K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to the University of Mumbai)

Cloud Computing Emerging Technologies
Internal Assessment - 2

Deploy a User Service on Local Kubernetes Clusters using Helm Chart

GitHub Link: https://github.com/khanjasir90/CCET_IA2

YouTube Link: https://youtu.be/mUzamQIWQ7k

Group Members:
1. Jasir Khan (2021007)
2. Rayyan Mulla (2021010)
3. Asim Ahmed Siddiqui (2021014)

Related Theory:

Microservices:

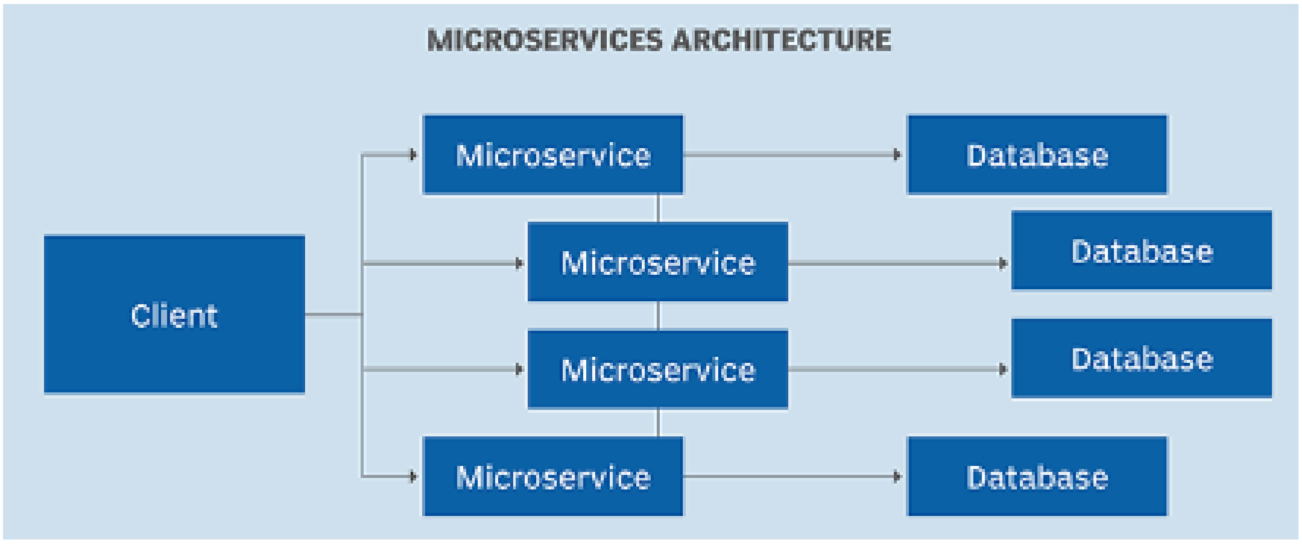
Microservices, or microservices architecture, is an approach to application development in which a large application is built from modular components or services. Each module supports a specific task or business goal and uses a simple, well-defined interface, such as an application programming interface (API), to communicate with other sets of services.
Software developer and author Martin Fowler is credited with promoting the idea of breaking down services in a service-oriented architecture (SOA) into microservices.

How microservices work

In a microservices architecture, an application is divided into services. Each service runs a unique process and usually manages its own database. A service can generate alerts, log data, support user interfaces (UIs), handle user identification or authentication and perform various other tasks.

The microservices paradigm provides development teams with a more decentralized approach to building software. Each service can be isolated, rebuilt, redeployed and managed independently.

For example, if a program isn't properly generating reports, IT staff can trace the problem to a specific service and then test, restart, patch and redeploy that service as needed, independent of other services.



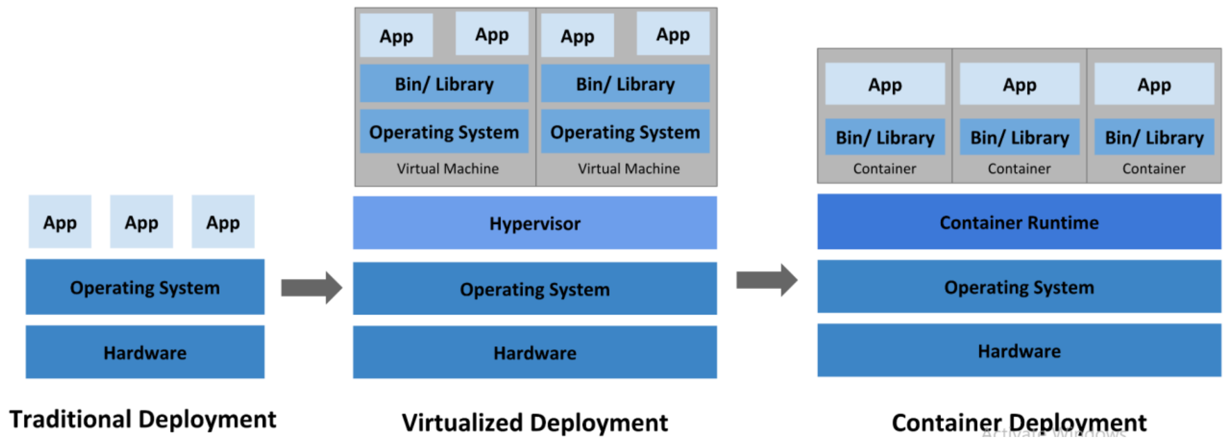
Benefits of a microservices architecture
Microservices designs and deployments have grown thanks to the cloud, containerization and hyperconnected systems. However, microservices pose a series of tradeoffs for software developers.

In terms of advantages, microservices:
- can be developed and deployed using different languages and tools;
- require less development time;
- can scale quickly;
- can be reused in different projects;
- contain better fault isolation;
- are faster and far less resource-intensive to deploy and load balance;

Challenges of a microservices architecture
- potentially too much granularity;
- extra effort designing for communication between services;
- complex testing;
- additional management and control; and
- comprehensive security.

Kubernetes:
Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF).

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.



Traditional deployment era:

Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.

For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform.

Virtualized deployment era:

As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines. Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

Container deployment era:

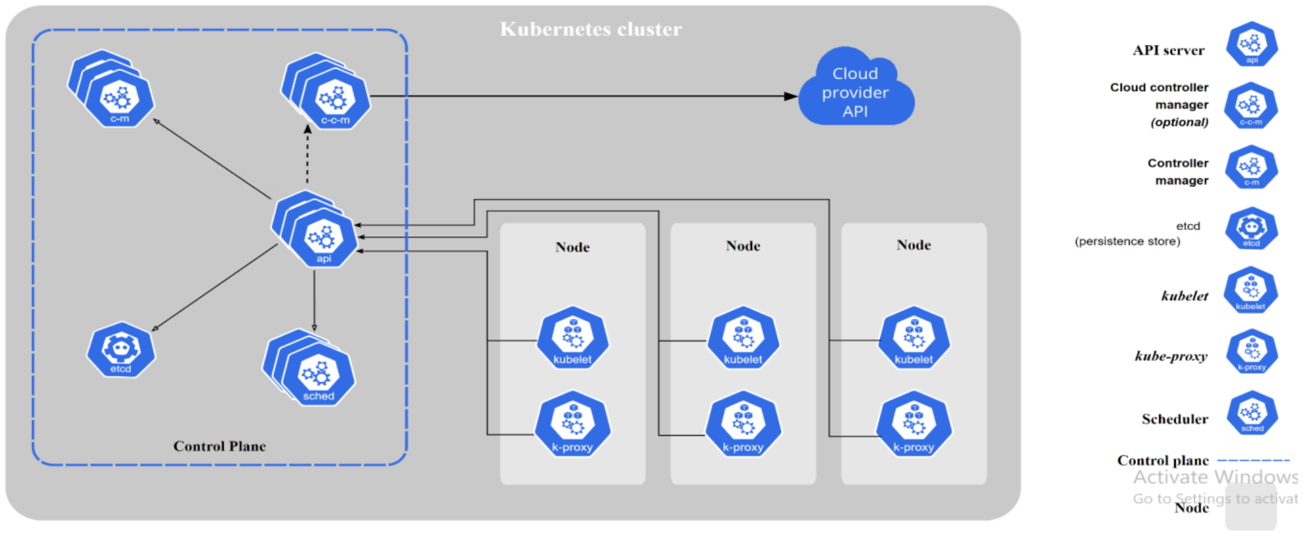
Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Need of Kubernetes:

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?

