CURRENT TRENDS IN SOFTWARE ENGINEERING

SE – 4010

MICROSERVICE ASSIGNMENT REPORT

Lasal Sandeepa Hettiarachchi

IT19132310

B.Sc. (Hons) in Information Technology Specializing in Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

Group ID:

TABLE OF CONTENTS

List Of Figures

1. INTRODUCTION

The following document is the implementation walkthrough of an e-commerce system developed and designed using modern dev-ops concepts using the microservice architecture. In this document, the design and development specifications with respect to the developed system will be discussed and the DevOps concepts will be elaborated under each task specified in the assignment document.

2. PROJECT OVERVIEW

The system developed is a solution for an eCommerce system to automate the end-to-end process of browsing the system for items to payment and delivery of the ordered item. The services that are provided in the system are as follows

Services

Name	Assignee	Language
User service	Lasal	Node JS
Product service	Senura	Java, Sprint Boot
Order service	Rusiru	Go
Cart service	Senura	Java, Sprint Boot
Payment service	Dilmi	Java, Sprint Boot
Frontend service	Shared	JavaScript, React
Auth service	Lasal	Node JS
Delivery service	Dilmi	Java, Sprint Boot
Email service	Rusiru	Node JS

Figure 0-1:service overview

The system comprises of 8 microservices that are communicating with one another to automate the eCommerce system. The services include a user service (responsible for handling user-related tasks in the system), a product service (responsible for handling product-related tasks), an order service (responsible for creating orders and managing tasks related to orders), a cart service (responsible for cart-related business logic), a payment service(responsible for payment logic), an auth service (responsible for authorizing users to access endpoints), a delivery service(responsible for delivery functions) and an email service (for sending emails to users)
All these services are developed using a multitude of languages ranging from Go, and JavaScript to Java and developed using the microservice architecture.

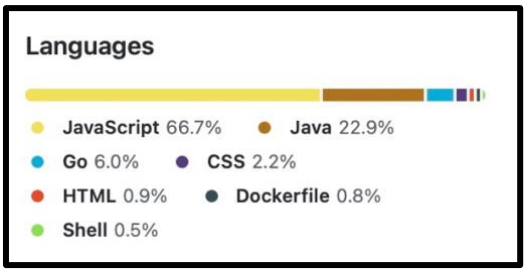


Figure 0-2:languages used for microservice project

The individual microservices communicate with one another using APIs to achieve the flow of the microservice system.

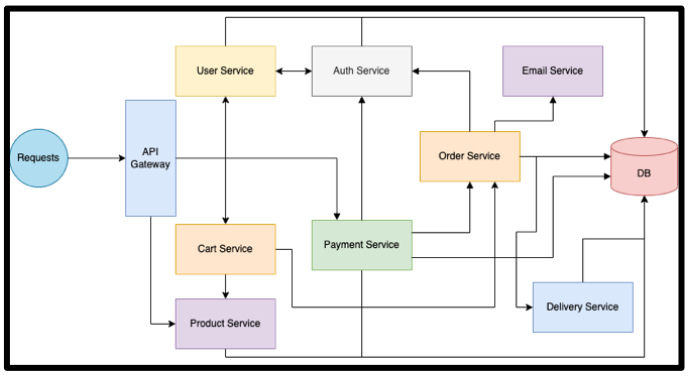


Figure 0-3:microservice system overview

3. GITHUB CONTRIBUTIONS

The project was developed collaboratively using version management tools such as git and repositories such as GitHub. In GitHub separate environments were maintained as development, staging, and master(production), and code was merged to the master branch using pull requests from other branches.
The main branch was access protected and never was allowed to commit directly.

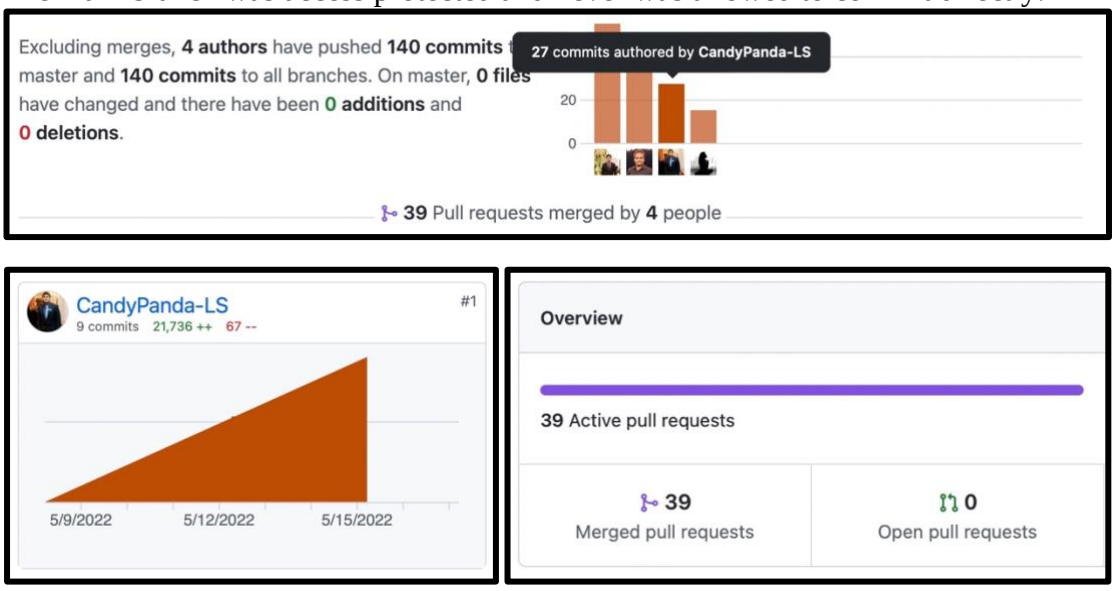


Figure 0-4:Github Contribution

GitHub Link: https://github.com/Research-Group-CDAP/CTSE-Assignment-2.git

4. PROJECT TECHNOLOGIES

The development of the microservices were done using a multitude of languages. The development was done with the help of tools such as git and GitHub. The developed microservices were containerized using Docker and Azure Kubernetes Service(AKS) was used as the container management system.

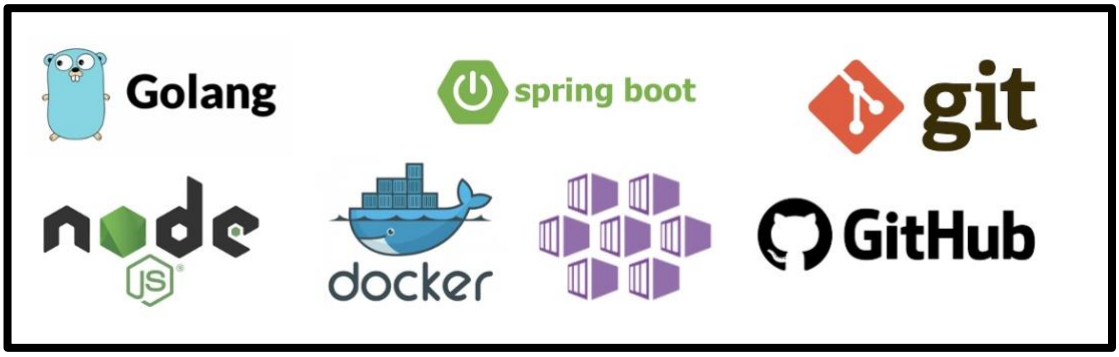


Figure 0-5:Tools & Technologies used in the project

Docker hub was used instead of Azure container registry to host the created docker images while the creation of docker images was manually done inside a VM machine environment in the manual process before automating.

5. SERVICE OVERVIEW(INDIVIDUAL)

5.1 User Service

The user service is responsible for handling basic tasks related to user management in the system. In the hypothesized scenario it was taken that the system mainly has 3 types of users.

- Seller
- Buyer
- Admin

The sellers in the system are responsible for mainly posting items in the system while buyers are responsible for browsing/buying items while the admin is responsible for managing the general system. Role-based authorization is done using JSON web tokens to authorize specific users to access certain services within the microservice.

