Git clone <link>

Cd name

Git checkout -b name

Git add

Git commit -m “commit”

Git push origin name

10. git checkout main

Git pull origin main

Git checkout name

Git clone<link>

Git clone<link>

Cd<repo name>

Code package.json

Save it

Modify file

Git add readme.md

Git commit -m “update readme.md”

Git push

normal kubernets

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 2

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.21

ports:

- containerPort: 80

kubectl apply -f nginx-deployment.yaml

kubectl get pods

kubectl expose deployment nginx-deployment --type=NodePort --name=nginx-service

kubectl get svc

go to chrom and type the number you get in x place

localhost:xxxxx

managing and verifying

kubectl scale deployment nginx-deployment --replicas=4

kubectl get pods

delete

kubectl delete svc nginx-service

kubectl delete deployment nginx-deployment

normal jenkins

docker pull jenkins/jenkins:lts

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

docker ps

go to chrome

localhost:8080

12 question – flask api docker and kub

App.py

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return jsonify({"message": "Welcome to the Flask API!"})

@app.route('/health')

def health():

return jsonify({"status": "Healthy"})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

dockerfile

FROM python:3.9-slim

WORKDIR /app

COPY app.py /app/

RUN pip install flask

EXPOSE 5000

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

1. In your terminal or command prompt, navigate to the directory containing app.py and Dockerfile.
2. **How:**
   * Run the following command

docker build -t your-dockerhub-username/flask-api:latest .

**Step 4: Push the Image to Docker Hub**

1. **Where:** In your terminal.
2. **How:**
   * Log in to Docker Hub:

bash

Copy code

docker login

Enter your Docker Hub username and password.

* + Push the image:

bash

Copy code

docker push your-dockerhub-username/flask-api:latest

**Step 5: Create Kubernetes Manifests**

1. **Where:** On your local machine, create two files: deployment.yaml and service.yaml.
2. **How:**
   * **deployment.yaml:**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: flask-api-deployment

spec:

replicas: 2

selector:

matchLabels:

app: flask-api

template:

metadata:

labels:

app: flask-api

spec:

containers:

- name: flask-api

image: your-dockerhub-username/flask-api:latest

ports:

- containerPort: 5000

* + **service.yaml:**

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: flask-api-service

spec:

type: NodePort

selector:

app: flask-api

ports:

- protocol: TCP

port: 5000

targetPort: 5000

nodePort: 30007

13th question – ci/cd docker

**steps to Run Flask App Without Installing Python Locally**

**1. Write Your Code**

1. Create a folder for your project (e.g., FlaskApp).
2. Inside this folder, create a file named app.py with the following content:

python

Copy code

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

return "Hello, Flask in Docker!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

**2. Create a Dockerfile**

1. In the same folder, create a file named Dockerfile:

Dockerfile

# Use an official Python runtime as a parent image

FROM python:3.9-slim

# Set the working directory

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install Flask

RUN pip install flask

# Make port 5000 available to the world outside this container

EXPOSE 5000

# Define the command to run the app

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

Copy code

**3. Build the Docker Image**

1. Open a terminal or command prompt in the folder where Dockerfile is located.
2. Run the following command to build the image:

bash

Copy code

docker build -t flask-app .

* + This creates a Docker image named flask-app.

**4. Run the Docker Container**

1. Start the container:

bash

Copy code

docker run -d -p 5000:5000 flask-app

1. Open your browser and visit:

arduino

Copy code

http:*//localhost:5000*

**5. (Optional) Push the Image to Docker Hub**

1. Log in to Docker Hub:

bash

Copy code

docker login

1. Tag your image:

bash

Copy code

docker tag flask-app your-dockerhub-username/flask-app

1. Push the image:

bash

Copy code

docker push your-dockerhub-username/flask-app

15th –

implement version control

Create a GitHub Repository and Implement Version Control Tasks: • Set up a repository for a team project. • Create branches for individual modules. • Merge branches with pull requests after code reviews. • Resolve conflicts during branch merging. • Document the workflow using the README file. • Submit the repository link

**Step 1: Set Up a GitHub Repository**

1. Go to [GitHub](https://github.com/).
2. Sign in to your account (or create one if you don't have one).
3. On the main page, click on the **+** icon at the top right corner and select **New repository**.
4. Fill out the repository details:
   * Repository name (e.g., team-project).
   * Description (e.g., A repository for our team project).
   * Choose **Public** or **Private** based on your preference.
   * Initialize the repository with a README file (optional, but helpful for documentation).
5. Click **Create repository**.

