

## **Summary for the Task B**

In this project, I have developed three containerized microservices responsible for the backend logic of our application. Firestore is the chosen database, consisting of two collections: "Reg" for storing registration data like name, password, email, and location, and "state" for maintaining user state information including online status and timestamps.

Container #1 handles the registration process by receiving registration details from the frontend and storing them in the Firestore database. This microservice is responsible for saving the data in the "Reg" collection.

Container #2 is responsible for validating login information provided by users. It checks the values against the database records to determine if the login is valid. Upon successful login, the user's state is updated to "online" in Firestore, and this information is displayed on the front page.

Container #3 is responsible for extracting state information from the database, specifically identifying which users are currently online. It maintains the user session from login to logout, and the session expires once the user clicks the logout button. When a user logs out, the state item in the Firestore database is updated accordingly.

To complete the tasks, I have built Docker images for the three containers and pushed them to an artifact registry repository. Since Google Container Registry (GCR) is deprecated, I used an alternative artifact registry. Finally, I deployed the containers to Google Cloud Run.

In addition to the backend microservices, I created three simple web pages using React JS to interact with the application. To ensure the application's quality, I wrote test cases and performed thorough testing.

Throughout the development process, I extensively studied the documentation for Google Cloud Run, GCR/Artifact Registry, and Docker Container. This allowed me to understand and effectively utilize these technologies in the application.

### **Deployed Application Link**

<https://image4-w5u6k6uuza-uc.a.run.app/>

The screenshot shows the Firebase console interface for a project named "ServerlessAssignment2". The left sidebar includes navigation links for Project Overview, Firestore Database, Build, Release and monitor, Analytics, Engage, and All products. A "Customise your navigation" section allows users to focus their console experience by customising their navigation. The main area is titled "Cloud Firestore" and shows a "Data" tab. It displays a single collection named "Reg" with one document named "Reg" containing a field "state". Promotional banners for budget creation and app check configuration are visible at the top.

Fig 1: Unpopulated Firestore

The screenshot shows the same Firebase console interface as Fig 1, but the "Reg" collection now contains multiple documents. The "state" field for each document is set to "offline" and includes a timestamp. The URL in the browser bar is https://console.firebaseio.google.com/u/0/project/serverlessassignment2-391518.firebaseio/data/-2Fstate~2Fultimate@dal.ca.

Document ID	state	timestamp
max@dal.ca	offline	1688695729909
sr55161@dal.ca	offline	1688695729909
testing1@dal.ca	offline	1688695729909
testing2@dal.ca	offline	1688695729909
testing3@dal.ca	offline	1688695729909
testing4@dal.ca	offline	1688695729909
testing5@dal.ca	offline	1688695729909
testing6@dal.ca	offline	1688695729909
testing7@dal.ca	offline	1688695729909
testing8@dal.ca	offline	1688695729909
testing9@dal.ca	offline	1688695729909
testing@dal.ca	offline	1688695729909
testingForDocument@dal.ca	offline	1688695729909
ultimate@dal.ca	offline	1688695729909

