1. **Setting Up the Microservices:** creating the Product Service, Cart Service and Notification Service as separate microservices with basic endpoints.

***Product Management Service Endpoints:***

* POST /api/Products: Add a product.
* GET /api/Products: Get all products.
* GET /api/Products/{id}: Get a single product by ID.
* PUT / api/Products: Update a product’s details.
* DELETE / api/Products{id}: Delete a product.

***Notification Service Endpoints:***

* POST /api/Notification?message : Send Notification.

***Cart Service Endpoints:***

* POST /api/Cart: Add a item to cart.
* GET /api/Cart: Get cart items.
* PUT / api/Cart/Checkout: Used for checkout and send message to kafka.

**2. Implement API Gateway**

a. Create Project name: OcelotAPIGateway

b. Install Ocelot

c. Configure Ocelot in Program.cs

builder.Configuration.AddJsonFile("ocelot.json", optional: false, reloadOnChange: true); builder.Services.AddOcelot();

app.UseOcelot().Wait();

d. add a new file called ocelot.json.

**3. Implementation of Consul Service Discovery**

a. Install Consul l in a Docker container. To install via Docker, use the following command:

docker run -p 8500:8500 -p 8600:8600/udp --name=consul consul:v0.6.4 agent -server -bootstrap -ui -client=0.0.0.0

This runs Consul on localhost:8500.

b. Install Required NuGet Packages in Customer Management Service and Account Management Service projects

dotnet add package Consul.AspNetCore

c. Configure Consul in appsettings.json

Add Consul configuration in the appsettings.json of each service.

d. Implement Consul Service Registration Middleware

e. Register Consul in Program.cs

**4. implement Centralized configuration management in consul**

a. Add Configuration Data to Consul:

Go to http://localhost:8500 and open the Key/Value store.

Add a new key for each configuration setting you want to use. For instance:

Key: CustomerService:Database:ConnectionString

Value: Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;

b. Add Consul NuGet Packages to ASP.NET Core Services

In each microservice project, add the following NuGet packages:

consul.Extensions.Configuration

Microsoft.Extensions.Configuration

c. Configure Each Microservice to Use Consul

d. Configure Each Microservice to Use Consul-Loaded Configuration

**5. implement cross cutting concerns Exception handling**

a. Create a Custom Exception Middleware

b. Register the Middleware

**6. to package the ExceptionHandlingMiddleware middleware into a reusable library to share across microservices.**

a. create a project and create ExceptionHandlingMiddleware.cs

b. Add an Extension Method for Middleware Registration

To simplify adding the middleware to the Program.cs or Startup.cs files of each service, create an extension method to register ExceptionHandlingMiddleware.

c. Register the Middleware in each microservices

**High-Level Design**

**Architecture Diagram**

* **Components**:
  + Microservices:
    - Product Service
    - Cart Service
    - Notification Service
  + Cross-cutting Concerns:
    - Logging
    - Tracing
    - Authentication/Authorization
  + Communication:
    - Synchronous for essential interactions (e.g., API Gateway -> Services).
    - Asynchronous for notifications/events (e.g., using a message queue like Kafka).

**Key Design Patterns**

* + **Database per Service**: Each service uses its own database (e.g., NoSQL for products, relational DB for orders). Assumed in-memory storage for this implementation.
  + **Event-Driven Communication**: Notifications triggered by events.
  + **Centralized API Gateway**: Single point of entry, handling routing, and security.
  + **Service Registry and Discovery**: Implemented using a tool like **Consul**.

**Pipeline Diagram**

1. **Source Code**: Version control in **GitHub**.
2. **Build**: Build microservices using tools like **Maven** or **Gradle**.
3. **Test**:
   * Unit testing for individual services.
   * Integration tests using Postman.
4. **Dockerization**:
   * Build Docker images for all services.
5. **Deploy**:
   * Deploy containers to Kubernetes.
6. **Monitor**:
   * Use **Prometheus**/**Grafana** for monitoring.