Cloud Services and Integration

CA

Practical Assessment with Report

On

Building A Cloud Based Microservices System

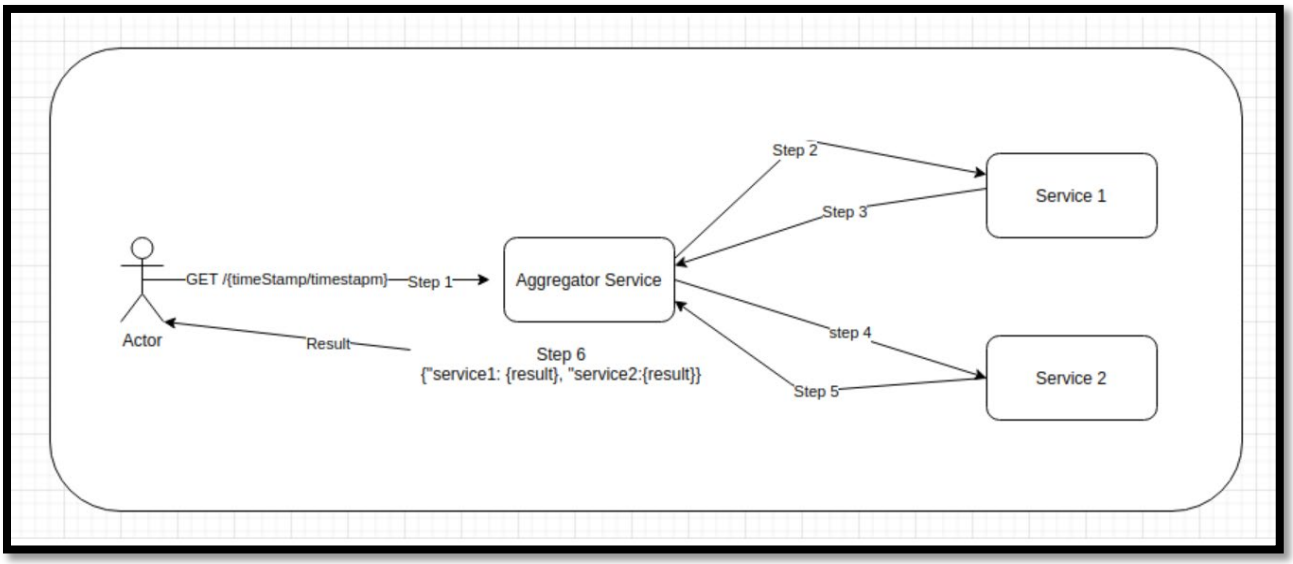
Submitted by: Akshay Jadhav (2020175)

Submitted to: Prof.David Gonzalez

1. Introduction

We are developing the microservices architecture which contains the 3 services like India-time service, US time service, and aggregator service.

The India service and US time service will convert the Unix timestamp into the required timezone (Indian and US time). The aggregator service will aggregate the two services and show the results as shown in the below diagram.



- **Why used Java?**

Chosen to use java for this microservices architecture development because especially everybody is aware of this language for a long time.

I personally find java code is easy to read and maintain as compared to other languages like Python, .NET, etc.

Spring Boot and Spark Java are famous Java microservices frameworks available for developing all types of solutions which provides higher functionality.

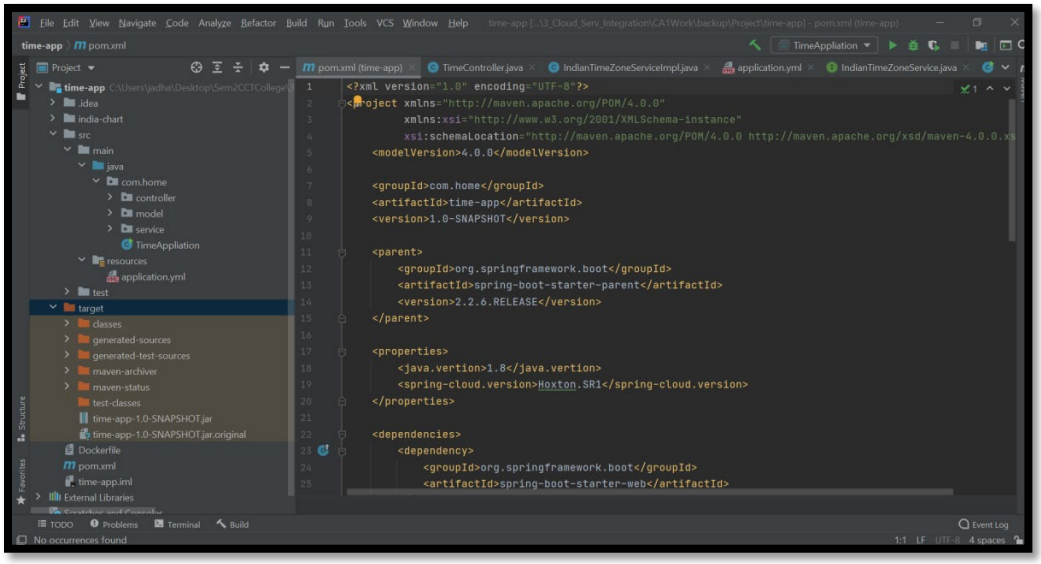
For development, I have used IntelliJ IDE because is easy to use and way faster than eclipse with options of more plugins.

2. Building Microservices

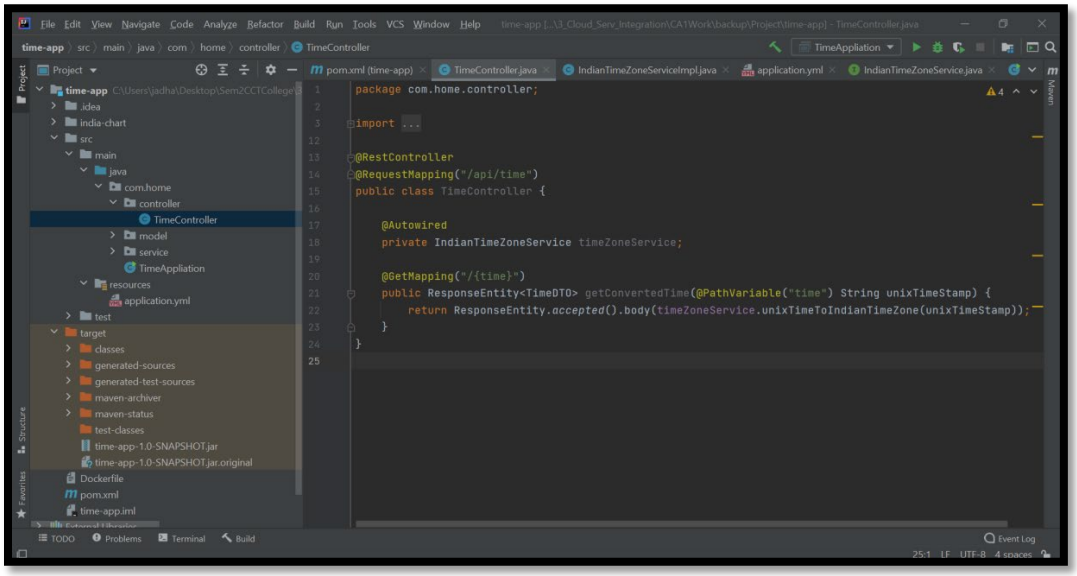
As per the requirments I have developed three microservices, out of which two microservices are for the two time zones (India & US time zone) and one is aggregator service which will merge the two time zone services.

The aggregator would be receiving GET request in **"GET /<timestamp>/time"** format**.**

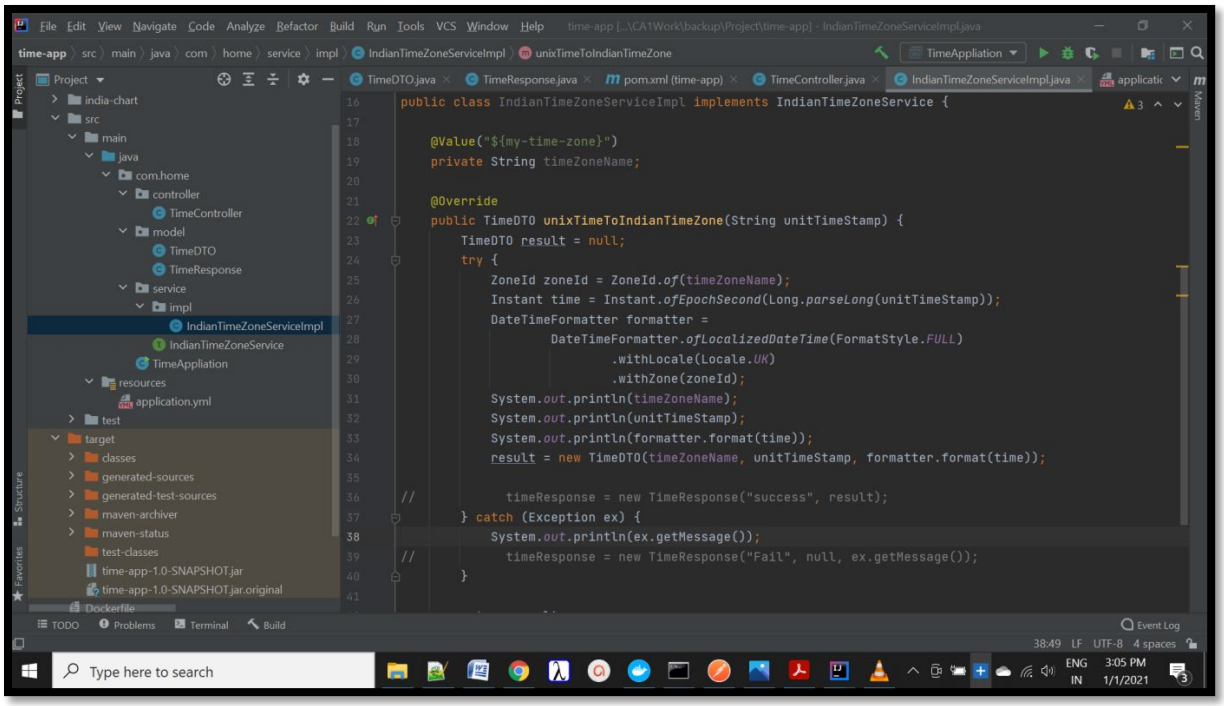
- Created the spring boot maven project time-app (India-time service).
- Pom.xml file with all the properties and spring boot dependency.



- Here, this is our REST controller class in that we actually creating an indian time API.
- Getting request from outside.

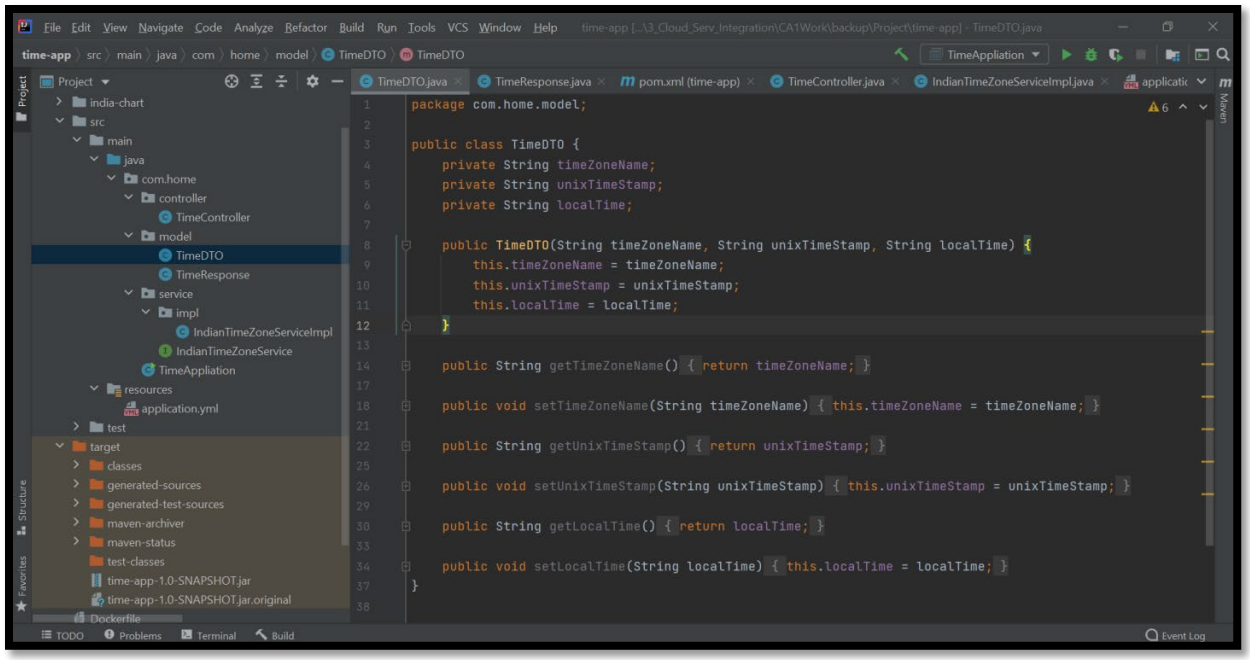


- Then it will call our service to perform main logic by using date time formatter.



