz -C /app/output .'
        ],
        file_outputs={'output': '/tmp/output.txt.tgz'},
    )

@kfp.dsl.pipeline(
    name='FastAPI Deployment Pipeline',
    description='Pipeline for deploying FastAPI application'
)
def deploy_pipeline(image: str):
    deploy_op = deploy_fastapi_app(image=image)

if __name__ == '__main__':
    # my ecr image
    ecr_image = '871927043079.dkr.ecr.us-east-1.amazonaws.com/kubeflow:latest'

    kfp.compiler.Compiler().compile(deploy_pipeline, 'deploy-pipeline.zip')
    kfp.Client().create_run_from_pipeline_func(deploy_pipeline, arguments={'image': ecr_image})
```

This will create a zip for pipeline- will need to upload the pipeline in as the source in Kubeflow pipeline dashboard.

Deploying a FastAPI application along with a HorizontalPodAutoscaler and a Service.

Deployment (phish-app-deployment.yaml):

- The deployment specifies the desired state for the FastAPI application.
- It ensures that there is always one replica of the application running.
- The container is pulled from the specified ECR image with the latest tag.
- The container exposes port 8000, and the deployment is labeled with app: phish-app.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: phish-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: phish-app
  template:
    metadata:
      labels:
        app: phish-app
    spec:
      containers:
      - name: phish-app
        image: 871927043079.dkr.ecr.us-east-1.amazonaws.com/kubeflow:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
---
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: phish-app-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: phish-app
  minReplicas: 1
  maxReplicas: 4
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
---
apiVersion: v1
kind: Service
metadata:
  name: phish-app-service
spec:
  selector:
    app: phish-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8000
  type: LoadBalancer
```

Deploying a service for our FastAPI application so any HTTP request to the specified host with a path prefix of "/" should be directed to the phish-app-service on port 80. (Ali, 2022)

Ingress service deployment (phish-app-ingress.yaml)

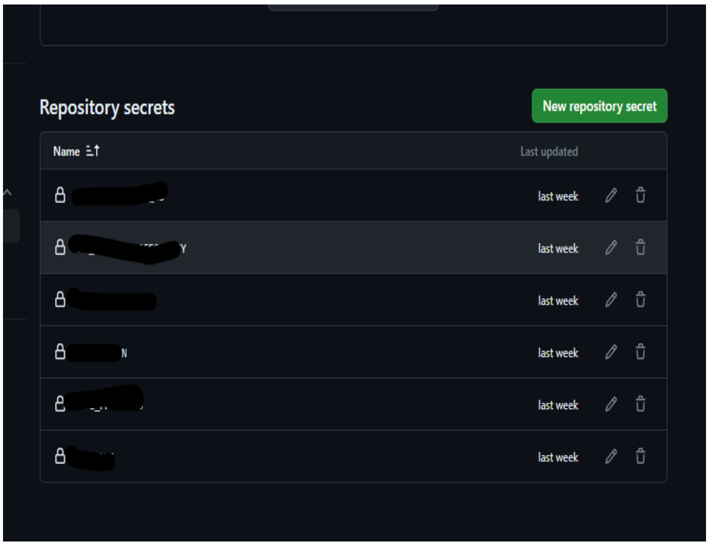
```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: phish-app-ingress
spec:
  rules:
  - host: ab4cba629e82842df92029dda4352d3f-1545613426.us-east-1.elb.amazonaws.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: phish-app-service
            port:
              number: 80
```

This YAML manifest defines a Kubernetes Ingress resource for your FastAPI application. The Ingress resource is used to expose your service externally and define how external HTTP/S traffic should be directed to your service.

