SWE645 – HW3: Build survey form using Angular and Spring Boot and setup Kubernetes cluster using Rancher

Team Member: Aniket Pandey, Siddhanth Kalyanpur, Tanush Singh

Step 1: Write POST and GET api using Spring Tool Suite and connect to SQL

- Following are the two APIs

SurveyApplication.java | SurveyService.java | SurveyController.java | application.properties | Survey.java

```
1  package com.survey.app.controller;
2
3  import java.util.List;
16
17 @RestController
18 @RequestMapping("/api/v1.0")
19 public class SurveyController {
20
21     private SurveyInterface surveyService;
22
23     public SurveyController(SurveyInterface surveyInterface) {
24         super();
25         this.surveyService = surveyInterface;
26     }
27
28     @PostMapping("/surveys")
29     public ResponseEntity<Survey> saveSurvey(@RequestBody Survey survey) {
30         return new ResponseEntity<Survey>(surveyService.saveSurvey(survey), HttpStatus.CREATED);
31
32     }
33
34     @GetMapping("/surveys")
35     public List<Survey> getAllSurveys(){
36         return surveyService.getAllSurveys();
37
38     }
39
```

  - GET https://{baseurl}/api/v1.0/surveys
  - POST **Error! Hyperlink reference not valid.**

  Body

```
{
   "firstName": "John",
   "lastName": "Doe",
   "email": "john.doe@mail.com",
   "address": "9155 Walnut St",
   "city": "Bentonville",
   "state": "AR",
   "zip": "22031",
   "telephone": "8188884114",
   "dateOfSurvey": "04-28-2022",
   "recommendation": "Very Likely",
   "interest": "Friends",
   "likings": "Students"
}
```

- The Data Model looks like the following fields

SurveyApplication.java | SurveyService.java | SurveyController.java | application.properties | Survey.java

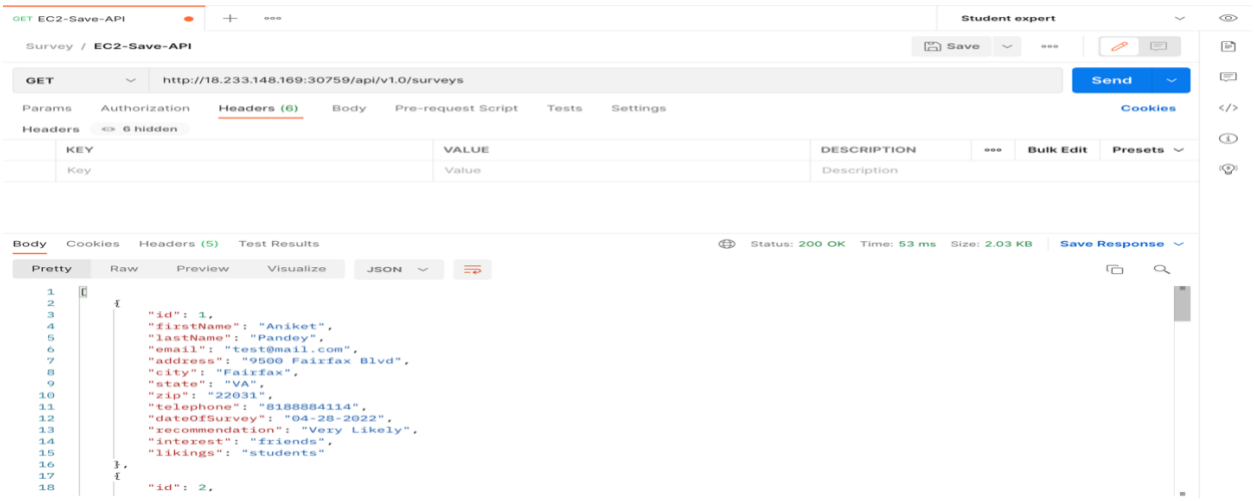
```java
package com.survey.app.model;

import javax.persistence.Column;

@Data
@Entity
@Table(name="survey")
public class Survey {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(name = "email", nullable = false)
    private String email;

    @Column(name = "address", nullable = false)
    private String address;

    @Column(name = "city", nullable = false)
    private String city;

    @Column(name = "state", nullable = false)
    private String state;

    @Column(name = "zip", nullable = false)
    private String zip;

    @Column(name = "telephone", nullable = false)
    private String telephone;

    @Column(name = "survey_date", nullable = false)
    private String dateOfSurvey;

    @Column(name = "recommendation")
    private String recommendation;

    @Column(name = "interest")
    private String interest;

    @Column(name = "likings")
    private String likings;

```

- Write the API and create MySQL database using AWS RDS
- Copy the database endpoint, port number, username and password
- Using JDBC connect to database and write the following lines in application.properties

