UE20CS351
Cloud Computing Project

6: Building an E-commerce Microservices Application on Cloud using Docker, Kubernetes, Jenkins, and Git

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Policharla Sai Sailaja PES1UG20CS671

Overview

The aim of this project is to develop an ecommerce microservices application that can be deployed on the cloud using Docker, Kubernetes, Jenkins, and Git. The application will consist of several microservices that will be deployed as Docker containers on a Kubernetes cluster. Jenkins will be used for continuous integration and deployment, while Git will be used for version control.

Documentation:

1. Docker documentation: https://docs.docker.com/
2. Kubernetes documentation: https://kubernetes.io/docs/home/
3. Jenkins documentation: https://www.jenkins.io/doc/
4. Git documentation: https://git-scm.com/doc ; Git tutorial for beginners: https://www.atlassian.com/git/tutorials
5. Deploying microservices with Kubernetes: https://kubernetes.io/docs/tutorials/kubernetes- basics/deploy-app/deploy-intro/
6. Dockerize a Node.js WebApp: https://nodejs.org/en/docs/guides/nodejs-docker-webapp/
7. REST API Design: https://restfulapi.net/
8. Postman documentation for API testing etc: https://learning.postman.com/docs/

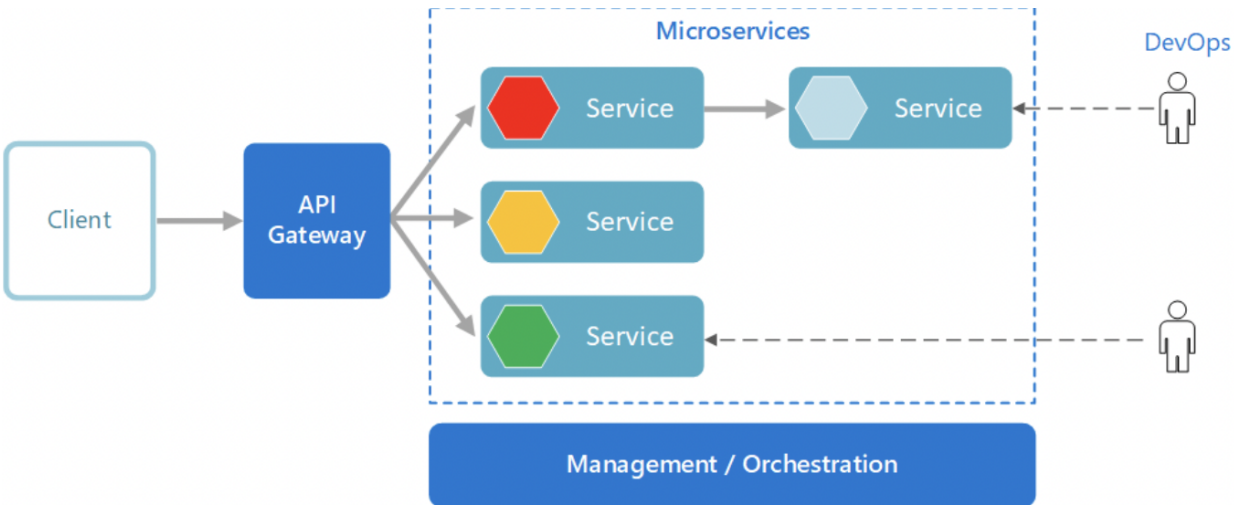
Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

Technologies Used:

- Java
- Spring Boot
- Maven
- Docker
- Kubernetes
- Jenkins
- Postman
- Postgres/MySQL
- MongoDB
- GitHub

Task 1: Design the Microservices Architecture

A microservices architecture consists of a collection of small, autonomous services. Each service is self-contained and should implement a single business capability within a bounded context. A bounded context is a natural division within a business and provides an explicit boundary within which a domain model exists.



- Microservices are small, independent, and loosely coupled. A single small team of developers can write and maintain a service.

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

- Each service is a separate codebase, which can be managed by a small development team.
- Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
- Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.
- Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
- Supports polyglot programming. For example, services don't need to share the same technology stack, libraries, or frameworks.

Besides for the services themselves, some other components appear in a typical microservices architecture:
Management/orchestration. This component is responsible for placing services on nodes, identifying failures, rebalancing services across nodes, and so forth. Typically this component is an off-the-shelf technology such as Kubernetes, rather than something custom built.
API Gateway. The API gateway is the entry point for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end. Advantages of using an API gateway include:

- It decouples clients from services. Services can be versioned or refactored without needing to update all of the clients.
- Services can use messaging protocols that are not web friendly, such as Kafka.
- The API Gateway can perform other cross-cutting functions such as authentication, logging, SSL termination, and load balancing.
- Out-of-the-box policies, like for throttling, caching, transformation, or validation.

Microservices-

- API Gateway (Keycloak)
- Order-service
- Inventory-service
- Product-service
- Notification-service (Kafka)
- Discovery Server (Eureka)

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

1. **API Gateway (Keycloak)**: The API Gateway is the entry point to the system and acts as a central hub for all incoming requests. It handles authentication and authorization, as well as routing requests to the appropriate microservices. Keycloak is an open-source identity and access management solution used for secure user authentication and authorization.
2. **Order-service**: The order-service handles all the orders placed by customers on the e-commerce platform. It provides features such as creating new orders and retrieving order information. Order service uses MongoDB, and every order contains items, which compose of a price, quantity and sku-code for unique identification.This information is stored in inventory-service. So order-service communicates with inventory-service.

   Example of an order item

```
{
        "orderLineItemsDtoList":[
                {
                        "skuCode":"iphone_13",
                        "price":1200,
                        "quantity":1
                }
        ]

}
```

3. **Inventory-service**: The inventory-service manages the stock levels of products available on the e-commerce platform. It is responsible for ensuring that there is enough inventory to fulfill orders, updating inventory levels when orders are placed or canceled, and providing information about available inventory to other services. It can only be accessed through a request call to order-service.
4. **Product-service**: The product-service manages the products available on the e-commerce platform. It provides features such as creating new products and retrieving product information. Every product is composed of an id, name, description and price. An example of this would be -

```
[
        {
                “id”:20937456349393,
                “name”: iPhone 13,
                “description”: iPhone 13,
                “price”:1000
        }
```

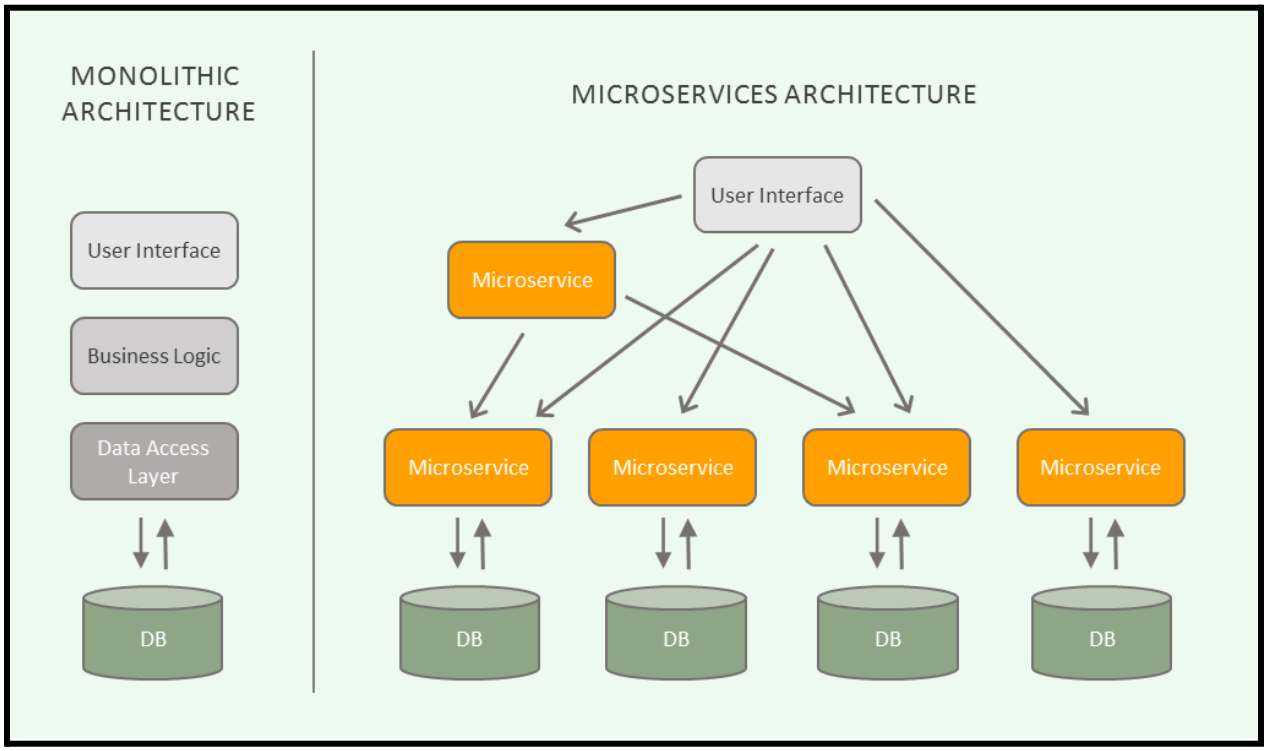
Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

]