5.1.1 Endpoints

In this section, the endpoints related to the user service will be discussed.

```
import express from "express"
import {login,register,updateUser,deleteUser,getUserDetailsbyID,getUserList,getUserDetailsbytoken} from "../controller/user.controller.js"
const userRoutes = express.Router();

userRoutes.post("/login",  login);
userRoutes.post("/register",  register);
userRoutes.put("/update/:id",  updateUser);
userRoutes.delete("/delete/:id",  deleteUser);
userRoutes.get("/user_token/getdetails",  getUserDetailsbytoken);
userRoutes.get("/:id",  getUserDetailsbyID);
userRoutes.get("/",  getUserList);

export default userRoutes;
```

Figure 0-6:User service endpoints

- <baseURL>/api/user/login

This endpoint authenticates to user using an email PW combination and if authenticated returns an authtoken created using JWT (also sends it to the auth service to be updated). If the email PW combination is incorrect an error will be returned

- <baseURL>/api/user/register

This endpoint lets the users to register to the system by providing the necessary details. Once registered the auth token associated with the created user will be sent as the response

- <baseURL>/api/user/update/:id

This endpoint allows the user to update the details by providing the id

- <baseURL>/api/user/delete/:id

This endpoints allows the admin to delete users from the system by providing an id

- <baseURL>/api/user/user_token/getdetails

This endpoints allows to get the details related to a user by providing their user authToken

- <baseURL>/api/user/:id

This endpoint allows the admin to get details of a certain user by providing their id

- <baseURL>/api/user/

This endpoint allows the admin to get the details of all the users in the system.

5.1.2 Model

In this section, the model related to the user is discussed

```
const UserSchema = new Schema({
    first_name: {
        type: String,
    },
    last_name: {
        type: String,
    },
    email: {
        type: String,
        unique: true,
    },
    mobileNumber: {
        type: String,
    },
    address: {
        type: String,
    },
    password: {
        type: String,
    },
    authToken: {
        type: String,
    },
    role: {
        type: String,
    },
    createdAt: {
        type: Date,
    },
    updatedAt: {
        type: Date,
    }

});
UserSchema.methods.generateAuthToken = async function () {
    const user = this;
    const secret = process.env.JWT_SECRET;
    const authToken = jwt.sign({ _id: user._id , role: user.role , email:user.email}, secret);
    user.authToken = authToken;
    await user.save();
    return authToken;
};
```

Figure 0-7:User service model class

The model has general attributes such as name, email password etc. Along with that, the model has attributes to store createdAt time and updatedAt time. The createdAt will timestamp the object creation time while updatedAt will timestamp whenever the object is updated.
GenerateAuthToken method is written to generate an auth token whenever logged or registered using the user's information.

5.2 Auth Service

The auth service is responsible for handling basic tasks related to the authorization of users in the system. In the hypothesized scenario it was taken that the system mainly has 3 types of users, and they are granted different levels of privileges based on their roles.

5.2.1 Endpoints

```
import express from "express"
import {addAuthConfig,authorize,authorizeSeller,authorizeBuyer,authorizeAdmin} from "../controller/auth.controller.js"
const authRoutes = express.Router();


authRoutes.get("/authorize",  authorize);
authRoutes.get("/authorizeSeller",  authorizeSeller);
authRoutes.get("/authorizeBuyer",  authorizeBuyer);
authRoutes.get("/authorizeAdmin",  authorizeAdmin);
authRoutes.post("/registerAuth",  addAuthConfig);



export default authRoutes;
```

Figure 0-8:Auth service endpoints

- <baseURL>/api/auth/authorizeSeller

This endpoint takes the auth header and checks whether the bearer token corresponding to the users has the seller privilege. The privileges are defined as an array using an Enum.

```
const privillages = [Role.Seller,Role.Admin]
```

the seller privilege is given to the seller role and the admin role

- <baseURL>/api/auth/authorizeBuyer

This endpoint takes the auth header and checks whether the bearer token corresponding to the users has the buyer privilege by decoding the provided JWT token. The privileges are defined as an array using an Enum.

```
const privillages = [Role.Seller,Role.Admin,Role.Buyer]
```

The buyer privilege is given to all users in the system

- <baseURL>/api/auth/authorizeAdmin

This endpoint takes the auth header and checks whether the bearer token corresponding to the users has the admin privilege by decoding the provided JWT token. The privileges are defined as an array using an Enum.

```
const privillages = [Role.Admin]
```

Only the admin role is given the root privillages in the system.

- <baseURL>/api/auth/registerAuth

This endpoint takes in user information and authToken and registers it in the system so that further comparisons can be made to grant authorizations.

5.2.2 Model

```
import mongoose from "mongoose";
const Schema = mongoose.Schema;

const AuthSchema = new Schema({
    first_name: {
        type: String,
    },
    email: {
        type: String,
    },
    password: {
        type: String,
    },
    authToken: {
        type: String,
    },
    role: {
        type: String,
    }
});

const Auth = mongoose.model("Auth", AuthSchema);
export default Auth;
```

Figure 0-9:Auth service model

The auth model contains the auth-specific information of the abstracting that out from the user model and the user-specific information. Further, this helps to grant the necessary authorizations to the user when the endpoints are called.

6. TASK 01 – CONTAINERIZE APPLICATION (INDIVIDUAL)

Docker allows the applications to be separated from the infrastructure that it is run upon making it unified to that the ability to package and run applications in a loosely isolated environment can be achieved using containerization concepts. When creating the docker images in the computer, docker pulls a base image (in this case the node JS alpine version since it is lightweight) and creates the service images according to the specifications in the Dockerfile. In both the Auth service and the User service, multi-stage building is used to reduce the size of the docker image.

6.1 Containerize Auth Service

6.1.1 Dockerfile of the Auth Service

```
FROM node:16-alpine3.14 AS BUILD_IMAGE
RUN apk add --no-cache nodejs npm
WORKDIR /auth_service
# COPY ["package.json", "./"]
COPY ["package.json", "./"]
RUN npm install
COPY . .
FROM node:14.18-alpine
WORKDIR /app
COPY --from=BUILD_IMAGE /auth_service /app/
EXPOSE 5002
ENTRYPOINT [ "npm", "run" ]
CMD [ "start" ]
```

6.1.2 Auth Service container image building

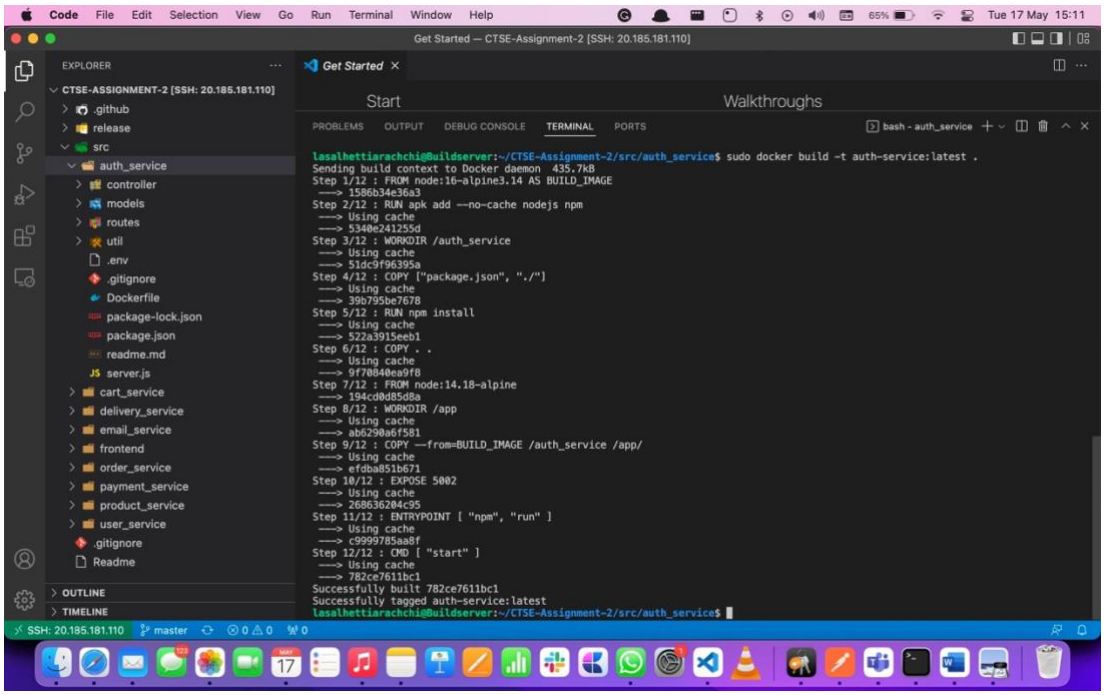


Figure 0-10:Auth service image building

6.1.3 Running auth service docker image

```
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/auth_service$ sudo docker run -p 5002:5002 auth-service:
latest

> auth_service@1.0.0 start /app
> node server.js

Server started on port 5002
connected to MongoDb
```

