

BUILD, DEPLOY AND RUN A CONTAINERIZED APPLICATION USING GCP

CSCI 5410 – Serverless Data Processing
Assignment 2 –Part A

Dhrumil Amish Shah (B00857606)
dh416386@dal.ca

Project Git Repository

https://git.cs.dal.ca/dashah/csci-5410-f2021-b00857606-dhrumil-amish-shah-/tree/main/assignment_2_code

Firebase Firestore Setup

Figures 1, 2, 3, 4, 5, and 6 display the screenshots of the configuration setup of the Firebase Firestore database. Initially, there is no data in the Firestore database.

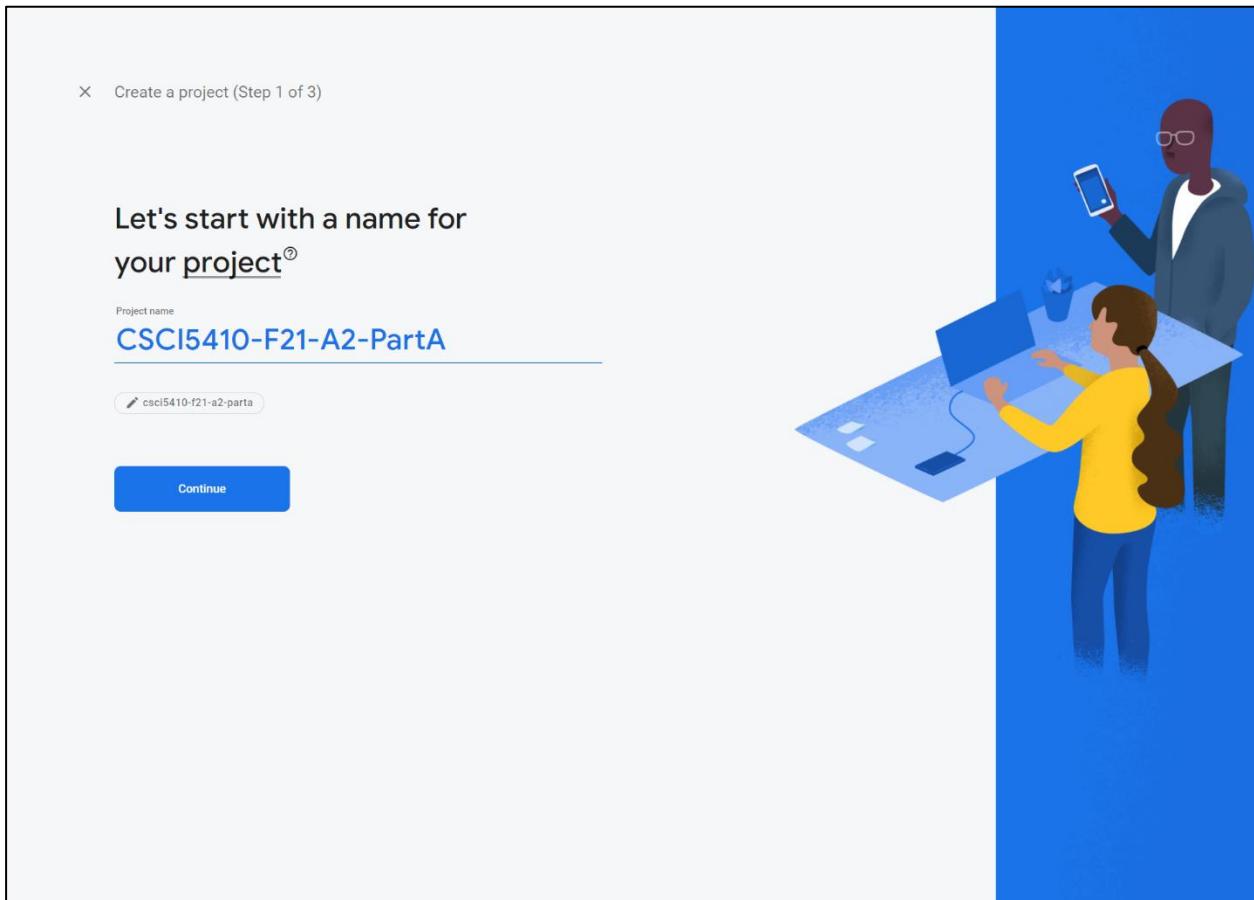


Figure 1 - Firebase Firestore project creation (Project name - CSCI5410-F21-A2-PartA) [1]

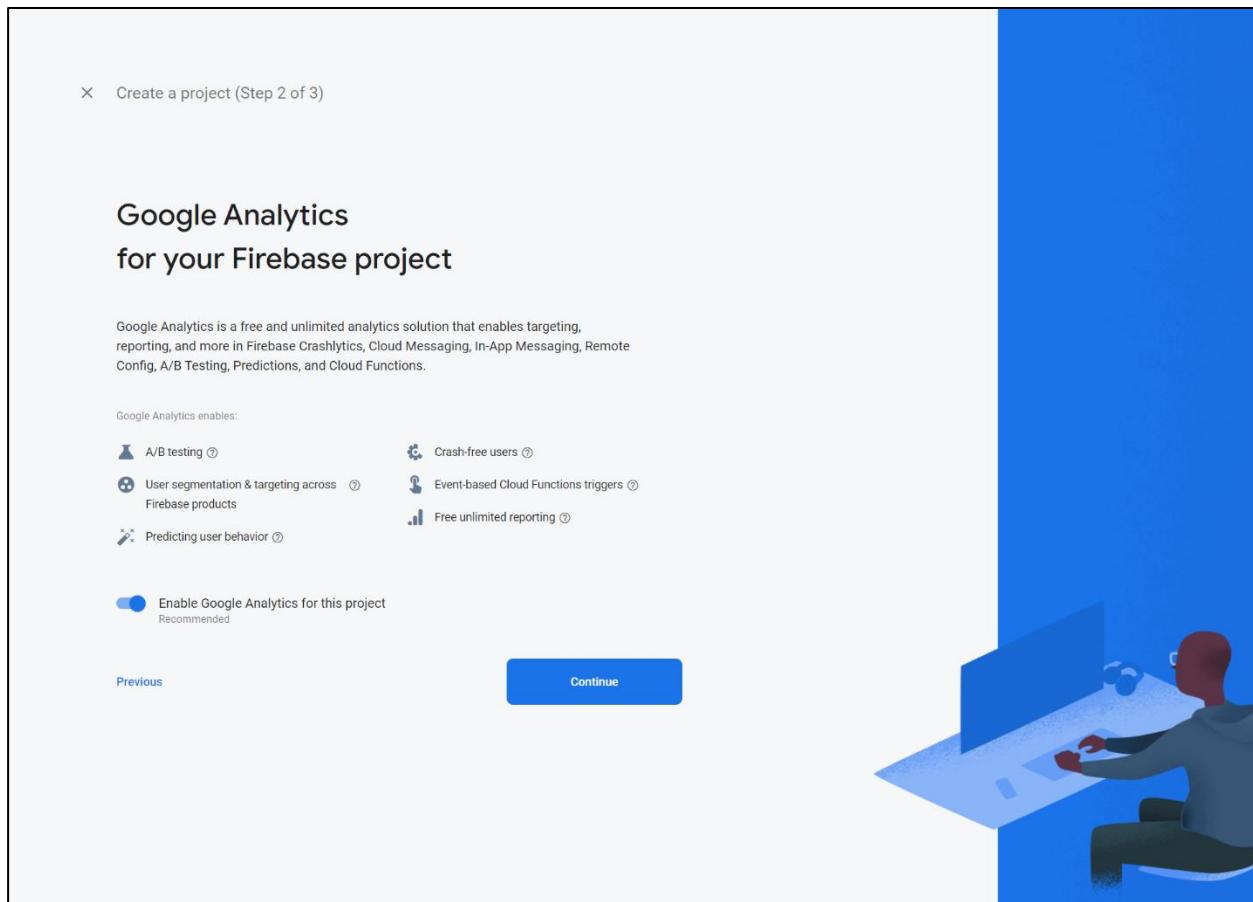


Figure 2 - Firebase Firestore project creation (Project name - CSCI5410-F21-A2-PartA) (contd.) [1]

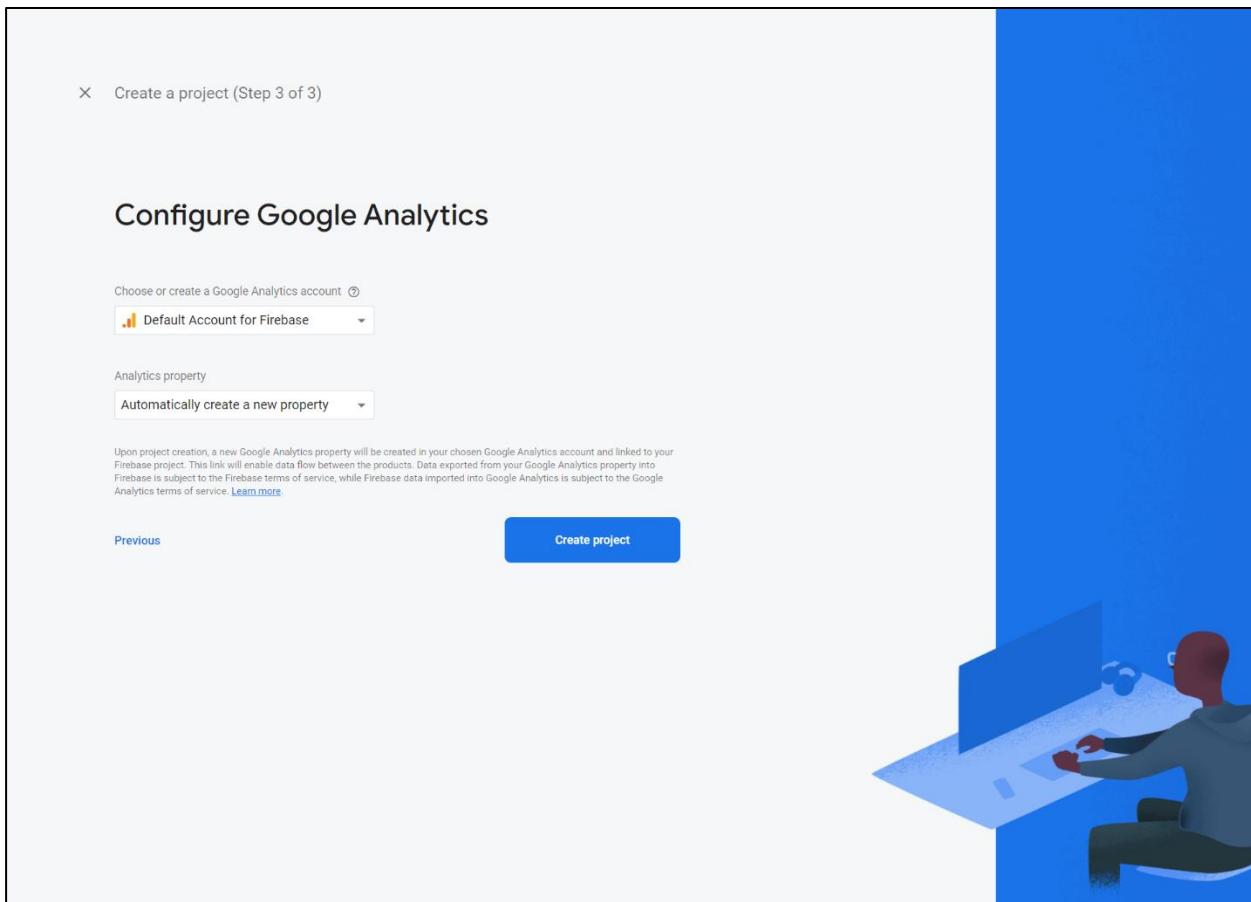


Figure 3 - Firebase Firestore project creation (Project name - CSCI5410-F21-A2-PartA) (contd.) [1]

