

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:

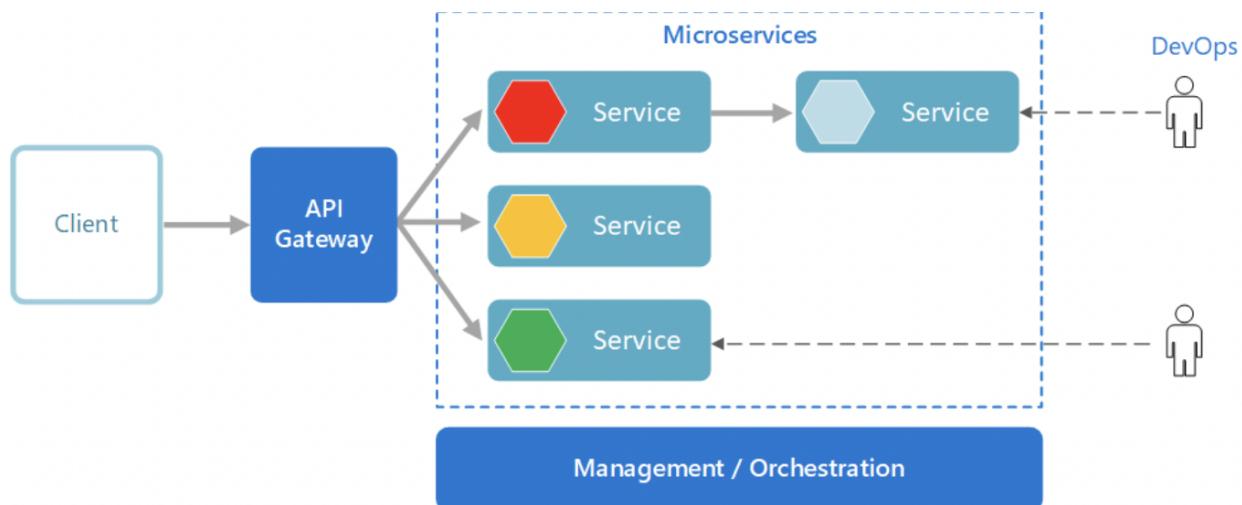
1. Docker documentation: <https://docs.docker.com/>
2. Kubernetes documentation: <https://kubernetes.io/docs/home/>
3. Jenkins documentation: <https://www.jenkins.io/doc/>
4. Git documentation: <https://git-scm.com/doc> ; Git tutorial for beginners: <https://www.atlassian.com/git/tutorials>
5. Deploying 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/>

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.

- 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)

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 -

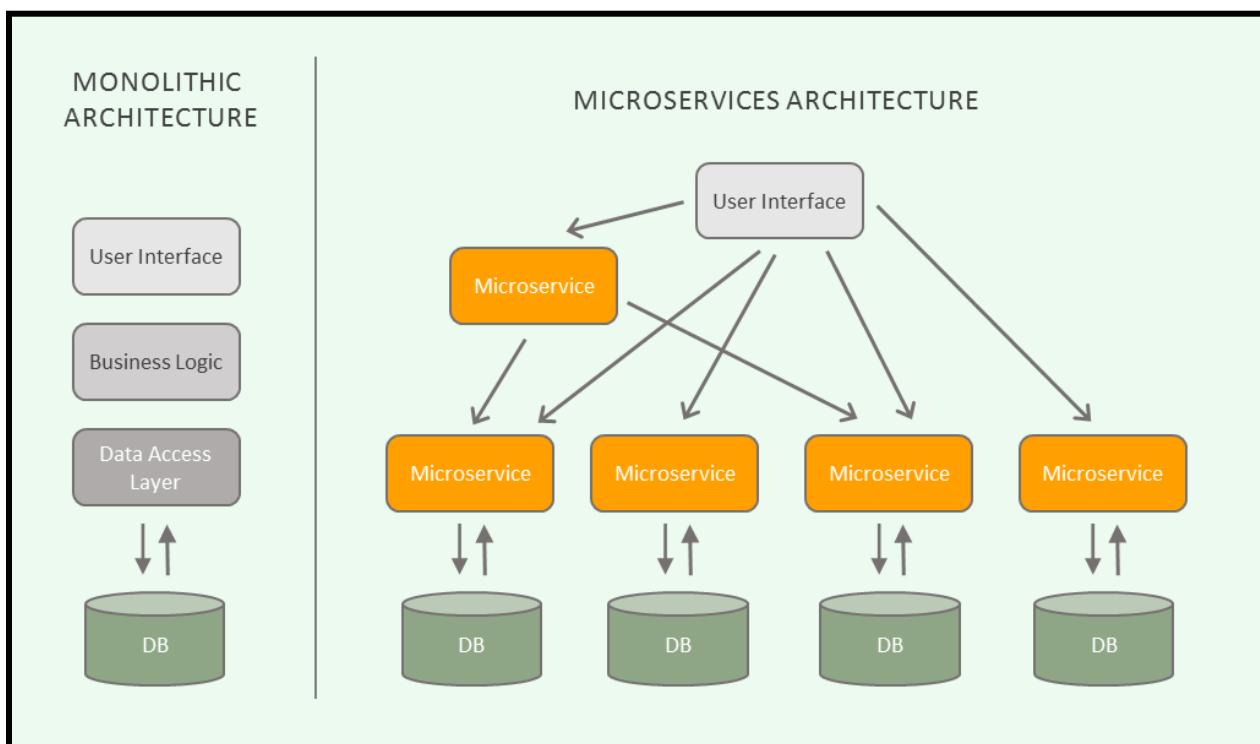
```
[  
    {  
        "id": 20937456349393,  
        "name": "iPhone 13",  
        "description": "iPhone 13",  
        "price": 1000  
    }
```