9 CI/CD Pipeline Setup

9.1 Setting Up CI/CD Pipeline with GitHub Actions:

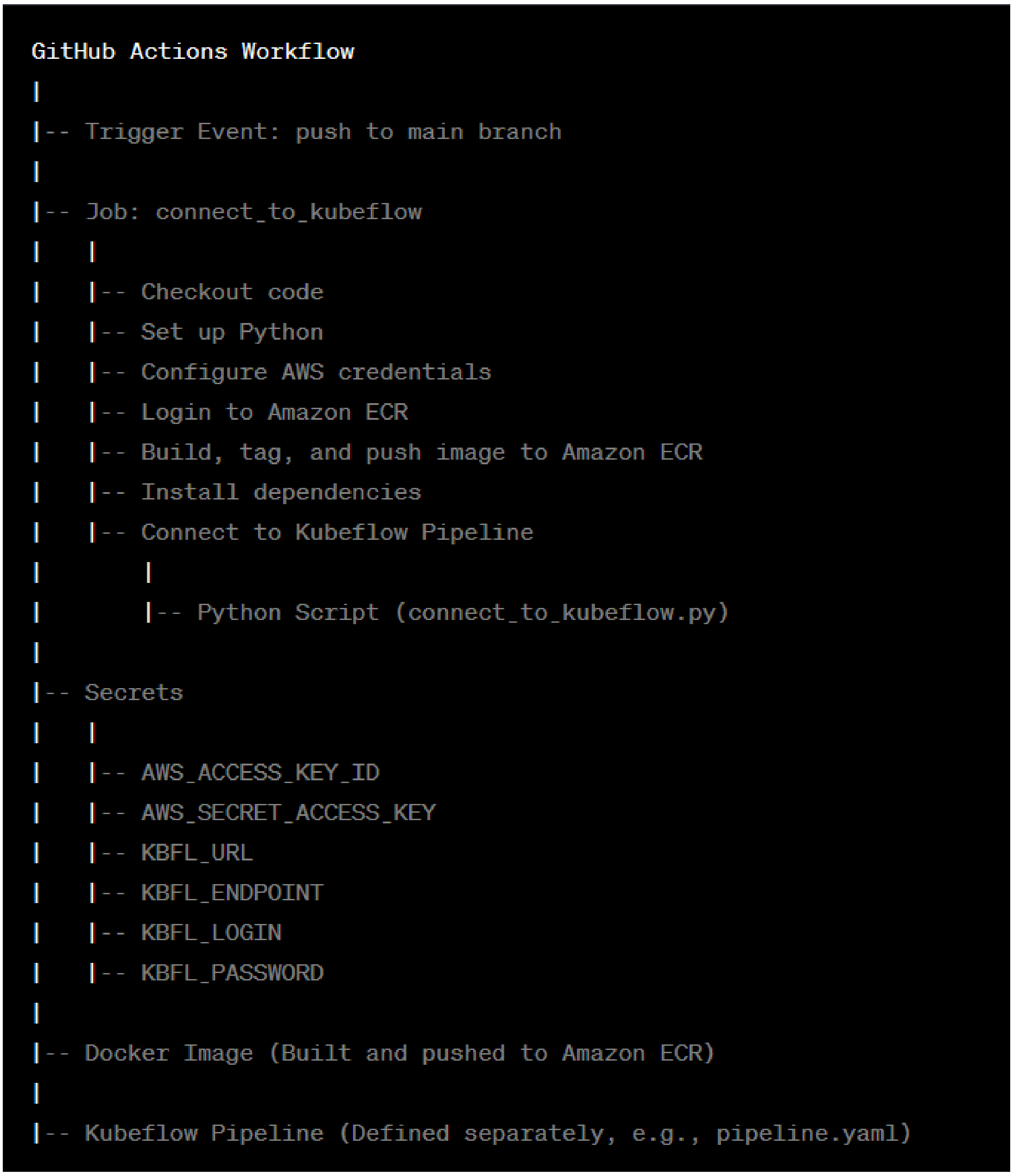
1. Initialize GitHub Repository: Ensure your project is in a GitHub repository.
2. Create GitHub Actions Workflow: In your repository, create a. github/workflows directory. Inside, create a YAML file (e.g., ci-cd-pipeline.yml) to define the workflow.
3. Define Workflow Steps:
   - Trigger event: Specify the occasions that trigger the workflow, including push or pull requests to precise branches.
   - Set up the environment: Use actions like moves/checkout to test out your repository and install the Python environment.
   - Install Dependencies: Upload steps to install required dependencies, e.g., pip set up -r necessities.txt.
   - Run tests: encompass commands to run computerized exams.
   - Build and Push Docker image: Add steps to construct the Docker image and push it to a container registry like AWS ECR.
   - Deploy to AWS EKS: Encompass steps to update the Kubernetes deployment on AWS EKS with the new Docker image.
   - Setup the secrets of aws which require to create a CI/CD pipeline



Name	Last updated
[redacted]	last week
[redacted]	last week
[redacted]	last week
[redacted]	last week
[redacted]	last week
[redacted]	last week

