# **Git commands**

**mkdir name**

**cd name**

**git config -–global user.name“username”**

**git config -–global user.email“email”**

**git init**

**git status**

**now create a file in the same directory as demo.txt**

**git add demo.txt**

**git status**

**git commit -m “message”**

**git clone (URL)**

**git branch name(**any name)

**git checkout name(**the given name in above step)

**git remote add origin** URL

**git remote -v**

**git push origin name(**name of branch given above)

**git pull origin name**

**git fetch origin name**

**git merge master**

# **Docker commands**

**docker build -t name .**

**docker run name**

**docker ps**

**docker ps -a**

**docker stops container id** (in the above step u will find the container id )

**docker rm container id(**any id from above step**)**

**docker rmi container id**

**docker images**

**docker logs container id**

**docker exec container id ls (**eg-docker exec faOdcb3588a2 ls**)**

**docker restart container id**

**docker save -o my\_image.tar name**

# **Practical 5: (Docker with Java)**

# **5a)hello world**

**Java file**

public class prac5a {

    public static void main(String[] args) {

        System.out.println("Hello world");

}

}

**Dockerfile (**save as Dockerfile (**without any extension**)**)**

FROM openjdk:latest

WORKDIR /app

COPY . /app

RUN javac prac5a.java //(name of your java file )

CMD  ["java","prac5a"] //(name of your java file )

# **5b) Factorial in java**

**Java file**

public class prac5b {

    public static void main(String[] args) {

        int number = 5;

        long factorial = calculateFactorial(number);

        System.out.println("Factorial of " + number + " is: " + factorial);

    }

    public static long calculateFactorial(int n) {

        long result = 1;

        for (int i = 1; i <= n; i++) {

            result \*= i;

        }

        return result;

    }

}

# **5c) Fibonacci**

class prac5c {

    public static void main(String args[])

    {

     int n1=0,n2=1,n3,i,count=10;

     System.out.print(n1+" "+n2);//printing 0 and 1

     for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed

     {

      n3=n1+n2;

      System.out.print(" "+n3);

      n1=n2;

      n2=n3;

     }

    }

}

# **6)Docker with nodejs**

# **A] Hello world**

**Node js file**

**App.js**

const http = require('http');

const hostname = '0.0.0.0';

const port = 3000;

const server = http.createServer((req, res) => {

  res.statusCode = 200;

  res.setHeader('Content-Type', 'text/plain');

  res.end('Hello, world!\n Burhanuddin 036');

});

server.listen(port, hostname, () => {

  console.log(`Server running at http://${hostname}:${port}/`);

});

**Dockerfile**

FROM node:14

WORKDIR /app

COPY package\*.json .

RUN npm install

COPY . .

EXPOSE 4000

CMD ["node", "app.js"]

**Run command (docker run -p 3000:3000 name)**

# **B] Odd and Even**

**Node js file**

**app2.js**

const http = require('http');

const hostname = '0.0.0.0';

const port = 4000;

const server = http.createServer((req, res) => {

    res.statusCode = 200;

    res.setHeader('Content-Type', 'text/plain');

    res.end('Hello, world!\nBurhanuddin 036\nODD AND EVEN');

    function isEven(number) {

        var number = 6;

        return number % 2 === 0;

    }

    function isOdd(number) {

        return number % 2 !== 0;

    }

    // Test the functions

    console.log(isEven(4));  // Output: true

    console.log(isEven(7));  // Output: false

    console.log(isOdd(4));   // Output: false

    console.log(isOdd(7));   // Output: true

});

server.listen(port, hostname, () => {

    console.log(`Server running at http://${hostname}:${port}/`);

});

**Dockerfile**

FROM node:14

WORKDIR /app

COPY package\*.json .

RUN npm install

COPY . .