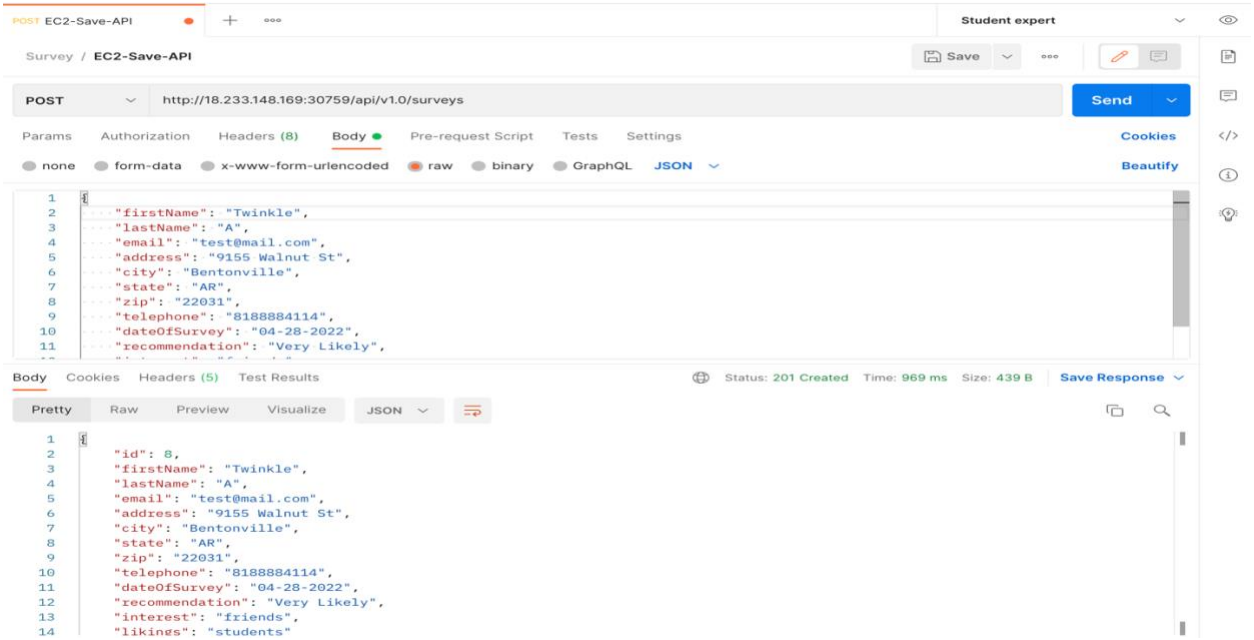
SurveyApplication.java | SurveyService.java | SurveyController.java | *application.properties | Survey.java

```
spring.datasource.url=jdbc:mysql://survey-database.cxltcu4iezvy.us-east-1.rds.amazonaws.com:3306/survey
spring.datasource.username=#######
spring.datasource.password=#######

# Hibernate properties
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect

# Create, Create-Drop
spring.jpa.hibernate.ddl-auto=update
```

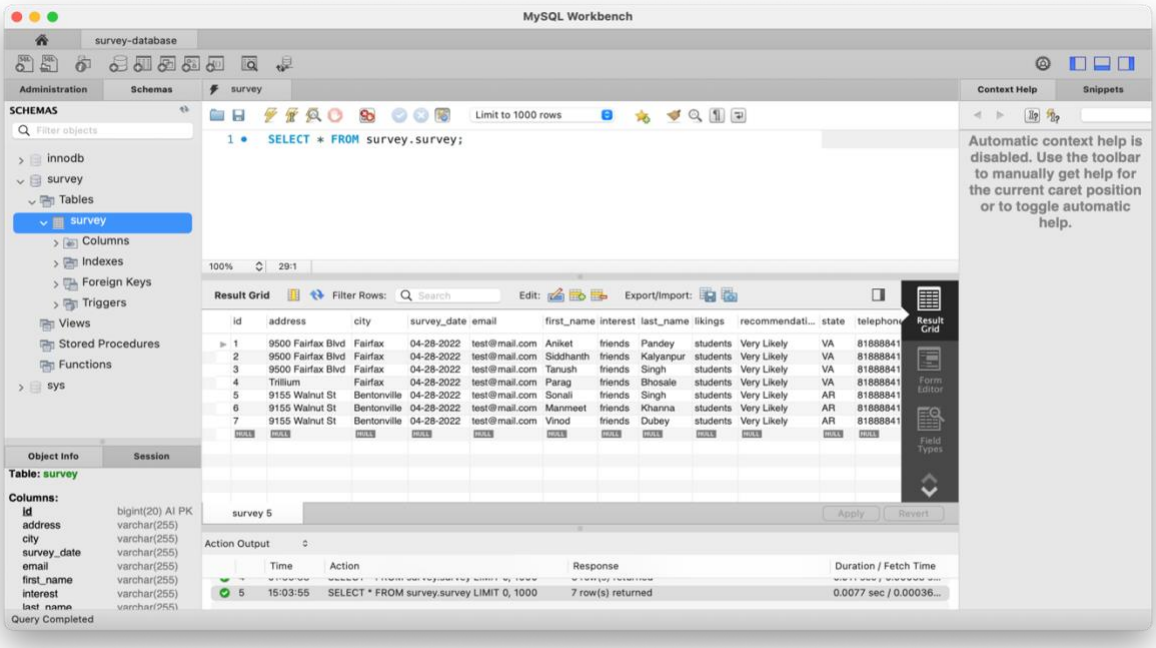
- The api can be tested using Postman
- GET API



- POST API



- The Database record can be viewed using MySQLWorkbench



- Now right click on the pom.xml file and build inside the Goal on the next step on build write clean build and the continue with the JAR packaging. The JAR file will be present inside target folder by default.

Step 2: Build Docker image and containerize survey api using Docker

- Write Dockerfile as below

Dockerfile

infra > Dockerfile > ...

```
1  FROM openjdk:17-jdk
2
3  LABEL author.name="Aniket Pandey"
4
5  EXPOSE 8080
6
7  ARG JAR_FILE=survey/target/survey-0.0.1-SNAPSHOT.jar
8
9  COPY ${JAR_FILE} .
10
11 CMD [ "java", "-jar",  "/survey-0.0.1-SNAPSHOT.jar"]
```

- Build docker image with the below command. My Dockerfile is inside infra folder and therefore I've specified path as infra/Dockerfile. As a standard practice all the infrastructure setup related files should be kept inside infra folder for easy segregation.

  docker build -f infra/Dockerfile --tag survey-api:amd64-v1.0 .
- Tag the docker image and push it on dockerhub using the below command

  docker tag survey-api:amd64-v1.0 aniket414/survey-api:amd64-v1.0

  docker push aniket414/survey-api:amd64-v1.0

Step 3: Create three EC2 instance, two for deployment and the other for Rancher setup

- Select the Ubuntu ami from AWS marketplace and create one instances for Rancher and select Amazon Linux 2 ami and create instance for deployment of rest api and ui.
- Allow inbound traffic using security group from following ports