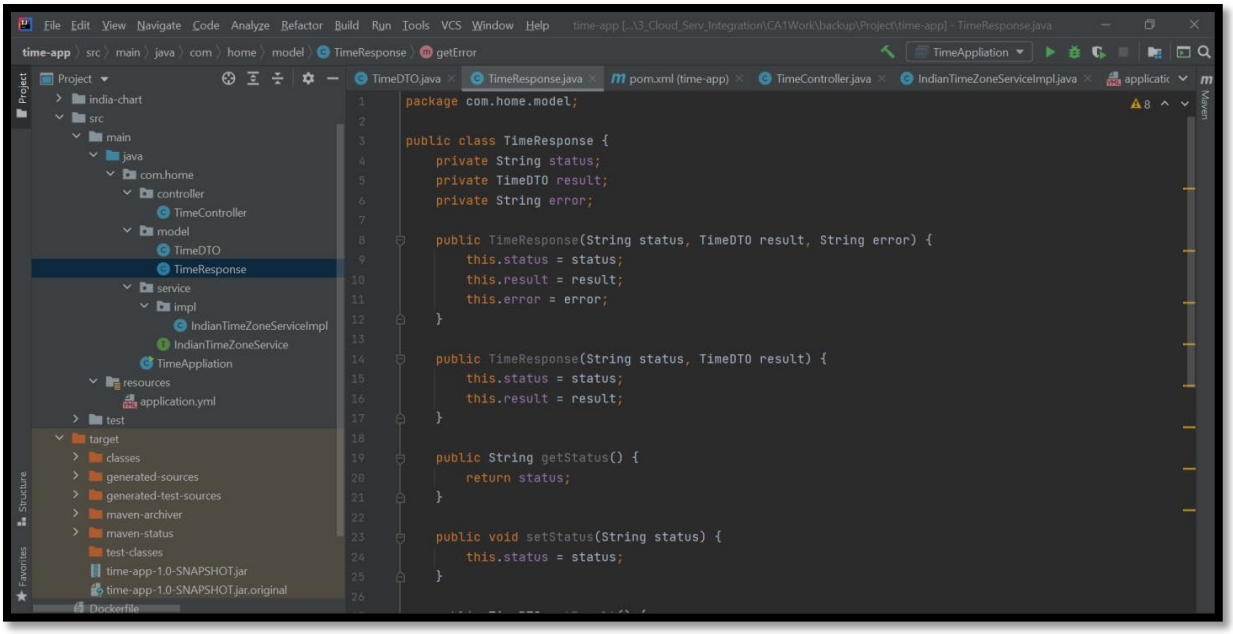
```
public class IndianTimeZoneServiceImpl implements IndianTimeZoneService {

    @Value("${my-time-zone}")
    private String timeZoneName;

    @Override
    public TimeDTO unixTimeToIndianTimeZone(String unitTimeStamp) {
        TimeDTO result = null;
        try {
            ZoneId zoneId = ZoneId.of(timeZoneName);
            Instant time = Instant.ofEpochSecond(Long.parseLong(unitTimeStamp));
            DateTimeFormatter formatter =
                    DateTimeFormatter.ofLocalizedDateTime(FormatStyle.FULL)
                            .withLocale(Locale.UK)
                            .withZone(zoneId);
            System.out.println(timeZoneName);
            System.out.println(unitTimeStamp);
            System.out.println(formatter.format(time));
            result = new TimeDTO(timeZoneName, unitTimeStamp, formatter.format(time));

//          timeResponse = new TimeResponse("success", result);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
//          timeResponse = new TimeResponse("Fail", null, ex.getMessage());
        }
```

- We are passing some values to outside .

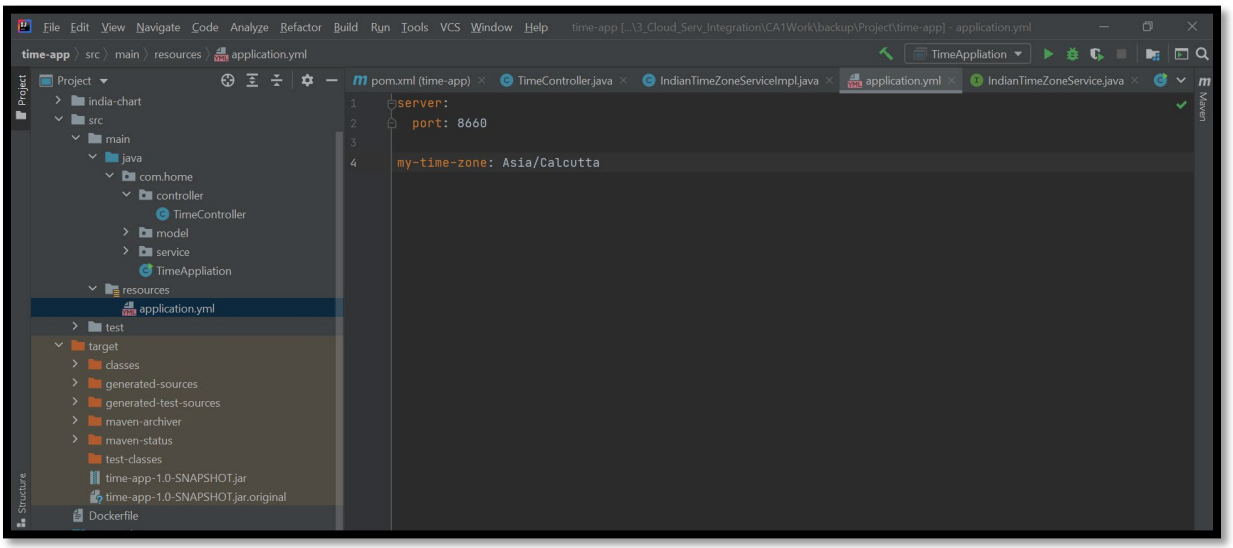


```
package com.home.model;

public class TimeDTO {
    private String timeZoneName;
    private String unixTimeStamp;
    private String localTime;

    public TimeDTO(String timeZoneName, String unixTimeStamp, String localTime) {
        this.timeZoneName = timeZoneName;
        this.unixTimeStamp = unixTimeStamp;
        this.localTime = localTime;
    }

    public String getTimeZoneName() { return timeZoneName; }

    public void setTimeZoneName(String timeZoneName) { this.timeZoneName = timeZoneName; }

    public String getUnixTimeStamp() { return unixTimeStamp; }

    public void setUnixTimeStamp(String unixTimeStamp) { this.unixTimeStamp = unixTimeStamp; }

    public String getLocalTime() { return localTime; }

    public void setLocalTime(String localTime) { this.localTime = localTime; }
}
```

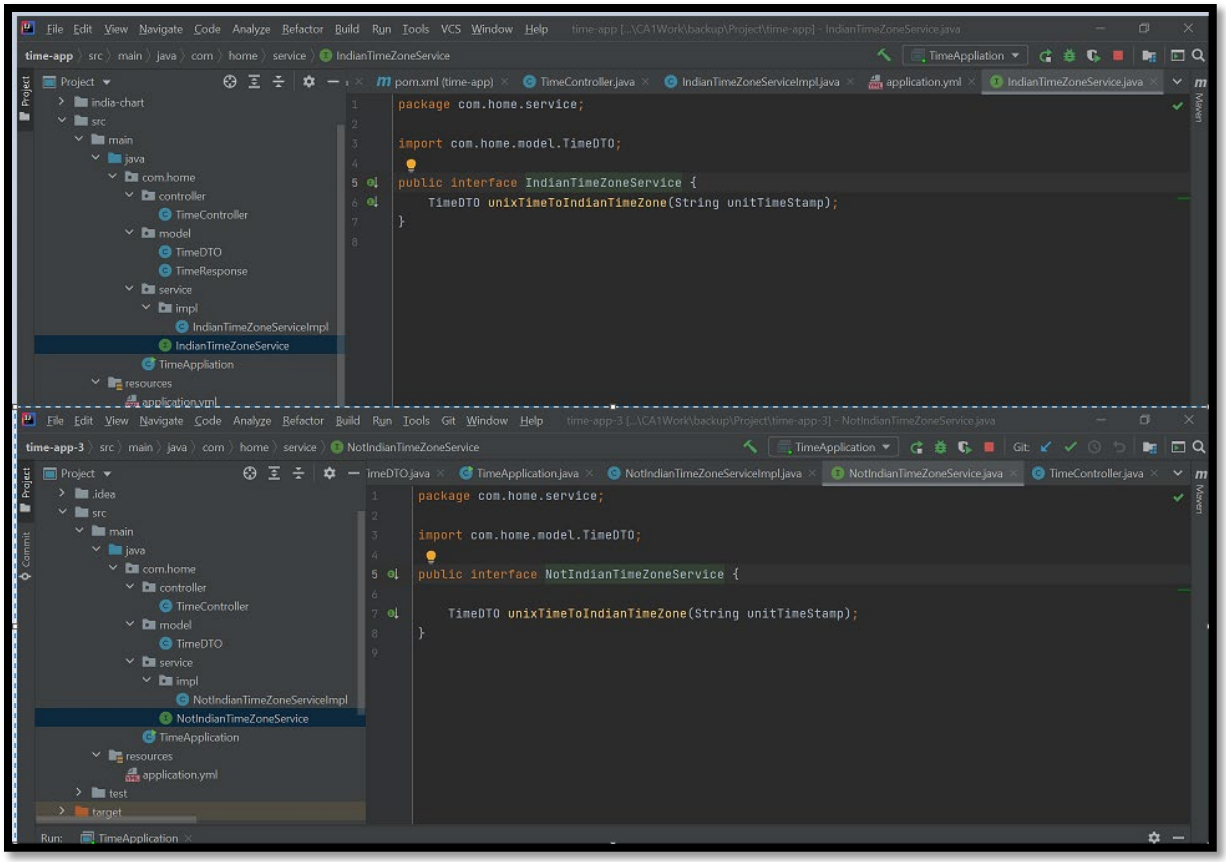
- This is the data we are actually passing outside to user like status, result and error and forward this reponse to controller class in the end.



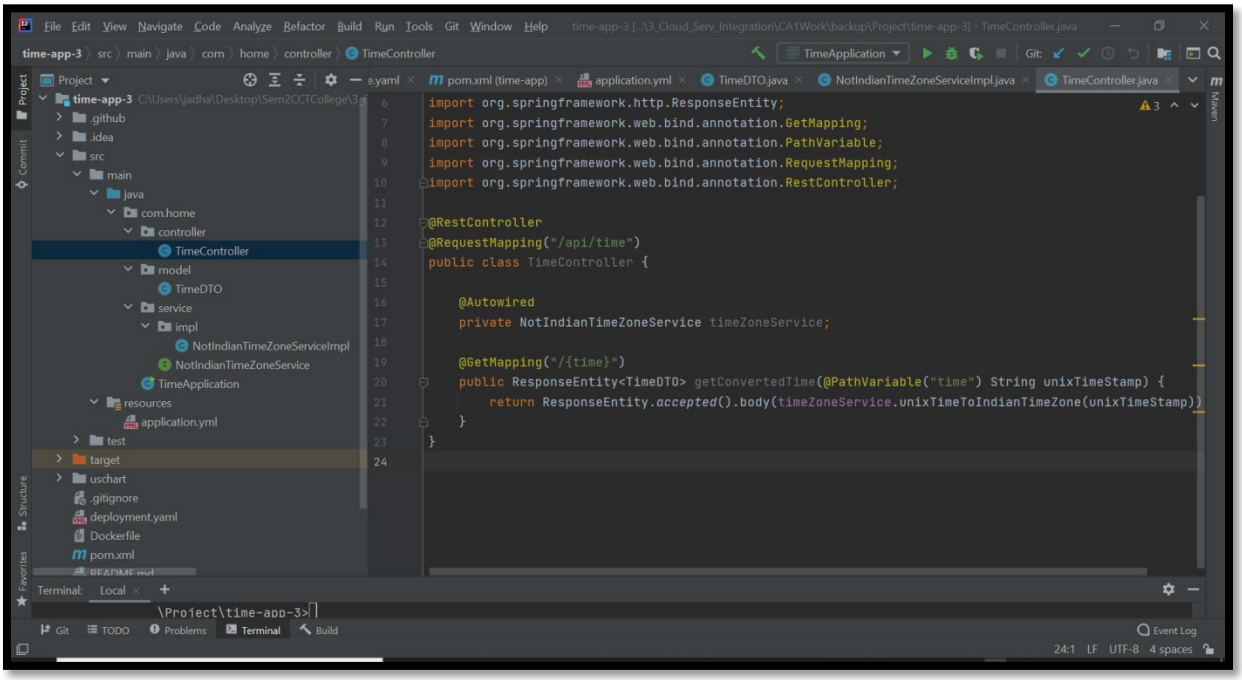
- In spring boot there is a feature that we can store some parameters in yml file as shown below.
- Here we mentioned the time zone name.

