Dockerized Microservices Application with Kubernetes

Phase 1: Define the Project Scope

1. **Understand the requirements and deliverables**:
   - Containerize a shopping cart app (3–4 microservices).
   - Deploy it on Kubernetes.
   - Implement a service mesh (Istio or Linkerd).
   - Set up CI/CD for automated deployments.
   - Configure monitoring and centralized logging.
2. **Decide the Tech Stack**:
   - Use Docker for containerization.
   - Kubernetes for orchestration (using Minikube, EKS, or AKS).
   - Choose a service mesh: Istio for advanced traffic control and security.
   - Use Jenkins or GitLab CI for CI/CD pipelines.
   - Monitoring: Prometheus + Grafana.
   - Logging: Loki or ELK stack.

Phase 2: Prepare the Development Environment

1. **Set up your local environment**:
   - Install Docker and Kubernetes (e.g., Minikube or Kubernetes on cloud like EKS/AKS).
   - Install kubectl, Helm, and Istio CLI.
   - Install Prometheus and Grafana (optional for now).
2. **Set up CI/CD tool**:
   - Install Jenkins or configure GitLab CI.
3. **Optional: Set up cloud resources**:
   - If using AWS, create an EKS cluster.
   - For Azure, create an AKS cluster.
4. **Install supporting tools**:
   - Install Helm for Kubernetes package management.
   - Install Prometheus and Grafana locally or in the cluster.

Phase 3: Develop and Containerize the Microservices

1. **Design the application**:
   - Identify services for the shopping cart app (e.g., product service, user service, cart service, order service).
   - Decide the APIs for communication between microservices.
2. **Develop microservices**:
   - Write each service in your preferred language (Node.js, Python, etc.).
   - Include a Dockerfile for each service.
3. **Test locally**:
   - Run each service independently using Docker Compose or directly with Docker.

Phase 4: Deploy to Kubernetes

1. **Set up Kubernetes resources**:
   - Write YAML manifests for each microservice (Deployments, Services, ConfigMaps, etc.).
   - Test local Kubernetes deployment using Minikube or Kind.
2. **Deploy to the cluster**:
   - Apply the Kubernetes manifests to deploy the microservices.
3. **Verify deployment**:
   - Ensure all pods are running using `kubectl get pods`.
   - Expose services using NodePort/LoadBalancer.

Phase 5: Add the Service Mesh

1. **Install Istio or Linkerd**:
   - Use Helm or the CLI to install Istio/Linkerd on the cluster.
2. **Integrate the service mesh**:
   - Inject Istio sidecars into the microservices.
   - Configure traffic policies (e.g., rate limiting, retries).
3. **Secure communication**:
   - Enable mutual TLS (mTLS) between services.
4. **Add traffic control**:
   - Set up advanced routing policies (e.g., canary deployments).

Phase 6: Set Up CI/CD Pipelines

1. **Configure Jenkins/GitLab CI**:
   - Write pipeline scripts for building, testing, and deploying the microservices.
2. **Automate deployment**:
   - Set up GitOps or push Docker images and deploy with Kubernetes manifests.
3. **Test the pipeline**:
   - Validate CI/CD by making a code change and checking auto-deployment.

Phase 7: Set Up Monitoring and Logging

1. **Install monitoring tools**:
   - Deploy Prometheus and Grafana for metrics.
   - Create dashboards in Grafana for visualizing cluster and application health.
2. **Set up centralized logging**:
   - Deploy Loki/ELK stack to capture logs from all services.
3. **Validate observability**:
   - Trigger application traffic and verify metrics and logs.

Deliverables Checklist:

- A working shopping cart app containerized with Docker.
- Microservices deployed on Kubernetes.
- Service mesh configured with advanced traffic and security.
- CI/CD pipelines for automated deployments.
- Monitoring and logging dashboards.
- Comprehensive documentation.

Project Workflow (SDLC Phases Mapped)

SDLC Phase	Actions
Plan	Define project scope, deliverables and objectives
Design	Architect the microservices and their interactions (API design, db schema)
Implement	Write the code for microservices and containerize with Docker
Test	Test microservices independently (unit test) & together (integration test)
Deploy	Deploy microservices to Kubernetes using YAML manifests
Maintain	Monitor, log and enhance the app with observability tools and CI/CD pipelines

---

Phase 1: Defining the Project Scope

1.Objective of this Project:

This project focuses on:

- Building a **microservices-based application** (shopping cart app with 3–4 services).
- Deploying the application on **Kubernetes** (a container orchestration platform).
- Using a **service mesh** (like Istio or Linkerd) for:
  - **Traffic management** (routing, load balancing, etc.).
  - **Security** (e.g., encrypted communication between services).
- Automating deployments with **CI/CD pipelines** (using Jenkins or GitLab CI).
- Setting up **monitoring** (Prometheus + Grafana) and **logging** (Loki or ELK stack) for centralized observability.

Goal: Provide an end-to-end example of deploying and managing a modern cloud-native application with advanced DevOps and Kubernetes practices.

---

2. Deliverables

By the end of the project, you will deliver:

1. A **shopping cart app** built with 3–4 microservices:
   - Example services: `User`, `Product`, `Cart`, `Order`.
2. Microservices deployed on **Kubernetes** using YAML manifests.
3. **Service mesh** for:
   - Advanced routing (e.g., retries, failovers).
   - Secured communication (mTLS).
4. **CI/CD pipelines** for automated deployments.
5. Centralized:
   - **Monitoring**: Metrics dashboards via Prometheus + Grafana.
   - **Logging**: Collected logs via Loki or ELK stack.

3. Project's Significance:

- **Real-world relevance**: Modern applications are moving towards microservices for scalability and agility.
- **Showcase skills**: It demonstrates expertise in **containerization**, **Kubernetes**, **DevOps**, and **observability tools**, making it attractive to recruiters.
- **Learning opportunity**: Hands-on experience with cutting-edge tools like Istio, Prometheus, and CI/CD.

---

4. Tech Stack:

Tool/Technology	**Purpose**
Docker	Containerizes the Microservices
Kubernetes	Orchestrates and manages containers
Istio/Linkerd	Adds service mesh capabilties
Jenkins/GitLab	Automates builds and deployments
Prometheus (optional)	Collects and monitors metrics
Grafana (optional)	Visualizes metrics and dashboards
Loki/ELK	Collects and centralizes logs

Languages/Frameworks

- Language: Python, Node.js, or Java (your choice based on familiarity).
- Frameworks: Flask/Express.js/Spring Boot.

Cluster Type

- Local: Use **Minikube** or **Kind** for local Kubernetes setup.
- Cloud: **AWS EKS** or **Azure AKS** for production-level deployments.

---

5. Project Workflow

Here's a high-level workflow for the entire project:

1. **Plan & Design**:
   - Decide on app structure (e.g., APIs, data flow).
   - Define Kubernetes architecture (pods, services, ConfigMaps).
2. **Build Microservices**:
   - Write and containerize each service.
   - Test services locally.
3. **Deploy to Kubernetes**:
   - Deploy the services and set up networking.
4. **Add Service Mesh**:
   - Install Istio/Linkerd and configure advanced features.
5. **Set Up CI/CD**:
   - Automate deployment pipelines.
6. **Set Up Observability**:
   - Add monitoring and logging.

Phase 2: Setting Up the Development Environment:

1. Install Prerequisite Tools

Tools Needed:

Tool	**Purpose**
Docker	Containerize microservices
Kubernetes	Orchestration platform (MiniKube)
Kubectl	CLI to manage K8s clusters
Helm	Manages K8s resources (e.g., Istio)

Code Editor (optional)	Writing code and YAML files
Prometheus/Grafana (optional)	Monitoring and dashboards
Loki/ELK	Centralized logging

---

2. Install Docker

Installation:

```
sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl
software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install -y docker-ce docker-ce-cli containerd.io
```

Verify:

```
docker --version
```

---

3. Install Kubernetes (Minikube)

Why Minikube?

Minikube creates a local Kubernetes cluster that’s perfect for development and testing.

Installation:

```
curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Start Minikube:

```
minikube start
```

Verify:

```
kubectl get nodes
```

---

4. Install kubectl

Why kubectl?

kubectl allows you to interact with your Kubernetes cluster to deploy and manage applications.

Installation:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/
```

Verify:

```
kubectl version --client
```

---

5. Install Helm

Why Helm?

Helm simplifies Kubernetes deployments by managing Kubernetes manifests as charts.

Installation:

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 |
bash
```

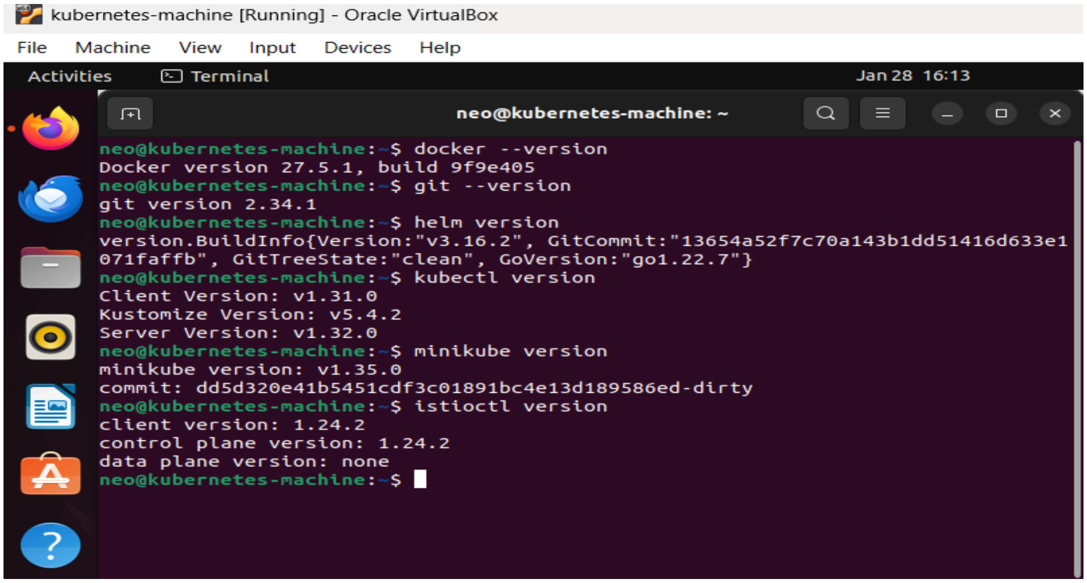
Verify:

```
helm version
```

---

6. Verify Everything

```
docker --version

minikube start

kubectl get nodes

helm version
```



kubernetes-machine [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Jan 28 16:13
neo@kubernetes-machine: ~

```
neo@kubernetes-machine:~$ docker --version
Docker version 27.5.1, build 9f9e405
neo@kubernetes-machine:~$ git --version
git version 2.34.1
neo@kubernetes-machine:~$ helm version
version.BuildInfo{Version:"v3.16.2", GitCommit:"13654a52f7c70a143b1dd51416d633e1
071faffb", GitTreeState:"clean", GoVersion:"go1.22.7"}
neo@kubernetes-machine:~$ kubectl version
Client Version: v1.31.0
Kustomize Version: v5.4.2
Server Version: v1.32.0
neo@kubernetes-machine:~$ minikube version
minikube version: v1.35.0
commit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty
neo@kubernetes-machine:~$ istioctl version
client version: 1.24.2
control plane version: 1.24.2
data plane version: none
neo@kubernetes-machine:~$ 
```

8. Folder Structure: Create a folder structure for the project:

```
project-root/

│

├── microservices/

│   ├── user-service/

│   ├── product-service/

│   ├── cart-service/

│   └── order-service/

│

├── kubernetes-manifests/
```

```
│   ├── deployments/
│   ├── services/
│   ├── configmaps/
│   └── ingress/
│
├── ci-cd-pipelines/
├── monitoring/
└── logging/
```

If we have an already planned project structure you can use a shell script to create the project structure like above:

Here's a shell script that creates the specified project directory structure. Save this script as `project_structure.sh` and run it in the terminal:

```bash
#!/bin/bash

PROJECT_ROOT="project-root"

DIRECTORIES=(
  "$PROJECT_ROOT/microservices/user-service"
  "$PROJECT_ROOT/microservices/product-service"
  "$PROJECT_ROOT/microservices/cart-service"
  "$PROJECT_ROOT/microservices/order-service"
  "$PROJECT_ROOT/kubernetes-manifests/deployments"
  "$PROJECT_ROOT/kubernetes-manifests/services"
  "$PROJECT_ROOT/kubernetes-manifests/configmaps"
  "$PROJECT_ROOT/kubernetes-manifests/ingress"
  "$PROJECT_ROOT/ci-cd-pipelines"
  "$PROJECT_ROOT/monitoring"
  "$PROJECT_ROOT/logging"
)

echo "Creating project directory structure..."
for DIR in "${DIRECTORIES[@]}"; do
  mkdir -p "$DIR"
  echo "Created: $DIR"
done

echo "Project directory structure created successfully."
```

Make this script executable:

```
Chmod +x project_structure.sh

./project_structure.sh
```

This will create the structure according to what we have planned.

Phase 3: Develop and Containerize the Microservices

1. Design the Application

Before writing any code, we need to design the architecture of our shopping cart app and define how the microservices will interact with each other. This design step aligns with the **planning phase of the Software Development Life Cycle (SDLC)**. Here's what it looks like:

1. **Identify Microservices**:
   - **User Service**: Handles user authentication and profile management.
   - **Product Service**: Manages the product catalog.
   - **Cart Service**: Handles adding/removing products to/from the cart.
   - **Order Service**: Processes and manages orders.
2. **Define APIs**: Each microservice will expose REST APIs for communication. For instance:
   - **User Service**:
     - `POST /users/login`: Authenticate user.
     - `GET /users/{id}`: Fetch user details.
   - **Product Service**:
     - `GET /products`: List all products.
     - `GET /products/{id}`: Fetch product details.
   - **Cart Service**:
     - `POST /cart`: Add an item to the cart.
     - `DELETE /cart/{itemId}`: Remove an item from the cart.
   - **Order Service**:
     - `POST /order`: Place an order.
     - `GET /order/{id}`: Fetch order details.
3. **Data Flow and Dependencies**:
   - The **User Service** provides user information and authentication.
   - The **Product Service** provides product details required by the **Cart Service**.
   - The **Cart Service** calculates cart totals and passes data to the **Order Service** for order creation.

> Here is a sample app.py code for all 4 microservices:

1. User Service

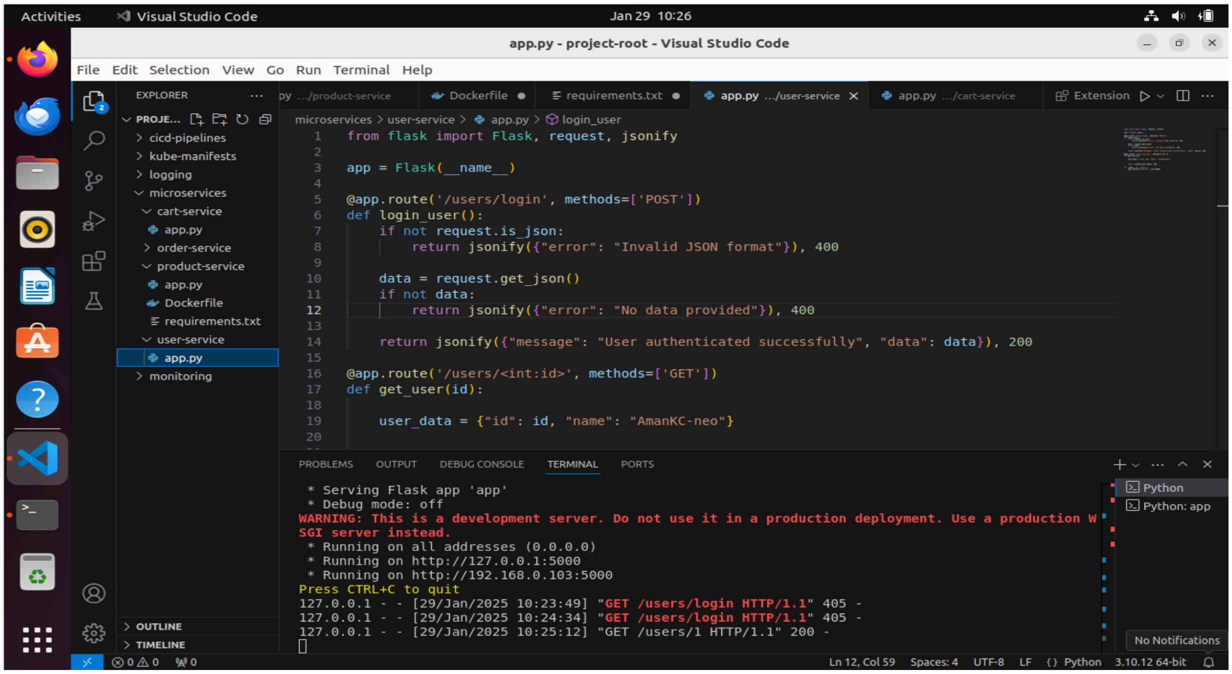
Purpose: Handles user authentication and profile management.

API Endpoints:

- POST /users/login: Authenticate user.
- GET /users/{id}: Fetch user details.

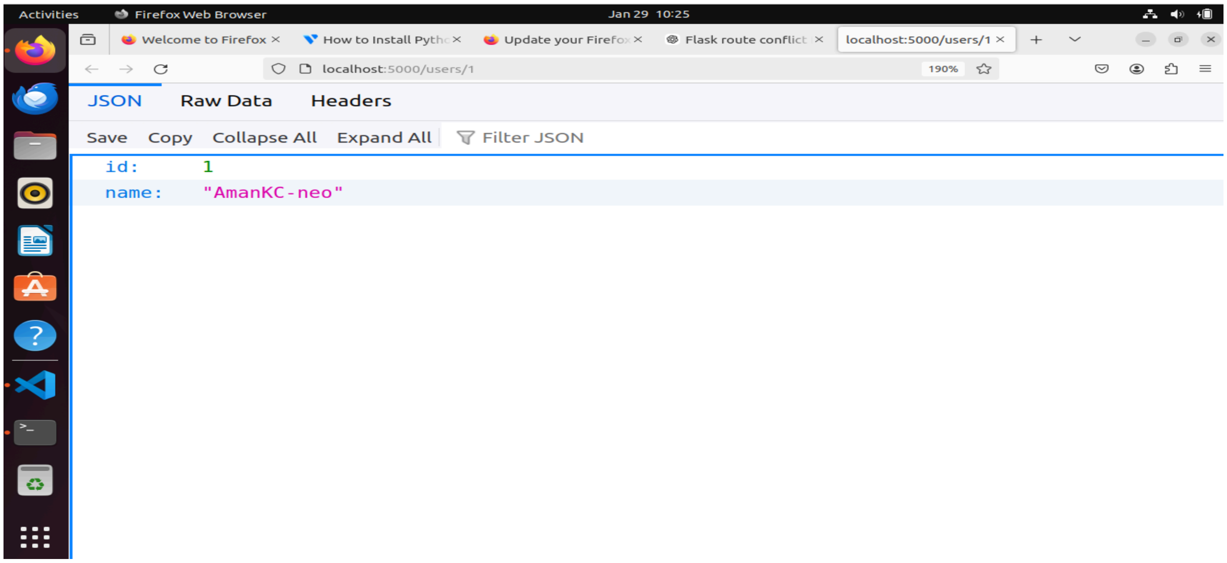
Implementation:

```
#user-service/app.py

from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/users/login', methods=['POST'])
def login_user():
    data = request.json
    return jsonify({"message": "User authenticated successfully", "data":
data}), 200

@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):
    return jsonify({"id": id, "name": "John Doe"}), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```



```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/users/login', methods=['POST'])
def login_user():
    if not request.is_json:
        return jsonify({"error": "Invalid JSON format"}), 400

    data = request.get_json()
    if not data:
        return jsonify({"error": "No data provided"}), 400

    return jsonify({"message": "User authenticated successfully", "data": data}), 200

@app.route('/users/<int:id>', methods=['GET'])
def get_user(id):

    user_data = {"id": id, "name": "AmanKC-neo"}

```

```
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production W
SGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.0.103:5000
Press CTRL+C to quit
127.0.0.1 - - [29/Jan/2025 10:23:49] "GET /users/login HTTP/1.1" 405 -
127.0.0.1 - - [29/Jan/2025 10:24:34] "GET /users/login HTTP/1.1" 405 -
127.0.0.1 - - [29/Jan/2025 10:25:12] "GET /users/1 HTTP/1.1" 200 -
```

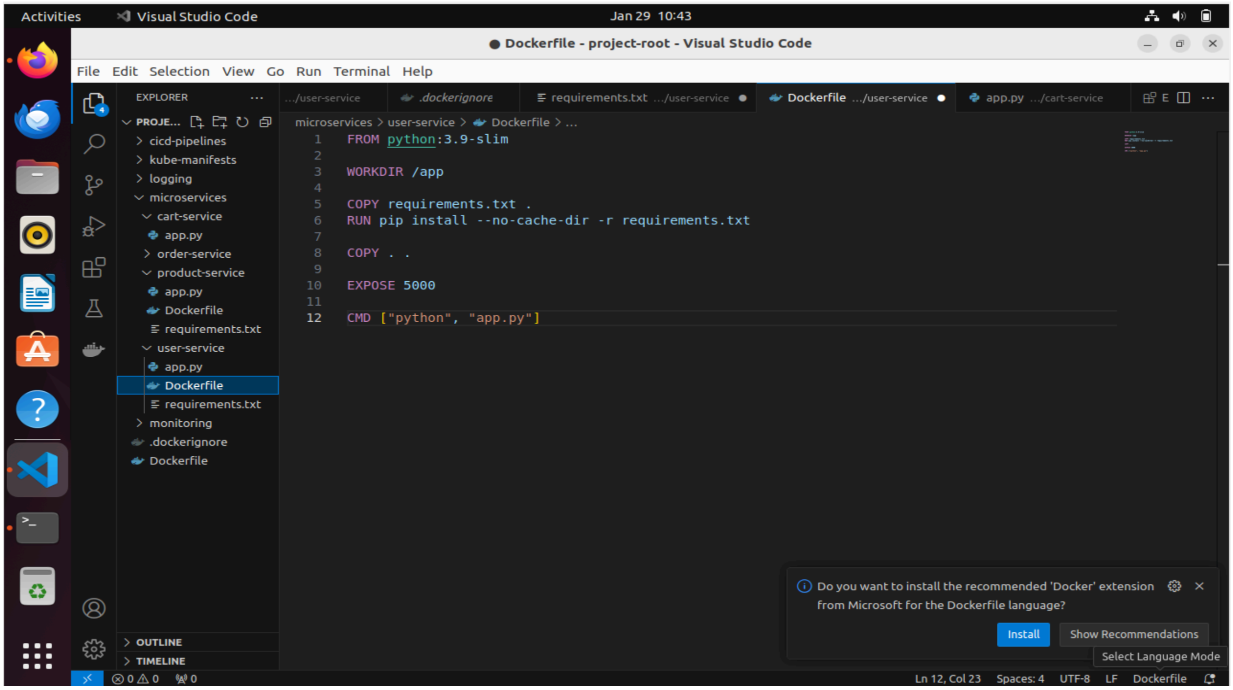


```
id:       1
name:     "AmanKC-neo"
```

#requirements.txt

flask
flask-cors
flask-jwt-extended
flask-sqlalchemy

#Dockerfile:



```
FROM python:3.9-slim

WORKDIR /app

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

COPY . .

EXPOSE 5000

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

Do you want to install the recommended 'Docker' extension from Microsoft for the Dockerfile language?

---

2. Product Service

Purpose: Manages the product catalog.

API Endpoints:

- `GET /products`: Retrieve all products.
- `GET /products/{id}`: Retrieve details of a single product.

Implementation:

`#app.py`:

```
from flask import Flask, jsonify, request

app = Flask(__name__)

products = [

    {"id": 1, "name": "Laptop", "price": 1000, "stock": 10},

    {"id": 2, "name": "Mouse", "price": 20, "stock": 200},

    {"id": 3, "name": "Keyboard", "price": 50, "stock": 150},

]
```

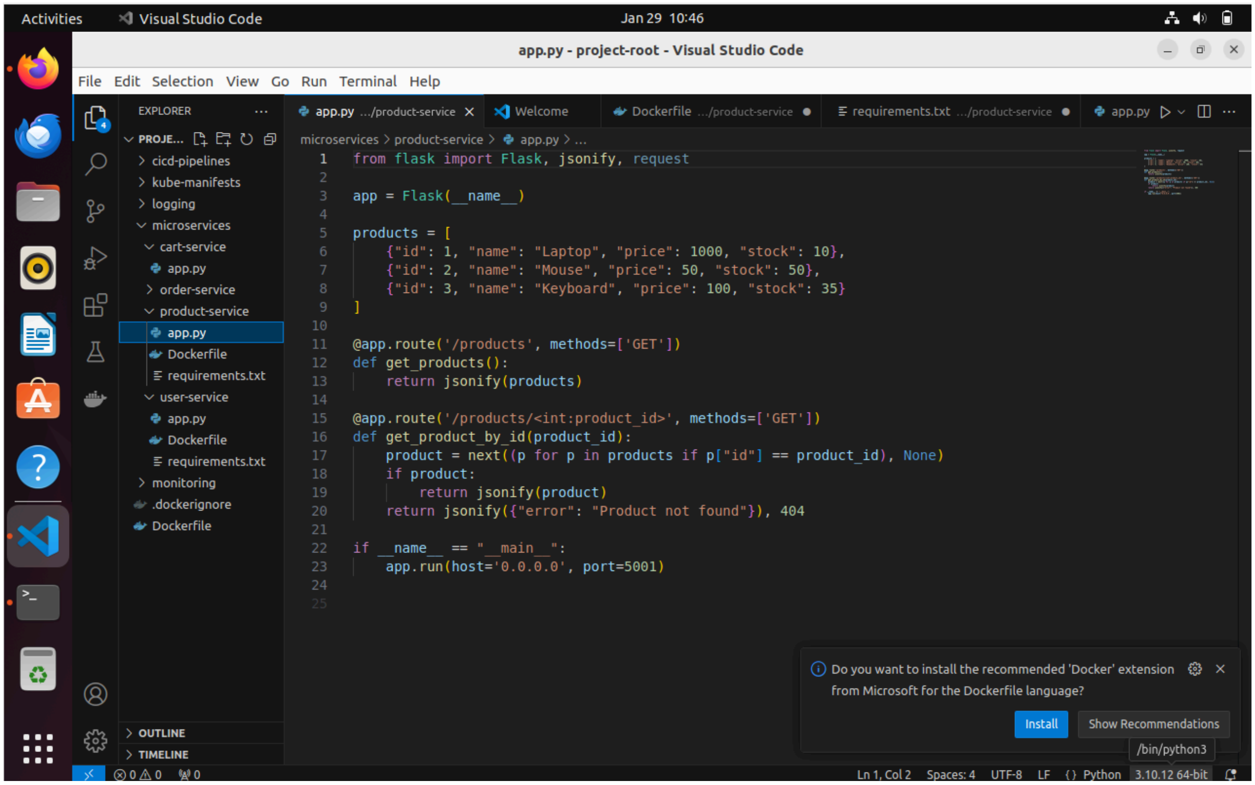
```
@app.route('/products', methods=['GET'])
def get_products():
    return jsonify(products)
@app.route('/products/<int:product_id>', methods=['GET'])
def get_product(product_id):
    product = next((p for p in products if p["id"] == product_id), None)
    if product:
        return jsonify(product)
    return jsonify({"error": "Product not found"}), 404

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5001)
```

#Dockerfile:

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

requirements.txt:

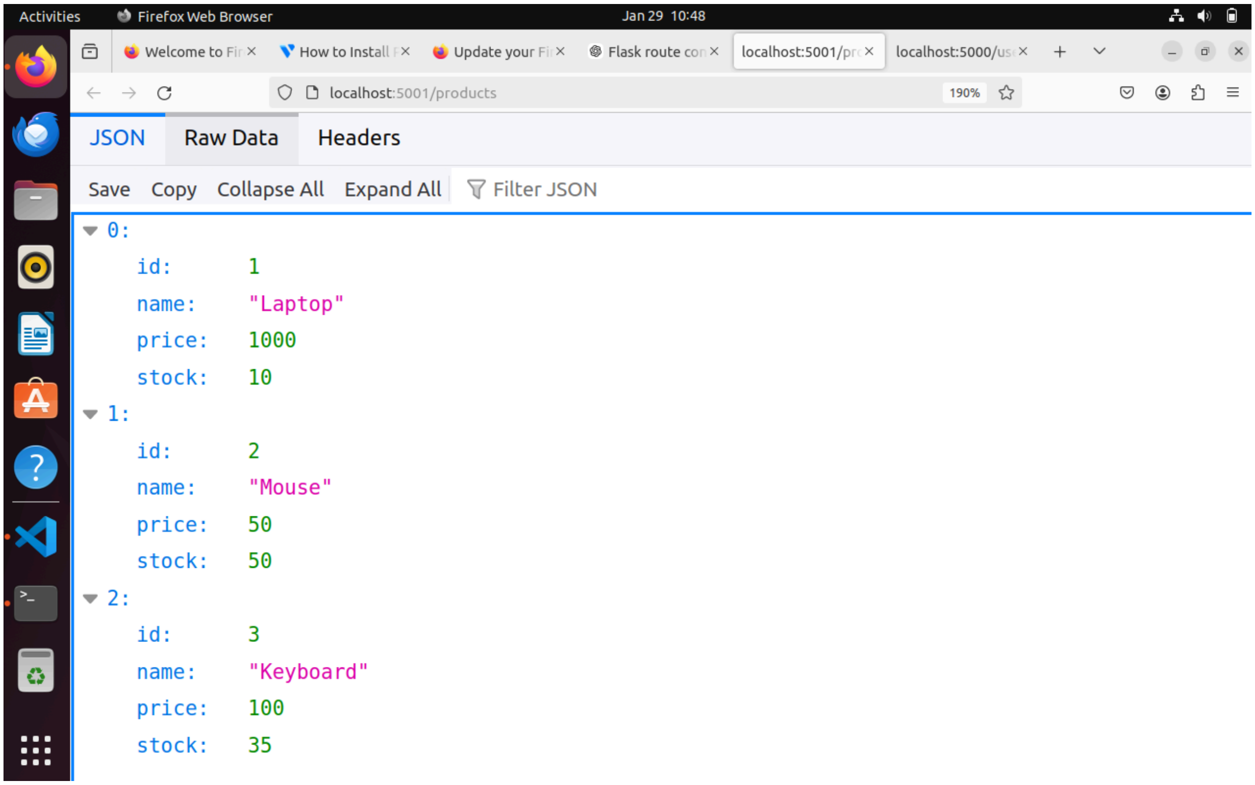
```
Flask==3.1.0
werkzeug==3.1.3
```