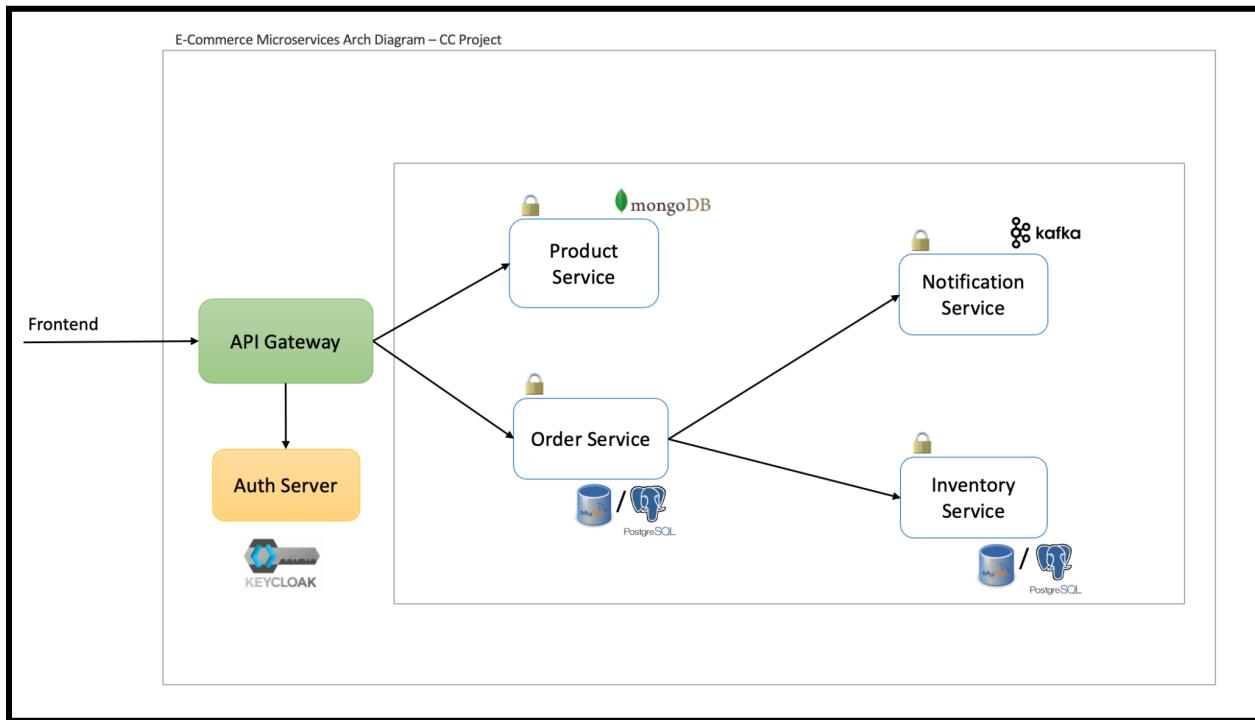
]

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 -





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 postgresSQL 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.

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

Lease expiration enabled: false
Renew threshold: 10
Renews (last min): 10

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - e30700ddeb65.api-gateway:8080
INVENTORY-SERVICE	n/a (1)	(1)	UP (1) - 8af566c7e500:inventory-service:8080
NOTIFICATION-SERVICE	n/a (1)	(1)	UP (1) - 37d3123477c2:notification-service:0
ORDER-SERVICE	n/a (1)	(1)	UP (1) - a88e3b9e7444:order-service:8080
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - bc7c490d101a:product-service:8080

General Info

Name	Value
total-avail-memory	90mb
num-of-cpus	8

2. Endpoint URL in keycloak configuration

```
{"issuer": "http://localhost:8080/realm/spring-boot-microservices-realm", "authorization_endpoint": "http://localhost:8080/realm/spring-boot-microservices-realm/protocol/openid-connect/auth", "token_endpoint": "http://localhost:8080/realm/spring-boot-microservices-realm/protocol/openid-connect/token", "introspection_endpoint": "http://localhost:8080/realm/spring-boot-microservices-realm/protocol/openid-connect/introspect", "userinfo_endpoint": "http://localhost:8080/realm/spring-boot-microservices-realm/protocol/openid-connect/userinfo", "end_session_endpoint": "http://localhost:8080/realm/spring-boot-microservices-realm/protocol/openid-connect/logout", "frontchannel_logout_supported": true, "frontchannel_logout_supported": true, "jwks_uri": "http://localhost:8080/realm/spring-boot-microservices-realm/protocol/openid-connect/certs", "check_session_iframe": "http://localhost:8080/realm/spring-boot-microservices-realm/protocol/openid-connect/connect/login-status-iframe.html", "grant_types_supported": ["authorization_code", "implicit", "refresh_token", "password", "client_credentials", "urn:ietf:params:oauth:grant-type:device_code", "urn:openid:params:grant-type:ciba"], "acr_values_supported": ["0", "1"], "response_types_supported": ["code", "none", "id_token", "token", "id_token token", "code id_token", "code token", "code id_token token"], "subject_types_supported": ["public", "pairwise"], "id_token_signing_alg_values_supported": ["PS384", "ES384", "RS256", "HS256", "E5256", "RS256", "HS384", "ESS12", "PS256", "PSS12", "RS512"], "id_token_encryption_alg_values_supported": ["RSA-OAEP", "RSA-OAEP-256", "RSA1_5"], "id_token_encryption_enc_values_supported": ["A256GCM", "A192GCM", "A128GCM", "A128CBC-HS256", "A192CBC-HS384", "A256CBC-HS512"], "user_info_signing_alg_values_supported": []}
```