Inbound rules (9) | Manage tags | Edit inbound rules

Filter security group rules | 1

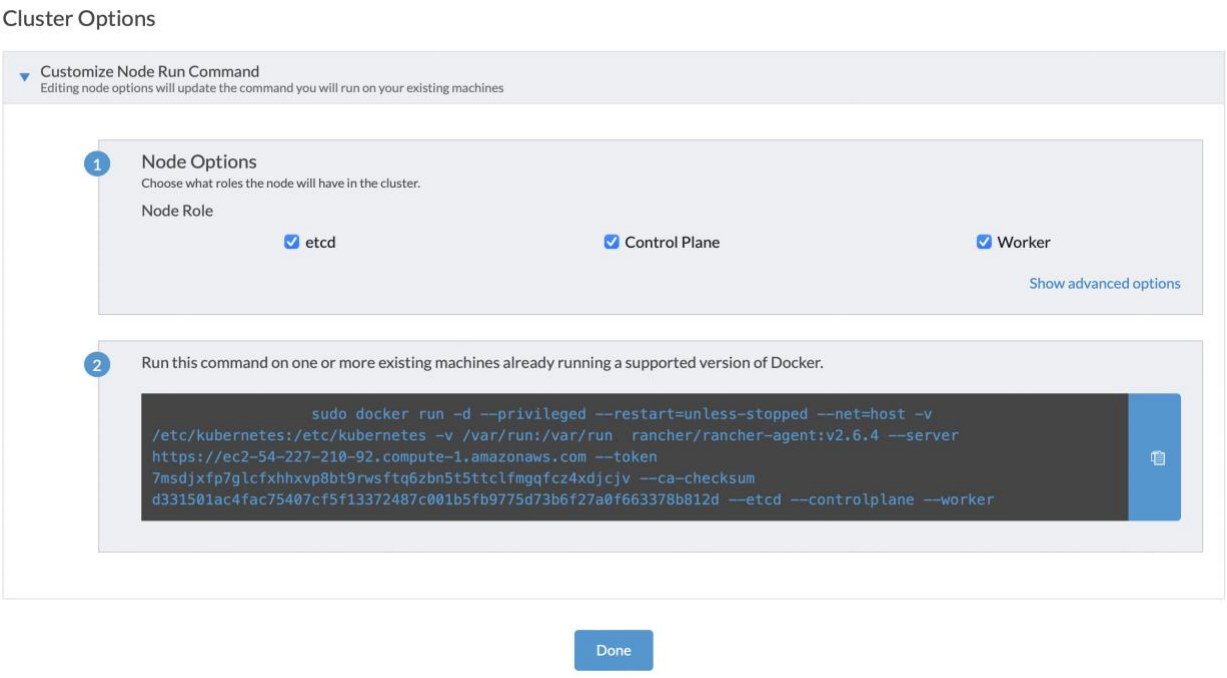
Security group rule...	IP version	Type	Protocol	Port range	Source
sgr-0225625f9a78f1e54	IPv4	HTTP	TCP	80	0.0.0.0/0
sgr-04640b85a1c4f2a9e	IPv6	Custom TCP	TCP	8080	::/0
sgr-06d9bba398e7fabd4	IPv4	All traffic	All	All	0.0.0.0/0
sgr-024dd4f67097e3ba3	IPv4	HTTPS	TCP	443	0.0.0.0/0
sgr-0246fd54e55a3a1e6	IPv6	HTTP	TCP	80	::/0
sgr-02bd1b51428929...	IPv6	HTTPS	TCP	443	::/0
sgr-021b81b98f23fd7f6	IPv4	SSH	TCP	22	0.0.0.0/0
sgr-0365140e4d19cf91b	IPv4	Custom TCP	TCP	8080	0.0.0.0/0
sgr-0d96147c2f4eb03f3	IPv6	All traffic	All	All	::/0

Step 4: Setup Rancher on one of the instance

- SSH into the instance using pem file
  ssh -i key-value.pem ubuntu@public-dns
- Install docker after updating
  sudo apt-get update
  sudo apt install docker.io
- Install Rancher using the following command

sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher

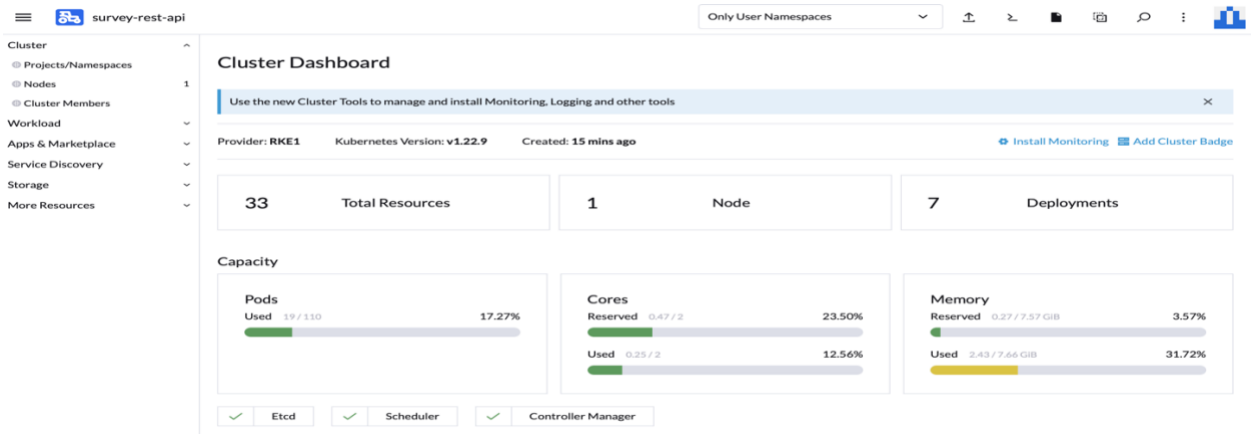
- After some time open instance public dns which will display Rancher login page where we'll setup username and password
- Login inside and click on create cluster
- Choose custom option from the list of available of available choices and click next
- Enter the cluster name and leave everything as default and click on next
- In the next section select control plane, etcd, and worker and copy the command generated