**Step 2: Clone the Repository Locally**

1. Open your terminal or Git Bash.
2. Clone the repository to your local machine by running the following command:

bash

Copy code

git clone https://github.com/username/team-project.git

1. Change into the project directory:

bash

Copy code

cd team-project

**Step 3: Create Branches for Individual Modules**

1. Create a branch for each module you're working on (e.g., module-a, module-b):

bash

Copy code

git checkout -b module-a

1. Make changes related to your module and commit them:

bash

Copy code

git add . git commit -m "Implemented Module A"

1. Push the branch to GitHub:

bash

Copy code

git push origin module-a

1. Repeat the process for other team members by creating different branches for their respective modules (e.g., module-b).

**Step 4: Create Pull Requests and Merge After Code Reviews**

1. Once a module is complete, push the branch to GitHub if not done already.
2. Go to the **Pull Requests** section of the repository on GitHub.
3. Click **New Pull Request** and select the base branch (usually main) and the compare branch (e.g., module-a).
4. Add a title and description, then request a review from team members.
5. After the review and approval, click **Merge pull request**.

**Step 5: Resolve Conflicts During Branch Merging**

1. If there are any merge conflicts, GitHub will notify you.
2. To resolve conflicts, pull the latest changes from the main branch:

bash

Copy code

git checkout main git pull origin main

1. Switch back to your module branch and merge the changes from main:

bash

Copy code

git checkout module-a git merge main

1. Resolve any conflicts manually in the conflicted files.
2. After resolving conflicts, stage the changes:

bash

Copy code

git add . git commit -m "Resolved merge conflicts"

1. Push the changes and complete the merge.

**Step 6: Document the Workflow Using the README File**

Edit the README.md file to document the repository's structure and workflow:

markdown

Copy code

# Team Project ## Workflow 1. Each team member works on a separate branch named after their module (e.g., `module-a`, `module-b`). 2. Once the changes are made, push the branch to GitHub and create a pull request to the `main` branch. 3. Pull requests should be reviewed before merging. 4. Resolve any merge conflicts and update the `main` branch. ## Getting Started Clone the repository: ```bash git clone https://github.com/username/team-project.git

**Branches**

* main: Main branch containing the latest stable version of the project.
* module-a: Branch for Module A (Add description here).
* module-b: Branch for Module B (Add description here).

vbnet

Copy code

### Step 7: Submit the Repository Link After everything is set up, you can share the link to the repository with your team members. The URL would look like this:

<https://github.com/username/team-project>

sql

Copy code

This workflow ensures that all changes are well-organized, reviewed, and integrated smoothly.

Git clone <link>

Cd name

Git checkout -b name

Git add

Git commit -m “commit”

Git push origin name

10. git checkout main

Git pull origin main

Git checkout name

Git clone<link>

Git clone<link>

Cd<repo name>

Code package.json

Save it

Modify file

Git add readme.md

Git commit -m “update readme.md”

Git push

9th and 10th questions (push, pull, merge)

git clone <link>

Cd <repository name>

git branch <branch name>

git checkout <branch name>

vim <any name>

[first type escape :wq]

git add <any name>

git commit -m “type any message”

12 question

App.py

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return jsonify({"message": "Welcome to the Flask API!"})

@app.route('/health')

def health():

return jsonify({"status": "Healthy"})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

dockerfile

FROM python:3.9-slim

WORKDIR /app

COPY app.py /app/

RUN pip install flask

EXPOSE 5000

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

1. In your terminal or command prompt, navigate to the directory containing app.py and Dockerfile.

2. How:

• Run the following command

docker build -t your-dockerhub-username/flask-api:latest .

Step 4: Push the Image to Docker Hub

1. Where: In your terminal.

2. How:

• Log in to Docker Hub:

bash

Copy code

docker login

Enter your Docker Hub username and password.