3. Spring boot microservice realm created on keycloak

The screenshot shows the Keycloak Admin Console interface. On the left, a sidebar menu for the 'Spring-boot-microservices-realm' is visible, with 'Realm Settings' selected. The main panel displays the 'General' tab of the realm configuration. The 'Name' field is set to 'spring-boot-microservices-realm'. Other fields like 'Display name', 'HTML Display name', 'Frontend URL', and 'Enabled' (set to 'ON') are also present. Under 'Endpoints', 'OpenID Endpoint Configuration' and 'SAML 2.0 Identity Provider Metadata' are listed. At the bottom right of the main panel are 'Save' and 'Cancel' buttons. The top navigation bar shows the URL 'localhost:8080/admin/master/console/#/realms/spring-boot-microservices-realm'. The system tray at the bottom indicates it's 12:29 PM on April 20, 2023.

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

The screenshot shows the Keycloak Admin Console interface. On the left, a sidebar menu for the 'Spring-boot-microservices-realm' is visible, with 'Clients' selected. The main panel displays the 'Credentials' tab for a client named 'spring-cloud-client'. Under 'Client Authenticator', 'Client Id and Secret' is selected, and a secret value 'L5vtV4DOCJuRUp3eK8rnHbsWoHqn5KK' is shown. A 'Regenerate Secret' button is available. Below this, there is a 'Registration access token' input field and a 'Regenerate registration access token' button. The top navigation bar shows the URL 'localhost:8080/admin/master/console/#/realms/spring-boot-microservices-realm/clients/8fa90fa2-b15b-4c35-b8ff-1b5e452...'. The system tray at the bottom indicates it's 12:30 PM on April 20, 2023.

The screenshot shows the Keycloak Admin UI for a realm named "Spring-boot-microservices-realm". The left sidebar shows navigation options like Clients, Client Scopes, Roles, Identity Providers, User Federation, and Authentication. The main content area is titled "Clients > spring-cloud-client". The "Client Scopes" tab is active. It displays two sections: "Default Client Scopes" and "Optional Client Scopes", each with a list of available scopes and buttons to "Add selected" or "Remove selected". The "Assigned Default Client Scopes" and "Assigned Optional Client Scopes" sections show the scopes that have been assigned to the client.

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

The screenshot shows the Postman application interface. On the left, there's a sidebar with "My Workspace" containing collections like "Basics of API" and "New Collection". The main workspace shows a "GET I Love you 3000" and a "GET Fetch Joke" request. The "GET Fetch Joke" request is selected and configured to "http://localhost:8181/api/product". The "Authorization" tab is selected, showing OAuth... type and Client Credentials grant type with "http://keycloak:8080/realm.../clients/spring-cloud-client" as the Access Token URL, "spring-cloud-client" as the Client ID, and "L5vtV4DOCjDuUp3eK8rnHbsWoHqnSKX" as the Client Secret. The "Body" tab shows a JSON response: [{"id": "643e68c207639f047f1229d4", "name": "iPhone 13", "description": "iPhone 13", "price": 1000}]. The bottom status bar shows "200 2.22 s".