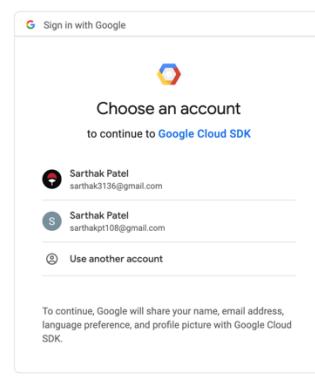
Fig 2: Populated Firestore

## Container 1

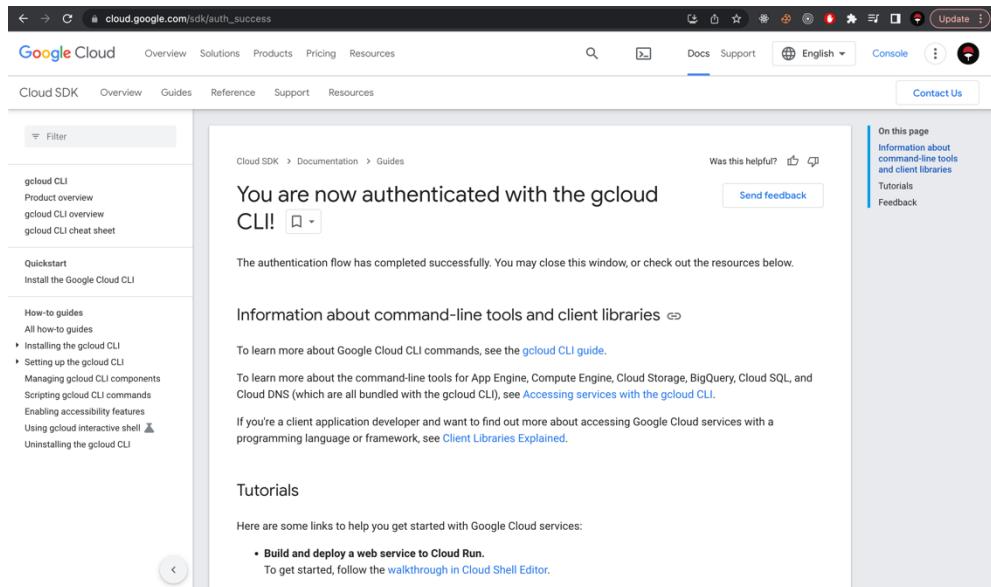
The GCloud auth login done to connect with the project in the Gmail account.

```
EXPLORER Container-1 > Dockerfile Container-1 > Dockerfile Container-3
Container-1 > Dockerfile
1 FROM node
2
3 WORKDIR /app
4
5 COPY package.json .
6
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 4000
12
13 CMD node app.js
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
sarthakpatel@garthaks-Air:~/Assignment-2-GCP$ cd Container-1
sarthakpatel@garthaks-Air:~/Assignment-2-GCP$ gcloud auth login
Your browser has been opened to visit:
https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=32555940559.apps.googleusercontent.com&redirect_uri=https%3A%2F%2Faccounts.googleapis.com%2Fauth%2Fcallback&state=32555940559%2Fwww.googleapis.com%2Fauth%2Faccounts.reauth&state=MQ9qyKsKvysDfheDcpu23aTLVw&access_type=offline&code_challenge=VpGrex
sarthakpatel@garthaks-Air:~/Assignment-2-GCP$
```

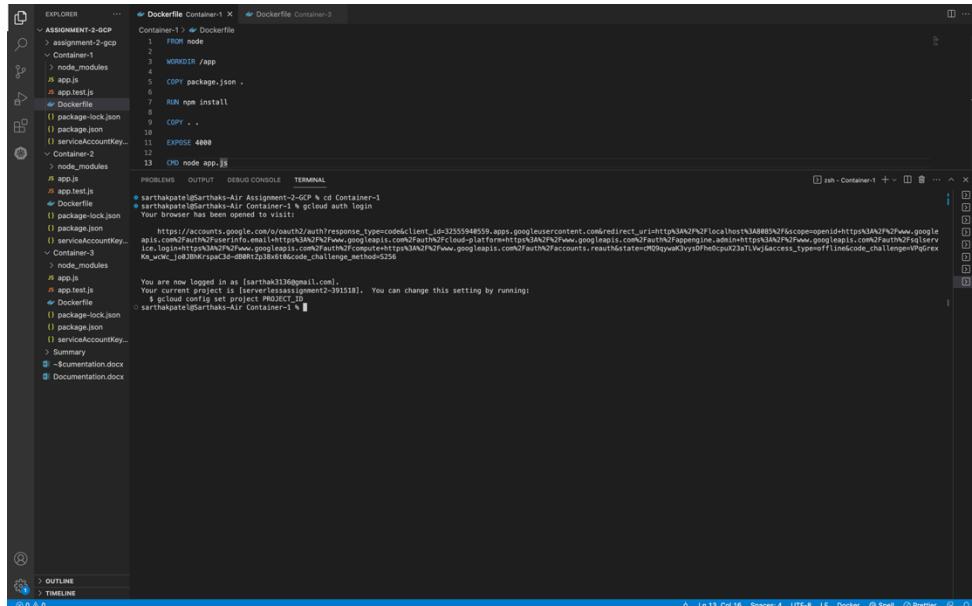
Fig 3: auth login to my Gmail account.