• Push the image:

bash

Copy code

docker push your-dockerhub-username/flask-api:latest

Step 5: Create Kubernetes Manifests

1. Where: On your local machine, create two files: deployment.yaml and service.yaml.

2. How:

• deployment.yaml:

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: flask-api-deployment

spec:

replicas: 2

selector:

matchLabels:

app: flask-api

template:

metadata:

labels:

app: flask-api

spec:

containers:

- name: flask-api

image: your-dockerhub-username/flask-api:latest

ports:

- containerPort: 5000

• service.yaml:

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: flask-api-service

spec:

type: NodePort

selector:

app: flask-api

ports:

- protocol: TCP

port: 5000

targetPort: 5000

nodePort: 30007

13th question

steps to Run Flask App Without Installing Python Locally

1. Write Your Code

1. Create a folder for your project (e.g., FlaskApp).

2. Inside this folder, create a file named app.py with the following content:

python

Copy code

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

return "Hello, Flask in Docker!"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Create a Dockerfile

1. In the same folder, create a file named Dockerfile:

Dockerfile

# Use an official Python runtime as a parent image

FROM python:3.9-slim

# Set the working directory

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install Flask

RUN pip install flask

# Make port 5000 available to the world outside this container

EXPOSE 5000

# Define the command to run the app

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

Copy code

3. Build the Docker Image

1. Open a terminal or command prompt in the folder where Dockerfile is located.

2. Run the following command to build the image:

bash

Copy code

docker build -t flask-app .

• This creates a Docker image named flask-app.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Run the Docker Container

1. Start the container:

bash

Copy code

docker run -d -p 5000:5000 flask-app

2. Open your browser and visit:

arduino

Copy code

http://localhost:5000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. (Optional) Push the Image to Docker Hub

1. Log in to Docker Hub:

bash

Copy code

docker login

2. Tag your image:

bash

Copy code

docker tag flask-app your-dockerhub-username/flask-app

3. Push the image:

bash

Copy code

docker push your-dockerhub-username/flask-app

15th

Create a GitHub Repository and Implement Version Control Tasks: • Set up a repository for a team project. • Create branches for individual modules. • Merge branches with pull requests after code reviews. • Resolve conflicts during branch merging. • Document the workflow using the README file. • Submit the repository link

Step 1: Set Up a GitHub Repository

1. Go to GitHub.

2. Sign in to your account (or create one if you don't have one).

3. On the main page, click on the + icon at the top right corner and select New repository.

4. Fill out the repository details:

• Repository name (e.g., team-project).

• Description (e.g., A repository for our team project).

• Choose Public or Private based on your preference.

• Initialize the repository with a README file (optional, but helpful for documentation).

5. Click Create repository.

Step 2: Clone the Repository Locally

1. Open your terminal or Git Bash.

2. Clone the repository to your local machine by running the following command:

bash

Copy code

git clone https://github.com/username/team-project.git

3. Change into the project directory:

bash

Copy code

cd team-project

Step 3: Create Branches for Individual Modules

1. Create a branch for each module you're working on (e.g., module-a, module-b):

bash

Copy code

git checkout -b module-a

2. Make changes related to your module and commit them:

bash

Copy code

git add . git commit -m "Implemented Module A"

3. Push the branch to GitHub:

bash

Copy code

git push origin module-a

4. Repeat the process for other team members by creating different branches for their respective modules (e.g., module-b).

Step 4: Create Pull Requests and Merge After Code Reviews

1. Once a module is complete, push the branch to GitHub if not done already.

2. Go to the Pull Requests section of the repository on GitHub.

3. Click New Pull Request and select the base branch (usually main) and the compare branch (e.g., module-a).

4. Add a title and description, then request a review from team members.

5. After the review and approval, click Merge pull request.

Step 5: Resolve Conflicts During Branch Merging

1. If there are any merge conflicts, GitHub will notify you.

2. To resolve conflicts, pull the latest changes from the main branch:

bash

Copy code

git checkout main git pull origin main

3. Switch back to your module branch and merge the changes from main:

bash

Copy code

git checkout module-a git merge main

4. Resolve any conflicts manually in the conflicted files.

5. After resolving conflicts, stage the changes:

bash

Copy code

git add . git commit -m "Resolved merge conflicts"

6. Push the changes and complete the merge.

Step 6: Document the Workflow Using the README File

Edit the README.md file to document the repository's structure and workflow:

markdown

Copy code

# Team Project ## Workflow 1. Each team member works on a separate branch named after their module (e.g., `module-a`, `module-b`). 2. Once the changes are made, push the branch to GitHub and create a pull request to the `main` branch. 3. Pull requests should be reviewed before merging. 4. Resolve any merge conflicts and update the `main` branch. ## Getting Started Clone the repository: ```bash git clone https://github.com/username/team-project.git

Branches

• main: Main branch containing the latest stable version of the project.