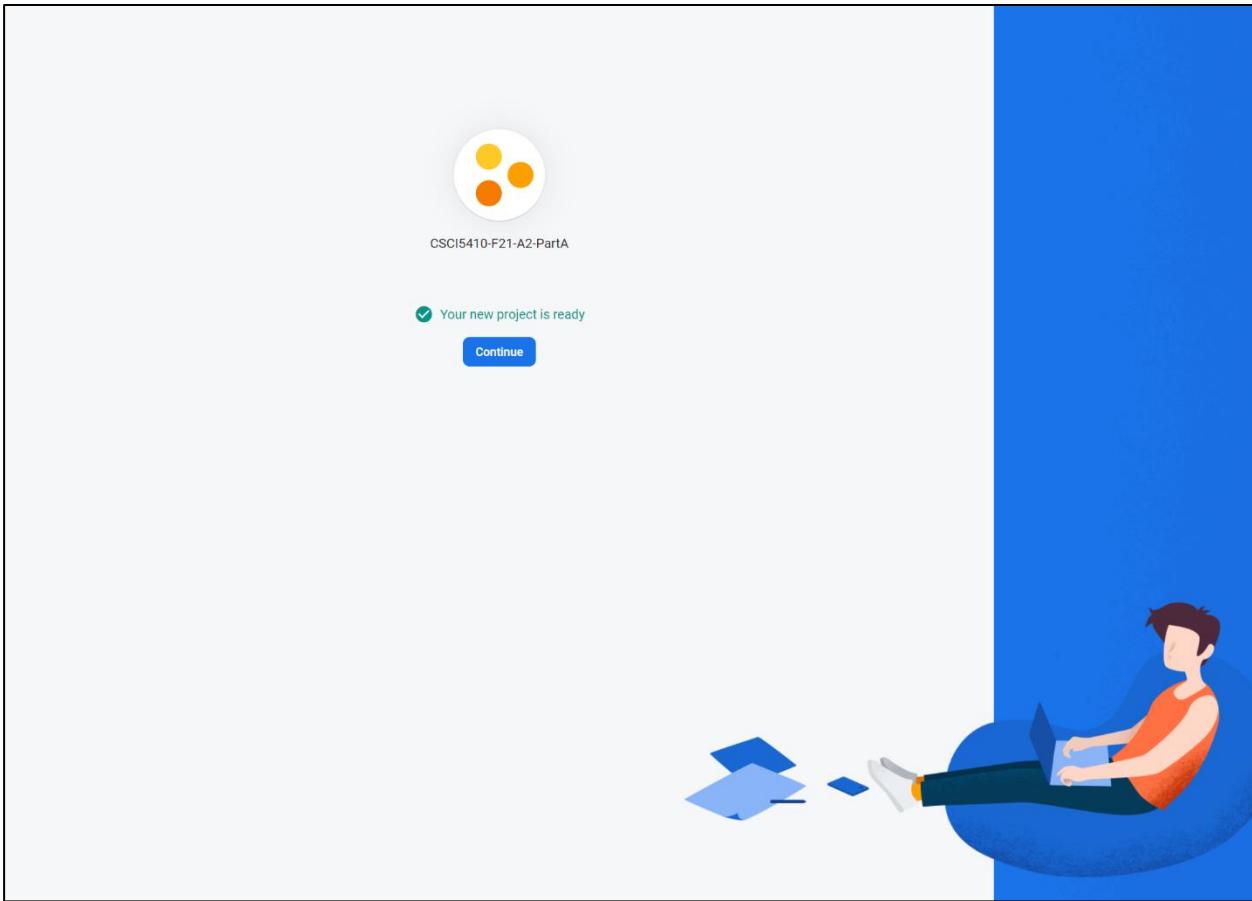


Figure 4 - Firebase Firestore project creation (Project name - CSCI5410-F21-A2-PartA) (contd.) [1]

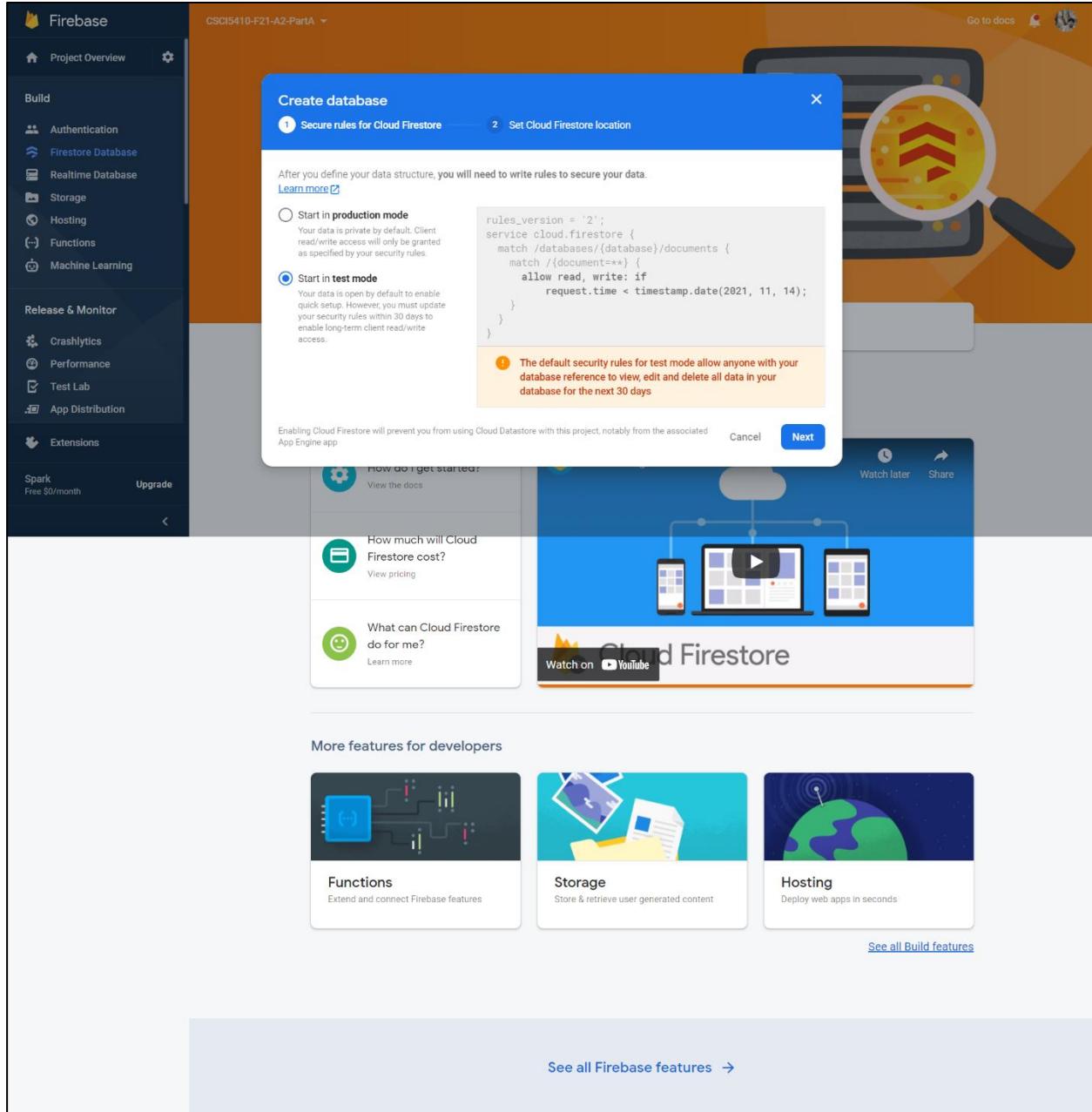


Figure 5 - Firebase Firestore project creation (Project name - CSCI5410-F21-A2-PartA) (contd.) [1]

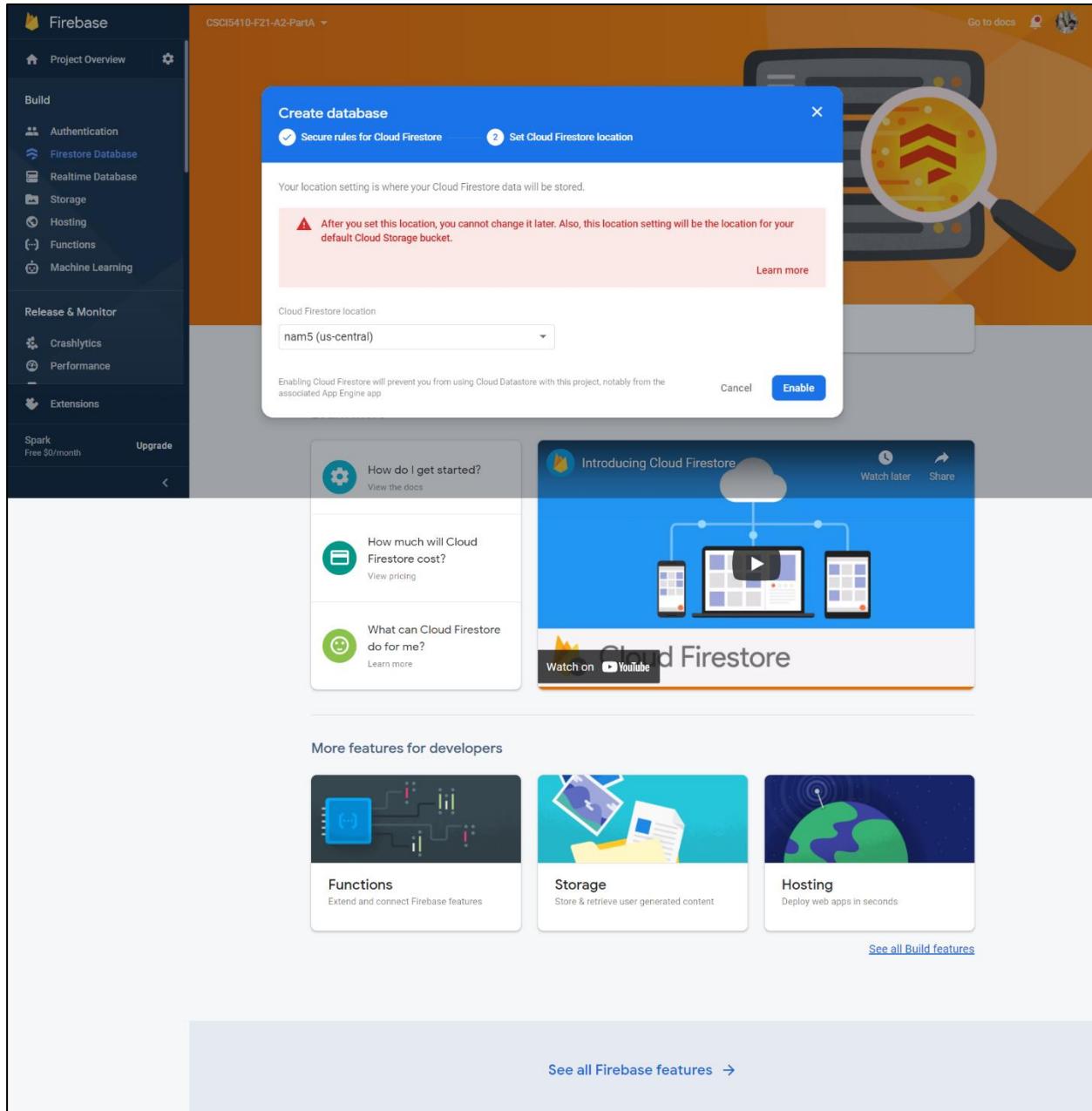


Figure 6 - Firebase Firestore project creation (Project name - CSCI5410-F21-A2-PartA) (contd.) [1]

Figure 7 displays the empty Firestore database (i.e., Database with no users registered).

The screenshot shows the Firebase Cloud Firestore interface for the project 'CSCI5410-F21-A2-PartA'. The left sidebar lists various services: Authentication, Firestore Database (selected), Realtime Database, Storage, Hosting, Functions, and Machine Learning. The main area is titled 'Cloud Firestore' and shows the 'Data' tab. It features a 'Get started' button with a note about the Local Emulator Suite. Below this is a section for collections, with a single entry 'csci5410-f21-a2-partA' and a 'Start collection' button. A large circular icon with server graphics is displayed, and the text 'Your database is ready to go. Just add data.' is shown. At the bottom, it says 'Cloud Firestore location: nam5 (us-central)'.

Figure 7 – Empty Firestore database (i.e., Database with no users) [1]

Container Images (Registration, Login, and Profile React applications)

Container 1 – Registration React Application

The **container_1_app** Registration React application is build using Docker. This application is responsible for accepting the registration information (i.e., first name, last name, email, and password) from a user. The accepted information is stored in the Firestore database. On successful registration, the user is redirected to the Login React application (i.e., **container_2_app** Login React application). Further, the status of the user registered is set to **offline**.

Figures 8 and 9 displays the Registration React application Docker image creation and successful creation of docker image in the dashboard. The name of the Docker image is **container_1_app**.

File Ed Selection View Go Run Terminal Help

all_applications - Visual Studio Code

EXPLORER TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

D:\Dalhouse\Total\Study_Material\Term_3\CSCI_5410_SDP\3_Assignments\Assignment_2\part_a\all_applications\container_1_app>docker build -t container_1_app .

