**Assignment 2 - CSCI 5410**

Arthy Umapathy

B00840100

**Part A - Build, Deploy, and run a Container Application on GCP**

**Introduction:**

An application was created using frontend HTML and backend node js with MySql as database and deployed onto Google cloud with the use of docker containerization. Three containers were used to serve the application were in each container running the backend was responsible for performing a certain CRUD operation interaction with the MySql database. Although the three containers work synchronously, they do not interaction with each other instead they perform a specific task with the database.

The frontend application is an online meeting application were in a user logs in or registers to the site to view other users who are logged in as well. The user details are stored in the database through backend in the registration page while the details are retrieved and validated during the log in task. The state and the session time of the users help to provide the list of users who are logged in currently.

The unit testing was performed before and after containerization and deployment of docker images on to the cloud. The testing was based on both the frontend and backend. The frontend testing includes components such as form validation, login request verification, registration check etc. whereas the backend testing included components such as database connection verification, api calling and fetching testing, get and post method verifications, asynchronous fetch services, consistency of redirections error handling etc.

Both the frontend and the backend have been connected, added as docker images and containerized and pushed onto the Google Cloud. Hence on deploying the application onto google cloud run both the front and the back end are deployed. Ultimately the whole application is containerized and deployed while the database in connected with the Amazon RDS instance.

**Steps followed:**

The following are the steps followed in order to containerize and deploy the application on to the Google Cloud Run using docker containerization.

1. **Create Docker containers:** In order to create 3 docker containers three different docker images for were built and each image was made available inside each container.

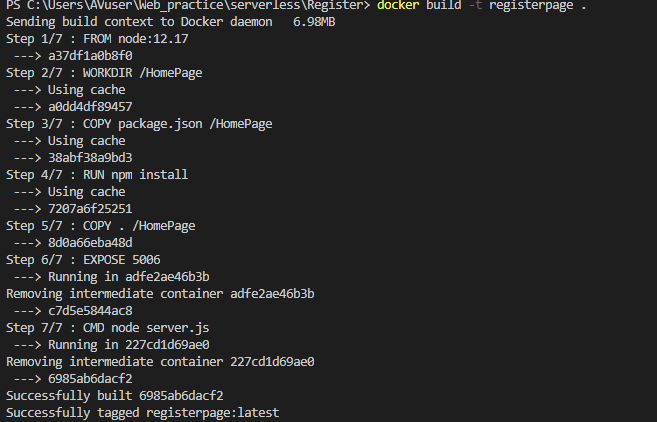
**Image1:** Registeration page docker image

Figure 1-Registration Page Docker Image

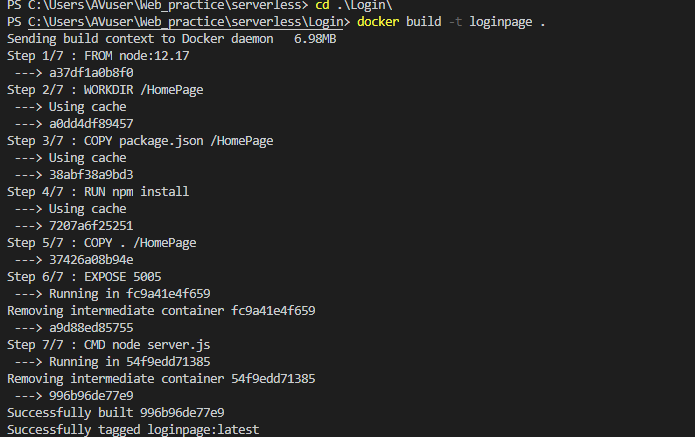
**Image 2:** Login page docker image

Figure 2- Login Page Docker Image

**Image 3:** Welcome page docker image

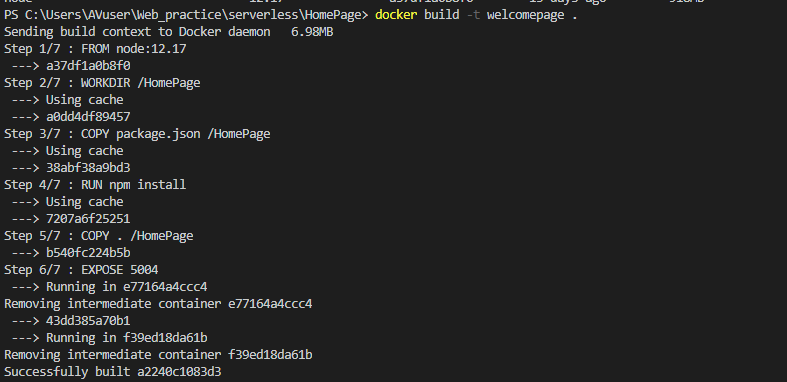


Figure 3- Welcome Page Docker Image

1. **Container 1:** The first Container is responsible for accepting registration details from frontend and store it in backend database.



Figure 4-Docker Container for image1

1. **Container 2:** The second Container is responsible for validating the login information.

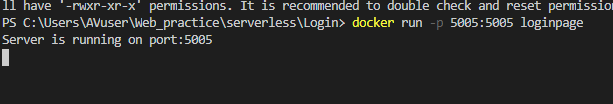


Figure 5-Docker Container for image2

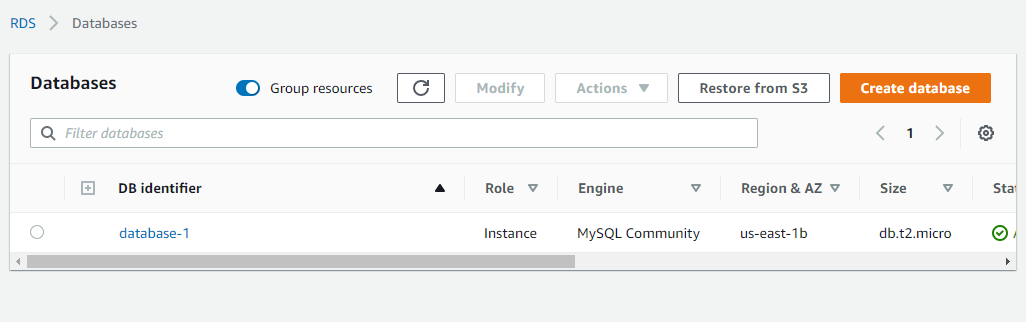
1. Once the user is logged in the state in the user state table changes to online and once logged out it changes to offline Figure 30-Updated state .
2. The 2 tables in the database used for storing user information are as follows. The user\_details table contains information such as user name, user emailid, user password, and the topic of meeting. The second table called the user\_state contains information such as the user emailid, their state (online/offline), and the timestamp of the session.

Figure 6-RDS database creation

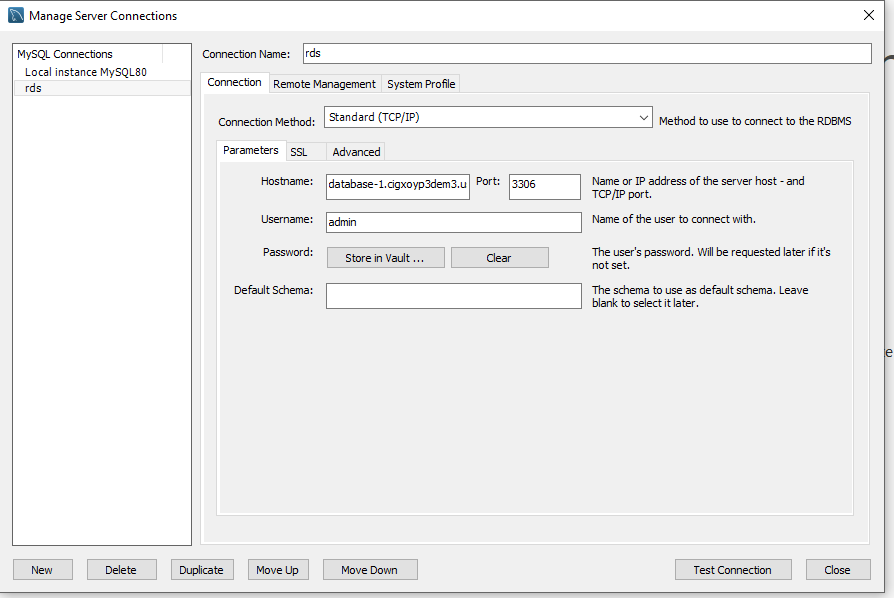
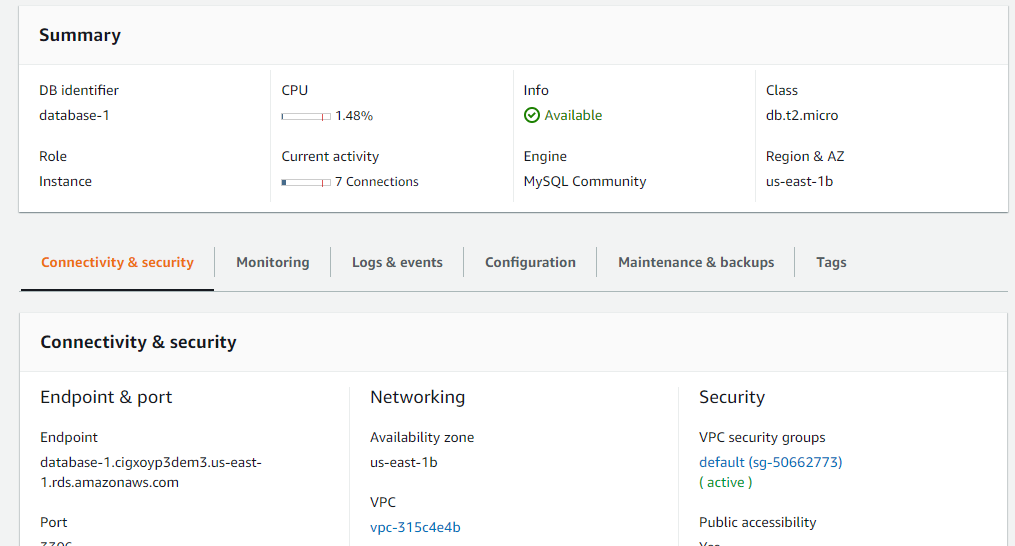


Figure 7-MySql connection

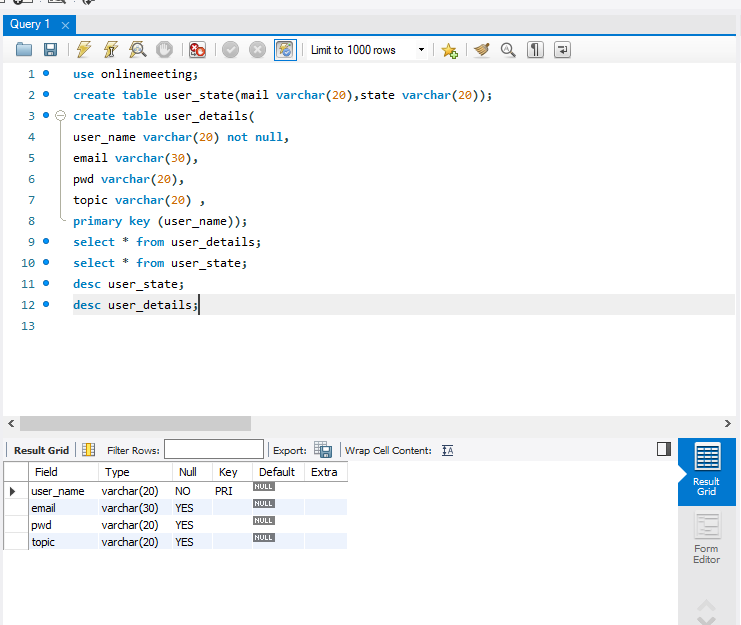


Figure 8-User\_details table Creation

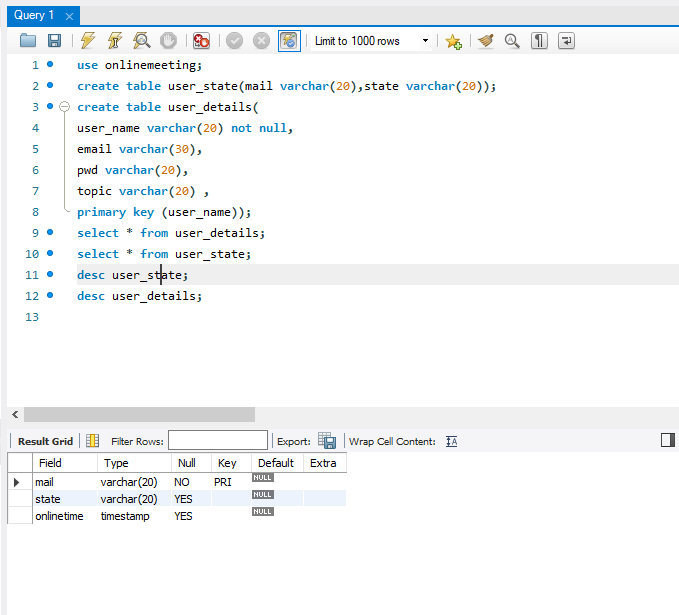


Figure 9-User\_State table creation

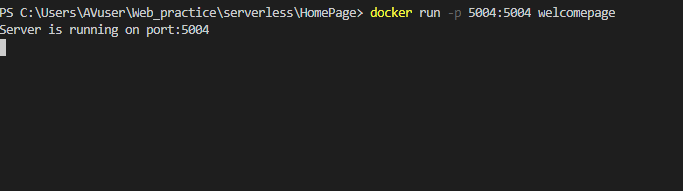
1. **Container 3:** The third Container is responsible for extracting information from database and displaying it on to the screen. 