• module-a: Branch for Module A (Add description here).

• module-b: Branch for Module B (Add description here).

vbnet

Copy code

### Step 7: Submit the Repository Link After everything is set up, you can share the link to the repository with your team members. The URL would look like this:

https://github.com/username/team-project

sql

Copy code

This workflow ensures that all changes are well-organized, reviewed, and integrated smoothly.

16th to do

1. Install Docker (if not already installed)

If you haven't installed Docker yet, follow these instructions:

Install Docker based on your operating system.

2. Install VS Code Extensions

To make your workflow smoother in VS Code, install the following extensions:

Docker: This extension allows you to manage Docker containers and images directly from VS Code.

Remote - Containers: This allows you to develop inside Docker containers using VS Code.

3. Create the Flask Application in VS Code

Open VS Code and create a new folder (for example: flask-todo-app).

Inside this folder, create the following files:

app.py: This contains the Flask application code.

Dockerfile: This defines how the container is built.

requirements.txt: This specifies the dependencies for the Flask app.

Contents of app.py:

python

Copy code

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

todo\_list = []

@app.route('/todos', methods=['GET'])

def get\_todos():

return jsonify(todo\_list)

@app.route('/todos', methods=['POST'])

def add\_todo():

todo\_item = request.json.get('todo')

if todo\_item:

todo\_list.append({'todo': todo\_item})

return jsonify({'message': 'Todo added!'}), 201

return jsonify({'message': 'No todo provided!'}), 400

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

Contents of requirements.txt:

makefile

Copy code

Flask==2.1.2

Contents of Dockerfile:

Dockerfile

Copy code

# Step 1: Use an official Python runtime as a parent image

FROM python:3.9-slim

# Step 2: Set the working directory in the container

WORKDIR /app

# Step 3: Copy the current directory contents into the container at /app

COPY . /app

# Step 4: Install the necessary dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Step 5: Expose the port the app runs on

EXPOSE 5000

# Step 6: Set environment variables

ENV FLASK\_APP=app.py

ENV FLASK\_RUN\_HOST=0.0.0.0

# Step 7: Run the Flask application

CMD ["flask", "run"]

4. Build and Run the Flask Application in Docker (Locally)

Open the Integrated Terminal in VS Code:

You can open the terminal in VS Code using the shortcut Ctrl + ~.

Build the Docker Image: In the terminal, run the following command (ensure you're in the folder containing your Dockerfile and other files):

bash

Copy code

docker build -t flask-todo .

Run the Docker Container: Run the following command to start the Flask application inside the container:

bash

Copy code

docker run -p 5000:5000 flask-todo

This command maps port 5000 in the Docker container to port 5000 on your local machine. Now you can access the Flask app in your browser at http://localhost:5000.

5. Test the Application Locally

You can test the functionality using Postman, cURL, or your browser.

Test GET request to view the to-do list:

bash

Copy code

curl http://localhost:5000/todos

The response should be an empty list:

json

Copy code

[]

Test POST request to add a new to-do item:

bash

Copy code

curl -X POST http://localhost:5000/todos -H "Content-Type: application/json" -d '{"todo": "Buy groceries"}'

The response will confirm that the to-do was added:

json

Copy code

{"message": "Todo added!"}

Test GET request again to see the updated list:

bash

Copy code

curl http://localhost:5000/todos

The response should now include the new to-do:

json

Copy code

[{"todo": "Buy groceries"}]

6. Stop and Remove the Docker Container

Once you've tested everything, you can stop and remove the Docker container.

Stop the container: Press Ctrl + C in the terminal where the Docker container is running, or use the following command in a new terminal:

bash

Copy code

docker stop <container\_id>

Remove the container: After stopping, you can remove it with:

bash

Copy code

docker rm <container\_id>

You can find the container ID by running:

bash

Copy code

docker ps -a

Conclusion

app.py

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return jsonify(message="Welcome to the Flask API!")

@app.route('/api/data', methods=['GET'])

def get\_data():

return jsonify(data={"key1": "value1", "key2": "value2"})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

dockerfile

# Use the official Python image

FROM python:3.9-slim

# Set the working directory

WORKDIR /app

# Copy the application code

COPY app.py /app

# Install Flask

RUN pip install flask

# Expose port 5000

EXPOSE 5000

# Run the application

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

requirement.txt

flask

flask==2.1.3