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

The screenshot shows the Postman interface. On the left, the 'My Workspace' sidebar lists collections, APIs, environments, mock servers, monitors, flows, and history. The main workspace shows a collection named 'Basics of API / Fetch Joke' containing a POST request to 'Fetch Joke'. The request URL is `http://localhost:8181/api/order`. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "orderLineItemsDtosList": [  
3     {  
4       "skuCode": "iphone_13",  
5       "price": 1200,  
6       "quantity": 1  
7     }  
8   ]  
9 }
```

The response status is 201 Created, with a time of 8.44 s and a size of 333 B. The response body is 'Order Placed'.

7. Zipkin trace log for order-request

The screenshot shows the Zipkin UI displaying a trace log for an order-request. The top navigation bar includes links for Eureka, Keycloak, localhost, You are, Keycloak, localhost, Spring Boot, localhost, and Zipkin. The main area shows a timeline of spans for the 'api-gateway' service. A specific trace ID is highlighted, showing the sequence of events:

- api-gateway: http get (Duration: 972.860ms)
- api-gateway: security filterchain before
- api-gateway: authorize -security-context-server-web-exchange
- api-gateway: secured request
- api-gateway: http get (Duration: 772.088ms)
- product-service: http get /api/product (Duration: 481.195ms)
- api-gateway: security filterchain after

On the right side, detailed information for the first span is shown:

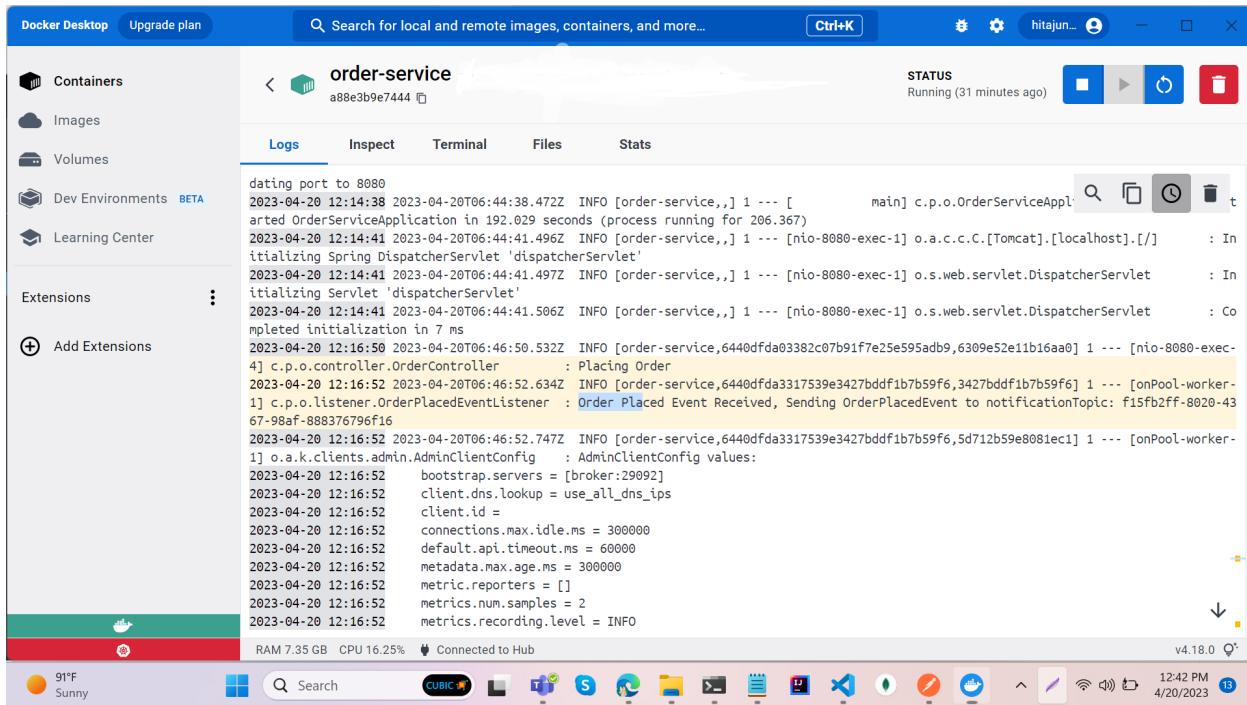
Start Time	Value
0ms	Server Start
972.860ms	Server Finish

Tags for the span include:

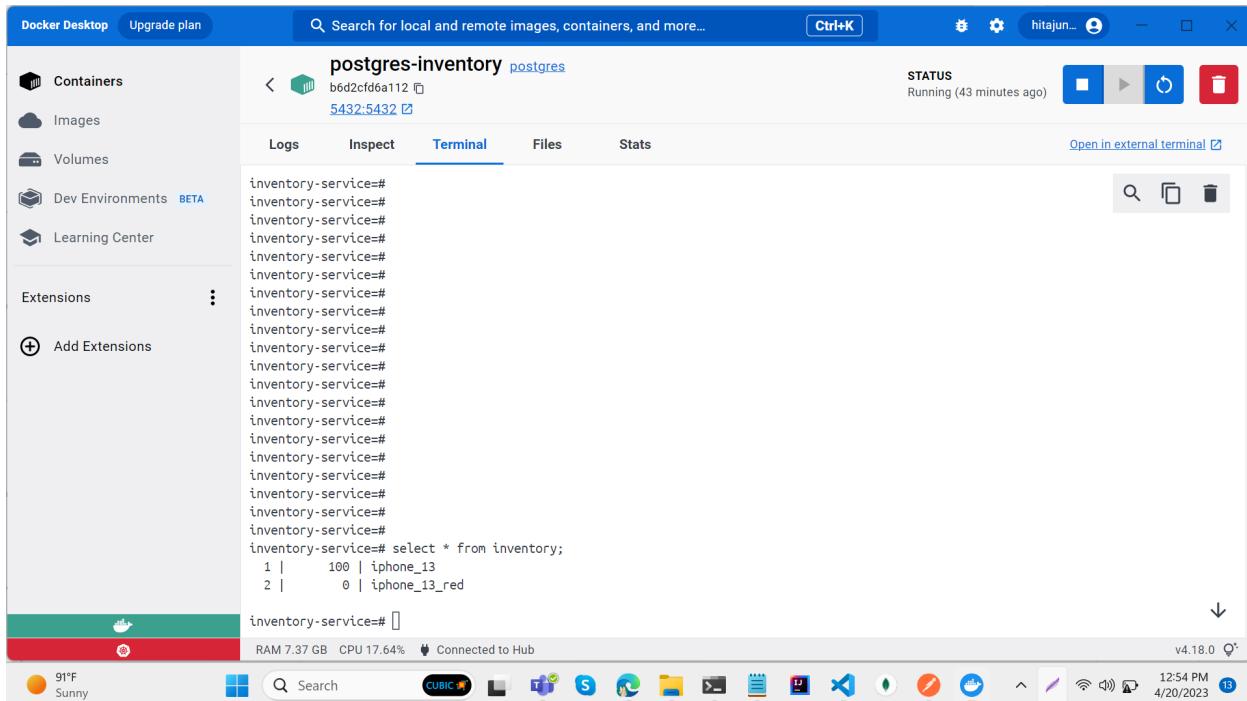
- exception: none
- http.url: /api/product
- method: GET
- outcome: SUCCESS
- status: 200
- uri: UNKNOWN

The bottom of the screen shows a taskbar with various icons and system status.

8. OrderPlaced event received



9. Inventory-service database



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

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
```

```
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:
```

```
- "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
```

```
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
```

```
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
```

```
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
```

```
- 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
```

```
- 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:
```

```
- 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
```

```
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
```

