CSCI 5401 Serverless Data Processing

Assignment-2

Instructor: Dr. Saurabh Dey

Date of Submission: 10 June 2020

Submitted By:

Kase Raviteja (B00823644)

DALHOUSIE UNIVERSITY

Faculty of Computer Science

Dalhousie University

Halifax, Nova Scotia

Part A

a) Docker container and MySQL setup in CloudSQL

CloudSQL setup in GCP:[1]

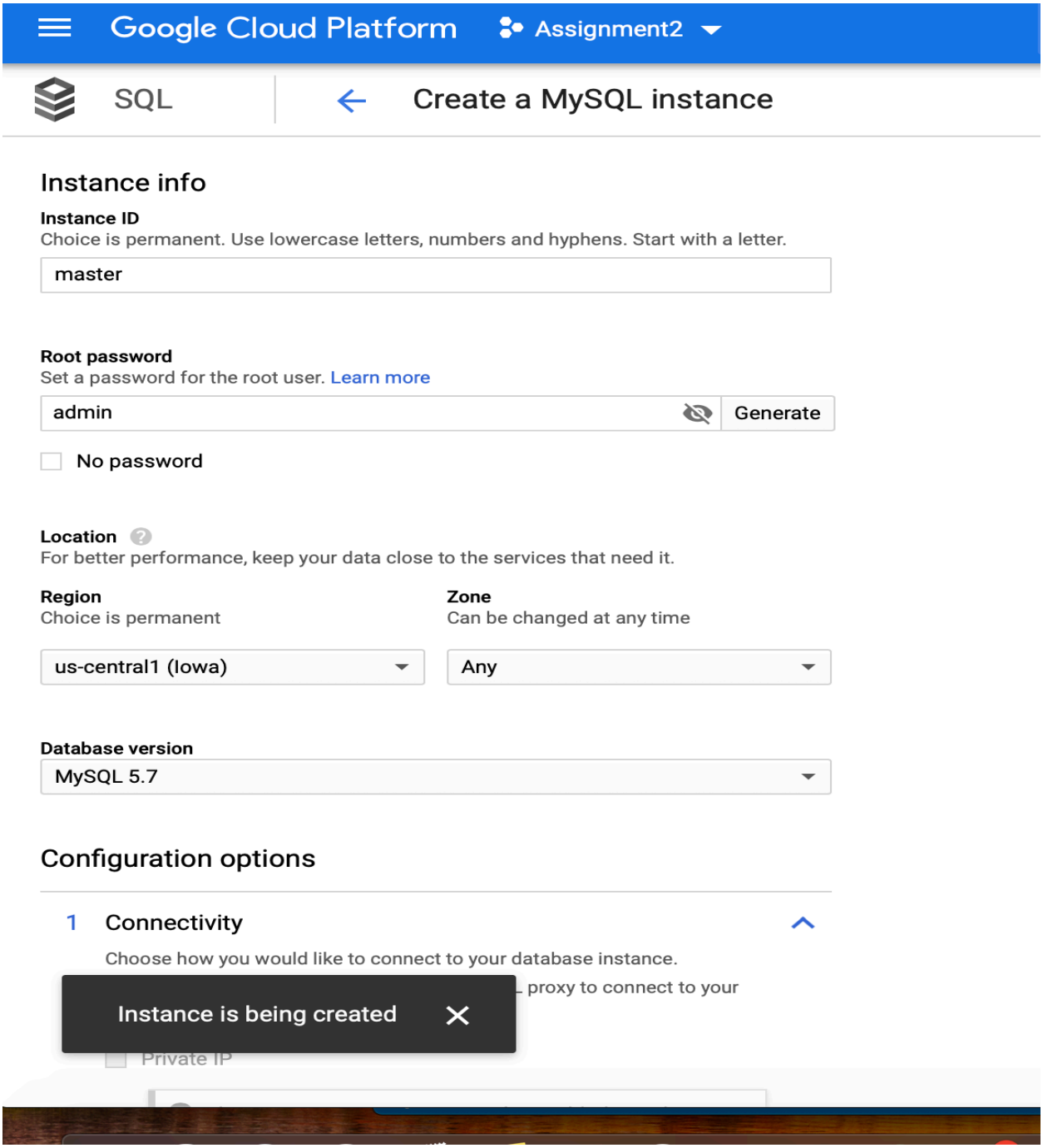

Figure 1:CloudSQL setup

DB creation in CloudSQL

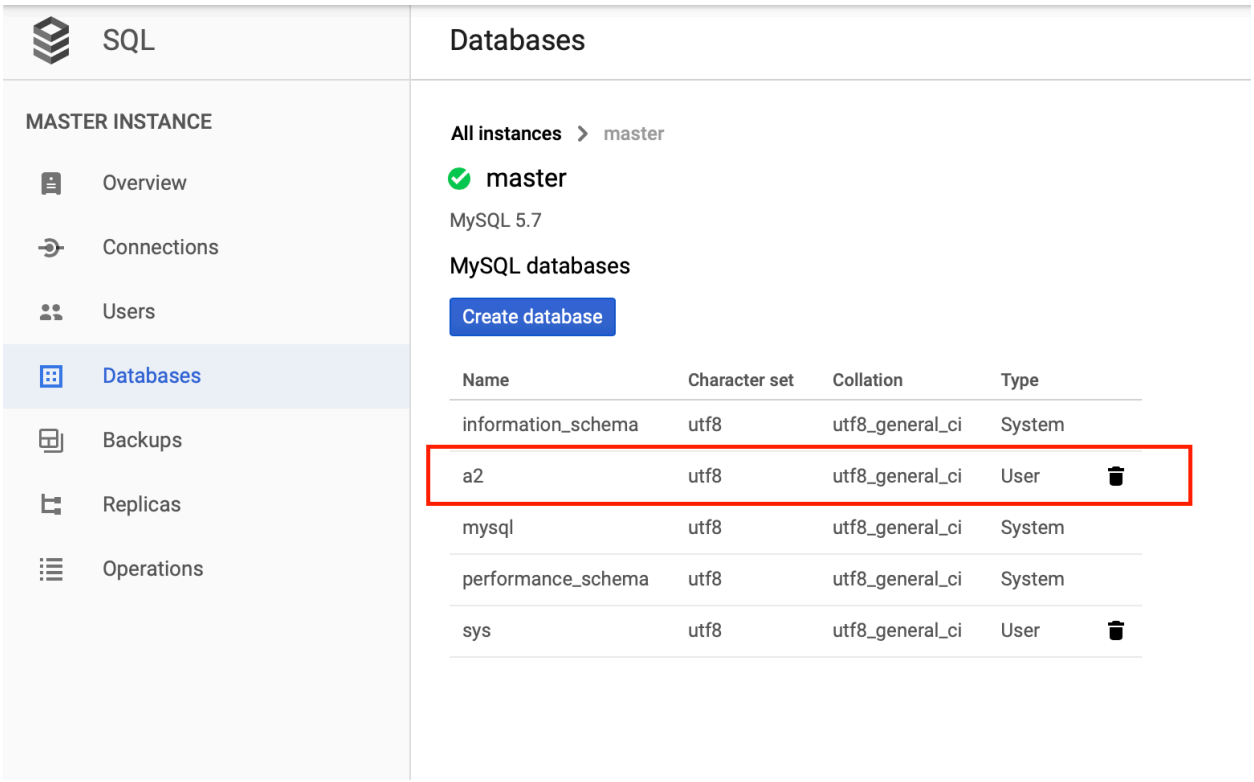

Figure 2:a2 database created

CloudSQL accessed in MySQL workbench:

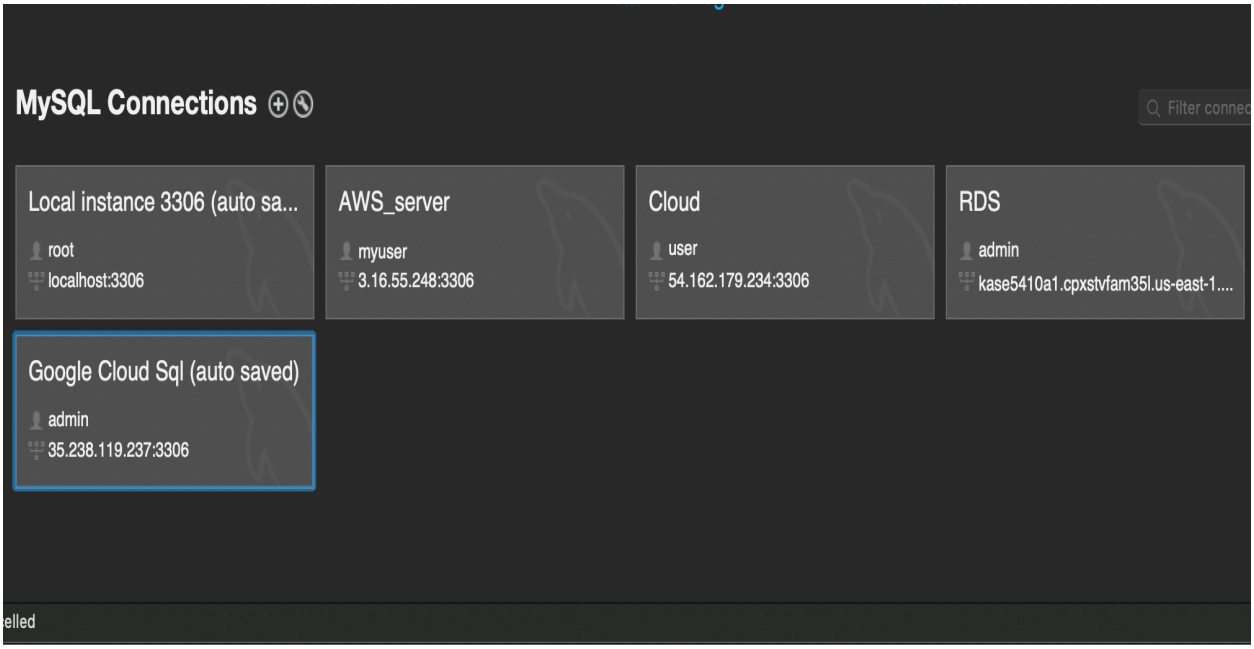

Figure 3: CloudSQL accessed in MySQL

APIs are developed using Spring Boot[2]. Docker images are built locally[3] and tagged to Google Container Registry[4].

b)Registration Docker image:

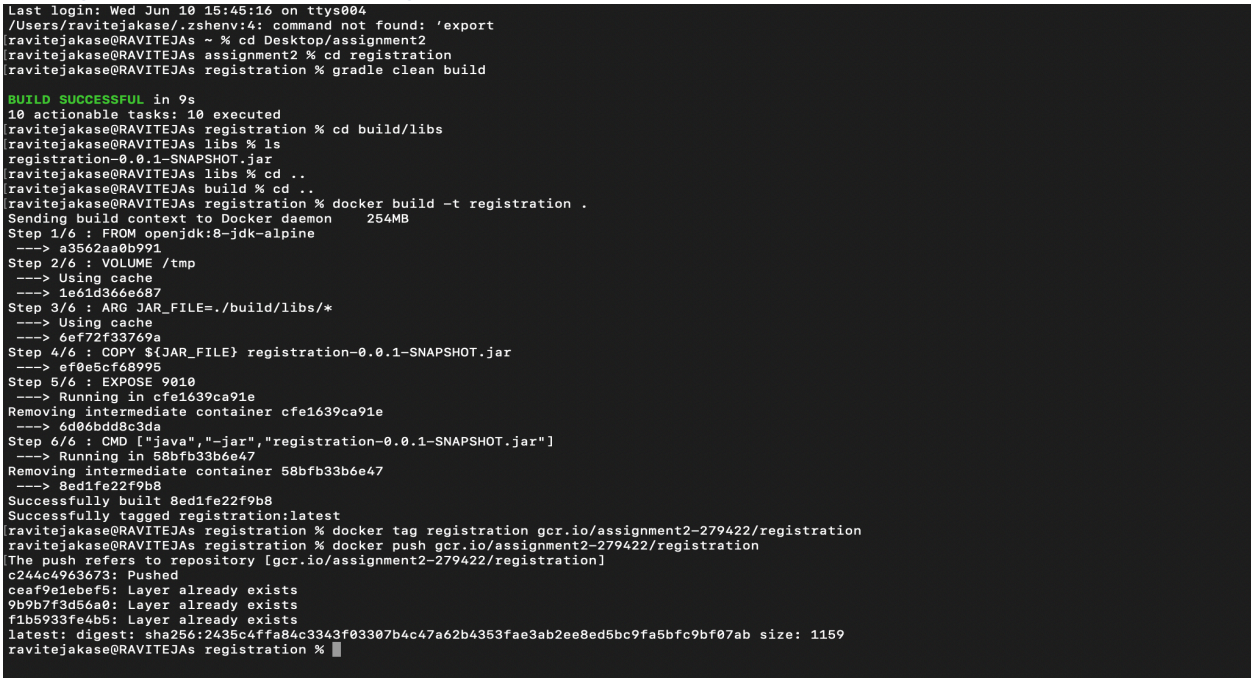

Figure 4:Registration Docker image

Registration Code snippet:

```
@CrossOrigin
@RestController
public class ResgistrationController {
	@Autowired
	private RegistrationService service;
	@PostMapping("/registerUser")
	public User registerUser(@RequestBody User user) throws Exception {
		String tempEmailId = user.getEmailId();
		if (tempEmailId != null && !"".equals(tempEmailId)) {
			User userObjUser = service.fetchUserByEmailId(tempEmailId);
			if (userObjUser != null) {
				throw new Exception("User already exists");
```

```
            }
        }
        User userObjUser = null;
        userObjUser = service.saveUser(user);
        return userObjUser;
    }
}
```

User Registered in Database:

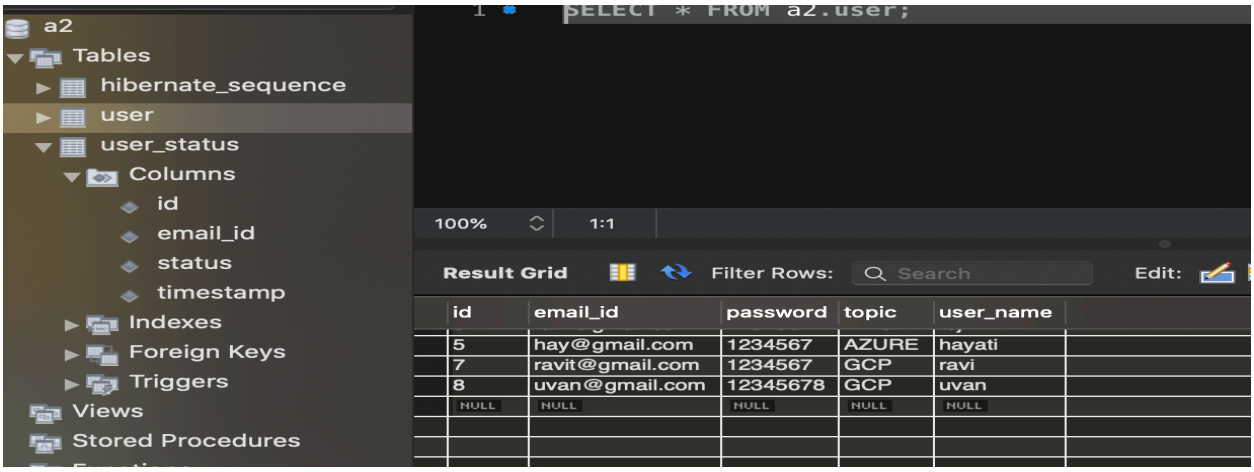

Figure 5:User registered in Database

c)Login Docker image:

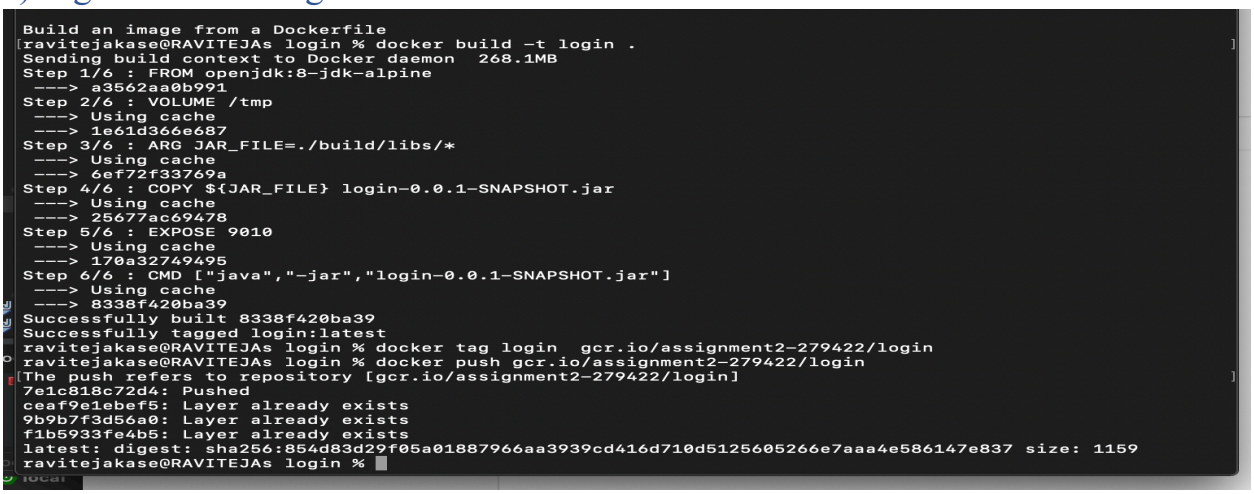

Figure 6:Login Docker image

Login Code snippet and Change state to Online:

```
@CrossOrigin
@RestController
public class LoginController {
	@Autowired
	private LoginService service;
	@Autowired
	private UserStatusService userStatusService;
	@PostMapping("/login")
	public User loginUser(@RequestBody User user) throws Exception {
		String tempEmailId = user.getEmailId();
		String tempPassword = user.getPassword();
		User userObjUser = null;
		if (tempEmailId != null && tempPassword != null) {
			userObjUser = service.fetchUserByEmailIdAndPassword(tempEmailId,
tempPassword);
		}
		if (userObjUser == null) {
			throw new Exception("Bad Credentials");
		}
		Date date = new Date();
		long time = date.getTime();
		Timestamp ts = new Timestamp(time);
		UserStatus statusObj = userStatusService.fetchUserByEmailId(tempEmailId);
		if (statusObj == null) {
			UserStatus sUserStatus = new UserStatus();
			sUserStatus.setEmailId(tempEmailId);
			sUserStatus.setStatus("online");
			sUserStatus.setTimestamp(ts);
			userStatusService.saveUserStatus(sUserStatus);
```

```
            } else {
                userStatusService.updateStatus(tempEmailId, "online", ts);
            }
            return userObjUser;
        }
}
```

d)State changed to Online in Database:

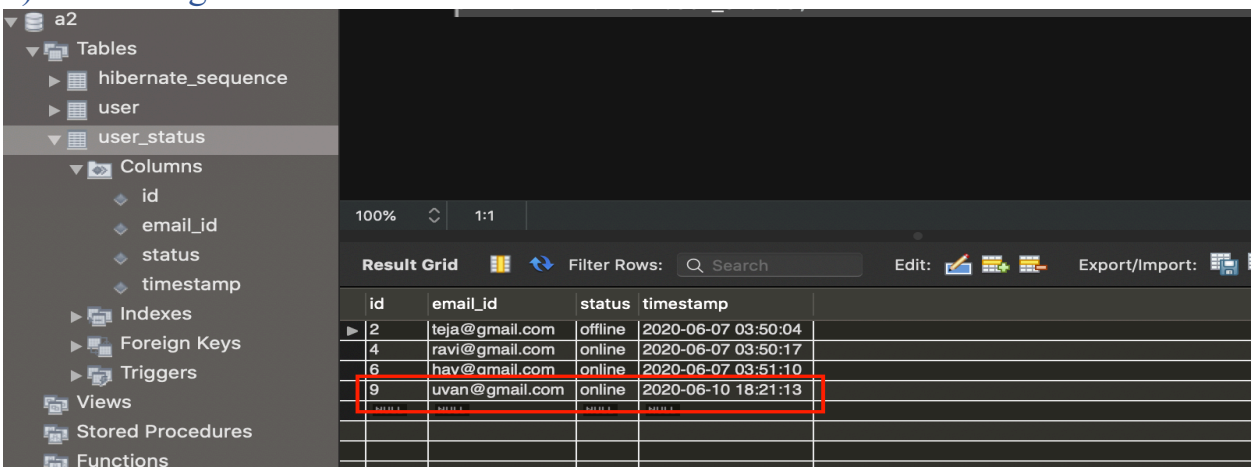

Figure 7:State changed to Online

e) Database contains only two tables and DB scripts are pasted in the submission folder.

f)Stateinfo Docker image:

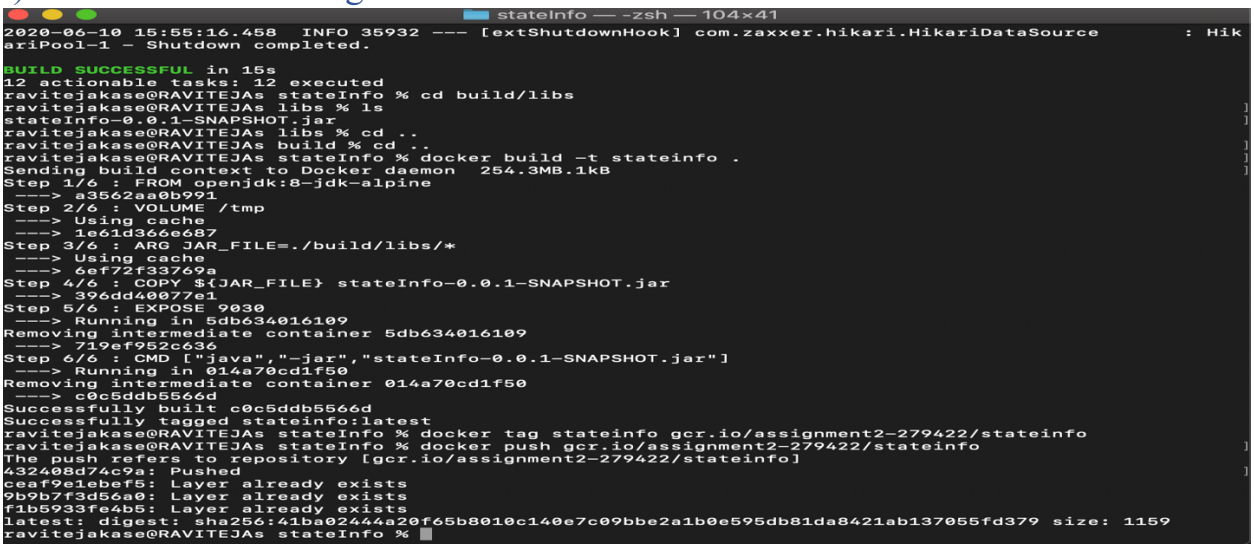

Figure 8:Stateinfo docker

Stateinfo Code snippet:

```
@CrossOrigin
@RestController
public class LogoutController {
	@Autowired
	private LoginService service;
	@Autowired
	private UserStatusService userStatusService;

	@PostMapping("/logout/{emailId}")
	public void loginUser(@PathVariable String emailId) {
		Date date = new Date();
		long time = date.getTime();
		Timestamp ts = new Timestamp(time);
		userStatusService.updateStatus(emailId, "offline", ts);
	}
	@GetMapping("/onlineUsers/{emailId}")
	public List<User> fetchAllOnlineUsers(@PathVariable String emailId) {
		List<User> onlineUsers = null;
		onlineUsers = userStatusService.fetchAllStatus("online", emailId);
		return onlineUsers;
	}
}
```

Stateinfo changed to Offline in Database:

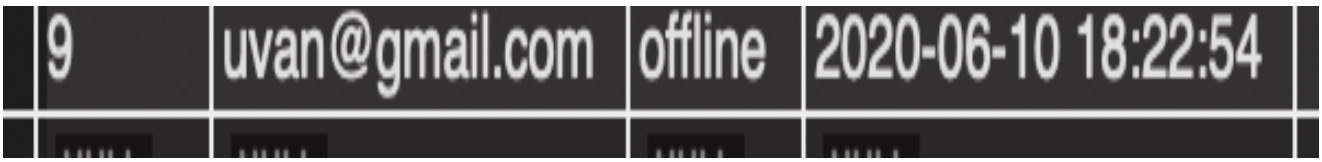

Figure 9:Stateinfo changed to Offline in database

g) Google Cloud run:

Docker images are tagged in Google Cloud registry, using Google Cloud run images are run in GCP.

