









# Documentation

## Part A. Build, deploy, and run a Containerized Application using GCP.

a. Create three containers using Docker. These containers are responsible for the backend logic. The database you will be using here is, MySQL

### 1. 3 Container directories

	.git	10-06-2020 22:05	File folder	
	.idea	10-06-2020 21:55	File folder	
	__pycache__	10-06-2020 00:53	File folder	
	Container1	10-06-2020 22:01	File folder	
	Container2	10-06-2020 22:02	File folder	
	Container3	10-06-2020 22:03	File folder	
	db script	09-06-2020 15:33	File folder	
	README.md	06-06-2020 22:54	MD File	1 KB

### 2. Google SQL database

Filter tree							?	III
<input type="checkbox"/>	Instance ID ? ↑	Type	Public IP address	Private IP address	Instance connection name	High		
<input type="checkbox"/>	✓ serverless-a2	MySQL 5.7	35.236.239.82		axial-mercury-27980...	AD	▼	⋮

**b.** Container 1 is responsible for accepting registration details from frontend and store it in backend database. (image 1)

1. User registration is done in the 1<sup>st</sup> container (url is shown in url bar). The first container contains 2 routes ('/' and 'register'). '/register' route works on pressing Register button.

The screenshot shows a web browser with the URL `https://register-pnh7joctka-uk.a.run.app`. The page title is "Assignment 2 (Serverless Data Processing)". Below the title, there are three distinct form boxes:

- Login:** Contains fields for "Username" and "Password", and a "Login" button.
- Register:** Contains fields for "Username" (filled with "newuser2"), "Email" (filled with "newuser@gmail.com"), "Password" (filled with "\*\*\*\*"), a "Select topic" dropdown menu (showing "2nd Option"), and a "Register" button.
- Login status:** Displays "Hi," followed by a blank input field, then "you are logged in", a "Logout" button, and a link that says "Here are other users who are online".

2. Registration complete

The screenshot shows the "Register" form with the following fields and values:

- Username: [empty]
- Email: [empty]
- Password: [empty]
- Select topic: 1st Option (dropdown menu)
- Register: [button]

Below the form, the text "Registration successfull" is displayed.

c. Container 2 is responsible for validating the Login information (image 2)

1. Login of user works on 2<sup>nd</sup> container. As the user has pressed the Register button, the URL bar shows the '/register' route and 1<sup>st</sup> container link. On pressing the login button, the system will fetch '/login' route present in the 2<sup>nd</sup> container.

The screenshot shows a web browser window with the address bar displaying `https://register-pnh7jockta-uk.a.run.app/register`. Below the browser window, the page is titled "Assignment 2 (Serverless Data Processing)". The page content is divided into three distinct panels:

- Login Panel:** Contains a "Login" heading, a "Username" input field with the value "newuser2", a "Password" input field with masked characters "....", and a "Login" button.
- Register Panel:** Contains a "Register" heading, "Username", "Email", and "Password" input fields, a "Select topic" dropdown menu with "1st Option" selected, a "Register" button, and a message "Registration successfull".
- Login status Panel:** Contains a "Login status" heading, a "Hi," greeting, a "Logout" button, and a section titled "Here are other users who are online".

2. Login complete

This screenshot shows the "Login" panel after a successful login. The "Username" and "Password" input fields are now empty. The "Login" button remains visible. Below the button, the text "User created" is displayed.

The values in the input are erased once the user presses the login button, hence they are not visible in the above image.

d. Once a user is logged in – the state changes to online, and it appears on the front page (image 3)

1. After the user logs in, the Login status box changes and the username appears. Users can log out using the Logout button and the box also shows other users currently online in the system. It is programmed as such that the current user does not have the authority to logout other users (which should be the obvious case).

The screenshot shows a web browser window with the address bar displaying `https://login-pnh7jockta-uk.a.run.app/login`. The page title is "Assignment 2 (Serverless Data Processing)". Below the title, there are three distinct form boxes:

- Login Form:** Contains fields for "Username" and "Password", a "Login" button, and a message "User created" below the button.
- Register Form:** Contains fields for "Username", "Email", and "Password", a "Select topic" dropdown menu (currently showing "1st Option"), and a "Register" button.
- Login status Form:** Displays "Hi," followed by a box containing "newuser2", the text "you are logged in", a "Logout" button, and a section "Here are other users who are online" with a box containing "harsh".