Figure 10-Docker Container for image3

**List of Containers:**

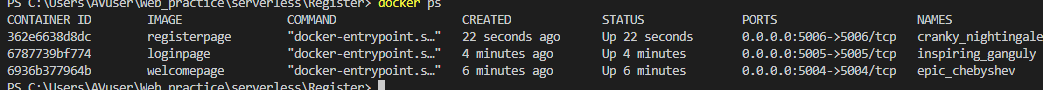


Figure 11-List of containers

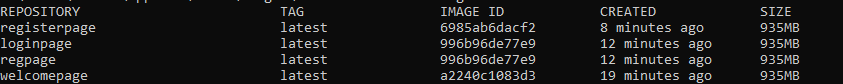
**List of Images:**

Figure 12-List of Images

1. After running the docker images using docker containers they are pushed on to the google container registry from which it is run using the google cloud run.

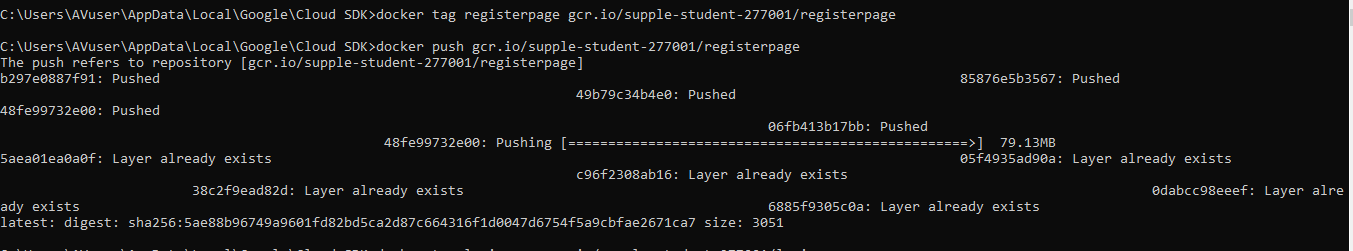
Pushing image1 to container registry

Figure 13-Image1 push to Cloud Registry

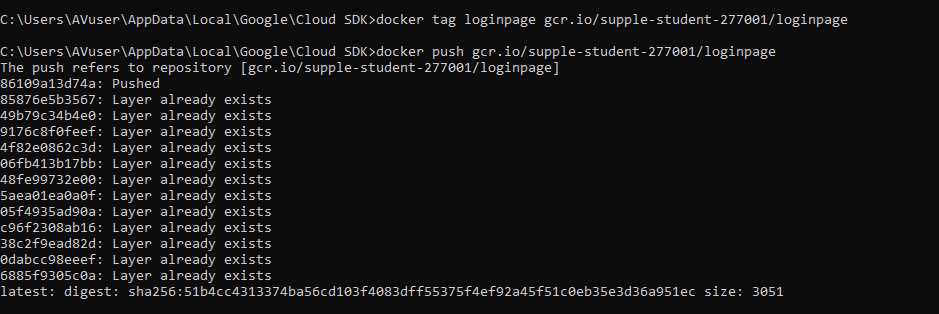
Pushing image2 to container registry 

Figure 14-Image2 push to Cloud Registry

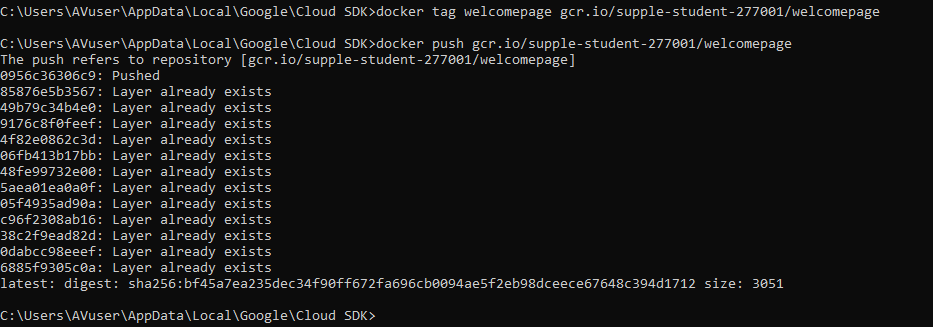
Pushing image3 to container registry 

Figure 15-Image3 push to Cloud Registry

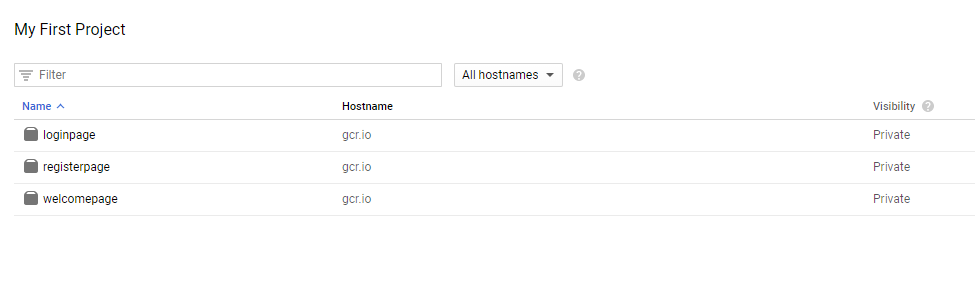
The Images are pushed to the container registry 

Figure 16-Images in Container Registry

Now from the container registry each image is deployed into the google cloud run.

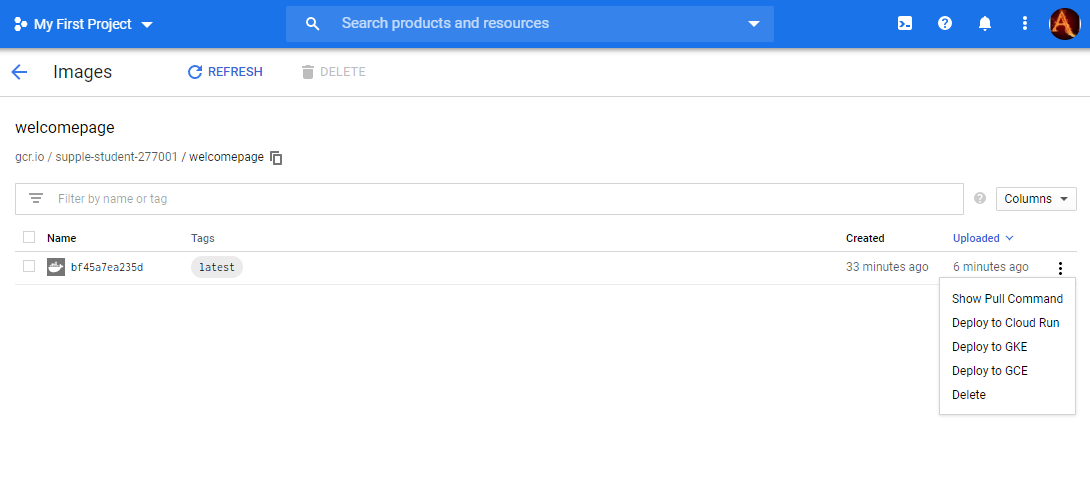
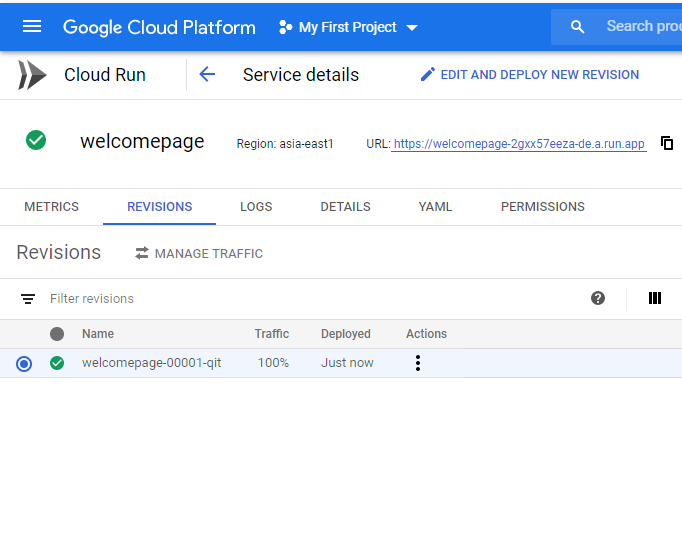
Deplyment of Image 3:

Figure 17-Image1 Deployment



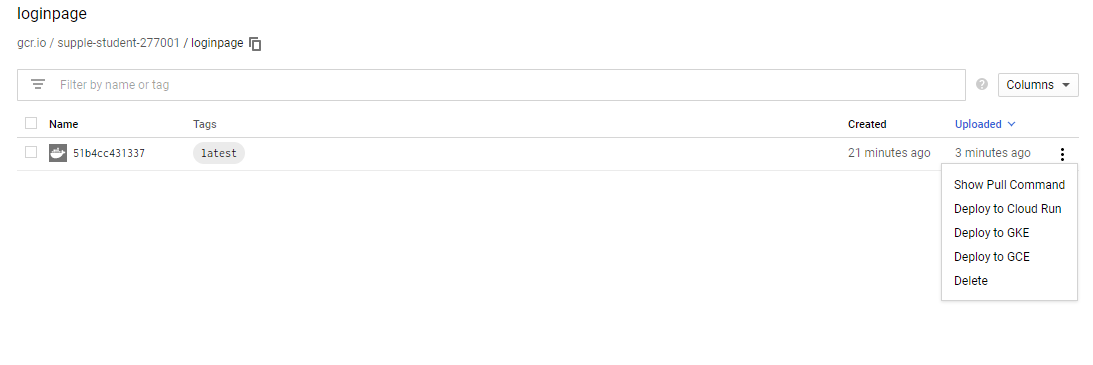
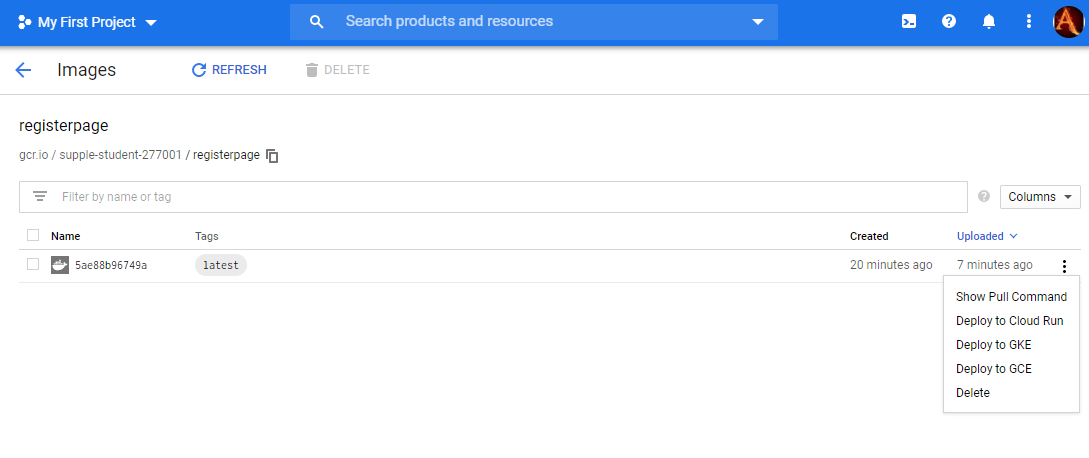
 