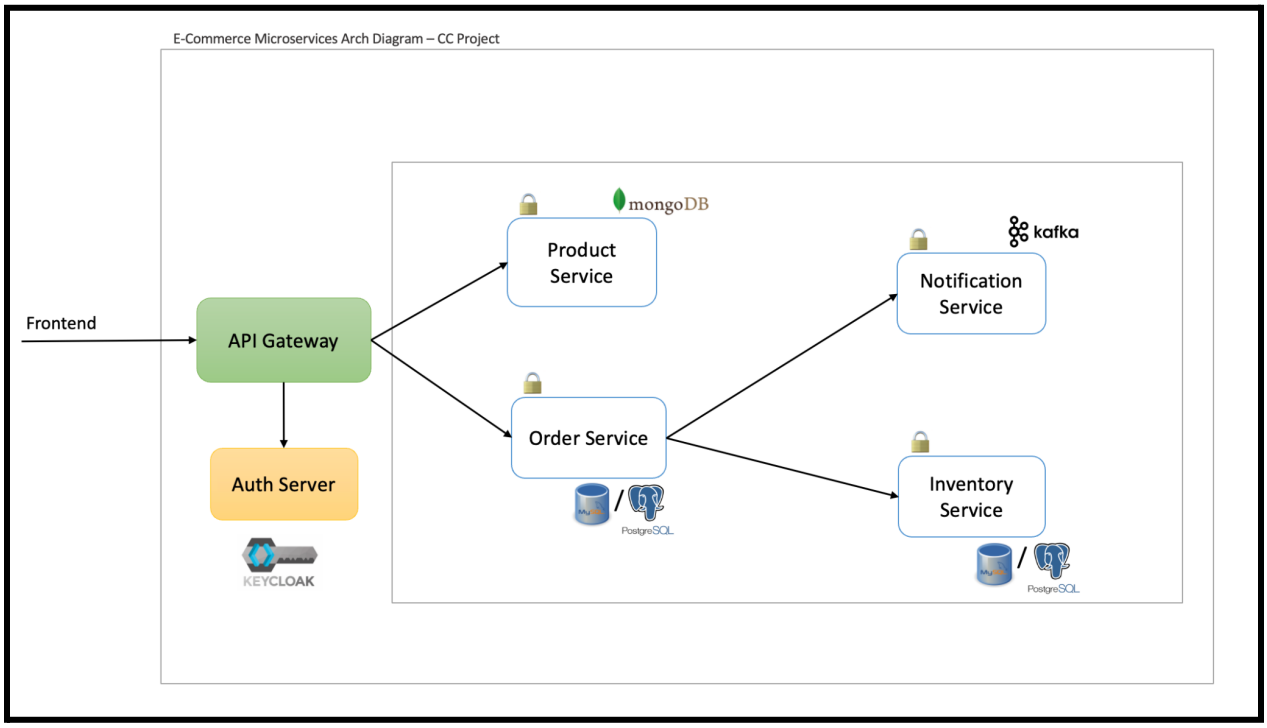
5. **Notification-service (Kafka)**: The notification-service is responsible for sending notifications-messages when an event occurs. These messages are produced by the producer service and consumed by consumer service. Kafka, an open-source distributed event streaming platform is used to carry out this.
6. **Discovery Server (Eureka**): The discovery server is used for service discovery and registration. It helps microservices to locate and communicate with each other in a distributed environment. Eureka is an open-source service discovery tool used for registering and discovering microservices in a cluster.

Task 2: Develop Microservices

Following is the microservices architecture -



Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671



All microservices are developed using Java, built with Maven. Each microservice is a Spring Boot framework application with various plugins/dependencies to support the functionality of that microservice.

Order-service is a Spring Boot Java application which uses MySQL/postgreSQL.

While locally developing the application we used MySQL, but used postgreSQL during containerisation.

Product-service is a Spring Boot Java application which uses MongoDB as it's database.

Inventory-service is a Spring Boot Java application which uses MySQL/postgreSQL.

Notification-service is a Spring Boot Java application that uses Kafka as a broker for listening for messages, handling notifications. It passes the event notification (such as OrderPlaced) from producer(order) to consumer(inventory).

Api-Gateway uses Keycloak to configure security, it authorizes exchange between the Eureka discovery server and the OAuth resource manager.

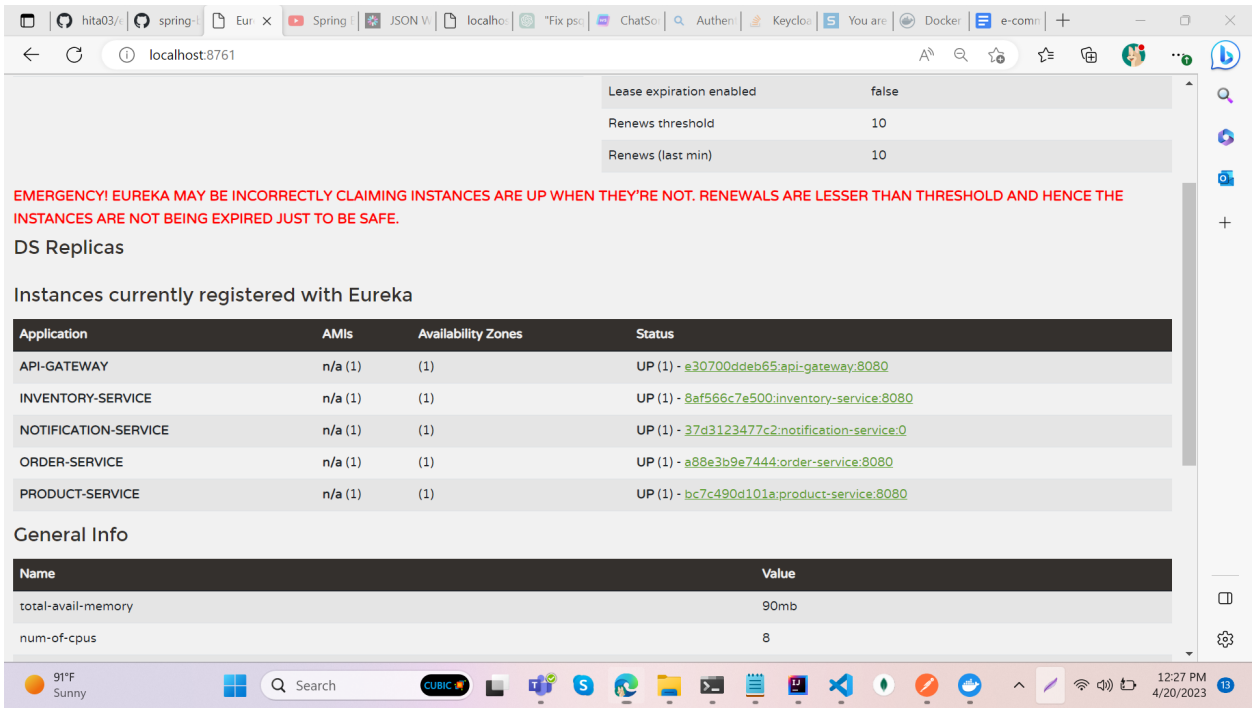
Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

Discovery Server starts up the eureka discovery service that all clients register to on startup. It stores routing information, such as service name and IP/PORT to route requests to the appropriate service.

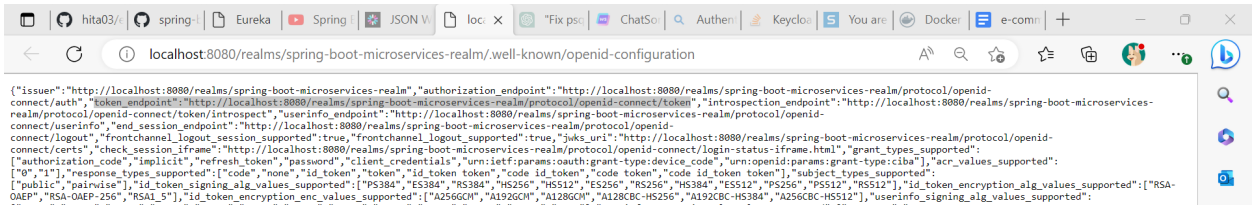
Zipkin is used as a service for trace logging a distributed system.