Images in Google Cloud Registry[4]:

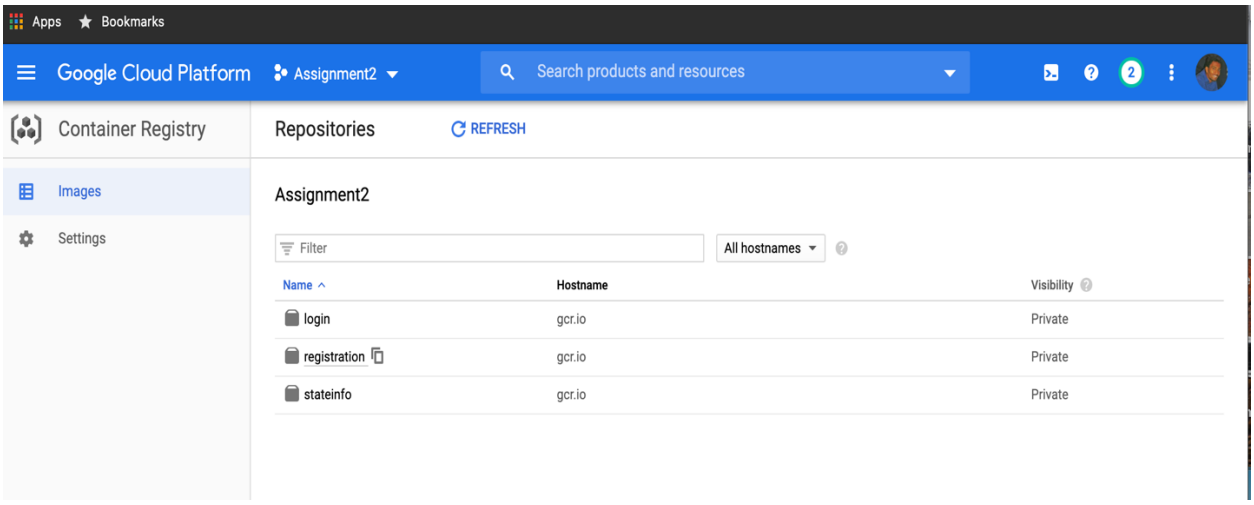

Figure 10:Containers registered in GCR

Registration container running using Google Cloud Run [5]:

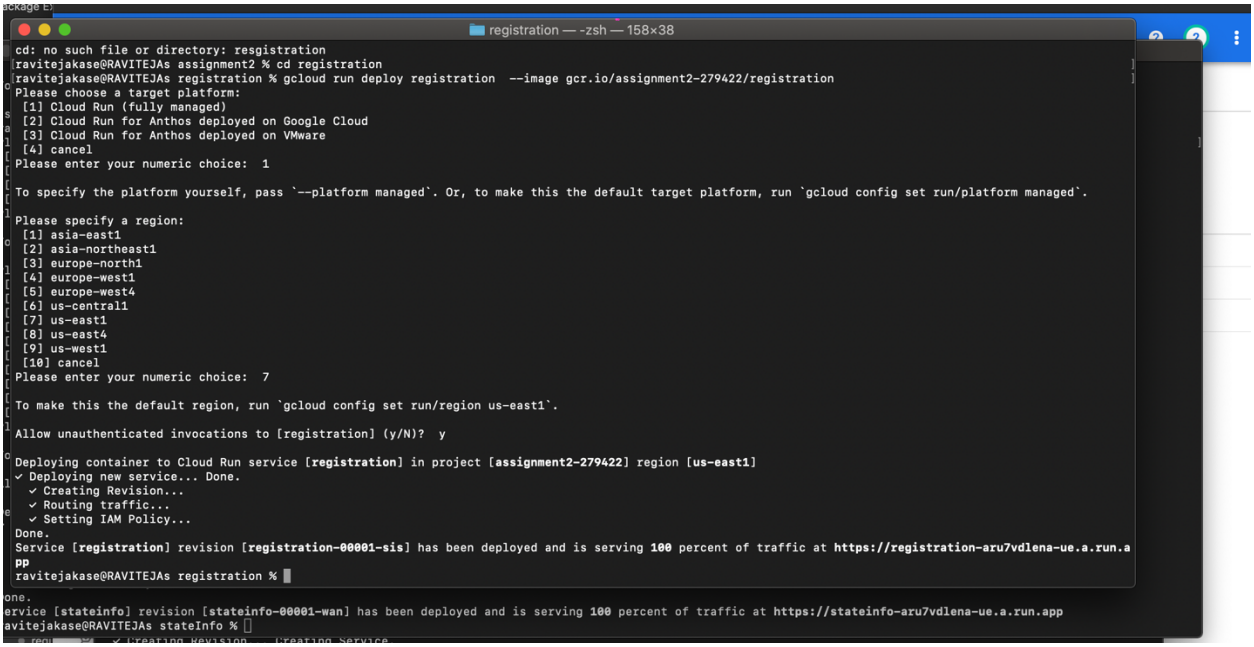

Figure 11:Registration container running using Google Cloud Run

Login container running using Google Cloud Run [5]:

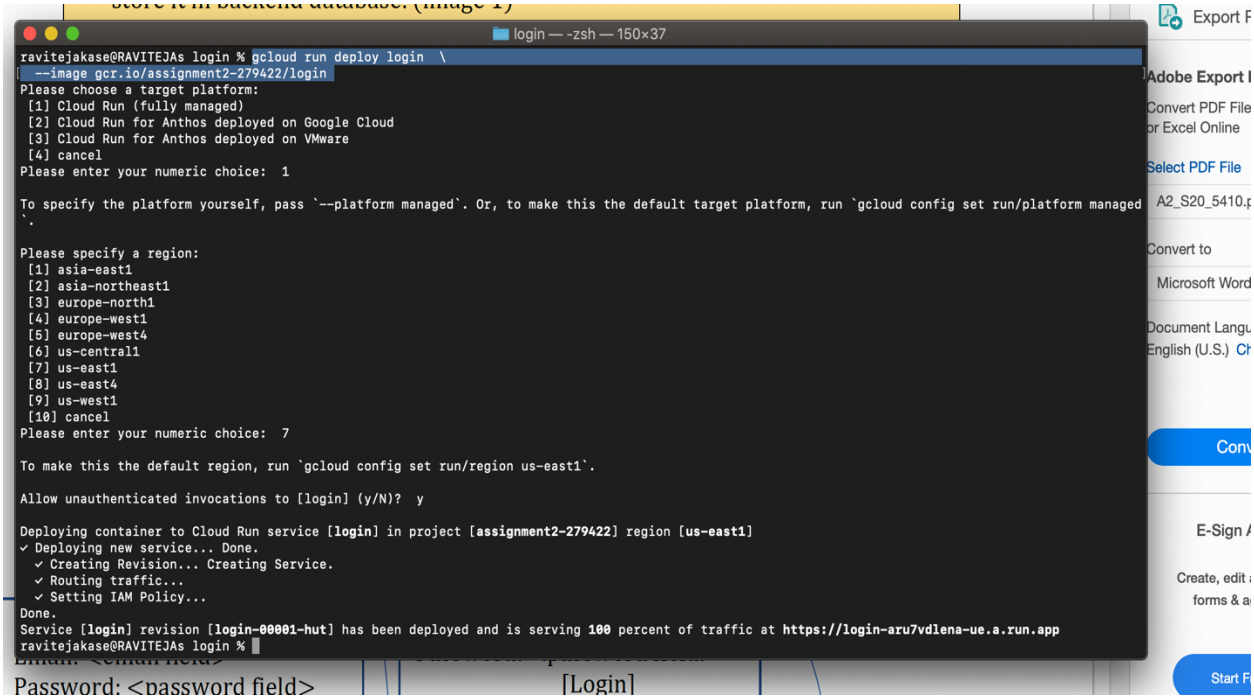

Figure 12:Login container running using Google Cloud Run

Statinfo container running using Google Cloud Run[5]:

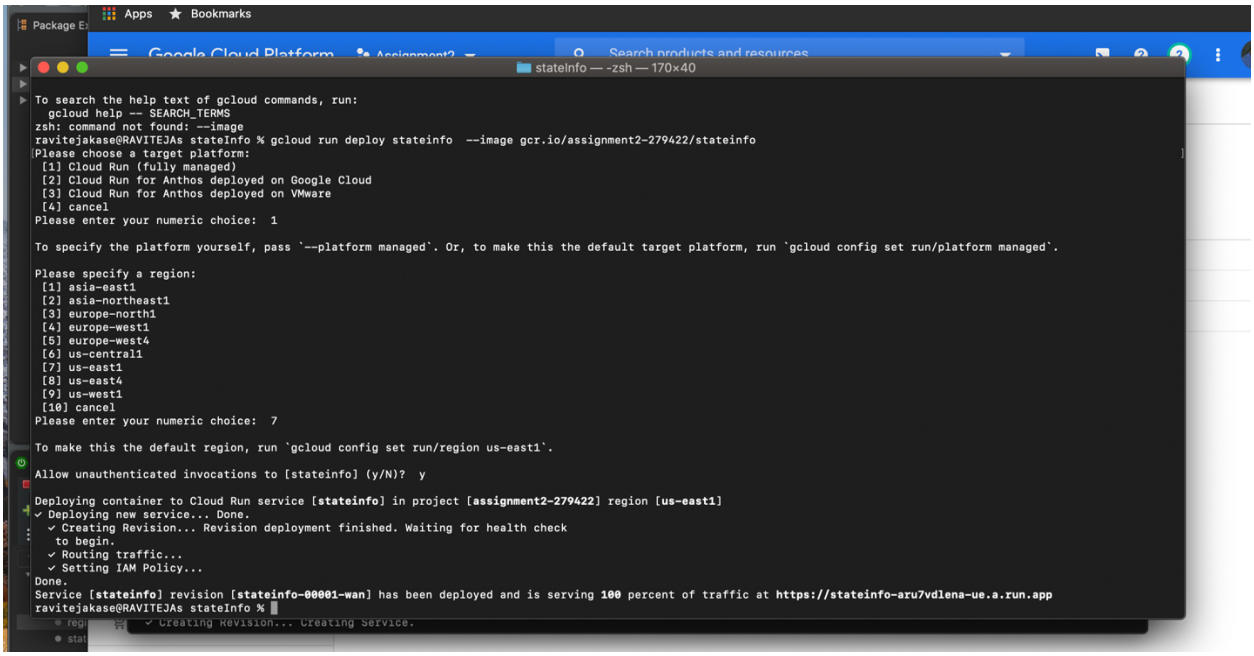

Figure 13:Stateinfo container running using Google Cloud Run

Containers running in Google Cloud run[5]:

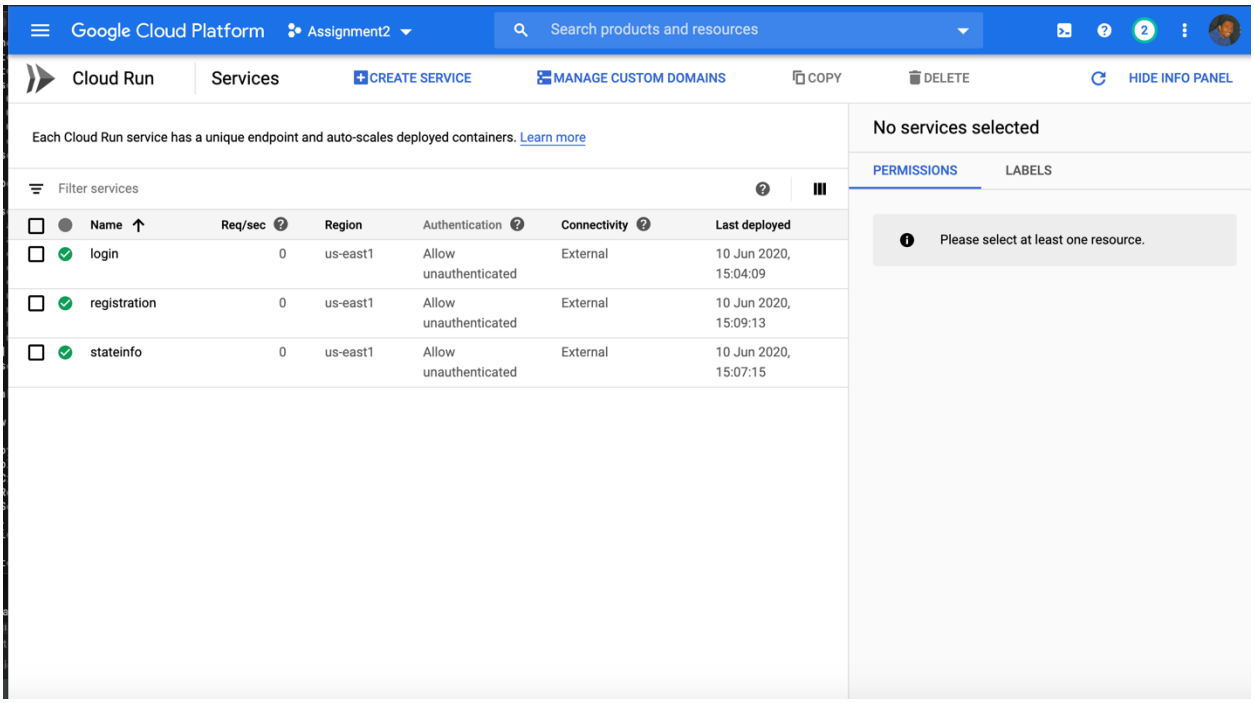

Figure 14:Containers running in Google Cloud run

h) Webpages:

Front end is built in angular. Code is placed in the submission folder. Please install node modules by running “npm install” command for node modules .

Registration:

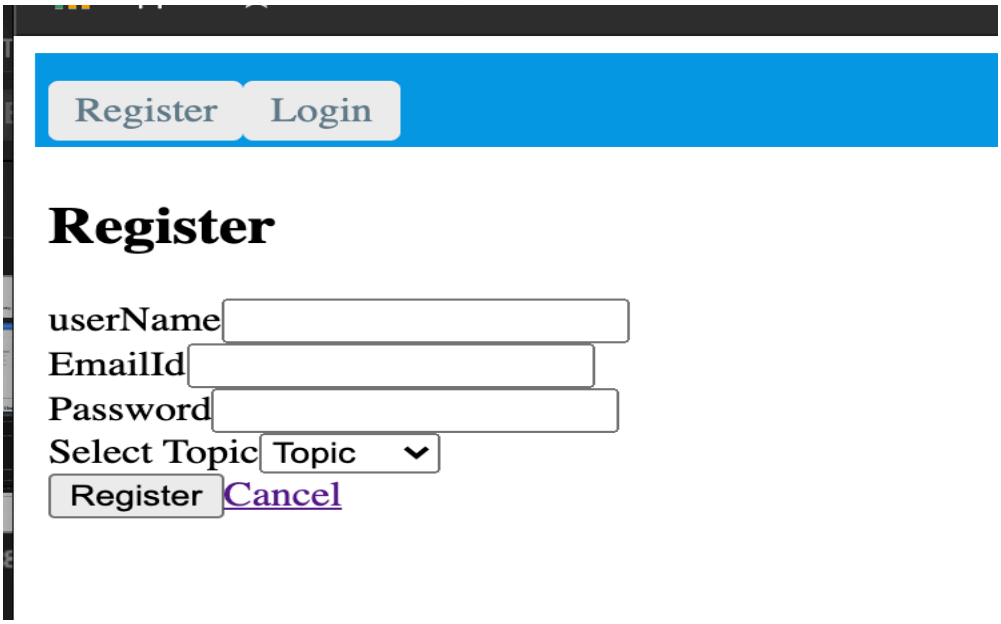

Figure 15:Registration

Login:

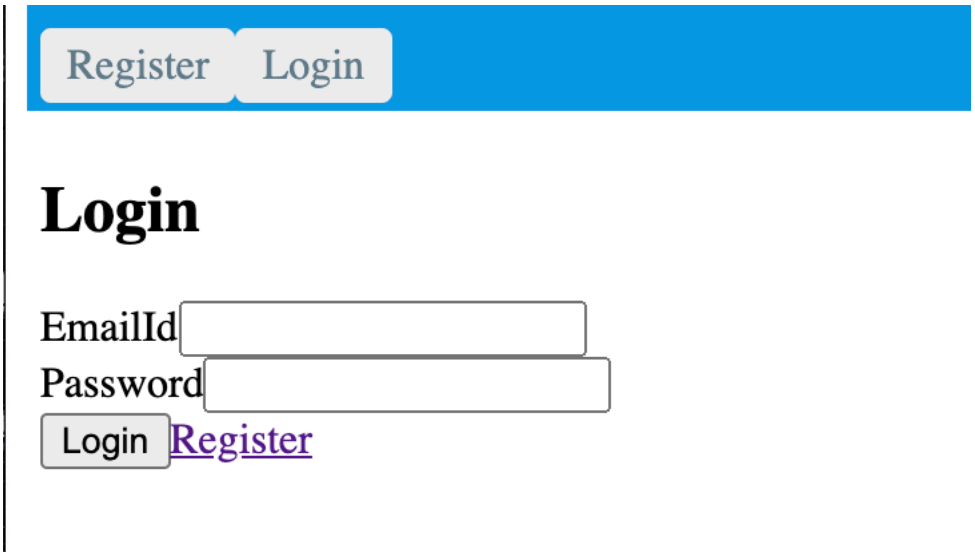

Figure 16:Login

i) Test cases:

Test Case Id	Test Case Scenario	Test steps	Test Data	Expected Results	Actual Result	Pass/Fail
TU01	Registration	Click on Register	Enter User name, Email, Password, Select topic from drop down	After entering the details, and click on register, if registration is successful should navigate to Login page	As expected	Pass
TU02	Login	Click on Login	Enter email and password	After click of login button, user state should be changed to online	As expected	Pass
TU03	Fetch online users	After clicking Login	No Input	Should fetch all other online users	As expected	Pass
T004	Logout	Click on Logout	No Input	User state must be	As expected	Pass

				changed to Offline		

TU01 (Registration)