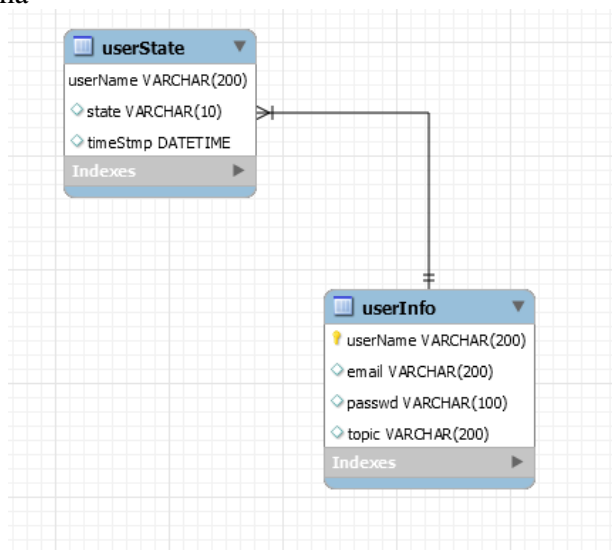
2. After the user clicks on Logout, the current user logs out and other users who are online in the system are still shown.

This screenshot shows the "Login status" form after a user has logged out. The form contains:

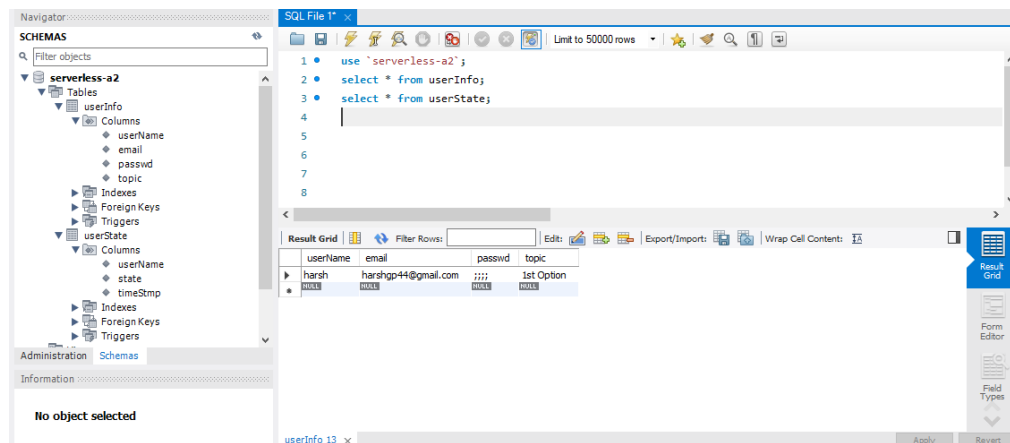
- The heading "Login status".
- A greeting "Hi," followed by an empty box.
- The text "you are logged in".
- A "Logout" button.
- A section "Here are other users who are online" followed by a box containing the name "harsh".

e. Your database should contain only 2 tables. One to contain data, another to contain user state (online, offline, timestamp, etc.) information.

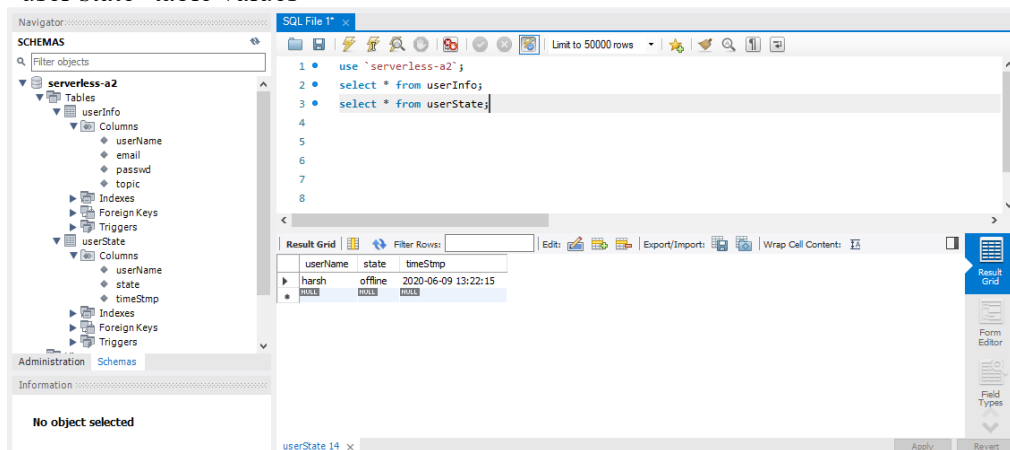
### 1. Database Schema



### 2. 'userInfo' table values



### 3. 'user state' table values



All these screenshots are provided in the 'Screenshots' folder to view properly.

f. Container 3 is responsible for extracting state information from the database. E.g. who is online. You need to maintain the session from login to logout. The session must expire after clicking the logout, which should update the state table.