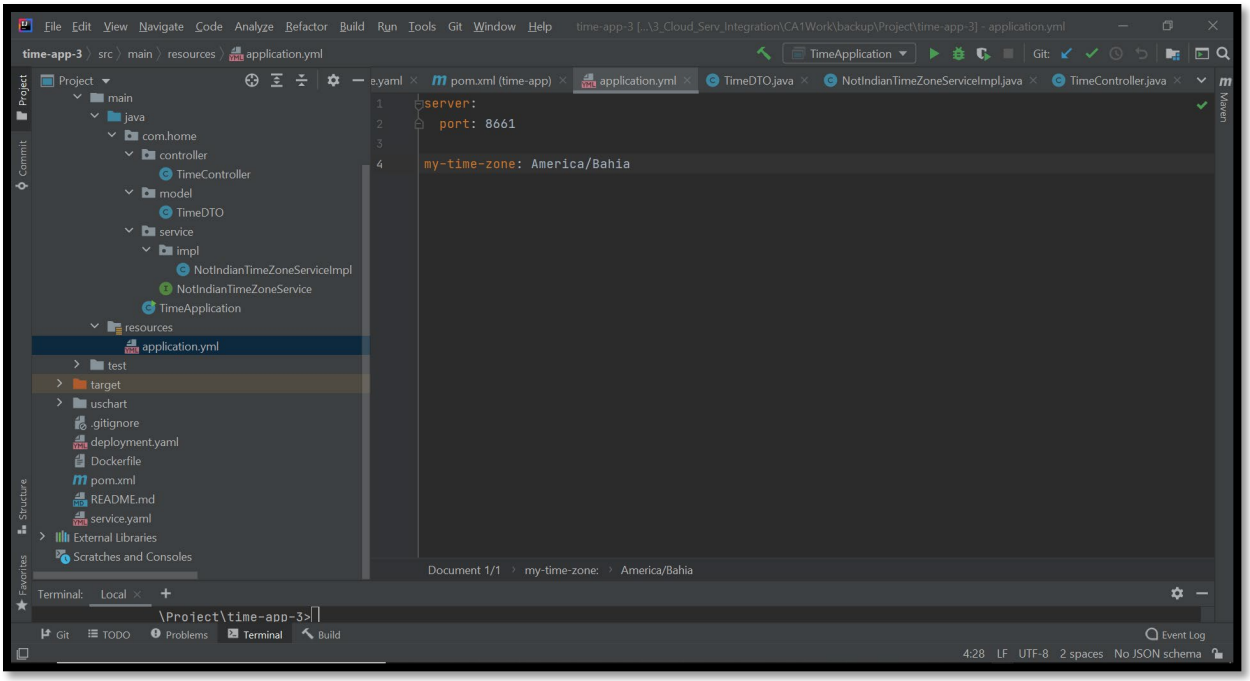


- Same as for US time servicegiven below.
- Interface for both services

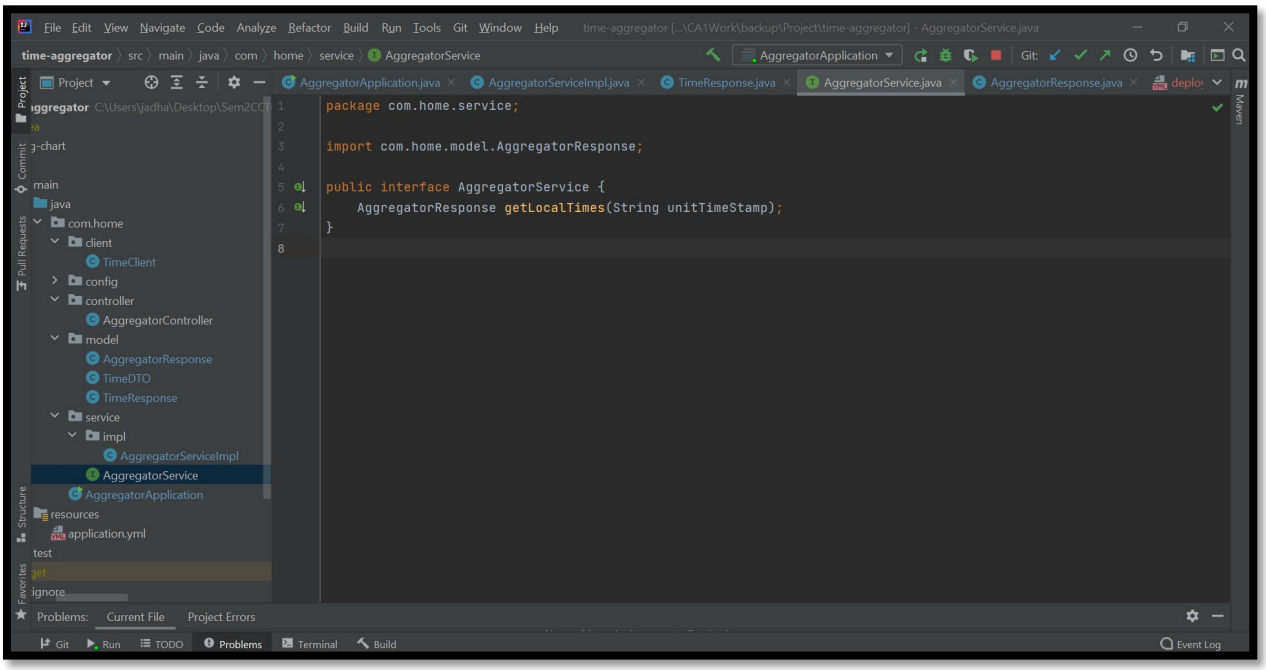


- US time service is exactly same as Indian time service except application.yml
- Below two are the snapshots of US time zone service.

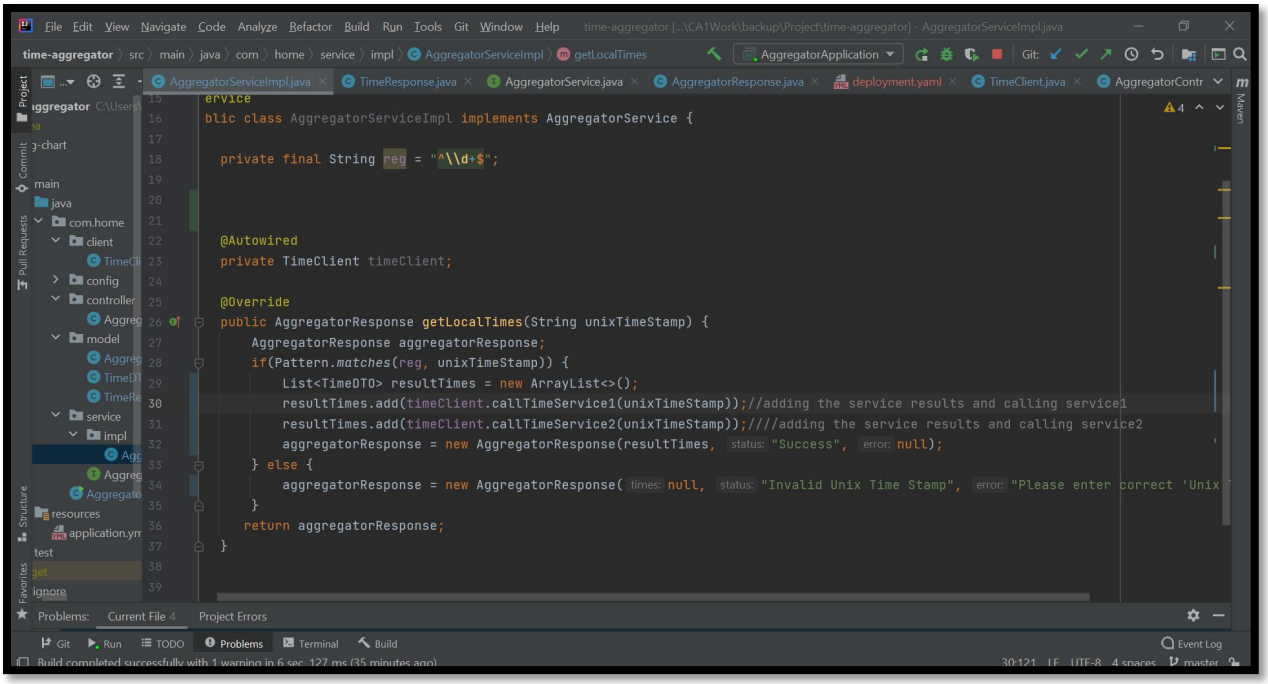




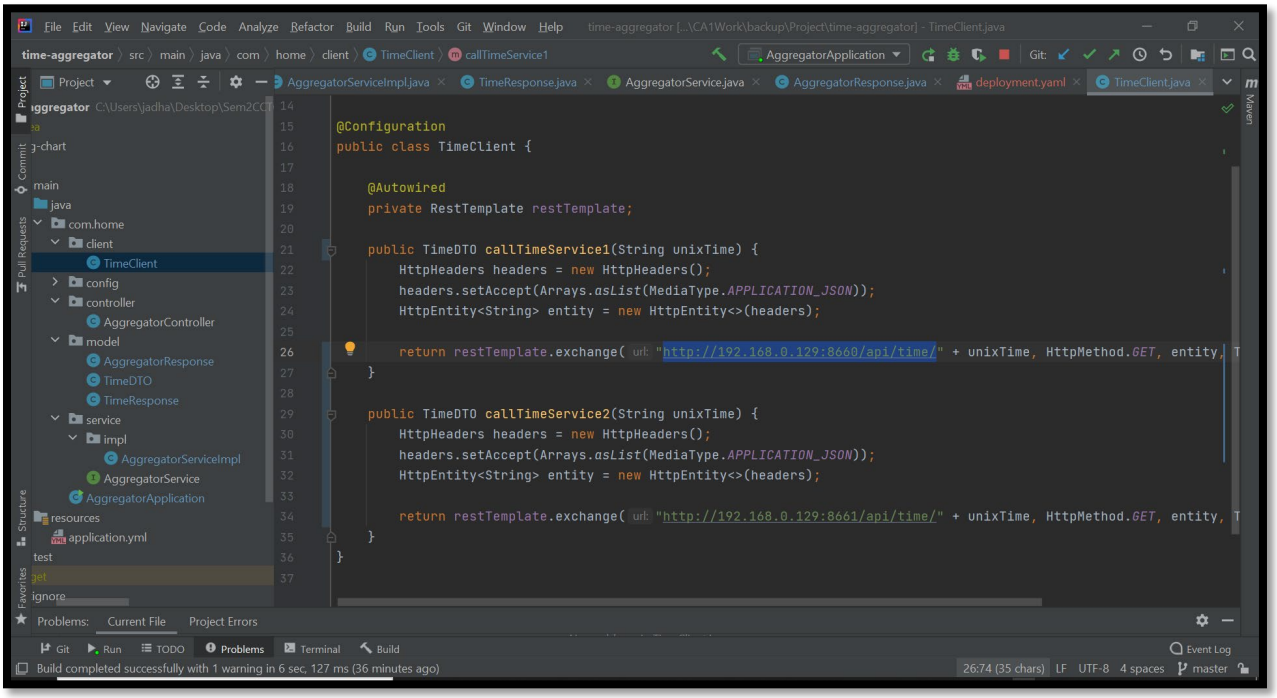
- Created aggregator service which will merge the response of two time zone services.
- Interface for aggregator.



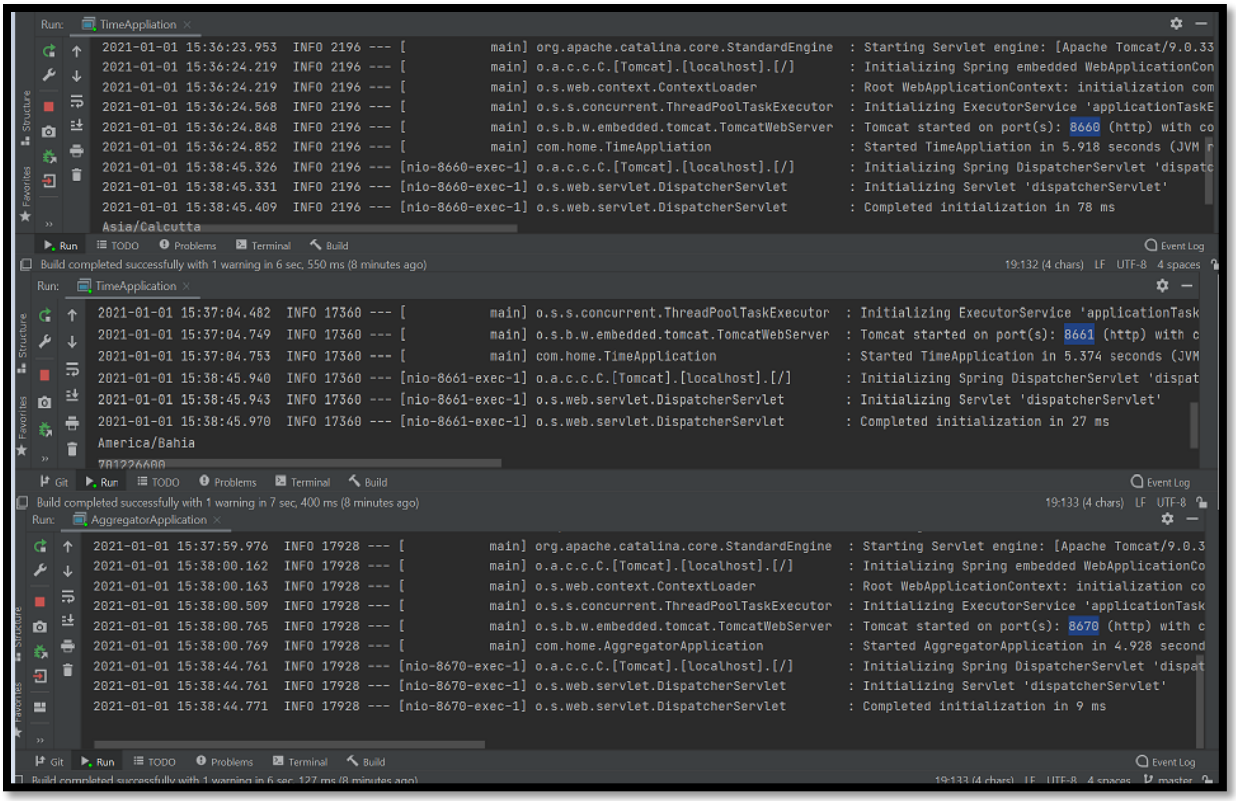
- This is the main buisness logic, in that we are validating the inputstrings using regex validator (regular expression).
- We are getting the response from the service in format of time response class and created the list of that objects.



- Here we need to pass the response time and it will automatically convert that result to this object.

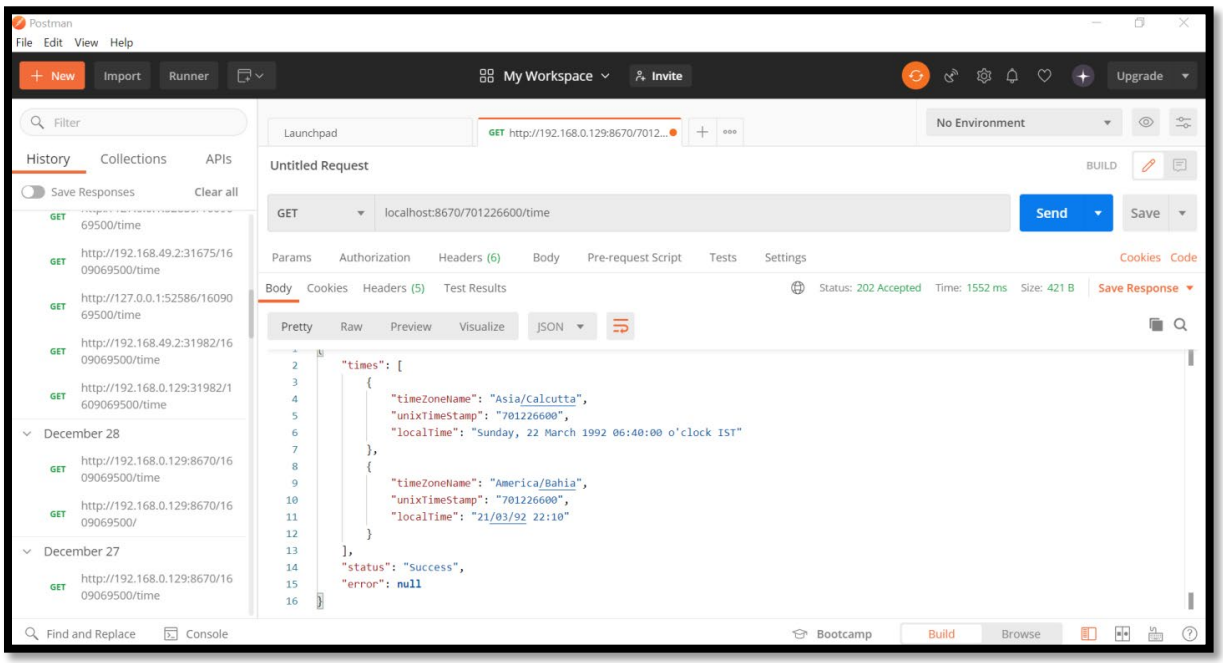


- Started all three services one by one.