- Service discovery and load balancing Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
- Storage orchestration Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- Automated rollouts and rollbacks You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate.

Kubernetes Components:



kube-apiserver

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

The main implementation of a Kubernetes API server is kube-apiserver. kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

etcd

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data. If your Kubernetes cluster uses etcd as its backing store, make sure you have a back up plan for that data. You can find in-depth information about etcd in the official documentation.

kube-scheduler

Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on. Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.

kube-controller-manager

Control plane component that runs controller processes. Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

cloud-controller-manager

A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

Helm:

There are thousands of people and companies packaging their applications for deployment on Kubernetes. This usually involves crafting a few different Kubernetes resource definitions that configure the application runtime, as well as defining the mechanism that users and other apps leverage to communicate with the application. There are some very common applications that users regularly look for guidance on deploying, such as databases, CI tools, and content management systems. These types of applications are usually not ones that are developed and iterated on by end users, but rather their configuration is customized to fit a specific use case. Once that application is deployed users can link it to their existing systems or leverage their functionality to solve their pain points.

xkcd Standards

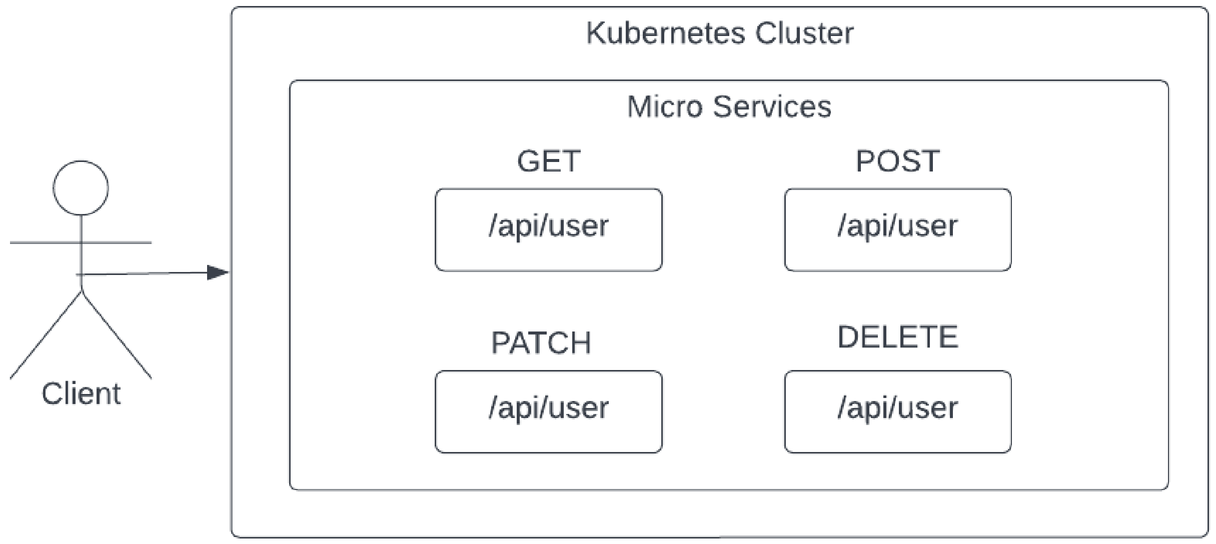
In this case, we're not creating Yet Another Place for Applications, rather promoting an existing one as the canonical location. As part of the Special Interest Group Apps work for the Kubernetes 1.4 release, we began to provide a home for these Kubernetes deployable applications that provides continuous releases of well documented and user-friendly packages. These packages are being created as Helm Charts and can be installed using the Helm tool. Helm allows users to easily template their Kubernetes manifests and provide a set of configuration parameters that allows users to customize their deployment. Helm is the package manager and Charts are packages. The home for these Charts is the Kubernetes Charts repository which provides continuous integration for pull requests, as well as automated releases of Charts in the master branch.

Why use Helm?

Writing and maintaining Kubernetes **YAML manifests** for all the required Kubernetes objects can be a time consuming and tedious task. For the simplest of deployments, you would need at least 3 YAML manifests with duplicated and hardcoded values. Helm simplifies this process and creates a single package that can be advertised to your cluster. Helm is a client/server application and, until recently, has relied on Tiller to be deployed in your cluster. This gets installed when installing the helm on your client machine. Tiller simply receives requests from the client and installs the package into your cluster.

Implementation Details:

System Architecture:



NodeJS:

Node.js uses an event-driven architecture and enables efficient, real-time application development. Node.js single-threading and asynchronous capabilities enable a non-blocking mechanism. We have used NodeJS as the backend for this application.

Client:

The end user interacts with the Application User Interface and requests for the services using various HTTP methods like GET, POST, PATCH, DELETE

Kubernetes:

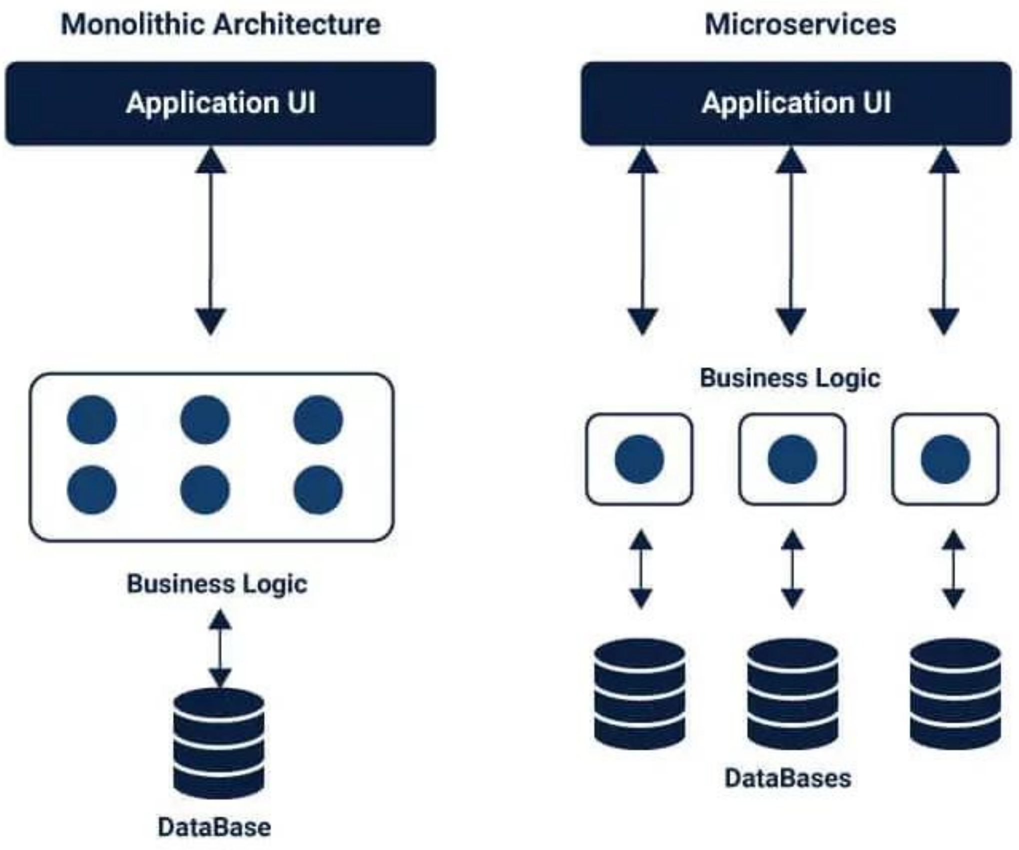
Kubernetes is an open-source system for automating deployments, scaling, and management of containerized applications.

There are several ways to create a Kubernetes cluster:

- Using a managed Kubernetes service like Google Kubernetes Service (GKE), Azure Kubernetes Service (AKS), or Amazon Elastic Kubernetes Service (EKS)
- Installing Kubernetes, yourself on cloud or on-premises infrastructure with a Kubernetes installation tool like kubeadm or kops
- Creating a Kubernetes cluster on your local machine with a tool like Minikube, MicroK8s, or k3s

Micro Services:

Building NodeJS applications on microservices help you focus on developing monofunctional modules with clearly defined operations and precise interfaces. The application development process becomes more agile, and the challenges of continuous testing are mitigated.