Figure 0-11:Running instance of the auth service docker image

6.1.4 Push Auth Service docker image to docker hub

```
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/auth_service$ sudo docker login -u lasalhettiarachchi
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/auth_service$ sudo docker tag auth-service:latest lasalh
ettiarachchi/auth-service:latest
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/auth_service$ sudo docker push lasalhettiarachchi/auth-s
ervice:latest
The push refers to repository [docker.io/lasalhettiarachchi/auth-service]
9857fc526dae: Pushed
0d7fd63fc805: Layer already exists
e0003e245f2e: Layer already exists
70b4a4ce362b: Layer already exists
b3520dc1ea2d: Layer already exists
8d3ac3489996: Layer already exists
latest: digest: sha256:410ef42a56a4c0d881a5ca63565e83787d5d41fa5bb7924dc6668c277bb7615a size: 1575
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/auth_service$
```

Figure 0-12:pushing the docker image to docker hub

6.1.5 Auth Service docker hub overview

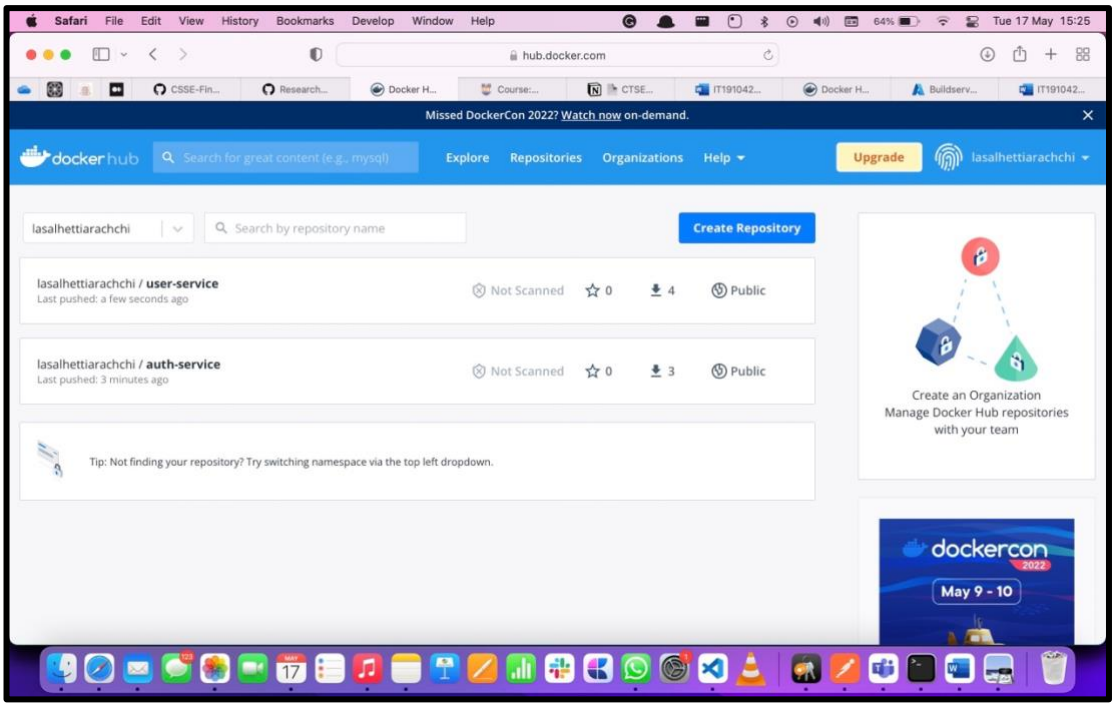


Figure 0-13:Auth service Image in docker hub

Dockerhub link : https://hub.docker.com/r/lasalhettiarachchi/auth-service

6.2 Containerize User Service

6.2.1 Dockerfile of the User Service

```
FROM node:14.18-alpine AS BUILD_IMAGE
RUN apk add --no-cache nodejs npm
WORKDIR /user_service
COPY package.json package-lock.json ./
RUN npm install
COPY . .

FROM node:14.18-alpine
WORKDIR /app
COPY --from=BUILD_IMAGE /user_service /app/
EXPOSE 5001
ENTRYPOINT [ "npm", "run" ]
CMD [ "start" ]
```

6.2.2 User Service container image building

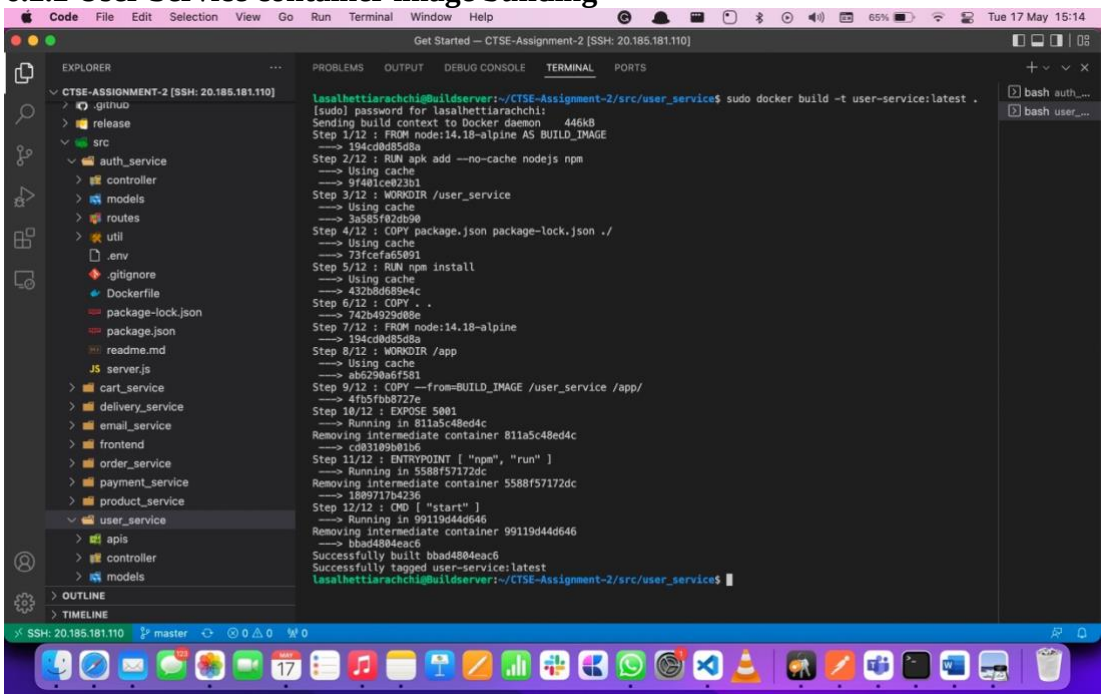


Figure 0-14:User service image building

6.2.3 Running user service docker image

```
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/user_service$ sudo docker run -p 5001:5001 user-service:
latest

> user_service@1.0.0 start /app
> node server.js

Server started on port 5001
connected to MongoDb
```

Figure 0-15:Running instance of the user service image

6.2.4 Push User Service docker image to docker hub

```
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/user_service$ sudo docker login -u lasalhettiarachchi
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/user_service$ sudo docker tag user-service:latest lasalh
ettiarachchi/user-service:latest
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/user_service$ sudo docker push lasalhettiarachchi/user-s
ervice:latest
The push refers to repository [docker.io/lasalhettiarachchi/user-service]
277c66f31e73: Pushed
0d7fd63fc805: Layer already exists
e0003e245f2e: Layer already exists
70b4a4ce362b: Layer already exists
b3520dc1ea2d: Layer already exists
8d3ac3489996: Layer already exists
latest: digest: sha256:39d5e0bc20b5d56c728dc48abe9b32de61cfe0352b567f15c5b1a85331ad3f0f size: 1575
lasalhettiarachchi@Buildserver:~/CTSE-Assignment-2/src/user_service$
```