Output:

- The output below shows the converted Indian time zone and the US time zone (different formats) with status and error details.

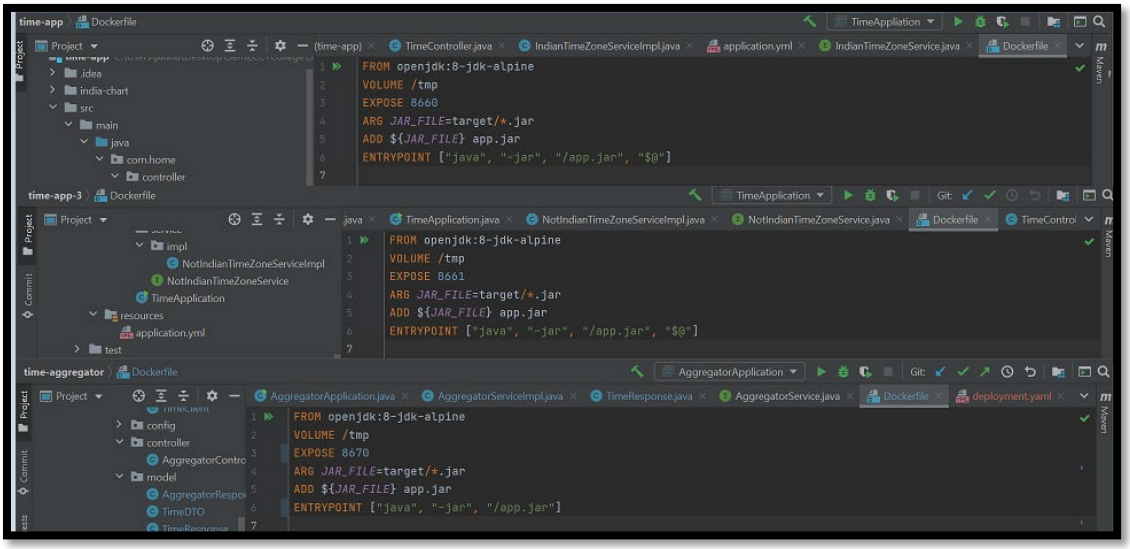


3. Docker Activities:

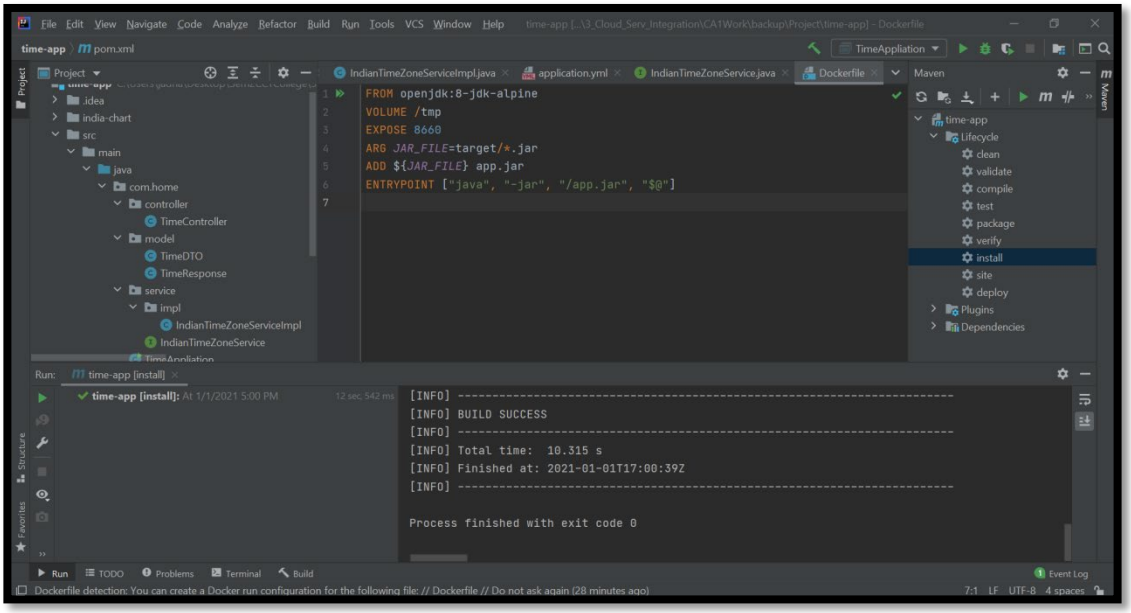
We use Docker containers to create, deploy, and execute different applications. These containers are nothing but the packaged application with all required libraries and dependencies to run on any platform.

So basically our application starts and stops from Docker without any manual intervention. Everything will be taken care of by Docker itself.

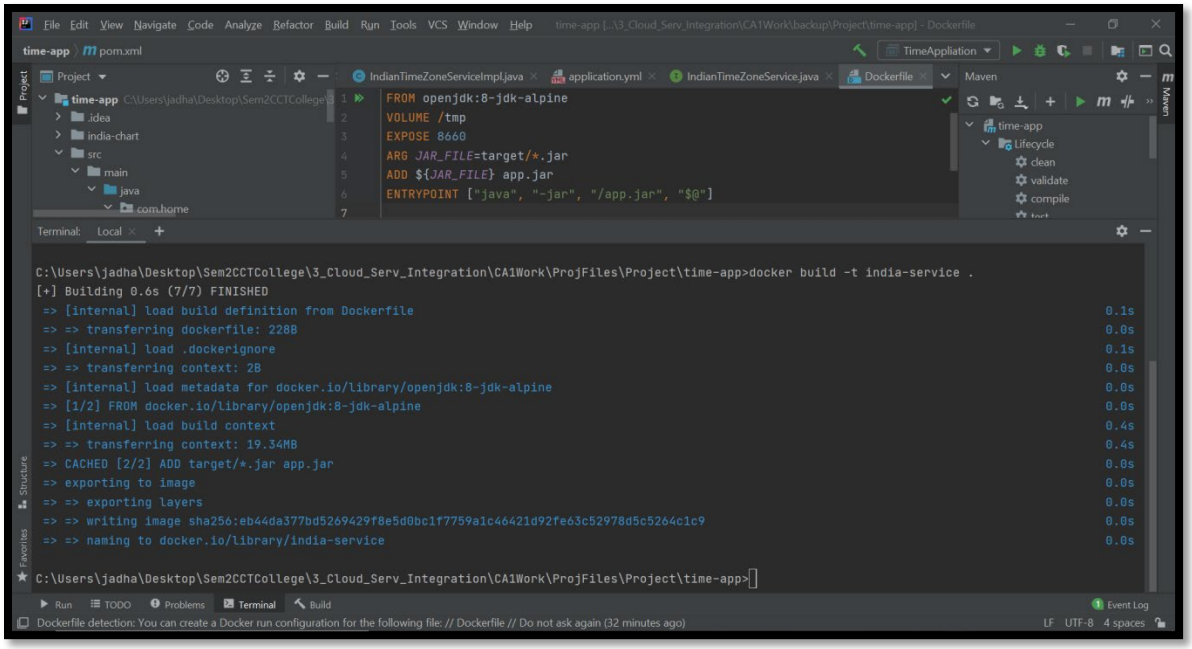
- Dockerfile for all three services.



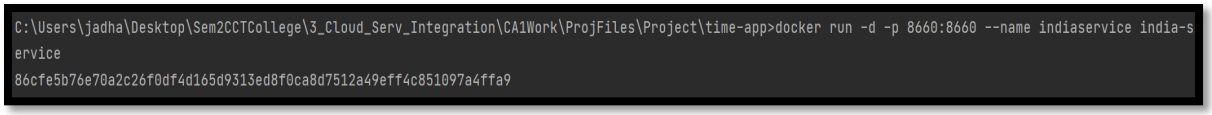
- Build the indian time service projectto create docker image.



- Docker build command for building docker image.

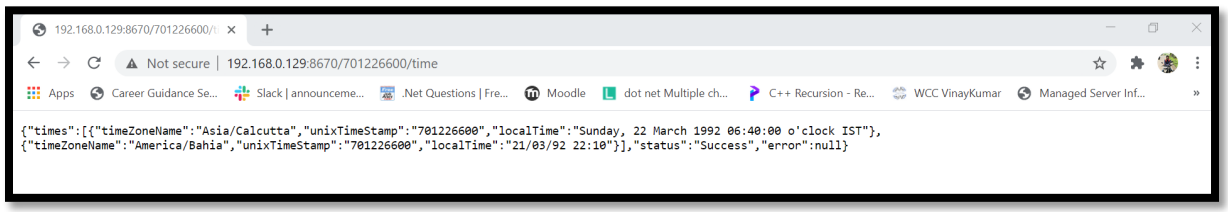


- Run the docker container in background mode (-d) and publish the port number on host (-p).



- Perform all the same steps for US time service and aggregator service remaining service.
- Now we have all three docker images available up and running.

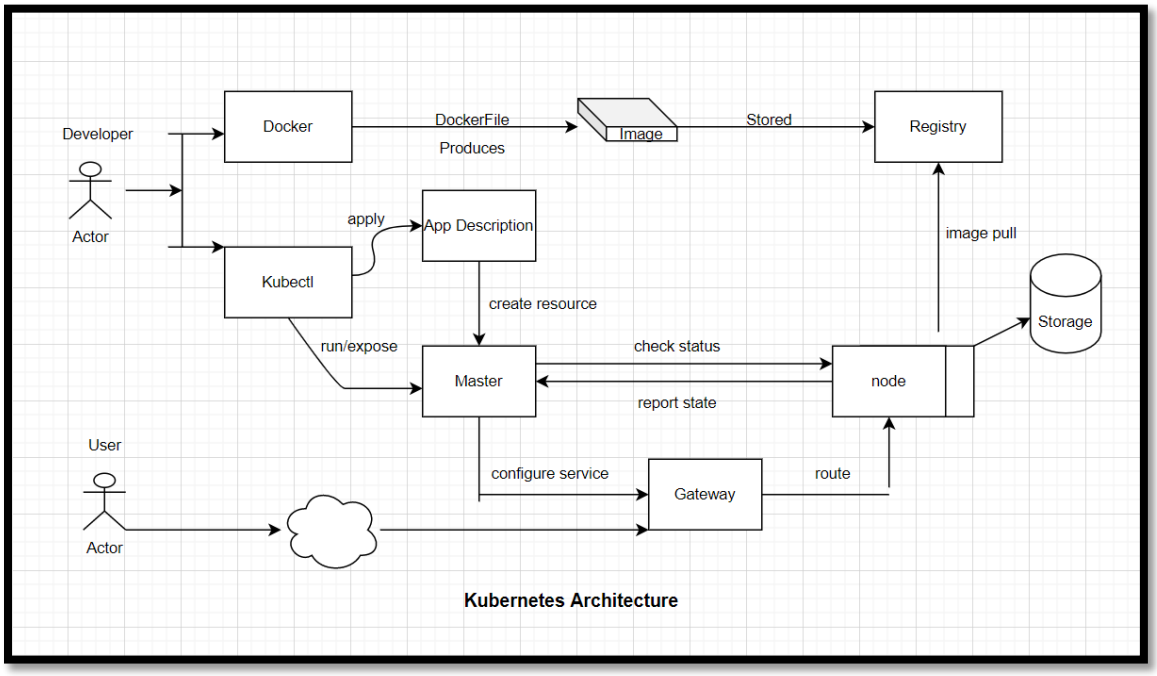
Output:



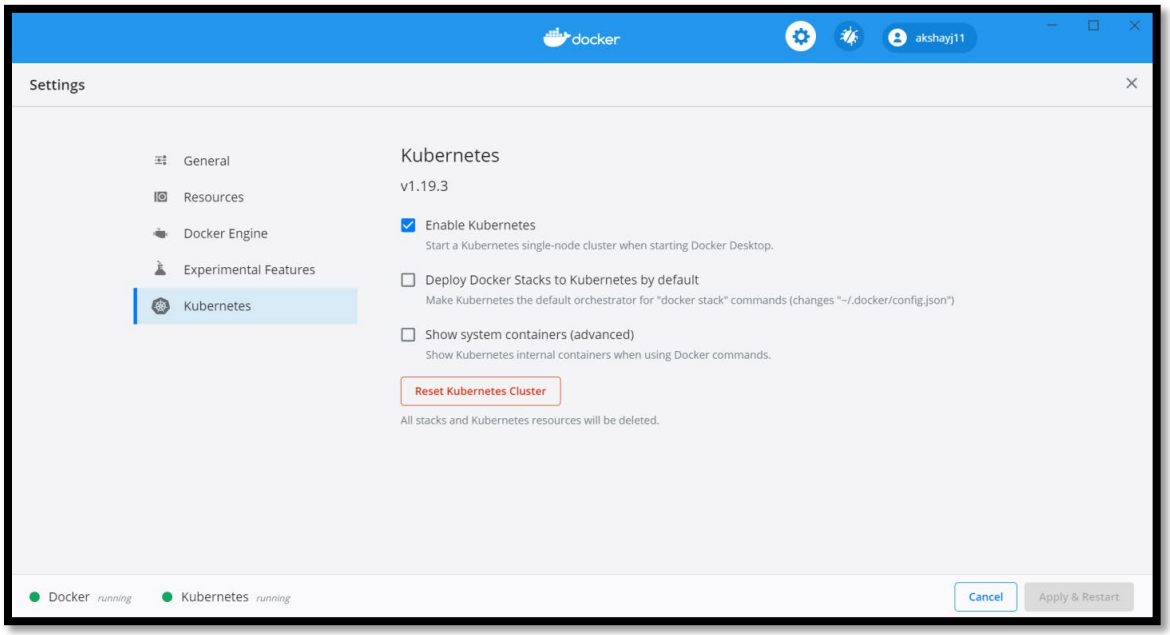
4. Kubernetes Activities:

The Kubernetes use over Docker for container orchestration activities. It helps in automating the deployment, scaling, and management of applications. The main parts of Kubernetes are deployment and service YAML files.