[+] Building 154.5s (11/11) FINISHED

> [internal] load build definition from dockerfile 0.45s

>> [internal] load build context 0.05s

>> [internal] load .dockerignore 0.45s

>> [internal] transfer dockerfile 0.05s

>> [internal] load metadata for docker.io/library/node:alpine 0.05s

>> [internal] load 1 FROM docker.io/library/node:alpine@sha256:417b385ed2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 0.45s

>> => resolve docker.io/library/node:alpine@sha256:417b385ed2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 0.05s

>> => sha256:aaf2f67e7083c59ea4532fa1fa0ea0e25392396e681da72b712b37182ac 2.359s / 2.359s 0.15s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 0.85s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 0.05s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 1.439s / 1.439s 0.05s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 1.164s / 1.164s 0.05s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 6.539s / 6.539s 0.05s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 2.839s / 2.839s 1.06s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 34.81MB / 34.81MB 3.85s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 281.28MB / 281.28MB 0.95s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 2.744MB / 2.744MB 0.05s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 77400.509s / 77400.509s 1.15s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 2.0771828s 0.44s

>> => sha255:417b385d2e50d385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009 17902.441585e57814818637624f576271f136220f68211758f86 0.05s

> [internal] load build context 0.45s

>> [internal] transfer context: 721.03KB 0.35s

[2/6] WORKDIR /app 3.25s

[3/6] COPY package.json ./ 0.25s

[4/6] COPY package-lock.json ./ 0.25s

[5/6] RUN npm install 114.65s

[6/6] COPY . . 0.26s

>> exporting to image 19.85s

>> exporting layers 19.85s

>> writing image sha256:58995638083956e7150cdf4cefd11943ac9f34c3578687b5bd25dc51b2567e7 0.05s

>> naming to docker.io/library/container_1_app 0.05s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

D:\Dalhouse\Total\Study_Material\Term_3\CSCI_5410_SDP\3_Assignments\Assignment_2\part_a\all_applications\container_1_app>docker image ls

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
container_1_app	latest	589956380839	About a minute ago	568MB

D:\Dalhouse\Total\Study_Material\Term_3\CSCI_5410_SDP\3_Assignments\Assignment_2\part_a\all_applications\container_1_app>docker run -d -p 3000:3000 container_1_app cf499fbdc56eb2372dcf0add68ae87a66d2c308e2edde99ba5f311723928e

D:\Dalhouse\Total\Study_Material\Term_3\CSCI_5410_SDP\3_Assignments\Assignment_2\part_a\all_applications\container_1_app>docker container

Usage: docker container COMMAND

master* 0 0 0 0 Type here to search

14°C Mostly cloudy ENG 641 PM 10/15/2021

Figure 8 - Docker image creation of container_1_app Registration React application

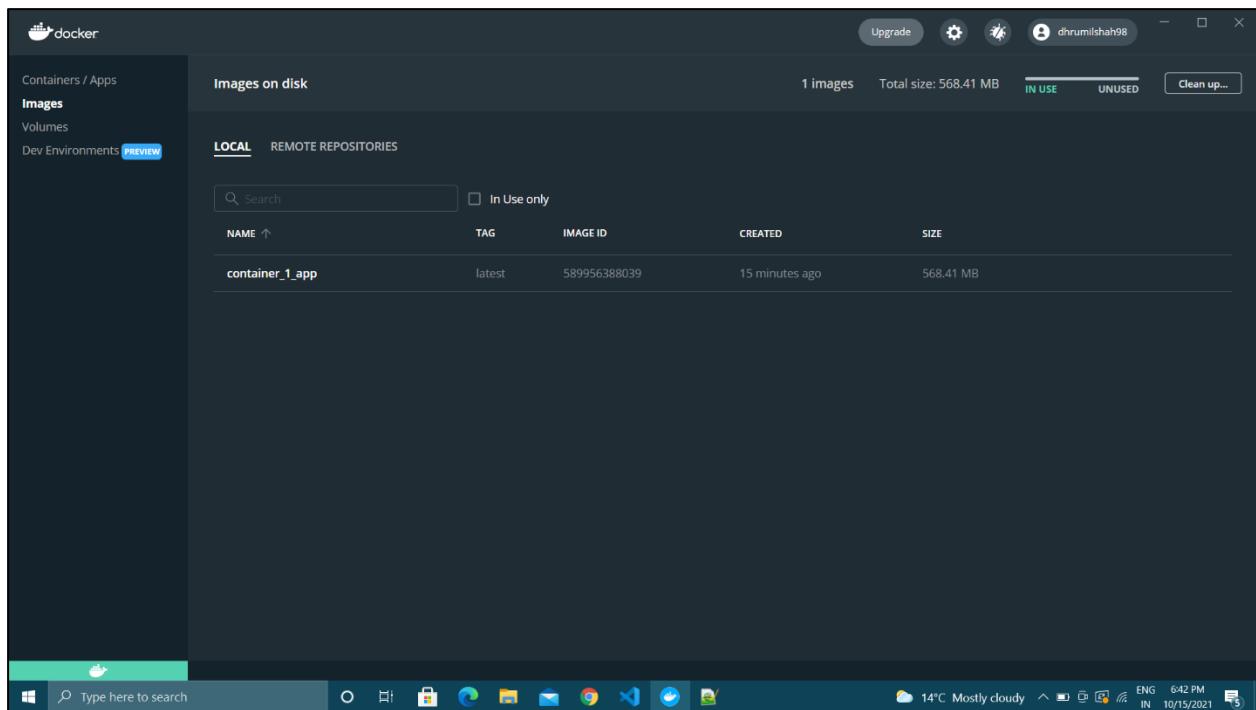


Figure 9 - Successful creation of Docker image of container_1_app Registration React application [2]

Container 2 – Login React Application

The **container_2_app** Login React application is build using Docker. This application is responsible for accepting the login information (i.e., email, and password), from a user and validate the information. The accepted information is verified in the Firestore database. On successful login, the user is redirected to the profile React application (i.e., **container_3_app** Profile React application). Further, the status of the logged-in user is set to **online**.

Figures 10 and 11 display the Login React application Docker image creation and successful creation of docker image in the dashboard. The name of the image is **container_2_app**.

The screenshot shows the Visual Studio Code interface with the Dockerfile open in the terminal tab. The terminal output displays the build process for the Dockerfile located at D:\DalhousieTotal\Study_Material\Term_3\CSCI_5410_SDP\3_Assignments\Assignment_2\part_a\all_applications\container_2_app\dockerfile. The build command used was `docker build -t container_2_app .`. The output shows the progress of the build, including transferring files, loading metadata, and running npm install. The final command shown is `docker image COMMAND`. Below the terminal, a list of Docker commands is provided:

```

Commands:
build      Build an image from a Dockerfile
history    Show the history of an image
import     Import the contents from a tarball to create a filesystem image
inspect   Display detailed information on one or more images
load       Load an image from a tar archive or STDIN
ls         List images
prune     Remove unused images
pull      Pull an image or a repository from a registry
push      Push an image or a repository to a registry
rm        Remove one or more Images
save      Save one or more Images to a tar archive (streamed to STDOUT by default)
tag       Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

```

At the bottom of the terminal, it says "Run 'docker image COMMAND --help' for more information on a command."

Figure 10 - Docker image creation of container_2_app Login React application

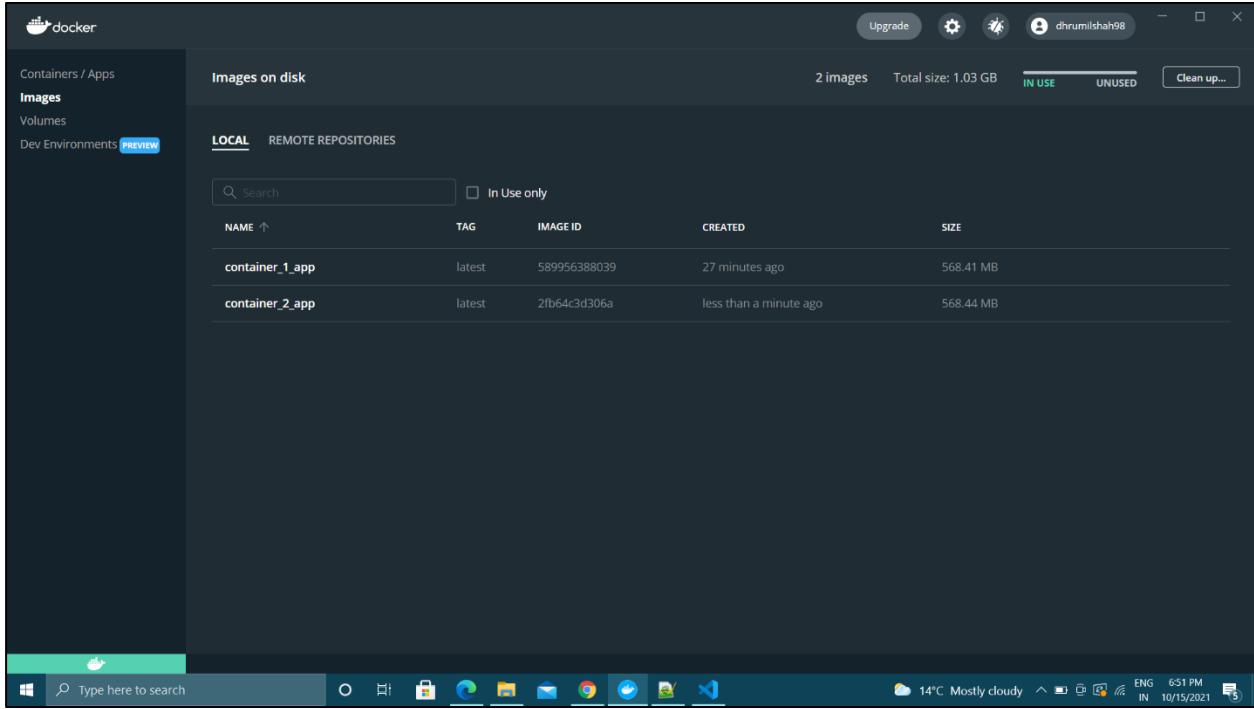


Figure 11 - Successful creation of Docker image of container_2_app Login React application [2]

Container 3 – Profile React Application

The **container_3_app** Profile React application is build using Docker. This application is responsible for displaying the profile information (i.e., first name, last name and email) of the logged-in user as well as the first name and last name of all the other online users. Also, there is a logout button that logs out the user and updates the status to **offline**. Further, on logout, the user is redirected back to the Login React application (i.e., **container_2_app** Login React application).

Figures 12 and 13 display the Profile React application Docker image creation and successful creation of docker image in the dashboard. The name of the image is **container_2_app**.

```

File Edit Selection View Go Run Terminal Help all_applications - Visual Studio Code
EXPLORER PROBLEMS OUTPUT DEBUG CONSOLE
TERMINAL
D:\DalhousieTotal\Study_Material\Term_3\CSCI_5410_SDP\3_Assignments\Assignment_2\part_a\all_applications\container_3_app>docker build -t container_3_app .
[*] Building 141.9s (12/12) FINISHED
    => [internal] load build definition from Dockerfile
    =>> transferring dockerfile: 311B
    => [internal] load .dockerignore
    =>> transferring context: 34B
    => [internal] load metadata for docker.io/library/node:alpine
    => [auth] library/node:pull token for registry-1.docker.io
    => [internal] load build context
    =>> transferring context: 699.67kB
    => [1/6] FROM docker.io/library/node:alpine@sha256:a17b3856d2e5d06385123f3924c36f5735fb1f690289ca69f2ac9c35fd06c009
    => [1/6] WORKDIR /app
    => [3/6] COPY package.json ./ 
    => [4/6] COPY package-lock.json ./ 
    => [5/6] RUN npm install
    => [6/6] COPY . / 
    =>> exporting to image
    =>> exporting layers
    =>> writing image sha256:6de8875f0967f1720043f6fdb79951c6750b335ea685529d0a209b27b9f3a0e
    =>> naming to docker.io/library/container_3_app

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

D:\DalhousieTotal\Study_Material\Term_3\CSCI_5410_SDP\3_Assignments\Assignment_2\part_a\all_applications\container_3_app>docker images
REPOSITORY          TAG      IMAGE ID      CREATED             SIZE
container_3_app     latest   6de8875f0967   23 seconds ago   571MB
container_2_app     latest   2fb64c3d306a   6 minutes ago    568MB
container_1_app     latest   589956388039   33 minutes ago   568MB

D:\DalhousieTotal\Study_Material\Term_3\CSCI_5410_SDP\3_Assignments\Assignment_2\part_a\all_applications\container_3_app>

```

Figure 12 - Docker image creation of container_3_app Profile React application

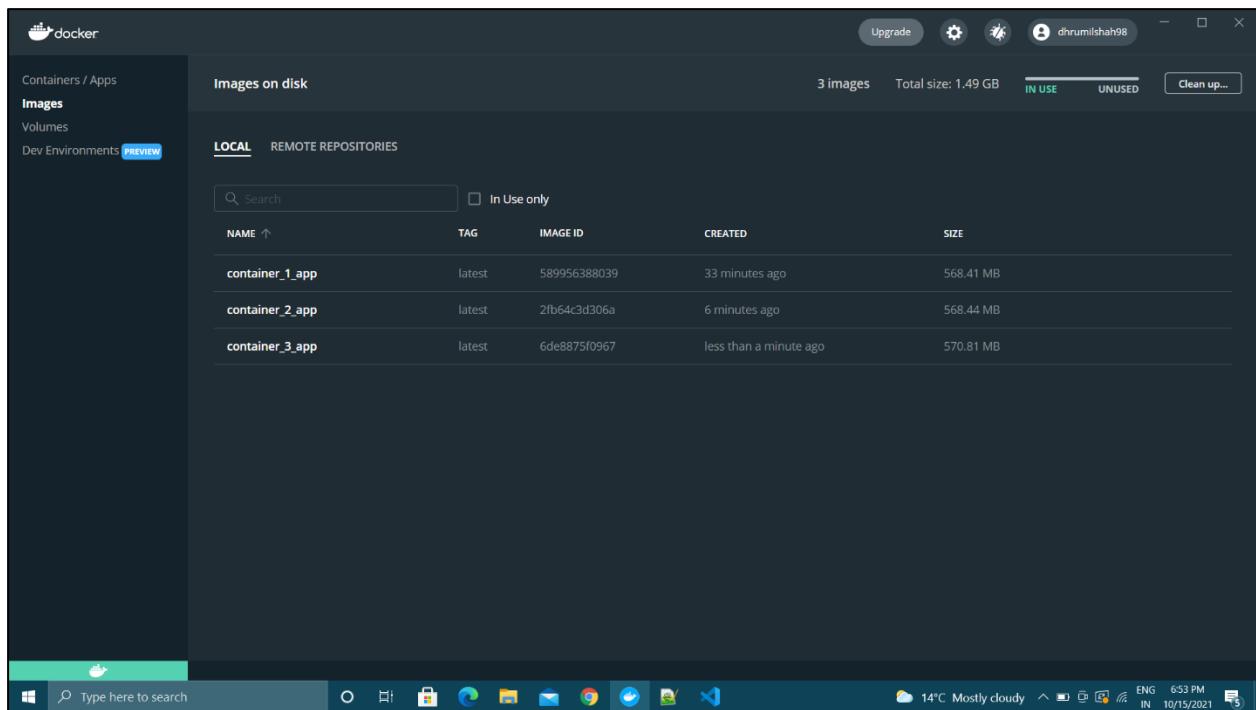


Figure 13 - Successful creation of Docker image of container_2_app Profile React application [2]

Running the Docker Images

Three Docker images namely Registration React application, Login React application and Profile React application runs on different ports namely 3000, 3001 and 3002 respectively.