1. As shown in this below image, as the user gets created and the message appears on the Login box, ('User-created'), The Login status box shows the value of the user currently logged in which signifies the session. Moreover, when the user clicks on the Logout button, the user gets removed from 'online' users which are shown in the above images. The other online users are also shown in the Login status box.

The screenshot shows a web browser window with the URL <https://login-pnh7joctka-uk.a.run.app/login>. Below the browser window, the title "Assignment 2 (Serverless Data Processing)" is centered. The main content area displays three side-by-side panels:

- Login Panel:** Contains a "Username" input field, a "Password" input field, a "Login" button, and a message "User created" below the button.
- Register Panel:** Contains a "Username" input field, an "Email" input field, a "Password" input field, a "Select topic" dropdown menu with "1st Option" selected, and a "Register" button.
- Login status Panel:** Displays "Hi, newuser2" in a box, followed by "you are logged in", a "Logout" button, and a section titled "Here are other users who are online" with a box containing the name "harsh".

g. Once the docker images are built, you need to run those using Google Cloud Run.

1. Build and Push for container1 is shown in the below image. Build and Push screenshots for other containers are available in the Screenshots folder.

```
Removing intermediate container 60d4ff682e3b
--> f9915e816ae0
Step 5/7 : EXPOSE 5000
--> Running in 42572f1463e0
Removing intermediate container 42572f1463e0
--> cd001e5f7639
Step 6/7 : ENV NAME register
--> Running in 3c2293e4cf40
Removing intermediate container 3c2293e4cf40
--> c4142b3e57e2
Step 7/7 : CMD ["python","src/Controller.py"]
--> Running in 7a206e97a297
Removing intermediate container 7a206e97a297
--> a6b62d6c3763
Successfully built a6b62d6c3763
Successfully tagged gcr.io/axial-mercury-279803/register:latest
harshgp44@cloudshell:~/Serverless-AS-2/Container1 (axial-mercury-279803)$ docker push gcr.io/axial-mercury-279803/register:latest
The push refers to repository [gcr.io/axial-mercury-279803/register]
717066fd66c6: Pushed
cb82897a86f2: Pushed
b06e423c2cf9: Layer already exists
1cc2f45986fd: Layer already exists
de4f53fbd5e1: Layer already exists
311f330fa783: Layer already exists
f55f65539fab: Layer already exists
30339f20ced0: Layer already exists
0eb22bfb707d: Layer already exists
a2ae92ffcd29: Layer already exists
latest: digest: sha256:c855310f6a134c7846ed690db8e3b20a20e531dd94e8973b8dca58d899b838b2 size: 2423
harshgp44@cloudshell:~/Serverless-AS-2/Container1 (axial-mercury-279803)$
```

## 2. Service Creation on Google Cloud Run

✓ Service settings

2 Configure the service's first revision

A service can have multiple revisions. The configurations of each revision are immutable.

Container image URL \*

gcr.io/axial-mercury-279803/register@sha256:06da01c495e5e23f: [SELECT](#)

E.g. gcr.io/cloudrun/hello  
Should listen for HTTP requests on \$PORT and not rely on local state. [How to build a container](#)

CONTAINER

VARIABLES

CONNECTIONS

Connect to other Google Cloud services like Google Cloud Storage or Google Cloud Firestore directly from your code. [Learn more](#)

Cloud SQL connections ⓘ

axial-mercury-279803:us-east4:serverless-a2

+ ADD CONNECTION

## 3. Google Cloud Run Service Activated for container1.

Cloud Run

Service details

EDIT AND DEPLOY NEW REVISION

register

Region: us-east4

URL: <https://register-pm7joc7kia-uk.a.run.app>

METRICS

REVISIONS

LOGS

DETAILS

YAML

PERMISSIONS

Revisions

MANAGE TRAFFIC

Filter revisions

Name	Traffic	Deployed	Actions
register-00001-deq	100%	11 minutes ago	

DETAILS

YAML

Container image URL

[gcr.io/axial-mercury-279803/register@sha256:c855310f6a134c7846ed690db8e3b20a20e531dd94e8973b8dca58d899b838b2](https://gcr.io/axial-mercury-279803/register@sha256:c855310f6a134c7846ed690db8e3b20a20e531dd94e8973b8dca58d899b838b2)

Container build

(no build information available)

Container source

(no source information available)

Container port

5000

Container command and args

(container entrypoint)

Auto-scaling

Up to 1,000 container instances

CPU allocated

1

Memory allocated

256Mi

Concurrency

80

Request timeout

300 seconds

Service account

[449794024176-compute@developer.gserviceaccount.com](mailto:449794024176-compute@developer.gserviceaccount.com)

VPC connector

None

Environment variables

Screenshots for other containers are available in Screenshots directory

h. To complete the tasks, and perform interaction, you need to build 3 simple web pages (or 1), using any technology of your choice.