- Created the basic Kubernetes working diagram for better understanding.

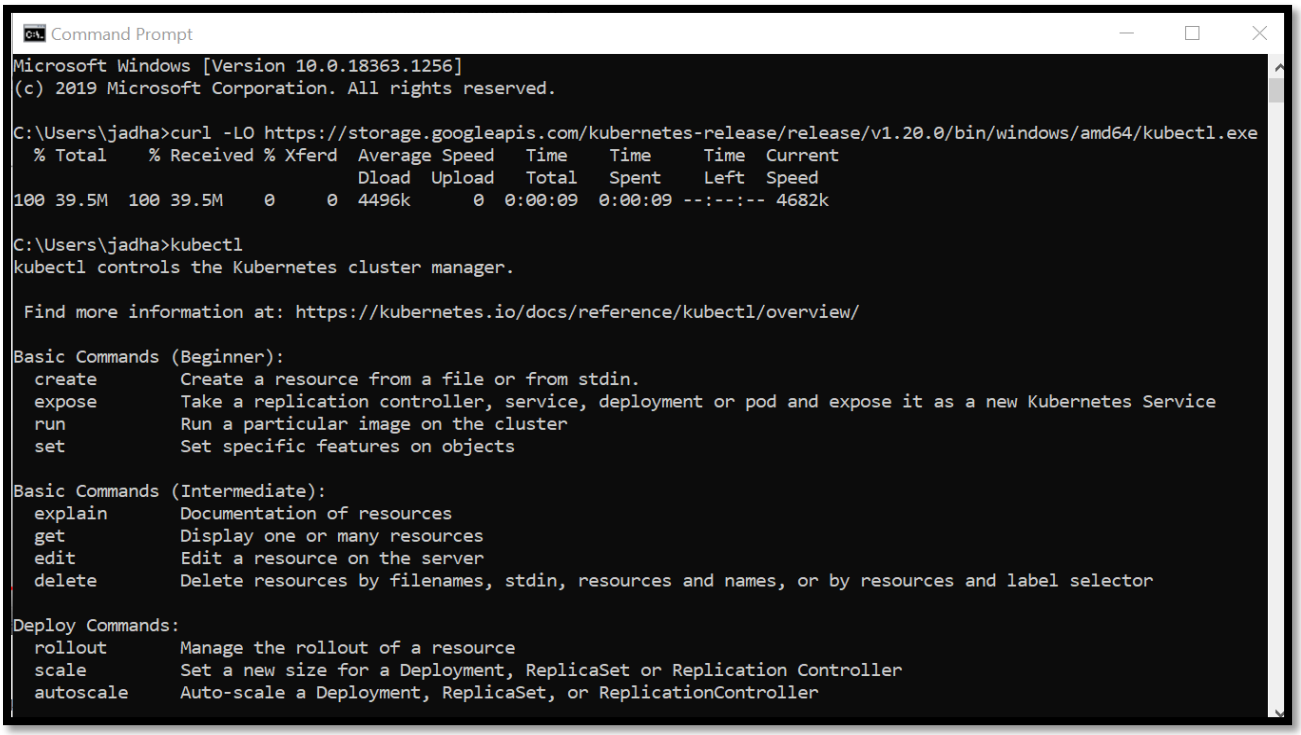


- First enable Kubernetes from Docker desktop.



- **Kubectl and Minikube Setup:**

Minikube will allow us to run a Kubernets cluster locally we will set up a Kubernetes single node cluster environment locally using minikube and to perform application deployment, inspection, cluster management we will install kubectl command-line tool.



- Minikubedownloaded and installed.

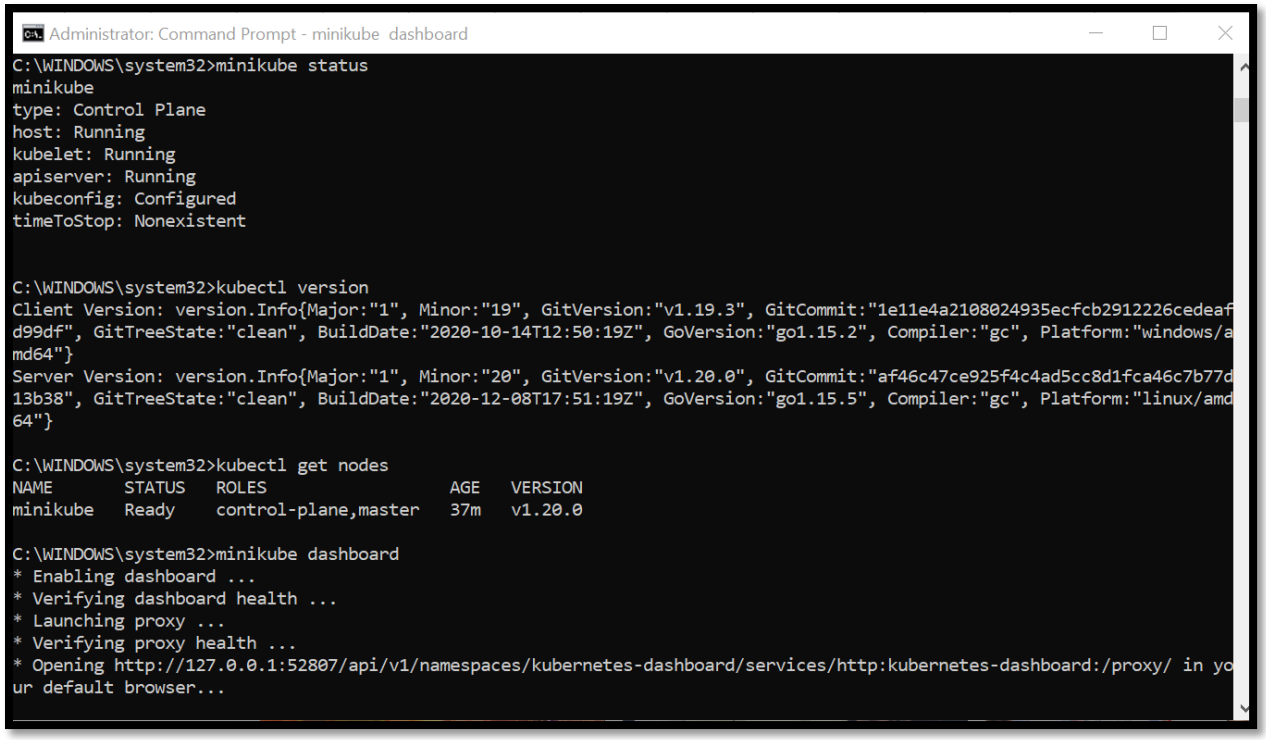
- Error while starting minikube cluster with “minikube start” command.

```
C:\WINDOWS\system32>minikube start --vm-driver=virtualbox
* minikube v1.16.0 on Microsoft Windows 10 Home Single Language 10.0.18363 Build 18363
* Using the virtualbox driver based on user configuration
* Downloading VM boot image ...
    > minikube-v1.16.0.iso.sha256: 65 B / 65 B [-------------] 100.00% ? p/s 0s
    > minikube-v1.16.0.iso: 212.62 MiB / 212.62 MiB [] 100.00% 4.02 MiB p/s 53s
* Starting control plane node minikube in cluster minikube
* Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
! StartHost failed, but will try again: creating host: create: precreate: This computer doesn't have VT-X/AMD-v enabled.
 Enabling it in the BIOS is mandatory
* Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
* Failed to start virtualbox VM. Running "minikube delete" may fix it: creating host: create: precreate: This computer d
oesn't have VT-X/AMD-v enabled. Enabling it in the BIOS is mandatory

X Exiting due to HOST_VIRT_UNAVAILABLE: Failed to start host: creating host: create: precreate: This computer doesn't ha
ve VT-X/AMD-v enabled. Enabling it in the BIOS is mandatory
* Suggestion: Virtualization support is disabled on your computer. If you are running minikube within a VM, try '--drive
r=docker'. Otherwise, consult your systems BIOS manual for how to enable virtualization.
* Related issues:
  - https://github.com/kubernetes/minikube/issues/3900
  - https://github.com/kubernetes/minikube/issues/4730
```

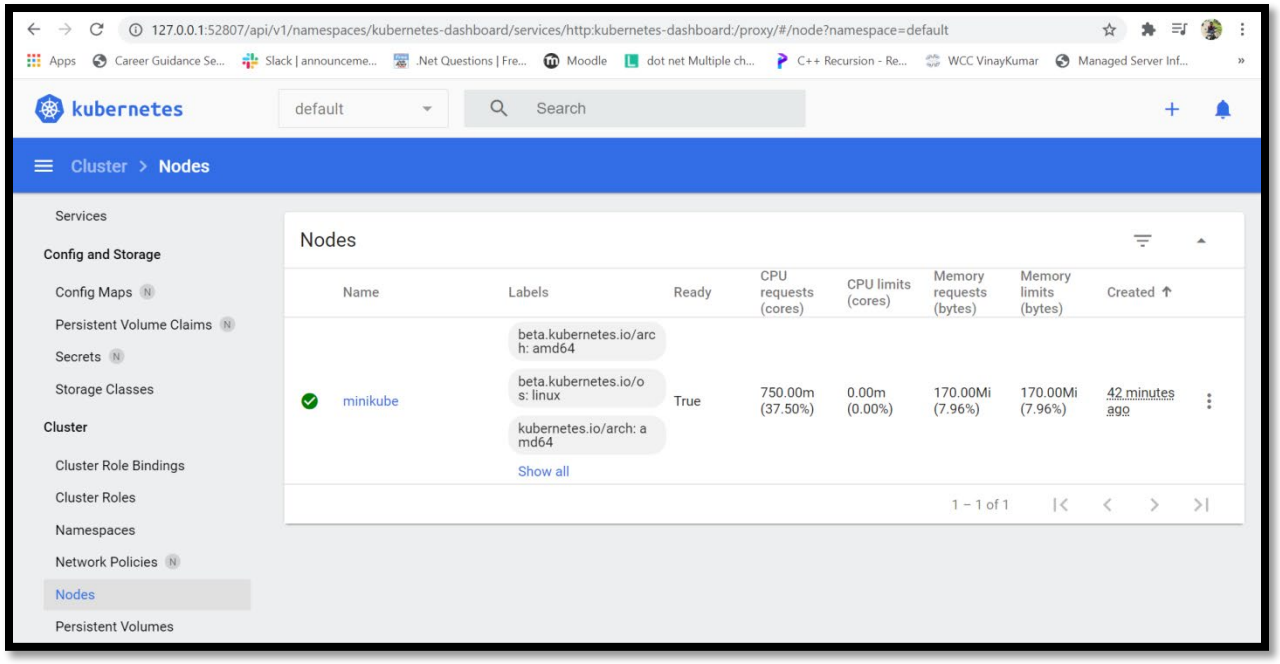
- Instead, I used “minikube start –no-vtx-check” command and started successfully as we can see the virtual machine is getting created as shown below.

```
Administrator: Command Prompt
  - https://github.com/kubernetes/minikube/issues/4730


C:\WINDOWS\system32>minikube start --no-vtx-check
* minikube v1.16.0 on Microsoft Windows 10 Home Single Language 10.0.18363 Build 18363
* Using the virtualbox driver based on existing profile
* Starting control plane node minikube in cluster minikube
* Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
* Preparing Kubernetes v1.20.0 on Docker 20.10.0 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Verifying Kubernetes components...
* Enabled addons: default-storageclass, storage-provisioner
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

C:\WINDOWS\system32>minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
timeToStop: Nonexistent


C:\WINDOWS\system32>
```

- Checked the minikube status, kubectl version, kubectl nodes and opened minikube dashboard.



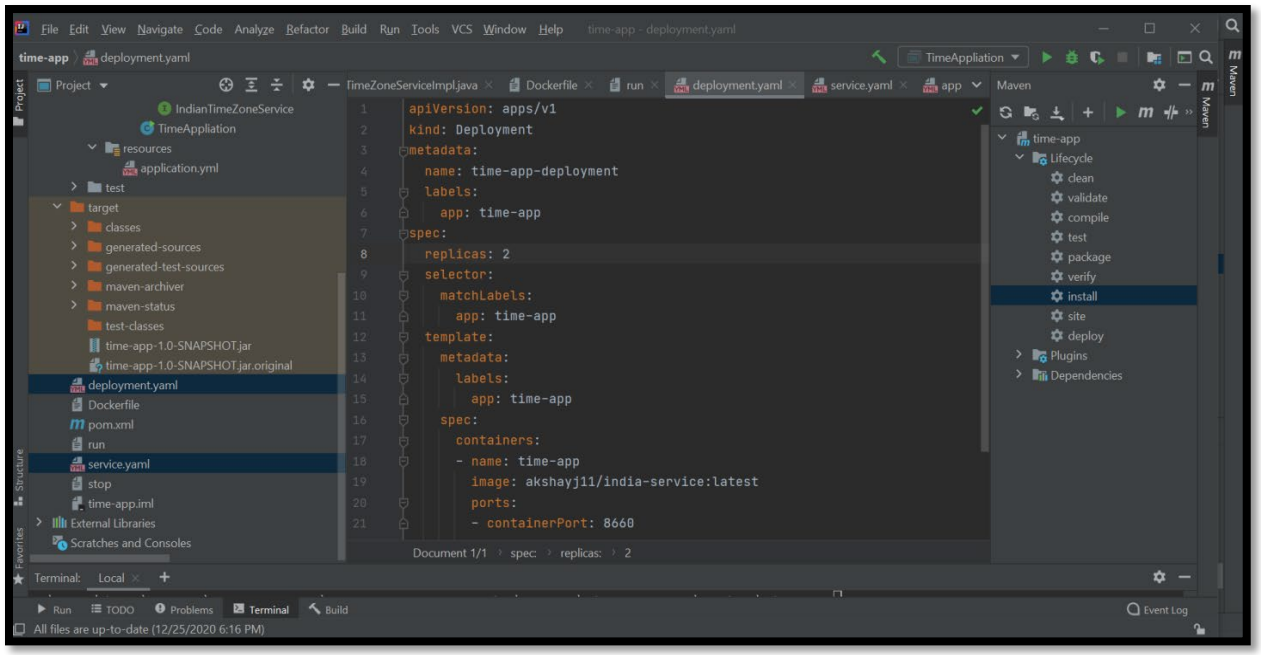
- Minikube is working fine as shown below.