Docker containers running(16/16)

```
PS C:\Users\seema> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
e52a417141a1 grafana/grafana-oss:8.5.2 "/run.sh" 7 minutes ago Up 30 seconds 0.0.0.0:3080->3080/tcp grafana
47dbf1c2b4b3 prom/prometheus:v2.37.1 "/bin/prometheus --c..." 7 minutes ago Up 37 seconds 0.0.0.0:9090->9090/tcp prometheus
96618a8aef5d99 hitajuneja/order-service "java -cp @/app/jib_..." 7 minutes ago Up 43 seconds
3cbaf2c55761 hitajuneja/inventory-service "java -cp @/app/jib_..." 7 minutes ago Up 43 seconds
7afe2c47b3d3 hitajuneja/notification-service "java -cp @/app/jib_..." 7 minutes ago Up 44 seconds notification-service
5acd74daab39 hitajuneja/product-service "java -cp @/app/jib_..." 7 minutes ago Up 43 seconds product-service
fff8a8f51c18 hitajuneja/api-gateway "java -cp @/app/jib_..." 7 minutes ago Up 49 seconds api-gateway
ad1a7fc72752 quay.io/keycloak/keycloak:18.0.0 "/opt/keycloak/bin/k..." 7 minutes ago Up 55 seconds keycloak
886d2d4a9f91 hitajuneja/discovery-server "java -cp @/app/jib_..." 7 minutes ago Up 58 seconds discovery-server
41f1f223223d mongo:4.4.14-rc0-focal "docker-entrypoint.s..." 7 minutes ago Up 7 minutes mongo
aee32d1ba823 postgres "docker-entrypoint.s..." 7 minutes ago Up 5 minutes postgres
d07ccdf3f798 postgres "docker-entrypoint.s..." 7 minutes ago Up 7 minutes postgres-order
93ea14f19e9e openzipkin/zipkin "start-zipkin" 7 minutes ago Up 5 minutes (healthy) zipkin
51316539c900 confluentinc/cp-kafka:7.0.1 "/etc/confluent/dock..." 3 hours ago Up 3 hours broker
e246cf91c5c confluentinc/cp-zookeeper:7.0.1 "/etc/confluent/dock..." 3 hours ago Up 3 hours zookeeper
PS C:\Users\seema>
```

Microservice running on docker desktop

Docker Desktop Upgrade plan

Containers Give feedback

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	minikube	gcr.io/k8s-minikube/kicb	Running	50971:22 ↗ Show all ports (5)	4 hours ago	⋮
<input type="checkbox"/>	spring-boot-microservice	-	Running (16/16)		42 minutes ago	⋮

Showing 2 items

RAM 7.38 GB CPU 10.57% Connected to Hub v4.18.0

Docker Images Pushed to Hub

The screenshot shows the Docker Hub interface. At the top, there's a search bar with 'hitajuneja' and a dropdown menu. Below the search bar, there are tabs for 'Explore', 'Repositories', 'Organizations', and 'Help'. On the right, there's an 'Upgrade' button and a profile icon for 'hitajuneja'. The main area displays a list of repositories:

- hitajuneja / notification-service (Inactive, 0 stars, 2 downloads, Public)
- hitajuneja / api-gateway (Inactive, 0 stars, 12 downloads, Public)
- hitajuneja / discovery-server (Inactive, 0 stars, 3 downloads, Public)
- hitajuneja / inventory-service (Inactive, 0 stars, 5 downloads, Public)
- hitajuneja / order-service (Inactive, 0 stars, 12 downloads, Public)
- hitajuneja / product-service (Inactive, 0 stars, 5 downloads, Public)
- hitajuneja / pes1ug20cs645 (Inactive, 0 stars, 4 downloads, Public)

To the right of the repository list, there's a sidebar with a 'Create an Organization' button and a 'Community All-Hands' section featuring a cartoon illustration of a stage.

Task 4: Orchestrate Microservices using Kubernetes

"Kubectl cluster-info"

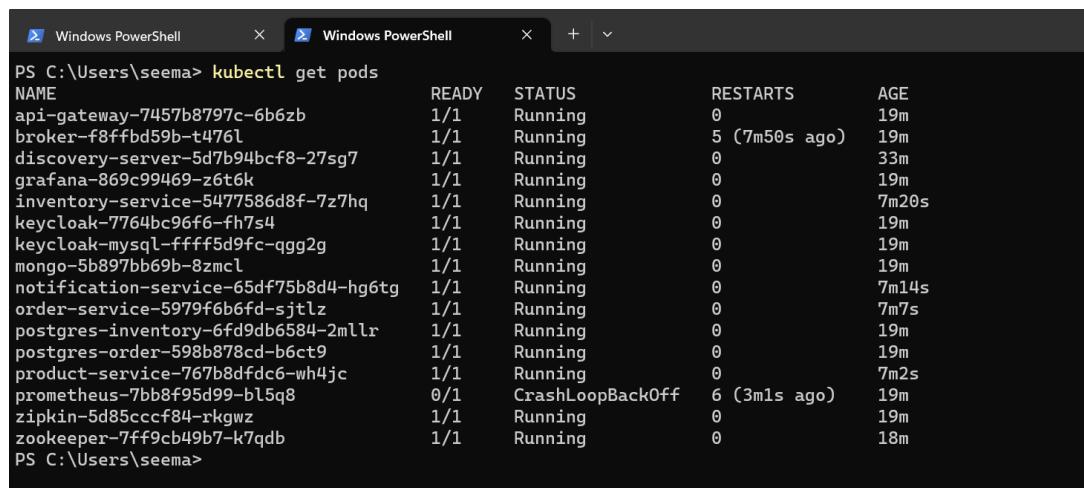
This command provides an overview of the Kubernetes cluster's endpoint and the Kubernetes API server's health status.

```
PS C:\Users\seema> kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:50975
CoreDNS is running at https://127.0.0.1:50975/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\seema>
```