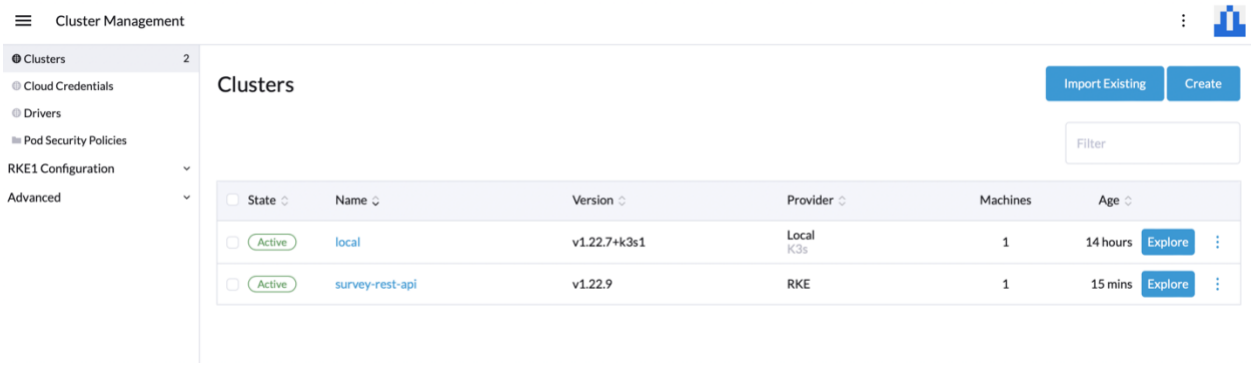
Step 5: Setting up another instance for api orchestration

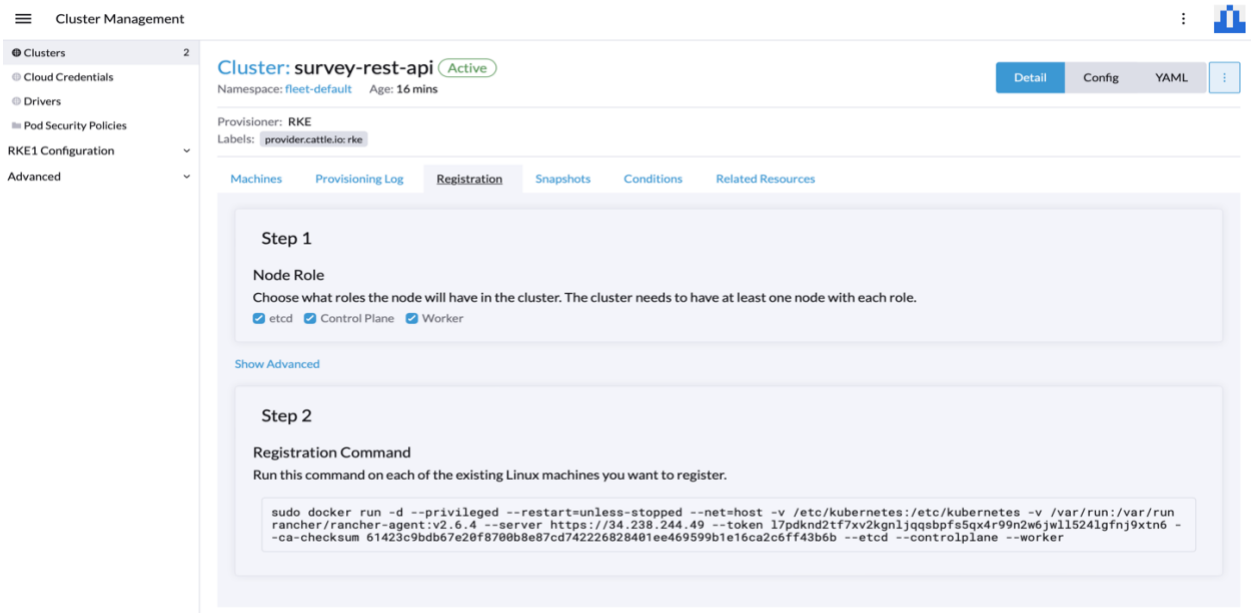
- SSH into the instance using pem file
  ssh -i key-value.pem ec2-user@public-dns
- Install docker after updating
  sudo yum update
  sudo yum install docker
  sudo service docker start

- Run the above copied command and after some time go to the Rancher page and you should see cluster provisioning has started.