Screenshots

1. All services registered on discovery server

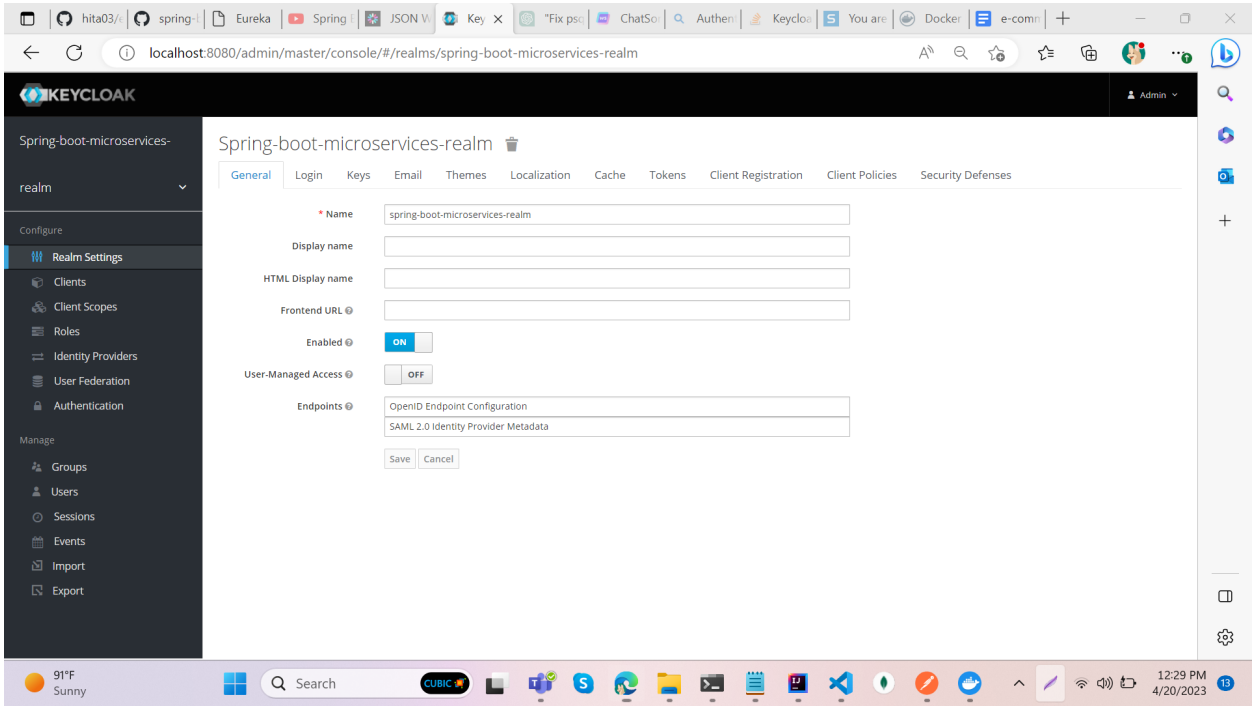


2. Endpoint URL in keycloak configuration

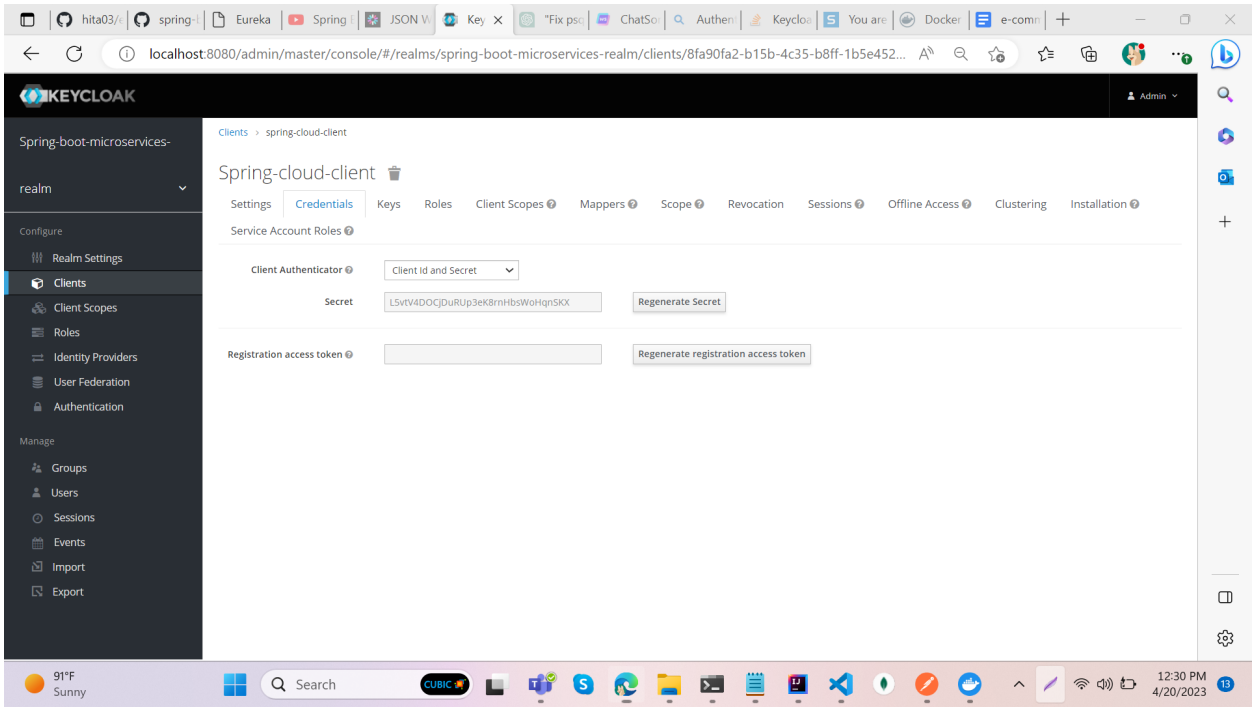


Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

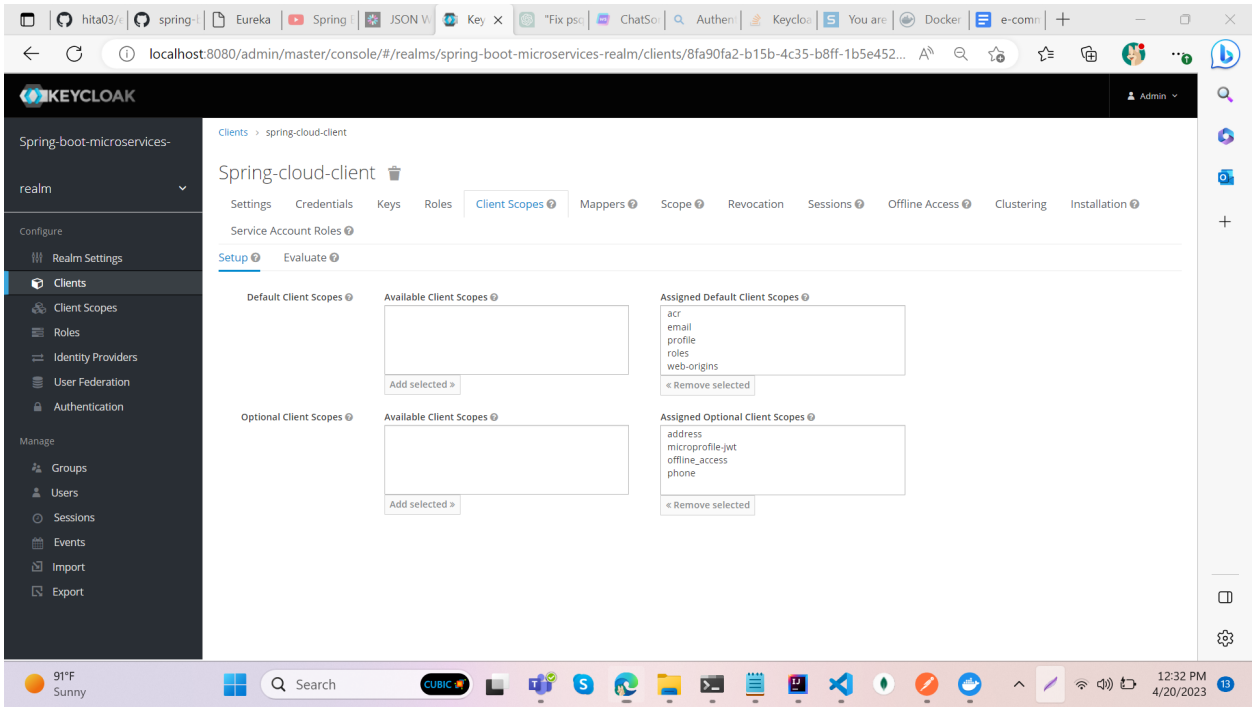
3. Spring boot microservice realm created on keycloak



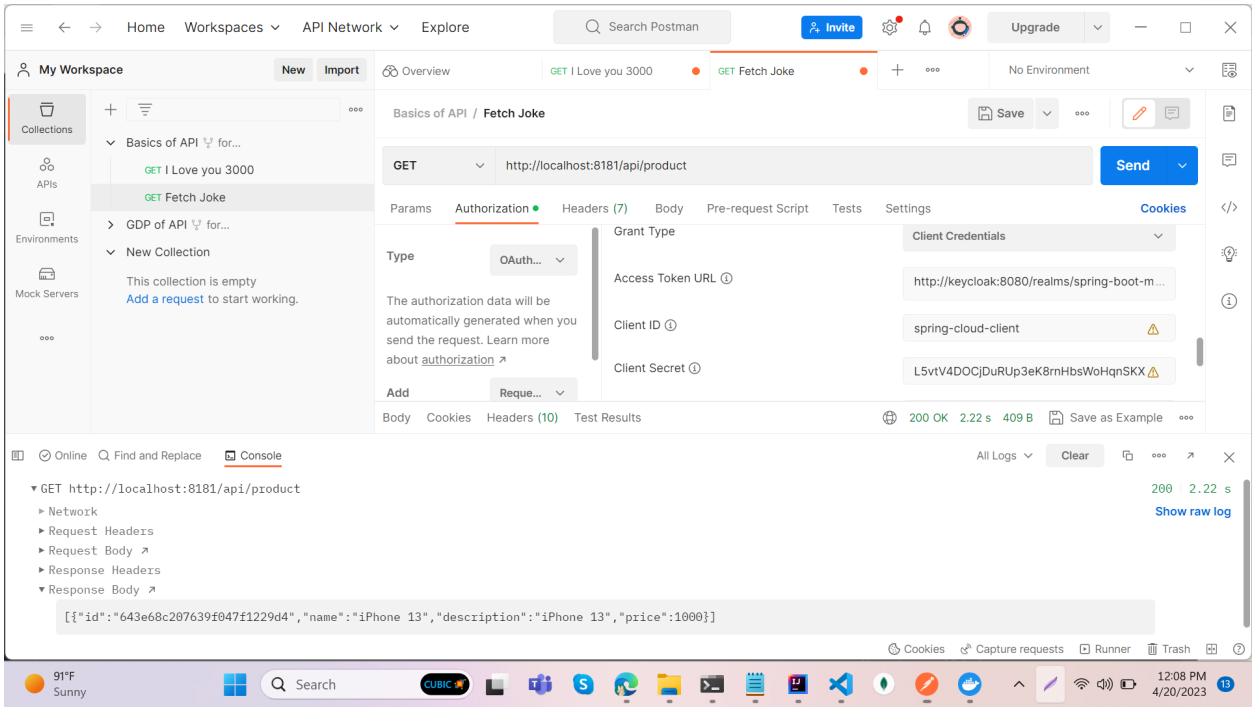
4. Spring-cloud-client created with openid token offline access(appropriate scope)



Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

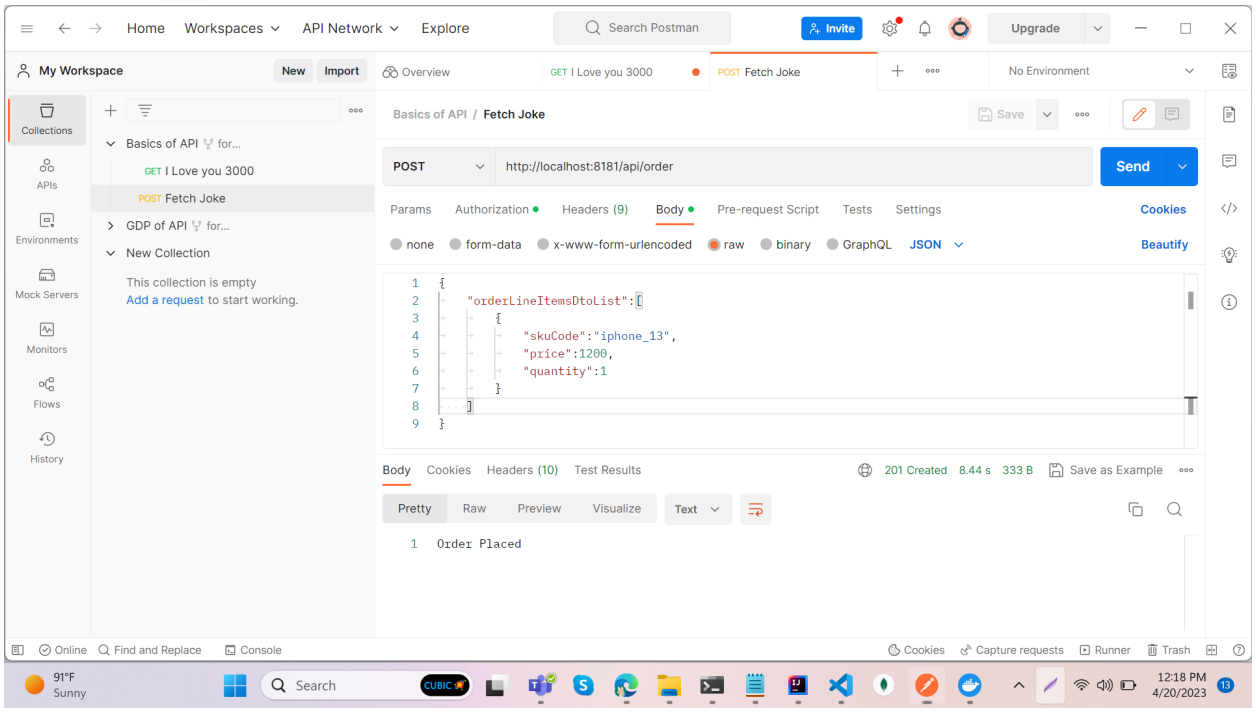


5. Successful GET request to product on Postman http://localhost:8181/api/product

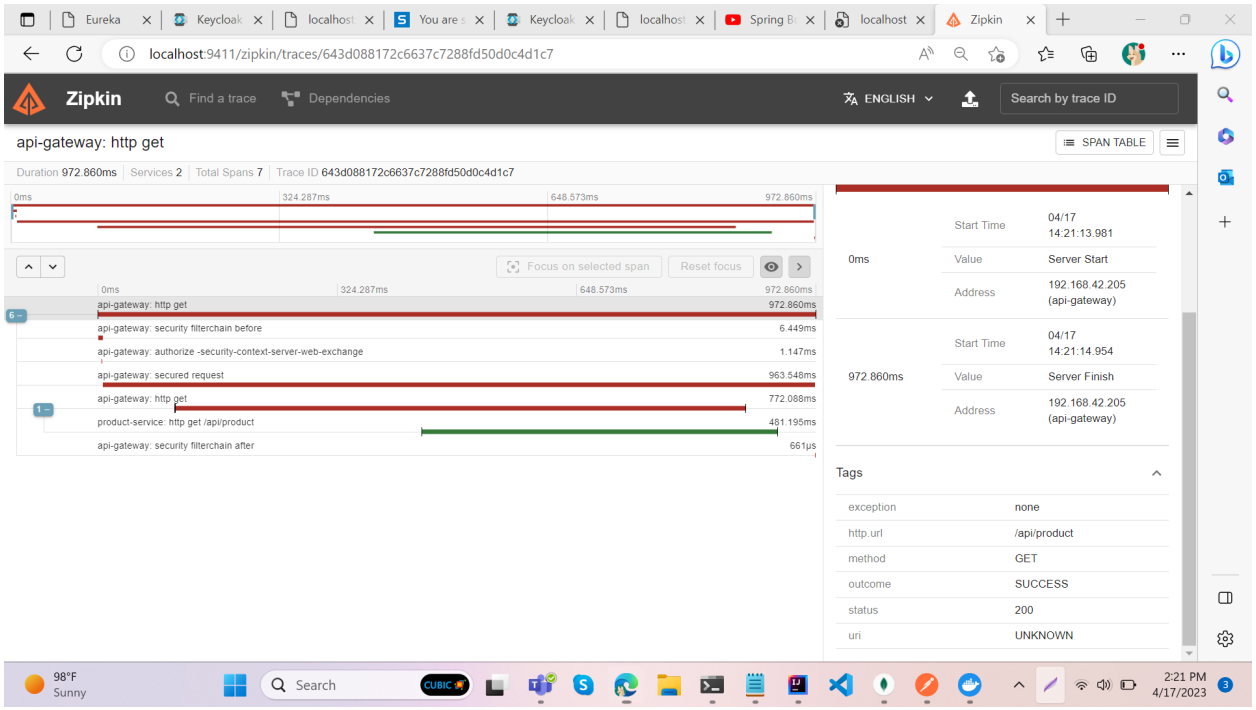


Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

6. Successful POST request to place an order on Postman http://localhost/8181/api/order

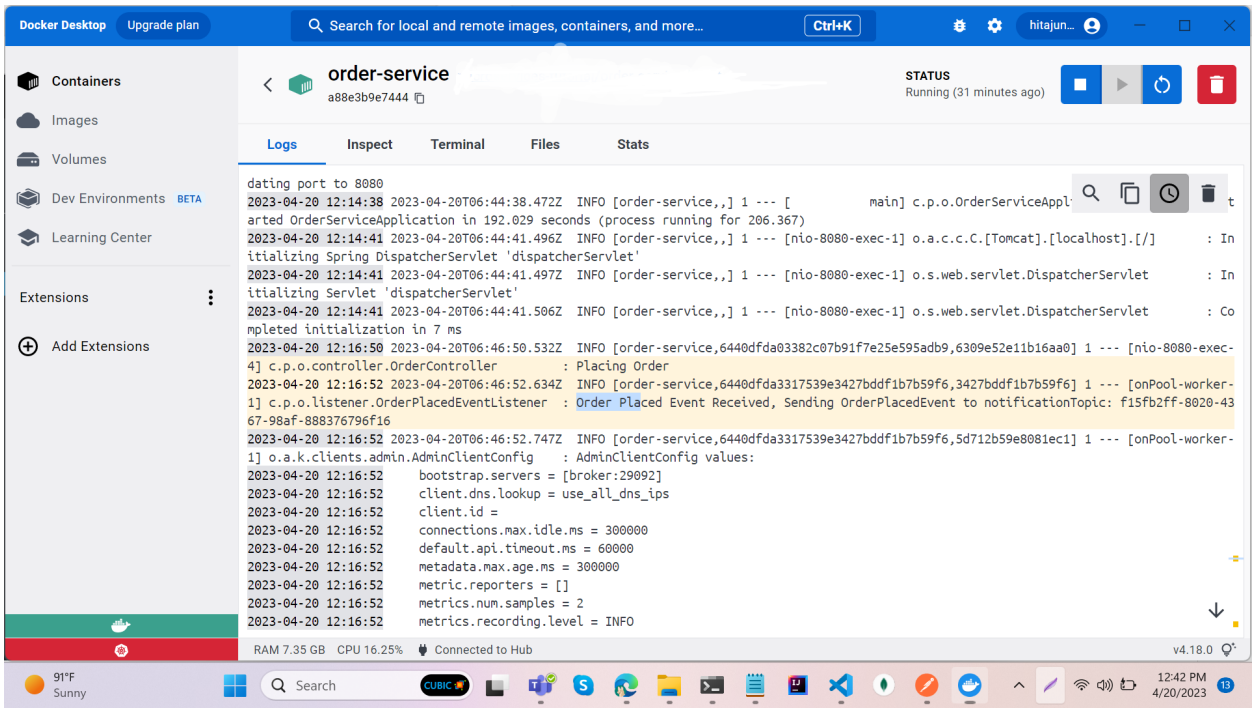


7. Zipkin trace log for order-request

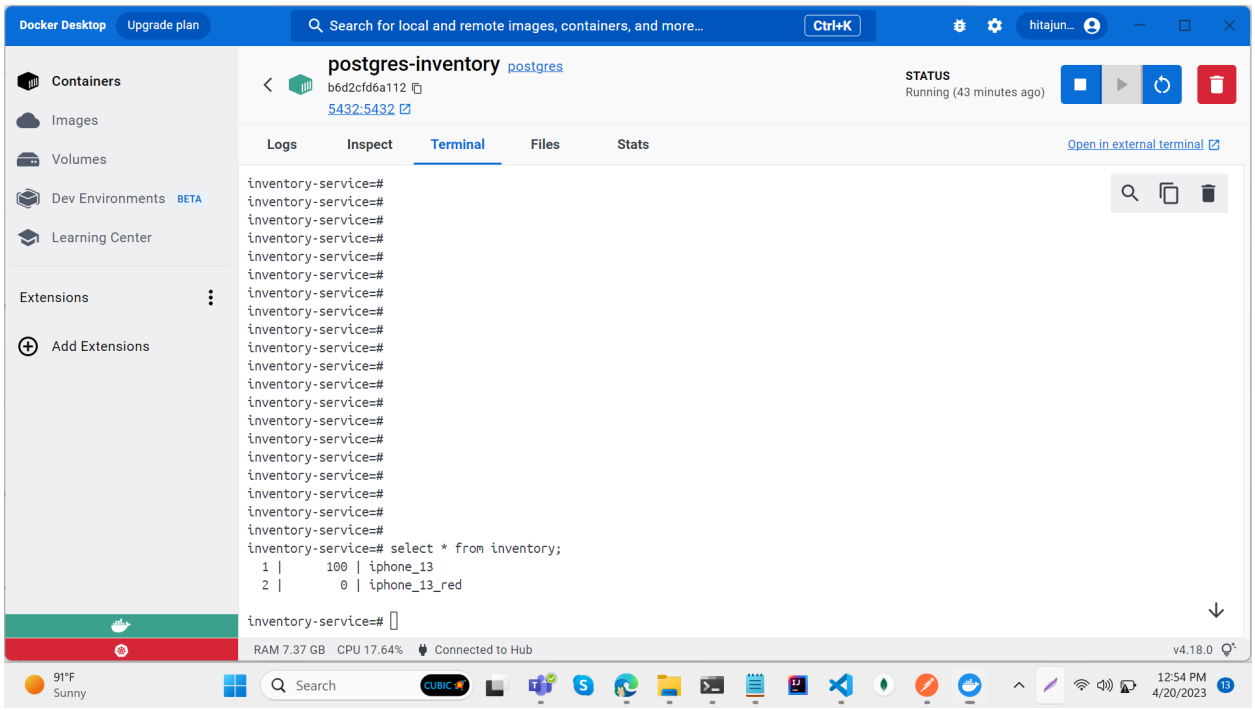


Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

8. OrderPlaced event received



9. Inventory-service database



Source code available here - https://github.com/hita03/ecommerce-microservices-CC