When you build applications on a monolithic architecture, the entire application needs to be deployed with every update. On the other hand, microservices have no dependency on the type of framework, technique or programming language being used to build them. Your ability to release REST-ful APIs for communication and other services is the only requisite for microservice architecture.

Steps followed to create NodeJS application:

Initialize NodeJS Server

The Node.js installation includes NPM, which is the Node.js package manager. NPM to bootstrap the project, install dependencies and execute the service.

We have used Express to build our service. Express is an established framework for Node applications and continues to benefit from the support of the Node.js Foundation.

Creating a Server to Accept Requests

Create a file in the root folder for your project called server.js
We define all the routing and app related configurations in this file.

Defining the Routes

The next step is to define the routes for the server and assign each to a target in our controller object. We'll build the controller in the next step.
We have four endpoints.

1. GET to fetch user details
2. POST to insert user details
3. PATCH to update the user details
4. Delete to delete the user record.

Adding Controller Logic

Create controller object to perform application logic, such as

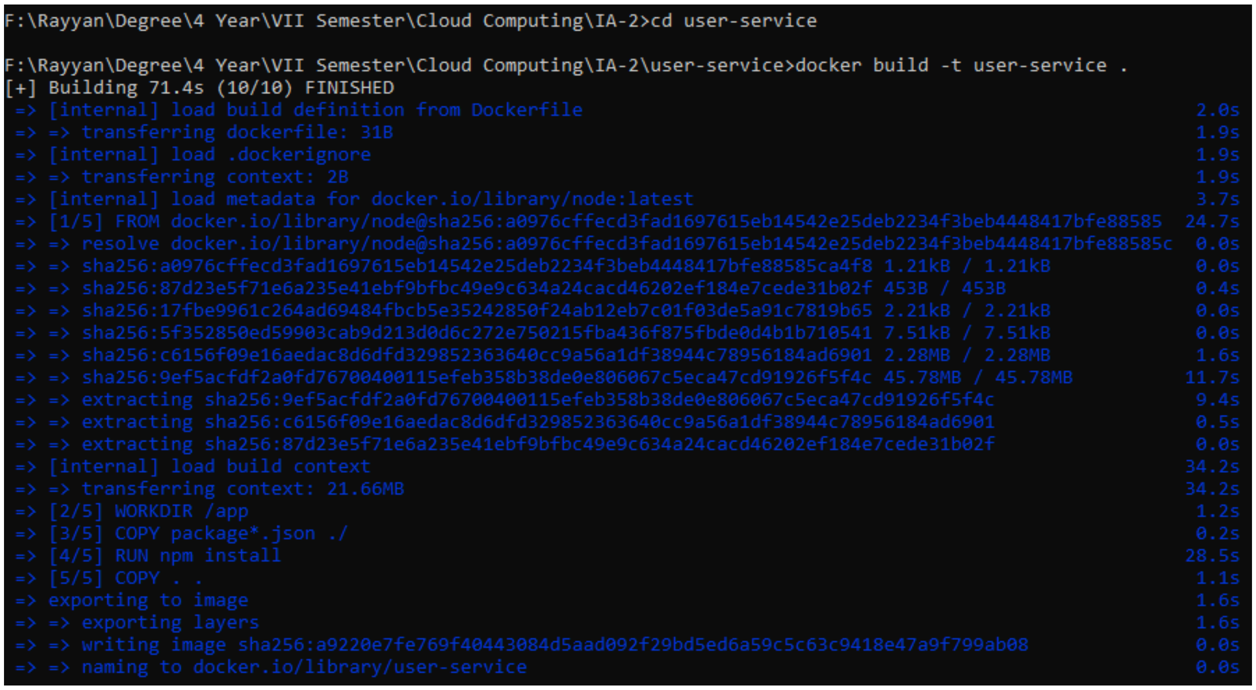
- Retrieve user details from MongoDB database and return this as a response
- Add new user using GET request and create a new document in MongoDB
- Update user details using POST request.
- Delete the user record using MongoDB ID

Making the External Call

Use the above created controller and routing to perform the application operations.
We have used Thunder Client to test and debug APIs created using NodeJS.

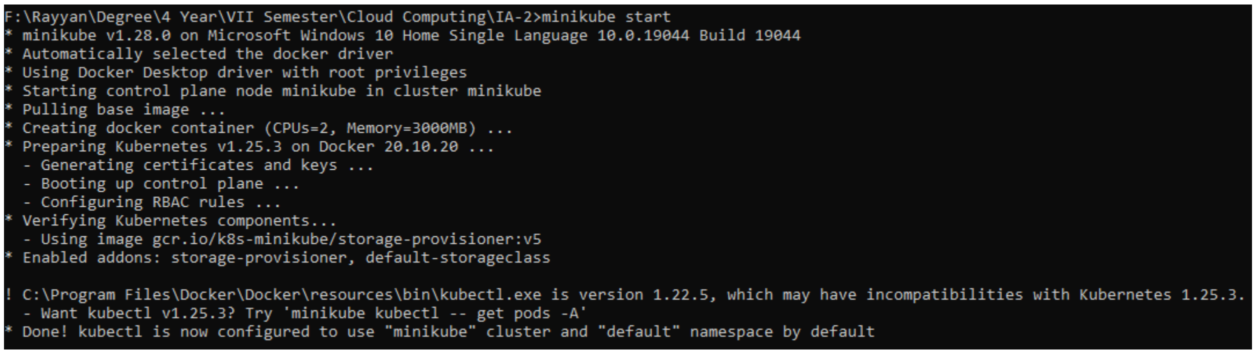
Commands to deploy the microservice on local kubernetes cluster:

`docker build -t user-service`



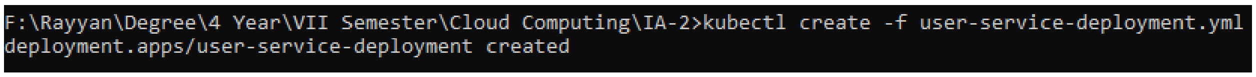
```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>cd user-service

F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2\user-service>docker build -t user-service .
[+] Building 71.4s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                                                     2.0s
 => => transferring dockerfile: 31B                                                                      1.9s
 => [internal] load .dockerignore                                                                        1.9s
 => => transferring context: 2B                                                                          1.9s
 => [internal] load metadata for docker.io/library/node:latest                                           3.7s
 => [1/5] FROM docker.io/library/node@sha256:a0976cffecd3fad1697615eb14542e25deb2234f3beb4448417bfe88585  24.7s
 => => resolve docker.io/library/node@sha256:a0976cffecd3fad1697615eb14542e25deb2234f3beb4448417bfe88585c  0.0s
 => => sha256:a0976cffecd3fad1697615eb14542e25deb2234f3beb4448417bfe88585ca4f8 1.21kB / 1.21kB            0.0s
 => => sha256:87d23e5f71e6a235e41ebf9bfbc49e9c634a24cacd46202ef184e7cede31b02f 453B / 453B                0.4s
 => => sha256:17fbe9961c264ad69484fbcb5e35242850f24ab12eb7c01f03de5a91c7819b65 2.21kB / 2.21kB            0.0s
 => => sha256:5f352850ed59903cab9d213d0d6c272e750215fba436f875fbde0d4b1b710541 7.51kB / 7.51kB            0.0s
 => => sha256:c6156f09e16aedac8d6dfd329852363640cc9a56a1df38944c78956184ad6901 2.28MB / 2.28MB            1.6s
 => => sha256:9ef5acfdf2a0fd76700400115efeb358b38de0e806067c5eca47cd91926f5f4c 45.78MB / 45.78MB         11.7s
 => => extracting sha256:9ef5acfdf2a0fd76700400115efeb358b38de0e806067c5eca47cd91926f5f4c                9.4s
 => => extracting sha256:c6156f09e16aedac8d6dfd329852363640cc9a56a1df38944c78956184ad6901                0.5s
 => => extracting sha256:87d23e5f71e6a235e41ebf9bfbc49e9c634a24cacd46202ef184e7cede31b02f                0.0s
 => [internal] load build context                                                                       34.2s
 => => transferring context: 21.66MB                                                                    34.2s
 => [2/5] WORKDIR /app                                                                                   1.2s
 => [3/5] COPY package*.json ./                                                                          0.2s
 => [4/5] RUN npm install                                                                               28.5s
 => [5/5] COPY . .                                                                                       1.1s
 => exporting to image                                                                                   1.6s
 => => exporting layers                                                                                  1.6s
 => => writing image sha256:a9220e7fe769f40443084d5aad092f29bd5ed6a59c5c63c9418e47a9f799ab08             0.0s
 => => naming to docker.io/library/user-service                                                          0.0s
```