EXPOSE 4000

CMD ["node", "app2.js"]

**Run command (docker run -p 4000:4000 name)**

# 7) working in docker with Ajax html

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title> AJAX</title>

</head>

<body>

    <div id="demo">

        <button type="button" onclick="loadinfo()">CHANGE</button>

    </div>

    <script>

        function loadinfo() {

            var xhttp = new XMLHttpRequest();

            xhttp.onreadystatechange = function() {

                if (this.readyState == 4 && this.status == 200) {

                    document.getElementById("demo").innerHTML = this.responseText;

                }

            };

            xhttp.open("GET", "ajax\_info.txt", true);

            xhttp.send();

        }

    </script>

</body>

</html>

**ajax\_info.txt file**

my name is -----

**dockerfile**

FROM nginx:alpine

COPY . /usr/share/nginx/html

**Run command (docker run -d -p 80:80 name)**

**Go to docker desktop, find 80:80 port and click**

# **8) Working with PHP**

**Index.php**

<?php

//  Print my name

echo "My Name is: fdhsjhffk ";

?>

**Dockerfile**

FROM php:7.4-cli

WORKDIR /app

COPY . /app

CMD ["php", "index.php"]

**Run command ( docker run name)**

# **11) Working with Python**

**main.py**

print("Hello, my name is ")

**dockerfile**

FROM python:3.9

WORKDIR /app

COPY main.py .

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

# **10)**Kubernetes with Java display Hello World

Step 1: Create HelloWorld.java

public class HelloWorld

{ public static void main(String[] args)

{ System.out.println("Hello, World!");

}

}

File Name: HelloWorld.java

**Step 2: Compile the Java Application:**

Open a terminal and navigate to the directory containing HelloWorld.java.

Compile the Java code using the javac command.

**Step 3: Create a Dockerfile:**

**Create a Dockerfile in the same directory as your Java source code.**

**This file specifies how the Docker image should be built.**

FROM openjdk:11

COPY HelloWorld.class /app/HelloWorld.class

WORKDIR /app

CMD ["java", "HelloWorld"]

**Step 4: Build the Docker Image:**

**Build the Docker image using the following command:**

docker build -t helloworld-java .

**This command creates a Docker image with the tag helloworld-java**

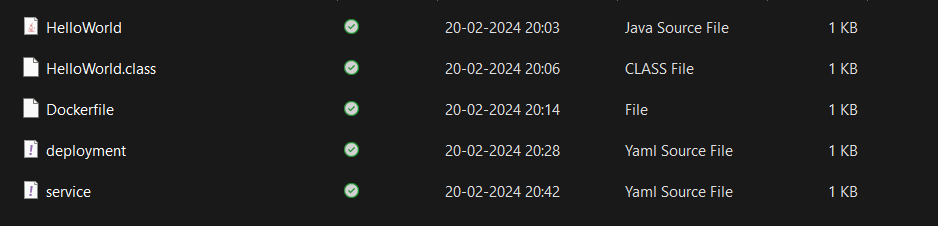
**Step 5: Run the Docker Container:**

**Run a Docker container based on the image you just built:**

docker run helloworld-java

**You should see the "Hello, World!" message printed in the console.**

**Step 6: Create a Kubernetes Deployment**

****

**Save as deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: helloworld-java-deployment

spec:

  replicas: 1

  selector:

    matchLabels:

      app: helloworld-java

  template:

    metadata:

      labels:

        app: helloworld-java

    spec:

      containers:

        - name: helloworld-java-container

          image: <your-docker-username>/helloworld-java

**Step 7: Apply the Deployment to Kubernetes:**

Kubectl apply -f deployment.yaml

**Step 8:create a service.yaml file**

apiVersion: v1

kind: Service

metadata:

  name: helloworld-java-service

spec:

  selector:

    app: helloworld-java

  ports:

    - protocol: TCP

      port: 80

      targetPort: 8080

      nodePort: 30000  # Choose any available port number between 30000 and 32767

  type: NodePort

**Step 9: Expose the Deployment as a Service:**

kubectl apply -f service.yaml

**Step 10 Access the Application:**

kubectl get svc helloworld-java-service

11) Kubernetes Command

# **9) Kubernetes Commands**

After creating Pract 10 open the file cmd and write the following code(continuation of pract 10)

**minikube start**

**minikube status**

**kubectl get service -o wide**

**kubectl get pod**

**kubectl describe pod**

**kubectl get all**

**minikube delete**

**minikube stop**

**kubectl get service**

**kubectl get nodes**

**kubectl cluster-info**

**kubectl get all**

**Kubectl delete node minikube**

**Create pod - kubectl apply -f mypod.yaml**