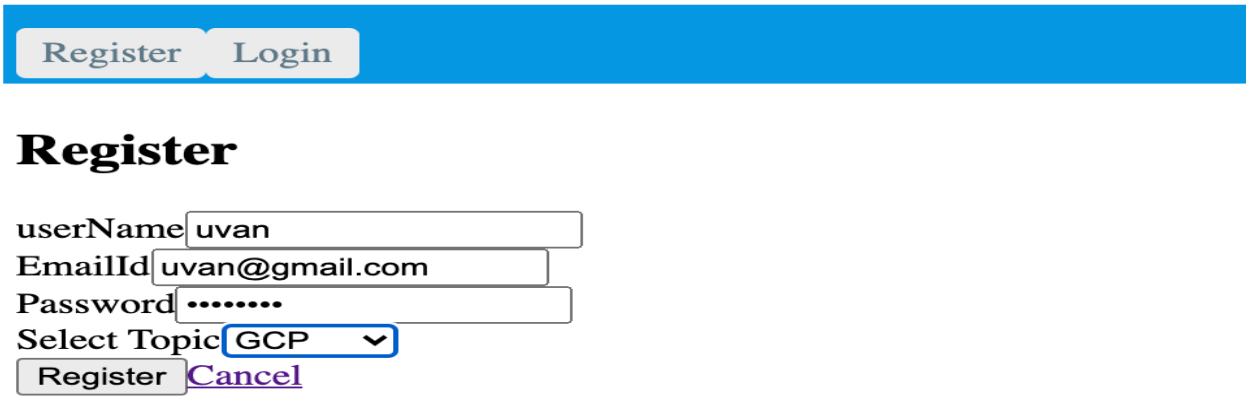

Figure 17:Registration

TU02 (Login):

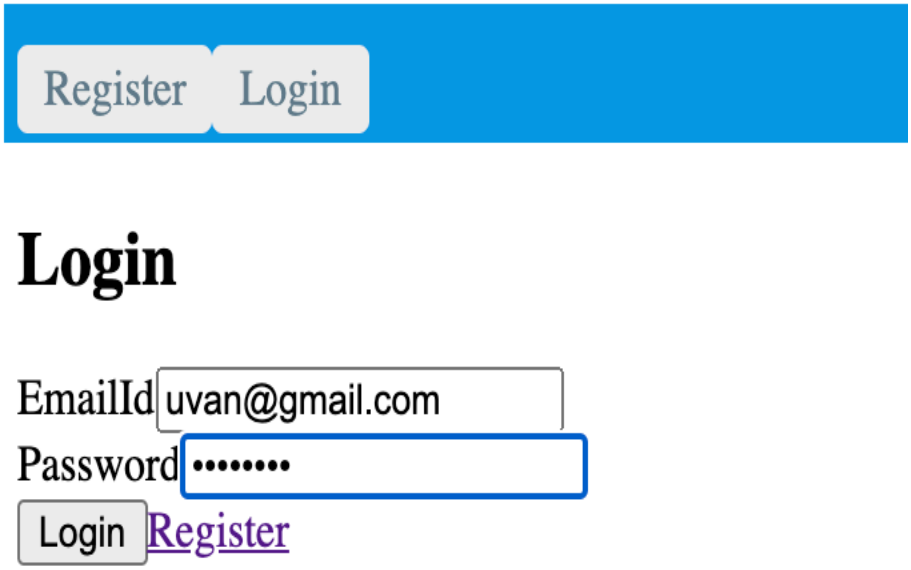

Figure 18:Login

TU03 (Fetch Online Users):

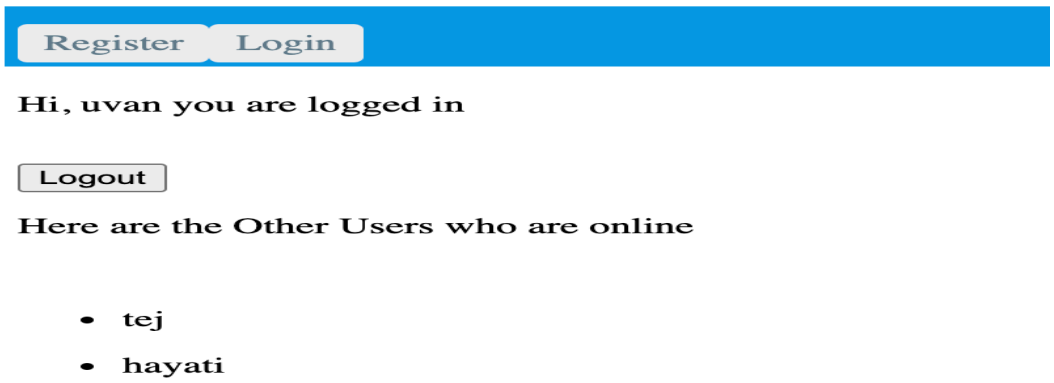

Figure 19: Fetch Online Users

TU04 (Logout):

Figure 20: After Clicking Logout User state changed to offline

j) Summary:

Google Cloud run:[7]
Google Cloud runs abstracts the infrastructure management and auto-scaling and it enables us to run containers written in any language or to run containers present in Google Cloud registry. Using Google Cloud run app development and deploying is easier.

Google Container Registry:[8]
GCR provides the functionality to place all docker images. From GCR, images can be run using GKE or the in-app engine or in Google Cloud run. Google Cloud runs directly works with the Docker command line. Managing container images is easy and it can be done through the console.

Docker Containers:[9]
Docker containers are stand-alone packages, which can be built locally and run anywhere irrespective of hardware specifications. Docker containers follow works as built once and run anywhere. Docker isolates software requirements and ensures it runs even though there are differences concerning operating systems.

Usage of technologies in the application built:
Dockerfiles are created for each API. Docker images are built locally. Such built images are tagged to the Google Cloud Registry. To access G cloud locally, CloudSDK is installed and configured to the project as default. Tagged images are pushed to the Google Cloud Registry. Using Google Cloud run, container images are run from the registered images in the Google Container registry. Endpoints from Google Cloud are utilized in the angular application developed.

Part B

a)Chatbot for Delivery and Take away are created:

Delivery

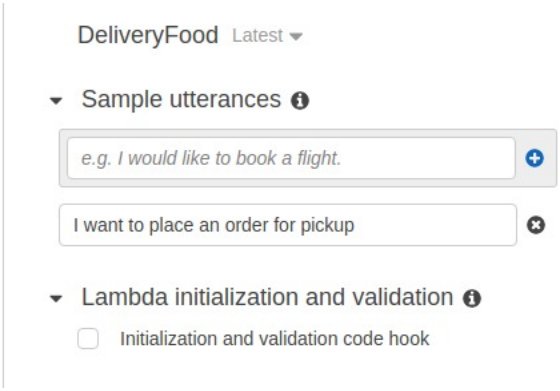

Figure 21:Delivery Food Chatbot

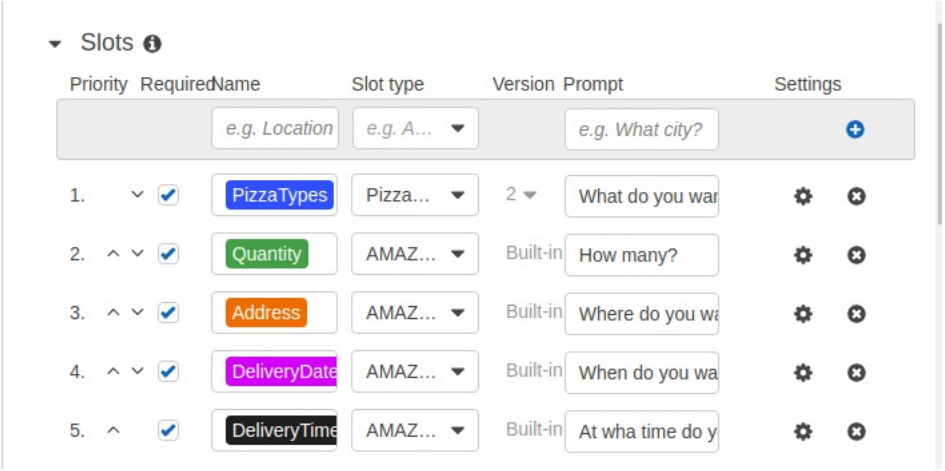

Figure 22:Delivery details

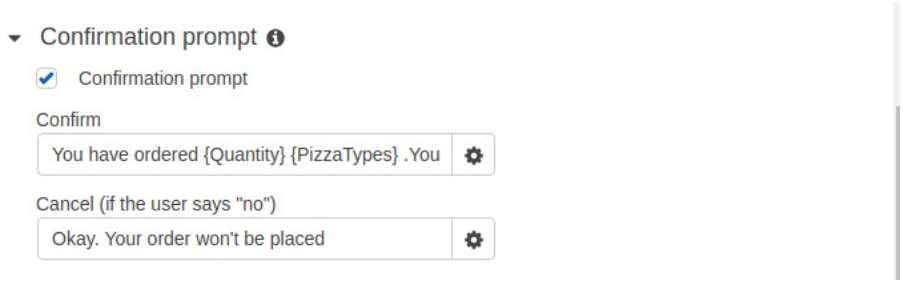

Figure 23:Confirmation Prompt

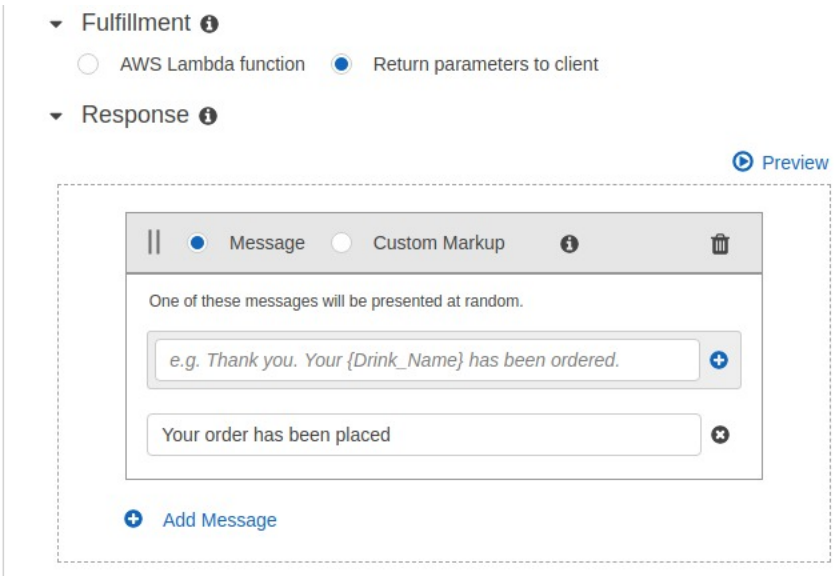

Figure 24:Response builder

Pickup

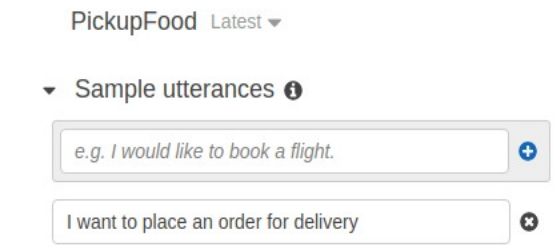

Figure 25:Pickup Food

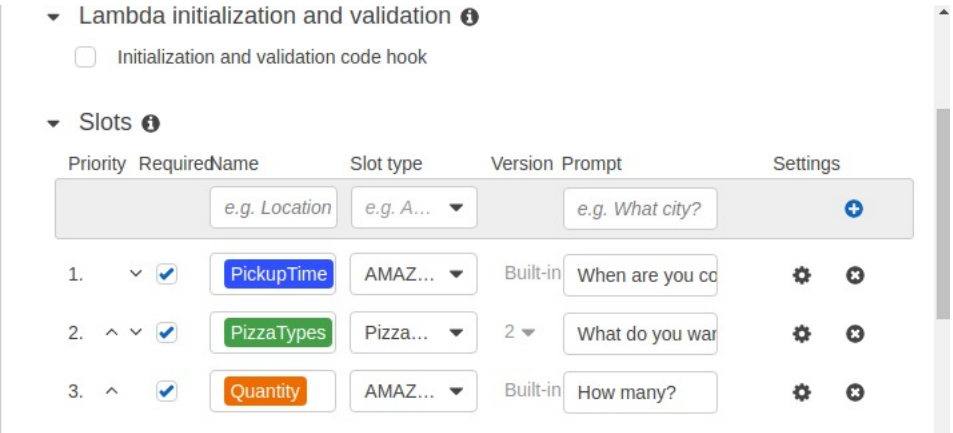

Figure 26:Pickup times

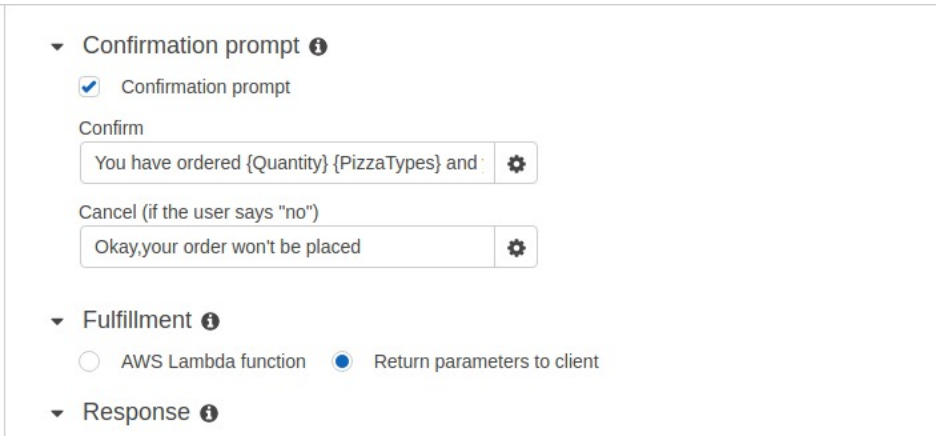

Figure 27:Confirmation Prompt

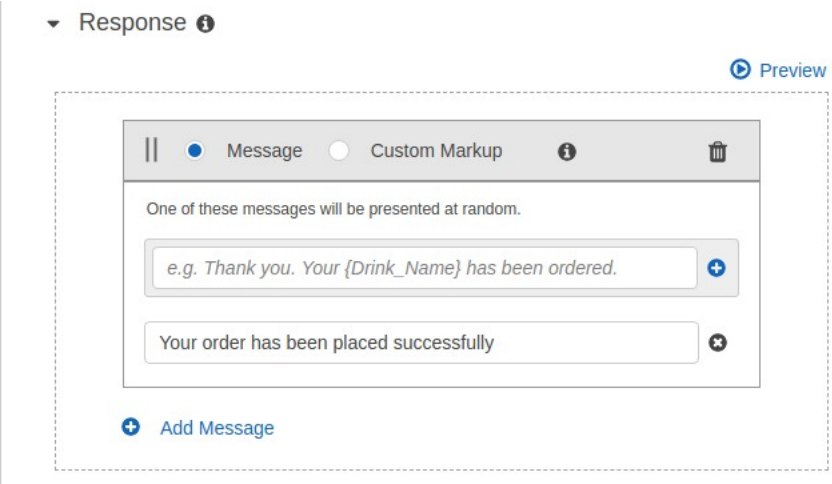

Figure 28:Ordered Message

b)Pizza types

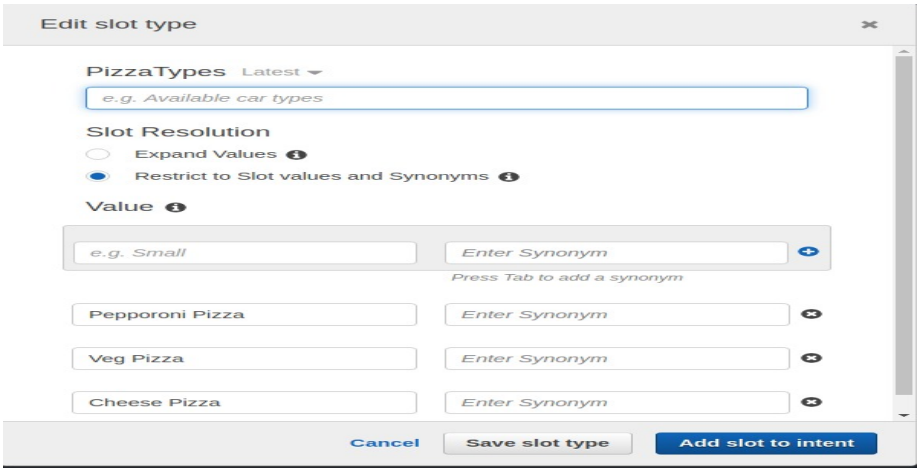

Figure 29:Pizza types

c) Chatbot accepts delivery or pickup based on the user input.

d)Delivery Testing:

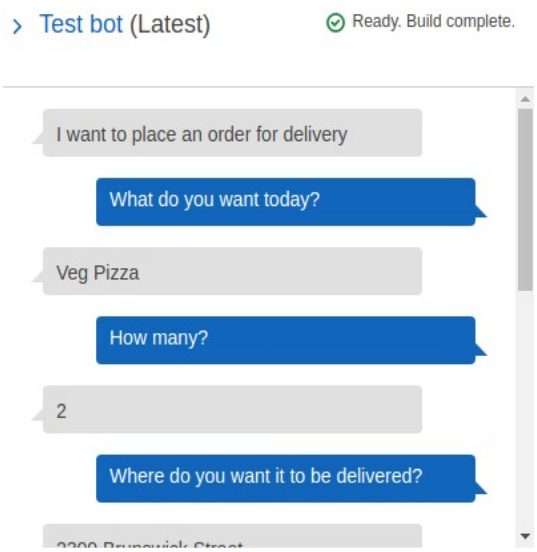

Figure 30:Test-1

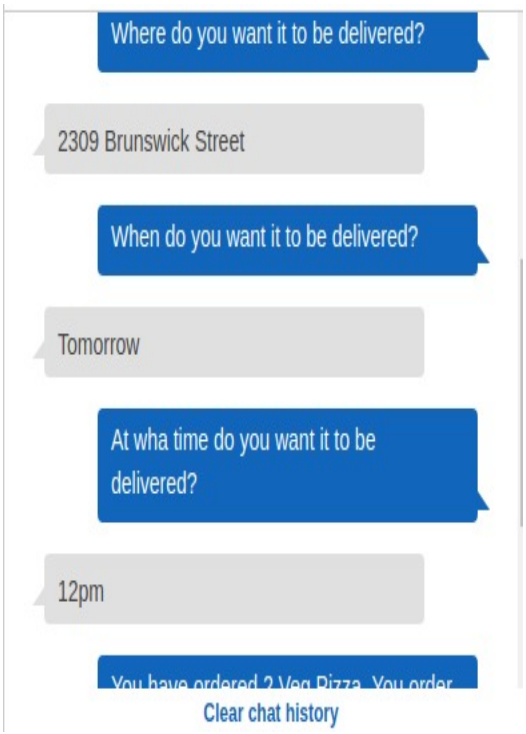

Figure 31:Test-2

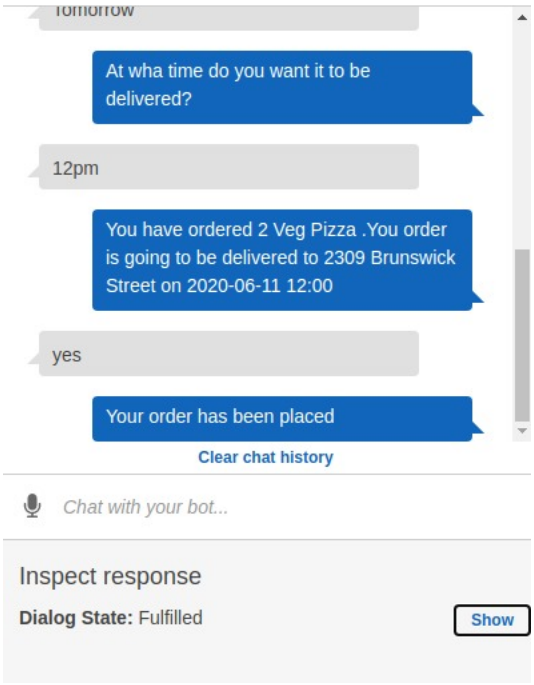

Figure 32:Test-3

e) Take away testing:

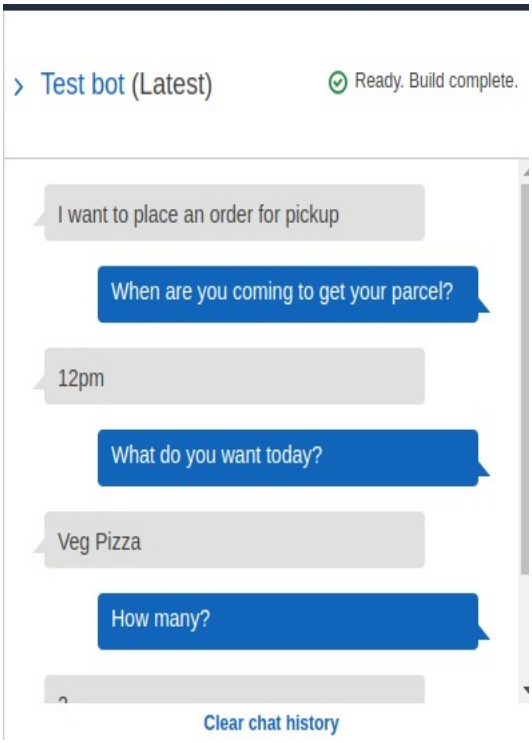

Figure 33:Take away test-1

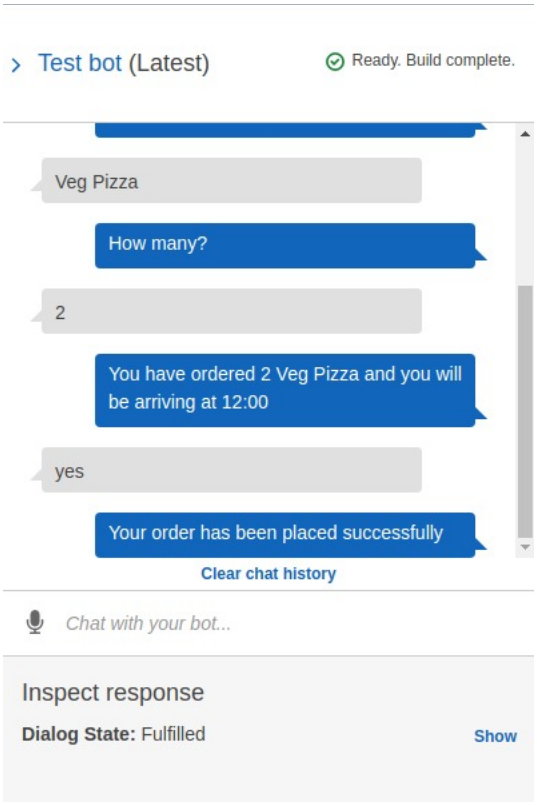

Figure 34:Take away testing -2

Operation Performed:[10]

Amazon Lex is used to building two chatbots, Pickup and delivery for Pizza orders. For each order type, utterances are created and slots are created. For pickup type pickup time, pizza type and quantities are created. For delivery type pizza type, address, quantity, delivery time and data are created. Confirmation message for an order is created for both the types and similarly response messages are created for final orders. Chatbots are tested successfully.

References:

[1] “Quickstart for Cloud SQL for MySQL | Google Cloud,” *Google Cloud*, 2020. [Online]. Available: https://cloud.google.com/sql/docs/mysql/quickstart. [Accessed: 03-Jun-2020].

[2]D. Kabii, “Developing a RESTful API With Spring Boot, Gradle, Groovy, JPA and MySQL in an hour,” *Medium*, 29-Jul-2019. [Online]. Available: https://medium.com/@david.kabii/developing-a-restful-api-with-spring-boot-gradle-groovy-jpa-and-mysql-in-2-hours-hour-1-a2066a141152. [Accessed: 04-Jun-2020].

[3] Sairam Krish, “Docker for Spring boot | Gradle | Java micro service,” *Medium*, 22-May-2018. [Online]. Available: https://medium.com/@sairamkrish/docker-for-spring-boot-gradle-java-micro-service-done-the-right-way-2f46231dbc06. [Accessed: 06-Jun-2020].

[4] “Pushing and pulling images | Container Registry Documentation,” *Google Cloud*, 2020. [Online]. Available: https://cloud.google.com/container-registry/docs/pushing-and-pulling. [Accessed: 06-Jun-2020].

[5] “Quickstart: Build and Deploy | Cloud Run Documentation | Google Cloud,” *Google Cloud*, 2020. [Online]. Available: https://cloud.google.com/run/docs/quickstarts/build-and-deploy#java. [Accessed: 06-Jun-2020].

[6] JavaInUse, “Angular 7 + Spring Boot Login Example,” *YouTube*. 2020.

[7] “Cloud Run Documentation | Google Cloud,” *Google Cloud*, 2020. [Online]. Available: https://cloud.google.com/run/docs. [Accessed: 07-Jun-2020].

[8] “Container Registry | Google Cloud,” *Google Cloud*, 2020. [Online]. Available: https://cloud.google.com/container-registry. [Accessed: 08-Jun-2020].

[9] “What is a Container? | Docker,” *Docker*, 2013. [Online]. Available: https://www.docker.com/resources/what-container. [Accessed: 08-Jun-2020].

[10] “Module 3 - May 25, 2020 - CSCI5410 - Serverless Data Processing (Sec 1) - 2020 Summer,” *Brightspace.com*, 2020. [Online]. Available: https://dal.brightspace.com/d2l/le/content/124063/viewContent/1706936/View. [Accessed: 09-Jun-2020].