Figure 14 displays the successful running of three Docker containers on three different ports.

```
C:\Windows\System32\cmd.exe
D:\Dalhousie\Total\Study_Material\Term_3\CSCI_5410_SOP\3_Assignments\Assignment_2\part_a\all_applications\container_1_app
49c5520fa8dad78e0e1d3577bd7d89261ec1af656d3e254f13d5ca2a09fe
D:\Dalhousie\Total\Study_Material\Term_3\CSCI_5410_SOP\3_Assignments\Assignment_2\part_a\all_applications\container_2_app
c99a0800beb1d4510d6fdde98a2eb396f599711a32ee3af4773525b128e357d
D:\Dalhousie\Total\Study_Material\Term_3\CSCI_5410_SOP\3_Assignments\Assignment_2\part_a\all_applications\container_3_app
f1432ee1e6f7f080deffd61150ef51968564d279fc3ca2a294c5825c2a378e7fc
D:\Dalhousie\Total\Study_Material\Term_3\CSCI_5410_SOP\3_Assignments\Assignment_2\part_a\all_applications\container_1_app
```

Figure 14 - Successful running of three Docker containers on ports 3000, 3001, and 3002 respectively

Figure 15 displays the successful running of three Docker containers in the Docker Dashboard.

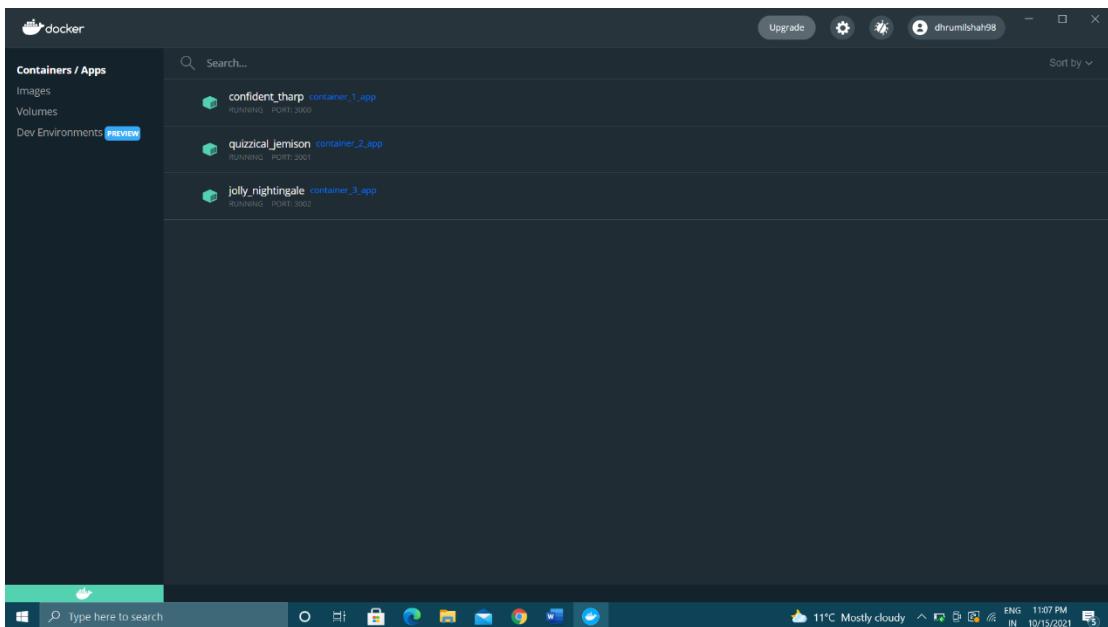


Figure 15 - Successful running of three Docker containers in the Docker Dashboard [2]

Website Working and Firestore Images

Figure 16 displays the registration page displayed using the first Docker container (i.e., container_1_app).

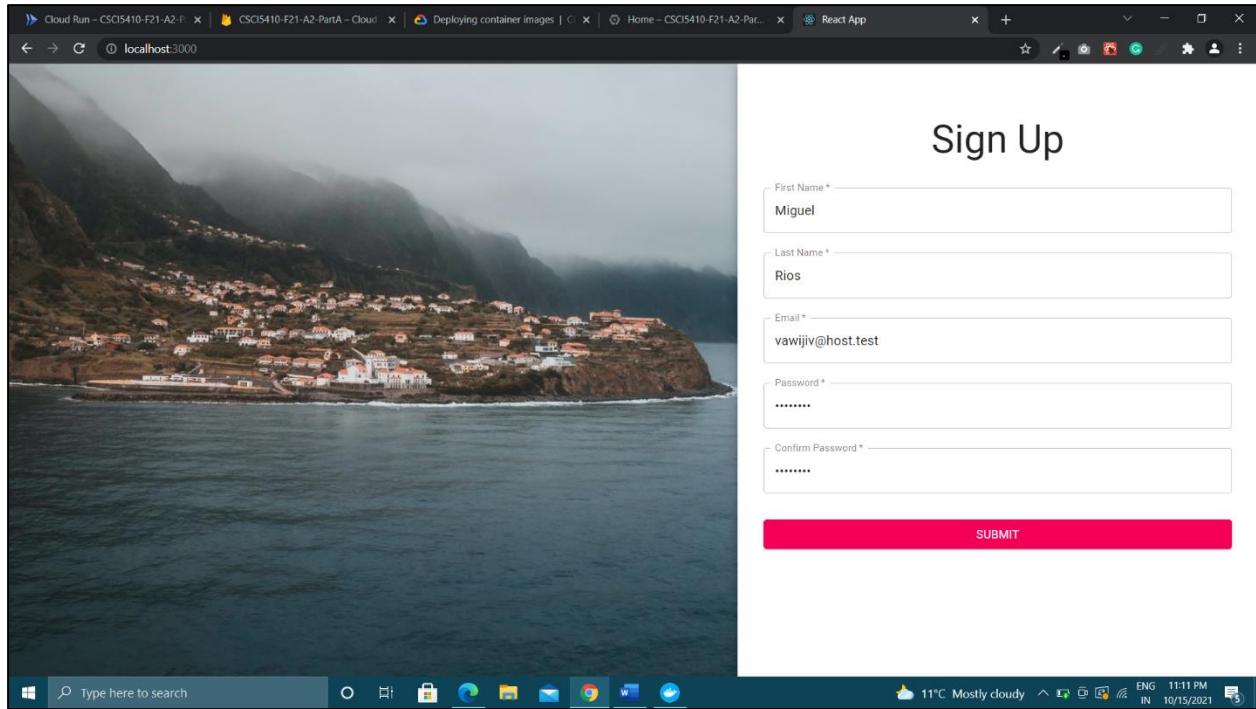


Figure 16 - Registration application displayed using container_1_app

Figure 17 displays that on successful registration, data is stored in the Firebase Firestore database with status value **offline**.

The screenshot shows the Firebase Cloud Firestore interface. On the left, there's a sidebar with 'Authentication', 'Firestore Database' (selected), 'Realtime Database', 'Storage', 'Hosting', 'Functions', and 'Machine Learning'. The main area is titled 'Cloud Firestore' with tabs for 'Data', 'Rules', 'Indexes', and 'Usage'. It shows a hierarchical view: 'users > v1RKse2ro1GJr7Y6Ru0t'. The right panel displays the document details for 'v1RKse2ro1GJr7Y6Ru0t' with fields: email: "vawijiv@host.test", firstName: "Miguel", lastName: "Rios", password: "p@ssw0rd", status: "offline", and timestamp: October 15, 2021 at 11:11:08 PM UTC-3.

Figure 17 - Successful user registration with status value offline [1]

Figure 18 displays the login page displayed using the second Docker container (i.e., container_2_app).

The screenshot shows a login page with a decorative background image of various flowers in jars. The page has a 'Sign In' header. It contains two input fields: 'Email *' with the value 'vawijiv@host.test' and 'Password *' with a masked value. A red 'SUBMIT' button is at the bottom. The browser address bar shows 'localhost:3001'.

Figure 18 – Login application displayed using conatiner_2_app

Figure 19 displays the profile application displayed using the third Docker container (i.e., container_3_app).

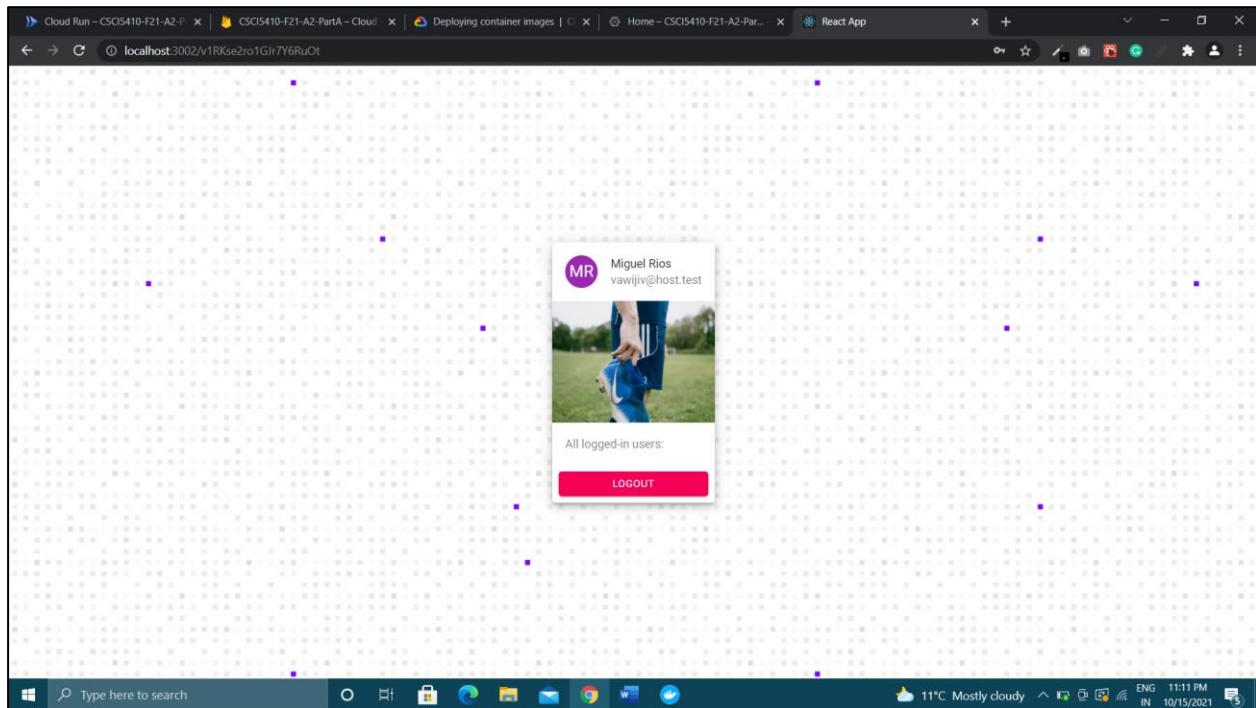


Figure 19 - Profile application displayed using container_3_app

Figure 20 displays the successful login of the user with the status value online.

The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar includes 'Authentication', 'Firestore Database' (which is selected), 'Realtime Database', 'Storage', 'Hosting', 'Functions', and 'Machine Learning'. The main area displays the 'Cloud Firestore' interface with a 'Data' tab selected. The path 'users > v1RKse2ro1GJr... > v1RKse2ro1GJr7Y6Ru0t' is shown. The document details are as follows:

```

email: "vawijiv@host.test"
firstName: "Miguel"
lastName: "Rios"
password: "p@ssw0rd"
status: "online"
timestamp: October 15, 2021 at 11:11:08 PM UTC-3

```

Figure 20 - Successful login changes the value of status to online [1]

Figure 21 displays the successful user logout data in Firestore database with the status changed to offline.