"Kubectl get pods"

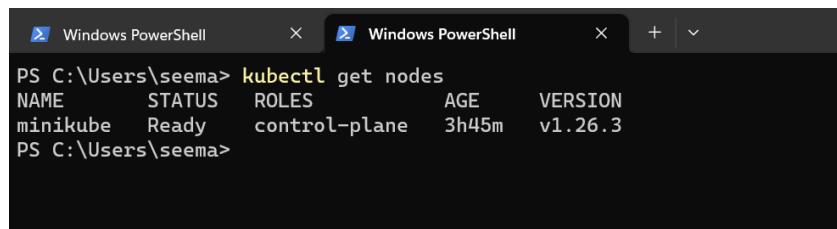
This command lists all the pods that are currently running in the cluster. A pod is the smallest deployable unit in Kubernetes, and it represents a single instance of a running process in the cluster.



```
PS C:\Users\seema> kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
api-gateway-7457b8797c-6b6zb     1/1     Running   0          19m
broker-f8ffbd59b-t476l           1/1     Running   5 (7m50s ago) 19m
discovery-server-5d7b94bcf8-27sg7  1/1     Running   0          33m
grafana-869c99469-z6t6k         1/1     Running   0          19m
inventory-service-5477586d8f-7z7hq 1/1     Running   0          7m20s
keycloak-7764bc96f6-fh7s4       1/1     Running   0          19m
keycloak-mysql-fffff5d9fc-qgg2g  1/1     Running   0          19m
mongo-5b897bb69b-8zmcl          1/1     Running   0          19m
notification-service-65df75b8d4-hg6tg 1/1     Running   0          7m14s
order-service-5979f6b6fd-sjtlz    1/1     Running   0          7m7s
postgres-inventory-6fd9db6584-2mllr 1/1     Running   0          19m
postgres-order-598b878cd-b6ct9   1/1     Running   0          19m
product-service-767b8dfdc6-wh4jc  1/1     Running   0          7m2s
prometheus-7bb8f95d99-bl5q8      0/1     CrashLoopBackOff 6 (3m1s ago) 19m
zipkin-5d85cccf84-rkgwz         1/1     Running   0          19m
zookeeper-7ff9cb49b7-k7qdb      1/1     Running   0          18m
PS C:\Users\seema>
```

"Kubectl get nodes"

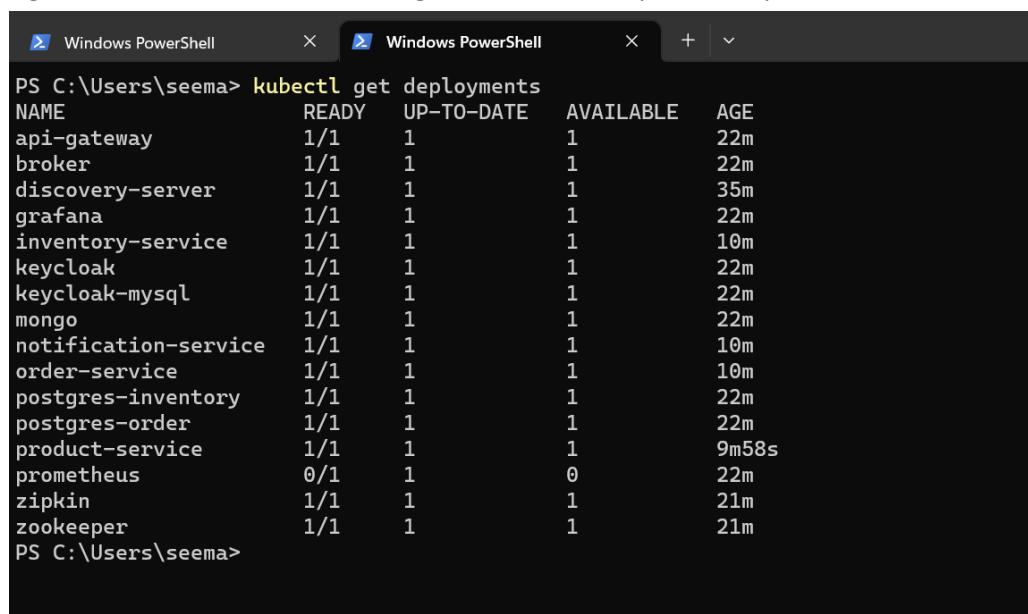
This command lists all the nodes that are currently part of the cluster. A node is a physical or virtual machine that runs Kubernetes and is responsible for running containers.



```
PS C:\Users\seema> kubectl get nodes
NAME     STATUS   ROLES      AGE     VERSION
minikube  Ready    control-plane  3h45m   v1.26.3
PS C:\Users\seema>
```