Figure 18-Image2 Deployment

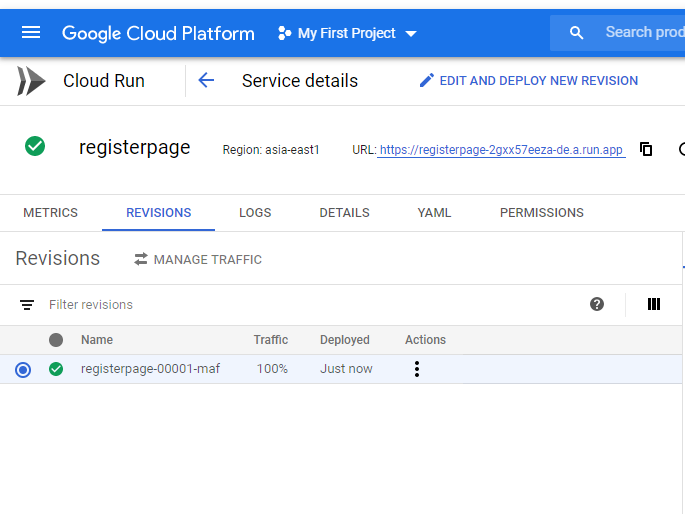


Figure 19-Image3 Deployment

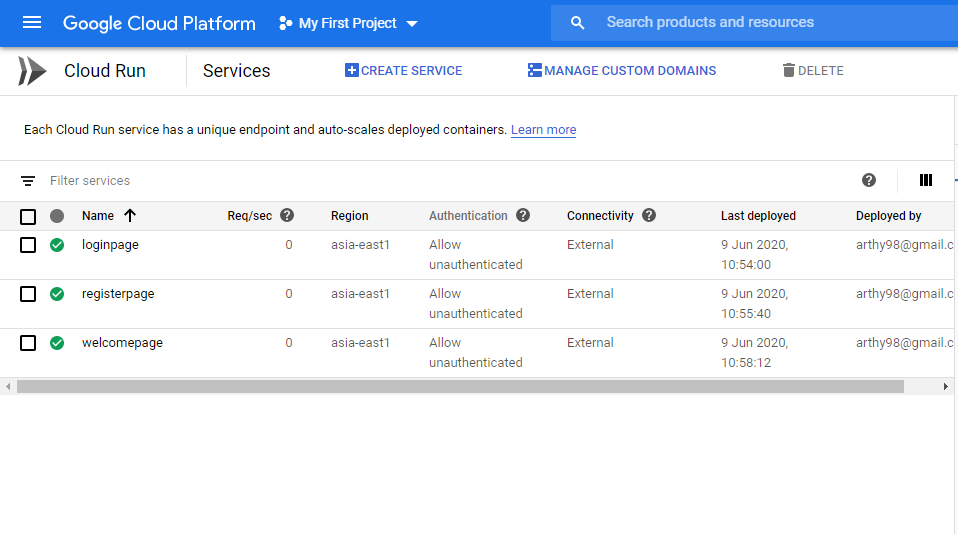


Figure 20-List of all deployed images in Google Cloud Run

1. The frontend UI design for the three pages was built using HTML and CSS while the backend logic functions on node js.

Image 1 running on GCR:

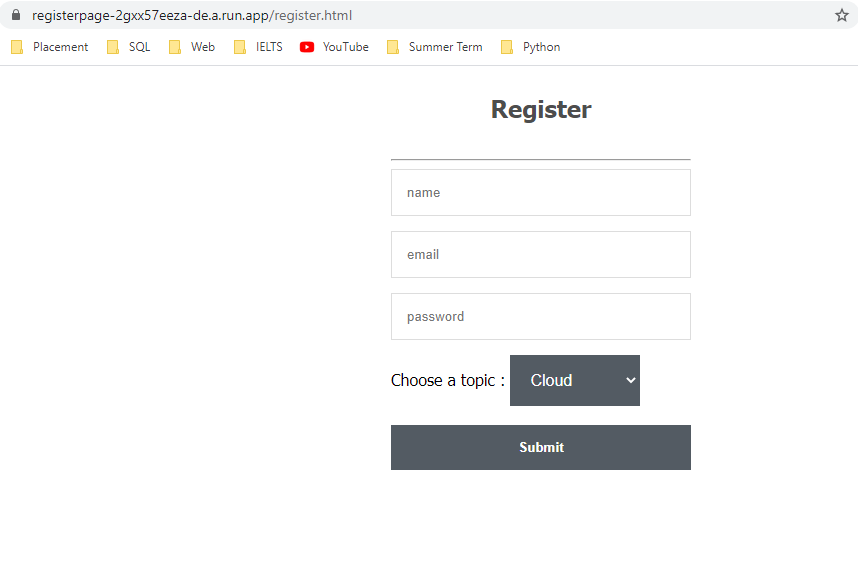


Figure 21-Image1 deployed

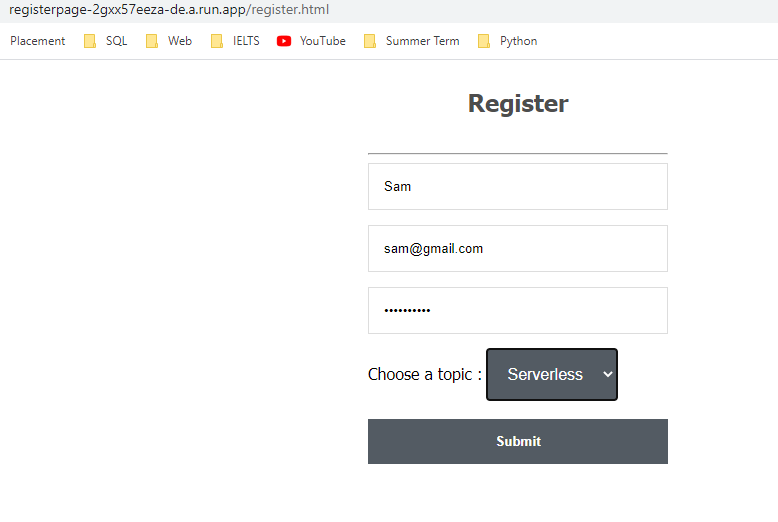




Figure 22-Nodejs logic on fetching

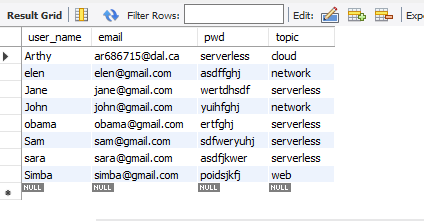


Figure 23-updated MySql database

New details added to user\_details api:

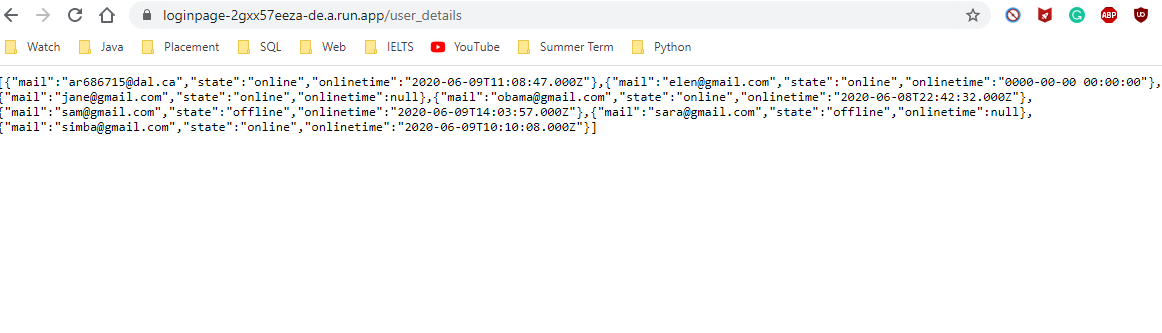


Figure 24-Updated Api

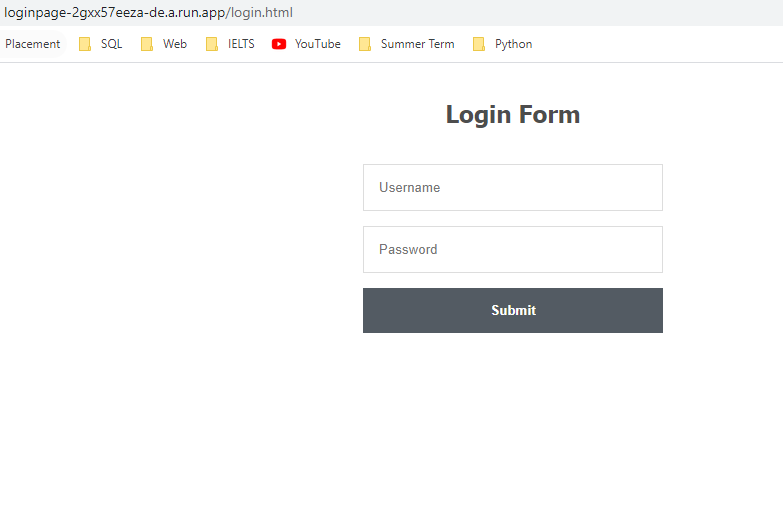
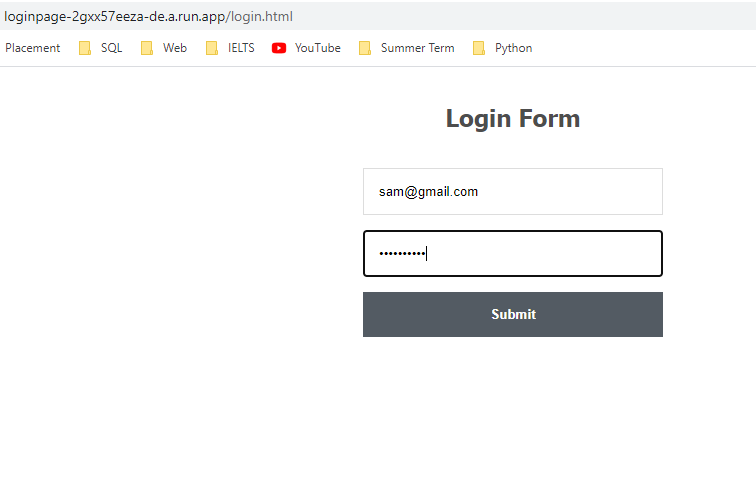