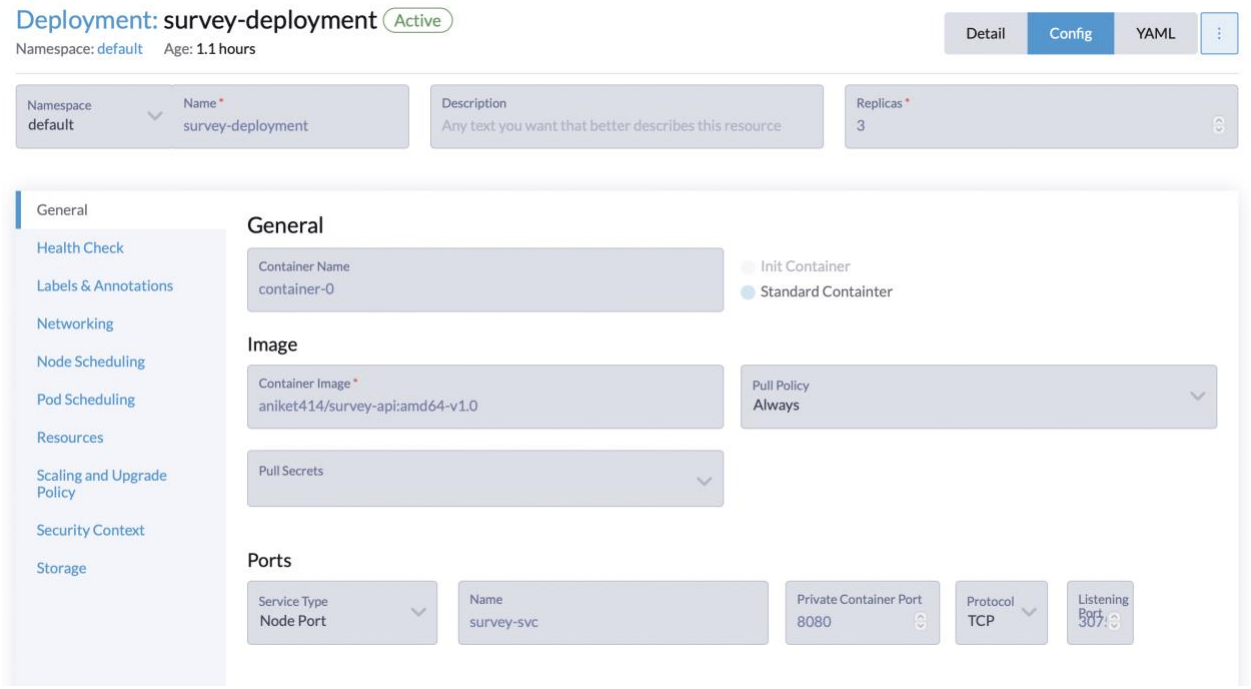
- Condition of the cluster is active and healthy



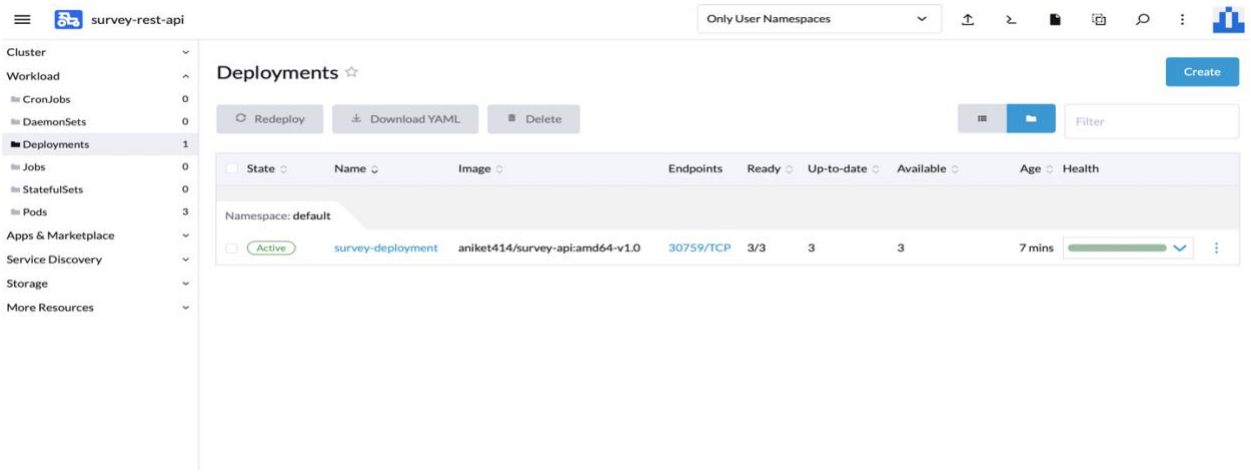


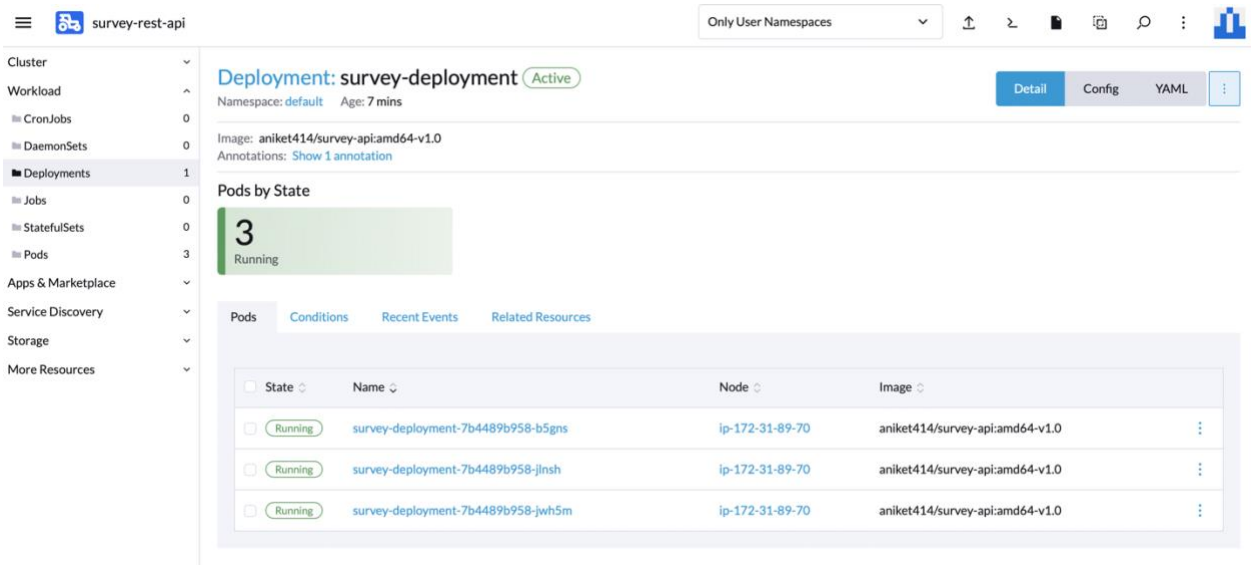
Step 6: Deploy the survey-api docker image on cluster on setup three pods running at all time

