# ✅ 1. Clone the Given GitHub Repository

```
bash
Copy code
# Step into your preferred local directory
cd path/to/your/folder

# Clone the GitHub repository
git clone <original-github-repo-link>
```

This will create a folder named `microservices-nodejs/`.

---

# ✅ 2. Push the Code to a New GitHub Repository

## A. Create a New Repo on GitHub

- Go to your GitHub account.
- Click `New repository`.
- Name it (e.g., `microservices-nodejs-dockerized`).
- Leave it **empty** (don't initialize with README or .gitignore).

## B. Change the Remote Origin and Push

```
bash
Copy code
cd microservices-nodejs

# Remove the original remote
git remote remove origin

# Add your new repository URL
git remote add origin https://github.com/your-username/microservices-nodejs-
dockerized.git

# Push all branches to your new repo
git branch -M main
git push -u origin main
```

## C. Add Your Main (Instructor/TA) as a Collaborator

- Go to your repo > `Settings` > `Collaborators and teams`
- Add their GitHub username as a collaborator.

---

# ✅ 3. Dockerize the Microservices

## A. Create a `Dockerfile` for Each Service

### Example: `customer-service/Dockerfile`

```
dockerfile
Copy code
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3001
CMD ["node", "app.js"]
```

Do the same for:

- `order-service/` (change port if different, e.g., `3002`)
- `restaurant-service/` (e.g., `3003`)

---

## B. Create a `docker-compose.yml` in the Root Directory

```
yaml
Copy code
version: '3.8'
services:
  customer-service:
    build: ./customer-service
    ports:
      - "3001:3001"
    volumes:
      - ./shared:/app/shared
    restart: always

  order-service:
    build: ./order-service
    ports:
      - "3002:3002"
    volumes:
      - ./shared:/app/shared
    restart: always

  restaurant-service:
    build: ./restaurant-service
    ports:
```

```
      - "3003:3003"
    volumes:
      - ./shared:/app/shared
    restart: always
```

This assumes all services import from `shared` as a relative path (`require('../shared/auth/jwt')`, etc.)

---

### C. Build and Run with Docker Desktop

```
bash
Copy code
# Make sure Docker Desktop is running
# From the project root:
docker-compose build
docker-compose up
```

Check Docker Desktop to confirm all containers are running.

---

## ✅ 4. Add GitHub Actions for CI/CD

### A. Create GitHub Actions Workflow File

Create folder and file:

```
bash
Copy code
mkdir -p .github/workflows
touch .github/workflows/docker-build.yml
```

### B. Example `docker-build.yml` Workflow

```
yaml
Copy code
name: Docker Build

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest
```

```
steps:
- name: Checkout code
  uses: actions/checkout@v3

- name: Set up Docker Buildx
  uses: docker/setup-buildx-action@v2

- name: Build customer-service image
  run: docker build ./customer-service -t customer-service

- name: Build order-service image
  run: docker build ./order-service -t order-service

- name: Build restaurant-service image
  run: docker build ./restaurant-service -t restaurant-service
```

You can enhance this by pushing to Docker Hub or GitHub Container Registry if needed.

Here's a summarized **notes file** covering the steps we discussed for building Docker images, tagging them, and pushing them to Docker Hub. You can use this for reference during your exam:

---

## Docker Image Build, Tagging, and Push Steps

### 1. Dockerfile Setup

- Ensure each service has its own `Dockerfile` with the following contents:
- `FROM node:18-alpine`
- 
- `WORKDIR /app`
- 
- `COPY package*.json ./`
- `RUN npm install`
- 
- `COPY . .`
- 
- `EXPOSE 3001`
- `CMD ["node", "app.js"]`

### 2. Docker Compose Setup

- In the main folder (`practice_instructor`), create a `docker-compose.yml`:
- `version: '3.8'`
- `services:`
- `  customer-service:`
- `    build: ./customer-service`

```
    ports:
      - "3001:3001"
    volumes:
      - ./shared:/app/shared
    restart: always

  order-service:
    build: ./order-service
    ports:
      - "3002:3002"
    volumes:
      - ./shared:/app/shared
    restart: always

  restaurant-service:
    build: ./restaurant-service
    ports:
      - "3003:3003"
    volumes:
      - ./shared:/app/shared
    restart: always
```

### 3. Building Docker Images

- Run the following command to build all the images defined in the `docker-compose.yml`:
- `docker-compose build`
- You should see images being built for each service: `customer-service`, `order-service`, and `restaurant-service`.

### 4. Check Built Images

- After the build completes, check the images with:
- `docker images`
- The images should be listed with a name format like:
  - `practice_instructor-customer-service`
  - `practice_instructor-order-service`
  - `practice_instructor-restaurant-service`

### 5. Tagging the Images

- Tag the locally built images to prepare them for pushing to Docker Hub:
- `docker tag practice_instructor-customer-service kashaf1234/customer-service`
- `docker tag practice_instructor-order-service kashaf1234/order-service`
- `docker tag practice_instructor-restaurant-service kashaf1234/restaurant-service`
  - Replace `practice_instructor-customer-service` with the exact image ID if needed.

### 6. Pushing the Images to Docker Hub

- Push the tagged images to Docker Hub:
- `docker push kashaf1234/customer-service`
- `docker push kashaf1234/order-service`
- `docker push kashaf1234/restaurant-service`

### 7. Login to Docker Hub (if not already logged in)

- Ensure you're logged into Docker Hub to push images:
- `docker login`
- Enter your Docker Hub username and password.

### 8. Pulling Images from Docker Hub

- Once pushed, you or others can pull the images on any system:
- `docker pull kashaf1234/customer-service`
- `docker pull kashaf1234/order-service`
- `docker pull kashaf1234/restaurant-service`

### 9. Troubleshooting Tips

- **Error during build**: Make sure your `Dockerfile` is in the right folder and contains the correct paths to `package.json`.
- **Missing image**: Ensure the image exists locally before pushing it to Docker Hub.
- **Tagging error**: Use the correct format for the image tag, i.e., `username/repository:tag`.

---

## Conclusion

- Follow the steps above to build Docker images, tag them with the correct repository name, push them to Docker Hub, and pull them from any system for use. Always ensure that your `docker-compose.yml` is set up properly, and that the services are built with the correct context and Dockerfile paths.

Good luck with your exam!

---

Feel free to use these notes as a guide during your exam! Let me know if you need further clarifications.