The screenshot shows the Firebase Cloud Firestore interface, identical to Figure 20 but after a successful logout. The document details are as follows:

```

email: "vawijiv@host.test"
firstName: "Miguel"
lastName: "Rios"
password: "p@ssw0rd"
status: "offline"
timestamp: October 15, 2021 at 11:11:08 PM UTC-3

```

Figure 21 - Successful logout changes the status value to offline [1]

Figure 22 displays the information of the other logged-in users (i.e., users with status – online).

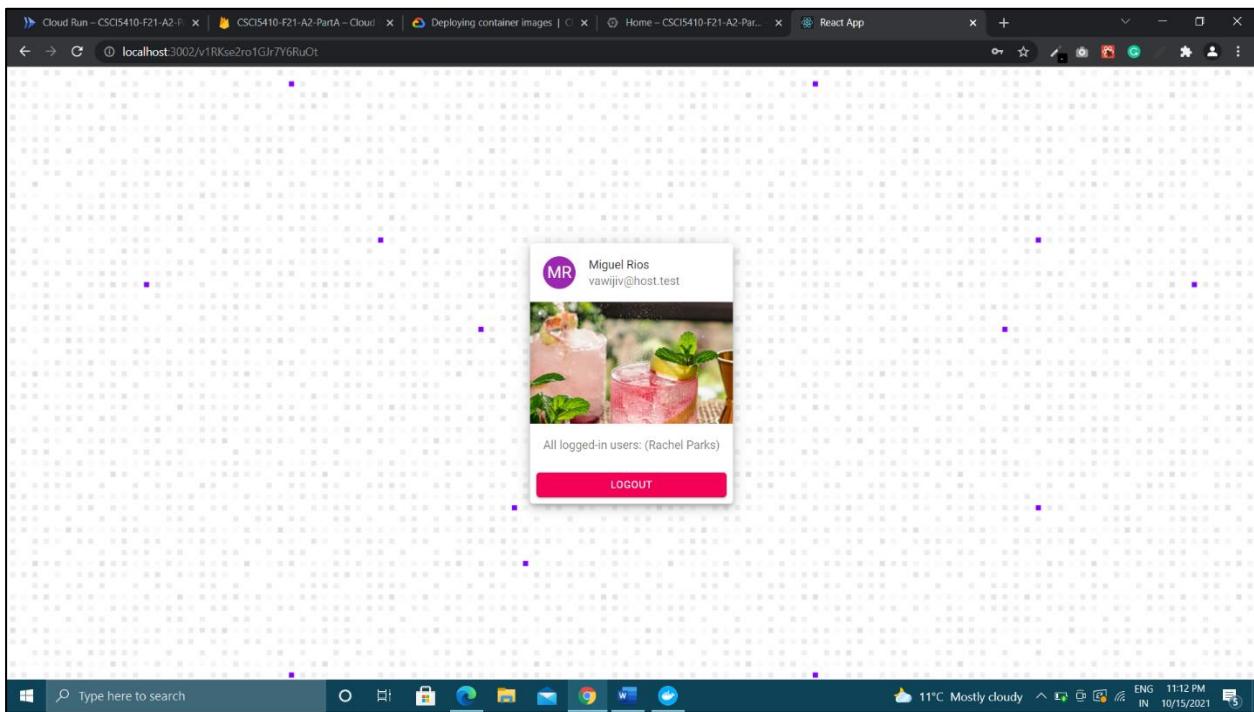


Figure 22 - Successful display of all the other users who are online

Figure 23 displays the Firestore data of other logged-in users (i.e., status – online).

The screenshot shows the Firebase Cloud Firestore console. The left sidebar lists 'Authentication', 'Firestore Database', 'Realtime Database', 'Storage', 'Hosting', 'Functions', and 'Machine Learning'. The main area is titled 'Cloud Firestore' and shows a 'Data' view. It displays a collection named 'users' with one document named 'gLPbIBzEScfljbINZUPV'. The document contains the following data:

```

email: "nu@example.com"
firstName: "Rachel"
lastName: "Parks"
password: "p@ssw0rd"
status: "online"
timestamp: October 15, 2021 at 11:12:32 PM UTC-3
  
```

Figure 23 - Users with status online are displayed in the Profile application [1]

Docker and Google Container Registry

Steps

Step 1 – Downloaded Google Cloud SDK [3] from Google’s official website and installed it on my machine.

Step 2 – Logged-in to Google Cloud SDK using my Google account.

Step 3 – Selected the project CSCI5410-F21-A2-PartA to work and copied the project id csci5410-f21-a2-partA for tagging the docker images.

Step 4 – View all docker images using command “**docker images**” (In my case I wanted to upload 3 Docker images container_1_app, container_2_app, and container_3_app)

Step 5 – Tag all the three images using commands

- docker tag container_1_app gcr.io/csci5410-f21-a2-partA/container_1_app
- docker tag container_2_app gcr.io/csci5410-f21-a2-partA/container_2_app
- docker tag container_3_app gcr.io/csci5410-f21-a2-partA/container_3_app

Step 6 – Configured Cloud Shell to authenticate my machine using command

- net localgroup docker-users DESKTOP-4P1CDV0\admin /add

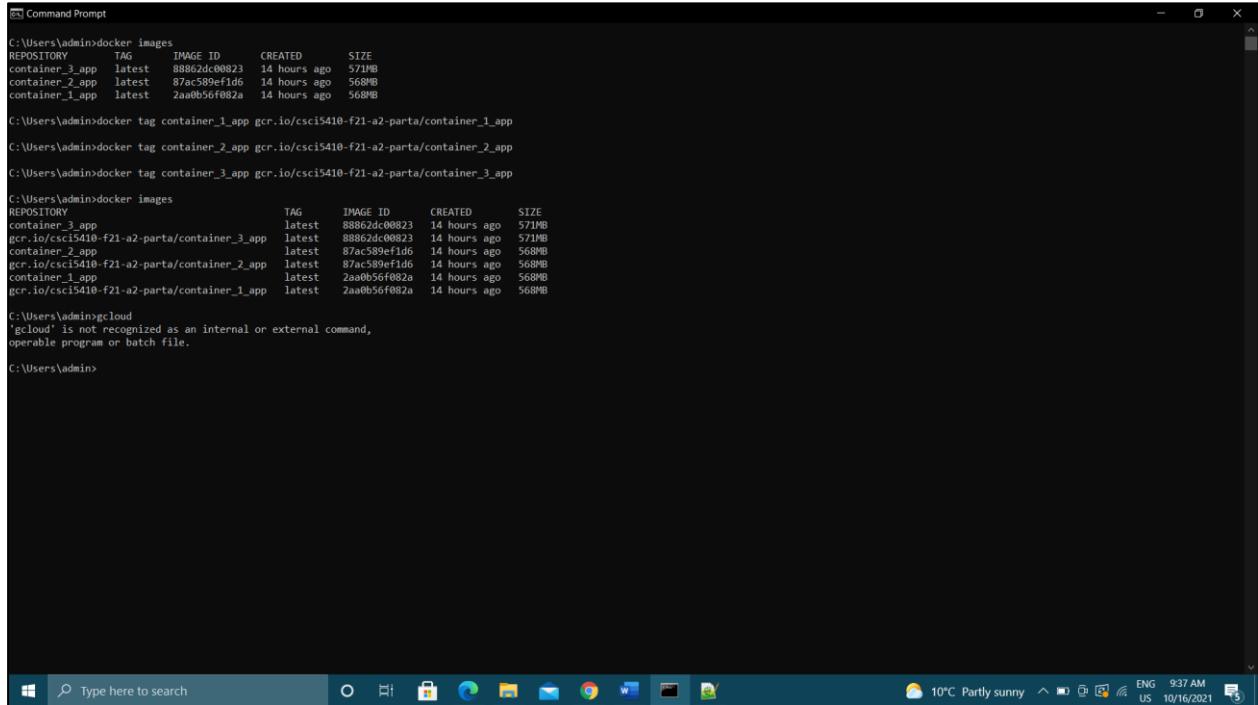
Step 7 – Configure Docker with the following command:

- gcloud auth configure-docker

Step 8 – Push images to Google Container Registry

- gcloud docker push gcr.io/csci5410-f21-a2-partA/container_1_app
- gcloud docker push gcr.io/csci5410-f21-a2-partA/container_2_app
- gcloud docker push gcr.io/csci5410-f21-a2-partA/container_3_app

Figure 24 shows tagging of three docker images namely container_1_app, container_2_app, and container_3_app.



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The command history is as follows:

```
C:\Users\admin>docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
container_3_app     latest   88862dc00823  14 hours ago  571MB
container_2_app     latest   87ac589ef1d6  14 hours ago  568MB
container_1_app     latest   2aa0b56f082a  14 hours ago  568MB

C:\Users\admin>docker tag container_1_app gcr.io/csci5410-f21-a2-parta/container_1_app
C:\Users\admin>docker tag container_2_app gcr.io/csci5410-f21-a2-parta/container_2_app
C:\Users\admin>docker tag container_3_app gcr.io/csci5410-f21-a2-parta/container_3_app

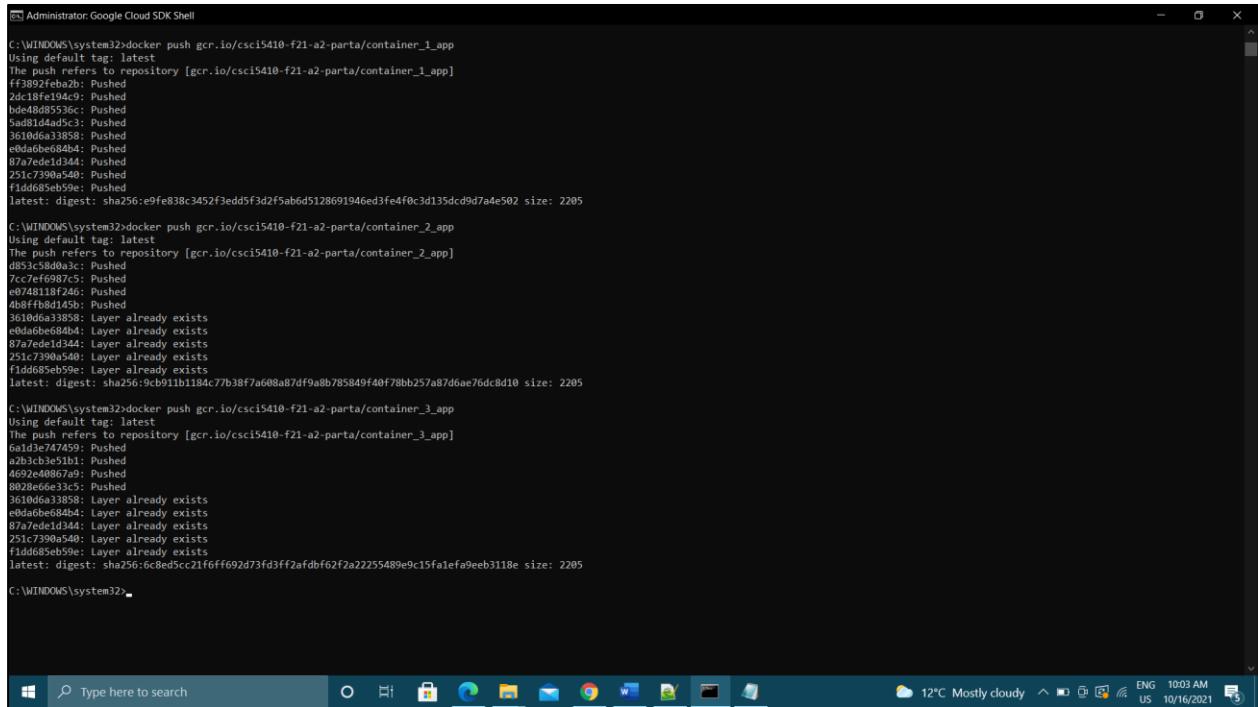
C:\Users\admin>docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
container_3_app     latest   88862dc00823  14 hours ago  571MB
gcr.io/csci5410-f21-a2-parta/container_3_app  latest   88862dc00823  14 hours ago  571MB
container_2_app     latest   87ac589ef1d6  14 hours ago  568MB
gcr.io/csci5410-f21-a2-parta/container_2_app  latest   87ac589ef1d6  14 hours ago  568MB
container_1_app     latest   2aa0b56f082a  14 hours ago  568MB
gcr.io/csci5410-f21-a2-parta/container_1_app  latest   2aa0b56f082a  14 hours ago  568MB

C:\Users\admin>gcloud
'gcloud' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\admin>
```

Figure 24 – Tagging of three docker images namely container_1_app, container_2_app, and container_3_app

Figure 25 shows pushing three tagged docker images to Google Container Registry. These images are inside the project – csci5410-f21-a2-parta.



```

Administrator: Google Cloud SDK Shell
C:\WINDOWS\system32>docker push gcr.io/csci5410-f21-a2-parta/container_1_app
Using default tag: latest
The push refers to repository [gcr.io/csci5410-f21-a2-parta/container_1_app]
ff3892feba2b: Pushed
2dc18fe194c9: Pushed
bde48d8536c: Pushed
5ad81d44d5c3: Pushed
3610d6a33b58: Pushed
e0748118f746: Pushed
a8bf118f746: Pushed
87a7edel1d34: Pushed
251c7390a540: Pushed
f1dd685eb9e: Pushed
latest: digest: sha256:e9fe838c3452f3edd5f3d2f5ab6d5128691946ed3fe4f0c3d135cd9d7a4e502 size: 2205

C:\WINDOWS\system32>docker push gcr.io/csci5410-f21-a2-parta/container_2_app
Using default tag: latest
The push refers to repository [gcr.io/csci5410-f21-a2-parta/container_2_app]
d853c58d0a3c: Pushed
7cce7f6987c5: Pushed
e0748118f746: Pushed
a8bf118f746: Pushed
87a7edel1d34: Layer already exists
251c7390a540: Layer already exists
f1dd685eb9e: Layer already exists
latest: digest: sha256:9cb91b1184c77b38f7a608a87df9ab785849f40f78bb257a87d6ae76dc8d10 size: 2205

C:\WINDOWS\system32>docker push gcr.io/csci5410-f21-a2-parta/container_3_app
Using default tag: latest
The push refers to repository [gcr.io/csci5410-f21-a2-parta/container_3_app]
6a1d3c747459: Pushed
a2b3c3b51b1: Pushed
a692e40867a9: Pushed
a8bf118f746: Pushed
3610d6a33b58: Layer already exists
e0748118f746: Layer already exists
87a7edel1d34: Layer already exists
251c7390a540: Layer already exists
f1dd685eb9e: Layer already exists
latest: digest: sha256:6c8ed5cc21f6ff692d73fd3ff2afdbf62f2a22255489e9c15fa1ef9eeb3118e size: 2205

C:\WINDOWS\system32>

```

Figure 25 - Docker push to Google Container Registry

Figure 26 displays three Docker images uploaded to Google Container Registry service.

