Online shoping platform

Team:17

[redacted]	محمد ابراهيم محمد محمد ابراهيم
[redacted]	يوسف حسن عاشور محمد
[redacted]	محمد محسن السيد عبدالرحمن محمد
[redacted]	محمد محمود ابراهيم محمد
[redacted]	محمد صالح امين فودة
[redacted]	احمد شعبان عبده حسن

1. Introduction

The E-Commerce Platform is a modern solution for managing user accounts, product catalogs, and shopping carts. This comprehensive documentation aims to provide a clear understanding of the project's structure, components, and deployment processes. Additionally, it will guide users on setting up development environments and deploying the platform on Kubernetes clusters.

2. Project Structure

The project consists of three main services:

A-User Management

- Responsible for managing user account.
- Includes **user_management.py** for user management logic.
- Dockerized with a **Dockerfile**.
- Containerized using **docker compose.**
- Kubernetes manifests (**deployment.yaml** and **service.yaml**) are located in the **kubernetes/user-management** directory.

B-Product Catalog

- Manages the catalog of products available for purchase.
- Contains **product_catalog.py** for product catalog logic.
- Dockerized with a **Dockerfile**.
- Containerized using **docker compose.**
- Kubernetes manifests (**deployment.yaml** and **service.yaml**) are located in the **kubernetes/product-catalog** directory.

C-Cart Management

- Handles user shopping carts and orders.
- Utilizes **cart_management.py** for cart management functionality.
- Dockerized with a **Dockerfile**.
- Containerized using **docker compose**.
- Kubernetes manifests (**deployment.yaml** and **service.yaml**) are located in the **kubernetes/cart-management** directory.

3. Development Environment Setup

To set up a development environment, we followed these steps:

1. Install Docker, Docker compose and Kubernetes on local machine.
2. Navigate to each component directory (**user_management**, **product_catalog**, **cart_management**) and build the Docker images using the provided **Dockerfile**.
3. Build containers for 3 services using same docker compose file and make 3 containers interact with each other on the same network
4. Ensure that Kubernetes is configured and running locally .
5. Apply the Kubernetes manifests located in the **kubernetes** directory for each component using **kubectl apply -f <manifest.yaml>**.

4. components

A-Backend Services:

1-user_management: Handles user-related functionalities.

2-product_catalog: Manages product-related operations.

3-cart_management: Responsible for managing the shopping cart.

B-Dockerfiles:

Each service directory contains a Dockerfile, which specifies the instructions for building Docker images for each service. Dockerfiles are used for containerizing the backend services.

C-Requirements Files:

Each service directory also contains a requirements.txt file, which lists the Python dependencies required for running the respective backend service.

D- **Docker compose file:**

Docker Compose used for defining and running multi-container Docker applications, specifying the services, networks, and volumes needed for the application.

E-Kubernetes Deployment Configurations:

Under the kubernetes directory, there are subdirectories for each service (user-management, product-catalog, cart-management), each containing:

1-deployment.yaml: Specifies the deployment configuration for the corresponding service, including the number of replicas, container images, and other settings.

2-service.yaml: Defines the Kubernetes service for the respective backend service, including details like port mapping and service type.

5. Deployment Process

Deploying the platform on Kubernetes involves the following steps:

1. Create a Kubernetes cluster.
2. Apply the Kubernetes manifests for each component (**deployment.yaml** and **service.yaml**) using **kubectl apply**.
3. Monitor the deployment status using **kubectl get pods** and **kubectl get services**.

6. Usage Instructions

After deployment, users can interact with the platform as follows:

1. Access the User Management service via **<user-management-service-ip>:3030**.
2. Access the Product Catalog service via **<product-catalog-service-ip>:3031**.
3. Access the Cart Management service via **<cart-management-service-ip>:3032**.
4. Utilize the provided APIs or user interfaces to manage users, products, and shopping carts.