Figure 25-Image2 Deployed



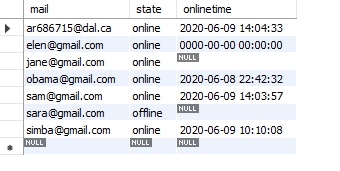


Figure 26-Update Mysql



Figure 27-Logic for validation

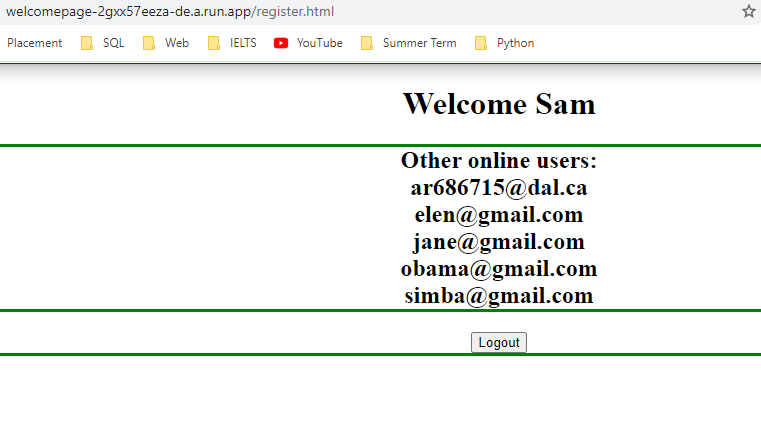
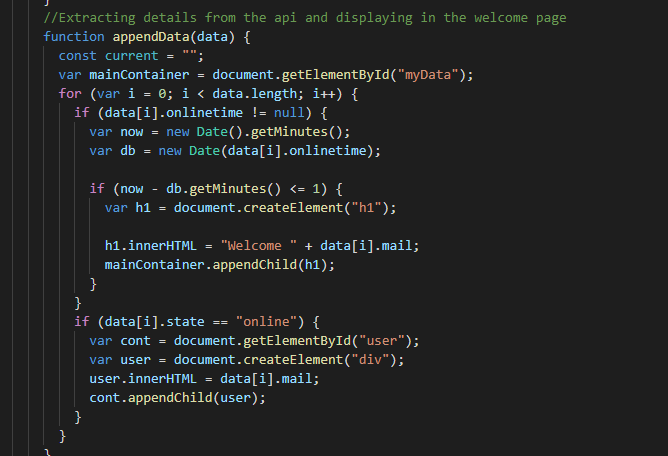


Figure 28-Image3 deployed



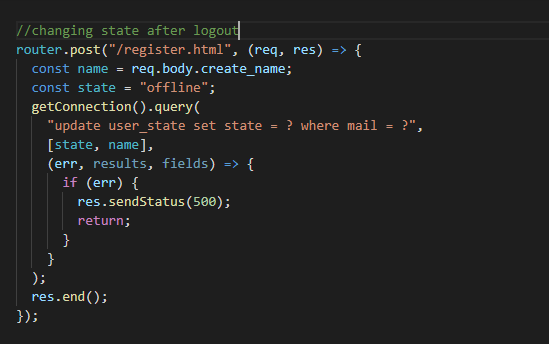


Figure 29-Logic for updating offline state

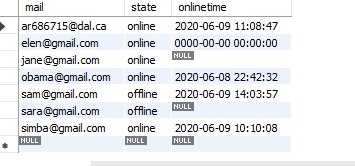


Figure 30-Updated state

1. **Test Cases:**

**BackEnd Test Cases:**

The Mysql database connectivity, api json object return, fetching promises have been tested and added within the source code. Some of them are listed below:

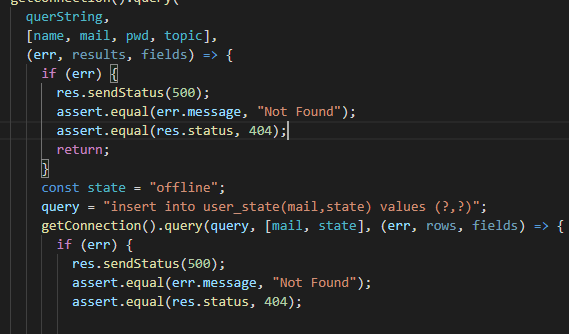


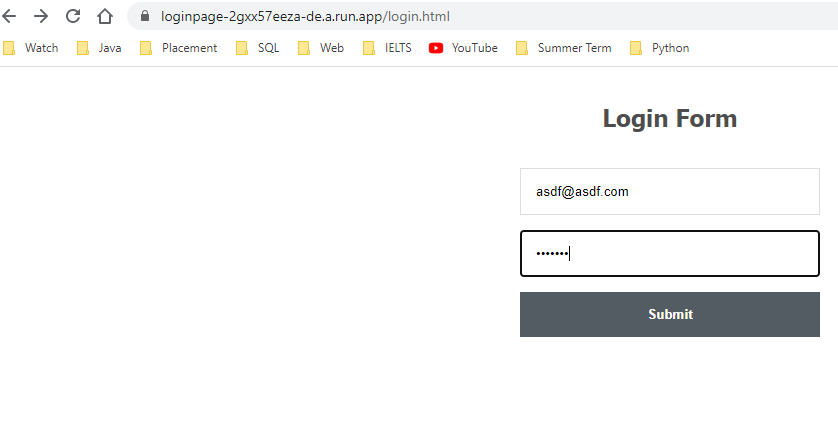
Figure -Backend testing

**FrontEnd Test Cases:**

1. Only the registered users are able to login



Figure -Frontend Login verification



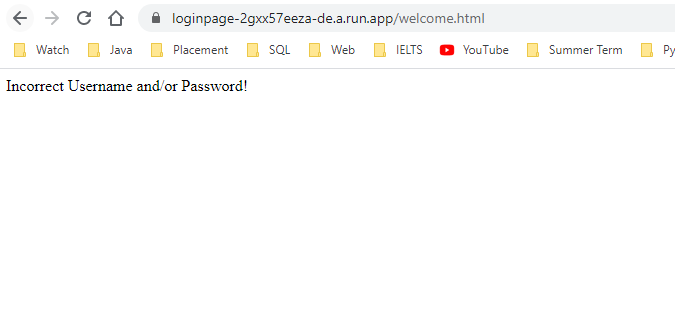


Figure -error message

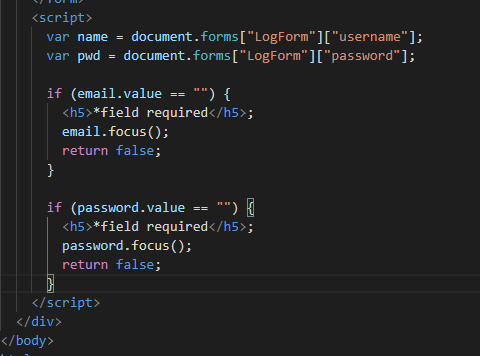
1. All the fields are required in the registration page. 

Figure -login form validation

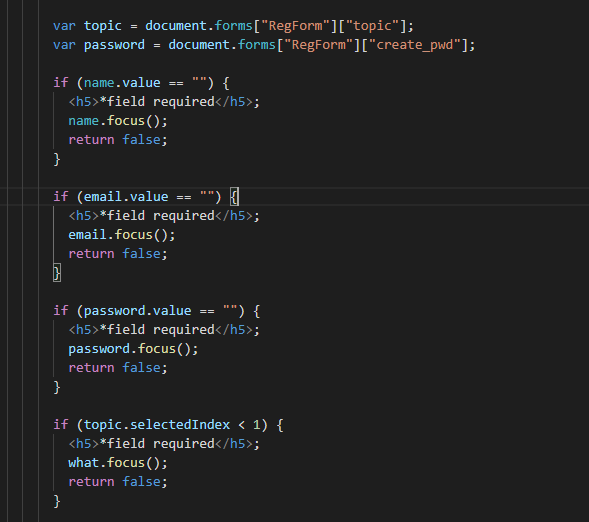


Figure -Registration Form validation

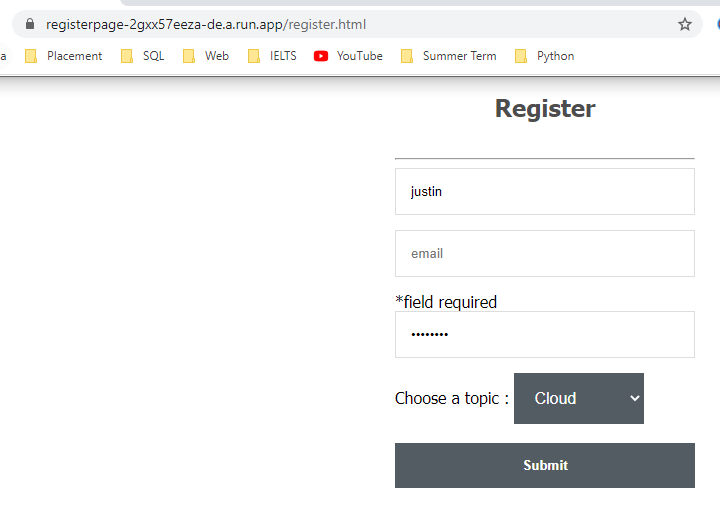
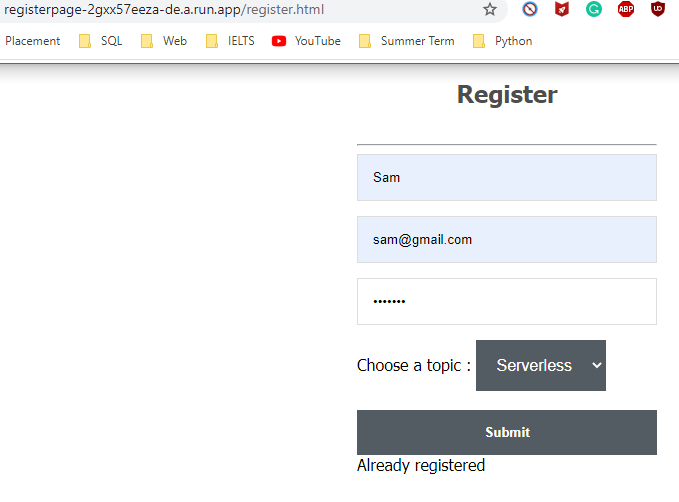


Figure -Error message

1. If the user is already registered, the registeration page does not allow the user to register again.
2. **Summary:**

The Google Cloud Run is used to deploy applications onto the cloud which once deployed can be removed but cannot be modified. Docker images which make the deployment easier is used to deploy the application on to the Google Cloud Run. Three separate images were created with each performing its own task. The images are made to run locally in docker containers. Each container is made to run on different ports. All the three containers were able to perform simultaneously were one was responsible for accepting registration details from frontend and store it in the database, the second was responsible for validating the login information and the third was responsible for extracting state information from the database. In order to deploy the docker images onto google cloud run the images were first pushed onto the google container registry from the local. Once the images were pushed on to the registry, they were deployed on to the Google Cloud Run. All the three images were able to run simultaneously. The first image url accepted the registration information and updated the database and api contents. Meanwhile, the second image allowed login by fetching and matching the details provided with the contents stored in the api and mysql. While the third image was able to fetch the users online from the updated content and displayed onto the frontend. Since both the backend and frontend were containerized and deployed the image URL was used to view the Frontend UI as well. Thus, the simultaneous synchrononization and database population was possible by deploying the MySql database onto the cloud platform. From the above operations it is visible that the containerization and deployment of application on to the cloud allows the application to perform its functionality by integrating with other containers irrespective of the platform on which they are deployed on. Thus, containerization allows easy and efficient deployment.

**Part B – AWS LEX Chat Bot**

**Building a Chatbot:**

The Amazon provides AWS Lex services to build customized chatbots for our application. The following screenshots are the steps followed to build a chatbot for ordering pizza.

**Step 1**: Login into AWS Educate account and navigate to AWS console. Then select Amazon Lex from the services available.

