Team Members: URL for Website:

Rachana Thota : G01237600 https://rthotaswe6451.s3.us-east-2.amazonaws.com/index.html

Tuljasree Bonam : G01179672

Project Description and Roles of Team Members:

Front End: *Angular application* -> *Rachana, Srujan*

Back End: Spring boot -> Rachana, Tulja

Database: Amazon RDS -> Srujan, Tulja

CI/CD Pipeline:

GitHub as source code repository for yml files.
 Build automation tool: Argo CD
 Automated build and deployment platform: Kubernetes
 Srujan

Instructions:

1. Front-End Angular Application

- 1.1. Download Angular CLI and make new project using ng new command.
- 1.2. Create all necessary components and routing files.
- 1.3. Write a service TS file which connects the angular application to the backend.
- 1.4. Use 'ng serve' command to check if angular application works locally.

2. Spring Boot Backend Application

- 2.1. Install Spring tool Service (STS) and use latest version of java JDK.
- 2.2. Create new spring boot project in STS and choose the JPA and MYSQL connector as dependencies.
- 2.3. Create three files: controller, model, repository along with necessary annotations which make the backend calls.
- 2.4. Setup tomcat server and make sure application runs locally.

3. Dockerize backend application and deploy in Kubernetes using ArgoCd

- 3.1. Dockerize a springboot application using maven and write a docker file that copies .jar file to specified path.
- 3.2. Use 'docker build' command to create a docker image of the backend application and push the image to Docker hub account.
- 3.3. Write deployment.yml and service.yml files for backend application and push these files in Git repository.
- 3.4. Using ArgoCD implement CI/CD pipeline on Google Cloud Platform.

- 3.5. Create a GKE cluster.
- 3.6. Set up Argo CD using command 'kubectl create namespace argood'
- 3.7. Change the argo cd server from node port to Load balancer 'kubectl -n argocd edit svc argocd-serve'
- 3.8. Run command 'kubectl get svc'.
- 3.9. In GCP go to Kubernetes engine Services and Ingress and access the external load balancer associated with the cluster and the relevant application name.
- 3.10. Access the external IP address and open the argo cd.
- 3.11. Create Application in AgroCd and link the backend application.
- 3.12. Open the external application of backend and verify if its working.

4. Dockerize angular frontend application and deploy in Kubernetes using ArgoCd

- 4.1. Write deployment.yml and service.yml files for angular application and push these files in Git repository.
- 4.2. To dockerize an angular application used following commands,
 - 4.2.1. npm install -g @angular/cli
 - 4.2.2. Create a docker file with below statements and add it into angular application.

```
FROM node:12.0 as build-stage
WORKDIR /app
COPY package*.json /app/
RUN npm install
COPY ./ /app/
ARG configuration=production
RUN npm run build -- --output-path=./dist/out --
configuration $configuration

FROM nginx:1.15
#Copy ci-dashboard-dist
COPY --from=build-stage /app/dist/out/
/usr/share/nginx/html
#Copy default nginx configuration
COPY ./nginx-custom.conf
/etc/nginx/conf.d/default.conf
```

- 4.3. Use 'docker build' command to create a docker image of the angular application and push the image to Docker hub account.
- 4.4. Change the url to the external ip address of backend application in service.ts file in angular application.
- 4.5. In GCP go to Kubernetes engine Services and Ingress and access the external load balancer associated with the cluster and the relevant application name.
- 4.6. Access the external IP address and open the argo cd.
- 4.7. Create Application in AgroCd and link the backend application.
- 4.8. Open the external application of frontend and verify if its working

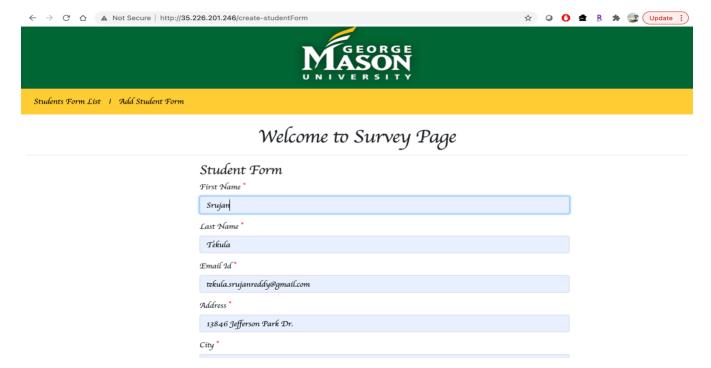
5. CI/CD Pipeline:

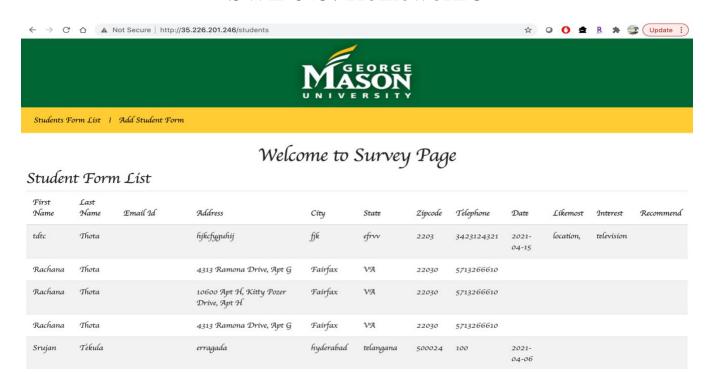
- 5.1. Wrote sangular-deployment.yml and sangular-service.yml files for angular application and wrote sspring-deployment.yml and sspring-service.yml files for backend application and pushed these files in Git repository.
- 5.2. Using ArgoCD we have implemented CI/CD pipeline on Google Cloud Platform and used below resources in order to implement it.
- 5.3. Create a GKE cluster.
- 5.4. Set up Argo CD using command 'kubectl create namespace argood'
- **5.5.** Change the argo cd server from node port to Load balancer 'kubectl -n argocd edit svc argocd-serve'
- 5.6. Run command 'kubectl get svc'.
- 5.7. In GCP go to Kubernetes engine Services and Ingress and access the external load balancer associated with the cluster and the relevant application name.
- 5.8. Access the external IP address and open the argo cd.
- 5.9. Create a new application with the path of the folder with YAML files available in git repository and the default namespace.
- 5.10. Now sync your application with the current state of your repository. You should be seeing the specified number of replicas as Pods when you use the command. 'kubectl get pods'.

Links:

Url of Website: https://rthotaswe6451.s3.us-east-2.amazonaws.com/index.html

Front End: http://35.226.201.246/students





Back End: http://35.230.54.187:8070/api/v1/students



 $\label{lem:com/rachana07/swe-ang} \ \ \underline{\text{https://github.com/rachana07/swe-ang}} \ \ , \ \underline{\text{https://github.com/rachana07/swe-argocd}}$

References:

- Angular: https://angular.io/docs
- Springboot: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
- Spring Boot dockerization using maven: https://medium.com/swlh/build-a-docker-image-using-maven-and-spring-boot-58147045a400
- Angular dockerization using nginx: https://betterprogramming.pub/7-steps-to-dockerize-your-angular-9-app-with-nginx-915f0f5acac
- Argo Cd: https://argoproj.github.io/argo-cd/getting_started/