TEST 2

Design and setup a microservices framework to deploy a python, nodejs and react js application
The application will allow users to sign up, click a photo, save it to their profile and do image recognition on top of it
Python will be used for the image processing whereas node for APIs and react as a frontend
Explain the procedure to setup a CI/CD pipeline for the app,
The application needs to be built with redundancy and high data flow in mind with 0 downtime performance

Conclusion

1. Create UI for reactjs where integration to called API will be nodejs for python backend.
According to architectural flow Dockerize reactjs will be running with multiple replicasets.

Content for login.js and store in Repo

```
import React, { useState } from "react";
import Form from "react-bootstrap/Form";
import Button from "react-bootstrap/Button";
import "./Login.css";

export default function Login() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  function validateForm() {
    return email.length > 0 && password.length > 0;
  }

  function handleSubmit(event) {
    event.preventDefault();
  }

  return (
    <div className="Login">
      <Form onSubmit={handleSubmit}>
        <Form.Group size="lg" controlId="email">
          <Form.Label>Email</Form.Label>
          <Form.Control
            autoFocus
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
          />
        </Form.Group>
        <Form.Group size="lg" controlId="password">
          <Form.Label>Password</Form.Label>
          <Form.Control
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
        </Form.Group>
        <Button block size="lg" type="submit"
disabled={!validateForm()}>
          Login
        </Button>
      </Form>
    </div>
  );
}
```

DOCKERFILE for Reactjs

```
# pull the base image
FROM node:alpine

# set the working direction
WORKDIR /app

# add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH

# install app dependencies
COPY package.json ./
COPY login.json ./
COPY package-lock.json ./

RUN npm install

# add app
COPY . ./

```

```
# start app
CMD ["npm", "start"]
```

2. Create frontend and Backend service to distribute traffic on pods.

Frontend.yml

```
---
apiVersion: v1
kind: Service
metadata:
  name: Frontend
spec:
  selector:
    app: Reactjs
    tier: frontend
  ports:
  - protocol: TCP
    port: 80
    targetPort: http
...
```

Backend.yml

```
---
apiVersion: v1
kind: Service
metadata:
  name: Backend
spec:
  selector:
    app: python-app
    tier: backend
  ports:
  - protocol: TCP
    port: 80
    targetPort: http
...
```