- Enter the cluster just created on Rancher and click on Deployment under the Workload section. Alternatively you can also copy the kubeconfig file and if you have kubectl setup done on your local then paste the kubeconfig file content in .kube/config and you can easily access the cluster and deployment using kubectl from your local.
- Enter the replica set as mentioned in assignment i.e. 3
- Enter the docker image uri
- Setup NodePort as a service

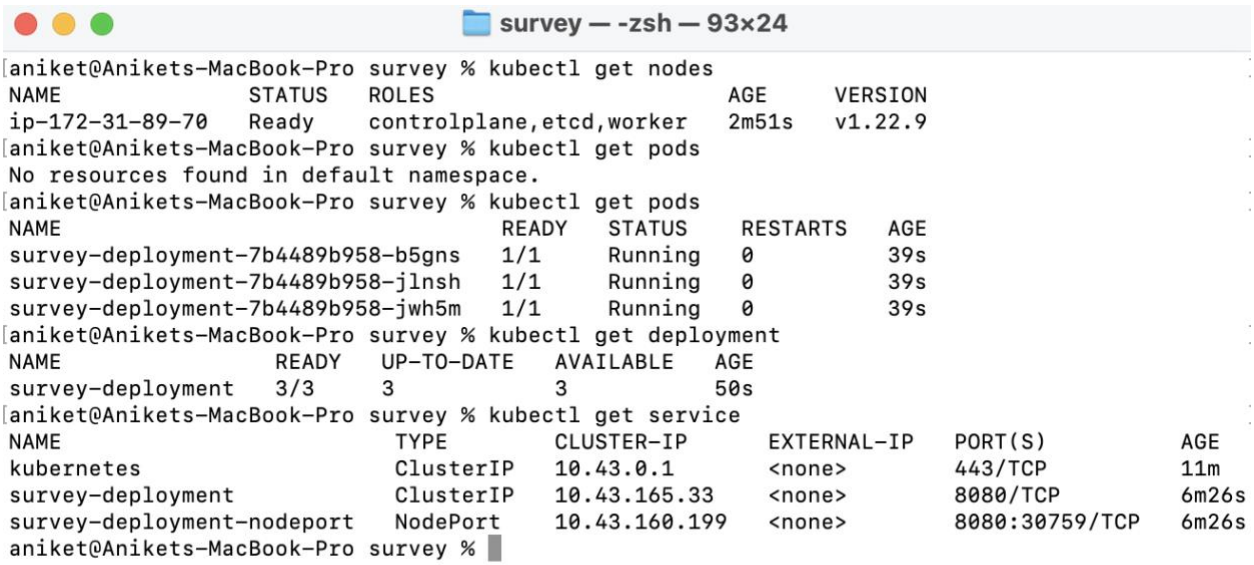


- Click on done and you should see your pods coming up.





- You can also verify the status from local using kubectl command
- Following are the screenshot of service, node, deployment, and pods

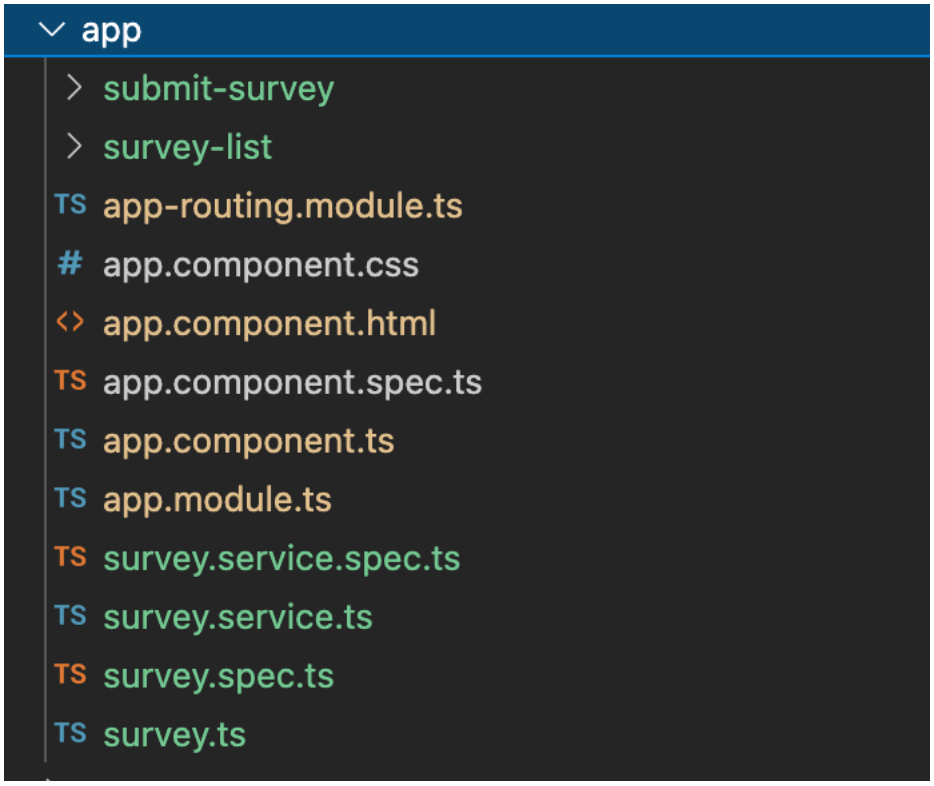


Step 7: Create Angular application

- Create a service call and make GET and POST calls to the api using HttpClient
- The url should be the url of the EC2 instance on which the api is deployed. Check the image below with baseUrl.

```typescript
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Survey } from './survey';

@Injectable({
  providedIn: 'root'
})
export class SurveyService {

  private baseUrl = "http://54.91.112.74:31867/api/v1.0/surveys";

  constructor(private httpClient: HttpClient) { }

  getAllSurveys() : Observable<Survey[]>{
    return this.httpClient.get<Survey[]>(`${this.baseUrl}`)
  }

  submitSurvey(survey: Survey): Observable<Object>{
    return this.httpClient.post(`${this.baseUrl}`, survey);
  }
}
```