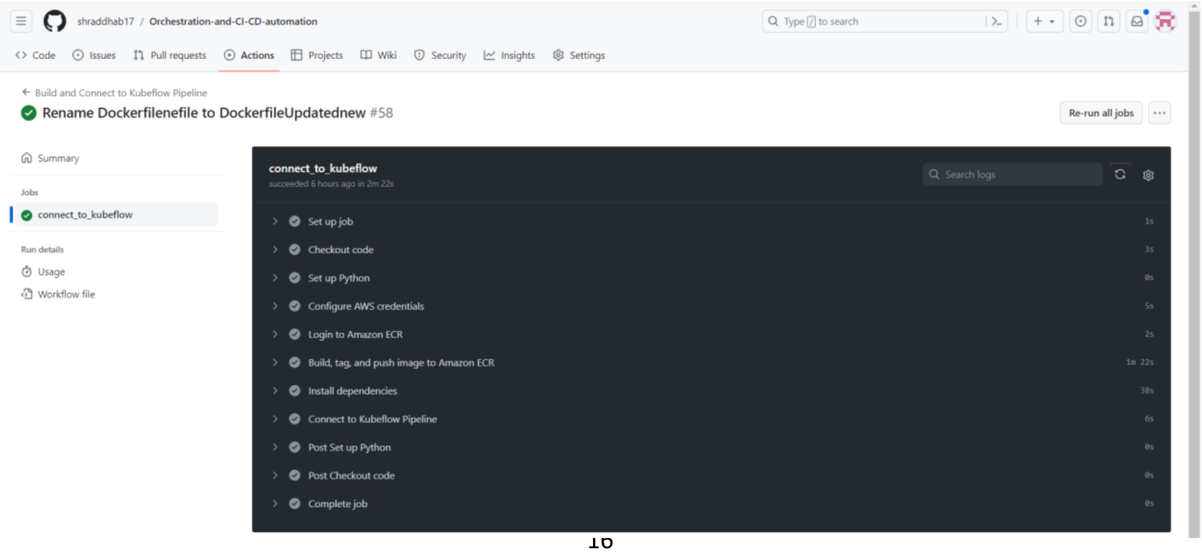
Creating a workflow for CI/CD pipeline

```
name: Build and Connect to Kubeflow Pipeline

on:
  push:
    branches: [main]

jobs:
  connect_to_kubeflow:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: 3.8

    - name: Configure AWS credentials
      run: |
        aws configure set aws_access_key_id ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws configure set aws_secret_access_key ${{ secrets.AWS_SECRET_ACCESS_KEY
}}
        aws configure set region us-east-1

    - name: Login to Amazon ECR
      run: aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 871927043079.dkr.ecr.us-east-1.amazonaws.com

    - name: Build, tag, and push image to Amazon ECR
      env:
        ECR_REPOSITORY: 871927043079.dkr.ecr.us-east-1.amazonaws.com/kubeflow
        IMAGE_TAG: latest
      run: |
        docker build -t $ECR_REPOSITORY:$IMAGE_TAG .
        docker push $ECR_REPOSITORY:$IMAGE_TAG

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install kfp==1.8.22 urllib3 requests requests-toolbelt mechanize bs4 cookiejar
        pip install -r requirements.txt
        pip show kfp

    - name: Connect to Kubeflow Pipeline
      run: |
       python connect_to_kubeflow.py  # replace with your script
      env:
        URL: ${{ secrets.KBFL_URL }}
        ENDPOINT: ${{ secrets.KBFL_ENDPOINT }}
        EMAIL: ${{ secrets.KBFL_LOGIN }}
        PASSWORD: ${{ secrets.KBFL_PASSWORD }}
```



```
GitHub Actions Workflow
|
|-- Trigger Event: push to main branch
|
|-- Job: connect_to_kubeflow
|    |
|    |-- Checkout code
|    |-- Set up Python
|    |-- Configure AWS credentials
|    |-- Login to Amazon ECR
|    |-- Build, tag, and push image to Amazon ECR
|    |-- Install dependencies
|    |-- Connect to Kubeflow Pipeline
|         |
|         |-- Python Script (connect_to_kubeflow.py)
|
|-- Secrets
|    |
|    |-- AWS_ACCESS_KEY_ID
|    |-- AWS_SECRET_ACCESS_KEY
|    |-- KBFL_URL
|    |-- KBFL_ENDPOINT
|    |-- KBFL_LOGIN
|    |-- KBFL_PASSWORD
|
|-- Docker Image (Built and pushed to Amazon ECR)
|
|-- Kubeflow Pipeline (Defined separately, e.g., pipeline.yaml)
```

CI pipeline status and running jobs.