3. Dockerize python script for image capture with following contents and label pods as backend

Image_cap.py

```
import cv2
import sys

cascPath = sys.argv[1]
faceCascade = cv2.CascadeClassifier(cascPath)

video_capture = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.cv.CV_HAAR_SCALE_IMAGE
    )

    # Draw a rectangle around the faces
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

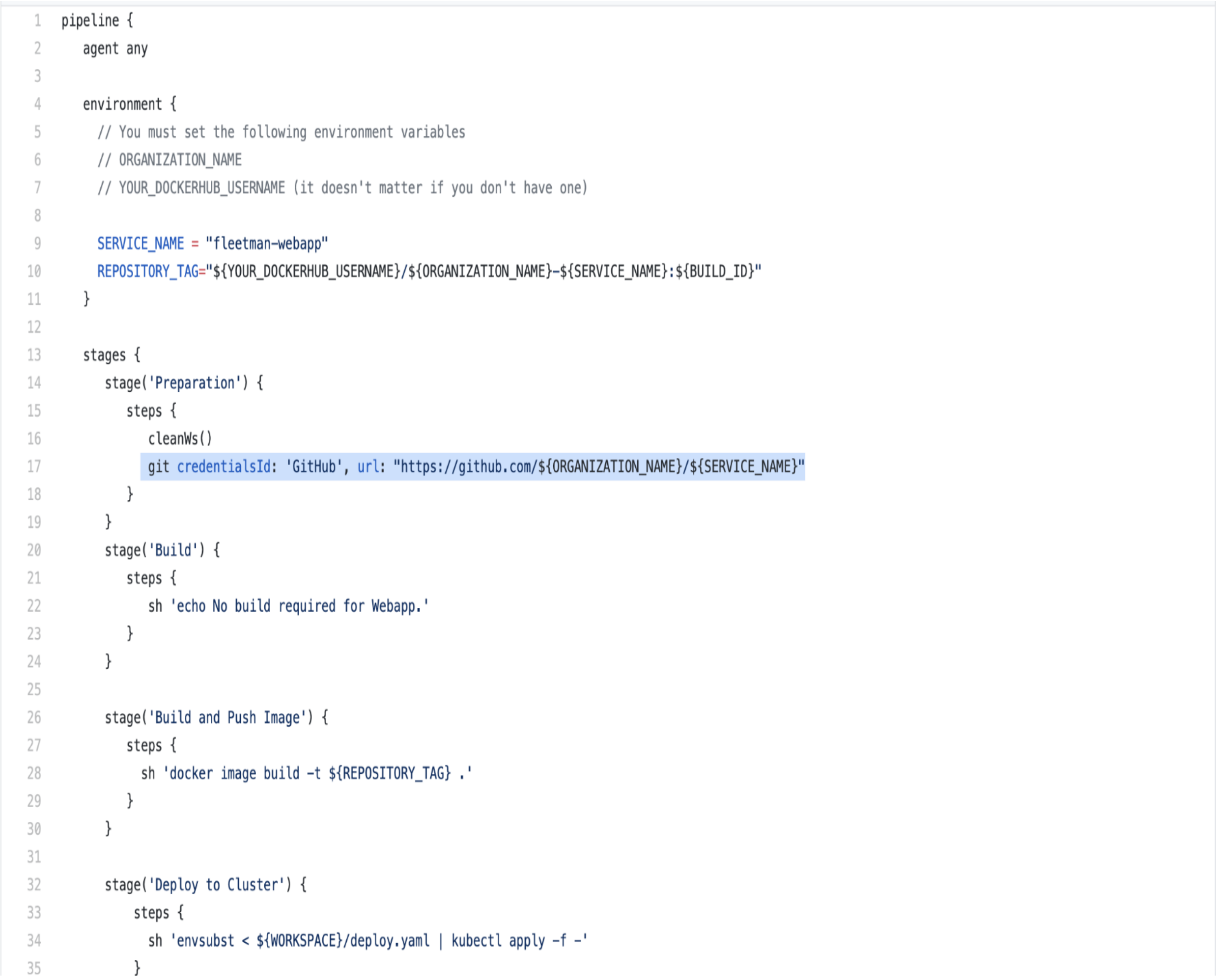
4. Put the code in gitlab or Bitbucket according to requirement and push images to docker hub or private container registry

CI/CD delivery

Install Kubectl on jenkins server
Copy your Cluster details to jenkins server to access kubernetes cluster:

sudo cp ~/.kube/config ~jenkins/.kube/$ sudo chown -R jenkins: ~jenkins/.kube/

Create pipeline.yml in each Repository



```
pipeline {
   agent any

   environment {
     // You must set the following environment variables
     // ORGANIZATION_NAME
     // YOUR_DOCKERHUB_USERNAME (it doesn't matter if you don't have one)

     SERVICE_NAME = "fleetman-webapp"
     REPOSITORY_TAG="${YOUR_DOCKERHUB_USERNAME}/${ORGANIZATION_NAME}-${SERVICE_NAME}:${BUILD_ID}"
   }

   stages {
      stage('Preparation') {
         steps {
            cleanWs()
            git credentialsId: 'GitHub', url: "https://github.com/${ORGANIZATION_NAME}/${SERVICE_NAME}"
         }
      }
      stage('Build') {
         steps {
            sh 'echo No build required for Webapp.'
         }
      }

      stage('Build and Push Image') {
         steps {
           sh 'docker image build -t ${REPOSITORY_TAG} .'
         }
      }

      stage('Deploy to Cluster') {
          steps {
            sh 'envsubst < ${WORKSPACE}/deploy.yaml | kubectl apply -f -'
          }
```

Since the setup is on kubernetes it has 0 downtime no matter what deployment strategy you use .,i.e, Rolling update (default), blue-green, canary.