- Following is the walk through of Angular project



  - Class for data binding: survey.ts
  - Service for API calls: survey.service.ts
  - Submit form component: submit-survey
  - Get list of all the surveys: survey-list

Step 8: Build Docker image and containerize survey ui using Docker

- Write Dockerfile as below

```
FROM node:latest as build

LABEL author.name="Aniket Pandey"

WORKDIR /survey-ui/app

RUN npm install -g @angular/cli

COPY ./survey-ui/package.json .

RUN npm install

COPY ./survey-ui .

RUN ng build

FROM nginx as runtime

COPY --from=build /survey-ui/app/dist/survey-ui /usr/share/nginx/html
```

- Build docker image with the below command. My Dockerfile is inside infra folder and therefore I've specified path as infra/ui.Dockerfile. As a standard practice all the infrastructure setup related files should be kept inside infra folder for easy segregation.

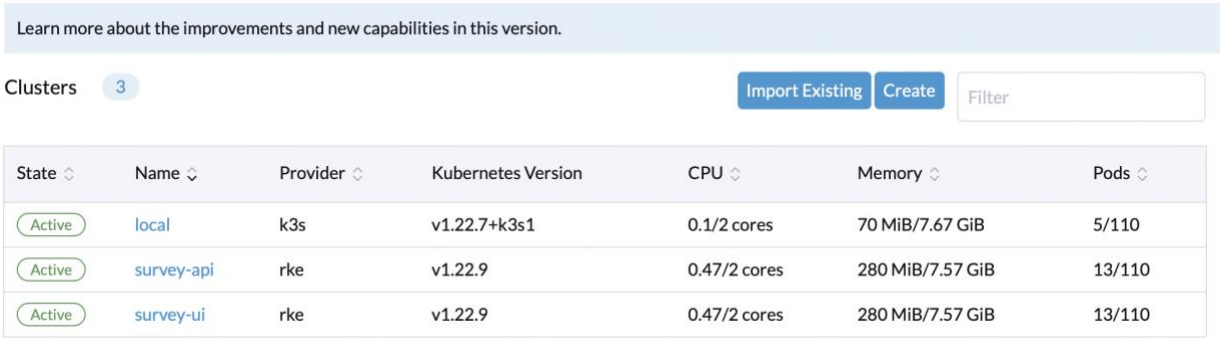
  docker build -f infra/ui.Dockerfile --tag survey-ui:amd64-v1.0 .
- Tag the docker image and push it on dockerhub using the below command

  docker tag survey-ui:amd64-v1.0 aniket414/survey-ui:amd64-v1.0

  docker push aniket414/survey-ui:amd64-v1.0

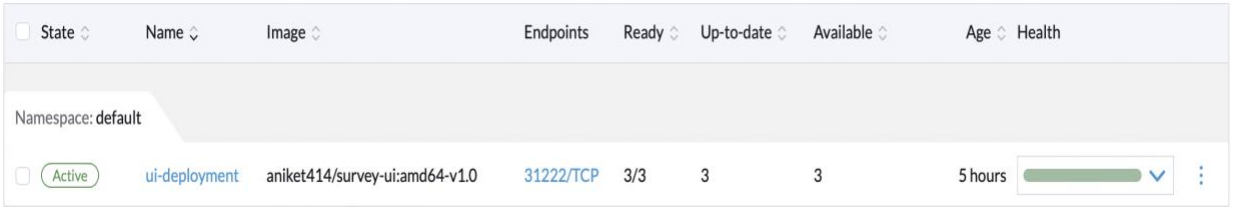
Step 9: Deploy the survey-ui docker image on cluster on setup three pods running at all time

- Now similarly perform Step 5 and Step 6 as mentioned above for Angular UI deployment.



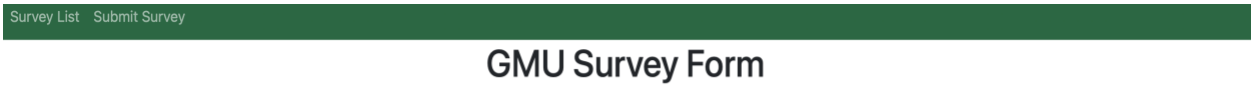
- The cluster for UI is setup above

- The deployment of 3 pods is done and can be seen as below which are in ready state.