Rename Dockerfilenefile to DockerfileUpdatednew #58

connect_to_kubeflow

connect_to_kubeflow
succeeded 6 hours ago in 2m 22s

```
> ✓ Set up job                                  1s
> ✓ Checkout code                               3s
> ✓ Set up Python                               0s
> ✓ Configure AWS credentials                   5s
> ✓ Login to Amazon ECR                         2s
> ✓ Build, tag, and push image to Amazon ECR    1m 22s
> ✓ Install dependencies                        38s
> ✓ Connect to Kubeflow Pipeline                6s
> ✓ Post Set up Python                          0s
> ✓ Post Checkout code                          0s
> ✓ Complete job                                0s
```

connect_to_kubeflow.py – This is used to connect our github action to Kubeflow pipeline trigger it on every push in the production environment.

```
import kfp
from datetime import datetime
import re
import os
import mechanize
from bs4 import BeautifulSoup
import urllib
import http.cookiejar as cookielib

# Get today's date for tags
today = str(datetime.now())

# Def
def get_id(text):
    """
    Function that retrieves a pipelines's ID from its logs
    Parameters
    ----------
    text : str
        string version of the logs.
    Returns
    -------
    str : Id of the pipeline.
    """
    match = re.search('{\'id\': \'(.+?)\',\\n', text)
    if match:
        found = match.group(1)
        return(found)

def get_cookie(text):
    """
    Function that retrieves login cookie
    Parameters
    ----------
    text : str
        string version of the logs.
    Returns
    -------
    str : cookie value.
    """
    match = re.search('authservice_session=(.+?) ', text)
    if match:
        found = match.group(1)
        return(found)

# Parameters
```

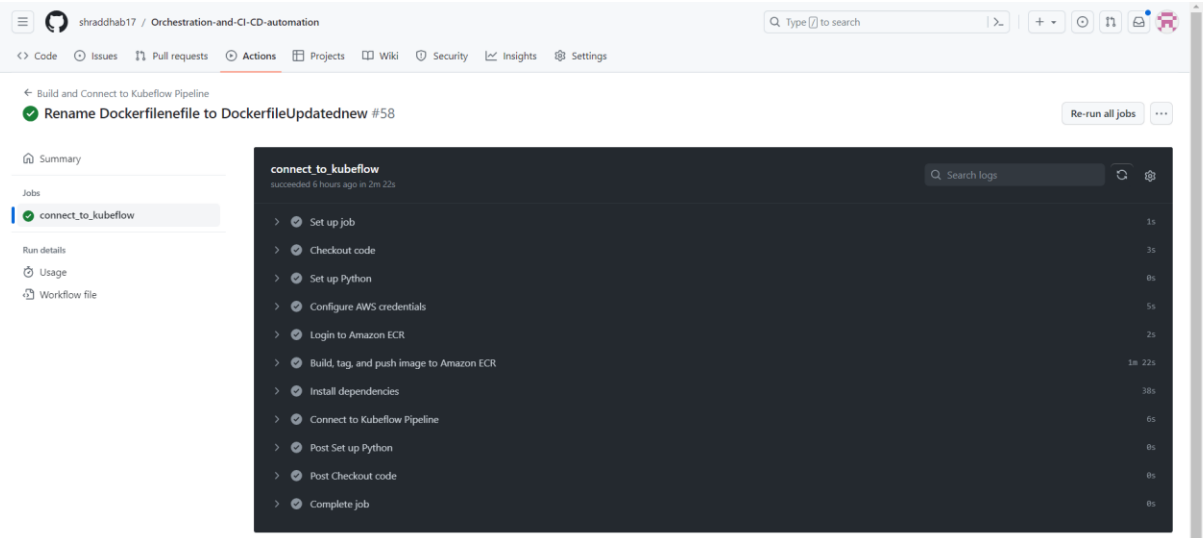
```
URL = os.getenv('URL')
pipeline_name = "advanced_pipeline"
job_name = 'job' + today

ENDPOINT = os.getenv('ENDPOINT')
EMAIL = os.getenv('EMAIL')
PASSWORD = os.getenv('PASSWORD')
# Run parameters
experiment_id = 'abc0d4b6-cea0-4681-a118-2d5715a0db10'
pipeline_id = '5eb5350c-77ad-436a-a138-d07a414efbb2'
# pipeline_id = get_id(str(pipe_logs))
version_id = '1'
params = {'image': '871927043079.dkr.ecr.us-east-1.amazonaws.com/kubeflow'}

# Get cookie value
cj = cookielib.CookieJar()
br = mechanize.Browser()
br.set_cookiejar(cj)
br.open(URL)

br.select_form(nr=0)
br.form['login'] = EMAIL
br.form['password'] = PASSWORD
br.submit()
authservice_session = 'authservice_session={}'.format(get_cookie(str(cj)))

# Connect to Kubeflow Pipelines Manager
client = kfp.Client(host=ENDPOINT, cookies=authservice_session)

# Run pipeline
client.run_pipeline(experiment_id=experiment_id,
                job_name=job_name,
                params=params,
                pipeline_id=pipeline_id)
```