1. I created a single page that performs all the required functionalities i.e. (Registration, Login, Login Status of the user).

The screenshot shows a web browser with the address bar displaying `https://register-pnh7jockta-uk.a.run.app`. The page title is "Assignment 2 (Serverless Data Processing)". Below the title, there are three distinct web pages displayed side-by-side:

- Login:** A form with fields for "Username" and "Password", and a "Login" button.
- Register:** A form with fields for "Username", "Email", "Password", and a "Select topic" dropdown menu (showing "1st Option"), and a "Register" button.
- Login status:** A page showing "Hi," followed by a line "you are logged in", a "Logout" button, and a section titled "Here are other users who are online".



- i. Write test case to test your application, and perform testing

After testing the application with all the test cases, I have mentioned their results as follows. There are 3 tables. The first one is for all the routes and services' testing, second is for Business logic function's testing and the last one is for DB interaction function's testing.

Routes	Expected output	Real output
/	Index.html	Index.html
/register	Index.html	Index.html
/login	Index.html	Index.html
/logout	Index.html	Index.html

Function Logic	Input	Output
getOnlineUserService()	No input	['hash','ABC','def']
registerService()	'registerUserName','registerEmail@email.com','registerPassword','Topic1'	True
loginService()	'userName', 'password'	True
logoutService()	'userName'	True

Function Logic	Input	Output
registerRepo()	'registerUserName','registerEmail@email.com','registerPassword','Topic1'	True
registerRepo()	'registerUserName','registerEmail@email.com','registerPassword','Topic1'	Registration Unsuccessfull
getOnlineUsersRepo()	No input	List of online users
activateState()	'username','online'	True
activateState()	'username','offline'	True
loginRepo()	'username','password'	'User logged in'
loginRepo()	'username','password'	'User already logged in'
logoutServiceRepo()	'username'	True

j. You need to explore, Google Cloud Run, GCR, Docker Container documents, and write a summary of ½ page explaining how you have used these technologies in your application.

First and foremost, I created a Google instance to explore docker and I tried creating docker containers using commands. After building and running the docker on google instance, I moved on to Google Cloud Run to do the same. I used Google cloud shell to build and push the docker. As this was the first time I was creating docker containers, I saw documentation for creating Dockerfile and created it. I cloned my repository from Github which contained Dockerfile and requirements.txt using which I finally built and pushed the docker container and gave it a tag. Then I created a service in Google Cloud Run and mentioned the port which I mentioned to expose in Dockerfile, and selected Google SQL database to connect with the container. Finally, my container service was ready which provided me a URL to run the application. There were some problems I faced while connecting to the Google SQL database with the docker. I resolved it by referring to the Google documentation and did some configuration with the database file and changed the SQL module in python. So I repeated this process for every container. And after I received the URL of the application, I changed my code routes accordingly which led me to again build and push the containers to update the application.

## Login Validation Code:

```
def loginRepo(self, userName, password):
    flag=False
    try:
        db = Database()
        db.connectDB()
        results=db.executeQueryWithResults('select * from userInfo')
        for i in results:
            if(i[0]==userName and Passwd().decrypt(i[2])==password):
                flag=True
        if(flag==True):
            if(self.activateState(userName,'online')):
                return "User logged in"
            else:
                return "User already logged in"
        return "Invalid credentials"
    except Exception as e:
        print(e)
        return "User doesn't exist"
```

```
def activateState(self,username,state):
    try:
        db = Database()
        db.connectDB()
        db.executeQuery('delete from userState where userName="{0}"'.format(username))
        db = Database()
        db.connectDB()
        db.executeQuery('insert into userState values("{0}","{1}","{2}")'.format(username, state,
                                                                              datetime.now().strftime(
                                                                              '%Y-%m-%d %H:%M:%S')))

        return True
    except Exception as e:
        print(e)
        return False
```

## Basic Password Encryption

```
class Passwd:
    def __init__(self):
        pass

    def encrypt(self, text):
        str=""
        for i in text:
            str+=chr(ord(i)+10)
        return str

    def decrypt(self, text):
        str=""
        for i in text:
            str+=chr(ord(i)-10)
        return str
```