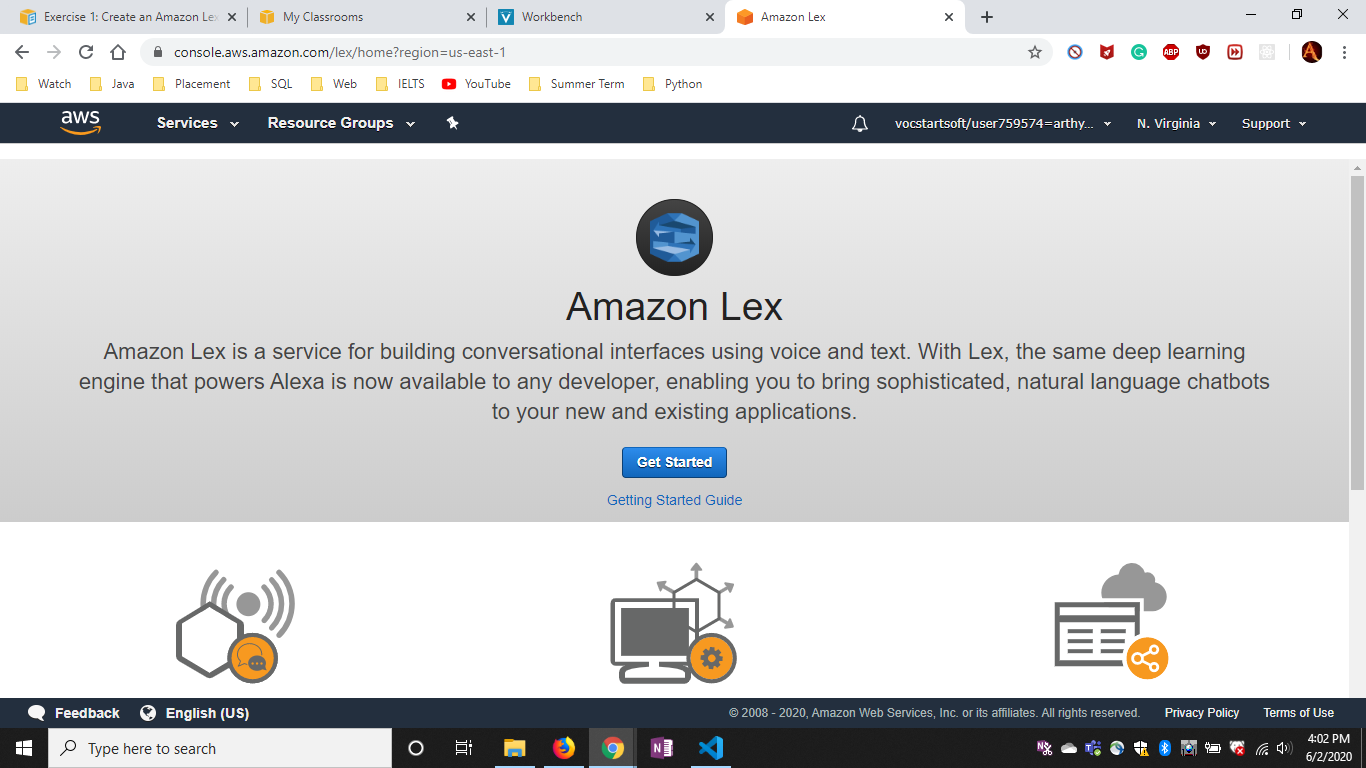


Figure -Lex service

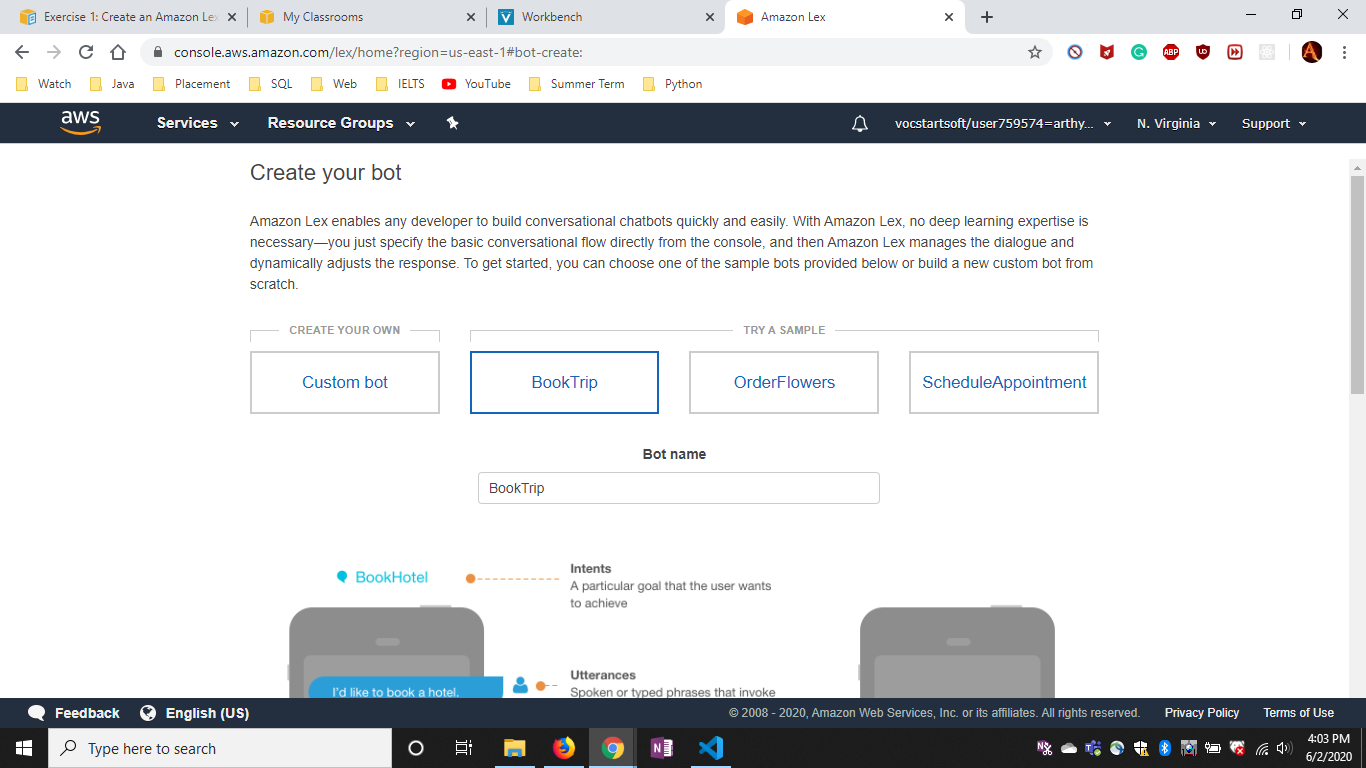
**Step 2:** Select create a new bot and click on custom bot tab.

Figure -Home page

**Step 3:** Provide the details of the new chat bot to be created.

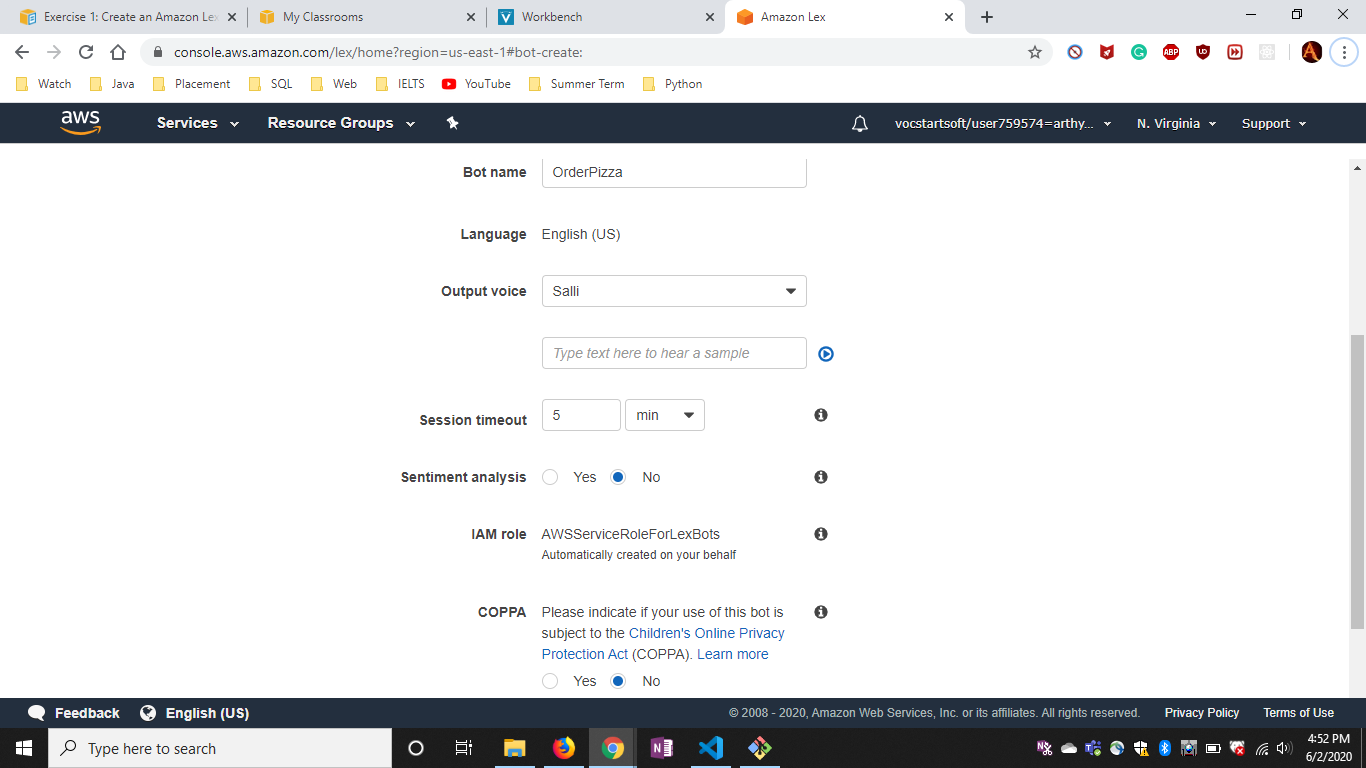


Figure -Create a new bot

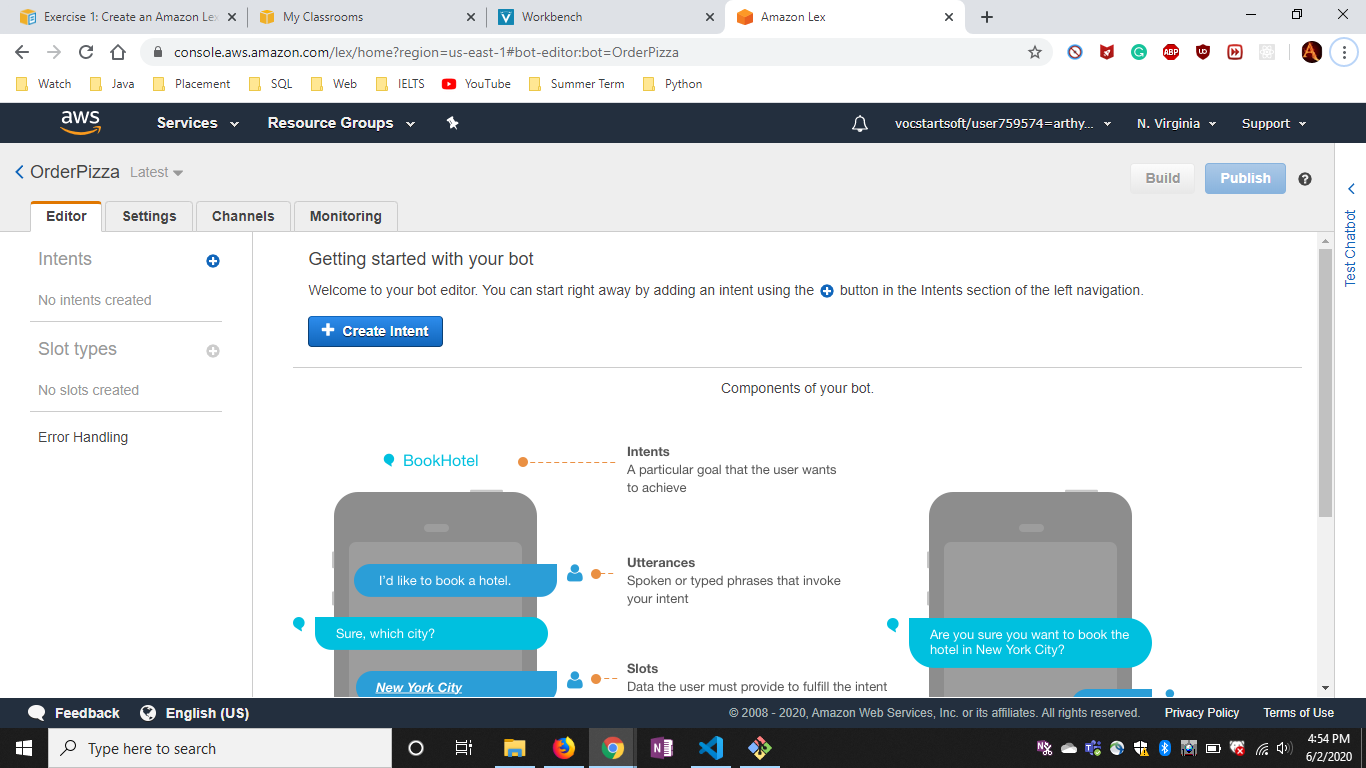
**Step 4:** In the created bot page we need to create new Intent.****