```
from flask import Flask, jsonify, request

app = Flask(__name__)

products = [
    {"id": 1, "name": "Laptop", "price": 1000, "stock": 10},
    {"id": 2, "name": "Mouse", "price": 50, "stock": 50},
    {"id": 3, "name": "Keyboard", "price": 100, "stock": 35}
]

@app.route('/products', methods=['GET'])
def get_products():
    return jsonify(products)

@app.route('/products/<int:product_id>', methods=['GET'])
def get_product_by_id(product_id):
    product = next((p for p in products if p["id"] == product_id), None)
    if product:
        return jsonify(product)
    return jsonify({"error": "Product not found"}), 404

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5001)
```

Do you want to install the recommended 'Docker' extension from Microsoft for the Dockerfile language?

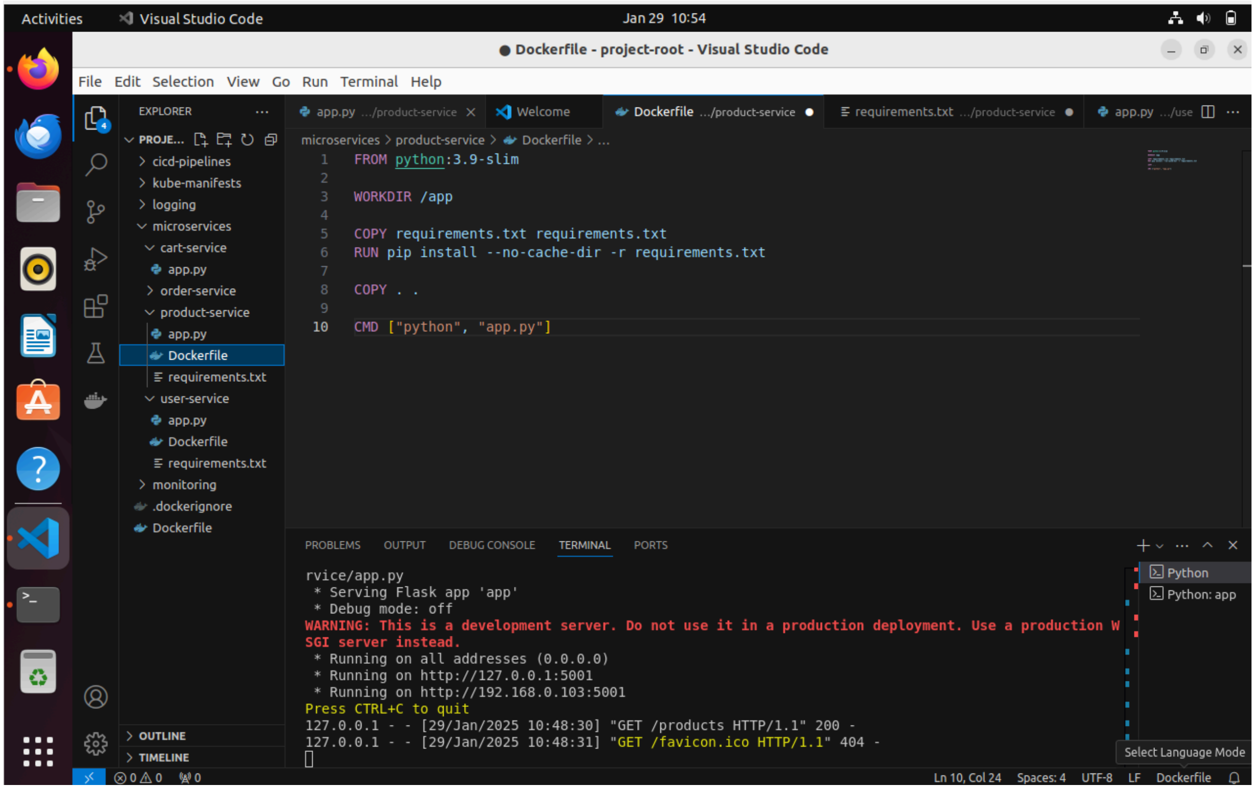
Install Show Recommendations



```
0:
    id:     1
    name:   "Laptop"
    price:  1000
    stock:  10
1:
    id:     2
    name:   "Mouse"
    price:  50
    stock:  50
2:
    id:     3
    name:   "Keyboard"
    price:  100
    stock:  35
```

#requirements.txt:

Flask==2.1.2



```
FROM python:3.9-slim

WORKDIR /app

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

COPY . .

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

```
rvice/app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production W
SGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5001
 * Running on http://192.168.0.103:5001
Press CTRL+C to quit
127.0.0.1 - - [29/Jan/2025 10:48:30] "GET /products HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2025 10:48:31] "GET /favicon.ico HTTP/1.1" 404 -
```

3. Cart Service

Purpose: Handles adding/removing products to/from the cart.

API Endpoints:

- `POST /cart`: Add an item to the cart (expects `product_id` and `quantity` in the request).
- `DELETE /cart/{item_id}`: Remove an item from the cart.

Implementation:

#app.py:

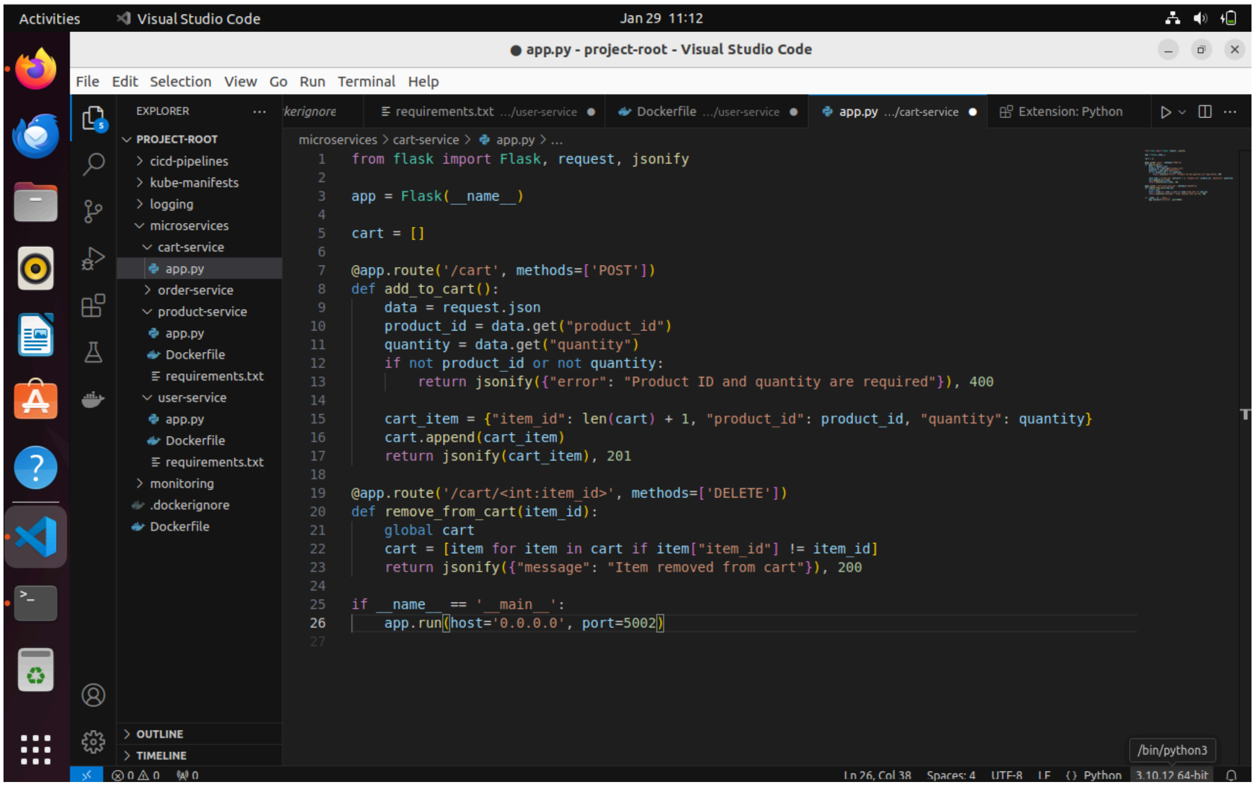
```
from flask import Flask, request, jsonify

app = Flask(__name__)

cart = []

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

def add_to_cart():

    data = request.json

    product_id = data.get("product_id")
    quantity = data.get("quantity")
    if not product_id or not quantity:
        return jsonify({"error": "Product ID and quantity are required"}), 400
    cart_item = {"item_id": len(cart) + 1, "product_id": product_id, "quantity": quantity}
    cart.append(cart_item)
    return jsonify(cart_item), 201
@app.route('/cart/<int:item_id>', methods=['DELETE'])
def remove_from_cart(item_id):
    global cart
    cart = [item for item in cart if item["item_id"] != item_id]
    return jsonify({"message": "Item removed from cart"}), 200
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5002)
```

#Dockerfile:

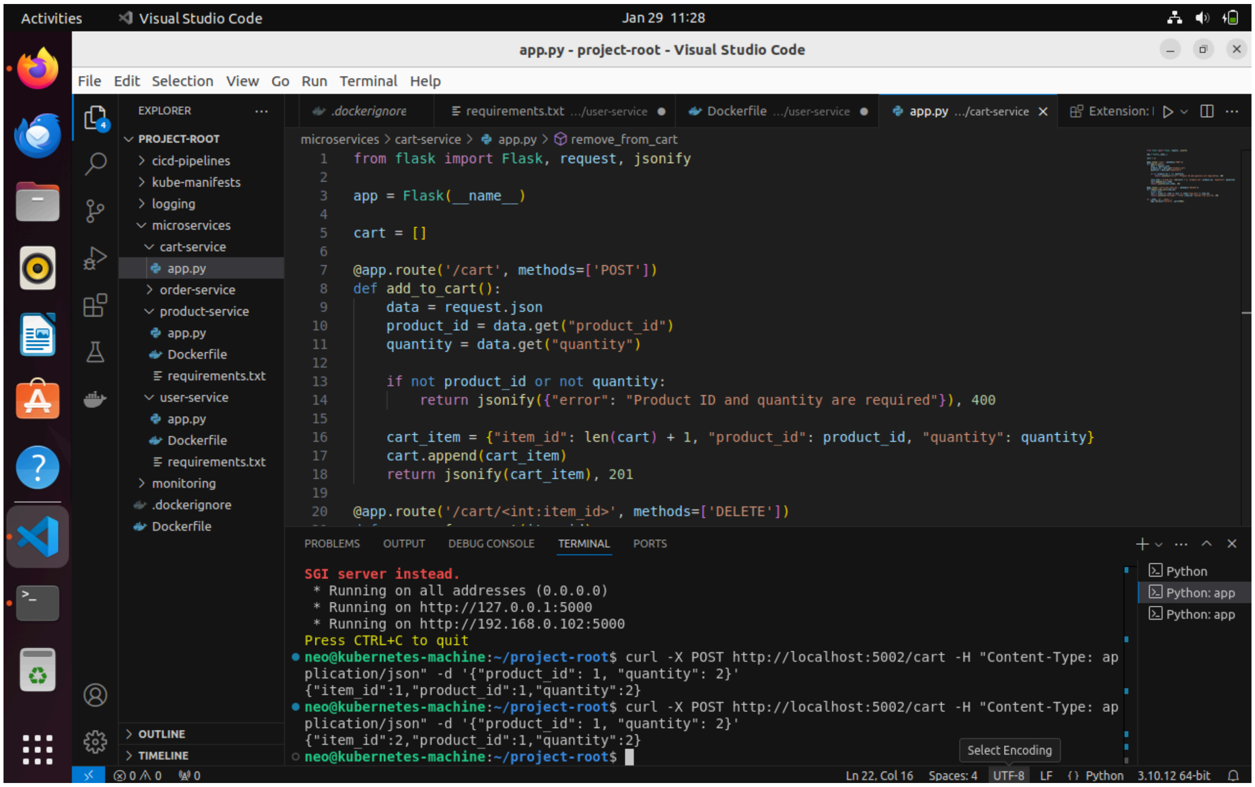
```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

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

requirements.txt:

```
Flask==2.1.2
werkzeug==3.1.3
```

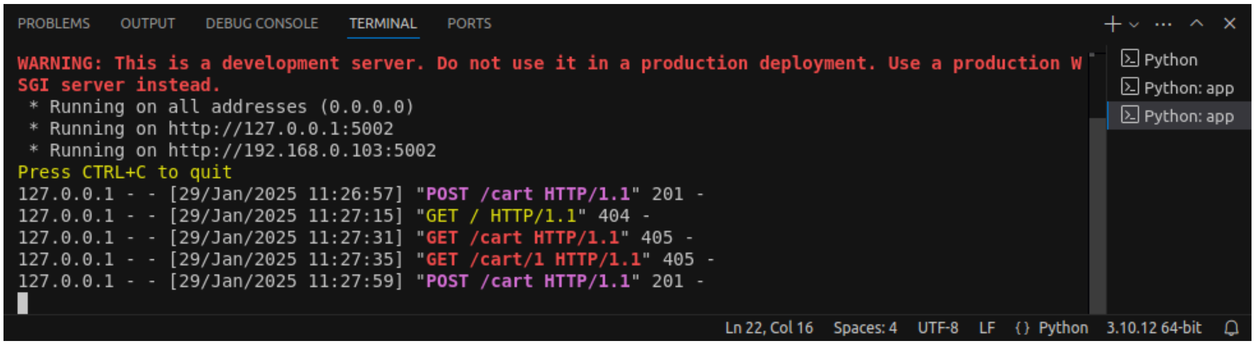

app.py - project-root - Visual Studio Code
microservices > cart-service > app.py > ...

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

cart = []

@app.route('/cart', methods=['POST'])
def add_to_cart():
    data = request.json
    product_id = data.get("product_id")
    quantity = data.get("quantity")
    if not product_id or not quantity:
        return jsonify({"error": "Product ID and quantity are required"}), 400

    cart_item = {"item_id": len(cart) + 1, "product_id": product_id, "quantity": quantity}
    cart.append(cart_item)
    return jsonify(cart_item), 201

@app.route('/cart/<int:item_id>', methods=['DELETE'])
def remove_from_cart(item_id):
    global cart
    cart = [item for item in cart if item["item_id"] != item_id]
    return jsonify({"message": "Item removed from cart"}), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5002)
```