**Fig 4:** Selecting the google cloud account.



**Fig 5:** Successful authentication



**Fig 6:** Successful login seen in the terminal

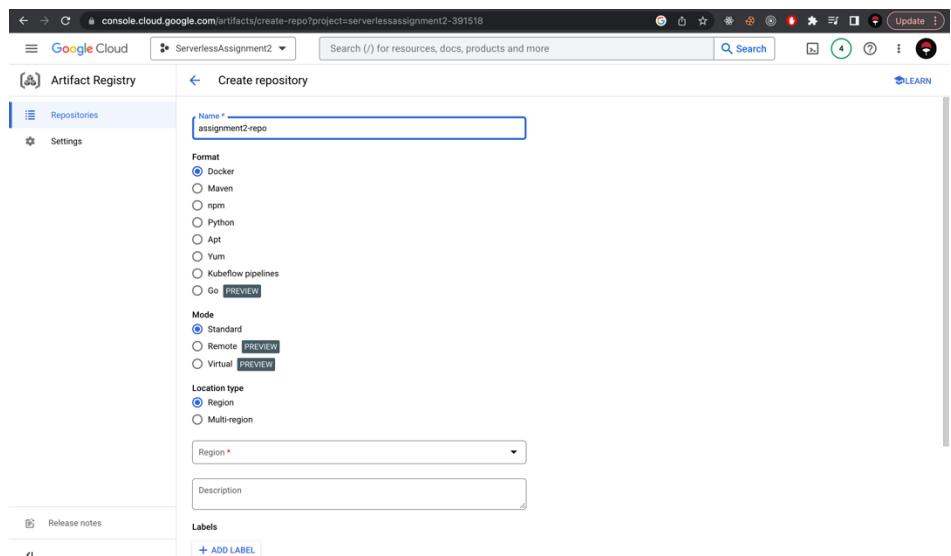


Fig 7: Creating an Artifact Registry repository

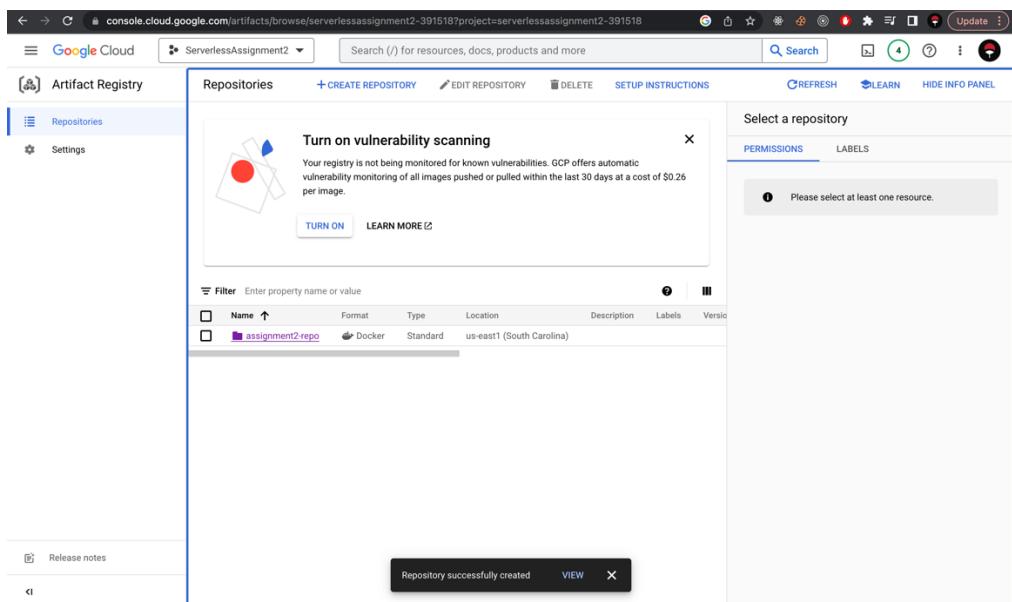
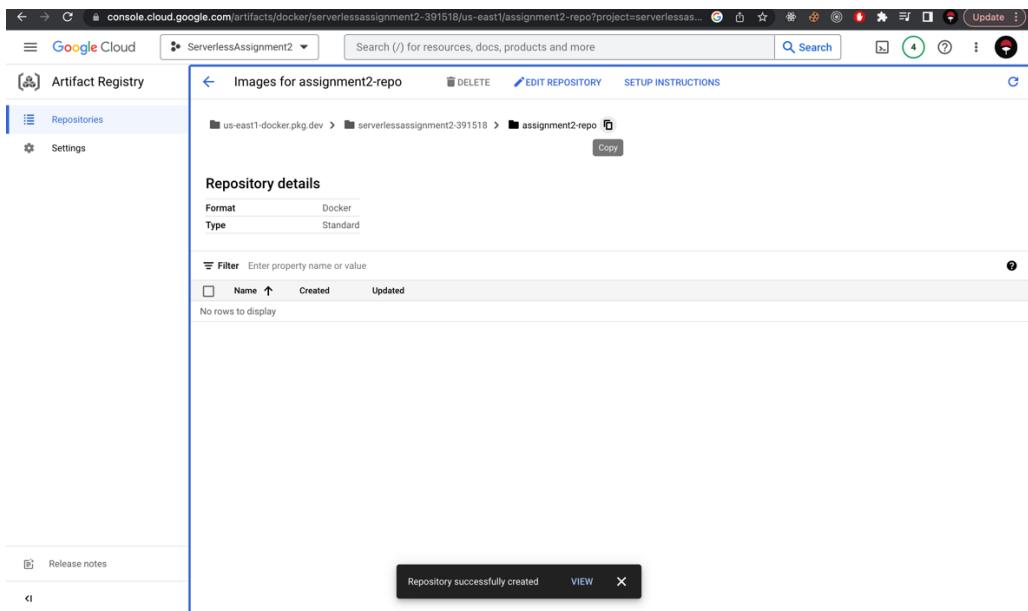


