To create a scalable microservices architecture for an **E-commerce Women's Dress Order Processing** application, here’s a structured outline you can follow:

**PART 1: DESIGN**

1. **Application Overview**  
   This application supports the purchase of women's dresses, managing customer orders, inventory, and payments. Users can browse products, add items to their cart, place orders, and track their order status.
2. **Scalable Architecture**  
   **Microservices to include**:
   * **User Service**: Manages user registration, authentication, and profile.
   * **Product Service**: Manages dress inventory, including details, categories, and availability.
   * **Order Service**: Handles order creation, status updates, and order history.
   * **Inventory Service**: Manages stock levels for each dress and updates inventory on order placement.
   * **Payment Service**: Processes payments for orders.
   * **Notification Service**: Sends email or SMS notifications about order status.

**Architecture Diagram**: Illustrate a diagram showing these services, including their interactions, data flow, and dependencies.

1. **System Operations**
   * **User Service**:
     + Register user, authenticate, update profile, view account.
   * **Product Service**:
     + List products, search products, get product details.
   * **Order Service**:
     + Create order, update order status, view order history.
   * **Inventory Service**:
     + Check stock, reserve stock, update stock after purchase.
   * **Payment Service**:
     + Process payment, validate payment, refund payment.
   * **Notification Service**:
     + Send order confirmation, dispatch notification, delivery confirmation.
2. **Assign Operations and Collaboration**  
   Example: The **Order Service** interacts with **Inventory** to check and reserve stock, calls **Payment** for payment processing, and then triggers **Notification** to send a confirmation to the user. All communication is through API calls.

**PART 2: IMPLEMENTATION**

1. **Implement Two Microservices**  
   Select any two services to implement, e.g., **Order Service** and **Inventory Service**. Ensure they are separate repositories and independent deployments.
2. **Technology Stack**
   * **Languages**: Python (FastAPI for REST APIs), or Node.js (Express).
   * **Database**: PostgreSQL for relational data, MongoDB for unstructured product data.
   * **Communication**: REST APIs for inter-service communication or RabbitMQ for event-driven messaging.
3. **Implementation Steps**
   * Define APIs for each service (e.g., /create-order in Order Service).
   * Establish database schema (e.g., orders table in Order Service, inventory table in Inventory Service).
   * Write logic for service operations, including request validation, data processing, and response formatting.
   * Implement error handling, logging, and monitoring features.

**PART 3: DEPLOYMENT**

1. **Dockerize Each Service**
   * Write Dockerfiles for each service, ensuring independent containerization. Example:

dockerfile

Copy code

FROM python:3.9

COPY . /app

WORKDIR /app

RUN pip install -r requirements.txt

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

1. **Kubernetes Deployment**
   * Define YAML files for **Deployment** and **Service** objects for each microservice.
   * Example YAML for Order Service:

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: order-service

spec:

replicas: 2

selector:

matchLabels:

app: order

template:

metadata:

labels:

app: order

spec:

containers:

- name: order

image: order-service:latest

ports:

- containerPort: 8000

* + Run Minikube locally, then deploy and monitor services with kubectl commands.

1. **Kubernetes Dashboard**
   * Access the Kubernetes Dashboard to view and analyze the deployments, resource usage, and ensure all services are up and running.

**DOCUMENTATION AND SUBMISSION**

1. **Documentation**
   * Include application overview, architecture diagram, system operations, project structure, deployment steps, and GitHub repository links.
   * Add snapshots of:
     + Service API calls and responses from Postman
     + Kubernetes dashboard with service deployments and pods
     + Console outputs from Minikube cluster
2. **Demonstration Video**
   * Record the deployment process, service testing, and Kubernetes monitoring.

I'll help you structure and provide code samples for key sections of your **E-commerce Women's Dress Order Processing** application, including Docker and Kubernetes configurations. Let's break down the code examples for two key microservices: **Order Service** and **Inventory Service**. I'll demonstrate using Python (with FastAPI) as the base technology.