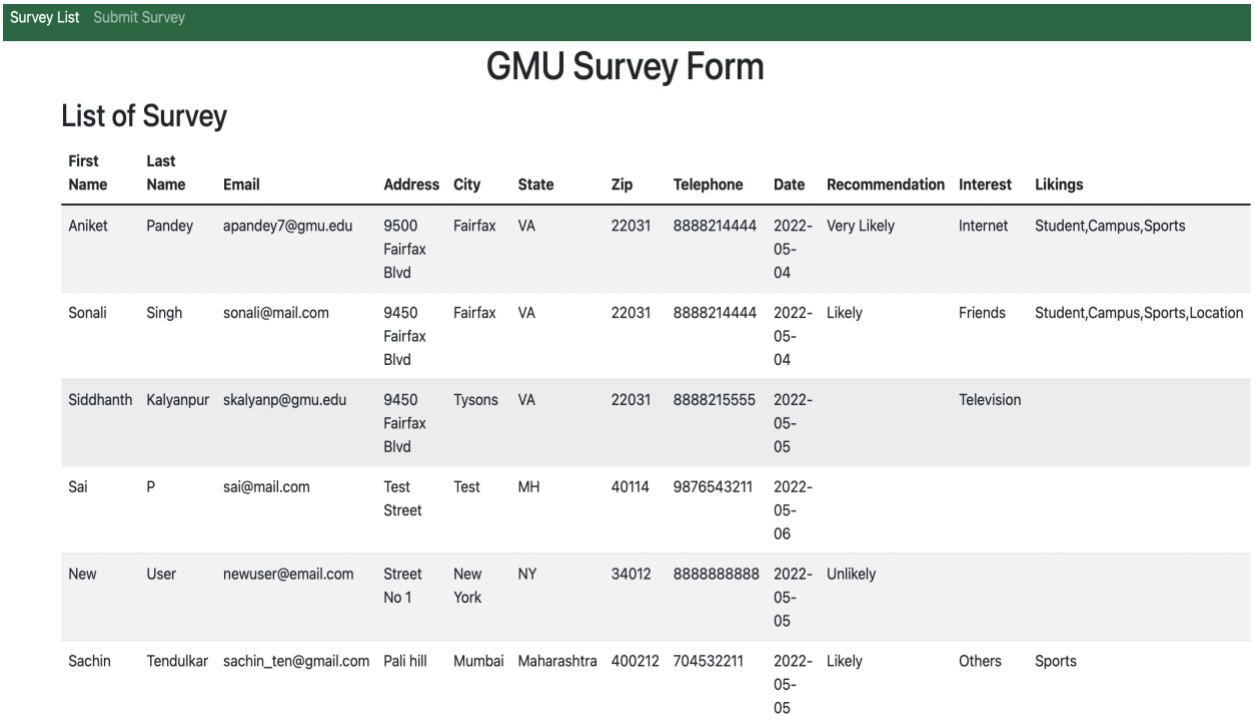


State	Name	Image	Endpoints	Ready	Up-to-date	Available	Age	Health
Namespace: default								
Active	ui-deployment	aniket414/survey-ui:amd64-v1.0	31222/TCP	3/3	3	3	5 hours	

- After deployment the UI will look like as below

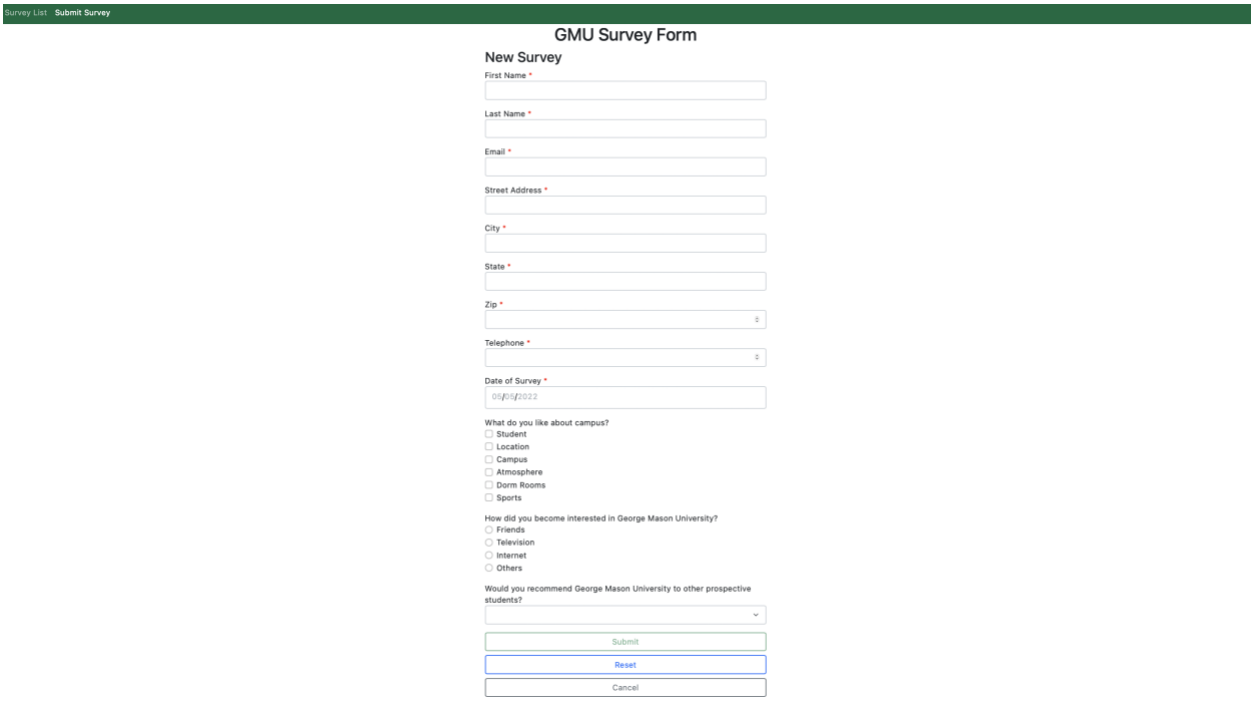


- This is the homepage it has two navigation options one to view list of surveys done till date and the other is for submitting new survey



First Name	Last Name	Email	Address	City	State	Zip	Telephone	Date	Recommendation	Interest	Likings
Aniket	Pandey	apandey7@gmu.edu	9500 Fairfax Blvd	Fairfax	VA	22031	8888214444	2022-05-04	Very Likely	Internet	Student,Campus,Sports
Sonali	Singh	sonali@mail.com	9450 Fairfax Blvd	Fairfax	VA	22031	8888214444	2022-05-04	Likely	Friends	Student,Campus,Sports,Location
Siddhanth	Kalyanpur	skalyanp@gmu.edu	9450 Fairfax Blvd	Tysons	VA	22031	8888215555	2022-05-05		Television	
Sai	P	sai@mail.com	Test Street	Test	MH	40114	9876543211	2022-05-06			
New	User	newuser@email.com	Street No 1	New York	NY	34012	8888888888	2022-05-05	Unlikely		
Sachin	Tendulkar	sachin_ten@gmail.com	Pali hill	Mumbai	Maharashtra	400212	704532211	2022-05-05	Likely	Others	Sports

- The above image displays list of surveys



- The above image displays the survey form with all the fields.
- The UI calls the API can also be verified from the browser inspect.