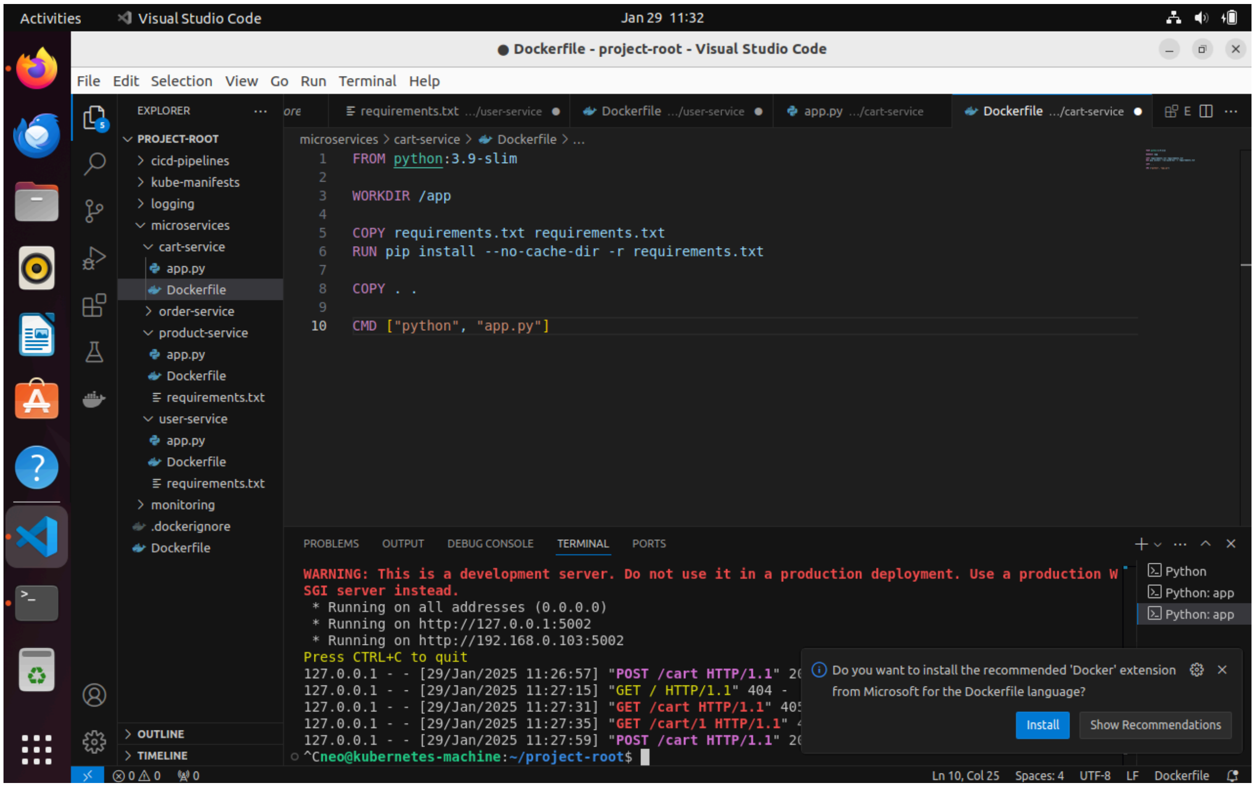

app.py - project-root - Visual Studio Code
microservices > cart-service > app.py > remove_from_cart

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

cart = []

@app.route('/cart', methods=['POST'])
def add_to_cart():
    data = request.json
    product_id = data.get("product_id")
    quantity = data.get("quantity")

    if not product_id or not quantity:
        return jsonify({"error": "Product ID and quantity are required"}), 400

    cart_item = {"item_id": len(cart) + 1, "product_id": product_id, "quantity": quantity}
    cart.append(cart_item)
    return jsonify(cart_item), 201

@app.route('/cart/<int:item_id>', methods=['DELETE'])
```

```
SGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.0.102:5000
Press CTRL+C to quit
neo@kubernetes-machine:~/project-root$ curl -X POST http://localhost:5002/cart -H "Content-Type: ap
plication/json" -d '{"product_id": 1, "quantity": 2}'
{"item_id":1,"product_id":1,"quantity":2}
neo@kubernetes-machine:~/project-root$ curl -X POST http://localhost:5002/cart -H "Content-Type: ap
plication/json" -d '{"product_id": 1, "quantity": 2}'
{"item_id":2,"product_id":1,"quantity":2}
neo@kubernetes-machine:~/project-root$
```



```
WARNING: This is a development server. Do not use it in a production deployment. Use a production W
SGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5002
 * Running on http://192.168.0.103:5002
Press CTRL+C to quit
127.0.0.1 - - [29/Jan/2025 11:26:57] "POST /cart HTTP/1.1" 201 -
127.0.0.1 - - [29/Jan/2025 11:27:15] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [29/Jan/2025 11:27:31] "GET /cart HTTP/1.1" 405 -
127.0.0.1 - - [29/Jan/2025 11:27:35] "GET /cart/1 HTTP/1.1" 405 -
127.0.0.1 - - [29/Jan/2025 11:27:59] "POST /cart HTTP/1.1" 201 -
```



```
FROM python:3.9-slim

WORKDIR /app

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

COPY . .

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

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production W
SGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5002
 * Running on http://192.168.0.103:5002
Press CTRL+C to quit
127.0.0.1 - - [29/Jan/2025 11:26:57] "POST /cart HTTP/1.1" 2(
127.0.0.1 - - [29/Jan/2025 11:27:15] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [29/Jan/2025 11:27:31] "GET /cart HTTP/1.1" 405
127.0.0.1 - - [29/Jan/2025 11:27:35] "GET /cart/1 HTTP/1.1" 4
127.0.0.1 - - [29/Jan/2025 11:27:59] "POST /cart HTTP/1.1" 2(
^Cneo@kubernetes-machine:~/project-root$
```

#requirements.txt

Flask==2.1.2

werkzeug==3.1.3

4. Order Service

Purpose: Processes and manages orders.

API Endpoints:

- `POST /order`: Place an order.
- `GET /order/{id}`: Fetch order details.

Implementation:

`#app.py`:

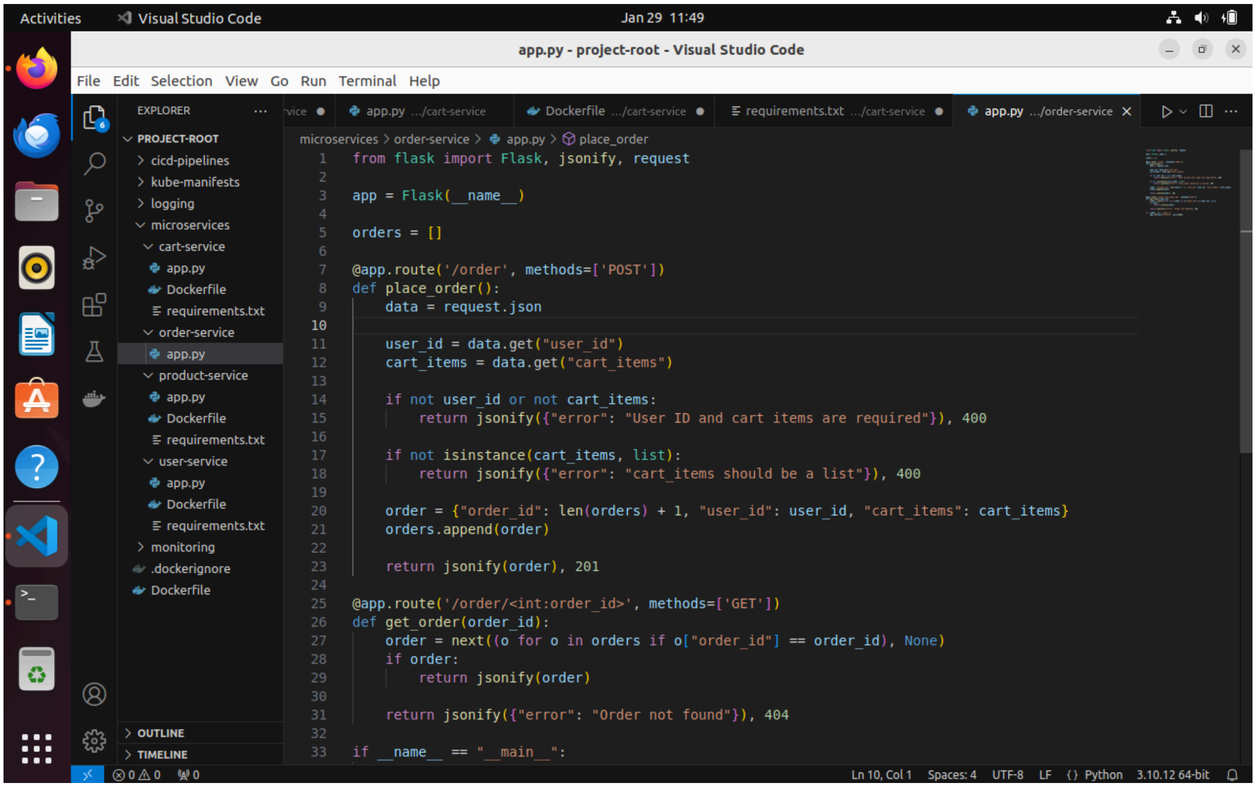
```
from flask import Flask, jsonify, request

app = Flask(__name__)

orders = []

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

def place_order():

    data = request.json

    user_id = data.get("user_id")

    cart_items = data.get("cart_items")

    if not user_id or not cart_items:

        return jsonify({"error": "User ID and cart items are required"}), 400

    order = {"order_id": len(orders) + 1, "user_id": user_id, "cart_items": cart_items}

    orders.append(order)

    return jsonify(order), 201

@app.route('/order/<int:order_id>', methods=['GET'])

def get_order(order_id):

    order = next((o for o in orders if o["order_id"] == order_id), None)

    if order:
        return jsonify(order)

    return jsonify({"error": "Order not found"}), 404

if __name__ == "__main__":

    app.run(host='0.0.0.0', port=5003)
```

#Dockerfile:

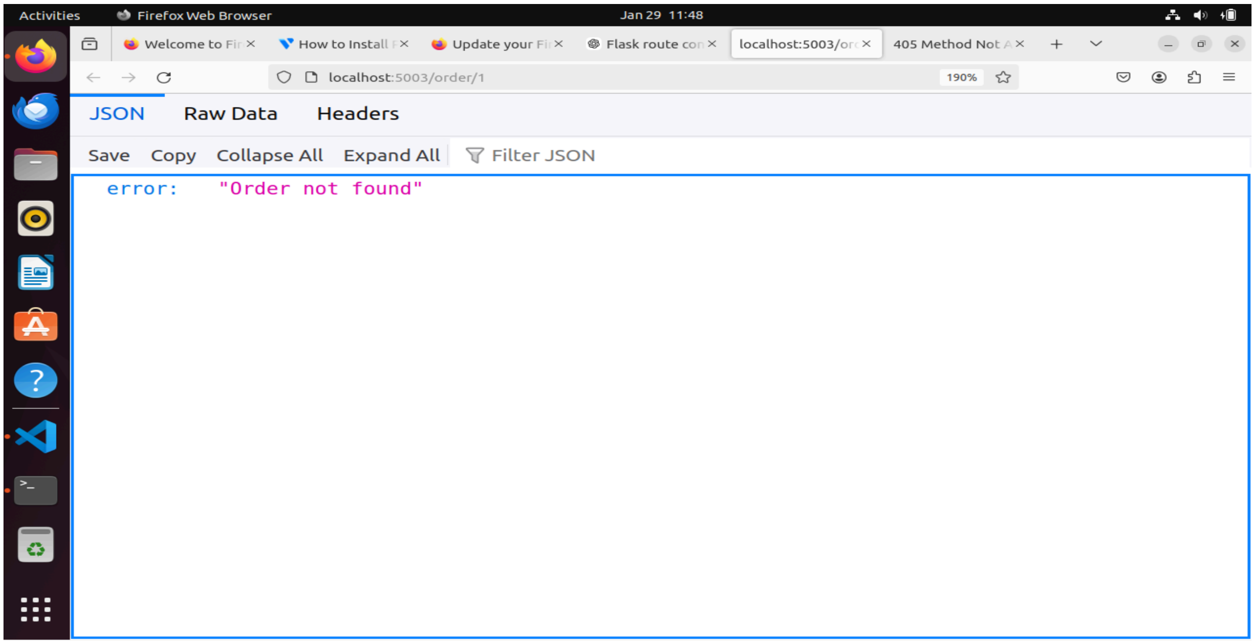
```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt

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

COPY . .

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

Requirements.txt:

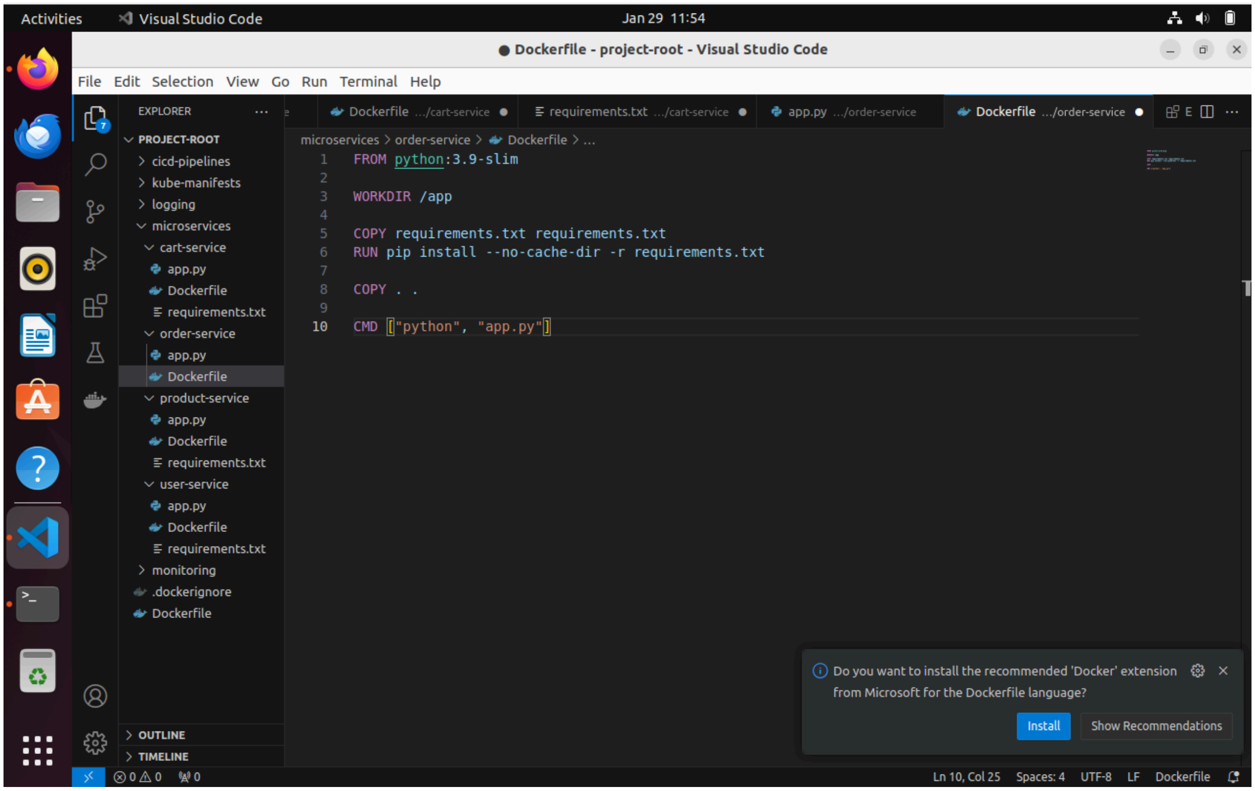
```
Flask==3.1.0

werkzeug==3.1.3
```



```
from flask import Flask, jsonify, request

app = Flask(__name__)

orders = []

@app.route('/order', methods=['POST'])
def place_order():
    data = request.json

    user_id = data.get("user_id")
    cart_items = data.get("cart_items")

    if not user_id or not cart_items:
        return jsonify({"error": "User ID and cart items are required"}), 400

    if not isinstance(cart_items, list):
        return jsonify({"error": "cart_items should be a list"}), 400

    order = {"order_id": len(orders) + 1, "user_id": user_id, "cart_items": cart_items}
    orders.append(order)

    return jsonify(order), 201

@app.route('/order/<int:order_id>', methods=['GET'])
def get_order(order_id):
    order = next((o for o in orders if o["order_id"] == order_id), None)
    if order:
        return jsonify(order)

    return jsonify({"error": "Order not found"}), 404

if __name__ == "__main__":
```



```
error:    "Order not found"
```



```
FROM python:3.9-slim

WORKDIR /app

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

COPY . .

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

Do you want to install the recommended 'Docker' extension from Microsoft for the Dockerfile language?

Data Flow:

1. **User Service**: Provides authenticated user information.
2. **Product Service**: Supplies product details for cart calculations.
3. **Cart Service**: Handles cart operations and forwards data to the **Order Service**.
4. **Order Service**: Completes the process by creating orders.

Test Locally:

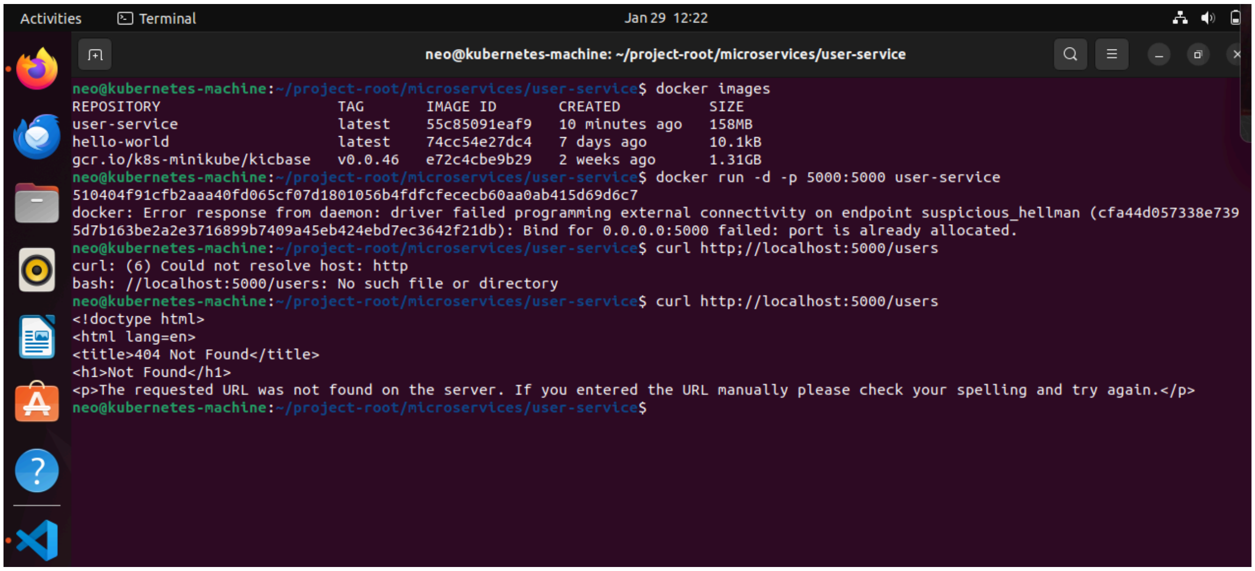
Build and run the container locally:

#user-service docker build:

```
cd microservices/user-service

docker build -t user-service .

docker run -d -p 5000:5000 user-service
```

Test endpoints using `curl` or Postman:

```
curl http://localhost:5000/users
```


Activities
Terminal
Jan 29 12:22
neo@kubernetes-machine: ~/project-root/microservices/user-service

```
neo@kubernetes-machine:~/project-root/microservices/user-service$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
user-service                  latest    55c85091eaf9   10 minutes ago   158MB
hello-world                   latest    74cc54e27dc4   7 days ago       10.1kB
gcr.io/k8s-minikube/kicbase   v0.0.46   e72c4cbe9b29   2 weeks ago      1.31GB
neo@kubernetes-machine:~/project-root/microservices/user-service$ docker run -d -p 5000:5000 user-service
510404f91cfb2aaa40fd065cf07d1801056b4fdfcfececb60aa0ab415d69d6c7
docker: Error response from daemon: driver failed programming external connectivity on endpoint suspicious_hellman (cfa44d057338e739
5d7b163be2a2e3716899b7409a45eb424ebd7ec3642f21db): Bind for 0.0.0.0:5000 failed: port is already allocated.
neo@kubernetes-machine:~/project-root/microservices/user-service$ curl http;//localhost:5000/users
curl: (6) Could not resolve host: http
bash: //localhost:5000/users: No such file or directory
neo@kubernetes-machine:~/project-root/microservices/user-service$ curl http://localhost:5000/users
<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>
neo@kubernetes-machine:~/project-root/microservices/user-service$
```

#product-service docker build:

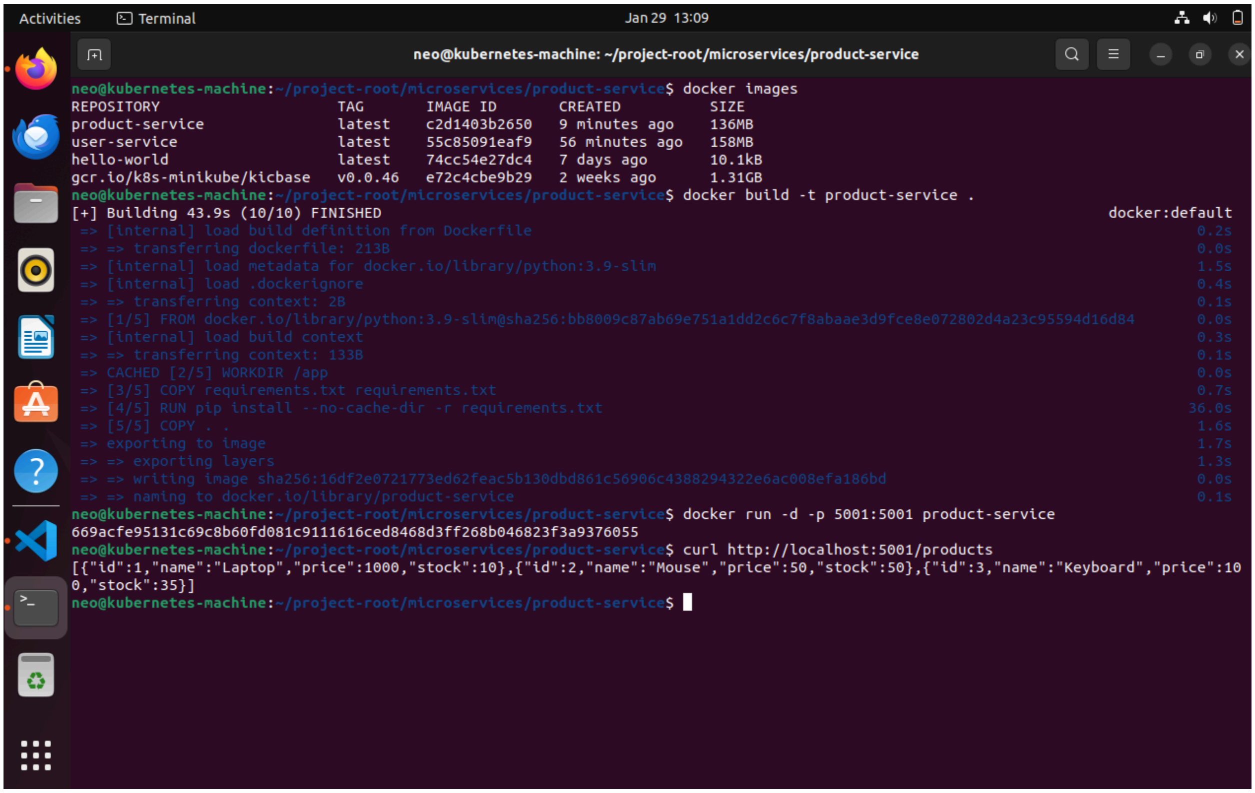
```
cd microservices/product-service
```

```
docker build -t product-service .
```

```
docker run -d -p 5001:5001 product-service
```

Test endpoints using `curl` or Postman:

```
curl http://localhost:5001/products
```



Activities
Terminal
Jan 29 13:09
neo@kubernetes-machine: ~/project-root/microservices/product-service

```
neo@kubernetes-machine:~/project-root/microservices/product-service$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
product-service               latest    c2d1403b2650   9 minutes ago    136MB
user-service                  latest    55c85091eaf9   56 minutes ago   158MB
hello-world                   latest    74cc54e27dc4   7 days ago       10.1kB
gcr.io/k8s-minikube/kicbase   v0.0.46   e72c4cbe9b29   2 weeks ago      1.31GB
neo@kubernetes-machine:~/project-root/microservices/product-service$ docker build -t product-service .
[+] Building 43.9s (10/10) FINISHED                                                                docker:default
 => [internal] load build definition from Dockerfile                                                          0.2s
 => => transferring dockerfile: 213B                                                                          0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                            1.5s
 => [internal] load .dockerignore                                                                             0.4s
 => => transferring context: 2B                                                                               0.1s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fce8e072802d4a23c95594d16d84   0.0s
 => [internal] load build context                                                                             0.3s
 => => transferring context: 133B                                                                             0.1s
 => CACHED [2/5] WORKDIR /app                                                                                 0.0s
 => [3/5] COPY requirements.txt requirements.txt                                                              0.7s
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                 36.0s
 => [5/5] COPY . .                                                                                            1.6s
 => exporting to image                                                                                        1.7s
 => => exporting layers                                                                                       1.3s
 => => writing image sha256:16df2e0721773ed62feac5b130dbd861c56906c4388294322e6ac008efa186bd                  0.0s
 => => naming to docker.io/library/product-service                                                            0.1s
neo@kubernetes-machine:~/project-root/microservices/product-service$ docker run -d -p 5001:5001 product-service
669acfe95131c69c8b60fd081c9111616ced8468d3ff268b046823f3a9376055
neo@kubernetes-machine:~/project-root/microservices/product-service$ curl http://localhost:5001/products
[{"id":1,"name":"Laptop","price":1000,"stock":10},{"id":2,"name":"Mouse","price":50,"stock":50},{"id":3,"name":"Keyboard","price":10
0,"stock":35}]
neo@kubernetes-machine:~/project-root/microservices/product-service$ 
```

#cart-service docker build:

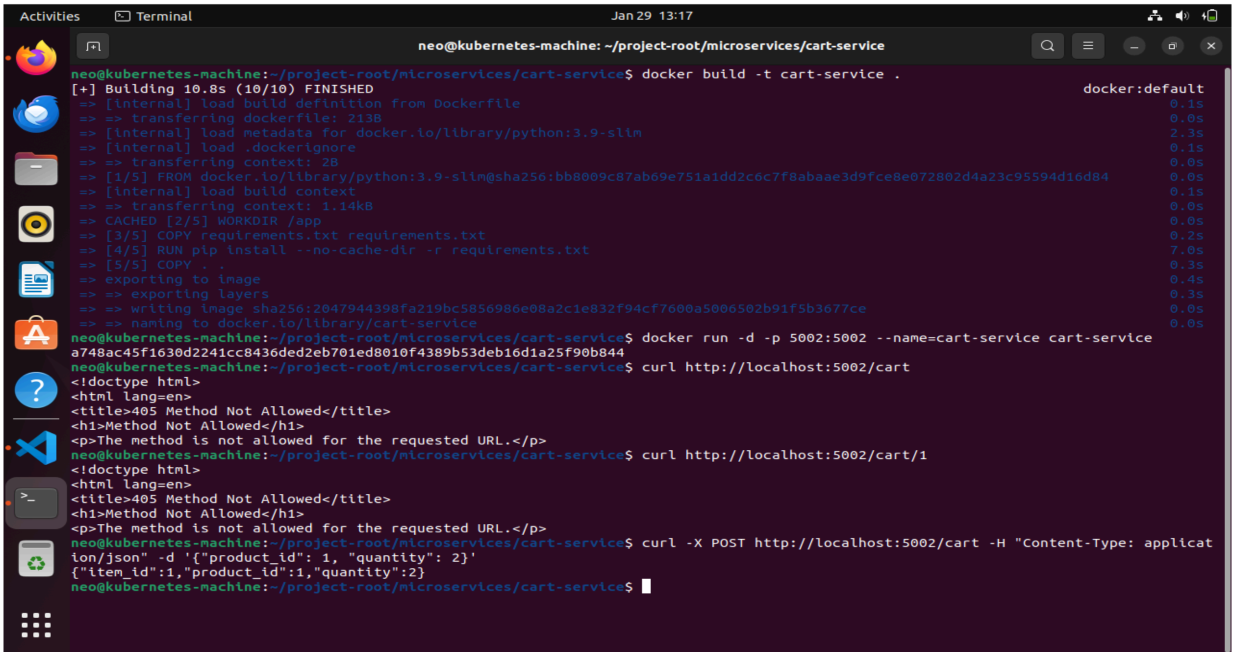
```
cd microservices/cart-service
```