9.2 Automated Testing and Deployment

- Automated testing: Configure automated tests to run with every push/pull request. Make certain assessments cover essential functionalities.
- Deployment Automation: Set up the workflow to automatically deploy the application to AWS EKS upon successful completion of tests.

To configure automated testing building and deployment we setup the workflow for every pull and run our pipeline to roll the product with new features.

Every push will run the pipeline jobs to push our product to market

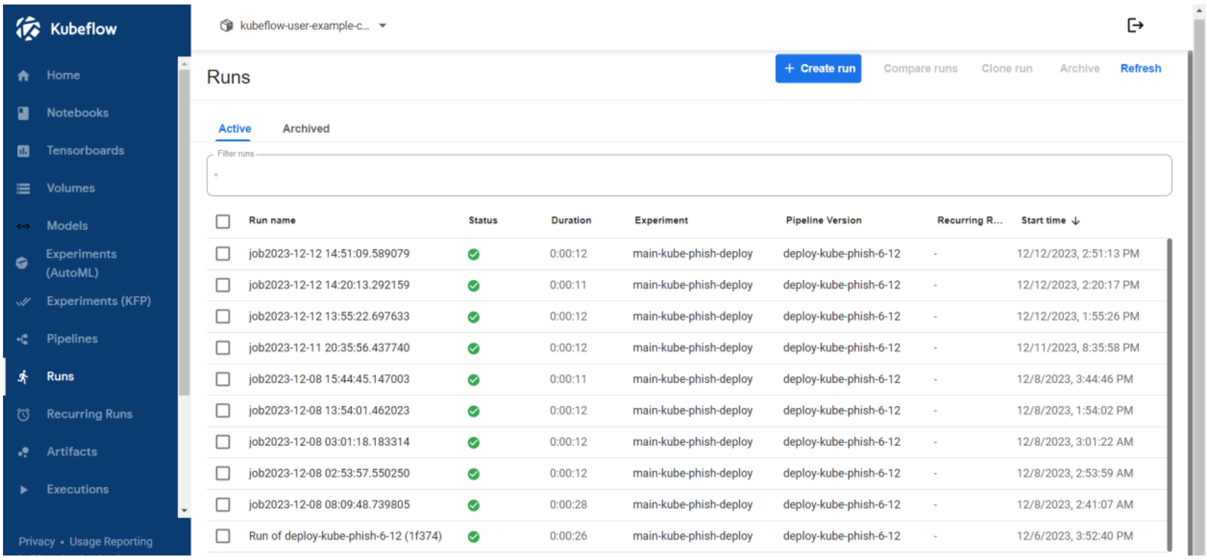


← Build and Connect to Kubeflow Pipeline

Rename Dockerfilenefile to DockerfileUpdatednew #58

connect_to_kubeflow

```
connect_to_kubeflow
succeeded 6 hours ago in 2m 22s

> Set up job                                   1s
> Checkout code                                3s
> Set up Python                                0s
> Configure AWS credentials                    5s
> Login to Amazon ECR                          2s
> Build, tag, and push image to Amazon ECR     1m 22s
> Install dependencies                         38s
> Connect to Kubeflow Pipeline                 6s
> Post Set up Python                           0s
> Post Checkout code                           0s
> Complete job                                 0s
```