Figure 0-16User service image being pushed to docker hub

6.2.5 User Service docker hub overview

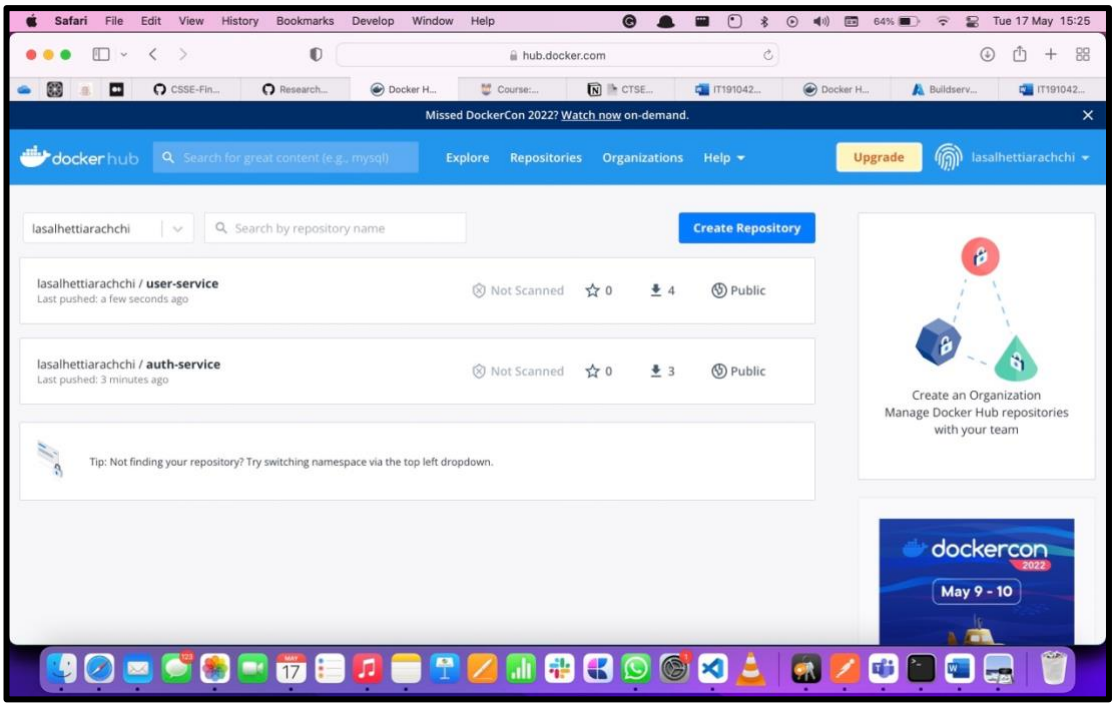


Figure 0-17:User service image in docker hub

Dockerhub link : https://hub.docker.com/r/lasalhettiarachchi/user-service

7. TASK 02 – DEPLOY SERVICES TO KUBERNETES CLUSTER (INDIVIDUAL)

Kubernetes is a container orchestration framework that can manage deployment, scaling and self-healing, and many aspects related to container deployments. It has gained in popularity in recent years due to features such as automated rollout and rollback, service discovery and load balancing, horizontal scaling, batch execution etc. This technology can be used to ease the process of container deployment and management. There are many solutions such as minikube which allows to create clusters in a local machine. There are also many cloud service providers that provides this service such as GCP, Digital ocean, Azure, AWS etc.

For this particular application AKS was used to create the Kubernetes cluster. A single node configuration was made for the cluster to deploy the microservice application.

7.1 Deploy the User service to K8 cluster

7.1.1 User Service deployment/service YAML

```
apiVersion: v1 # Kubernetes API version
kind: Service # Kubernetes resource kind we are creating
metadata: # Metadata of the resource kind we are creating
  name: user-service
spec:
  selector:
    app: userservice
  ports:
    - protocol: "TCP"
      port: 5001 # The port that the service is running on in the cluster
      targetPort: 5001 # The port exposed by the service
  type: LoadBalancer # type of the service. LoadBalancer indicates that our service will be external.
---
apiVersion: apps/v1
kind: Deployment # Kubernetes resource kind we are creating
metadata:
  name: userservice
spec:
  selector:
    matchLabels:
      app: userservice
```

```
  replicas: 2 # Number of replicas that will be created for this deployment
  template:
    metadata:
      labels:
        app: userservice
    spec:
      containers:
        - name: userservice
          image: docker.io/lasalhettiarachchi/user-service:v1.0.3 # Image that will be used to containers in
the cluster
          imagePullPolicy: Always
          ports:
            - containerPort: 5001 # The port that the container is running on in the cluster
```

7.1.2 Logging into the Kubernetes cluster through the VM

First the azure command line tools were installed to the VM to access the k8 cluster from the terminal

