DOCKER GUIDE

To **Dockerize** your e-commerce microservices system based on the component diagram, follow this step-by-step guide:

Step 1: Structure Your Project

Each microservice (customer, cart, item, payment, order, shipment, comment) will be in a separate directory with its own Dockerfile.

pgsql

CopyEdit

ecommerce-microservices/

|— customer/

| ├— app.py # Example Flask API

| ├— Dockerfile

| ├— requirements.txt

|— cart/

| ├— app.py

| ├— Dockerfile

| ├— requirements.txt

|— item/

| ├— app.py

| ├— Dockerfile

| ├— requirements.txt

|— order/

|— payment/

|— shipment/

|— comment/

|— docker-compose.yml

|— nginx/

| ├— nginx.conf

Step 2: Create a Dockerfile for Each Service

Each microservice will use Python and Flask for simplicity.

Example: customer/Dockerfile

dockerfile

CopyEdit

```
# Use Python as base image
FROM python:3.9

# Set working directory
WORKDIR /app

# Copy source files
COPY . .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Expose port
EXPOSE 5000

# Run application
CMD ["python", "app.py"]
```

Step 3: Create a Flask API for Each Microservice

Each service (customer, cart, item, etc.) will have a simple Flask-based API.

Example: customer/app.py

python

CopyEdit

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/api/customers', methods=['GET'])
def get_customers():
    return jsonify({"message": "Customer service running!"})

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

Each service (cart, item, etc.) will have a similar app.py with relevant API endpoints.

Step 4: Create docker-compose.yml

To manage multiple services with Docker Compose.

yaml

CopyEdit

```
version: '3.8'

services:
  customer:
```

```
  build: ./customer
  container_name: customer_service
  ports:
   - "5001:5000"
  depends_on:
   - db
  networks:
   - ecommerce_network

cart:
  build: ./cart
  container_name: cart_service
  ports:
   - "5002:5000"
  depends_on:
   - db
  networks:
   - ecommerce_network

item:
  build: ./item
  container_name: item_service
  ports:
   - "5003:5000"
  depends_on:
   - db
```

```
  networks:
    - ecommerce_network

order:
  build: ./order
  container_name: order_service
  ports:
    - "5004:5000"
  depends_on:
    - db
  networks:
    - ecommerce_network

payment:
  build: ./payment
  container_name: payment_service
  ports:
    - "5005:5000"
  networks:
    - ecommerce_network

shipment:
  build: ./shipment
  container_name: shipment_service
  ports:
    - "5006:5000"
```

```
  networks:
    - ecommerce_network

comment:
  build: ./comment
  container_name: comment_service
  ports:
    - "5007:5000"
  networks:
    - ecommerce_network

db:
  image: postgres
  container_name: postgres_db
  environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: password
    POSTGRES_DB: ecommerce
  ports:
    - "5432:5432"
  networks:
    - ecommerce_network

nginx:
  image: nginx:latest
  container_name: api_gateway
```

```
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf
  ports:
    - "80:80"
  depends_on:
    - customer
    - cart
    - item
  networks:
    - ecommerce_network

networks:
 ecommerce_network:
  driver: bridge
```

Step 5: Configure NGINX as API Gateway

File: nginx/nginx.conf

```
nginx
CopyEdit
server {
  listen 80;

  location /api/customers/ {
    proxy_pass http://customer:5000/;
  }
```

```
    location /api/carts/ {
        proxy_pass http://cart:5000/;
    }

    location /api/items/ {
        proxy_pass http://item:5000/;
    }

    location /api/orders/ {
        proxy_pass http://order:5000/;
    }

    location /api/payments/ {
        proxy_pass http://payment:5000/;
    }

    location /api/shipments/ {
        proxy_pass http://shipment:5000/;
    }

    location /api/comments/ {
        proxy_pass http://comment:5000/;
    }
}
```

Step 6: Build and Run with Docker Compose

Navigate to the project directory and run:

sh

CopyEdit

docker-compose up --build

This will: ✅ Build each microservice
✅ Start them in separate containers
✅ Set up PostgreSQL as the database
✅ Expose APIs through NGINX

Step 7: Test the System

Check Running Containers

sh

CopyEdit

docker ps

Test API Endpoints

sh

CopyEdit

curl http://localhost/api/customers/

curl http://localhost/api/items/

curl http://localhost/api/orders/

Summary

✓ Each microservice runs in its own container
✓ Docker Compose manages all services
✓ PostgreSQL is used as the database
✓ NGINX routes requests to correct services