Figure -Chat Bot page

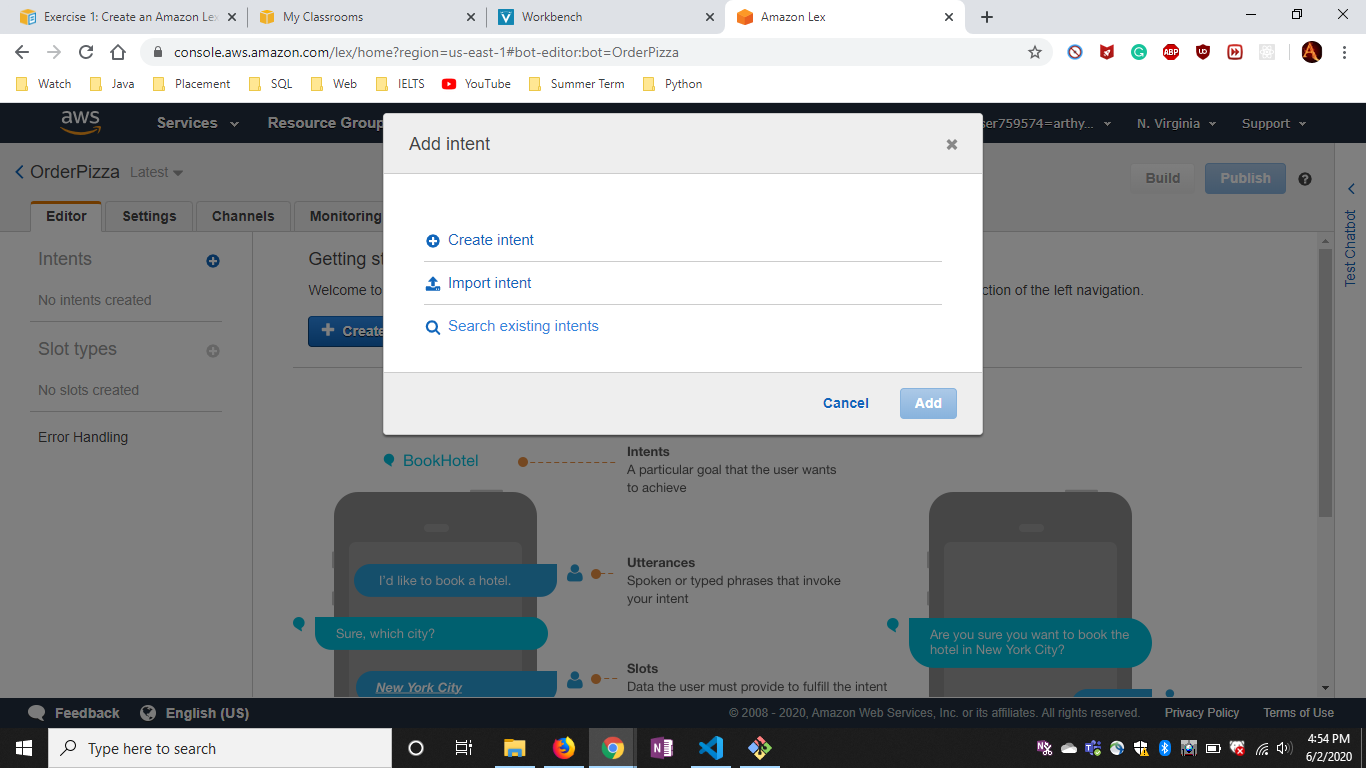
**Step 5:** Create a new Intent by providing the name of the intent.

Figure -Create new Intent

**Step 6:**

**Hello Intent :** The intent acts as an introduction to the customer which request further services. The intent is followed by corresponding utterances and slot type as required.

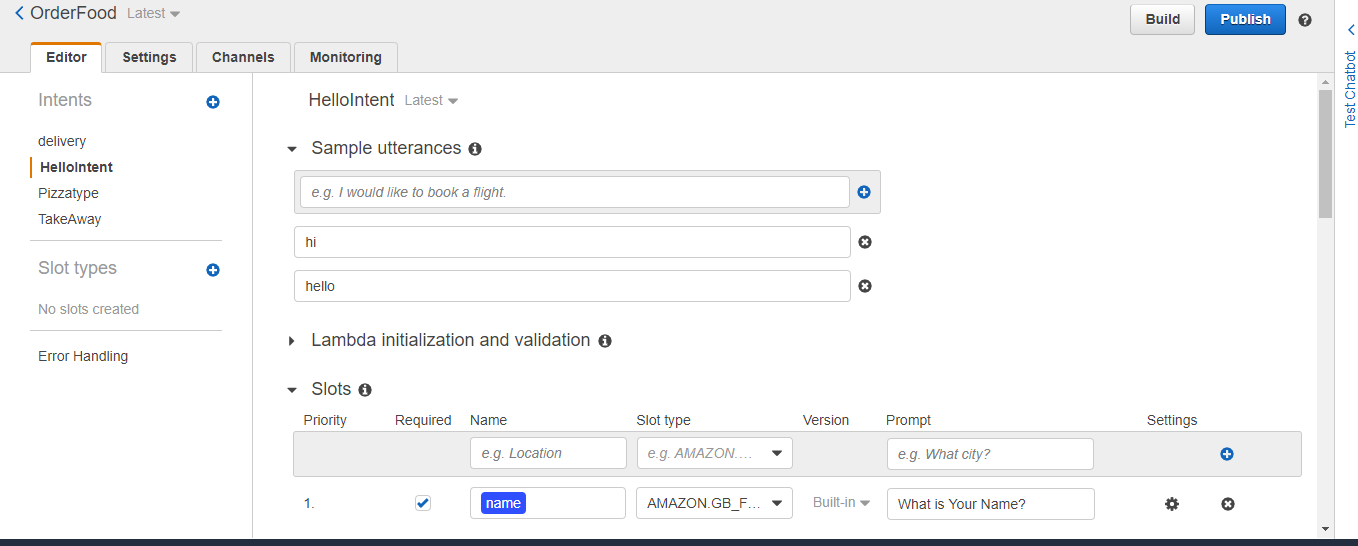


Figure -Sample Utterances

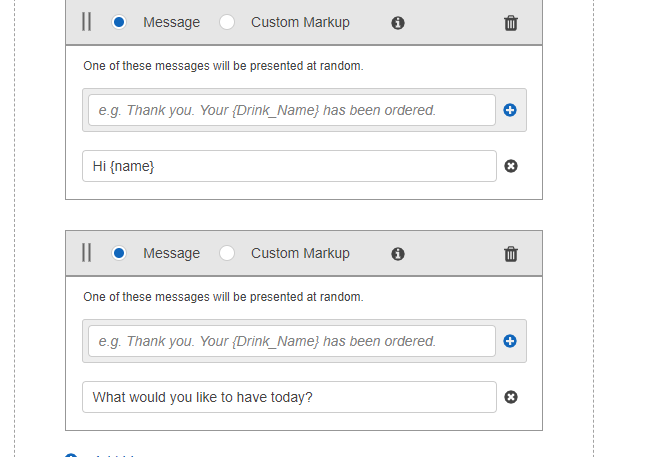
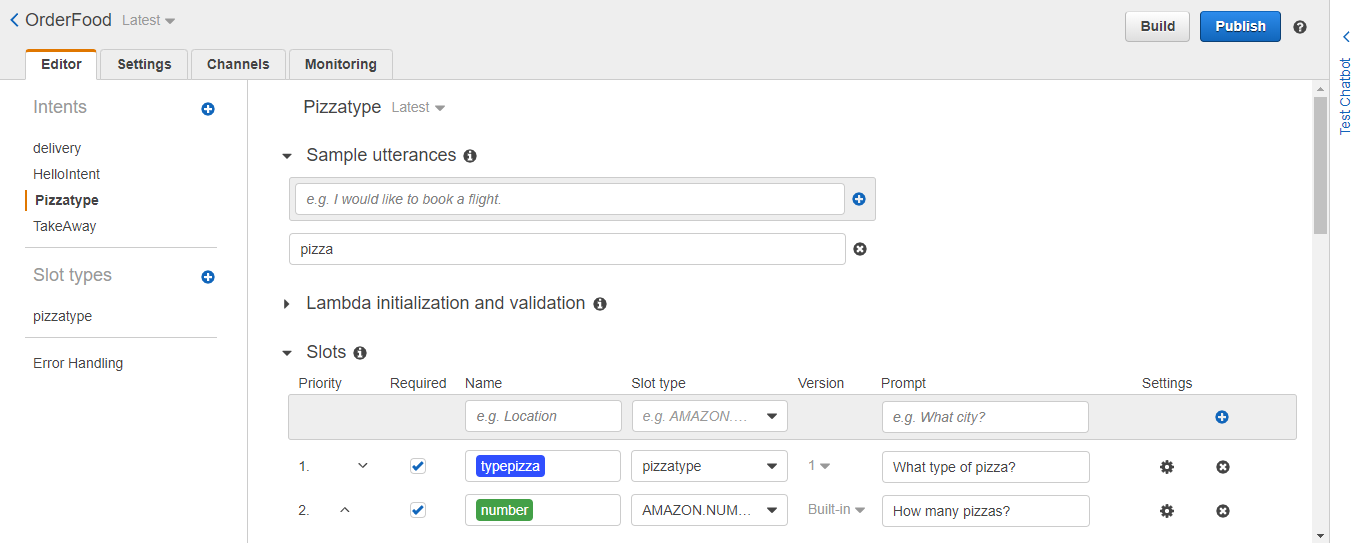