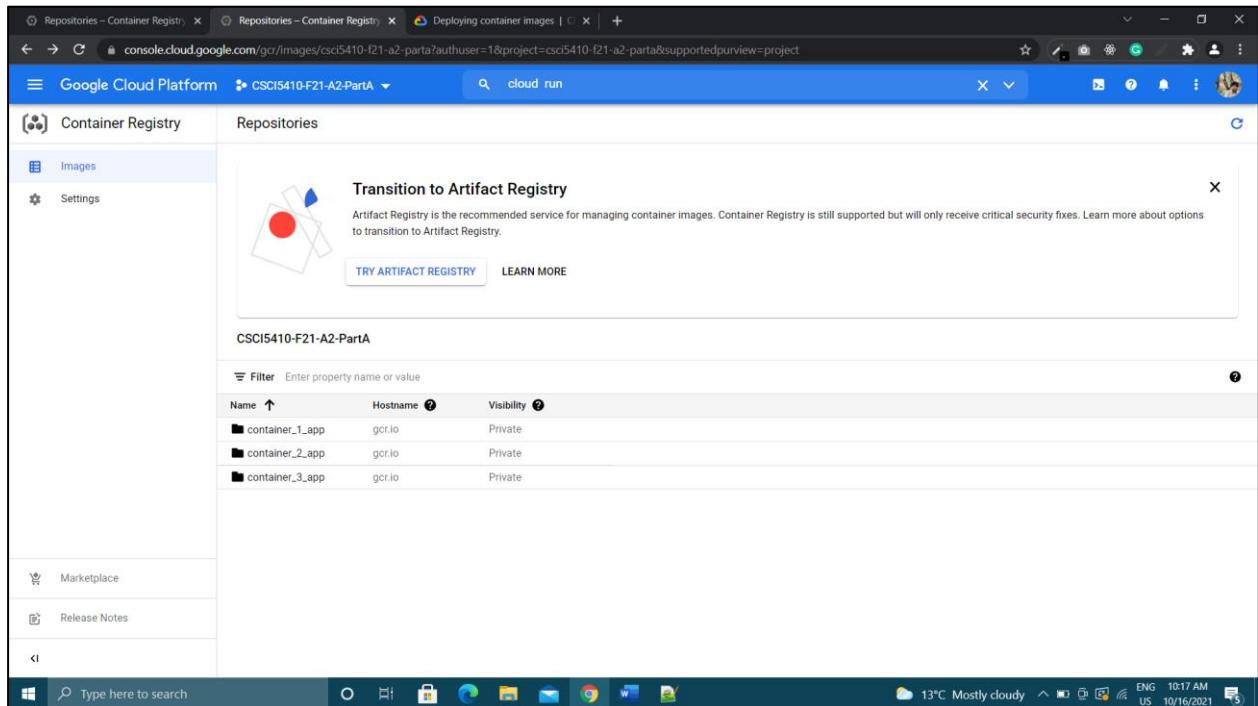


Figure 26 - Docker images uploaded to Google Container Registry [4]

Google Cloud Run and Website hosted on GCR

Figure 27, 28, 29, 30, 31, 32, 33, 34, and 35 displays the configuration of Docker images namely container_1_app, container_2_app, and container_3_app on Google Cloud Run.

The screenshot shows the 'Create service' page in the Google Cloud Platform Cloud Run interface. The service is named 'container_1_app' and is deployed to the 'us-central1 (Iowa)' region. The Docker image URL is set to 'gcr.io/csci5410-f21-a2-partA/container_1_app:latest'. The CPU allocation is set to 'CPU is only allocated during request processing'. Autoscaling is configured with a minimum of 0 instances and a maximum of 4 instances. Advanced settings include a container port of 3000 and a command of 'java -jar app.jar'. Capacity settings show 512 MB of memory and 1 vCPU allocated. Request timeout is set to 300 seconds, and maximum concurrent requests per container are set to 80. The 'NEXT' button is visible at the bottom.

Figure 27 - Configuration of container_1_app Docker image on Google Cloud Run [5]

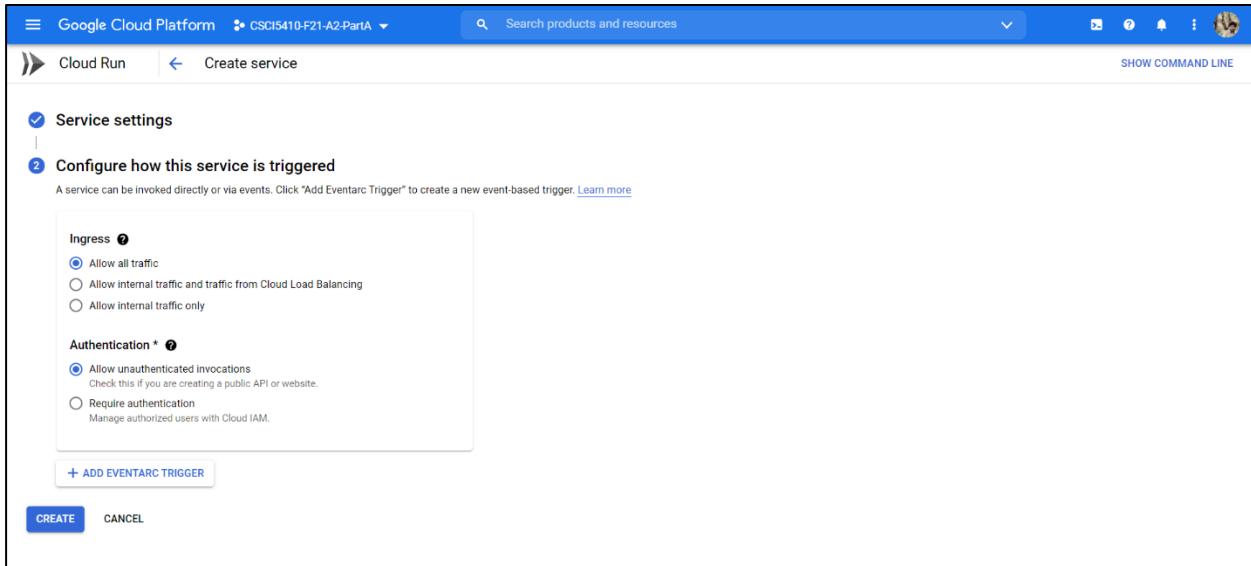


Figure 28 - Configuration of container_1_app Docker image on Google Cloud Run (contd.) [5]

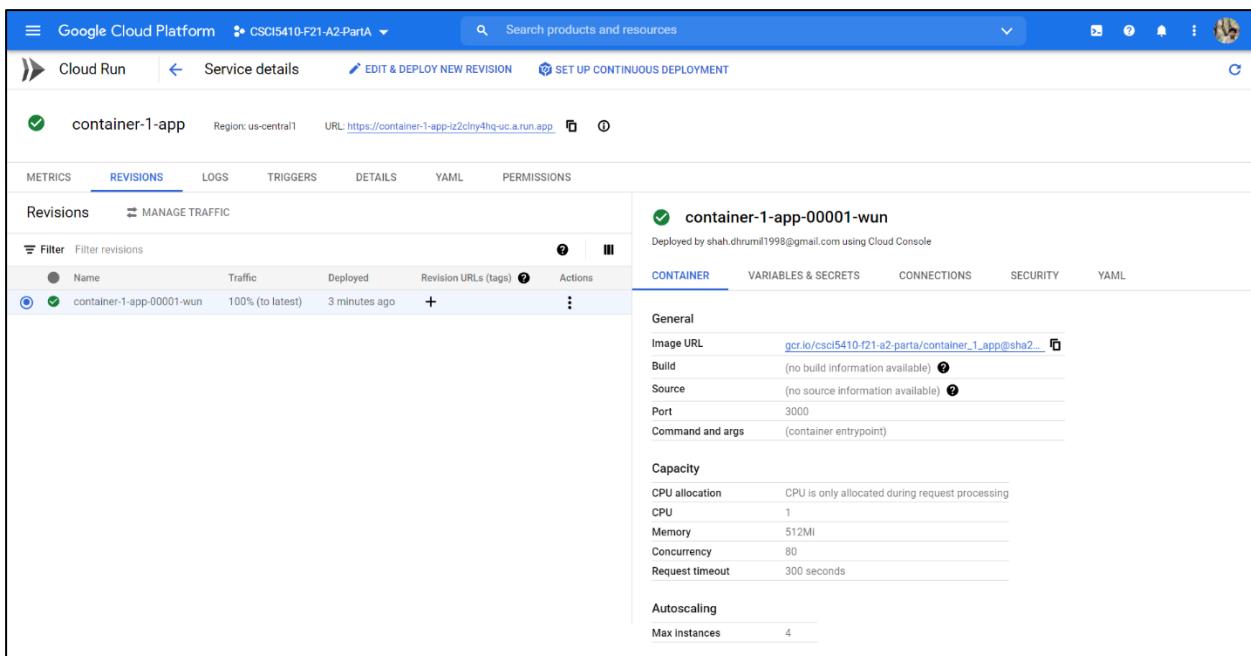


Figure 29 - container_1_app (React Registration application) hosted on Google Cloud Run [5]

The screenshot shows the 'Create service' page in the Google Cloud Platform Cloud Run interface. The service is named 'container_2_app'. The configuration includes:

- Service settings:** Deploy one revision from an existing container image (gcr.io/csci5410-f21-a2-partA/container_2_app:latest).
- CPU allocation and pricing:** CPU is only allocated during request processing.
- Autoscaling:** Minimum number of instances: 0; Maximum number of instances: 4.
- Advanced settings:**
 - General:** Container port: 3001.
 - Capacity:** Memory: 512 MB; CPU: 1 vCPU.
 - Request timeout:** 300 seconds.
 - Maximum requests per container:** 80.

At the bottom, there is a 'NEXT' button and 'CREATE' or 'CANCEL' options.