```
docker build -t cart-service .
```

```
docker run -d -p 5000:5000 cart-service
```

Test endpoints using `curl` or Postman:

curl -X POST http://localhost:5002/cart -H “Content-Type:application/json” -d ‘{“user_id:1, “cart_items’:[{“product_id:1}, “quantity”:2}]}’



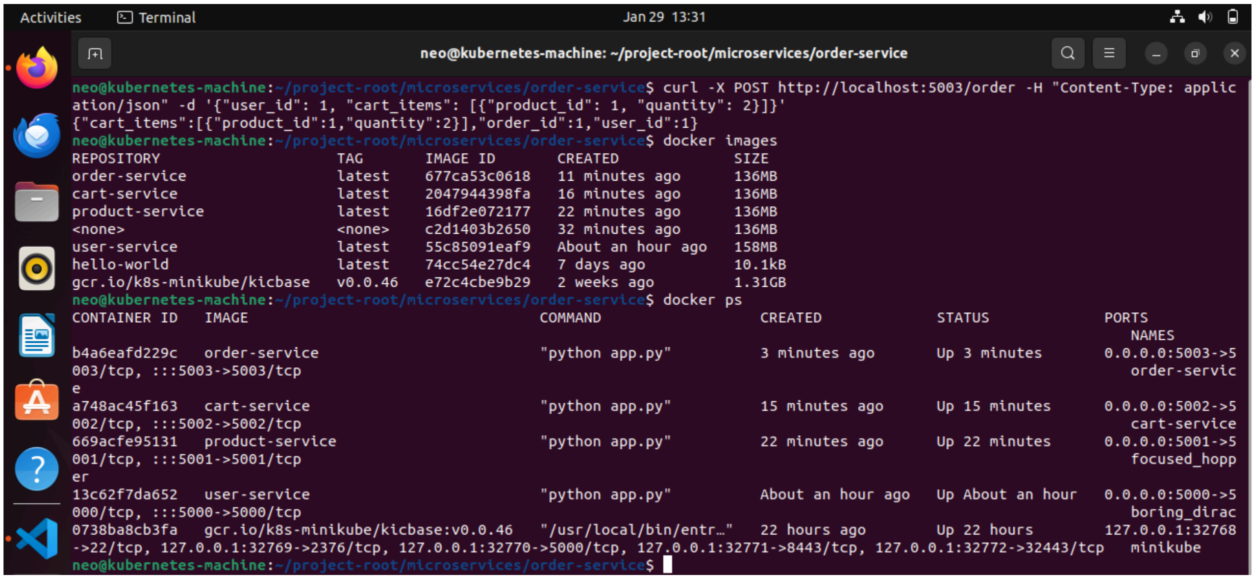
```
neo@kubernetes-machine:~/project-root/microservices/cart-service$ docker build -t cart-service .
[+] Building 10.8s (10/10) FINISHED                                                        docker:default
 => [internal] load build definition from Dockerfile                                                 0.1s
 => => transferring dockerfile: 213B                                                                 0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                   2.3s
 => [internal] load .dockerignore                                                                    0.1s
 => => transferring context: 2B                                                                      0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fce8e072802d4a23c95594d16d84  0.0s
 => [internal] load build context                                                                    0.1s
 => => transferring context: 1.14kB                                                                  0.0s
 => CACHED [2/5] WORKDIR /app                                                                        0.0s
 => [3/5] COPY requirements.txt requirements.txt                                                     0.2s
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt                                         7.0s
 => [5/5] COPY . .                                                                                   0.3s
 => exporting to image                                                                               0.4s
 => => exporting layers                                                                              0.3s
 => => writing image sha256:2047944398fa219bc5856986e08a2c1e832f94cf7600a5006502b91f5b3677ce         0.0s
 => => naming to docker.io/library/cart-service                                                      0.0s
neo@kubernetes-machine:~/project-root/microservices/cart-service$ docker run -d -p 5002:5002 --name=cart-service cart-service
a748ac45f1630d2241cc8436ded2eb701ed8010f4389b53deb16d1a25f90b844
neo@kubernetes-machine:~/project-root/microservices/cart-service$ curl http://localhost:5002/cart
<!doctype html>
<html lang=en>
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
neo@kubernetes-machine:~/project-root/microservices/cart-service$ curl http://localhost:5002/cart/1
<!doctype html>
<html lang=en>
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
neo@kubernetes-machine:~/project-root/microservices/cart-service$ curl -X POST http://localhost:5002/cart -H "Content-Type: applicat
ion/json" -d '{"product_id": 1, "quantity": 2}'
{"item_id":1,"product_id":1,"quantity":2}
neo@kubernetes-machine:~/project-root/microservices/cart-service$ 
```

#order-service docker build:

```
cd microservices/order-service

docker build -t order-service .

docker run -d -p 5003:5003 order-service
```

Test endpoints using `curl` or Postman:

curl -X POST http://localhost:5003/order -H "Content-Type:application/json" -d '{"user_id:1, "cart_items': [{"product_id:1}, "quantity":2}]}'



```
neo@kubernetes-machine:~/project-root/microservices/order-service$ curl -X POST http://localhost:5003/order -H "Content-Type: applic
ation/json" -d '{"user_id": 1, "cart_items": [{"product_id": 1, "quantity": 2}]}'
{"cart_items":[{"product_id":1,"quantity":2}],"order_id":1,"user_id":1}
neo@kubernetes-machine:~/project-root/microservices/order-service$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED              SIZE
order-service                 latest    677ca53c0618   11 minutes ago       136MB
cart-service                  latest    2047944398fa   16 minutes ago       136MB
product-service               latest    16df2e072177   22 minutes ago       136MB
<none>                        <none>    c2d1403b2650   32 minutes ago       136MB
user-service                  latest    55c85091eaf9   About an hour ago    158MB
hello-world                   latest    74cc54e27dc4   7 days ago           10.1kB
gcr.io/k8s-minikube/kicbase   v0.0.46   e72c4cbe9b29   2 weeks ago          1.31GB
neo@kubernetes-machine:~/project-root/microservices/order-service$ docker ps
CONTAINER ID   IMAGE                                 COMMAND                  CREATED             STATUS             PORTS
                                                                                                                      NAMES
b4a6eafd229c   order-service                         "python app.py"          3 minutes ago       Up 3 minutes       0.0.0.0:5003->5
003/tcp, :::5003->5003/tcp                                                                                            order-servic
e
a748ac45f163   cart-service                          "python app.py"          15 minutes ago      Up 15 minutes      0.0.0.0:5002->5
002/tcp, :::5002->5002/tcp                                                                                            cart-service
669acfe95131   product-service                       "python app.py"          22 minutes ago      Up 22 minutes      0.0.0.0:5001->5
001/tcp, :::5001->5001/tcp                                                                                            focused_hopp
er
13c62f7da652   user-service                          "python app.py"          About an hour ago   Up About an hour   0.0.0.0:5000->5
000/tcp, :::5000->5000/tcp                                                                                            boring_dirac
0738ba8cb3fa   gcr.io/k8s-minikube/kicbase:v0.0.46   "/usr/local/bin/entr…"   22 hours ago        Up 22 hours        127.0.0.1:32768
->22/tcp, 127.0.0.1:32769->2376/tcp, 127.0.0.1:32770->5000/tcp, 127.0.0.1:32771->8443/tcp, 127.0.0.1:32772->32443/tcp   minikube
neo@kubernetes-machine:~/project-root/microservices/order-service$ 
```

* You can also test locally via Docker Compose similar to below given instructions*

1. Setting Up `docker-compose.yml`

Since our microservices need to communicate, we will use **Docker Compose** to define and run them together in a local network.

Folder Structure:

Inside your project directory (`project-root/`), create a `docker-compose.yml` file:

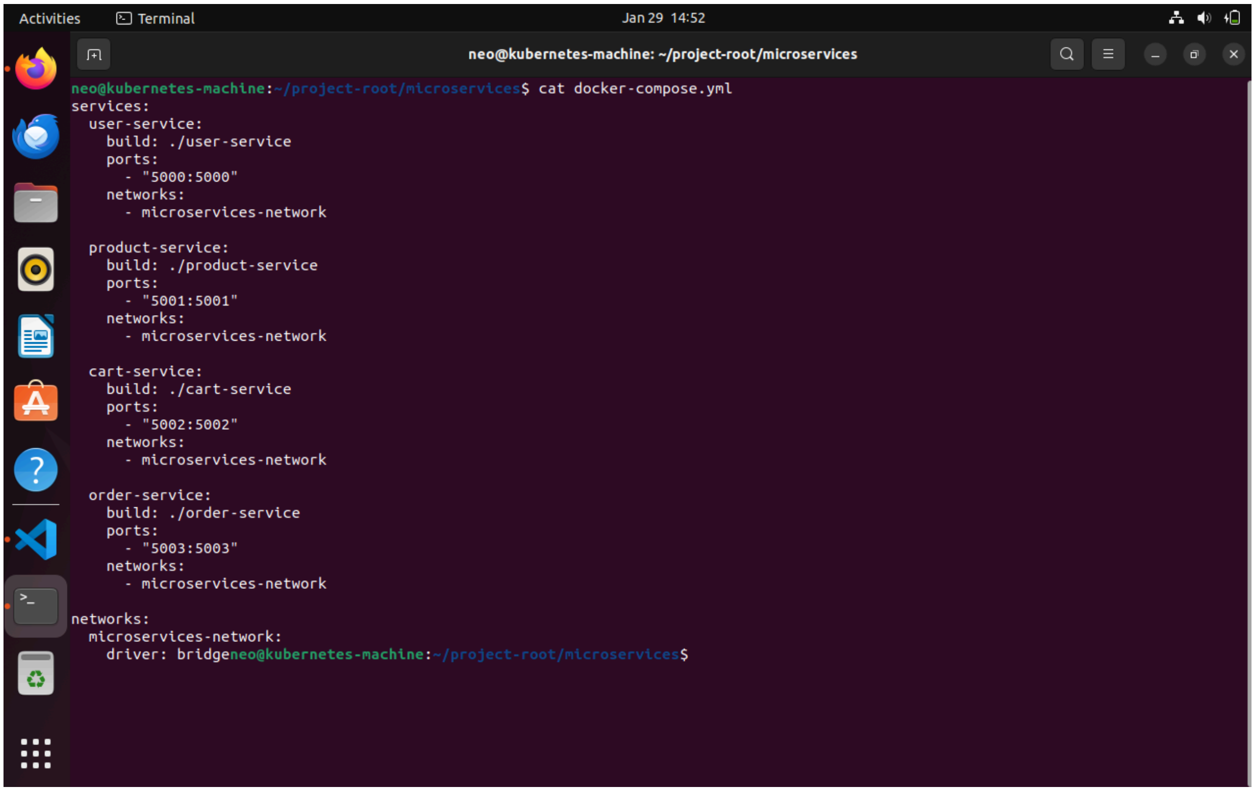
```
project-root/
|── microservices/
|   ├── user-service/
|   |   ├── app.py  (Flask/Express API)
|   |   ├── Dockerfile
|   |   ├── requirements.txt  (if using Python)
|   ├── product-service/
|   ├── cart-service/
|   ├── order-service/
|── docker-compose.yml
|── .env
```

`docker-compose.yml` Example

This file defines:

- Four microservices (User, Product, Cart, Order).
- A shared **network** for service-to-service communication.
- Database (SQLLite) for persistence.
- **Environment variables** to pass configuration settings.

```yaml
version: "3.8"

services:
  user-service:
    build: ./microservices/user-service
    ports:
      - "5001:5000"
    networks:
      - microservices-network

  product-service:
    build: ./microservices/product-service
    ports:
      - "5002:5000"
    networks:
      - microservices-network

  cart-service:
    build: ./microservices/cart-service
    ports:
      - "5003:5000"
    networks:
      - microservices-network

  order-service:
    build: ./microservices/order-service
    ports:

      - "5004:5000"

    networks:

      - microservices-network


networks:

  microservices-network:

    driver: bridge
```



```
neo@kubernetes-machine:~/project-root/microservices$ cat docker-compose.yml
services:
  user-service:
    build: ./user-service
    ports:
      - "5000:5000"
    networks:
      - microservices-network

  product-service:
    build: ./product-service
    ports:
      - "5001:5001"
    networks:
      - microservices-network

  cart-service:
    build: ./cart-service
    ports:
      - "5002:5002"
    networks:
      - microservices-network

  order-service:
    build: ./order-service
    ports:
      - "5003:5003"
    networks:
      - microservices-network

networks:
  microservices-network:
    driver: bridgeneo@kubernetes-machine:~/project-root/microservices$
```

2. Modify Each Microservice to Use SQLite

Each microservice (User, Product, Cart, Order) is using SQLite as its database.

Update `app.py` in Each Microservice

Modify the database configuration to use SQLite:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///service.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)

with app.app_context():
    db.create_all()
```

✅ Repeat this in **User Service, Product Service, Cart Service, and Order Service**.
✅ Each service will now store its data in a **local SQLite file** (e.g., `service.db`).

3. Running Docker Compose

Once the `docker-compose.yml` file is set up, navigate to the project root and run:

```
docker-compose up --build
```

- The `--build` flag ensures that Docker rebuilds images if changes were made.
- Logs will show the services starting up.

To run in **detached mode** (background), use:

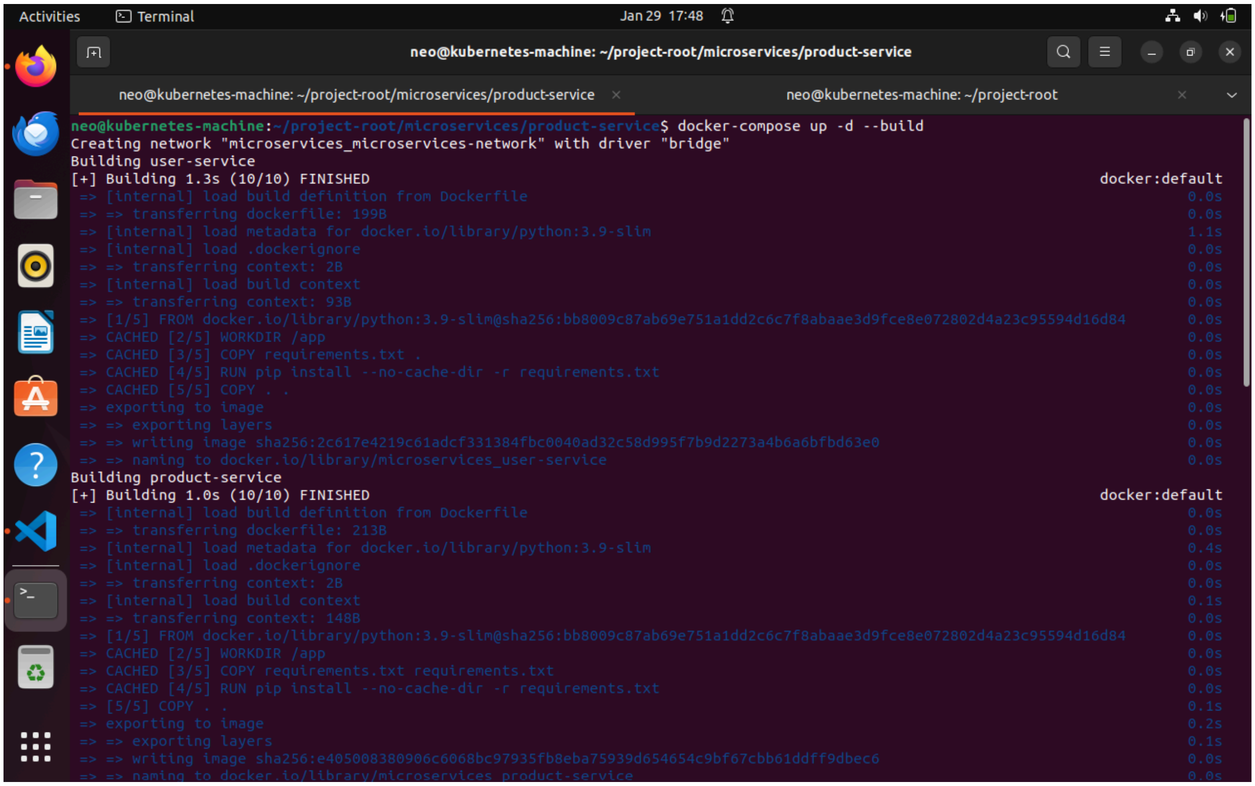
```
docker-compose up -d
```

To check running services:

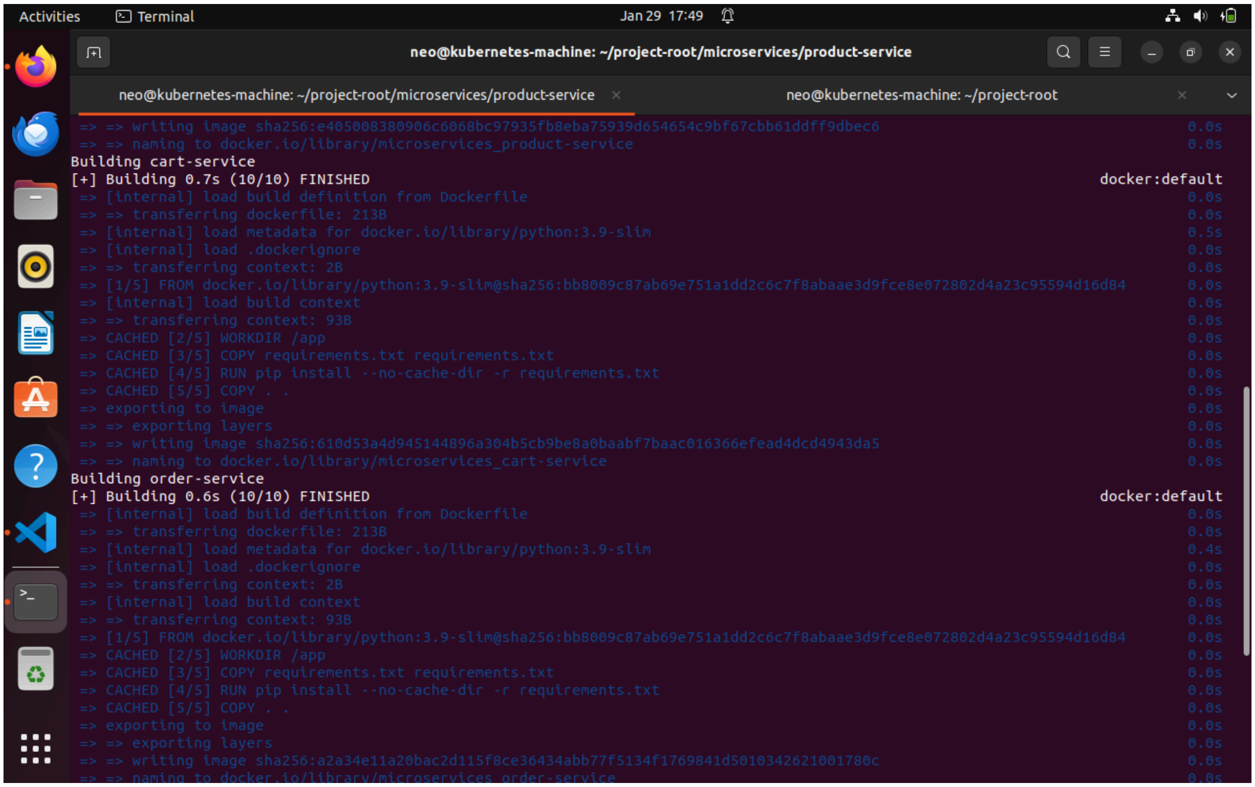
```
docker ps
```

To stop the services:

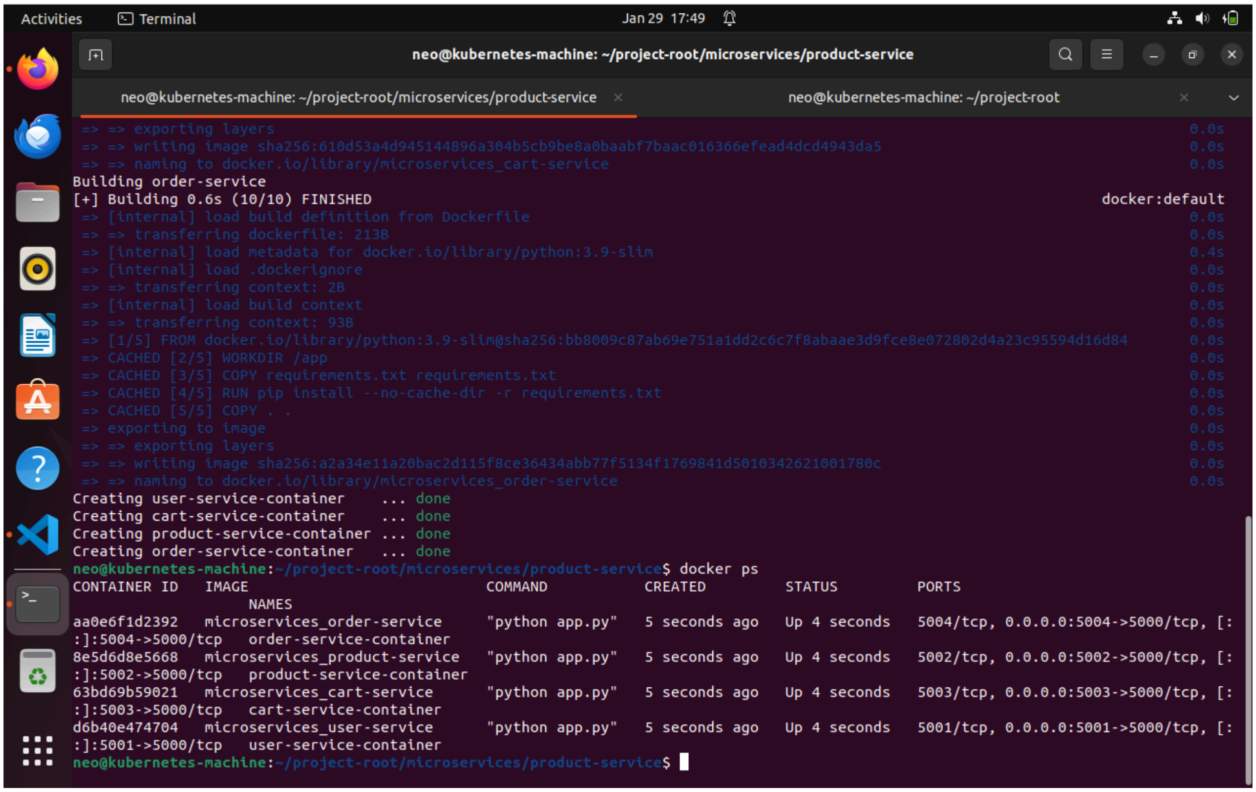
```
docker-compose down
```


Activities
Terminal
Jan 29 17:48
neo@kubernetes-machine: ~/project-root/microservices/product-service
neo@kubernetes-machine: ~/project-root/microservices/product-service
neo@kubernetes-machine: ~/project-root

```
neo@kubernetes-machine:~/project-root/microservices/product-service$ docker-compose up -d --build
Creating network "microservices_microservices-network" with driver "bridge"
Building user-service
[+] Building 1.3s (10/10) FINISHED                                                                                   docker:default
 => [internal] load build definition from Dockerfile                                                                           0.0s
 => => transferring dockerfile: 199B                                                                                           0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                             1.1s
 => [internal] load .dockerignore                                                                                              0.0s
 => => transferring context: 2B                                                                                                0.0s
 => [internal] load build context                                                                                              0.0s
 => => transferring context: 93B                                                                                               0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fce8e072802d4a23c95594d16d84       0.0s
 => CACHED [2/5] WORKDIR /app                                                                                                  0.0s
 => CACHED [3/5] COPY requirements.txt .                                                                                       0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                            0.0s
 => CACHED [5/5] COPY . .                                                                                                      0.0s
 => exporting to image                                                                                                         0.0s
 => => exporting layers                                                                                                        0.0s
 => => writing image sha256:2c617e4219c61adcf331384fbc0040ad32c58d995f7b9d2273a4b6a6bfbd63e0                                   0.0s
 => => naming to docker.io/library/microservices_user-service                                                                  0.0s
Building product-service
[+] Building 1.0s (10/10) FINISHED                                                                                   docker:default
 => [internal] load build definition from Dockerfile                                                                           0.0s
 => => transferring dockerfile: 213B                                                                                           0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                             0.4s
 => [internal] load .dockerignore                                                                                              0.0s
 => => transferring context: 2B                                                                                                0.0s
 => [internal] load build context                                                                                              0.1s
 => => transferring context: 148B                                                                                              0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fce8e072802d4a23c95594d16d84       0.0s
 => CACHED [2/5] WORKDIR /app                                                                                                  0.0s
 => CACHED [3/5] COPY requirements.txt requirements.txt                                                                        0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                            0.0s
 => [5/5] COPY . .                                                                                                             0.1s
 => exporting to image                                                                                                         0.2s
 => => exporting layers                                                                                                        0.1s
 => => writing image sha256:e405008380906c6068bc97935fb8eba75939d654654c9bf67cbb61ddff9dbec6                                   0.0s
 => => naming to docker.io/library/microservices_product-service                                                               0.0s
```


Activities
Terminal
Jan 29 17:49
neo@kubernetes-machine: ~/project-root/microservices/product-service
neo@kubernetes-machine: ~/project-root/microservices/product-service
neo@kubernetes-machine: ~/project-root

```
 => => writing image sha256:e405008380906c6068bc97935fb8eba75939d654654c9bf67cbb61ddff9dbec6                                   0.0s
 => => naming to docker.io/library/microservices_product-service                                                               0.0s
Building cart-service
[+] Building 0.7s (10/10) FINISHED                                                                                   docker:default
 => [internal] load build definition from Dockerfile                                                                           0.0s
 => => transferring dockerfile: 213B                                                                                           0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                             0.5s
 => [internal] load .dockerignore                                                                                              0.0s
 => => transferring context: 2B                                                                                                0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fce8e072802d4a23c95594d16d84       0.0s
 => [internal] load build context                                                                                              0.0s
 => => transferring context: 93B                                                                                               0.0s
 => CACHED [2/5] WORKDIR /app                                                                                                  0.0s
 => CACHED [3/5] COPY requirements.txt requirements.txt                                                                        0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                            0.0s
 => CACHED [5/5] COPY . .                                                                                                      0.0s
 => exporting to image                                                                                                         0.0s
 => => exporting layers                                                                                                        0.0s
 => => writing image sha256:610d53a4d945144896a304b5cb9be8a0baabf7baac016366efead4dcd4943da5                                   0.0s
 => => naming to docker.io/library/microservices_cart-service                                                                  0.0s
Building order-service
[+] Building 0.6s (10/10) FINISHED                                                                                   docker:default
 => [internal] load build definition from Dockerfile                                                                           0.0s
 => => transferring dockerfile: 213B                                                                                           0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                             0.4s
 => [internal] load .dockerignore                                                                                              0.0s
 => => transferring context: 2B                                                                                                0.0s
 => [internal] load build context                                                                                              0.0s
 => => transferring context: 93B                                                                                               0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fce8e072802d4a23c95594d16d84       0.0s
 => CACHED [2/5] WORKDIR /app                                                                                                  0.0s
 => CACHED [3/5] COPY requirements.txt requirements.txt                                                                        0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                            0.0s
 => CACHED [5/5] COPY . .                                                                                                      0.0s
 => exporting to image                                                                                                         0.0s
 => => exporting layers                                                                                                        0.0s
 => => writing image sha256:a2a34e11a20bac2d115f8ce36434abb77f5134f1769841d5010342621001780c                                   0.0s
 => => naming to docker.io/library/microservices_order-service                                                                 0.0s
```


Activities
Terminal
Jan 29 17:49
neo@kubernetes-machine: ~/project-root/microservices/product-service
neo@kubernetes-machine: ~/project-root/microservices/product-service
neo@kubernetes-machine: ~/project-root

```
=> => exporting layers                                                                                                  0.0s
=> => writing image sha256:610d53a4d945144896a304b5cb9be8a0baabf7baac016366efead4dcd4943da5                             0.0s
=> => naming to docker.io/library/microservices_cart-service                                                            0.0s
Building order-service
[+] Building 0.6s (10/10) FINISHED                                                                            docker:default
=> [internal] load build definition from Dockerfile                                                                     0.0s
=> => transferring dockerfile: 213B                                                                                     0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim                                                       0.4s
=> [internal] load .dockerignore                                                                                        0.0s
=> => transferring context: 2B                                                                                          0.0s
=> [internal] load build context                                                                                        0.0s
=> => transferring context: 93B                                                                                         0.0s
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fce8e072802d4a23c95594d16d84 0.0s
=> CACHED [2/5] WORKDIR /app                                                                                            0.0s
=> CACHED [3/5] COPY requirements.txt requirements.txt                                                                  0.0s
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                      0.0s
=> CACHED [5/5] COPY . .                                                                                                0.0s
=> exporting to image                                                                                                   0.0s
=> => exporting layers                                                                                                  0.0s
=> => writing image sha256:a2a34e11a20bac2d115f8ce36434abb77f5134f1769841d5010342621001780c                             0.0s
=> => naming to docker.io/library/microservices_order-service                                                           0.0s
Creating user-service-container    ... done
Creating cart-service-container    ... done
Creating product-service-container ... done
Creating order-service-container   ... done
neo@kubernetes-machine:~/project-root/microservices/product-service$ docker ps
CONTAINER ID   IMAGE                             COMMAND           CREATED         STATUS         PORTS
                    NAMES
aa0e6f1d2392   microservices_order-service     "python app.py"   5 seconds ago   Up 4 seconds   5004/tcp, 0.0.0.0:5004->5000/tcp, [:
:]:5004->5000/tcp   order-service-container
8e5d6d8e5668   microservices_product-service   "python app.py"   5 seconds ago   Up 4 seconds   5002/tcp, 0.0.0.0:5002->5000/tcp, [:
:]:5002->5000/tcp   product-service-container
63bd69b59021   microservices_cart-service      "python app.py"   5 seconds ago   Up 4 seconds   5003/tcp, 0.0.0.0:5003->5000/tcp, [:
:]:5003->5000/tcp   cart-service-container
d6b40e474704   microservices_user-service      "python app.py"   5 seconds ago   Up 4 seconds   5001/tcp, 0.0.0.0:5001->5000/tcp, [:
:]:5001->5000/tcp   user-service-container
neo@kubernetes-machine:~/project-root/microservices/product-service$ 
```

4. API Endpoints and Test Flows

Now that our microservices are running, we need to document their API endpoints for testing.

User Service API

Method	EndPoint	Description
POST	/users/login	Authenticate user
GET	/users/{id}	Feth user details

Product Service API

Method	EndPoint	Description
GET	/products	List all products
GET	/products/{id}	Feth product details

Cart Service API

Method	EndPoint	Description
POST	/cart	Add item to cart
DELETE	/cart/{itemId}	Remove item from cart

Order Service API

Method	EndPoint	Description
POST	/order	Place an order
GET	/order/{id}	Fetch order details

---

4. Test API Endpoints

Now, test your microservices using **cURL** or **Postman**.

User Service (Port 5000)

✅ **Login**

```
curl -X POST http://localhost:5001/users/login -H "Content-Type: application/json" -d '{"username":"testuser", "password":"testpassword"}'
```

✅ **Get User Details**

```
curl -X GET http://localhost:5001/users/1
```

Product Service (Port 5001)

✅ **Get All Products**

```
curl -X GET http://localhost:5002/products
```

✅ **Get Product by ID**

```
curl -X GET http://localhost:5002/products/1
```

Cart Service (Port 5002)

✅ **Add to Cart**

```
curl -X POST http://localhost:5003/cart -H "Content-Type: application/json" -d '{"user_id": 1, "product_id": 2, "quantity": 1}'
```

✅ **Remove from Cart**

```
curl -X DELETE http://localhost:5003/cart/1
```

Order Service (Port 5003)

✅ **Place Order**

```
curl -X POST http://localhost:5004/order -H "Content-Type: application/json"
-d '{"user_id": 1, "cart_id": 1}'
```

✅ **Get Order Details**

```
curl -X GET http://localhost:5004/order/1
```

Outcome

✅ We have now successfully:

1. **Containerized and run all microservices together using Docker Compose**.
2. **Set up API endpoints for communication**.
3. **Tested API flows using `curl`**.