☐	Run name	Status	Duration	Experiment	Pipeline Version	Recurring R...	Start time ↓
☐	job2023-12-12 14:51:09.589079	✓	0:00:12	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/12/2023, 2:51:13 PM
☐	job2023-12-12 14:20:13.292159	✓	0:00:11	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/12/2023, 2:20:17 PM
☐	job2023-12-12 13:55:22.697633	✓	0:00:12	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/12/2023, 1:55:26 PM
☐	job2023-12-11 20:35:56.437740	✓	0:00:12	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/11/2023, 8:35:58 PM
☐	job2023-12-08 15:44:45.147003	✓	0:00:11	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/8/2023, 3:44:46 PM
☐	job2023-12-08 13:54:01.462023	✓	0:00:12	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/8/2023, 1:54:02 PM
☐	job2023-12-08 03:01:18.183314	✓	0:00:12	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/8/2023, 3:01:22 AM
☐	job2023-12-08 02:53:57.550250	✓	0:00:12	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/8/2023, 2:53:59 AM
☐	job2023-12-08 08:09:48.739805	✓	0:00:28	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/8/2023, 2:41:07 AM
☐	Run of deploy-kube-phish-6-12 (1f374)	✓	0:00:26	main-kube-phish-deploy	deploy-kube-phish-6-12	-	12/6/2023, 3:52:40 PM

10 Troubleshooting

Common Issues and Solutions

- Failed Builds: Check for errors in the Dockerfile or dependency conflicts. Ensure all necessary files are included in the build context.
- Deployment Failures: Verify Kubernetes configuration files and AWS credentials. Ensure the Docker image is correctly tagged and accessible.
- Test Failures: Ensure test cases are up to date with the application code. Check for environmental differences between local and CI/CD environments.

11 Conclusion

This configuration manual serves as a comprehensive guide for setting up and managing the orchestration and CI/CD automation using MLOps for cloud-native container deployments. The steps outlined aim to facilitate a smooth setup and operational experience.

References

Ali, M. (2022) Build machine learning pipeline in python and package it as a Docker container, Medium. Available at: https://moez-62905.medium.com/build-machine-learning-pipeline-in-python-and-package-it-as-a-docker-container-13f28972fb2b (Accessed: 14 December 2023).

Anunaya, S. (2023) Data preprocessing in data mining - A hands on guide (updated 2023), Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/08/data-preprocessing-in-data-mining-a-hands-on-guide/ (Accessed: 14 December 2023).

E, S. (2021) Mlops end-to-end machine learning pipeline-CICD, Medium. Available at: https://medium.com/analytics-vidhya/mlops-end-to-end-machine-learning-pipeline-cicd-1a7907698a8e (Accessed: 14 December 2023).

Ekpang, M. (2019) Creating a CI/CD pipeline using Github Actions, Medium. Available at: https://medium.com/@michaelekpang/creating-a-ci-cd-pipeline-using-github-actions-b65bb248edfe (Accessed: 14 December 2023).

entando (2021) Installation on Amazon Elastic Kubernetes Service (EKS), Installation on Amazon Elastic Kubernetes Service (EKS) | Entando Developers. Available at: https://developer.entando.com/v6.3/tutorials/devops/installation/elastic-kubernetes-service/eks-install.html#appendix-a-troubleshooting (Accessed: 14 December 2023).

Insaid (2022) Containerization using Docker: AI end-to-end series (part-6), Medium. Available at: https://insaid.medium.com/containerization-using-docker-ai-end-to-end-series-part-6-d5d8c0979292 (Accessed: 14 December 2023).

Junaid Qazi, P. (2022) A30: Logistic Regression (part-2)&gt;&gt; behind the scene!, Medium. Available at: https://medium.com/mlearning-ai/a30-logistic-regression-part-2-behind-the-scene-38a98b70192a (Accessed: 14 December 2023).

Ledenev, A. (2016) Testing strategies for Docker containers, Medium. Available at: https://medium.com/hackernoon/testing-strategies-for-docker-containers-f633e261e75a (Accessed: 14 December 2023).

Ratz, A.V. (2022) Multinomial NAÏVE Bayes' for documents classification and Natural Language Processing (NLP), Medium. Available at: https://towardsdatascience.com/multinomial-na%C3%AFve-bayes-for-documents-classification-and-natural-language-processing-nlp-e08cc848ce6 (Accessed: 14 December 2023).

Thomas, R. (2021) Deploy a python API with Docker and Kubernetes, Medium. Available at: https://betterprogramming.pub/python-fastapi-kubernetes-gcp-296e0dc3abb6 (Accessed: 14 December 2023).