`minikube start`



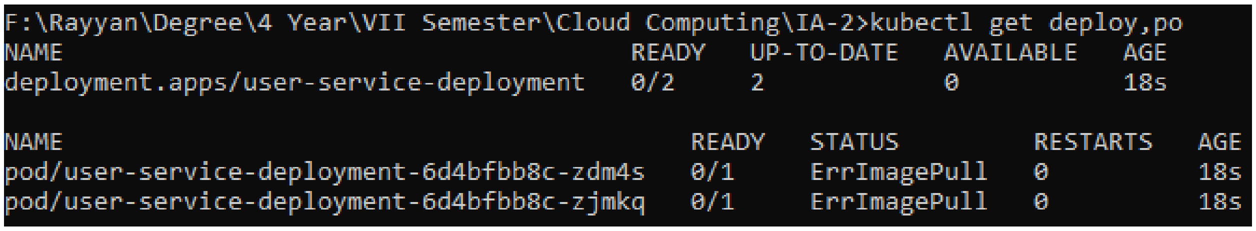
```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>minikube start
* minikube v1.28.0 on Microsoft Windows 10 Home Single Language 10.0.19044 Build 19044
* Automatically selected the docker driver
* Using Docker Desktop driver with root privileges
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Creating docker container (CPUs=2, Memory=3000MB) ...
* Preparing Kubernetes v1.25.3 on Docker 20.10.20 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass

! C:\Program Files\Docker\Docker\resources\bin\kubectl.exe is version 1.22.5, which may have incompatibilities with Kubernetes 1.25.3.
  - Want kubectl v1.25.3? Try 'minikube kubectl -- get pods -A'
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

`kubectl create -f user-service-deployment.yml`



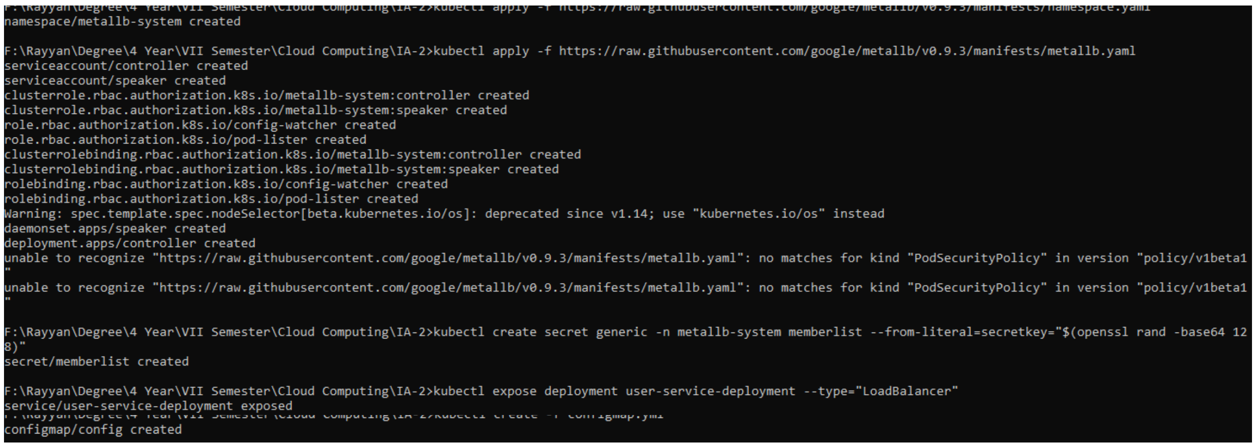
```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl create -f user-service-deployment.yml
deployment.apps/user-service-deployment created
```

```
kubectl get deploy,po
```



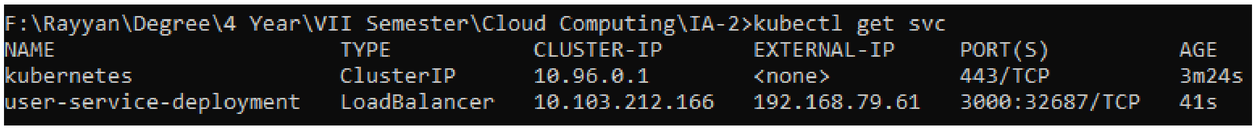
```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl get deploy,po
NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/user-service-deployment   0/2     2            0           18s

NAME                                          READY   STATUS         RESTARTS   AGE
pod/user-service-deployment-6d4bfbb8c-zdm4s   0/1     ErrImagePull   0          18s
pod/user-service-deployment-6d4bfbb8c-zjmkq   0/1     ErrImagePull   0          18s
```

```
kubectl apply -f
https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/namespace.yaml

kubectl apply -f
https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/metallb.yaml

kubectl create secret generic -n metallb-system memberlist --from-
literal=secretkey="$(openssl rand -base64 128)"

kubectl expose deployment user-service-deployment --type="LoadBalancer"

kubectl create -f configmap.yml
```



```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl apply -f https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/namespace.yaml
namespace/metallb-system created

F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl apply -f https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/metallb.yaml
serviceaccount/controller created
serviceaccount/speaker created
clusterrole.rbac.authorization.k8s.io/metallb-system:controller created
clusterrole.rbac.authorization.k8s.io/metallb-system:speaker created
role.rbac.authorization.k8s.io/config-watcher created
role.rbac.authorization.k8s.io/pod-lister created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:controller created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:speaker created
rolebinding.rbac.authorization.k8s.io/config-watcher created
rolebinding.rbac.authorization.k8s.io/pod-lister created
Warning: spec.template.spec.nodeSelector[beta.kubernetes.io/os]: deprecated since v1.14; use "kubernetes.io/os" instead
daemonset.apps/speaker created
deployment.apps/controller created
unable to recognize "https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/metallb.yaml": no matches for kind "PodSecurityPolicy" in version "policy/v1beta1
"
unable to recognize "https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/metallb.yaml": no matches for kind "PodSecurityPolicy" in version "policy/v1beta1
"

F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64 12
8)"
secret/memberlist created

F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl expose deployment user-service-deployment --type="LoadBalancer"
service/user-service-deployment exposed
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl create -f configmap.yml
configmap/config created
```

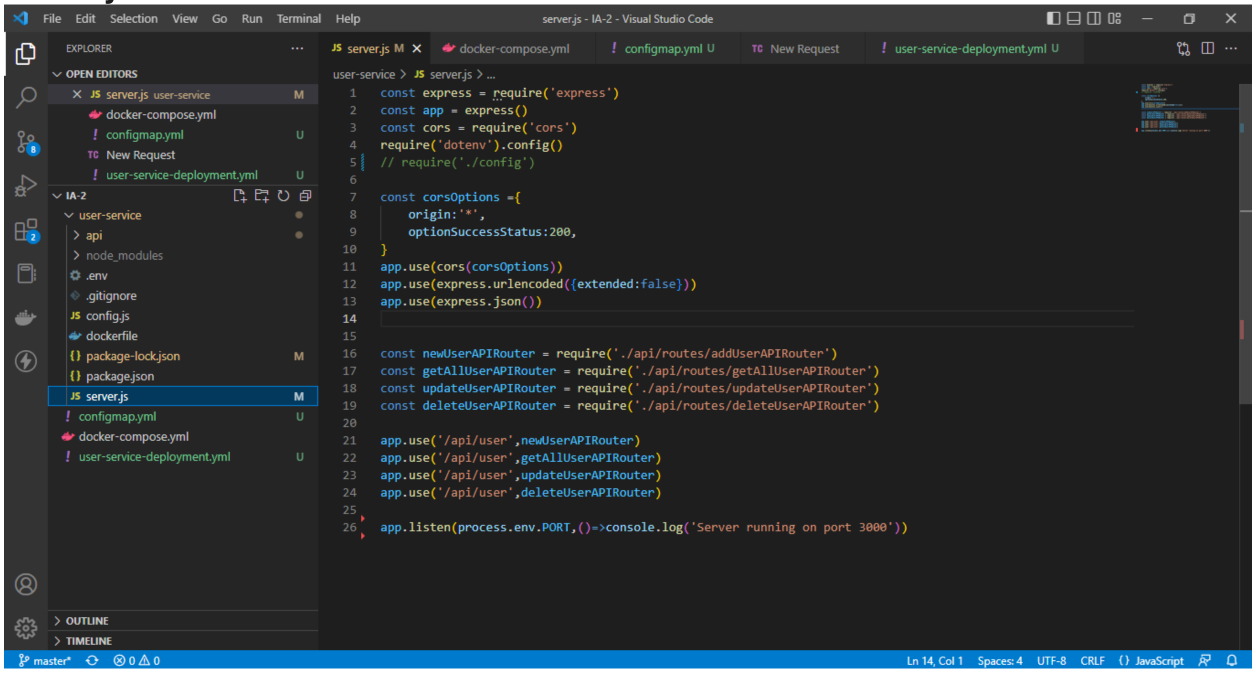
```
kubectl get svc
```