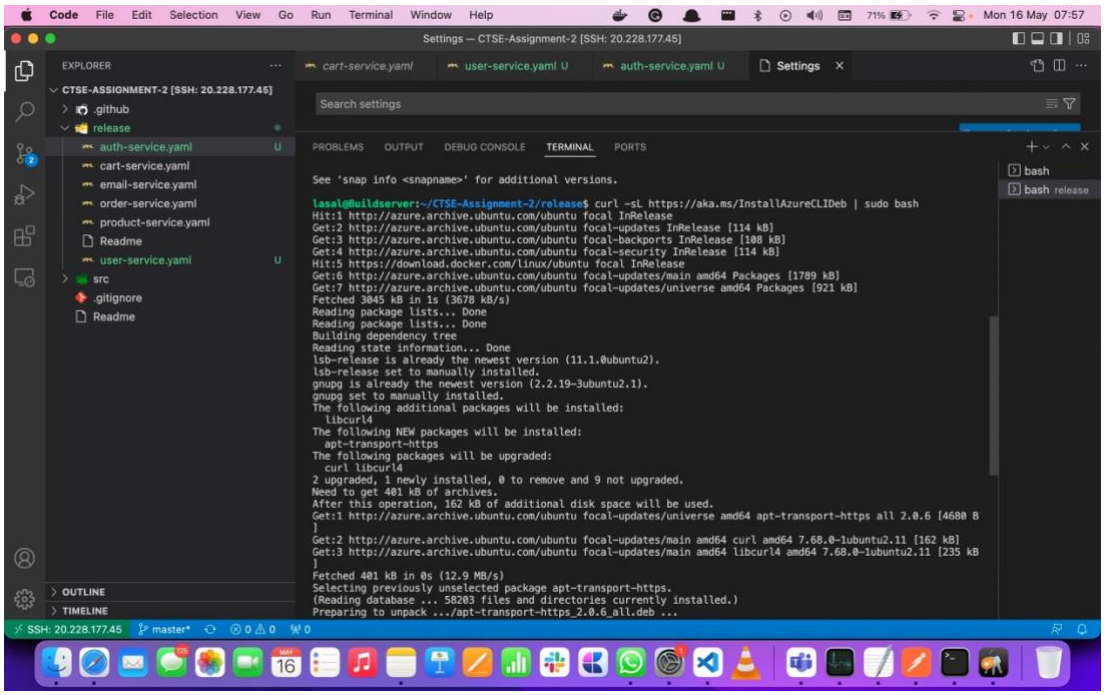


Figure 0-18:Installing az command line tools to VM

From that using ,
az login -u <username> -p <password>

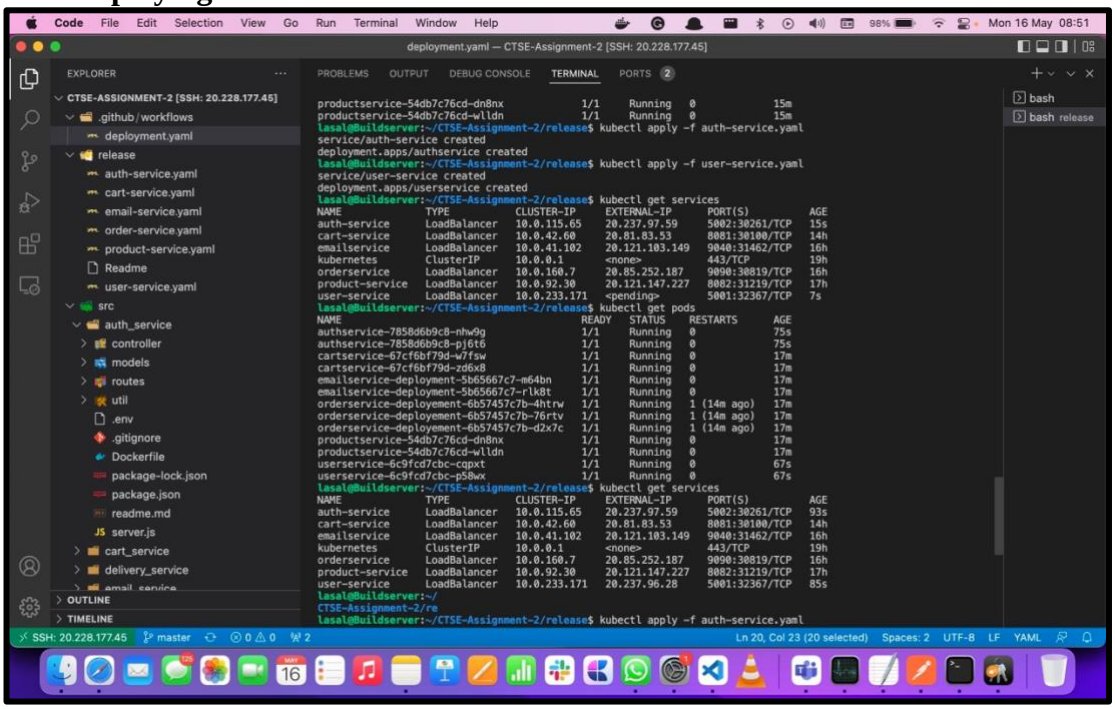
the user was logged into the azure account terminal

```
lasal@Buildserver:~/CTSE-Assignment-2/release$ az aks get-credentials --resource-group CTSE --name ctse
Merged "ctse" as current context in /home/lasal/.kube/config
lasal@Buildserver:~/CTSE-Assignment-2/release$ kubectl get pods

Command 'kubectl' not found, but can be installed with:

sudo snap install kubectl

lasal@Buildserver:~/CTSE-Assignment-2/release$ sudo snap install kubectl
error: This revision of snap "kubectl" was published using classic confinement and thus may perform
       arbitrary system changes outside of the security sandbox that snaps are usually confined to,
       which may put your system at risk.

       If you understand and want to proceed repeat the command including --classic.
lasal@Buildserver:~/CTSE-Assignment-2/release$ sudo snap install kubectl --classic
kubectl 1.24.0 from Canonical✓ installed
lasal@Buildserver:~/CTSE-Assignment-2/release$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS      AGE
cartservice-67cf6bf79d-w7fsw            1/1     Running   0             15m
cartservice-67cf6bf79d-zd6x8            1/1     Running   0             15m
emailservice-deployment-5b65667c7-m64bn 1/1     Running   0             15m
emailservice-deployment-5b65667c7-rlk8t 1/1     Running   0             15m
orderservice-deployement-6b57457c7b-4htrw 1/1   Running   1 (12m ago)   15m
orderservice-deployement-6b57457c7b-76rtv 1/1   Running   1 (12m ago)   15m
orderservice-deployement-6b57457c7b-d2x7c 1/1   Running   1 (12m ago)   15m
productservice-54db7c76cd-dn8nx         1/1     Running   0             15m
productservice-54db7c76cd-wlldn         1/1     Running   0             15m
```

Figure 0-19:Installing kubectl to the cluster

From that using
Az aks get-credentials –resource-group CTSE –name ctse
The terminal of the Kubernetes cluster was accessed. Once accessed, kuubectl was installed. The above snapshot shows the pods in the cluster before applying the user-service.yaml and auth-service.yaml.

7.1.3 Deploying the services to K8



By applying the yaml files inside the release folder, it is possible to deploy the services to the K8 cluster. As displayed in the above screenshot, it is possible to apply each yaml file individually aswell.

7.2 Deploy the Auth service to K8 cluster

7.2.1 Auth Service deployment/service YAML

```yaml
apiVersion: v1 # Kubernetes API version
kind: Service # Kubernetes resource kind we are creating
metadata: # Metadata of the resource kind we are creating
  name: auth-service
spec:
  selector:
    app: authservice
  ports:
    - protocol: "TCP"
      port: 5002 # The port that the service is running on in the cluster
      targetPort: 5002 # The port exposed by the service
  type: LoadBalancer # type of the service. LoadBalancer indicates that our service will be external.
---
apiVersion: apps/v1
kind: Deployment # Kubernetes resource kind we are creating
metadata:
  name: authservice
spec:
  selector:
    matchLabels:
      app: authservice
  replicas: 2 # Number of replicas that will be created for this deployment
  template:
    metadata:
      labels:
        app: authservice
    spec:
      containers:
        - name: authservice
```

```
        image: docker.io/lasalhettiarachchi/auth-service:v1.0.3 # Image that will be used to containers in
the cluster
        imagePullPolicy: Always
        ports:
          - containerPort: 5002 # The port that the container is running on in the cluster
        imagePullPolicy: Always
        ports:
          - containerPort: 5001 # The port that the container is running on in the cluster
```

7.2.2 Logging into the Kubernetes cluster through the VM

First the azure command line tools were installed to the VM to access the k8 cluster from the terminal

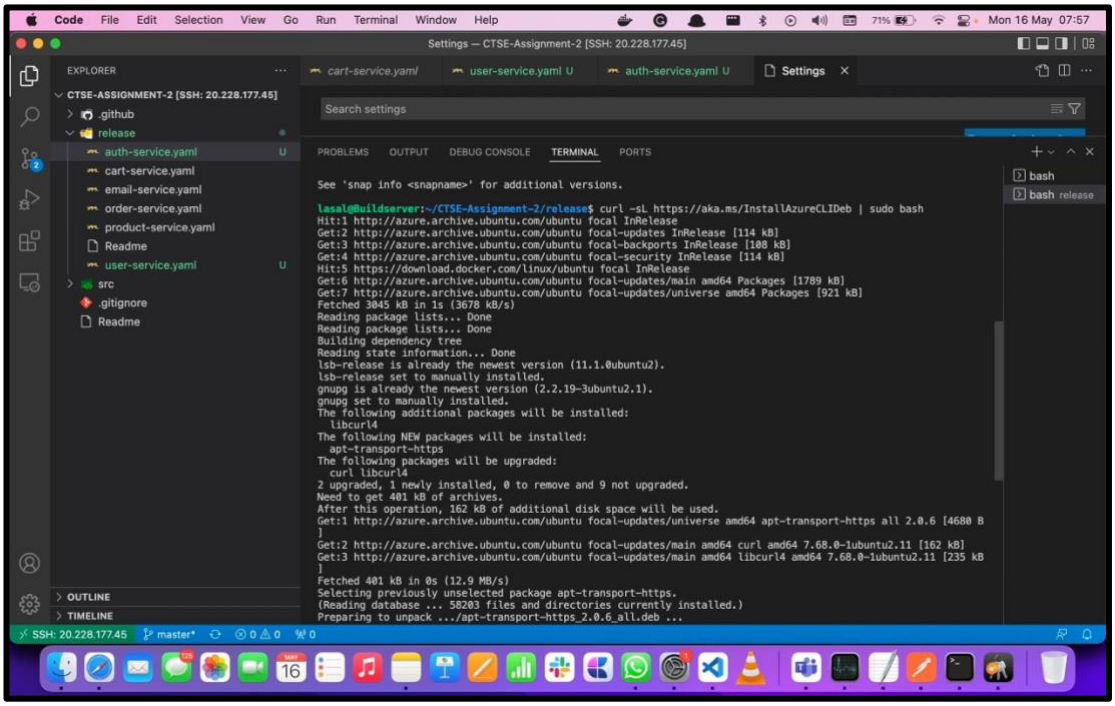


Figure 0-20:Installing az command line tools to VM

From that using ,
az login -u <username> -p <password>
the user was logged into the azure account terminal