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

Task 3: Containerized Microservices using Docker

docker-compose.yml

```
---

version: '3.7'

services:

 ## MySQL Docker Compose Config

 postgres-order:

   container_name: postgres-order

   image: postgres

   environment:

     POSTGRES_DB: order-service

     POSTGRES_USER: root

     POSTGRES_PASSWORD: password

     PGDATA: /data/postgres

   volumes:

     - ./postgres-order:/data/postgres

   expose:

     - "5431"

   ports:

     - "5431:5431"

   command: -p 5431

   restart: always
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

```
postgres-inventory:
  container_name: postgres-inventory
  image: postgres
  environment:
    POSTGRES_DB: inventory-service
    POSTGRES_USER: root
    POSTGRES_PASSWORD: password
    PGDATA: /data/postgres
  volumes:
    - ./postgres-inventory:/data/postgres
  ports:
    - "5432:5432"
  restart: always


## Mongo Docker Compose Config
mongo:
  container_name: mongo
  image: mongo:4.4.14-rc0-focal
  restart: always
  ports:
    - "27017:27017"
  expose:
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

```
    - "27017"
  volumes:
    - ./mongo-data:/data/db


## Keycloak Config with Mysql database
keycloak-mysql:
  container_name: keycloak-mysql
  image: mysql:5.7
  volumes:
    - ./mysql_keycloak_data:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: keycloak
    MYSQL_USER: keycloak
    MYSQL_PASSWORD: password


keycloak:
  container_name: keycloak
  image: quay.io/keycloak/keycloak:18.0.0
  command: [ "start-dev", "--import-realm" ]
  environment:
    DB_VENDOR: MYSQL
    DB_ADDR: mysql
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

```
    DB_DATABASE: keycloak
    DB_USER: keycloak
    DB_PASSWORD: password
    KEYCLOAK_ADMIN: admin
    KEYCLOAK_ADMIN_PASSWORD: admin
  ports:
    - "8080:8080"
  volumes:
    - ./realms/:/opt/keycloak/data/import/
  depends_on:
    - keycloak-mysql


zookeeper:
  image: confluentinc/cp-zookeeper:7.0.1
  container_name: zookeeper
  ports:
    - "2181:2181"
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000


broker:
  image: confluentinc/cp-kafka:7.0.1
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

```
  container_name: broker

  ports:

    - "9092:9092"

  depends_on:

    - zookeeper

  environment:

    KAFKA_BROKER_ID: 1

    KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'

    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_INTERNAL:PLAINTEXT

    KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://localhost:9092,PLAINTEXT_INTERNAL://broker:29092

    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

    KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1

    KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1



## Zipkin

zipkin:

  image: openzipkin/zipkin

  container_name: zipkin

  ports:

    - "9411:9411"



## Eureka Server
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

```
discovery-server:

  image: hitajuneja/discovery-server:latest

  container_name: discovery-server

  ports:

    - "8761:8761"

  environment:

    - SPRING_PROFILES_ACTIVE=docker

  depends_on:

    - zipkin



api-gateway:

  image: hitajuneja/api-gateway:latest

  container_name: api-gateway

  ports:

    - "8181:8080"

  expose:

    - "8181"

  environment:

    - SPRING_PROFILES_ACTIVE=docker

    - LOGGING_LEVEL_ORG_SPRINGFRAMEWORK_SECURITY= TRACE

  depends_on:

    - zipkin

    - discovery-server
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

```
    - keycloak



## Product-Service Docker Compose Config

product-service:

  container_name: product-service

  image: hitajuneja/product-service:latest

  environment:

    - SPRING_PROFILES_ACTIVE=docker

  depends_on:

    - mongo

    - discovery-server

    - api-gateway



## Order-Service Docker Compose Config

order-service:

  container_name: order-service

  image: hitajuneja/order-service:latest

  environment:

    - SPRING_PROFILES_ACTIVE=docker

    - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres-order:5431/order-service

  depends_on:

    - postgres-order

    - broker
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

```
    - zipkin
    - discovery-server
    - api-gateway


## Inventory-Service Docker Compose Config
inventory-service:
  container_name: inventory-service
  image: hitajuneja/inventory-service:latest
  environment:
    - SPRING_PROFILES_ACTIVE=docker
    -
SPRING_DATASOURCE_URL=jdbc:postgresql://postgres-inventory:5432/inventory-service
  depends_on:
    - postgres-inventory
    - discovery-server
    - api-gateway


## Notification-Service Docker Compose Config
notification-service:
  container_name: notification-service
  image: hitajuneja/notification-service:latest
  environment:
    - SPRING_PROFILES_ACTIVE=docker
  depends_on:
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

```
    - zipkin
    - broker
    - discovery-server
    - api-gateway


## Prometheus
prometheus:
  image: prom/prometheus:v2.37.1
  container_name: prometheus
  restart: always
  ports:
    - "9090:9090"
  volumes:
    - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
  depends_on:
    - product-service
    - inventory-service
    - order-service
    - notification-service


grafana:
  image: grafana/grafana-oss:8.5.2
  container_name: grafana
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

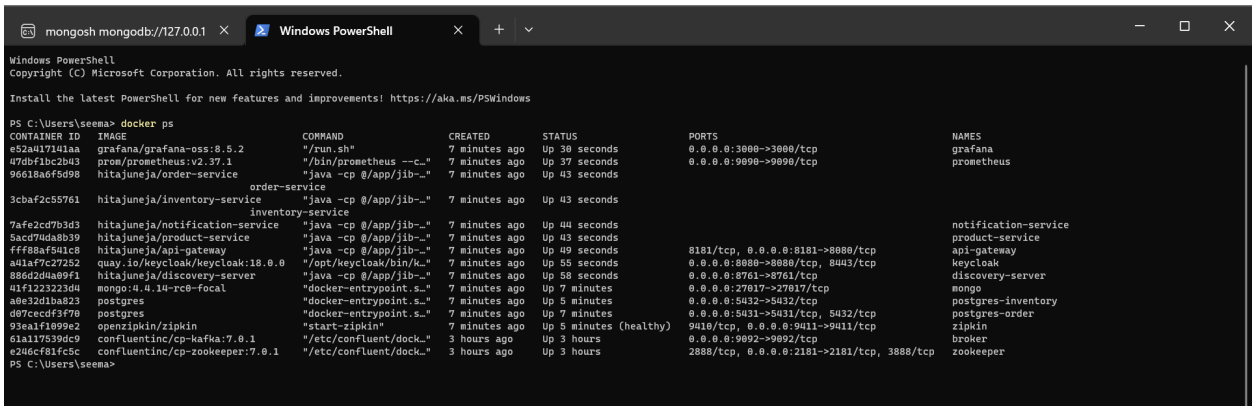
```
restart: always

ports:

  - "3000:3000"

links:

  - prometheus:prometheus

volumes:

  - ./grafana:/var/lib/grafana

environment:

  - GF_SECURITY_ADMIN_USER=admin

  - GF_SECURITY_ADMIN_PASSWORD=password
```

Common Dockerfile for running all microservices locally

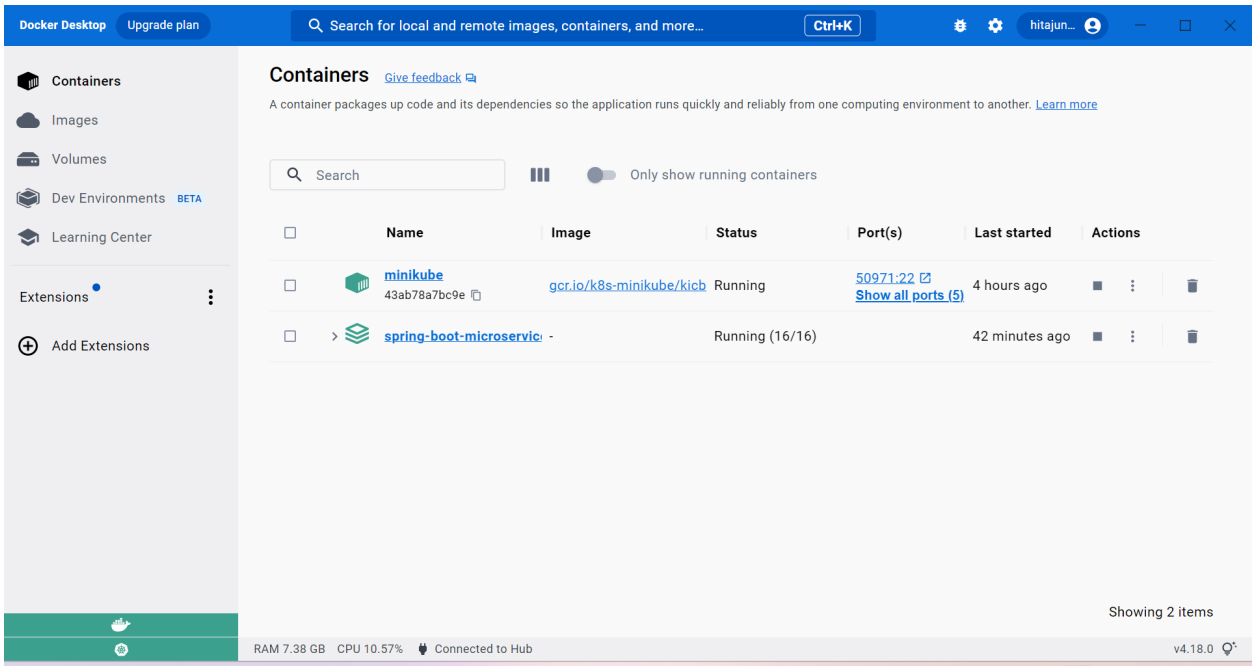
```
1   FROM openjdk:17
2
3   COPY target/*.jar app.jar
4
5   ENTRYPOINT ["java","-jar","/app.jar"]
6
```

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

Docker containers running(16/16)

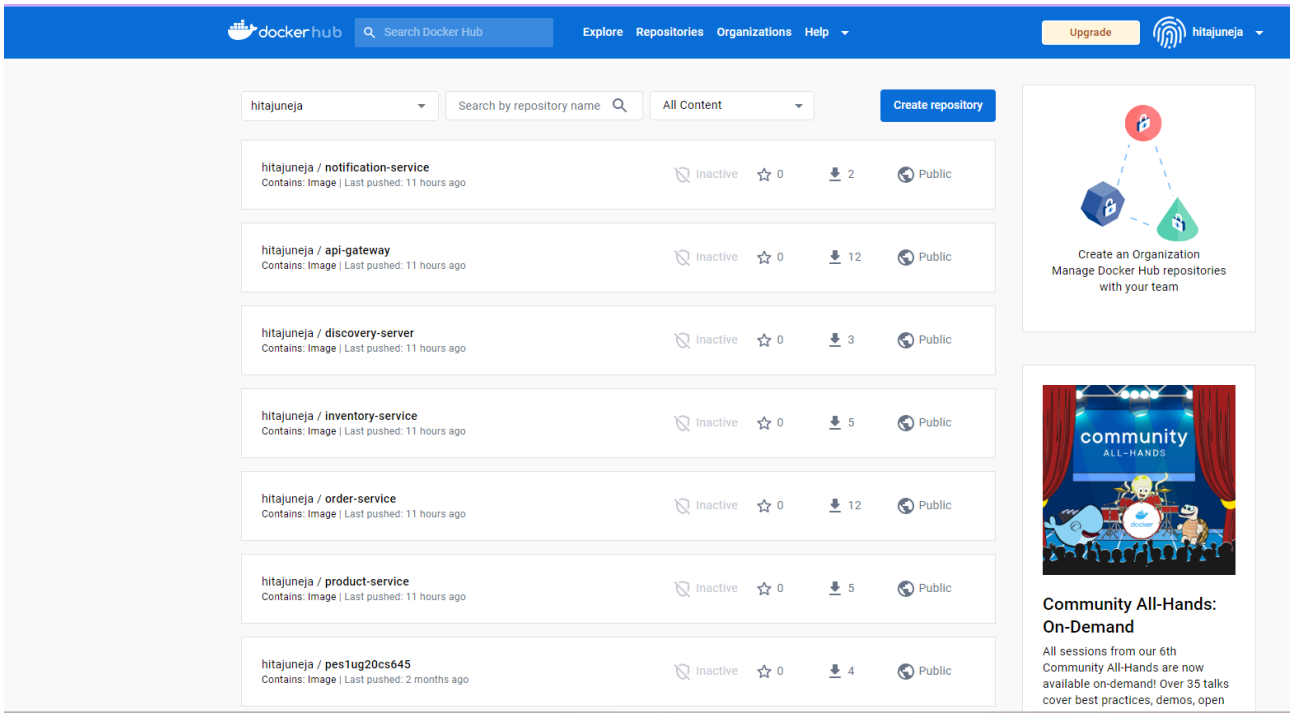


Microservice running on docker desktop



Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

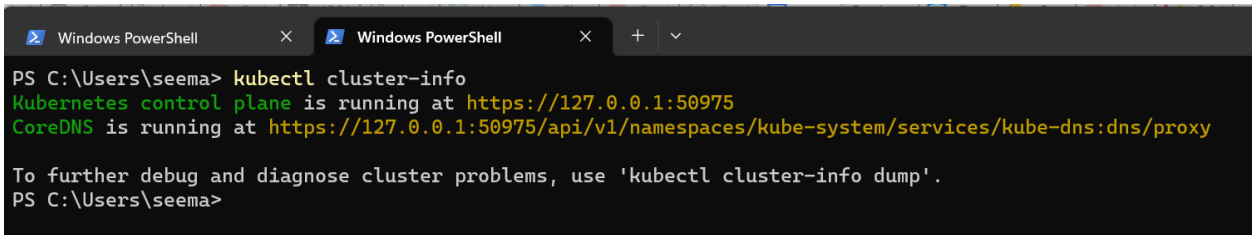
Docker Images Pushed to Hub



Task 4: Orchestrate Microservices using Kubernetes

“Kubectl cluster-info”

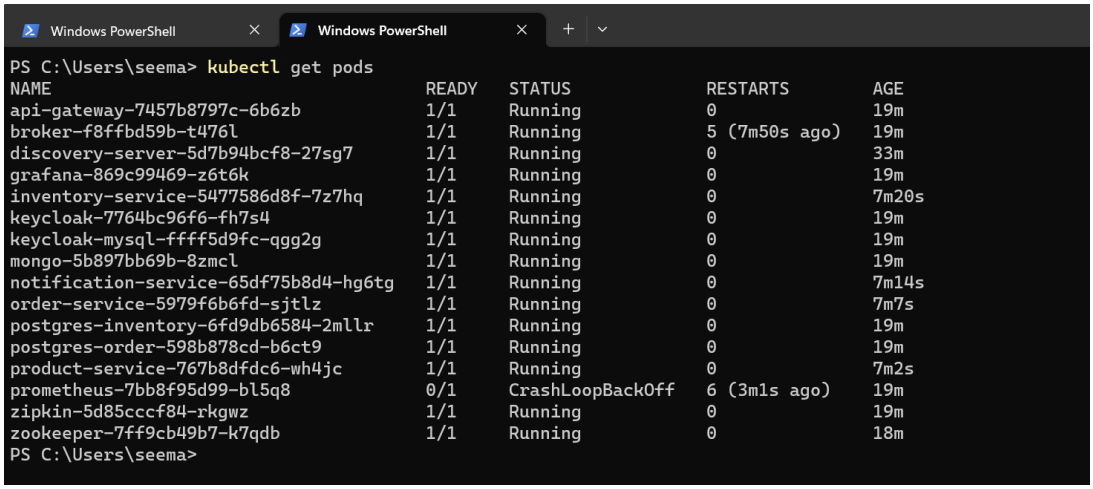
This command provides an overview of the Kubernetes cluster's endpoint and the Kubernetes API server's health status.



“Kubectl get pods”

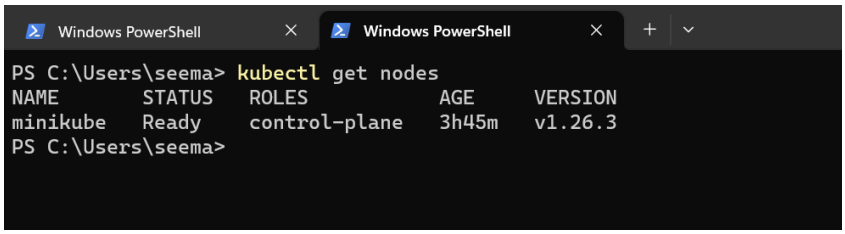
This command lists all the pods that are currently running in the cluster. A pod is the smallest deployable unit in Kubernetes, and it represents a single instance of a running process in the cluster.

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671



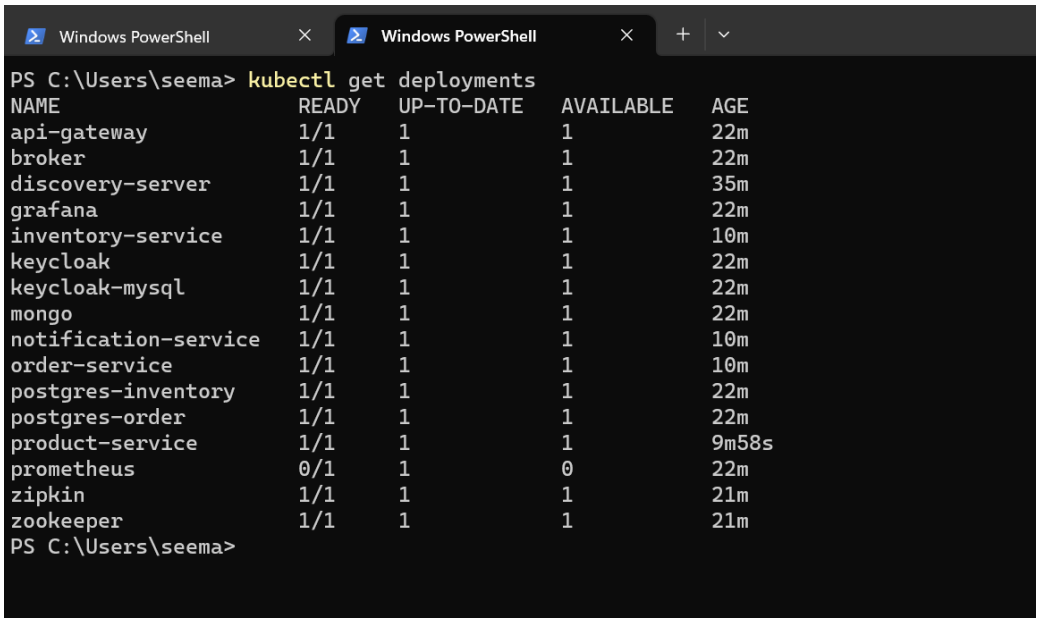
“Kubectl get nodes”

This command lists all the nodes that are currently part of the cluster. A node is a physical or virtual machine that runs Kubernetes and is responsible for running containers.



“Kubectl get deployments”

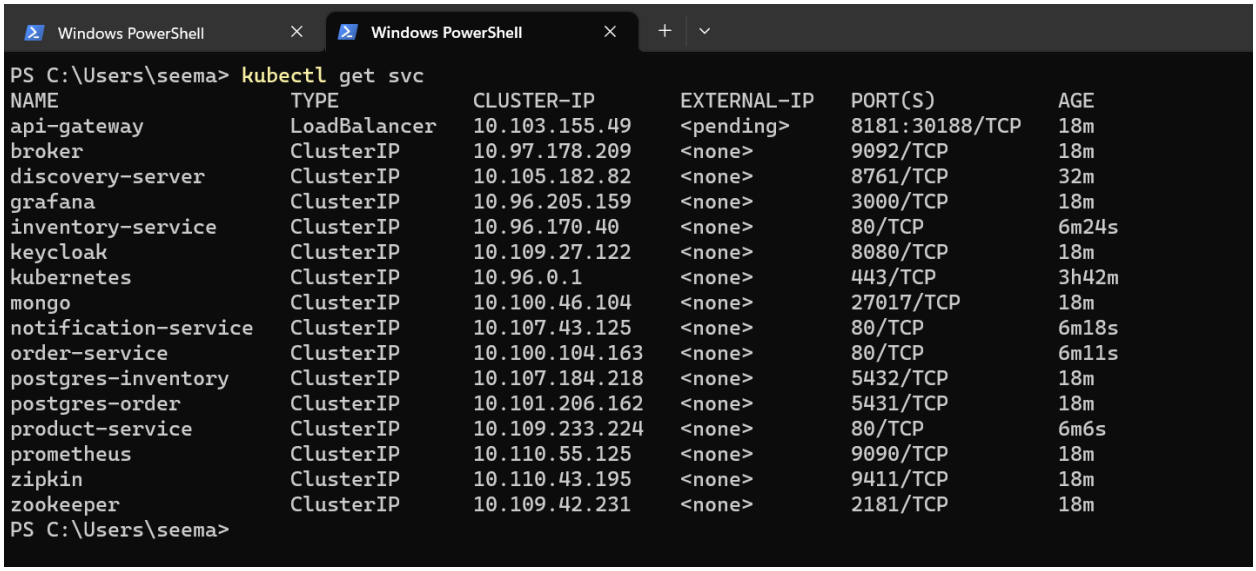
This command lists all the deployments that are currently running in the cluster. A deployment is a higher-level abstraction that manages one or more replicas of a pod.



Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