```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl get svc
NAME                      TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)          AGE
kubernetes                ClusterIP      10.96.0.1        <none>          443/TCP          3m24s
user-service-deployment   LoadBalancer   10.103.212.166   192.168.79.61   3000:32687/TCP   41s
```

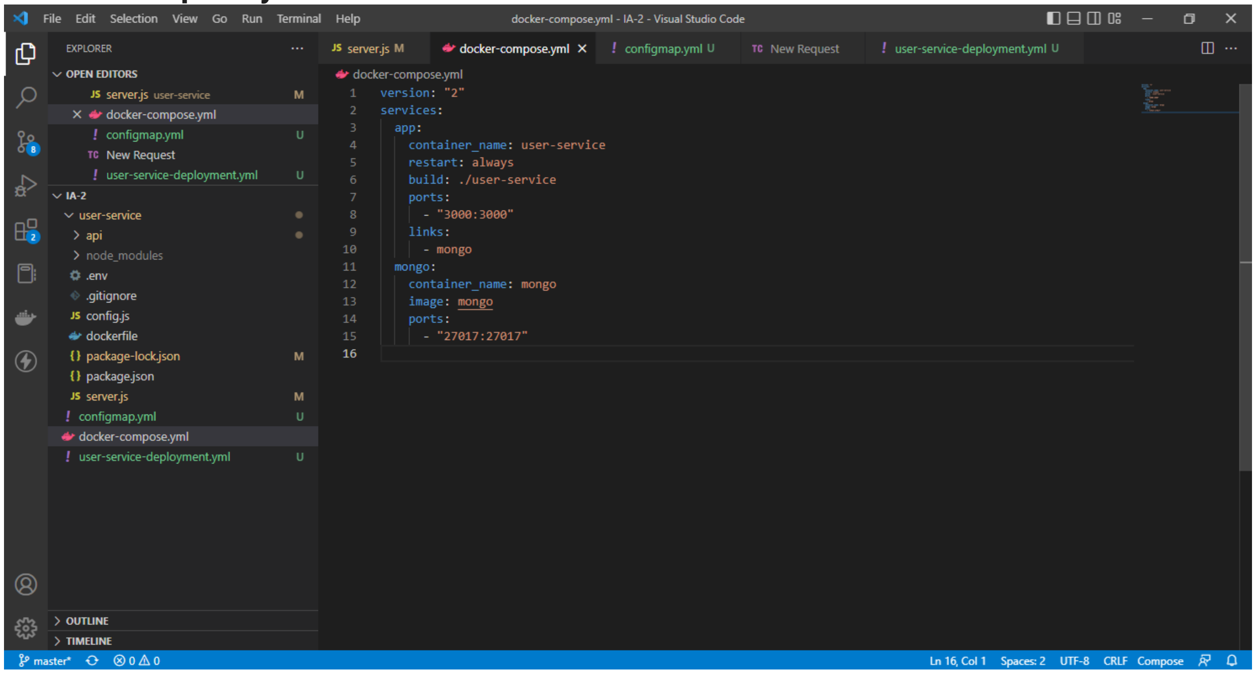
Screenshots:

Server.js file



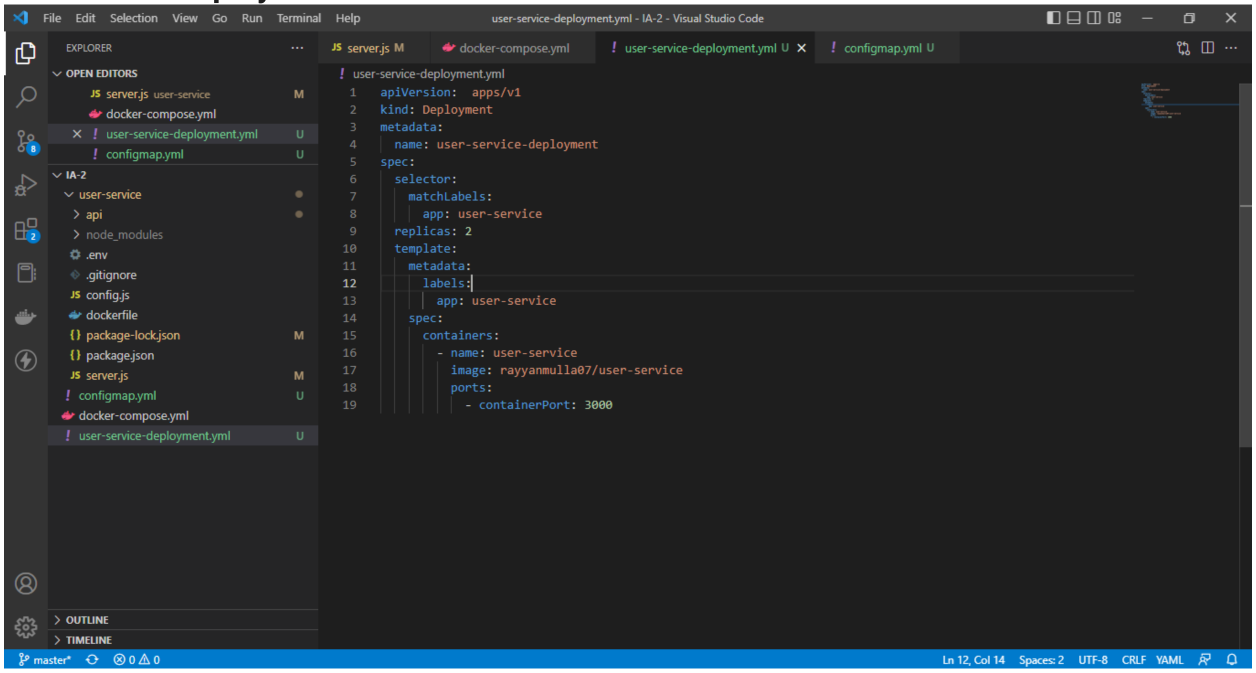
```javascript
const express = require('express')
const app = express()
const cors = require('cors')
require('dotenv').config()
// require('./config')

const corsOptions ={
    origin:'*',
    optionSuccessStatus:200,
}
app.use(cors(corsOptions))
app.use(express.urlencoded({extended:false}))
app.use(express.json())


const newUserAPIRouter = require('./api/routes/addUserAPIRouter')
const getAllUserAPIRouter = require('./api/routes/getAllUserAPIRouter')
const updateUserAPIRouter = require('./api/routes/updateUserAPIRouter')
const deleteUserAPIRouter = require('./api/routes/deleteUserAPIRouter')

app.use('/api/user',newUserAPIRouter)
app.use('/api/user',getAllUserAPIRouter)
app.use('/api/user',updateUserAPIRouter)
app.use('/api/user',deleteUserAPIRouter)