Figure -Response Message

Pizza intent

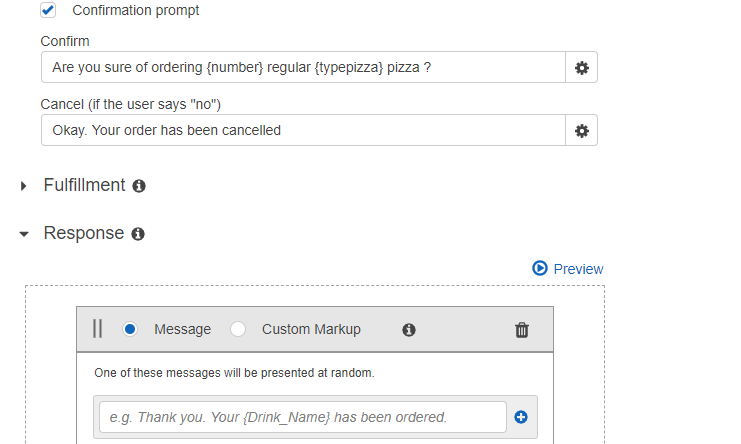
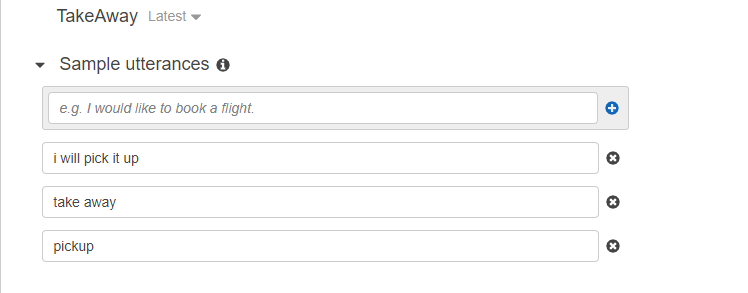
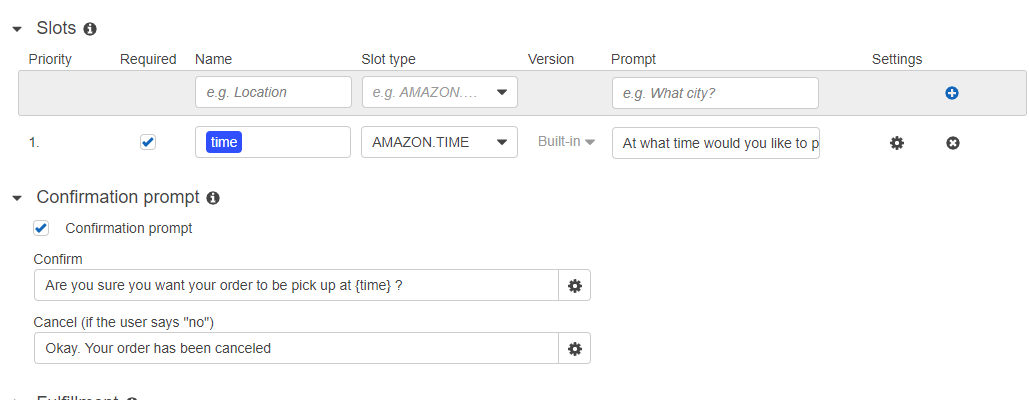
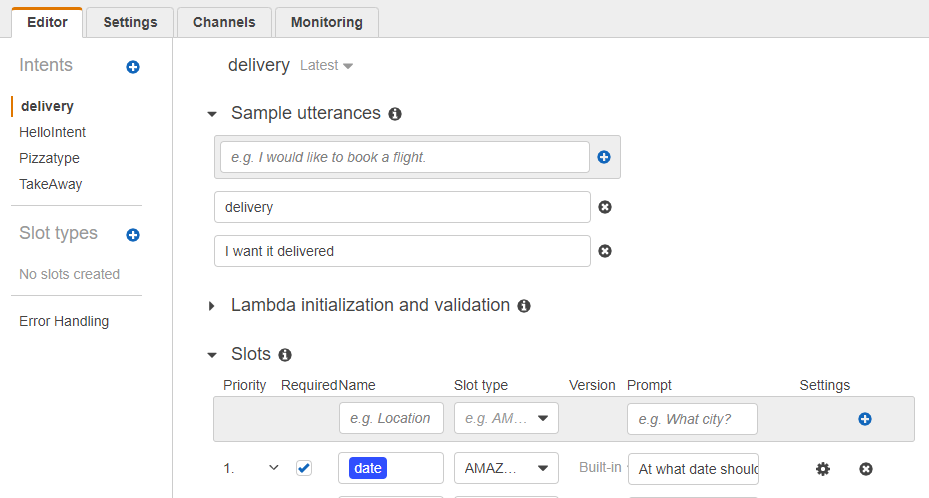


Figure -confirmation box

Take Away Intent



Delivery Intent



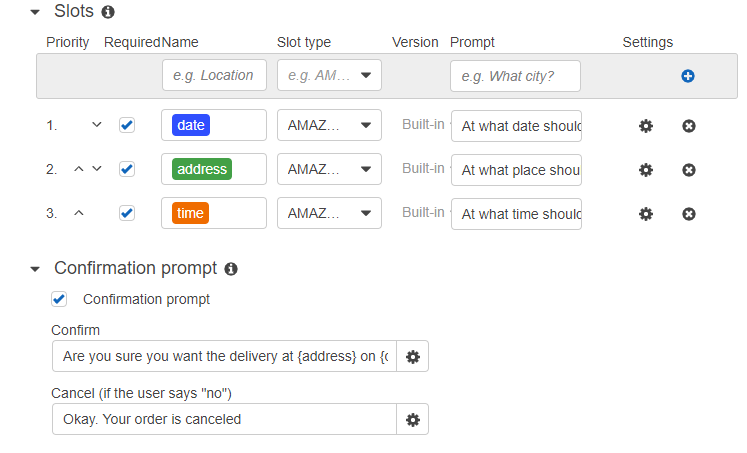
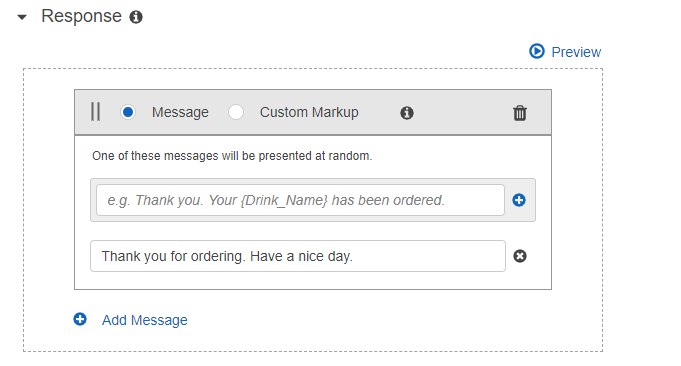