**Project Structure**

plaintext

Copy code

ecommerce-app/

├── order-service/

│ ├── main.py

│ ├── requirements.txt

│ └── Dockerfile

├── inventory-service/

│ ├── main.py

│ ├── requirements.txt

│ └── Dockerfile

└── k8s/

├── order-service-deployment.yaml

└── inventory-service-deployment.yaml

**Order Service**

**order-service/main.py**

python

Copy code

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

import uuid

app = FastAPI()

# In-memory "database" to store orders

orders\_db = {}

class Order(BaseModel):

product\_id: str

quantity: int

user\_id: str

@app.post("/create-order")

def create\_order(order: Order):

order\_id = str(uuid.uuid4())

orders\_db[order\_id] = {

"order\_id": order\_id,

"product\_id": order.product\_id,

"quantity": order.quantity,

"user\_id": order.user\_id,

"status": "pending"

}

# Normally, you'd call Inventory Service to reserve stock here

return {"message": "Order created", "order\_id": order\_id}

@app.get("/order/{order\_id}")

def get\_order(order\_id: str):

order = orders\_db.get(order\_id)

if not order:

raise HTTPException(status\_code=404, detail="Order not found")

return order

**order-service/requirements.txt**

Copy code

fastapi

uvicorn

**order-service/Dockerfile**

dockerfile

Copy code

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY main.py .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

**Inventory Service**

**inventory-service/main.py**

python

Copy code

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

app = FastAPI()

# In-memory "database" for inventory stock levels

inventory\_db = {

"dress\_1": 50,

"dress\_2": 20,

"dress\_3": 30

}

class ReserveStock(BaseModel):

product\_id: str

quantity: int

@app.post("/reserve-stock")

def reserve\_stock(reservation: ReserveStock):

product\_id = reservation.product\_id

if product\_id not in inventory\_db:

raise HTTPException(status\_code=404, detail="Product not found")

if inventory\_db[product\_id] < reservation.quantity:

raise HTTPException(status\_code=400, detail="Not enough stock")

inventory\_db[product\_id] -= reservation.quantity

return {"message": "Stock reserved", "product\_id": product\_id, "remaining\_stock": inventory\_db[product\_id]}

@app.get("/stock/{product\_id}")

def check\_stock(product\_id: str):

if product\_id not in inventory\_db:

raise HTTPException(status\_code=404, detail="Product not found")

return {"product\_id": product\_id, "stock": inventory\_db[product\_id]}

**inventory-service/requirements.txt**

Copy code

fastapi

uvicorn

**inventory-service/Dockerfile**

dockerfile

Copy code

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY main.py .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8001"]

**Docker Commands**

To build and run each service locally:

1. **Build Images**:

bash

Copy code

docker build -t order-service ./order-service

docker build -t inventory-service ./inventory-service

1. **Run Containers**:

bash

Copy code

docker run -d -p 8000:8000 order-service

docker run -d -p 8001:8001 inventory-service

**Kubernetes Configuration (Minikube)**

**k8s/order-service-deployment.yaml**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: order-service

spec:

replicas: 2

selector:

matchLabels:

app: order-service

template:

metadata:

labels:

app: order-service

spec:

containers:

- name: order-service

image: order-service:latest

ports:

- containerPort: 8000

---

apiVersion: v1

kind: Service

metadata:

name: order-service

spec:

selector:

app: order-service

ports:

- protocol: "TCP"

port: 80

targetPort: 8000

**k8s/inventory-service-deployment.yaml**

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: inventory-service

spec:

replicas: 2

selector:

matchLabels:

app: inventory-service

template:

metadata:

labels:

app: inventory-service

spec:

containers:

- name: inventory-service

image: inventory-service:latest

ports:

- containerPort: 8001

---

apiVersion: v1

kind: Service

metadata:

name: inventory-service

spec:

selector:

app: inventory-service

ports:

- protocol: "TCP"

port: 80

targetPort: 8001

**Deploy on Minikube**

1. **Start Minikube**:

bash

Copy code

minikube start

1. **Deploy Services**:

bash

Copy code

kubectl apply -f k8s/order-service-deployment.yaml

kubectl apply -f k8s/inventory-service-deployment.yaml

1. **Access Kubernetes Dashboard**:

bash

Copy code

minikube dashboard

**Testing with Postman**

Use Postman to interact with your services:

* **Order Service**:
  + POST http://<minikube-ip>:<node-port>/create-order
  + GET http://<minikube-ip>:<node-port>/order/{order\_id}
* **Inventory Service**:
  + POST http://<minikube-ip>:<node-port>/reserve-stock
  + GET http://<minikube-ip>:<node-port>/stock/{product\_id}