"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.



```
PS C:\Users\seema> kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
api-gateway 1/1     1           1           22m
broker       1/1     1           1           22m
discovery-server 1/1     1           1           35m
grafana      1/1     1           1           22m
inventory-service 1/1     1           1           10m
keycloak     1/1     1           1           22m
keycloak-mysql 1/1     1           1           22m
mongo        1/1     1           1           22m
notification-service 1/1     1           1           10m
order-service 1/1     1           1           10m
postgres-inventory 1/1     1           1           22m
postgres-order 1/1     1           1           22m
product-service 1/1     1           1           9m58s
prometheus    0/1     1           0           22m
zipkin        1/1     1           1           21m
zookeeper    1/1     1           1           21m
PS C:\Users\seema>
```

“Kubectl get svc”

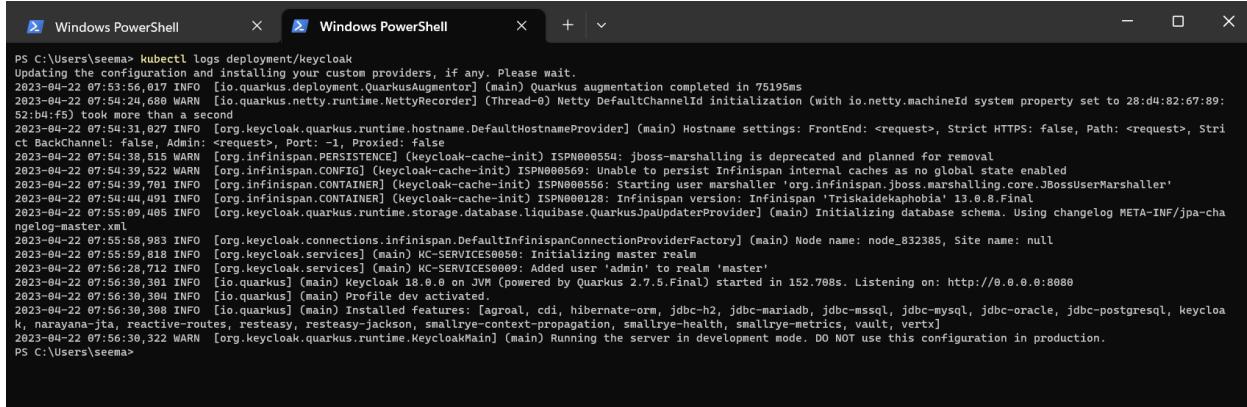
This command lists all the services that are currently running in the cluster. A service is an abstraction that provides a stable IP address and DNS name for accessing one or more pods that provide the same functionality.

```
PS C:\Users\seema> kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP  PORT(S)        AGE
api-gateway    LoadBalancer  10.103.155.49 <pending>  8181:30188/TCP  18m
broker         ClusterIP    10.97.178.209 <none>       9092/TCP      18m
discovery-server ClusterIP  10.105.182.82 <none>       8761/TCP      32m
grafana        ClusterIP    10.96.205.159 <none>       3000/TCP      18m
inventory-service ClusterIP  10.96.170.40 <none>       80/TCP        6m24s
keycloak        ClusterIP    10.109.27.122 <none>       8080/TCP      18m
kubernetes      ClusterIP    10.96.0.1     <none>       443/TCP       3h42m
mongo           ClusterIP    10.100.46.104 <none>       27017/TCP     18m
notification-service ClusterIP  10.107.43.125 <none>       80/TCP        6m18s
order-service    ClusterIP    10.100.104.163 <none>       80/TCP        6m11s
postgres-inventory ClusterIP  10.107.184.218 <none>       5432/TCP      18m
postgres-order   ClusterIP    10.101.206.162 <none>       5431/TCP      18m
product-service  ClusterIP    10.109.233.224 <none>       80/TCP        6m6s
prometheus       ClusterIP    10.110.55.125 <none>       9090/TCP      18m
zipkin           ClusterIP    10.110.43.195 <none>       9411/TCP      18m
zookeeper        ClusterIP    10.109.42.231 <none>       2181/TCP      18m
PS C:\Users\seema>
```