Figure 30 - Configuration of container_2_app Docker image on Google Cloud Run [5]

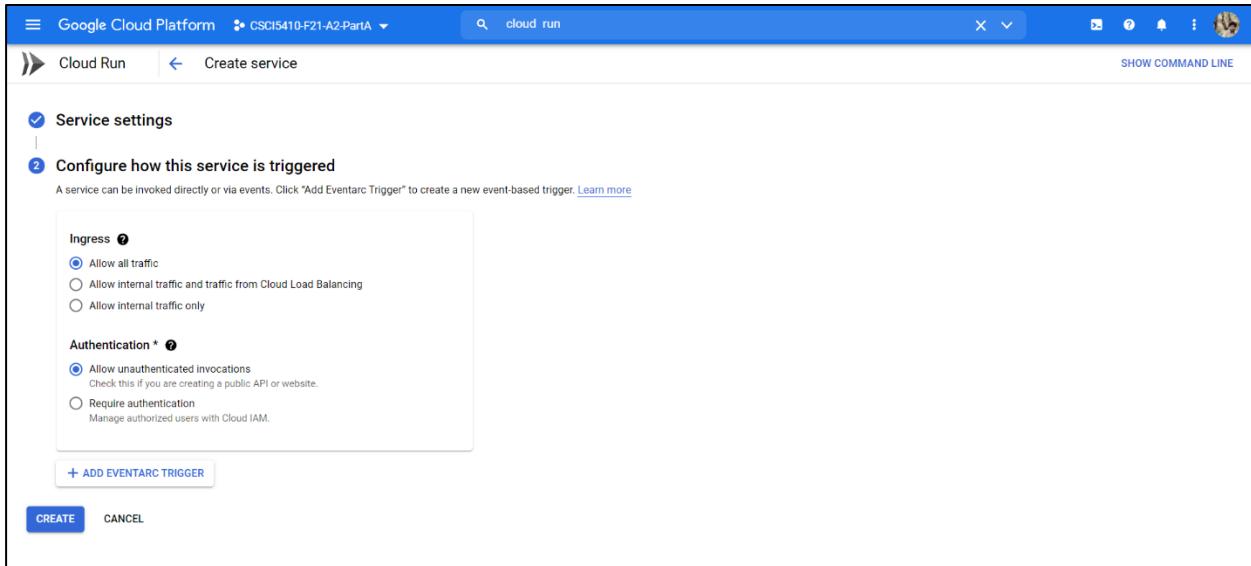


Figure 31 - Configuration of container_2_app Docker image on Google Cloud Run (contd.) [5]

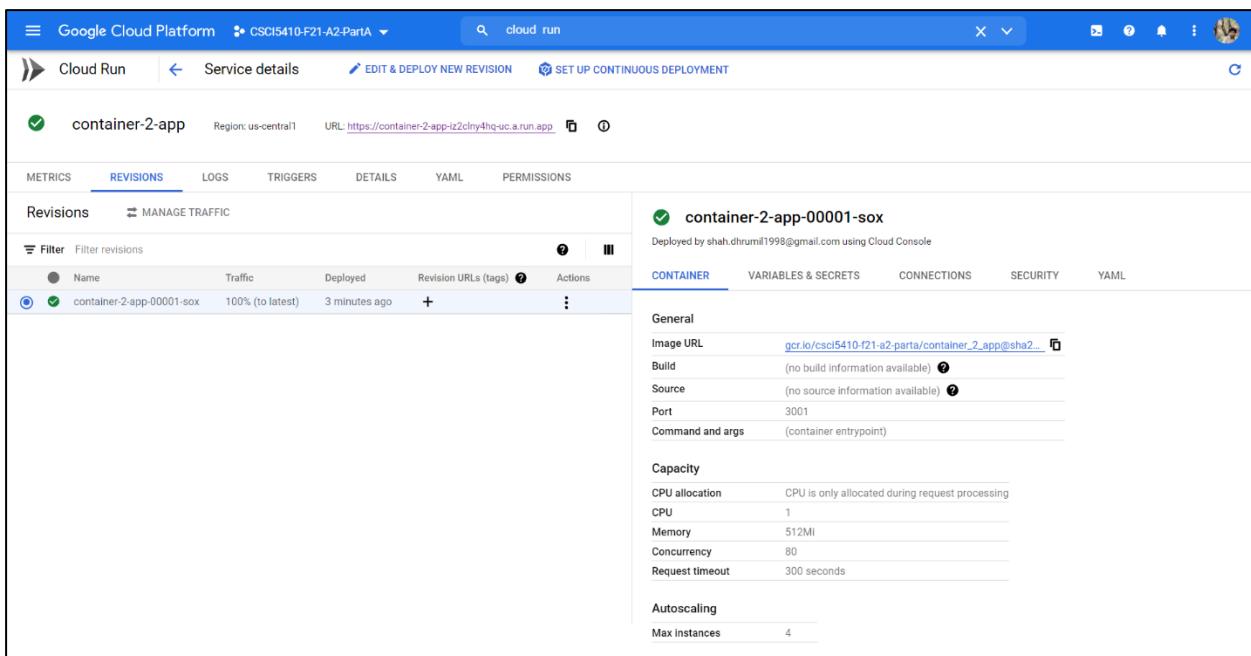


Figure 32 - container_2_app (Login React application) hosted on Google Cloud Run [5]

The screenshot shows the 'Create service' page in the Google Cloud Platform Cloud Run interface. The service is named 'container_3-app'. The configuration includes:

- Service settings:** Deploy one revision from an existing container image (gcr.io/csci5410-f21-a2-partA/container_3_app:latest).
- CPU allocation and pricing:** CPU is only allocated during request processing.
- Autoscaling:** Minimum number of instances: 0; Maximum number of instances: 4.
- Advanced settings:**
 - General:** Container port: 3002.
 - Capacity:** Memory: 512 MB; CPU: 1 vCPU.
 - Request timeout:** 300 seconds.
 - Maximum requests per container:** 80.

At the bottom, there is a 'NEXT' button and 'CREATE' or 'CANCEL' options.

Figure 33 - Configuration of container_2_app Docker image on Google Cloud Run [5]

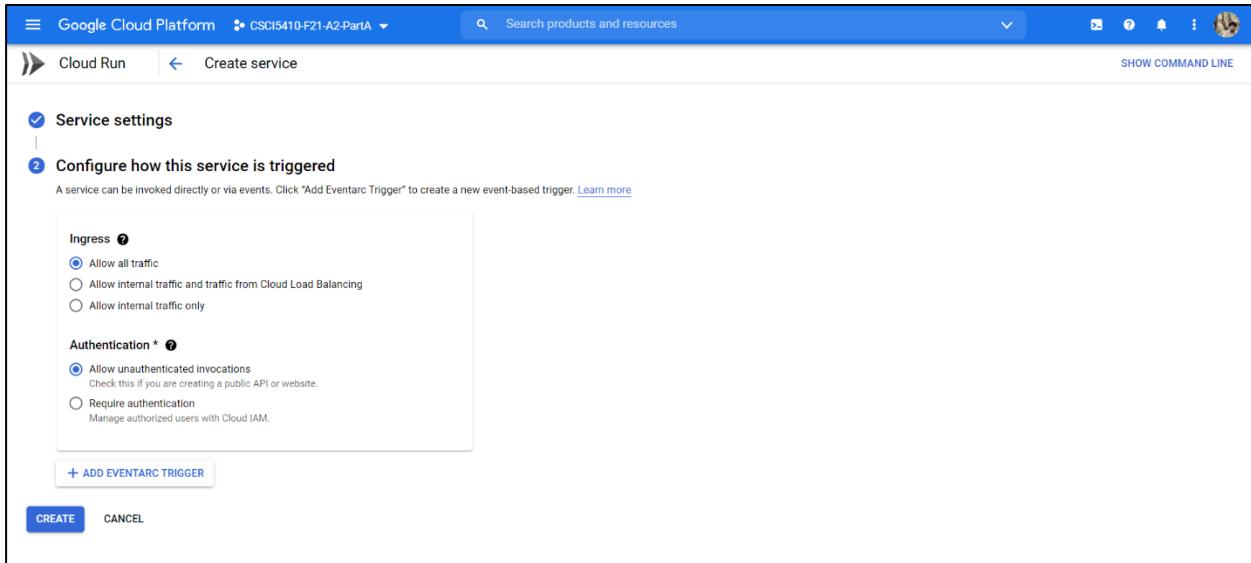


Figure 34 - Configuration of container_2_app Docker image on Google Cloud Run (contd.) [5]

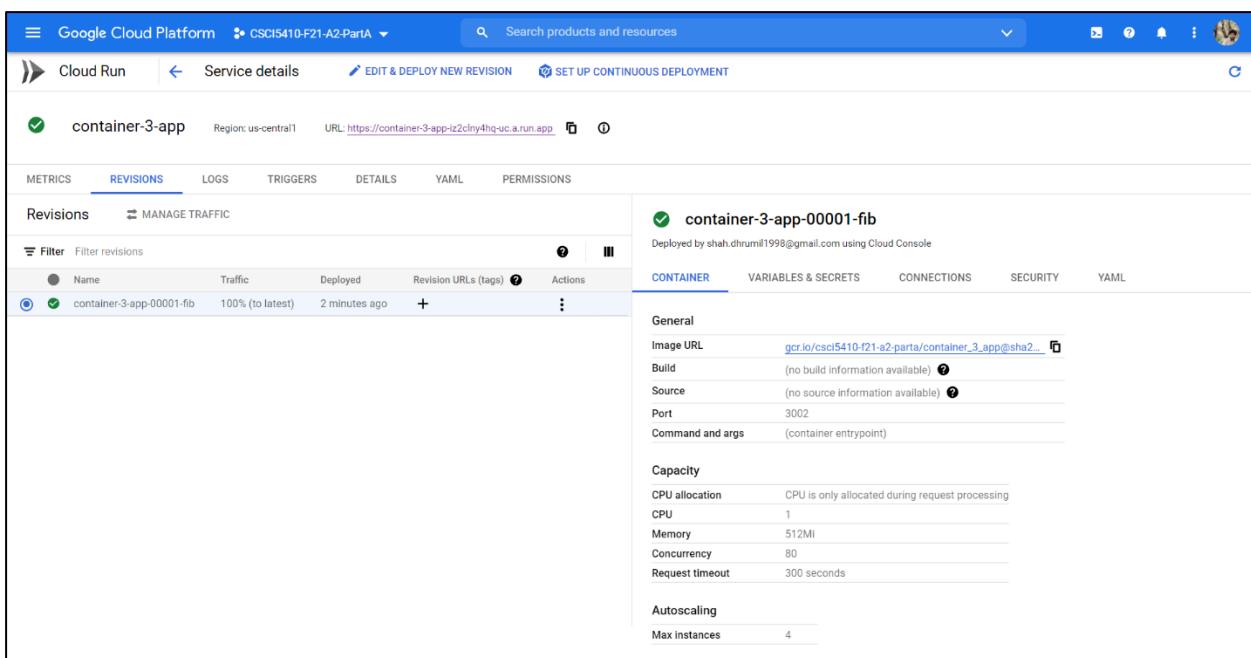


Figure 35 - container_3_app (Profile React application) hosted on Google Cloud Run [5]

Figure 36, 37 and 38 displays the website hosted on Google Cloud Run.

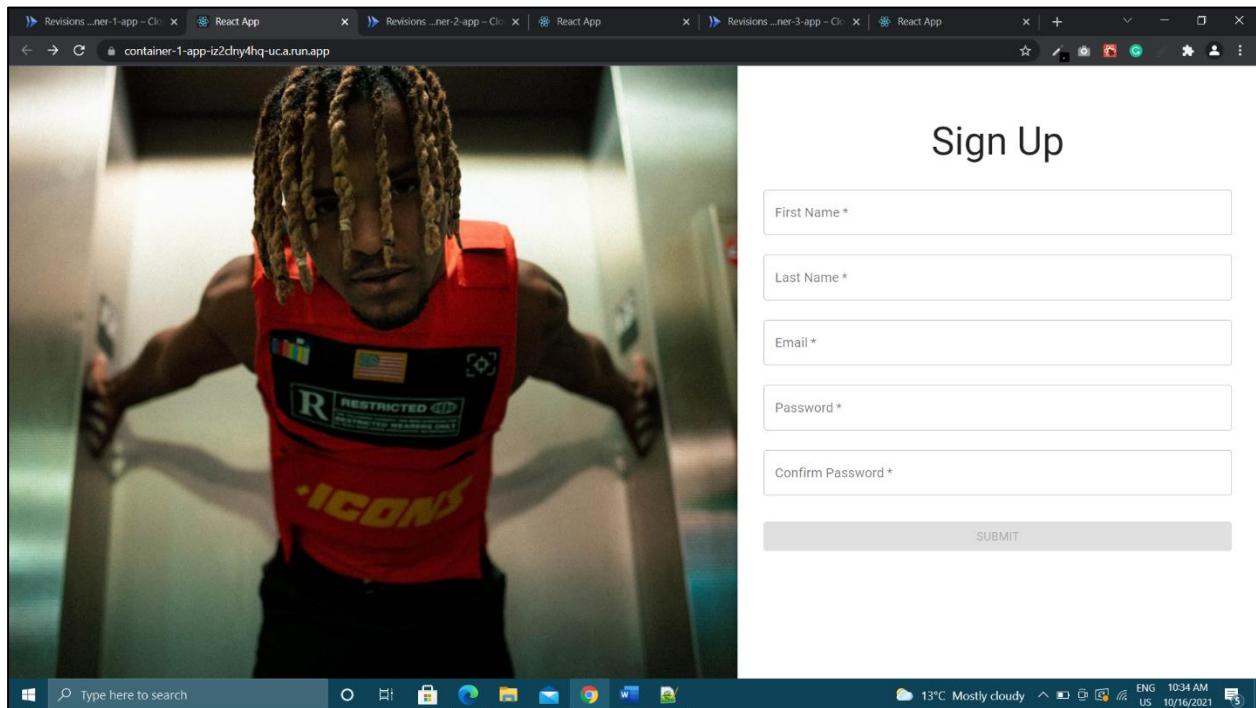


Figure 36 - container_1_app (Registration React application) website hosted on Google Cloud Run