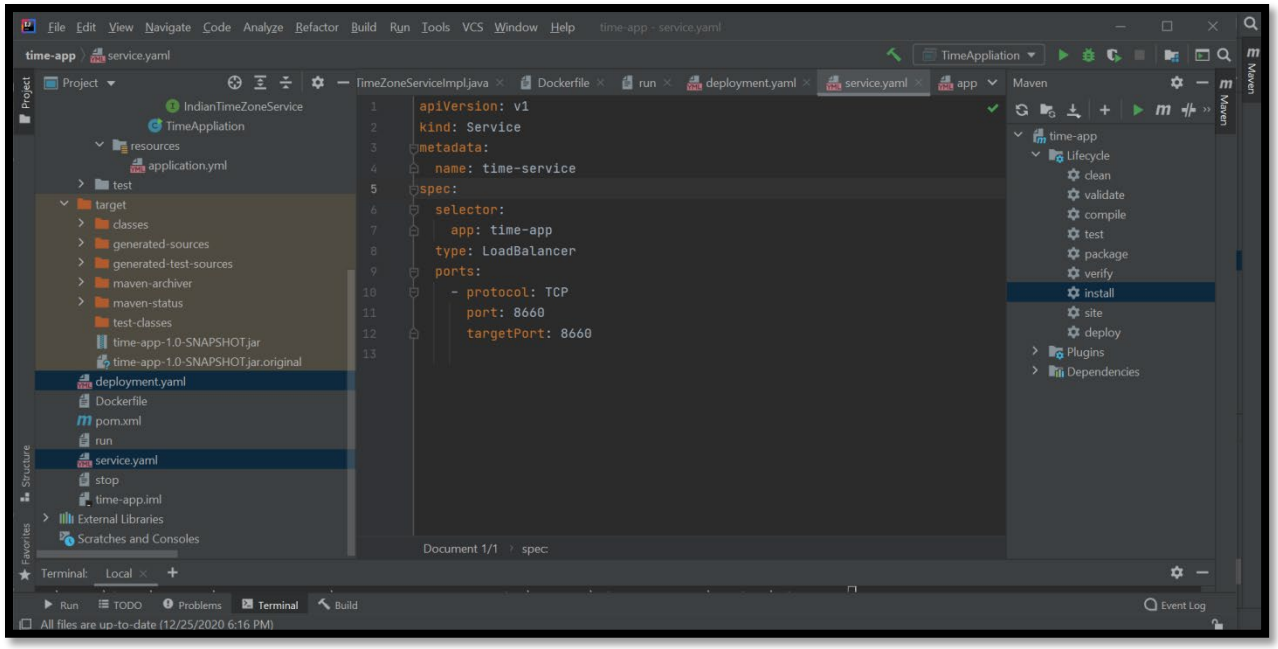
- Pushed the Docker images of all three services to the Docker hub public repository called micro-service.

```
C:\WINDOWS\system32>docker tag eb44da377bd5 akshayj11/micro_services:indiatime

C:\WINDOWS\system32>docker push akshayj11/micro_services:indiatime
The push refers to repository [docker.io/akshayj11/micro_services]
211fb8fb639b: Mounted from akshayj11/india-service
ceaf9e1ebef5: Layer already exists
9b9b7f3d56a0: Layer already exists
f1b5933fe4b5: Layer already exists
indiatime: digest: sha256:1f07e1abc2b2e5fd242839b62381f725585486402a4146045517e16576ad9a1c size: 1159

C:\WINDOWS\system32>docker tag 0bc8dd48d6a4 akshayj11/micro_services:agg-service

C:\WINDOWS\system32>docker push akshayj11/micro_services:agg-service
The push refers to repository [docker.io/akshayj11/micro_services]
81b5bcd26f70: Pushed
ceaf9e1ebef5: Layer already exists
9b9b7f3d56a0: Layer already exists
f1b5933fe4b5: Layer already exists
agg-service: digest: sha256:814a45d89d16225b3b252b81bb5d9b92052d379bc6473a7c3666d12b5996b5c5 size: 1159

C:\WINDOWS\system32>
```

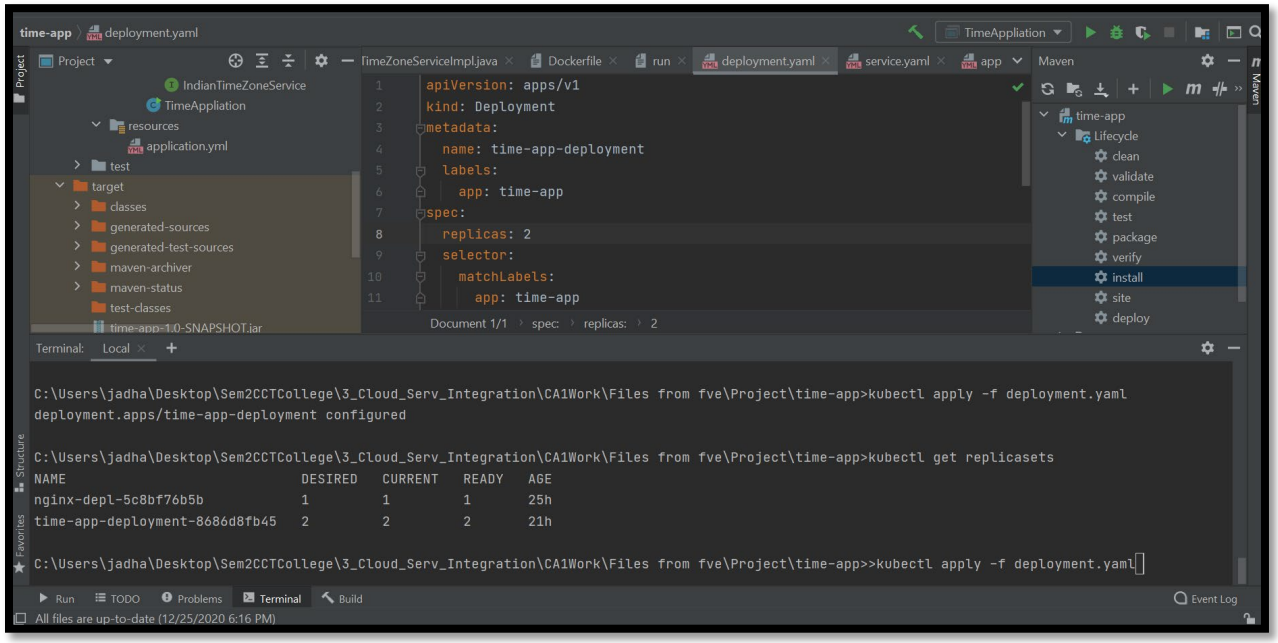
- Created docker deployment YAML file for India – time service.



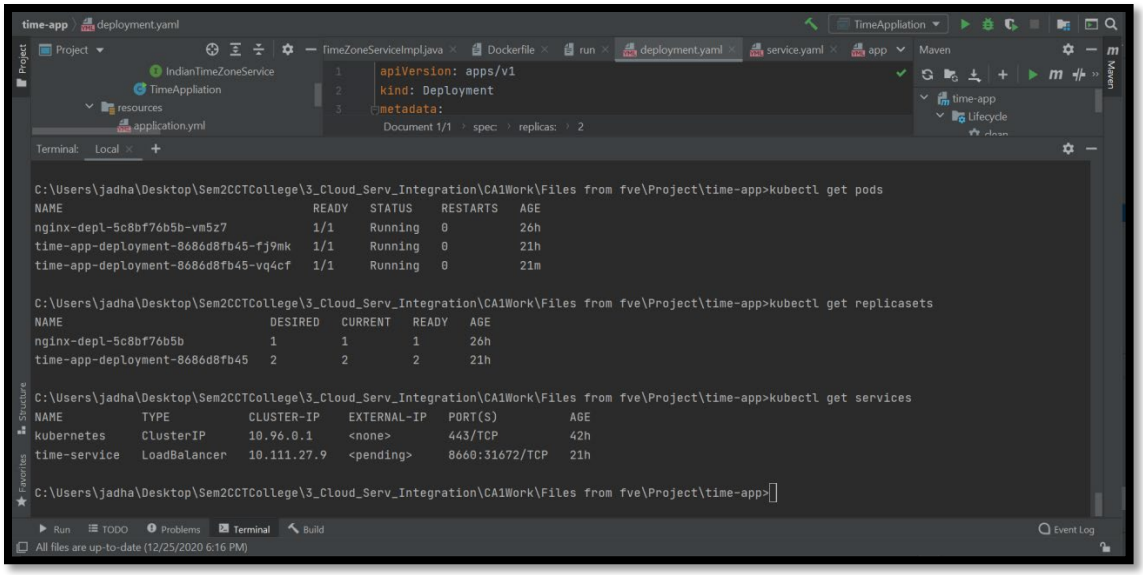
- Created service YAML file for India-time service with “LoadBalancer” type (later on changed to NodeIP).



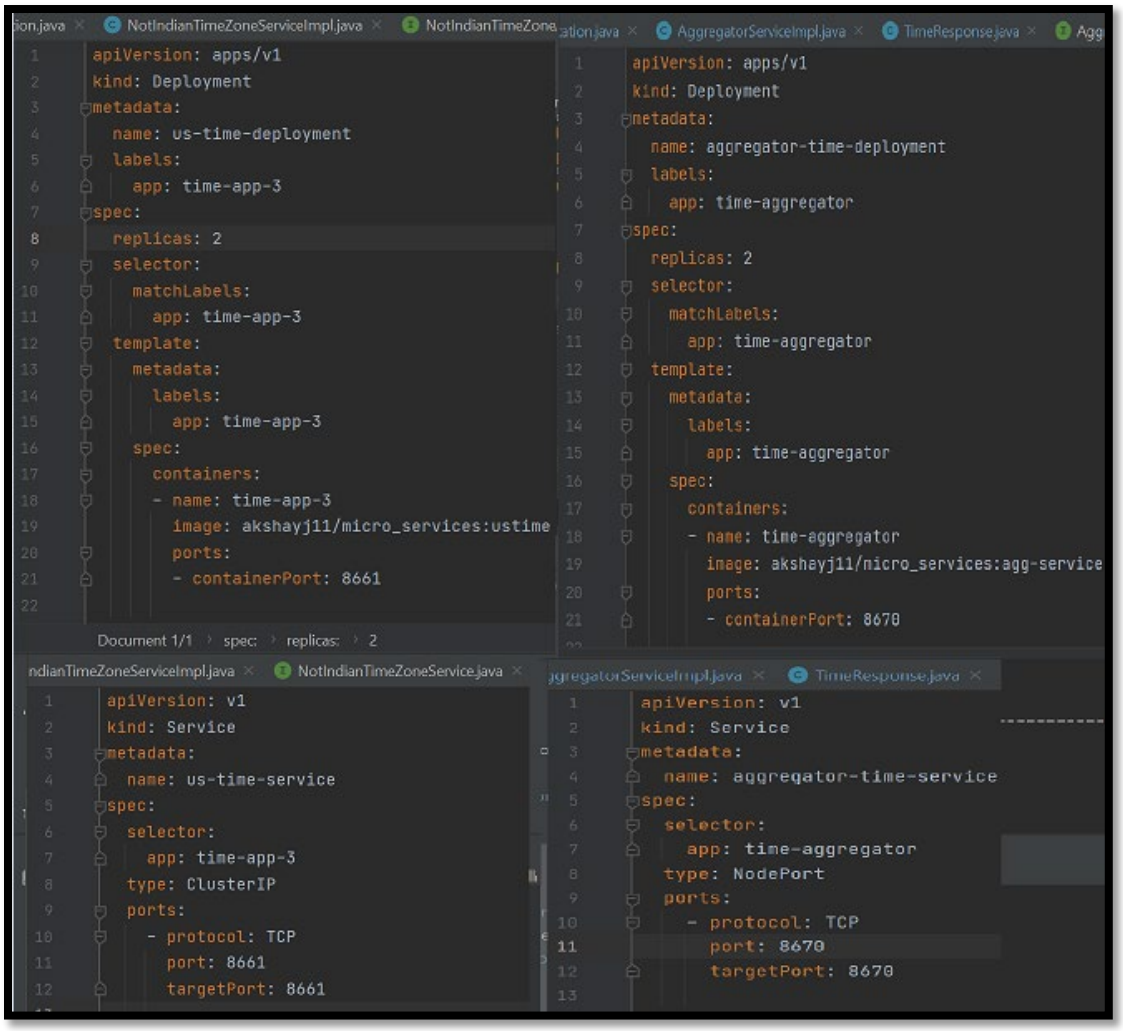
- Applied deployment and service YAMLin Kubernetes using "kubectl apply –f deployment.yaml" command.



- You can see two replica set means two pods running in Kubernetes for this service.



- YAML files of both US time and aggregator services.



Applied deployment and service YAML in Kubernetes for both US time and aggregator service using the same command as stated earlier.

All the deployment and services to access these time service has been placed on Kubernetes cluster.