app.listen(process.env.PORT,()=>console.log('Server running on port 3000'))
```

Docker-compose.yml file



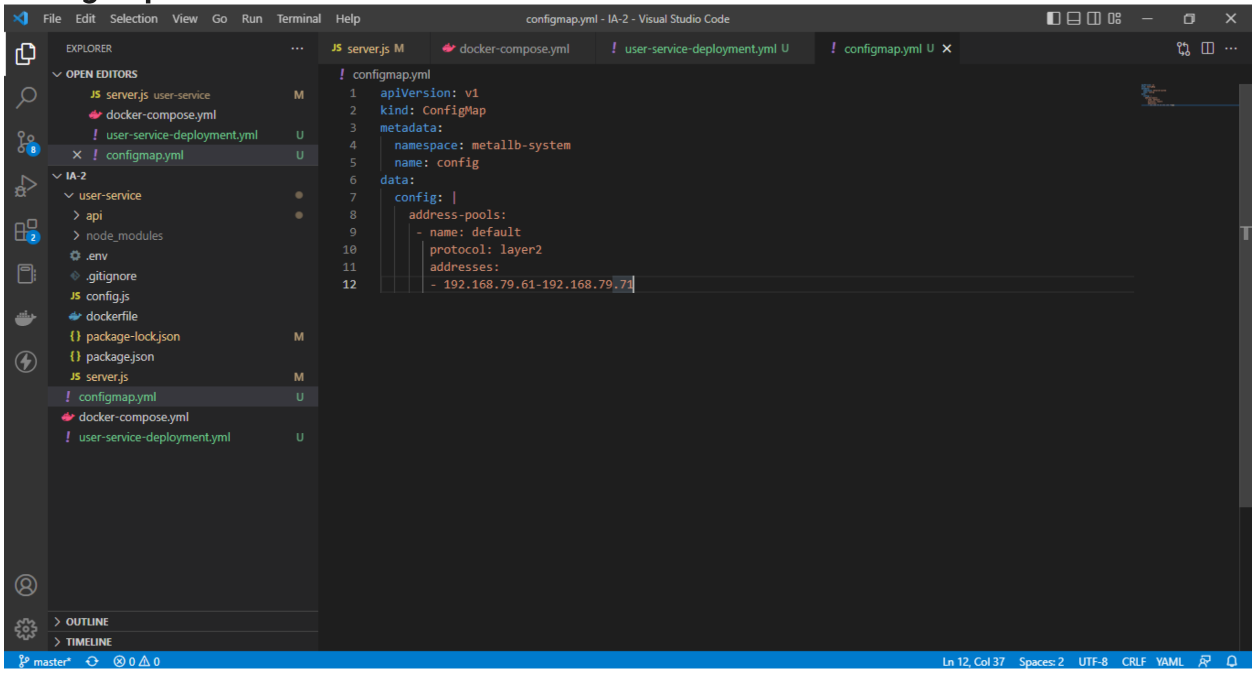
```yaml
version: "2"
services:
  app:
    container_name: user-service
    restart: always
    build: ./user-service
    ports:
      - "3000:3000"
    links:
      - mongo
  mongo:
    container_name: mongo
    image: mongo
    ports:
      - "27017:27017"
```

Kubernetes deployment file

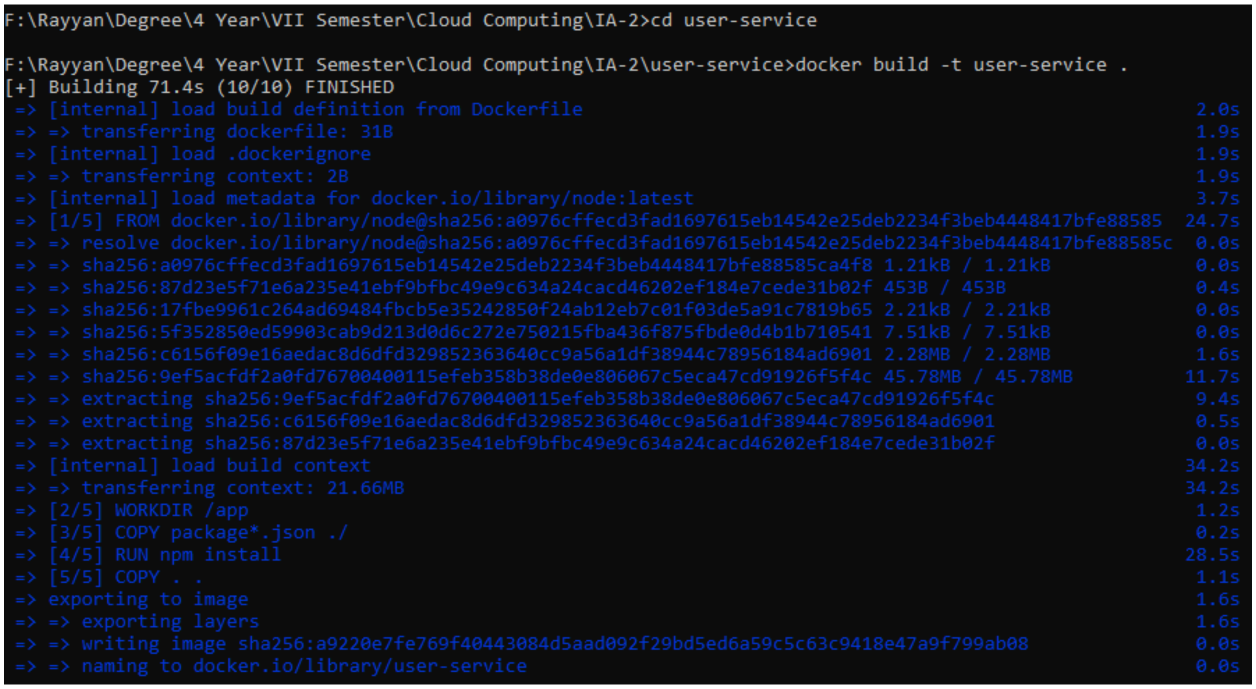


```yaml
apiVersion:  apps/v1
kind: Deployment
metadata:
  name: user-service-deployment
spec:
  selector:
    matchLabels:
      app: user-service
  replicas: 2
  template:
    metadata:
      labels:
        app: user-service
    spec:
      containers:
        - name: user-service
          image: rayyanmulla07/user-service
          ports:
            - containerPort: 3000
```

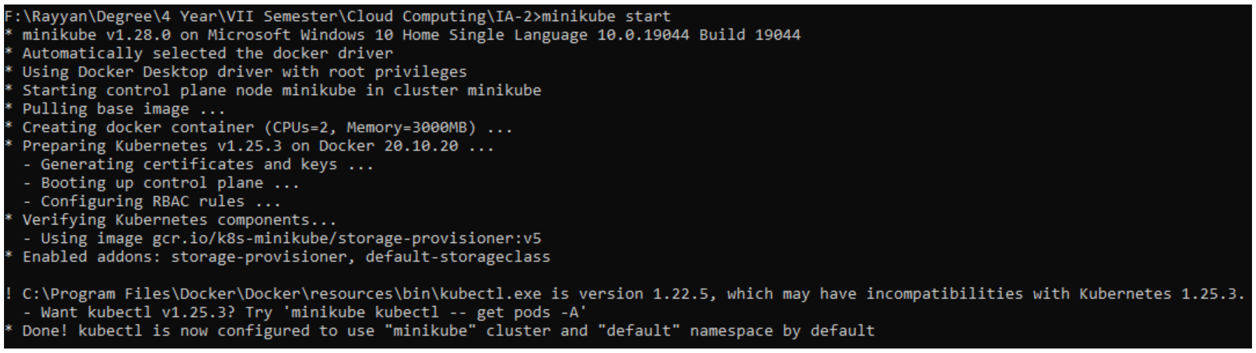
Configmap file



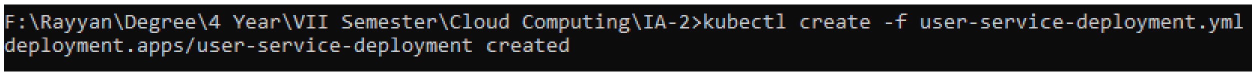
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.79.61-192.168.79.71
```