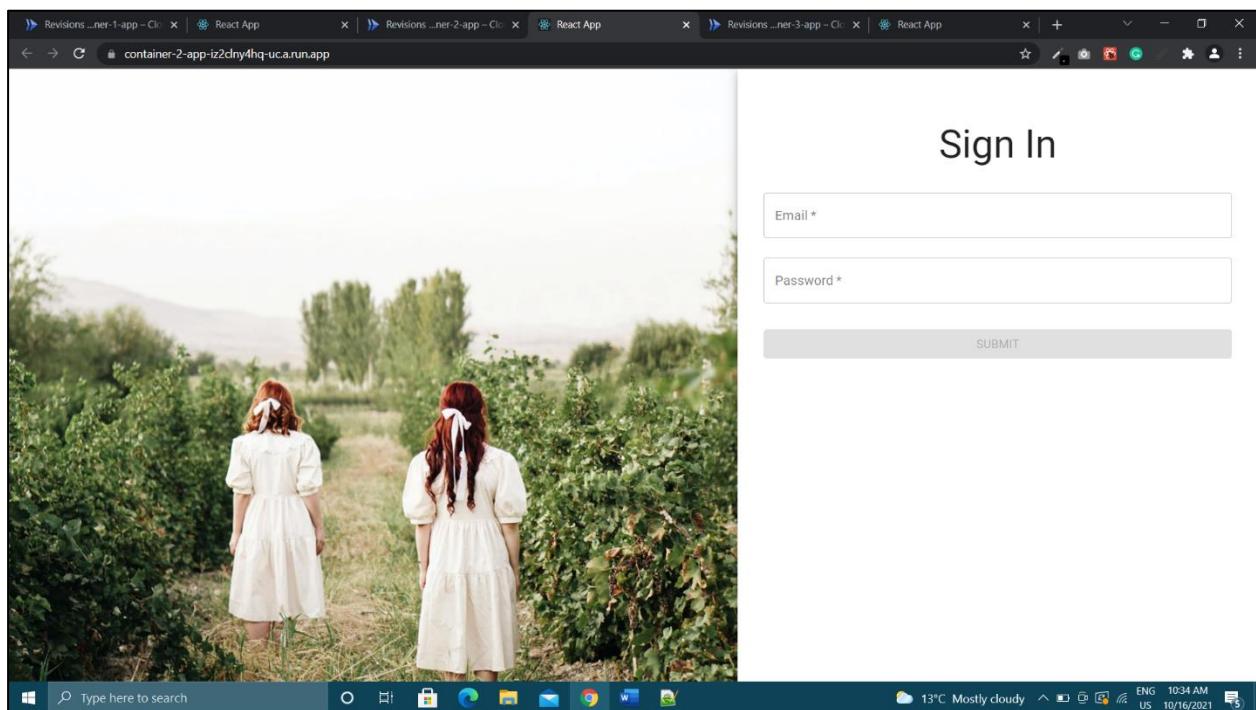


Figure 37 - container_2_app (Login React application) website hosted on Google Cloud Run

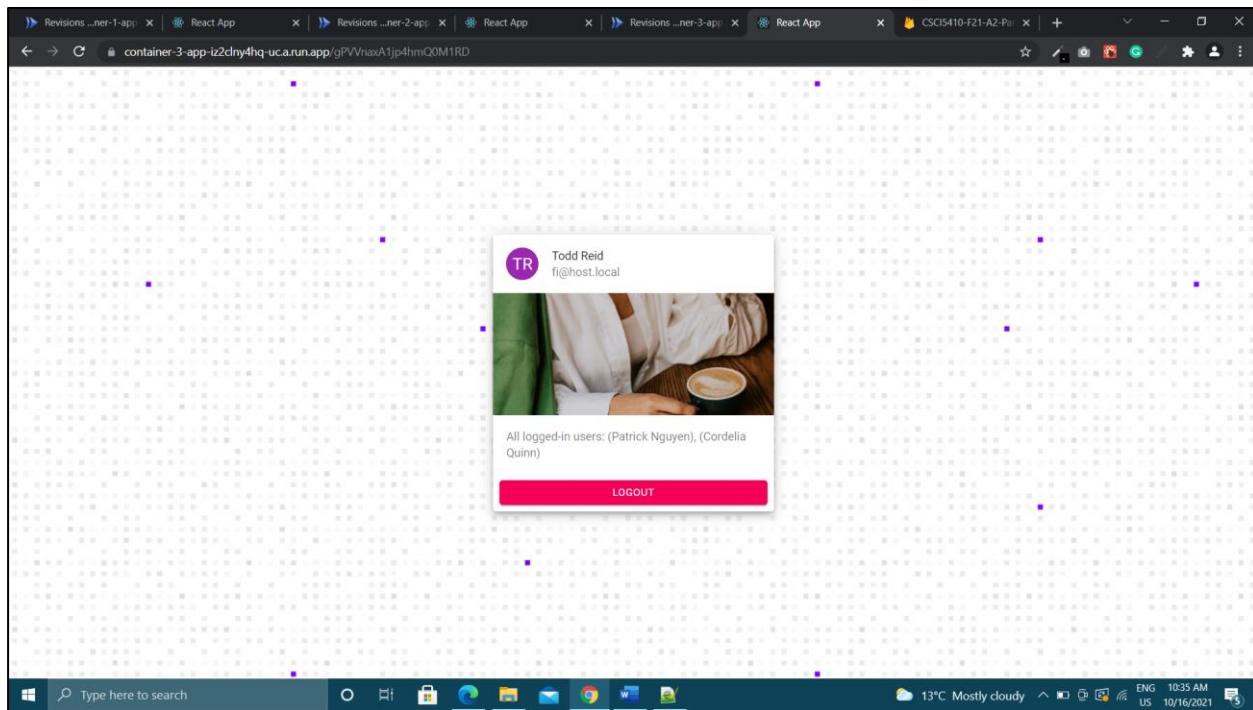


Figure 38 - container_3_app (Profile React application) hosted on Google Cloud Run

Test Cases and Important Code

- ⇒ Method to validate first name, last name, email, password and confirm password. (Registration React application)

```
const validate = (e) => {
  switch (e.target.name) {
    case 'firstName':
      const isFNameCorrect = RegExp(/^[A-Za-z\d]+$/).test(e.target.value);
      if (e.target.value === "" || e.target.value === null) {
        errors["firstName"] = "First name is required."
        errors["firstNameValid"] = false;
      } else if (!isFNameCorrect) {
        errors["firstName"] = "First name can only have alpha-numeric characters."
        errors["firstNameValid"] = false;
      } else {
        errors["firstName"] = "";
        errors["firstNameValid"] = true;
      }
      break;
    case 'lastName':
      const isLNameCorrect = RegExp(/^[A-Za-z\d]+$/).test(e.target.value);
      if (e.target.value === "" || e.target.value === null) {
        errors["lastName"] = "Last name is required."
        errors["lastNameValid"] = false;
      } else if (!isLNameCorrect) {
        errors["lastName"] = "Last name can only have alpha-numeric characters."
        errors["lastNameValid"] = false;
      } else {
        errors["lastName"] = "";
        errors["lastNameValid"] = true;
      }
      break;
    case 'email':
      // Email regex source - https://stackoverflow.com/questions/46155/how-to-validate-an-email-address-in-javascript
      const isEmailCorrect =
        RegExp(/^\w+([.-]\w+)*@\w+([.-]\w+)*\.\w{2,3}((\.\w{2,3})?)+$/).test(e.target.value);
      if (e.target.value === "" || e.target.value === null) {
        errors["email"] = "Email name is required."
        errors["emailValid"] = false;
      } else if (!isEmailCorrect) {
        errors["email"] = "Please enter a valid email address."
        errors["emailValid"] = false;
      }
  }
}
```

```
        } else {
            errors["email"] = ""
            errors["emailValid"] = true;
        }
        break;
    case 'password':
        const isPasswordCorrect = RegExp(/^[A-Za-z\d@$!%*#?&_]+$/).test(e.target.value);
        if (e.target.value === "" || e.target.value === null) {
            errors["password"] = "Password is required."
            errors["passwordValid"] = false;
        } else if (e.target.value.length < 8) {
            errors["password"] = "Password must be at least 8 characters long."
            errors["passwordValid"] = false;
        } else if (!isPasswordCorrect) {
            errors["password"] = "Password can only have alpha-numeric and special characters."
            errors["passwordValid"] = false;
        } else {
            errors["password"] = ""
            errors["passwordValid"] = true;
        }
        break;
    case 'confirmPassword':
        if (e.target.value === "" || e.target.value === null) {
            errors["confirmPassword"] = "Confirm password is required."
            errors["confirmPasswordValid"] = false;
        } else if (e.target.value !== userData.password) {
            errors["confirmPassword"] = "Confirm password must match the password field."
            errors["confirmPasswordValid"] = false;
        } else {
            errors["confirmPassword"] = ""
            errors["confirmPasswordValid"] = true;
        }
        break;
    default:
        break;
}
setErrors(errors);
};
```

⇒ Method to store the data in Firebase Firestore (Registration React application) – User registration process.

```
const storeDataInFirestore = async () => {
  try {
    const docRef = await addDoc(collection(firestoreDB, "users"), {
      firstName: userData.firstName,
      lastName: userData.lastName,
      email: userData.email,
      password: userData.password,
      status: 'offline',
      timestamp: serverTimestamp(),
    });
    console.log("Document written with ID: ", docRef.id);
    updateUserData({
      firstName: "",
      lastName: "",
      email: "",
      password: "",
      confirmPassword: "",
    });
    window.location.href = 'http://localhost:3001/';
  } catch (e) {
    console.error("Error adding user: ", e);
  }
}
```

⇒ Method to perform user login (Login React application) – User login

```
const loginUser = async () => {
  try {
    const queryObj = query(collection(firestoreDB, "users"), where("email", "==", userData.email));
    const querySnapshot = await getDocs(queryObj);
    var userExists = false;
    querySnapshot.forEach(async (userDoc) => {
      if (userData.password === userDoc.data().password) {
        userExists = true;
        console.log("User found!");
        console.log(userDoc.id, " => ", userDoc.data());
        const userDocRef = doc(firestoreDB, "users", userDoc.id);
        await updateDoc(userDocRef, {
          status: "online",
        });
      }
    });
  } catch (e) {
    console.error("Error logging in user: ", e);
  }
}
```

```

        window.location.href = 'http://localhost:3002/' + userDoc.id;
    } else {
        userExists = false;
    }
});
if (!userExists) {
    console.error("User not found!");
}
} catch (e) {
    console.error("Error reading user: ", e);
}
};

```

⇒ Method to fetch all the users who are online. (Profile React application)

```

useEffect(async () => {
    const onlineUsersQueryObj = query(collection(firestoreDB, "users"), where("status", "==", "online"));
    const onlineUserQuerySnapshot = await getDocs(onlineUsersQueryObj);
    var allOnlineUsers = "";
    var firstTime = true;
    onlineUserQuerySnapshot.forEach(async (userDoc) => {
        if (userDoc.data().status === "online") {
            if (userDoc.id !== userDocId) {
                if (firstTime) {
                    allOnlineUsers = "(" + userDoc.data().firstName + " " + userDoc.data().lastName
+ ")";
                    firstTime = false;
                } else {
                    allOnlineUsers = allOnlineUsers + ", " + "(" + userDoc.data().firstName + " " +
userDoc.data().lastName + ")";
                }
            } else {
                setLoggedInUser({
                    firstName: userDoc.data().firstName,
                    lastName: userDoc.data().lastName,
                    email: userDoc.data().email,
                });
            }
        });
    });
    setOnlineUsers(allOnlineUsers);
});

```

⇒ Method to perform logout (Profile React application)

```
const logout = async () => {
  const userDocRef = doc(firestoreDB, "users", userDocId);
  await updateDoc(userDocRef, {
    status: "offline",
  });
  window.location.href = 'http://localhost:3001/';
}
```

Summary

Serverless Data Processing allows containerizing applications and running them remotely using the cloud services provided by providers like Amazon, Google, Microsoft Azure, and so on. The modules visited for this exercise are Docker containers, Google Container Registry, and Google Cloud Run.

Docker Containers are services in the form of platform to create, run, and deploy applications in various containers, each of which works independently irrespective of the other containers. These containers contain the web applications required to be run on the cloud platform. A docker container is a light weight package of software that works in isolation. Running these docker containers on cloud platforms, helps the applications to be accessed from anywhere.

Google Container Registry is a service by Google used to store private images. The service uses Artifact Registry functions like creating multiple registries in same or multiple regions, and so on. The registry helps store this images from Cloud Build, deploy them to Google Cloud Run directly using services such as Google Cloud Run, Compute Engine, Kubernetes Engine. Identity and Access Management control provides consistent credentials and access control. The containers created here are securely built from the images. The uploaded images in the Container Registry are then deployed on one of the engines provided by Google to run these containers remotely from anywhere.

Google Cloud Run is a computing platform managed by Google that eases the job of developers for deploying and running containers that may be invoked by multiple requests or events. It is serverless in nature, which means it hides away the infrastructure so that developers can focus on the development and not on the infrastructure and management. The process to use Google Cloud Run is straightforward as GCR automates most of the process. Job of developer is to simply tell where the container is, what memory, CPU are required, and click “Create”. The application can be accessed using the http endpoint.

References

- [1] Google, "Cloud Firestore," Google, [Online]. Available: <https://firebase.google.com/products/firestore>. [Accessed 15 October 2021].
- [2] Docker, "Developers Love Docker," Docker, [Online]. Available: <https://www.docker.com/>. [Accessed 14 October 2021].
- [3] Google, "Installing Cloud SDK," Google, [Online]. Available: <https://cloud.google.com/sdk/docs/install>. [Accessed 15 October 2021].
- [4] Google, "Container Registry," Google, [Online]. Available: <https://cloud.google.com/container-registry>. [Accessed 15 October 2021].
- [5] Google, "Cloud Run," Google, [Online]. Available: <https://cloud.google.com/run>. [Accessed 15 October 2021].