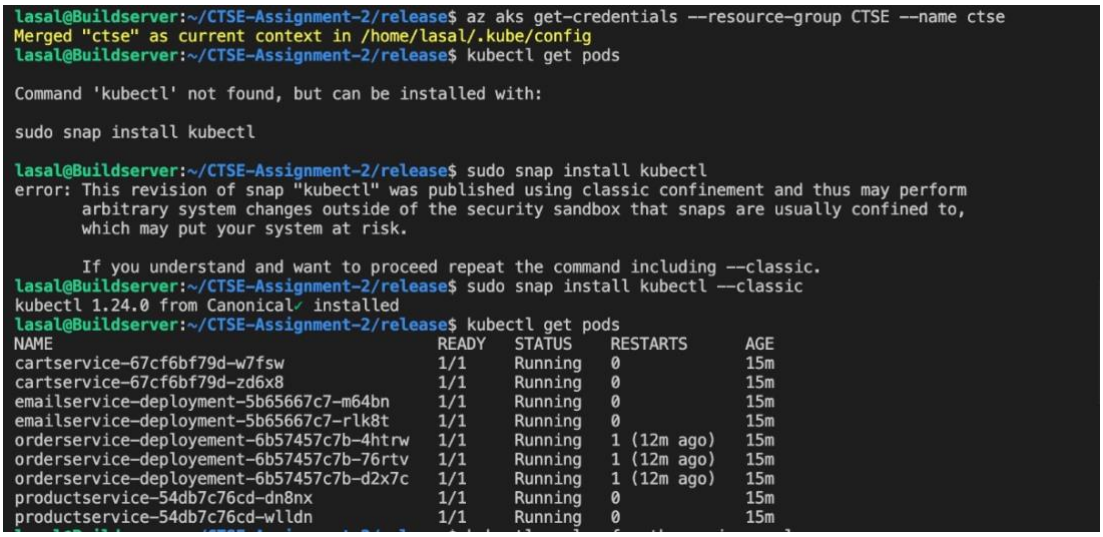
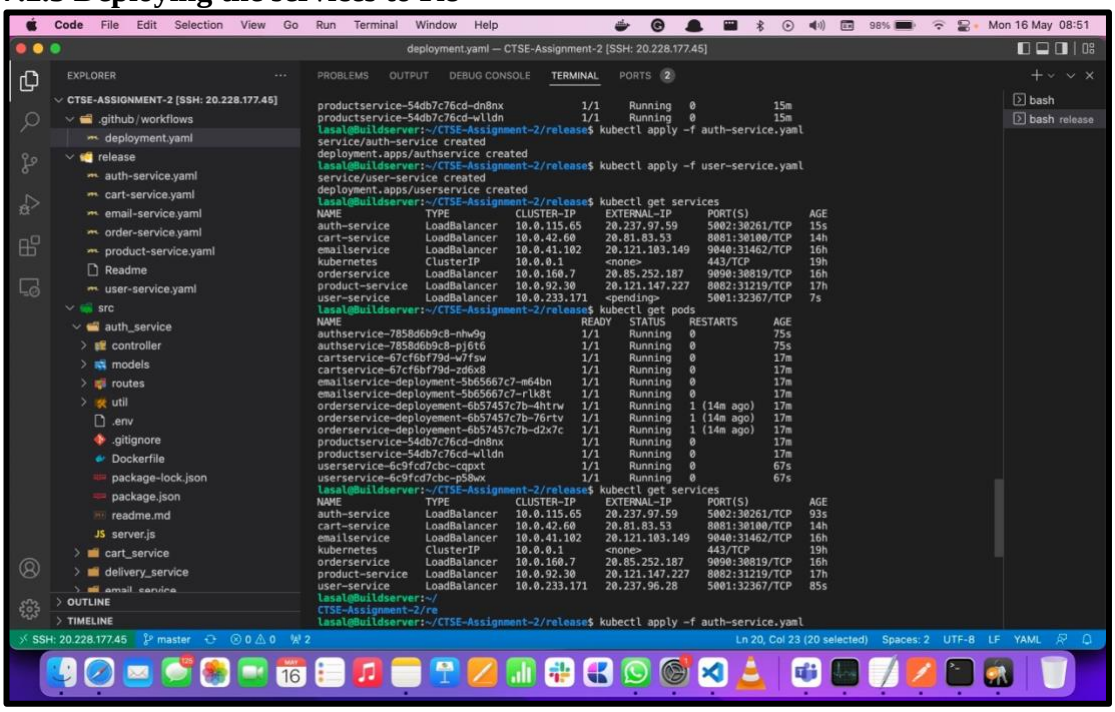


Figure 0-21:Installing kubectl to the cluster

From that using
Az aks get-credentials –resource-group CTSE –name ctse
The terminal of the Kubernetes cluster was accessed. Once accessed, kuubectl was installed. The above snapshot shows the pods in the cluster before applying the user-service.yaml and auth-service.yaml.

7.2.3 Deploying the services to K8



By applying the yaml files inside the release folder, it is possible to deploy the services to the K8 cluster. As displayed in the above screenshot, it is possible to apply each yaml file individually as well.

Although both the services above are specified as loadbalancers with external IP addresses, they should all be cluster Ips with an API gateway being the only accessible loadbalancer with an external IP. But for demonstration purposes and testing purposes, all services are kept as load balancers.

8. TASK 03 – CI/CD PIPELINE IN GITHUB ACTIONS (INDIVIDUAL)

This project uses a CI/ CD pipeline to automatically build the container images and push them to the relevant DockerHub account. After the building process and pushing process are completed for all the services, the deployment pipeline will deploy the new changes to the k8s cluster. GitHub secretes are used to store DockerHub credentials and k8s cluster credentials. Therefore, the credentials are not visible to the public.

8.1 Deployment YAML for the GitHub action

A YAML file was written to trigger a pipeline when code is pushed to the master branch. This YAML file is responsible for creating the respective docker images, pushing them to the respective docker hub accounts, and creating the Kubernetes deployment using the pushed docker images.

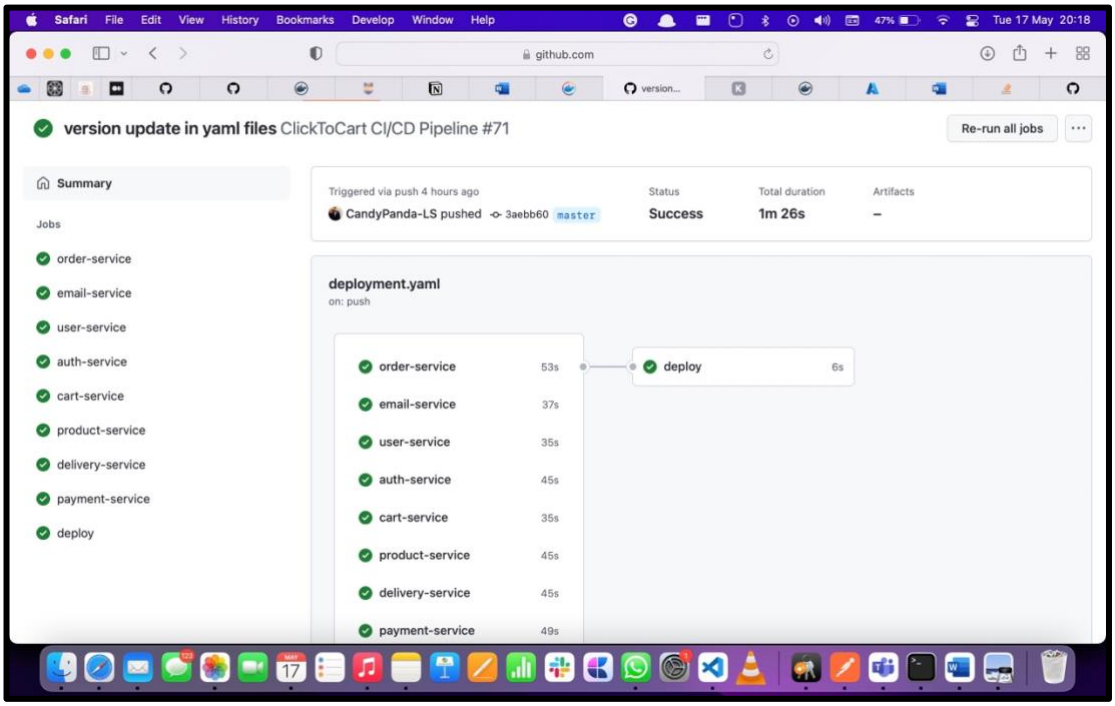
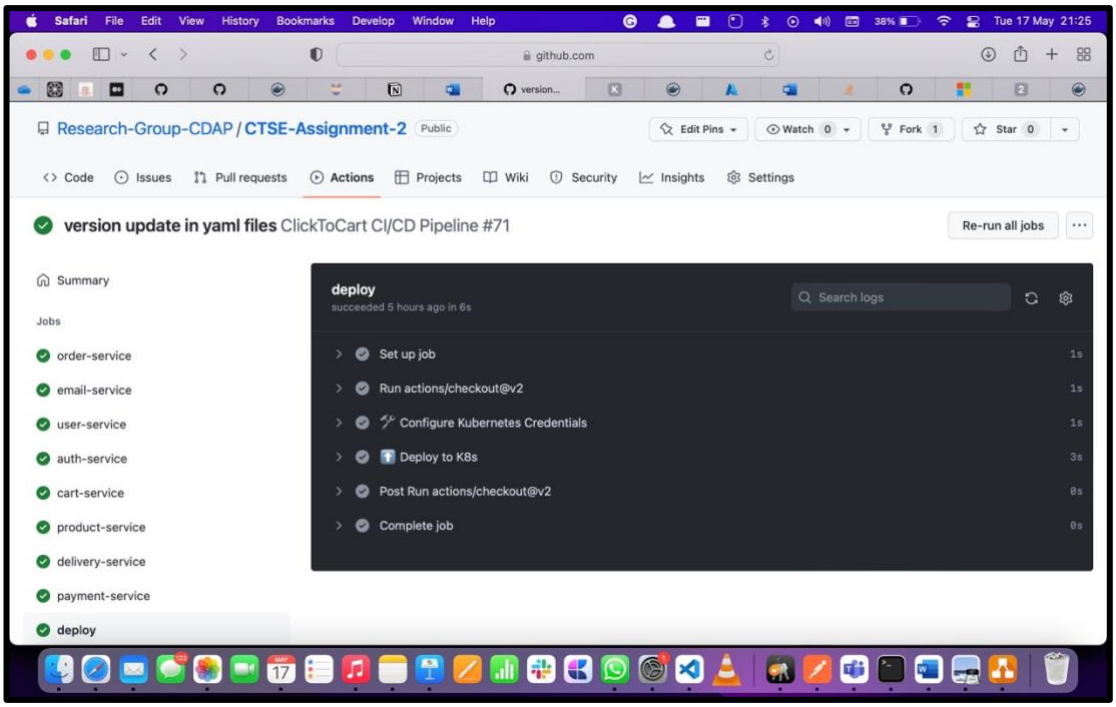