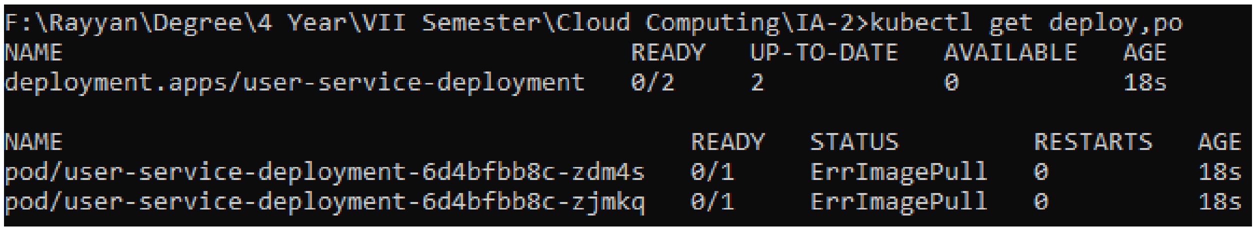
```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>cd user-service

F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2\user-service>docker build -t user-service .
[+] Building 71.4s (10/10) FINISHED
 => [internal] load build definition from Dockerfile                                                         2.0s
 => => transferring dockerfile: 31B                                                                          1.9s
 => [internal] load .dockerignore                                                                            1.9s
 => => transferring context: 2B                                                                              1.9s
 => [internal] load metadata for docker.io/library/node:latest                                               3.7s
 => [1/5] FROM docker.io/library/node@sha256:a0976cffecd3fad1697615eb14542e25deb2234f3beb4448417bfe88585  24.7s
 => => resolve docker.io/library/node@sha256:a0976cffecd3fad1697615eb14542e25deb2234f3beb4448417bfe88585c  0.0s
 => => sha256:a0976cffecd3fad1697615eb14542e25deb2234f3beb4448417bfe88585ca4f8 1.21kB / 1.21kB               0.0s
 => => sha256:87d23e5f71e6a235e41ebf9bfbc49e9c634a24cacd46202ef184e7cede31b02f 453B / 453B                   0.4s
 => => sha256:17fbe9961c264ad69484fbcb5e35242850f24ab12eb7c01f03de5a91c7819b65 2.21kB / 2.21kB               0.0s
 => => sha256:5f352850ed59903cab9d213d0d6c272e750215fba436f875fbde0d4b1b710541 7.51kB / 7.51kB               0.0s
 => => sha256:c6156f09e16aedac8d6dfd329852363640cc9a56a1df38944c78956184ad6901 2.28MB / 2.28MB               1.6s
 => => sha256:9ef5acfdf2a0fd76700400115efeb358b38de0e806067c5eca47cd91926f5f4c 45.78MB / 45.78MB            11.7s
 => => extracting sha256:9ef5acfdf2a0fd76700400115efeb358b38de0e806067c5eca47cd91926f5f4c                    9.4s
 => => extracting sha256:c6156f09e16aedac8d6dfd329852363640cc9a56a1df38944c78956184ad6901                    0.5s
 => => extracting sha256:87d23e5f71e6a235e41ebf9bfbc49e9c634a24cacd46202ef184e7cede31b02f                    0.0s
 => [internal] load build context                                                                           34.2s
 => => transferring context: 21.66MB                                                                        34.2s
 => [2/5] WORKDIR /app                                                                                       1.2s
 => [3/5] COPY package*.json ./                                                                              0.2s
 => [4/5] RUN npm install                                                                                   28.5s
 => [5/5] COPY . .                                                                                           1.1s
 => exporting to image                                                                                       1.6s
 => => exporting layers                                                                                      1.6s
 => => writing image sha256:a9220e7fe769f40443084d5aad092f29bd5ed6a59c5c63c9418e47a9f799ab08                 0.0s
 => => naming to docker.io/library/user-service                                                              0.0s
```



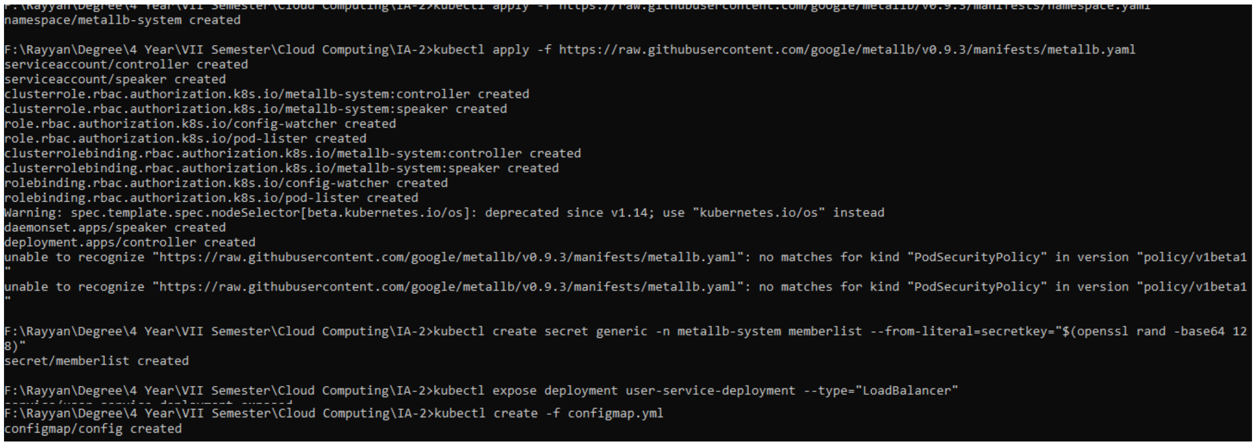
```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>minikube start
* minikube v1.28.0 on Microsoft Windows 10 Home Single Language 10.0.19044 Build 19044
* Automatically selected the docker driver
* Using Docker Desktop driver with root privileges
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Creating docker container (CPUs=2, Memory=3000MB) ...
* Preparing Kubernetes v1.25.3 on Docker 20.10.20 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass

! C:\Program Files\Docker\Docker\resources\bin\kubectl.exe is version 1.22.5, which may have incompatibilities with Kubernetes 1.25.3.
  - Want kubectl v1.25.3? Try 'minikube kubectl -- get pods -A'
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```



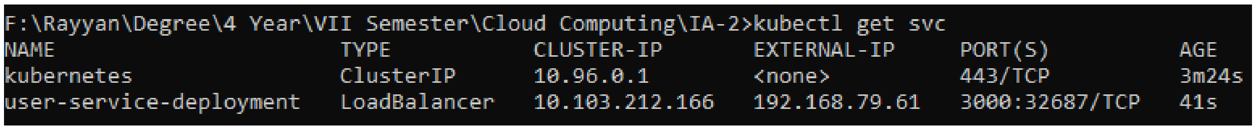
```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl create -f user-service-deployment.yml
deployment.apps/user-service-deployment created
```



```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl get deploy,po
NAME                                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/user-service-deployment   0/2     2            0           18s