Phase 4: Deploying microservices to Kubernetes:

1. **Create Kubernetes YAML Files**
   - Define `Deployment` and `Service` YAML files for each microservice.
   - Ensure the `Service` type is `ClusterIP` for internal communication.
2. **Apply Kubernetes Manifests**
   - Use `kubectl apply -f` to deploy your services.
3. **Expose Services Locally**
   - Use `kubectl port-forward` or Minikube service to access the services.
4. **Verify Deployment and Logs**
   - Check running pods: `kubectl get pods`
   - Check logs: `kubectl logs <pod-name>`

Here is the deployment and service YAML configuration code for all 4 microservices (user, product, cart, order).

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: user-service
  template:
    metadata:
      labels:
```

```yaml
        app: user-service
    spec:
      containers:
        - name: user-service
          image: user-service:latest
          ports:
            - containerPort: 5000
---
apiVersion: v1
kind: Service
metadata:
  name: user-service
spec:
  selector:
    app: user-service
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product-service
  template:
    metadata:
      labels:
        app: product-service
    spec:
      containers:
        - name: product-service
          image: product-service:latest
          ports:
            - containerPort: 5001
---
apiVersion: v1
kind: Service
metadata:
  name: product-service
spec:
  selector:
    app: product-service
  ports:
    - protocol: TCP
      port: 5001
      targetPort: 5001
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cart-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cart-service
  template:
    metadata:
      labels:
        app: cart-service
    spec:
      containers:
        - name: cart-service
          image: cart-service:latest
          ports:
            - containerPort: 5002
---
apiVersion: v1
kind: Service
```

```
metadata:
  name: cart-service
spec:
  selector:
    app: cart-service
  ports:
    - protocol: TCP
      port: 5002
      targetPort: 5002
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order-service
  template:
    metadata:
      labels:
        app: order-service
    spec:
      containers:
        - name: order-service
          image: order-service:latest
          ports:
            - containerPort: 5003
---
apiVersion: v1
kind: Service
metadata:
  name: order-service
spec:
  selector:
    app: order-service
```

```
 ports:
  - protocol: TCP
    port: 5003
    targetPort: 5003
```

Apply the YAML files:

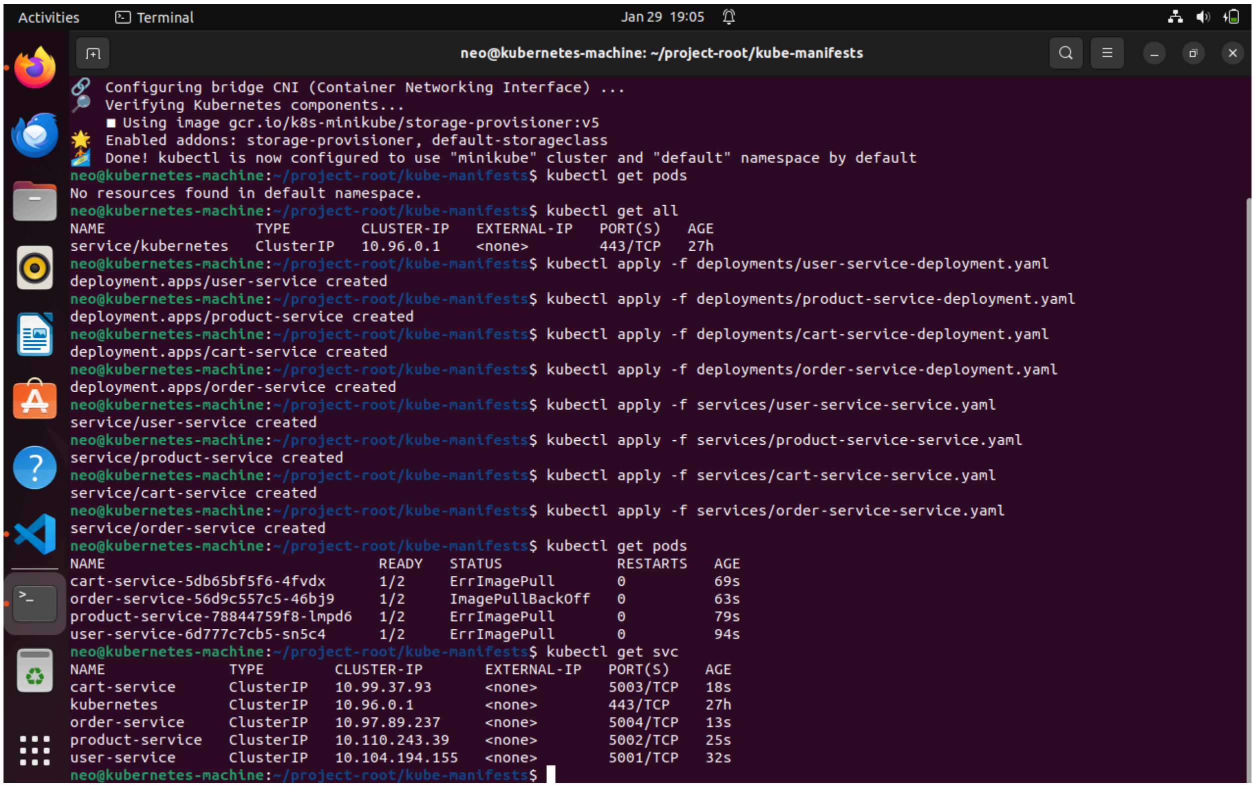
```
kubectl apply -f k8s_microservices.yaml
```

Verify deployments and services:

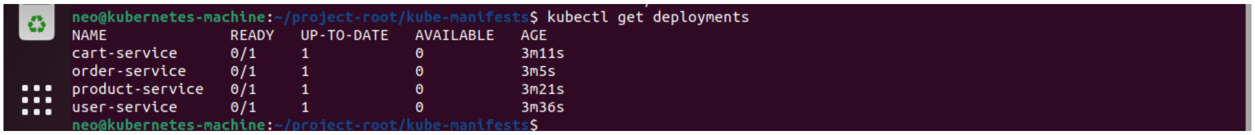
```
kubectl get pods

kubectl get services
```

Check logs:

```
kubectl logs -f <pod-name>
```



```
🔗 Configuring bridge CNI (Container Networking Interface) ...
🔎 Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Enabled addons: storage-provisioner, default-storageclass
🏄 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl get pods
No resources found in default namespace.
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   27h
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl apply -f deployments/user-service-deployment.yaml
deployment.apps/user-service created
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl apply -f deployments/product-service-deployment.yaml
deployment.apps/product-service created
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl apply -f deployments/cart-service-deployment.yaml
deployment.apps/cart-service created
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl apply -f deployments/order-service-deployment.yaml
deployment.apps/order-service created
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl apply -f services/user-service-service.yaml
service/user-service created
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl apply -f services/product-service-service.yaml
service/product-service created
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl apply -f services/cart-service-service.yaml
service/cart-service created
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl apply -f services/order-service-service.yaml
service/order-service created
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl get pods
NAME                               READY   STATUS             RESTARTS   AGE
cart-service-5db65bf5f6-4fvdx      1/2     ErrImagePull       0          69s
order-service-56d9c557c5-46bj9     1/2     ImagePullBackOff   0          63s
product-service-78844759f8-lmpd6   1/2     ErrImagePull       0          79s
user-service-6d777c7cb5-sn5c4      1/2     ErrImagePull       0          94s
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl get svc
NAME              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
cart-service      ClusterIP   10.99.37.93      <none>        5003/TCP   18s
kubernetes        ClusterIP   10.96.0.1        <none>        443/TCP    27h
order-service     ClusterIP   10.97.89.237     <none>        5004/TCP   13s
product-service   ClusterIP   10.110.243.39    <none>        5002/TCP   25s
user-service      ClusterIP   10.104.194.155   <none>        5001/TCP   32s
neo@kubernetes-machine:~/project-root/kube-manifests$
```



```
neo@kubernetes-machine:~/project-root/kube-manifests$ kubectl get deployments
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
cart-service      0/1     1            0           3m11s
order-service     0/1     1            0           3m5s
product-service   0/1     1            0           3m21s
user-service      0/1     1            0           3m36s
neo@kubernetes-machine:~/project-root/kube-manifests$
```

Phase 5: Helm for Managing Kubernetes Resources

1. Set Up Helm for Microservices Deployment

Creating a **Helm chart** for each microservice to easily manage their Kubernetes resources. This includes Deployments, Services, etc.

Helm Chart Structure:

```
microservices/

  ├── user-service/

  │   ├── Chart.yaml

  │   ├── values.yaml

  │   ├── templates/

  │   │   ├── deployment.yaml

  │   │   ├── service.yaml
```

Create a Helm Chart for `user-service`:

```
helm create user-service

helm install user-service ./user-service
```

---

2. Prometheus and Grafana for Monitoring (optional as this a demo environment)

Set Up Prometheus:

Prometheus will scrape metrics from our microservices to collect monitoring data.

Install Prometheus using Helm:

```
helm install prometheus prometheus-community/kube-prometheus-stack
```

Configure Prometheus to scrape metrics from our microservices.

Set Up Grafana:

Grafana is used to visualize Prometheus metrics.

Access Grafana: After installing Grafana (using the same Helm chart as Prometheus), get the Grafana dashboard URL:

```
kubectl get svc -n monitoring
```

1. **Create Dashboards**:
    - You can import existing dashboards from Grafana’s dashboard library or create custom ones based on the metrics from your services.
2. **Configure Grafana to Use Prometheus** as a data source:
    - Go to Grafana UI and add Prometheus as a data source.
    - Build dashboards to monitor key metrics such as:
        - CPU/Memory usage of your microservices.
        - Request/Response times for each service.
        - Latency and throughput.

---

3. Jenkins for CI/CD (optional)

Set Up Jenkins in Kubernetes:

Install Jenkins using Helm:

```
helm install jenkins jenkins/jenkins
```

Set Up Jenkins Pipelines:

Creating Jenkinsfiles for each microservice to build Docker images, pushing them to a container registry, and deploy to Kubernetes.

Example Jenkinsfile:

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        script {
          docker.build("user-service")
        }
      }
    }
    stage('Push to Docker Hub') {
      steps {
        script {
          docker.withRegistry('https://registry.hub.docker.com',
'docker-hub-credentials') {
            docker.image('user-service').push()
          }
        }
      }
    }
    stage('Deploy to Kubernetes') {
      steps {
        script {
          kubernetesDeploy(
            configs: "k8s/user-service.yaml",
            kubeconfigId: "kube-config"
          )
        }
      }
    }
  }
}
```

○

Automate Deployment:

- Trigger Jenkins builds on GitHub push events using **GitHub Webhooks** to automatically deploy new changes to our microservices in Kubernetes.

The below Jenkinsfiles are designed to build Docker images for each microservice, push the images to a container registry, and optionally deploy them to our Kubernetes cluster.

The structure uses**Jenkins pipeline syntax** with **Declarative Pipeline**.

1. Jenkinsfile for `user-service`

```
pipeline {
    agent any

    environment {
        REGISTRY = "docker.io"
        IMAGE_NAME = "user-service"
        IMAGE_TAG = "latest"
        REGISTRY_CREDENTIALS = 'docker-credentials'
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    docker.build("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}", "-f microservices/user-service/Dockerfile .")
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    docker.withRegistry('', REGISTRY_CREDENTIALS) {

docker.image("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}").push()
                    }
                }
            }
        }

        stage('Deploy to Kubernetes') {
            steps {
                script {
                    // Apply Kubernetes deployment using kubectl
                    sh """
                    kubectl apply -f
kubernetes-manifests/deployments/user-service-deployment.yaml
                    kubectl rollout restart deployment user-service
                    """
                }
            }
        }
```

```
    }

    post {
        success {
            echo "Build and Deployment Successful!"
        }
        failure {
            echo "Build or Deployment Failed!"
        }
    }
}
```

---

2. Jenkinsfile for `product-service`

```
pipeline {
    agent any

    environment {
        REGISTRY = "docker.io"
        IMAGE_NAME = "product-service"
        IMAGE_TAG = "latest"
        REGISTRY_CREDENTIALS = 'docker-credentials'
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    docker.build("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}", "-f
microservices/product-service/Dockerfile .")
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    docker.withRegistry('', REGISTRY_CREDENTIALS) {

docker.image("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}").push()
                    }
                }
            }
        }

        stage('Deploy to Kubernetes') {
            steps {
                {
                script {
                    sh """
                    kubectl apply -f kubernetes-manifests/deployments/product-service-deployment.yaml
                    kubectl rollout restart deployment product-service
                    """
                }
            }
        }
    }

    post {
        success {
            echo "Build and Deployment Successful!"
        }
        failure {
            echo "Build or Deployment Failed!"
        }
    }
}
```

3. Jenkinsfile for `cart-service`

```
pipeline {
    agent any

    environment {
        REGISTRY = "docker.io"
        IMAGE_NAME = "cart-service"
        IMAGE_TAG = "latest"
        REGISTRY_CREDENTIALS = 'docker-credentials'
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    docker.build("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}", "-f microservices/cart-service/Dockerfile .")
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    docker.withRegistry('', REGISTRY_CREDENTIALS) {
```

```
docker.image("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}").push()
                    }
                }
            }
        }

        stage('Deploy to Kubernetes') {
            steps {
                script {
                    sh """
                    kubectl apply -f
kubernetes-manifests/deployments/cart-service-deployment.yaml
                    kubectl rollout restart deployment cart-service
                    """
                }
            }
        }
    }

    post {
        success {
            echo "Build and Deployment Successful!"
        }
        failure {
            echo "Build or Deployment Failed!"
        }
    }
}
```

4. Jenkinsfile for `order-service`

```
pipeline {
    agent any

    environment {
        REGISTRY = "docker.io"
        IMAGE_NAME = "order-service"
        IMAGE_TAG = "latest"
        REGISTRY_CREDENTIALS = 'docker-credentials'
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    docker.build("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}", "-f microservices/order-service/Dockerfile .")
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    docker.withRegistry('', REGISTRY_CREDENTIALS) {
                        docker.image("${REGISTRY}/${IMAGE_NAME}:${IMAGE_TAG}").push()
                    }
                }
            }
        }

        stage('Deploy to Kubernetes') {
            steps {
                script {
                    sh """
                    kubectl apply -f kubernetes-manifests/deployments/order-service-deployment.yaml
                    kubectl rollout restart deployment order-service
                    """
                }
            }
        }
    }

    post {
        success {
            echo "Build and Deployment Successful!"
        }
        failure {
            echo "Build or Deployment Failed!"
        }
    }
}
```

Key Points in Each Jenkinsfile:

- **Checkout**: This pulls the code from your Git repository.
- **Build Docker Image**: It uses `docker.build()` to build the Docker image from the Dockerfile of each service.
- **Push Docker Image**: The image is pushed to a container registry like DockerHub or AWS ECR. You'll need to set up Jenkins credentials for the registry (`docker-credentials`).
- **Deploy to Kubernetes**: After pushing the Docker image, the service is deployed to Kubernetes using `kubectl` commands, applying the relevant Kubernetes YAML files and restarting the deployment.

What You Need:

1. Ensure you have a **container registry** configured, and the Jenkins credentials (`docker-credentials`) are set up to authenticate with it.
2. Your Kubernetes manifests (`deployment.yaml`, `service.yaml`, etc.) should be correctly defined in your project to deploy each service.
3. If you're using AWS ECR or another private registry, adjust the `REGISTRY` variable accordingly.

This structure can be replicated for each microservice. You just need to replace `user-service`, `product-service`, etc., in the relevant sections.

Thank you for reading this document

Aman Kr Choudhary