Figure 0-22:Github action



8.1.1 Github secrets

Github secrets were used to hide the credentials from YAML files

```
env:
  DOCKER_USER_LASAL: ${{secrets.DOCKER_USER_LASAL}}
  DOCKER_PASSWORD_LASAL: ${{secrets.DOCKER_PASSWORD_LASAL}}
  AUTH_REPO_NAME_LASAL: ${{secrets.AUTH_REPO_NAME_LASAL}}
  USER_REPO_NAME_LASAL: ${{secrets.USER_REPO_NAME_LASAL}}
```

8.1.2 Deployment YAML configuration of Auth service

```
jobs:
  auth-service:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Docker login
      run: | # Login to Dockerhub - Rusriu
        docker login -u $DOCKER_USER_LASAL -p $DOCKER_PASSWORD_LASAL
    - name: Build auth service docker image
      run: |
        cd src/auth_service
```

```
      docker build . --file Dockerfile --tag
$DOCKER_USER_LASAL/$AUTH_REPO_NAME_LASAL:v1.0.3
  - name: Push auth service docker image
    run: docker push $DOCKER_USER_LASAL/$AUTH_REPO_NAME_LASAL:v1.0.3
```

8.1.3 Deployment YAML configuration of User service

```
jobs:
 user-service:
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v2
  - name: Docker login
    run: | # Login to Dockerhub - Rusriu
      docker login -u $DOCKER_USER_LASAL -p $DOCKER_PASSWORD_LASAL
  - name: Build user service docker image
    run: |
      cd src/user_service
      docker build . --file Dockerfile --tag
$DOCKER_USER_LASAL/$USER_REPO_NAME_LASAL:v1.0.3
  - name: Push user service docker image
    run: docker push $DOCKER_USER_LASAL/$USER_REPO_NAME_LASAL:v1.0.3
```

8.1.3 Deployment YAML configuration for the deployment to the cluster.

```
jobs:
 deploy:
  needs: [order-service, email-service, cart-service, product-service,user-service,auth-service,delivery-
service,payment-service]
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v2
  - name: 🛠 Configure Kubernetes Credentials
    uses: Azure/aks-set-context@v1
    with:
      creds: '${{ secrets.AZURE_CREDENTIALS }}'
      cluster-name: ctse
      resource-group: CTSE
```

```
    - name: ⬆ Deploy to K8s
      run: kubectl apply -f release/
```

8.2 Deployment YAML configuration file for all services

```
name: ClickToCart CI/CD Pipeline

on:
  push:
    branches: [ master ]
  workflow_dispatch:

env:
  DOCKER_USER_RUSIRU: ${{secrets.DOCKER_USER_RUSIRU}}
  DOCKER_PASSWORD_RUSIRU: ${{secrets.DOCKER_PASSWORD_RUSIRU}}
  ORDER_REPO_NAME_RUSIRU: ${{secrets.ORDER_REPO_NAME_RUSIRU}}
  EMAIL_REPO_NAME_RUSIRU: ${{secrets.EMAIL_REPO_NAME_RUSIRU}}
  DOCKER_USER_SENURA: ${{secrets.DOCKER_USER_SENURA}}
  DOCKER_PASSWORD_SENURA: ${{secrets.DOCKER_PASSWORD_SENURA}}
  CART_REPO_NAME_SENURA: ${{secrets.CART_REPO_NAME_SENURA}}
  PRODUCT_REPO_NAME_SENURA: ${{secrets.PRODUCT_REPO_NAME_SENURA}}
  DOCKER_USER_LASAL: ${{secrets.DOCKER_USER_LASAL}}
  DOCKER_PASSWORD_LASAL: ${{secrets.DOCKER_PASSWORD_LASAL}}
  AUTH_REPO_NAME_LASAL: ${{secrets.AUTH_REPO_NAME_LASAL}}
  USER_REPO_NAME_LASAL: ${{secrets.USER_REPO_NAME_LASAL}}
  DOCKER_USER_DILMI: ${{secrets.DOCKER_USER_DILMI}}
  DOCKER_PASSWORD_DILMI: ${{secrets.DOCKER_PASSWORD_DILMI}}
  DELIVERY_REPO_NAME_DILMI: ${{secrets.DELIVERY_REPO_NAME_DILMI}}
  PAYMENT_REPO_NAME_DILMI: ${{secrets.PAYMENT_REPO_NAME_DILMI}}

jobs:
  order-service:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Docker login
```

```
      run: | # Login to Dockerhub - Rusriu
        docker login -u $DOCKER_USER_RUSIRU -p $DOCKER_PASSWORD_RUSIRU
    - name: Build order service docker image
      run: |
        cd src/order_service
        docker build . --file Dockerfile --tag
$DOCKER_USER_RUSIRU/$ORDER_REPO_NAME_RUSIRU:v1.0.1
    - name: Push order service docker image
      run: docker push $DOCKER_USER_RUSIRU/$ORDER_REPO_NAME_RUSIRU:v1.0.1

  email-service:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Docker login
      run: | # Login to Dockerhub - Rusriu
        docker login -u $DOCKER_USER_RUSIRU -p $DOCKER_PASSWORD_RUSIRU
    - name: Build email service docker image
      run: |
        cd src/email_service
        docker build . --file Dockerfile --tag
$DOCKER_USER_RUSIRU/$EMAIL_REPO_NAME_RUSIRU:v1.0.0
    - name: Push email service docker image
      run: docker push $DOCKER_USER_RUSIRU/$EMAIL_REPO_NAME_RUSIRU:v1.0.0

  user-service:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Docker login
      run: | # Login to Dockerhub - Rusriu
        docker login -u $DOCKER_USER_LASAL -p $DOCKER_PASSWORD_LASAL
    - name: Build user service docker image
      run: |
        cd src/user_service
        docker build . --file Dockerfile --tag
$DOCKER_USER_LASAL/$USER_REPO_NAME_LASAL:v1.0.3
```

```
    - name: Push user service docker image
      run: docker push $DOCKER_USER_LASAL/$USER_REPO_NAME_LASAL:v1.0.3

  auth-service:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Docker login
      run: | # Login to Dockerhub - Rusriu
        docker login -u $DOCKER_USER_LASAL -p $DOCKER_PASSWORD_LASAL
    - name: Build auth service docker image
      run: |
        cd src/auth_service
        docker build . --file Dockerfile --tag
$DOCKER_USER_LASAL/$AUTH_REPO_NAME_LASAL:v1.0.3
    - name: Push auth service docker image
      run: docker push $DOCKER_USER_LASAL/$AUTH_REPO_NAME_LASAL:v1.0.3

  cart-service:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Java version
      uses: actions/setup-java@v1
      with:
        java-version: '1.8'
    - name: Build with Maven
      run: |
        cd src/cart_service
        mvn clean package
    - name: Docker login
      run: | # Login to Dockerhub - Senura
        docker login -u $DOCKER_USER_SENURA -p $DOCKER_PASSWORD_SENURA
    - name: Build cart service docker image
      run: |
        cd src/cart_service
```

```
      docker build . --file Dockerfile --tag
$DOCKER_USER_SENURA/$CART_REPO_NAME_SENURA:v1.0.2
  - name: Push cart service docker image
    run: docker push $DOCKER_USER_SENURA/$CART_REPO_NAME_SENURA:v1.0.2

 product-service:
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v2
  - name: Set up Java version
    uses: actions/setup-java@v1
    with:
      java-version: '1.8'
  - name: Build with Maven
    run: |
      cd src/product_service
      mvn clean package
  - name: Docker login
    run: | # Login to Dockerhub - Senura
      docker login -u $DOCKER_USER_SENURA -p $DOCKER_PASSWORD_SENURA
  - name: Build product service docker image
    run: |
      cd src/product_service
      docker build . --file Dockerfile --tag
$DOCKER_USER_SENURA/$PRODUCT_REPO_NAME_SENURA:v1.0.2
  - name: Push cart service docker image
    run: docker push $DOCKER_USER_SENURA/$PRODUCT_REPO_NAME_SENURA:v1.0.2

 delivery-service:
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v2
  - name: Set up Java version
    uses: actions/setup-java@v1
    with:
      java-version: '1.8'
  - name: Build with Maven
```

```
      run: |
        cd src/delivery_service
        mvn clean package
    - name: Docker login
      run: | # Login to Dockerhub - Dilmi
        docker login -u $DOCKER_USER_DILMI -p $DOCKER_PASSWORD_DILMI
    - name: Build delivery service docker image
      run: |
        cd src/delivery_service
        docker build . --file Dockerfile --tag
$DOCKER_USER_DILMI/$DELIVERY_REPO_NAME_DILMI:latest
    - name: Push cart service docker image
      run: docker push $DOCKER_USER_DILMI/$DELIVERY_REPO_NAME_DILMI:latest


  payment-service:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Java version
      uses: actions/setup-java@v1
      with:
        java-version: '1.8'
    - name: Build with Maven
      run: |
        cd src/payment_service
        mvn clean package
    - name: Docker login
      run: | # Login to Dockerhub - Dilmi
        docker login -u $DOCKER_USER_DILMI -p $DOCKER_PASSWORD_DILMI
    - name: Build payment service docker image
      run: |
        cd src/payment_service
        docker build . --file Dockerfile --tag
$DOCKER_USER_DILMI/$PAYMENT_REPO_NAME_DILMI:latest
    - name: Push cart service docker image
      run: docker push $DOCKER_USER_DILMI/$PAYMENT_REPO_NAME_DILMI:latest
```

```
  deploy:
    needs: [order-service, email-service, cart-service, product-service,user-service,auth-service,delivery-service,payment-service]
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: 🛠 Configure Kubernetes Credentials
      uses: Azure/aks-set-context@v1
      with:
        creds: '${{ secrets.AZURE_CREDENTIALS }}'
        cluster-name: ctse
        resource-group: CTSE
    - name: ⍐ Deploy to K8s
      run: kubectl apply -f release/
```