Figure -required slots added



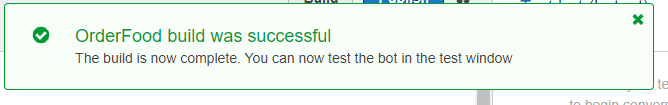


Figure -Success Message

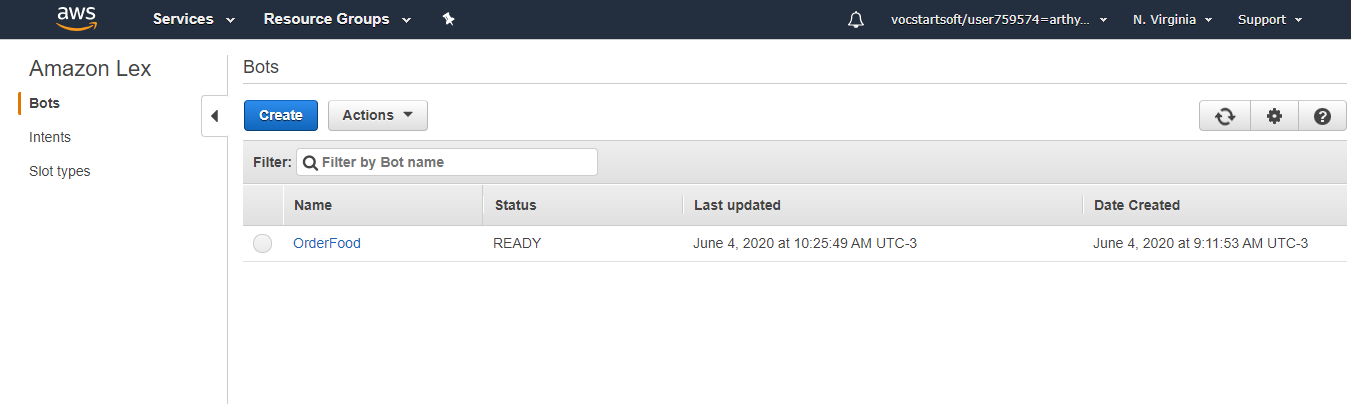


Figure -List of Bots

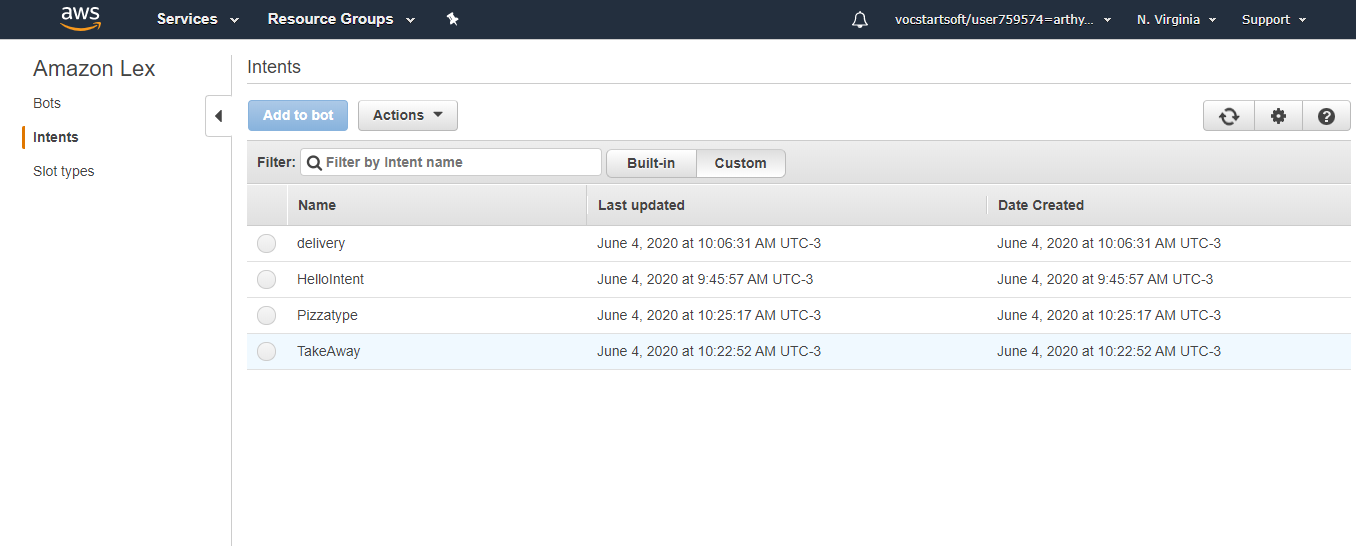


Figure -List of Intents

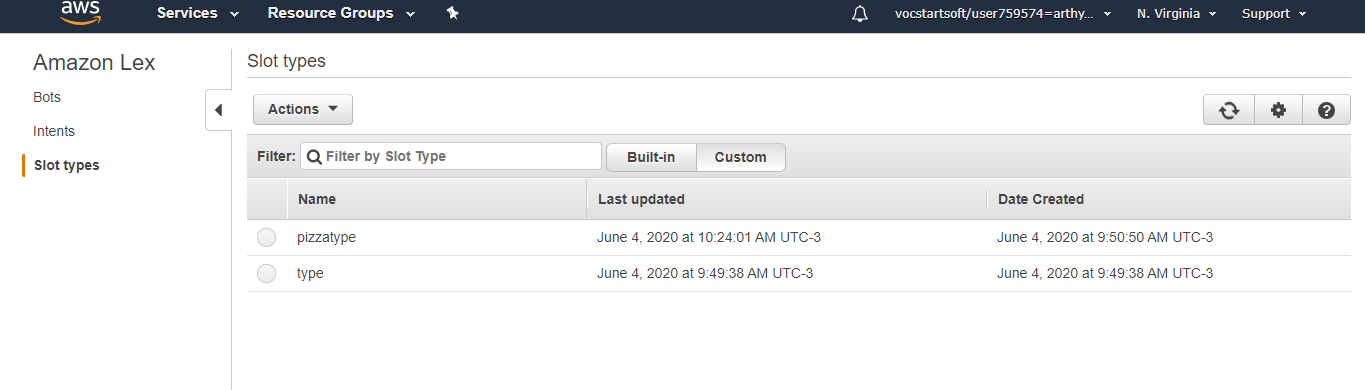


Figure -List of Slot Types

**Testing Chat Bot:**

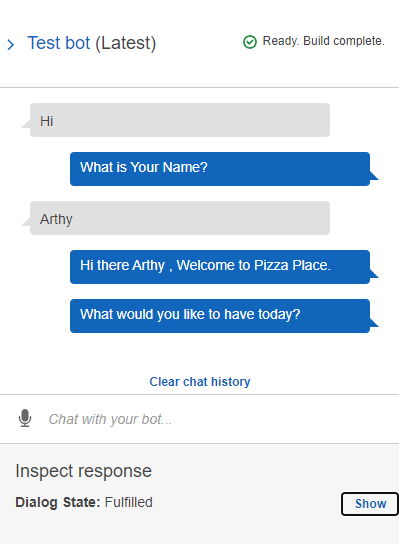


Figure -Chat1

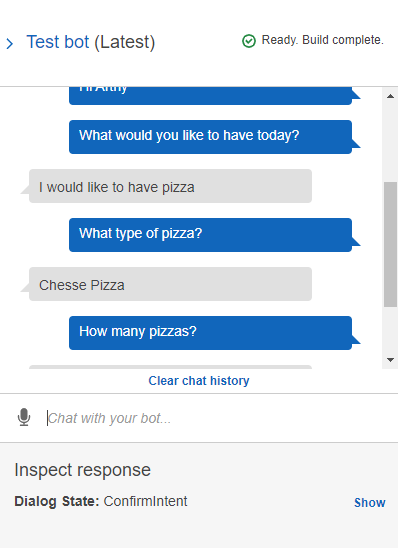


Figure - Chat 2

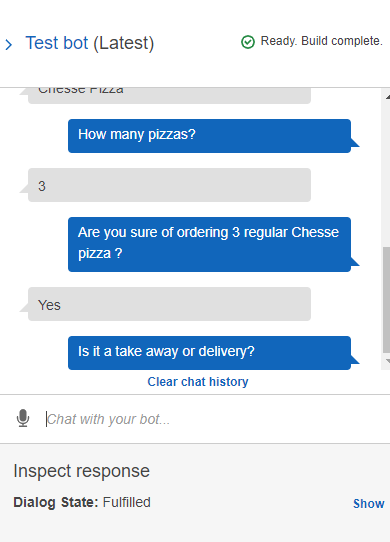


Figure -Chat 3

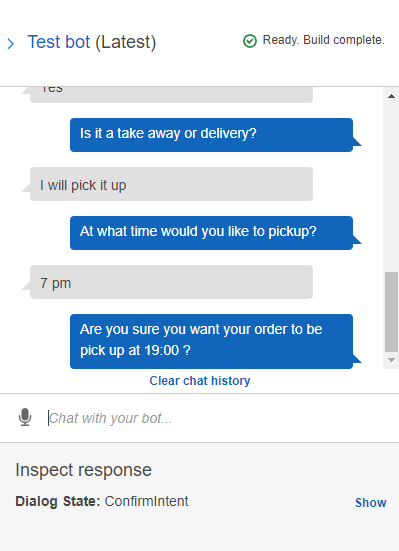


Figure -Chat 4

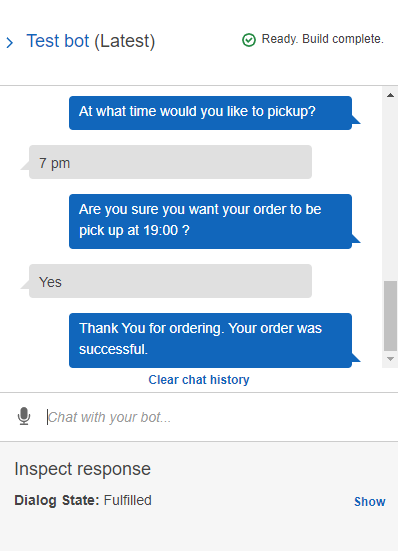


Figure -Chat 5

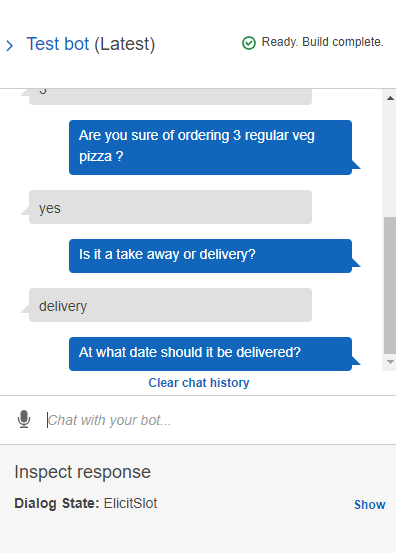


Figure -Chat 6

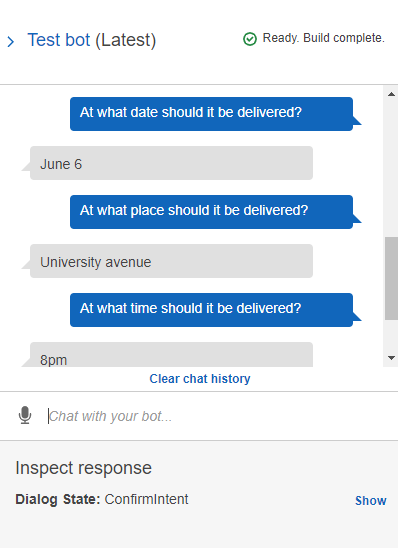


Figure -Chat 7

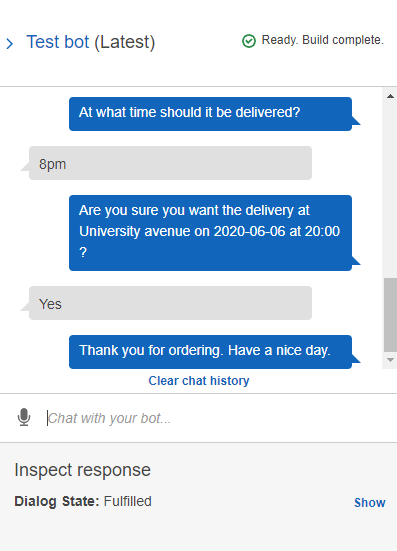


Figure -Chat 8

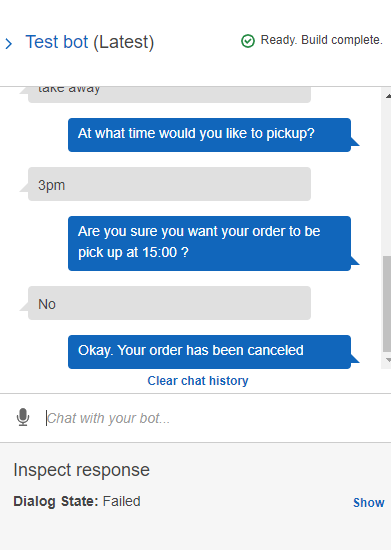


Figure -Chat 9

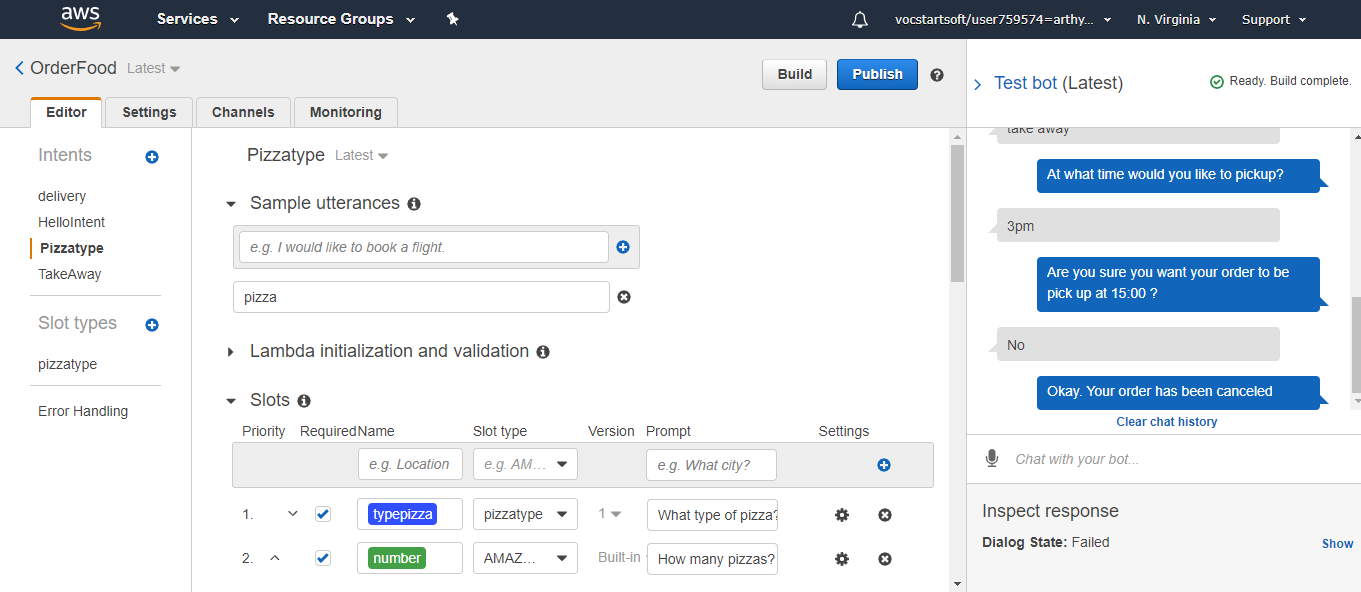


Figure -Overall picture of LEX

**Summary:**

The AWS Lex service allows us to create chat bot system which can be customized to our needs. The Lex service provides option of choosing ready made chat bots or to create our own customized chat bot. In order to create a new chat bot, we need to provide the intents for the chat bot. The chat bot simply matches the utterances provided with the intent and prompts based on that intent. The utterances may be spoken or typed phrases.

The slot provided must be in such a way to fulfil the intent. The slot type may be inbuilt or can also be created by providing a set of values to the slot type.

The different types of slots the user might provide must be predetermined and added to the slot corresponding utterances. This allows the bot to capture the slot type provided in curly brackets {} and use it for fulfilling the intent. The chat bot then provides a response to the user as specified in the response tab. The slots provided can be required or unrequired and it can also be given a certain order to be prompted to the user.

The response provided by the chat bot can be in such a so that the next utterance provided by the user can be matched with another intent. Any number of intents can be added and matched on the user utterances provided.

The chat bot initially starts by matching the utterances typed by the user with the built intents and responds by requesting slots in the order provided. Once the slots have fulfilled the intent, a confirmation messages can be displayed and if the user responds with acceptance then it is followed by the response else if the user rejects the confirmation the bot simply reverts back the request.

Thus, the LEX provides cost less, efficient, customer satisfactory chat bot service.

# References

# Docker Image vs Container: Everything You Need to Know. (2020). Retrieved 9 June 2020, from <https://stackify.com/docker-image-vs-container-everything-you-need-to-know/>

Get started using Docker containers with Node.js. (2020). Retrieved 9 June 2020, from <https://docs.microsoft.com/en-us/windows/nodejs/containers>

Pushing and Pulling to and from Docker Hub. (2020). Retrieved 8 June 2020, from <https://ropenscilabs.github.io/r-docker-tutorial/04-Dockerhub.html>

Creating A Chatbot With Amazon Lex And AWS Lambda. (2020). Retrieved 7 June 2020, from <https://medium.com/@CobusGreyling/creating-a-chatbot-with-amazon-lex-and-aws-lambda-a72eb2e08598>