- Check using “kubectl get deployment” and “kubectl get services”.

```
C:\Users\jadha\Desktop\Sem2CCTCollege\3_Cloud_Serv_Integration\CA1Work\ProjFiles\Project\time-aggregator>kubectl get deployment
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
nginx-depl                  1/1     1            1           27h
time-aggregator-deployment  2/2     2            2           42s
time-app-3-deployment       2/2     2            2           7m9s
time-app-deployment         2/2     2            2           22h

C:\Users\jadha\Desktop\Sem2CCTCollege\3_Cloud_Serv_Integration\CA1Work\ProjFiles\Project\time-aggregator>kubectl get services
NAME                     TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes               ClusterIP      10.96.0.1        <none>        443/TCP          44h
time-aggregator-service  LoadBalancer   10.100.4.42      <pending>     8670:30405/TCP   52s
time-app-3-service       LoadBalancer   10.109.184.203   <pending>     8661:30477/TCP   5m45s
time-service             LoadBalancer   10.111.27.9      <pending>     8660:31672/TCP   22h

C:\Users\jadha\Desktop\Sem2CCTCollege\3_Cloud_Serv_Integration\CA1Work\ProjFiles\Project\time-aggregator>
```

- Faced the API server minikube issue then deleted and started minikube again.

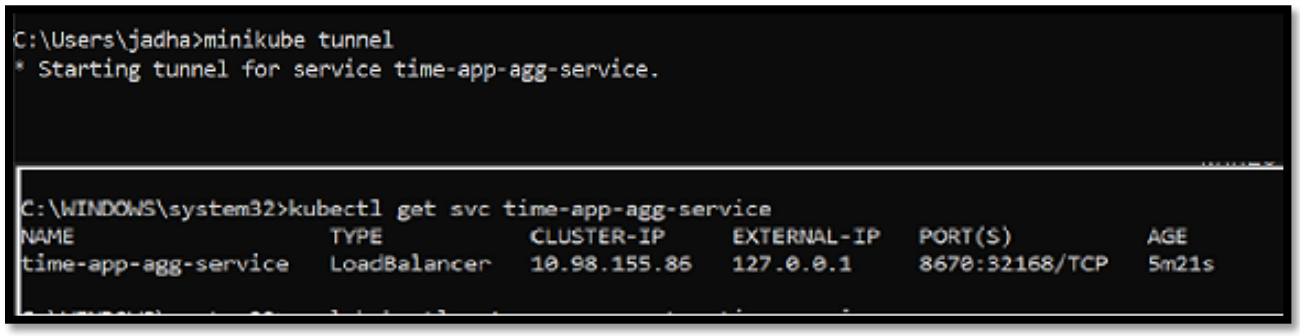
```
C:\>minikube delete --purge --all
* Removing C:\Users\jadha\.minikube\machines\minikube ...
* Removed all traces of the "minikube" cluster.
* Successfully deleted all profiles
* Successfully purged minikube directory located at - [C:\Users\jadha\.minikube]

C:\>minikube start --no-vtx-check
* minikube v1.16.0 on Microsoft Windows 10 Home Single Language 10.0.18363 Build 18363
* Automatically selected the docker driver. Other choices: hyperv, virtualbox
* Starting control plane node minikube in cluster minikube
* Downloading Kubernetes v1.20.0 preload ...
    > preloaded-images-k8s-v8-v1....: 360.05 MiB / 491.00 MiB  73.33% 1.88 MiB
```

We can expose the services type NodePort or Load Balancer by using the minikube tunnel command. It creates the network route from the host to the service which is on the cluster via cluster IP.

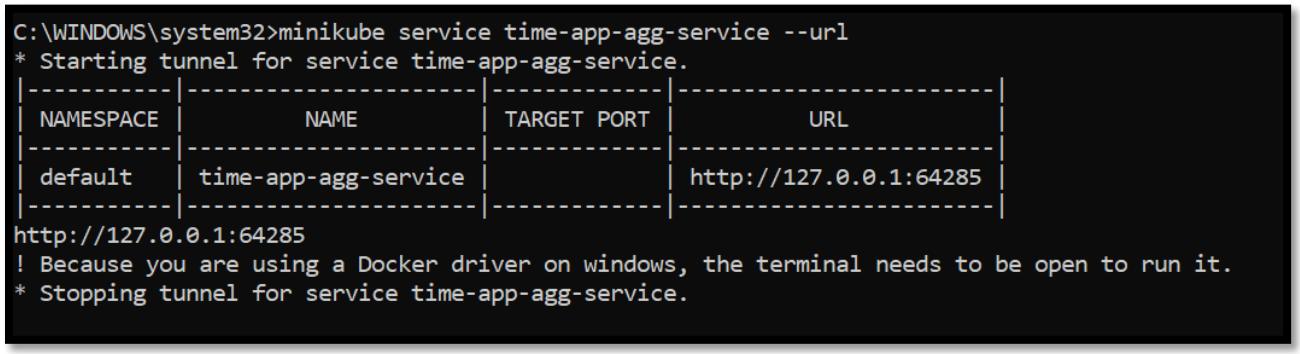
- As we can see the External IP status is pending without running the minikube tunnel.

```
Administrator: Command Prompt
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
* Stopping tunnel for service aggregator-time-service.

C:\WINDOWS\system32>kubectl get svc aggregator-time-service
NAME                       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
aggregator-time-service    NodePort   10.99.205.113   <none>        8670:30809/TCP   27h

C:\WINDOWS\system32>
```

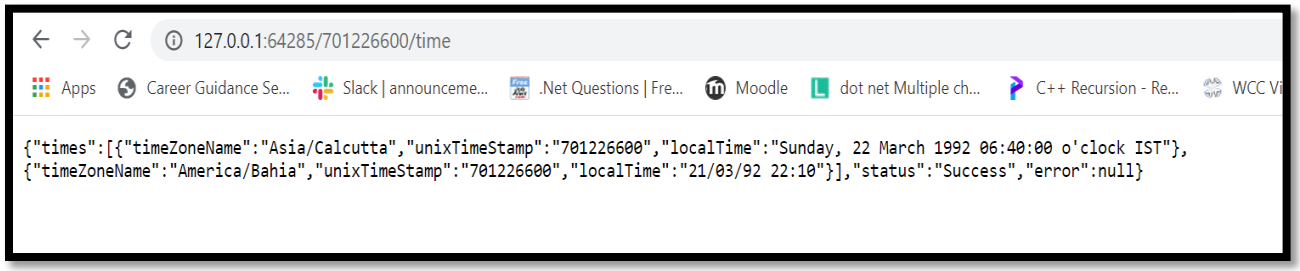
- After running the minikube tunnel in another terminal, we got the external IP of one LoadBalancer type service.



- Accessing the time aggregator LoadBalancer type service deployed in kubernetes.
- Use “minikube service time-app-agg-service –url” command.



Output:



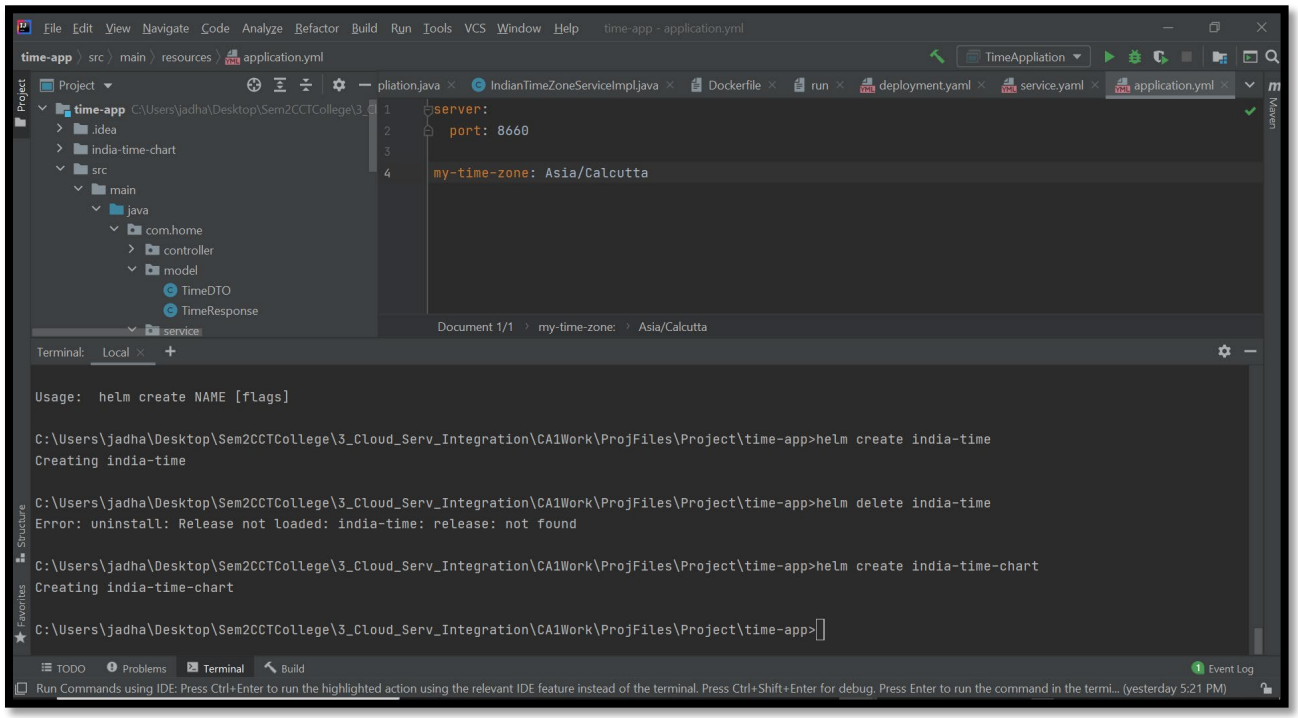
Successfully tested the application in Kubernetes environment.

5. Helm Chart

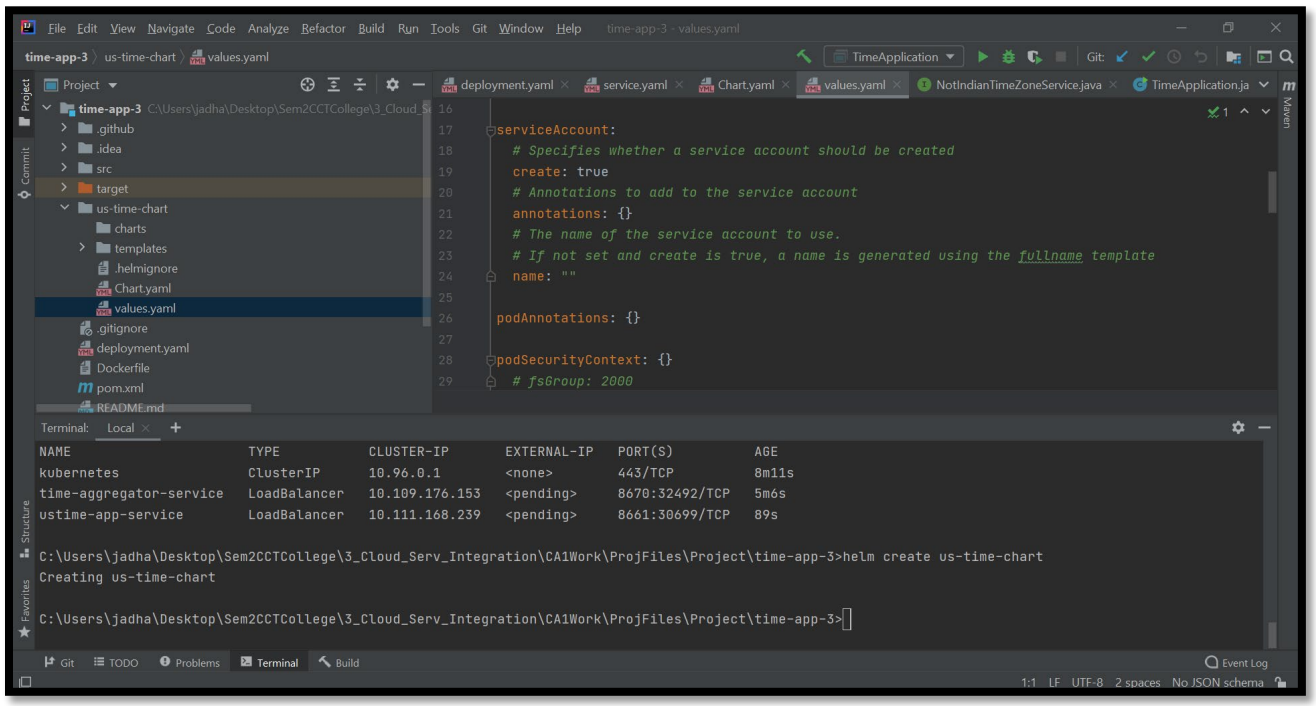
It is a Kubernetes package manager and the packages in that are called a chart.Mostly used to install, upgrade, and define the complex application as well as we can share and publish the helm charts.

- Installed the Helm and added to the path of environment variable.