9. K8 CLUSTER OVERVIEW

9.1 Cluster information

Kubernetes cluster deployed in AKS.

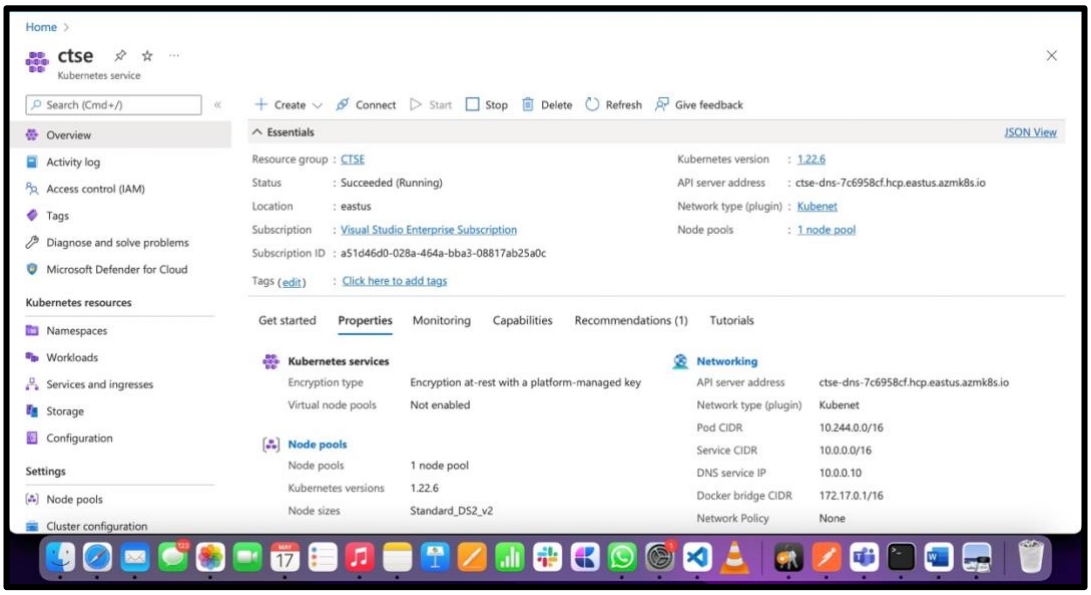


Figure 0-23:AKS cluster information

9.2 Running pods in the cluster

```
(base) lasalhettiarachchi@Lasals-MacBook-Pro Documentation % az aks get-credentials --resource-group CTSE --name ctse

Merged "ctse" as current context in /Users/lasalhettiarachchi/.kube/config
(base) lasalhettiarachchi@Lasals-MacBook-Pro Documentation % kubectl get pods
NAME                                    READY   STATUS    RESTARTS        AGE
authservice-76f8447746-2hfq6            1/1     Running   0               4h11m
authservice-76f8447746-mzpkx            1/1     Running   0               4h11m
cartservice-5dbb44f9-5gvwn              1/1     Running   0               6h54m
cartservice-5dbb44f9-5qcwk              1/1     Running   0               6h54m
deliveryservice-589565c497-9pm7g        1/1     Running   0               6h54m
deliveryservice-589565c497-s7nc6        1/1     Running   0               6h54m
emailservice-deployment-5b65667c7-tgmxt 1/1     Running   0               6h54m
emailservice-deployment-5b65667c7-wjwxg 1/1     Running   0               6h54m
orderservice-deployement-55c5c7bf67-8f8jg 1/1   Running   0               6h54m
orderservice-deployement-55c5c7bf67-fvkx5 1/1   Running   1 (6h46m ago)   6h54m
orderservice-deployement-55c5c7bf67-wrs5b 1/1   Running   0               6h54m
paymentservice-66b6b67678-5rd9c         1/1     Running   0               6h54m
paymentservice-66b6b67678-tm426         1/1     Running   0               6h54m
productservice-c799bc6f5-hkmqf          1/1     Running   0               6h54m
productservice-c799bc6f5-sqtnv          1/1     Running   0               6h54m
userservice-77f9784fd8-8znnl            1/1     Running   0               4h11m
userservice-77f9784fd8-ftgvr            1/1     Running   0               4h11m
```

Figure 0-24:Pods in K8

9.3 Running services in the cluster

```
(base) lasalhettiarachchi@Lasals-MacBook-Pro Documentation % kubectl get services
NAME              TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
auth-service      LoadBalancer   10.0.115.65    20.237.97.59     5002:30261/TCP   36h
cart-service      LoadBalancer   10.0.42.60     20.81.83.53      8081:30100/TCP   2d2h
delivery-service  LoadBalancer   10.0.147.123   20.237.34.84     9091:32299/TCP   33h
emailservice      LoadBalancer   10.0.41.102    20.121.103.149   9040:31462/TCP   2d5h
kubernetes        ClusterIP      10.0.0.1       <none>           443/TCP          2d7h
orderservice      LoadBalancer   10.0.160.7     20.85.252.187    9090:30819/TCP   2d5h
payment-service   LoadBalancer   10.0.8.34      20.237.34.98     9092:32196/TCP   33h
product-service   LoadBalancer   10.0.92.30     20.121.147.227   8082:31219/TCP   2d5h
user-service      LoadBalancer   10.0.233.171   20.237.96.28     5001:32367/TCP   36h
```

Figure 0-25:Services in K8

9.4 Running services in the browser