## Registration module

```
def registerRepo(self, registerUserName, registerEmail, registerPassword, registerTopic):
    encrypted_password=Passwd().encrypt(registerPassword)
    try:
        db = Database()
        db.connectDB()
        db.executeQuery('insert into userInfo values("{0}", "{1}", "{2}", "{3}")'.format(registerUserName,
registerEmail,
                                                                    encrypted_password, registerTopic))
        return True
    except Exception as e:
        print(e)
        return False
```

## Part B. Building a Chatbot:

### a. Using AWS Lex - Create a chatbot on OrderFood

b. Consider it as a pizza place. (assumptions: they have 3 types of regular size pizzas –veg, cheese, pepperoni.

c. The chatbot can accept information on food delivery or pickup

d. If it is delivery, then customer address, delivery date, and time is important

e. If it is takeaway, then assuming the same day, it should ask the arrival time of the customer.

### 1. Chatbot Creation page.

▼ Slots ⓘ

Priority	Required	Name	Slot type	Version	Prompt	Settings	
		<input type="text" value="e.g. Location"/>	<input type="text" value="e.g. AMAZON.US_..."/>		<input type="text" value="e.g. What city?"/>		
3.	▼	<input checked="" type="checkbox"/>	<input type="text" value="Pizza"/>	<input type="text" value="pizzas_type"/>	1 ▼	<input type="text" value="Hey, what kind of pizza do you want to"/>	
6.	^ ▼	<input checked="" type="checkbox"/>	<input type="text" value="Count"/>	<input type="text" value="AMAZON.NUMBER"/>	Built-in ▼	<input type="text" value="Also, how many pizza do you need?"/>	
7.	^ ▼	<input checked="" type="checkbox"/>	<input type="text" value="Time"/>	<input type="text" value="AMAZON.TIME"/>	Built-in ▼	<input type="text" value="When will you arrive to pickup the ord"/>	
8.	^ ▼	<input checked="" type="checkbox"/>	<input type="text" value="Date"/>	<input type="text" value="AMAZON.DATE"/>	Built-in ▼	<input type="text" value="At what Date?"/>	
9.	^	<input checked="" type="checkbox"/>	<input type="text" value="Address"/>	<input type="text" value="AMAZON.US_CITY"/>	Built-in ▼	<input type="text" value="Please provide your address"/>	

▼ Confirmation prompt ⓘ

☒ Confirmation prompt

Confirm

Cancel (if the user says "no")

## 2. Full Chatflow for Pizza Delievery

Deliver pizza at my place

Hey, what kind of pizza do you want to get delivered?

Cheese

Clear chat history

Cheese

Also, how many pizza do you need?

12

When will you arrive to pickup the order?

Clear chat history

When will you arrive to pickup the order?

Around 9 in the morning

At what Date?

tomorrow

Clear chat history

tomorrow

Please provide your address

New York

Clear chat history

Please provide your address

New York

Your order has been placed. It will get delivered to your place by 09:00

[Clear chat history](#)

---

### 3. Full chat flow for pizza pickup

want to pickup some pizzas

Hey, what kind of pizza do you want to pickup?

Cheese

Clear chat history

Cheese

When will you arrive to pickup the order?

Maybe 9 in the morning

Also, how many pizza do you need?

Clear chat history

Also, how many pizza do you need?

3

Your order has been placed. You can come and pick it up at 09:00

Clear chat history

### 4. Utterances for Pickup

#### ▼ Sample utterances ⓘ

e.g. I would like to book a flight.



Hey, I want to pickup some pizzas





## 5. Delivery utterances

### ▼ Sample utterances ⓘ

+

Deliver pizza to my place



order me a pizza for delivery



## References

- [1]"Cloud Computing Services | Google Cloud", *Google Cloud*, 2020. [Online]. Available: <https://cloud.google.com/>. [Accessed: 9- Jun- 2020].
- [2]"Docker Documentation", *Docker Documentation*, 2020. [Online]. Available: <https://docs.docker.com/>. [Accessed: 9- Jun- 2020].
- [3]"Set up your first microservice using Flask and Docker: Part 2", Medium, 2020. [Online]. Available: <https://medium.com/hacksnextdoor/set-up-your-first-microservice-using-flask-and-docker-part-2-d8e078357500>. [Accessed: 9- Jun- 2020].
- [4]2020. [Online]. Available: <https://www.youtube.com/watch?v=KTa1T14nkbw>. [Accessed: 9- Jun- 2020].