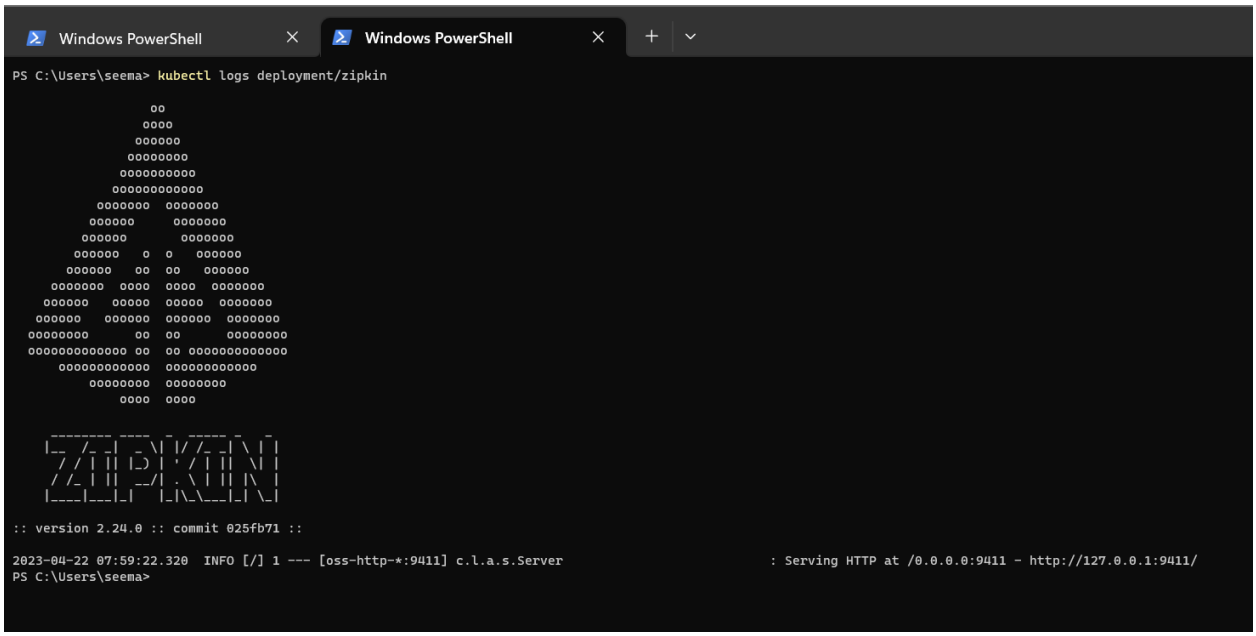
“Kubectl get svc”

This command lists all the services that are currently running in the cluster. A service is an abstraction that provides a stable IP address and DNS name for accessing one or more pods that provide the same functionality.



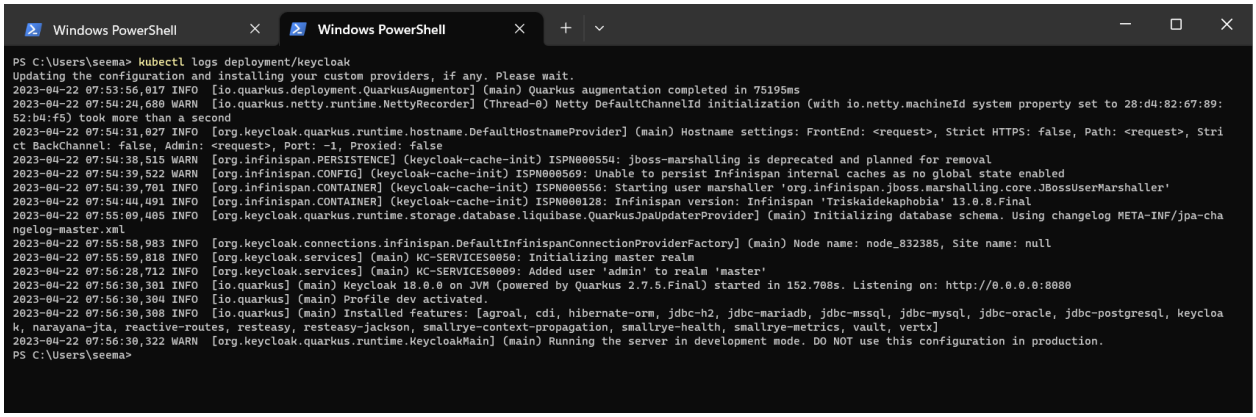
Testing and validating the Kubernetes deployment of a few services: Zipkin and Keycloak.

1. Logs of Zipkin service



Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

2. Logs of keycloak service



Task 5: Implement Continuous Integration and Deployment using Jenkins

Checkout: This stage typically involves checking out the source code from a version control system such as Git or SVN. This stage is essential because it ensures that the latest changes are pulled from the repository before the build process starts.

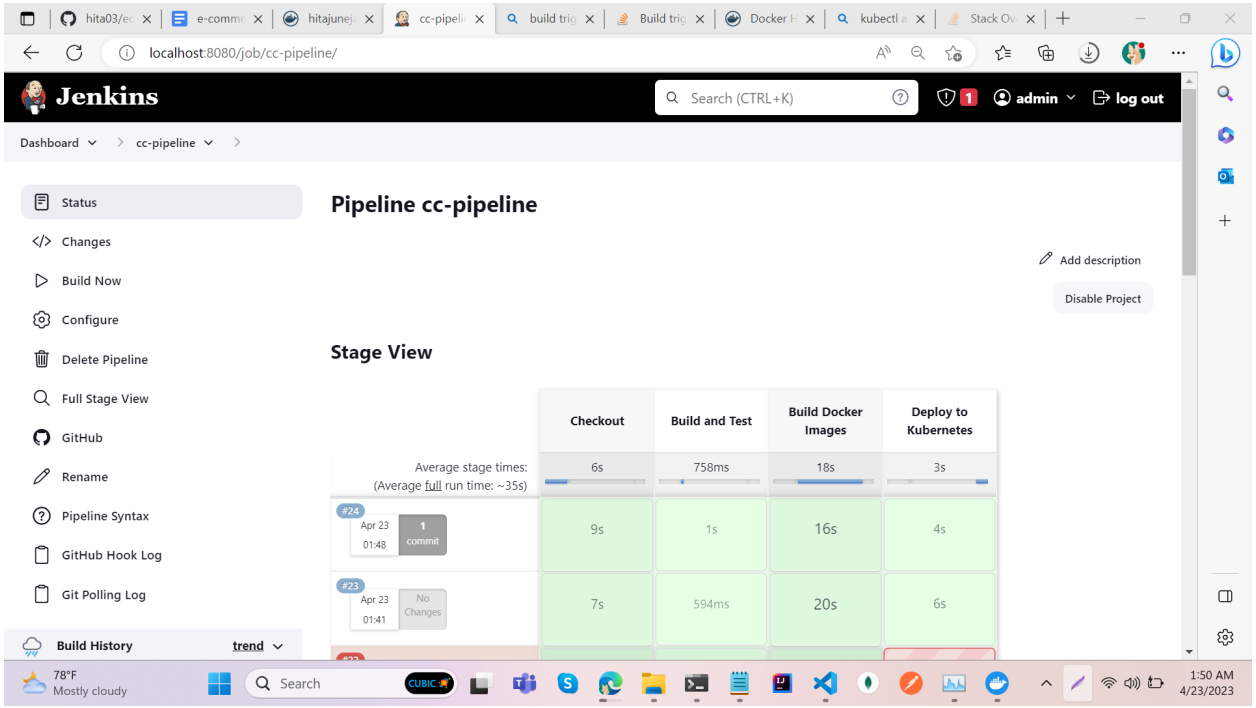
Build and test: This stage involves compiling the code and running automated tests to ensure that the code is working as expected. This stage may include several sub-stages such as unit tests, integration tests, and functional tests.

Build docker images: This stage involves building the Docker images that will be used to deploy the application. Docker images are typically used to package an application along with its dependencies into a portable container.

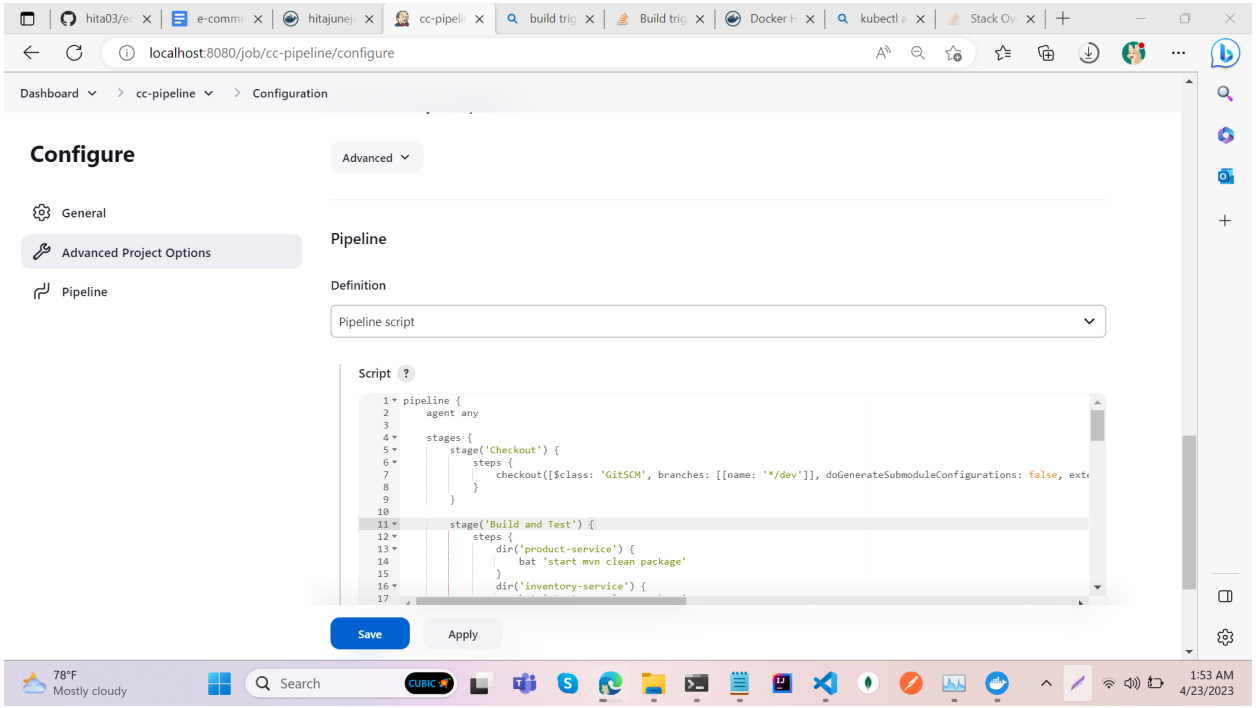
Deploy to Kubernetes: This stage involves deploying the Docker images to a Kubernetes cluster. Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

Jenkins Job: cc-pipeline



Pipeline Configuration:



Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

Pipeline script:

```
pipeline {
   agent any

   stages {
       stage('Checkout') {
           steps {
               checkout([$class: 'GitSCM', branches: [[name: '*/dev']],
doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [],
userRemoteConfigs: [[credentialsId: 'ae4f6f2b-bd6a-4930-8a82-de11b050d9a3', url:
'https://github.com/hita03/ecommerce-microservices-CC']]])
           }
       }

       stage('Build and Test') {
           steps {
               dir('product-service') {
                   bat 'start mvn clean package'
               }
               dir('inventory-service') {
                   bat 'start mvn clean package'
               }
               dir('order-service') {
                   bat 'start mvn clean package'
               }
               dir('notification-service') {
                   bat 'start mvn clean package'
               }
           }
       }

       stage('Build Docker Images') {
           steps {
               dir('product-service') {
                   script {

docker.withRegistry('https://registry.hub.docker.com','docker-registry-credentials')
                       {
                           bat 'docker push hitajuneja/product-service:latest'
                       }
```

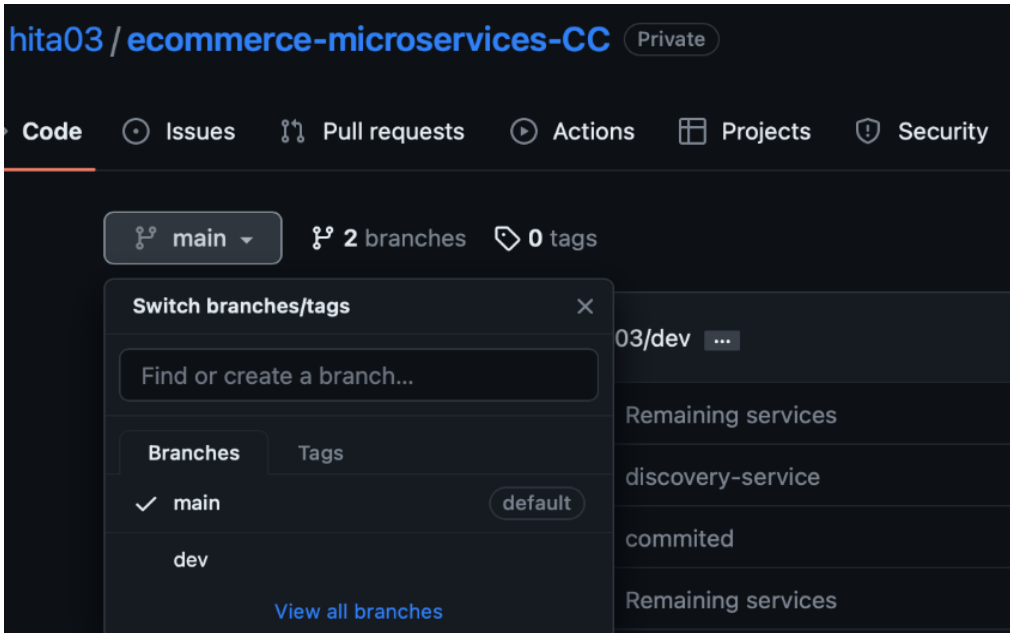
Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671

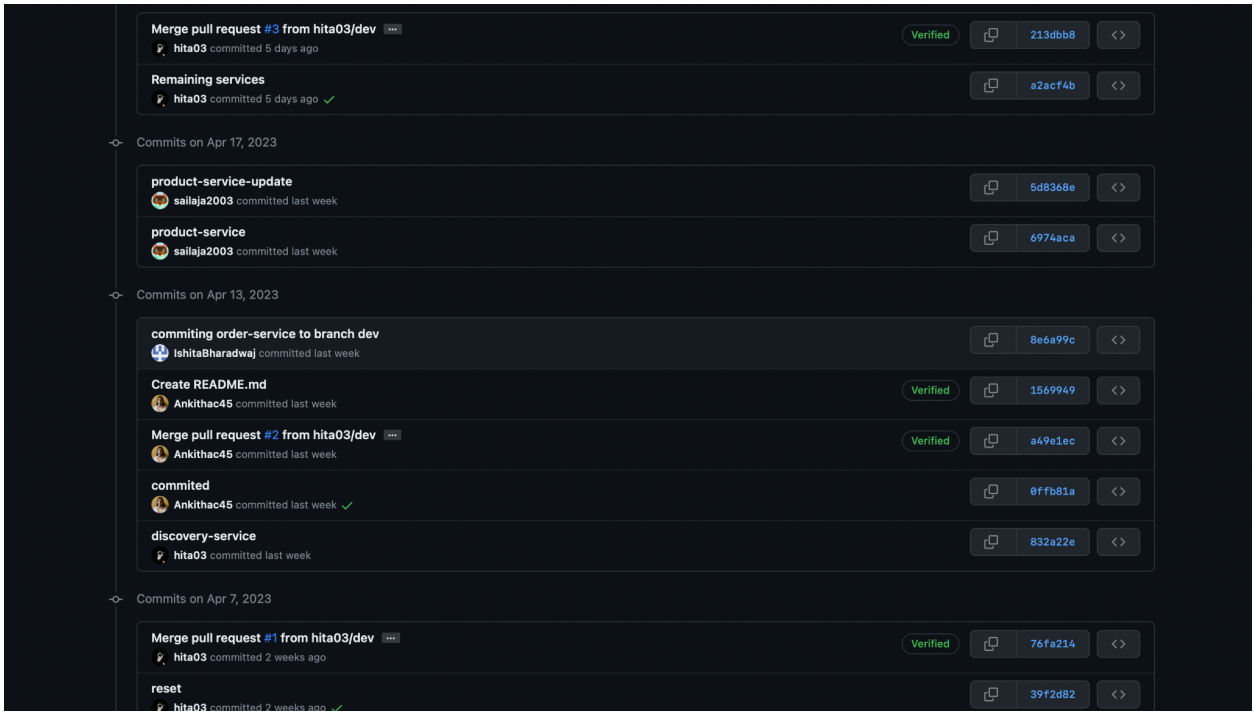
```
docker.withRegistry('https://registry.hub.docker.com','docker-registry-credentials')
                        {
                            bat 'docker push hitajuneja/notification-service:latest'
                        }

docker.withRegistry('https://registry.hub.docker.com','docker-registry-credentials')
                        {
                            bat 'docker push hitajuneja/order-service:latest'
                        }

docker.withRegistry('https://registry.hub.docker.com','docker-registry-credentials')
                        {
                            bat 'docker push hitajuneja/inventory-service:latest'
                        }

                    }
                }
            }
        }

        stage('Deploy to Kubernetes') {
            steps {
                script{
                    bat 'kubectl apply -f k8s/services/product-service/'
                    bat 'kubectl apply -f k8s/services/order-service/'
                    bat 'kubectl apply -f k8s/services/inventory-service/'
                    bat 'kubectl apply -f k8s/services/notification-service/'
                }
            }
        }
    }
}
```

Task 6: Version Control using Git

We have two branches, main and dev.
It is important to keep the main branch stable and only merge changes from the dev branch after testing. This ensures that the main branch always contains a stable version of the code.

Ankitha C PES1UG20CS626
Hita Juneja PES1UG20CS645
Ishita Bharadwaj PES1UG20CS648
Sai Sailaja Policharla PES1UG20CS671





Project Link : https://github.com/hita03/ecommerce-microservices-CC

THANK YOU!

🐳🐳🐳🐳🐳🐳🐳🐳