NAME                                          READY   STATUS         RESTARTS   AGE
pod/user-service-deployment-6d4bfbb8c-zdm4s   0/1     ErrImagePull   0          18s
pod/user-service-deployment-6d4bfbb8c-zjmkq   0/1     ErrImagePull   0          18s
```

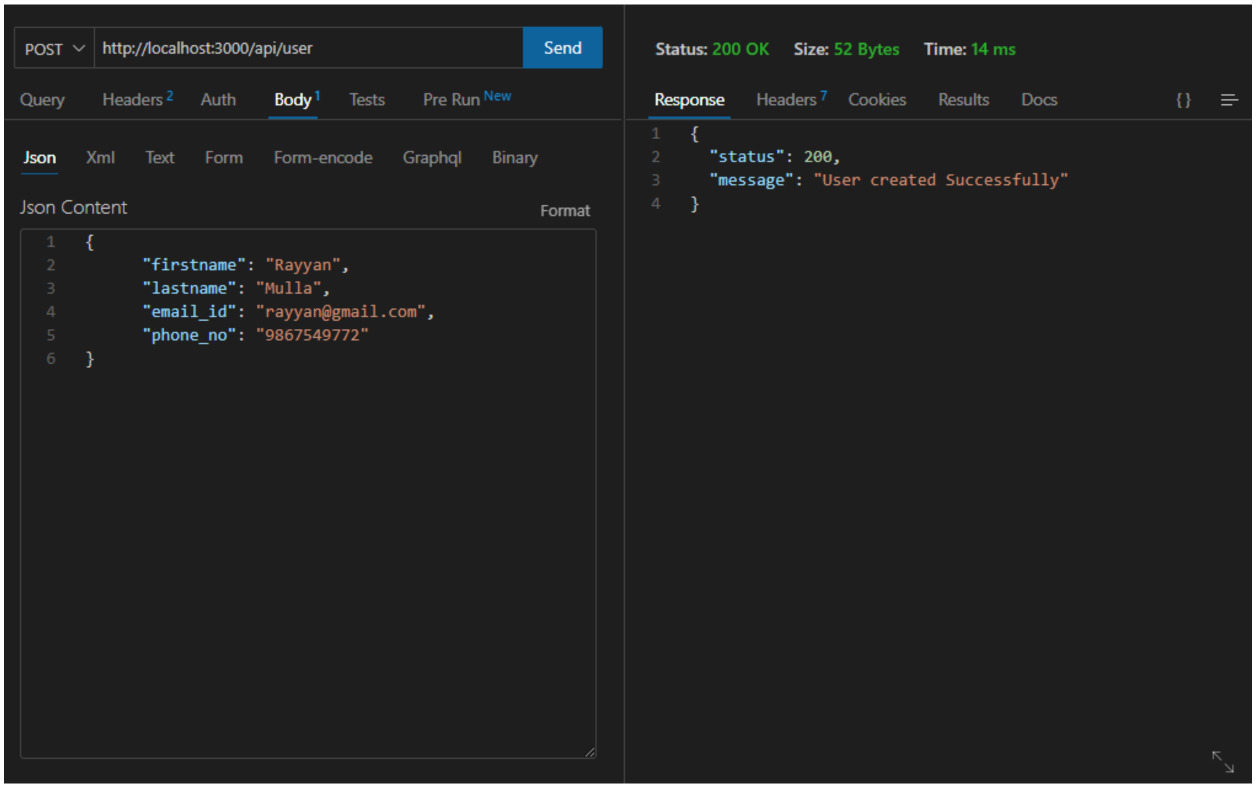


```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl apply -f https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/namespace.yaml
namespace/metallb-system created

F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl apply -f https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/metallb.yaml
serviceaccount/controller created
serviceaccount/speaker created
clusterrole.rbac.authorization.k8s.io/metallb-system:controller created
clusterrole.rbac.authorization.k8s.io/metallb-system:speaker created
role.rbac.authorization.k8s.io/config-watcher created
role.rbac.authorization.k8s.io/pod-lister created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:controller created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:speaker created
rolebinding.rbac.authorization.k8s.io/config-watcher created
rolebinding.rbac.authorization.k8s.io/pod-lister created
Warning: spec.template.spec.nodeSelector[beta.kubernetes.io/os]: deprecated since v1.14; use "kubernetes.io/os" instead
daemonset.apps/speaker created
deployment.apps/controller created
unable to recognize "https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/metallb.yaml": no matches for kind "PodSecurityPolicy" in version "policy/v1beta1
"
unable to recognize "https://raw.githubusercontent.com/google/metallb/v0.9.3/manifests/metallb.yaml": no matches for kind "PodSecurityPolicy" in version "policy/v1beta1
"

F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64 12
8)"
secret/memberlist created

F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl expose deployment user-service-deployment --type="LoadBalancer"
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl create -f configmap.yml
configmap/config created
```



```
F:\Rayyan\Degree\4 Year\VII Semester\Cloud Computing\IA-2>kubectl get svc
NAME                      TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)          AGE
kubernetes                ClusterIP      10.96.0.1        <none>          443/TCP          3m24s
user-service-deployment   LoadBalancer   10.103.212.166   192.168.79.61   3000:32687/TCP   41s
```

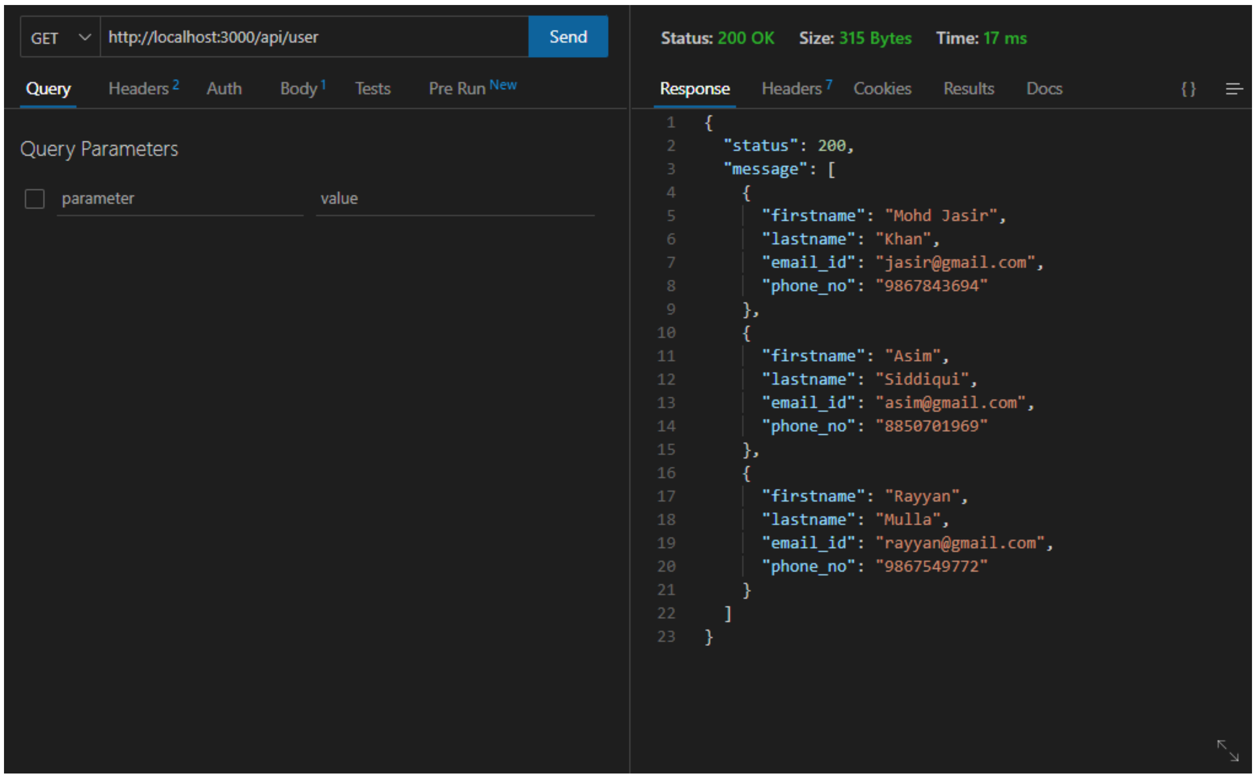
POST Request



```
{
        "firstname": "Rayyan",
        "lastname": "Mulla",
        "email_id": "rayyan@gmail.com",
        "phone_no": "9867549772"
}
```

```
{
  "status": 200,
  "message": "User created Successfully"
}
```

GET Request



```
{
  "status": 200,
  "message": [
    {
      "firstname": "Mohd Jasir",
      "lastname": "Khan",
      "email_id": "jasir@gmail.com",
      "phone_no": "9867843694"
    },
    {
      "firstname": "Asim",
      "lastname": "Siddiqui",
      "email_id": "asim@gmail.com",
      "phone_no": "8850701969"
    },
    {
      "firstname": "Rayyan",
      "lastname": "Mulla",
      "email_id": "rayyan@gmail.com",
      "phone_no": "9867549772"
    }
  ]
}
```

References:

1. https://kubernetes.io/
2. https://kubernetes.io/docs/concepts/overview/
3. https://www.bmc.com/blogs/kubernetes-helm-charts/
4. https://microservices.io/
5. https://www.techtarget.com/searchapparchitecture/definition/microservices
6. https://blog.cloud66.com/beginners-guide-to-building-real-world-microservices-with-node-js
7. https://www.bacancytechnology.com/blog/how-to-build-microservices-with-node-js
8. https://eskala.io/tutorial/the-node-js-developers-guide-to-kubernetes-part-i/