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

1. Logs of Zipkin service

2. Logs of keycloak service



The screenshot shows two adjacent Windows PowerShell windows. Both windows are running the command `kubectl logs deployment/keycloak`. The logs output detailed information about the startup of the Keycloak service, including configuration updates, Quarkus augmentation completion, and various INFO and WARN messages related to Infinispan and JBoss marshalling. The logs also mention the start of the Keycloak server on port 8080.

```
PS C:\Users\seema> kubectl logs deployment/keycloak
Updating the configuration and installing your custom providers, if any. Please wait.
2023-04-22 07:53:56.017 INFO [io.quarkus.deployment.QuarkusAugmentor] (main) Quarkus augmentation completed in 75195ms
2023-04-22 07:54:24.680 WARN [io.quarkus.netty.runtime.NettyRecorder] (Thread-0) Netty DefaultChannelId initialization (with io.netty.machineId system property set to 28:d4:82:67:89:52:b4:f5) took more than a second
2023-04-22 07:54:31.927 INFO [org.keycloak.quarkus.runtime.hostname.DefaultHostnameProvider] (main) Hostname settings: FrontEnd: <request>, Strict HTTPS: false, Path: <request>, Strict BackChannel: false, Admin: <request>, Port: -1, Proxied: false
2023-04-22 07:54:38.515 WARN [org.infinispan.PERSISTENCE] (Keycloak-cache-init) ISPN000554: jboss-marshalling is deprecated and planned for removal
2023-04-22 07:54:39.522 WARN [org.infinispan.CONFIG] (Keycloak-cache-init) ISPN000569: Unable to persist Infinispan internal caches as no global state enabled
2023-04-22 07:54:39.781 INFO [org.infinispan.CONTAINER] (Keycloak-cache-init) ISPN000556: Starting user marshaller 'org.infinispan.jboss.marshalling.core.JBossUserMarshaller'
2023-04-22 07:54:40.491 INFO [org.infinispan.CONTAINER] (Keycloak-cache-init) ISPN000128: Infinispan version: Infinispan 'Triskalidekaphobia' 13.0.8.Final
2023-04-22 07:55:09.485 INFO [org.keycloak.quarkus.runtime.storage.database.liquibase.QuarkusJpaUpdaterProvider] (main) Initializing database schema. Using changelog META-INF/jpa-changelog-master.xml
2023-04-22 07:55:09.983 INFO [org.keycloak.connections.infinispan.DefaultInfinispanConnectionProviderFactory] (main) Node name: node_832385, Site name: null
2023-04-22 07:55:59.818 INFO [org.keycloak.services] (main) KC-SERVICE0890: Initializing master realm
2023-04-22 07:56:28.712 INFO [org.keycloak.services] (main) KC-SERVICE0809: Added user 'admin' to realm 'master'
2023-04-22 07:56:30.301 INFO [io.quarkus] (main) Keycloak 18.0.0 on JVM (powered by Quarkus 2.7.5.Final) started in 152.798s. Listening on: http://0.0.0.0:8080
2023-04-22 07:56:30.304 INFO [io.quarkus] (main) Profile dev activated.
2023-04-22 07:56:30.308 INFO [io.quarkus] (main) Installed features: [lagro, cdi, hibernate-orm, jdbc-h2, jdbc-mariadb, jdbc-msql, jdbc-mysql, jdbc-oracle, jdbc-postgresql, keycloak, narayana-jta, reactive-routes, resteasy, resteasy-jackson, smallrye-context-propagation, smallrye-health, smallrye-metrics, vault, vertx]
2023-04-22 07:56:30.322 WARN [org.keycloak.quarkus.runtime.KeycloakMain] (main) Running the server in development mode. DO NOT use this configuration in production.
PS C:\Users\seema>
```

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.

Jenkins Job: cc-pipeline

The screenshot shows the Jenkins Pipeline cc-pipeline stage view. It displays two builds, #24 and #23, with their respective stage details and metrics.

Build	Stage	Time
#24	Checkout	6s
#24	Build and Test	758ms
#24	Build Docker Images	18s
#24	Deploy to Kubernetes	3s
#23	Checkout	9s
#23	Build and Test	1s
#23	Build Docker Images	16s
#23	Deploy to Kubernetes	4s
#24	Build and Test	~35s
#23	Build and Test	~35s

Build History (trend ▾):

- Build #24: Apr 23 01:48 (1 commit)
- Build #23: Apr 23 01:41 (No Changes)

Metrics:

- Average stage times: (Average full run time: ~35s)
- Checkout: 6s, 9s
- Build and Test: 758ms, 1s, 20s
- Build Docker Images: 18s, 16s
- Deploy to Kubernetes: 3s, 4s, 6s

Pipeline Configuration:

The screenshot shows a web browser window with multiple tabs open, including 'hit03/e...', 'e-comm...', 'hitajunej...', 'cc-pipeline...', 'build trig...', 'Build trig...', 'Docker...', 'kubectl...', 'Stack Ov...', and '+'. The main content area is titled 'Configure' under 'Dashboard > cc-pipeline > Configuration'. On the left, there are navigation links for 'General', 'Advanced Project Options' (which is currently selected), and 'Pipeline'. The 'Pipeline' section has a 'Definition' tab selected, showing a 'Pipeline script' input field containing Groovy code for a Jenkins pipeline. The code defines a pipeline with stages for 'Checkout' and 'Build and Test'. At the bottom, there are 'Save' and 'Apply' buttons.

```
1* pipeline {  
2    agent any  
3  
4    stages {  
5        stage('Checkout') {  
6            steps {  
7                checkout([$class: 'GitSCM', branches: [[name: '*/dev']], doGenerateSubmoduleConfigurations: false, ext  
8            }  
9        }  
10       stage('Build and Test') {  
11           steps {  
12               dir('product-service') {  
13                   bat 'start mvn clean package'  
14               }  
15               dir('inventory-service') {  
16                   bat 'start mvn clean package'  
17               }  
18           }  
19       }  
20   }  
21 }
```

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'
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
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.

The screenshot shows a GitHub repository named **hita03 / ecommerce-microservices-CC**. The repository is private. The main page displays the following information:

- Code** tab is selected.
- Issues**, **Pull requests**, **Actions**, **Projects**, and **Security** tabs are also present.
- main** branch is selected.
- 2 branches** and **0 tags** are listed.
- A modal window titled **Switch branches/tags** is open, showing the current selection of **03/dev**.
- Branches** tab is selected, showing **main** (checked) and **dev**.
- Tags** tab is also present.
- Remaining services** listed include **discovery-service** and **committed**.
- Commits** section shows commits from April 17, 2023, and April 13, 2023, and April 7, 2023.
- Each commit includes details like author, date, commit hash, and copy/paste options.

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

THANK YOU!