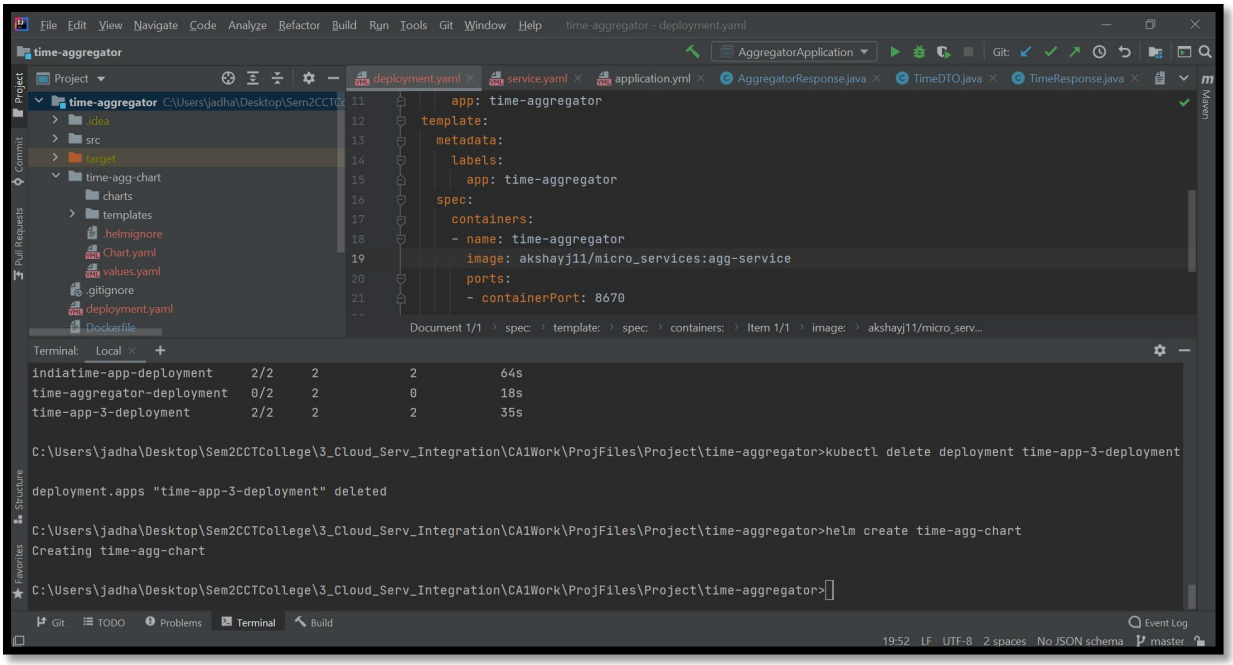
- Created the helm chart for API service (e.g., India-time service).
- Helm create India-time-chart



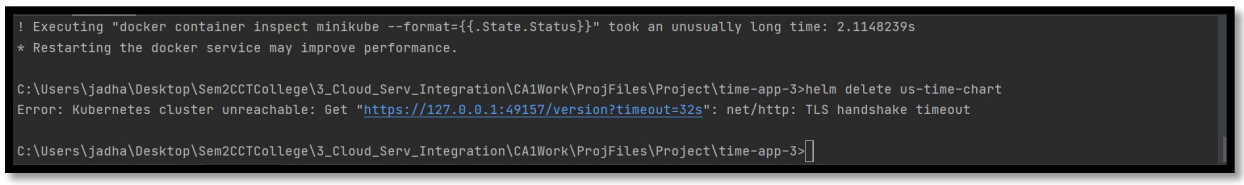
- Created the Helm chart for time-app-3 (US time service).



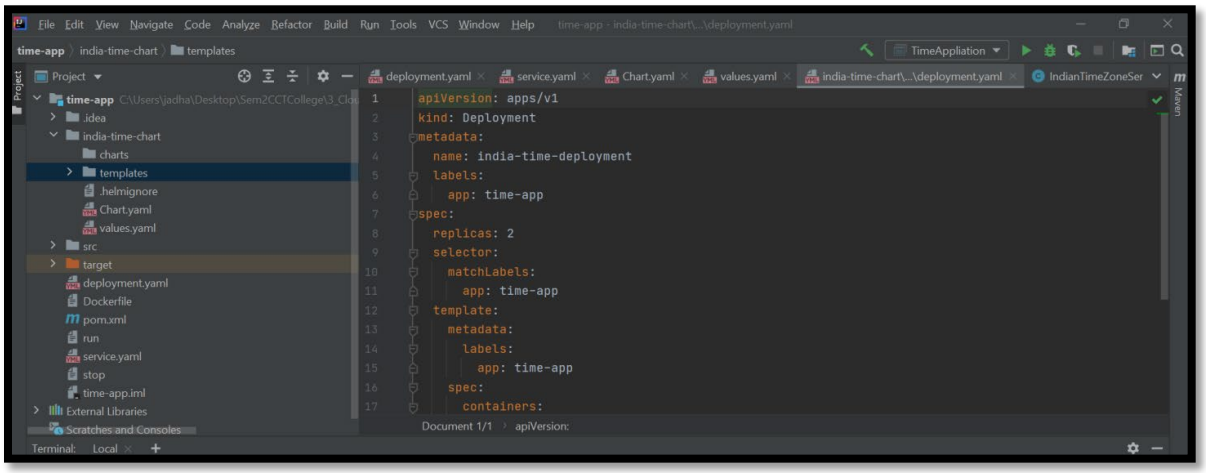
- Same for aggregator service.



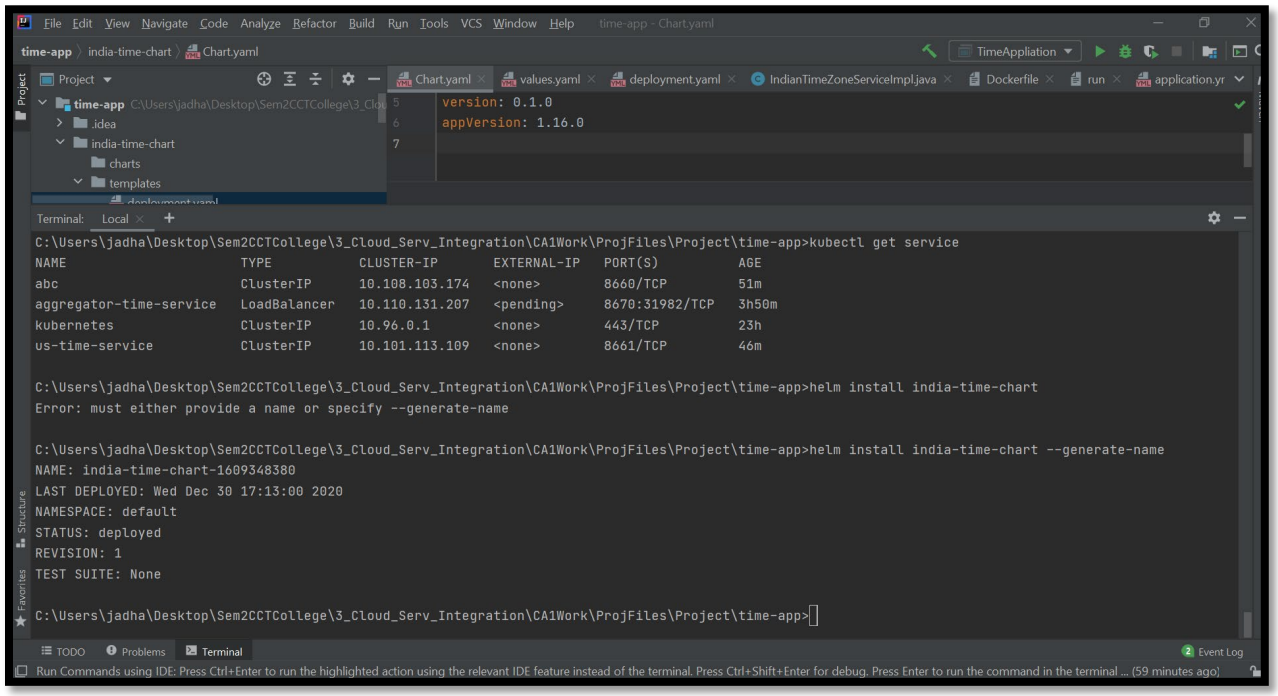
- Again, faced the issue of “kubernetes unreachable …net/http: TLS handshake timeout” as shown below:



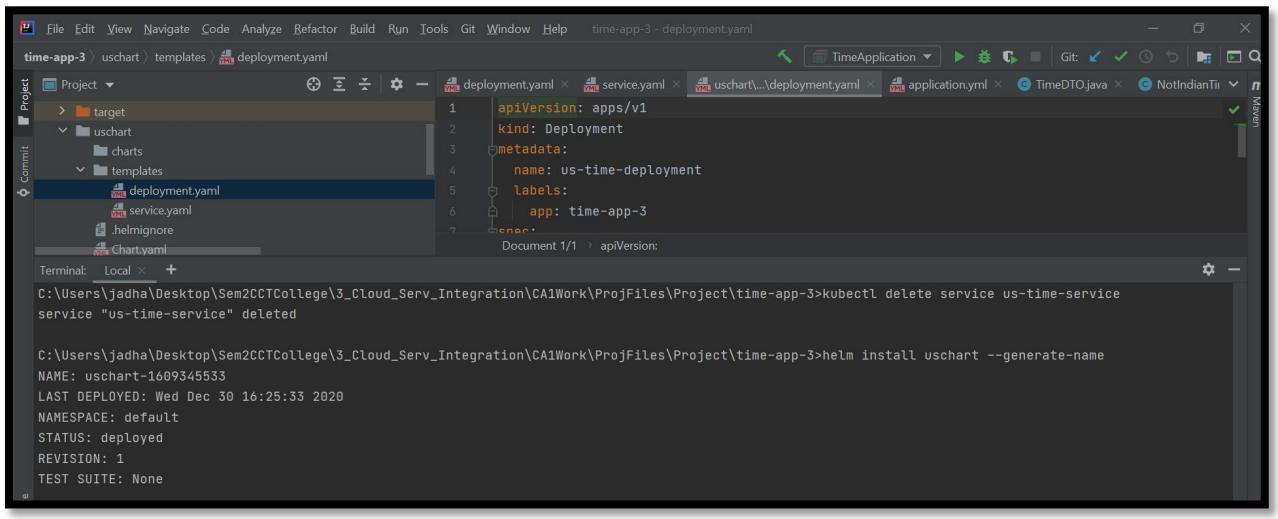
- Again, restarted the minikube.
- Copied all YAML files (deployment and service) in template folder of helm chart.
- Remove the deployment and service YAML from Kubernetes otherwise helm would face an error while recreating these deployments in its process.



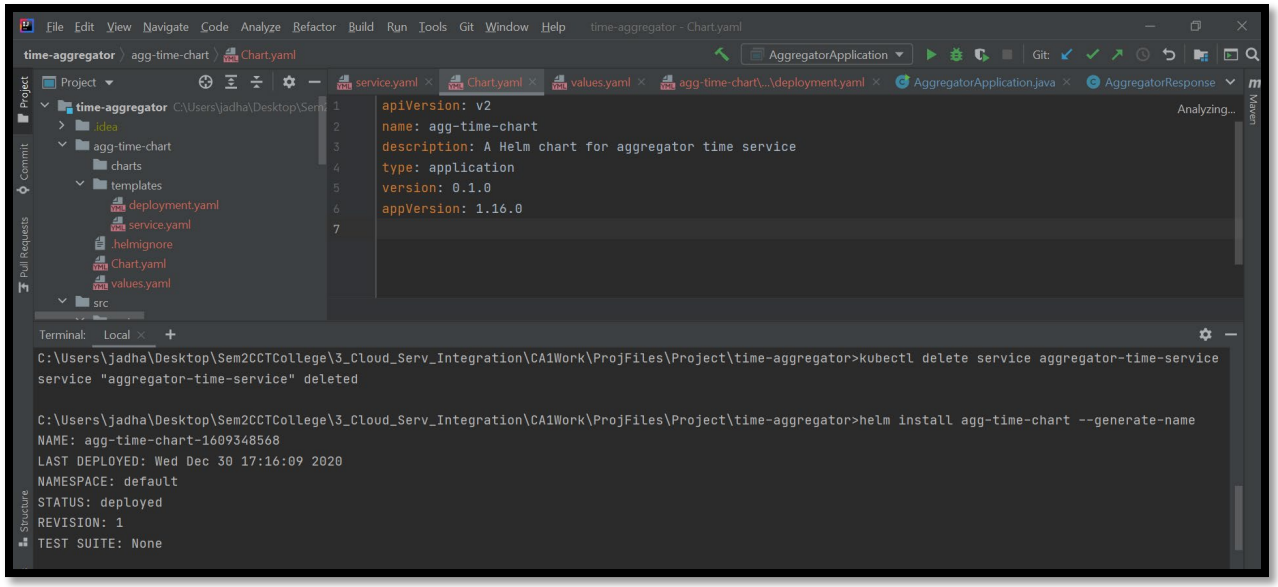
- Installed helm india-time-chart using “helm install india-time-chart –generate-name command”.



- Installed US time chart



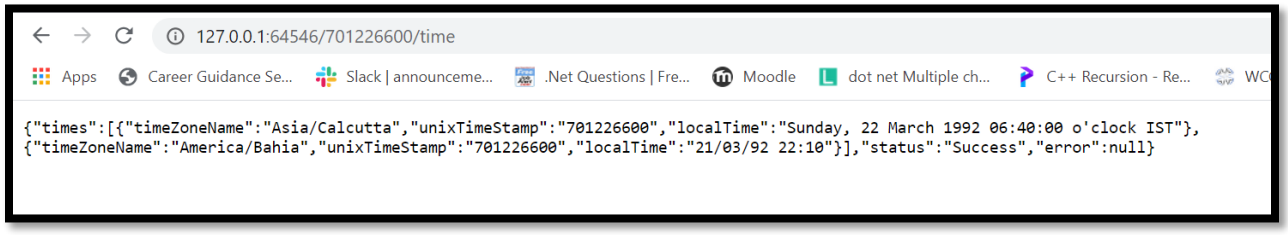
- Installed aggregator time chart.



- Accessing the time aggregator NodePort type service deployed and controlled by Helm Chart.
- Use “minikube service aggregator-time-service –url” command.

```
C:\WINDOWS\system32>kubectl get svc aggregator-time-service
NAME                      TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
aggregator-time-service   NodePort   10.99.205.113   <none>        8670:30809/TCP   27h

C:\WINDOWS\system32>minikube service aggregator-time-service --url
* Starting tunnel for service aggregator-time-service.
|-----------|-------------------------|-------------|-----------------------|
| NAMESPACE |          NAME           | TARGET PORT |          URL          |
|-----------|-------------------------|-------------|-----------------------|
| default   | aggregator-time-service |             | http://127.0.0.1:64546 |
|-----------|-------------------------|-------------|-----------------------|
http://127.0.0.1:64546
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

Output:



Successfully tested the application after packaged in Helm chart.

Conclusion:

In this way, I have learned how to build cloud-based microservice, including creating a docker image (containerization), container management using Kubernetes, and packing all Kubernetes services in Helm chart. It gives me a clear understanding of how multiple teams can work on each service and deploy the application faster. It also minimizes the development work and improves code reusability.

This way of developing a solution based on a set of services that communicate through an API. In addition to all this using docker container, Kubernetes, and Helm chart make it easy to deploy, maintain an application with high availability as well as scalability.