Fig 8: assignment2-repo created in artifact registry



**Fig 9:** Initially the repository is empty

The screenshot shows the VS Code interface with the Docker extension. The Explorer sidebar shows a project structure with files like 'Dockerfile', 'Container-1', 'Container-2', and 'Container-3'. The Dockerfile tab displays the following code:

```

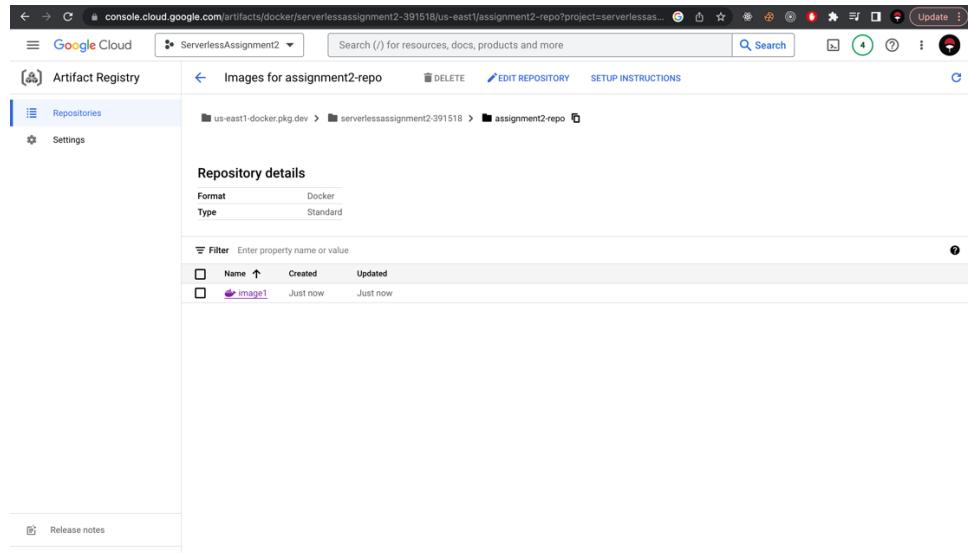
VERSION:2-0CP
FROM node
WORKDIR /app
COPY package.json .
RUN npm install
COPY .
EXPOSE 4000
CMD node app.js

```

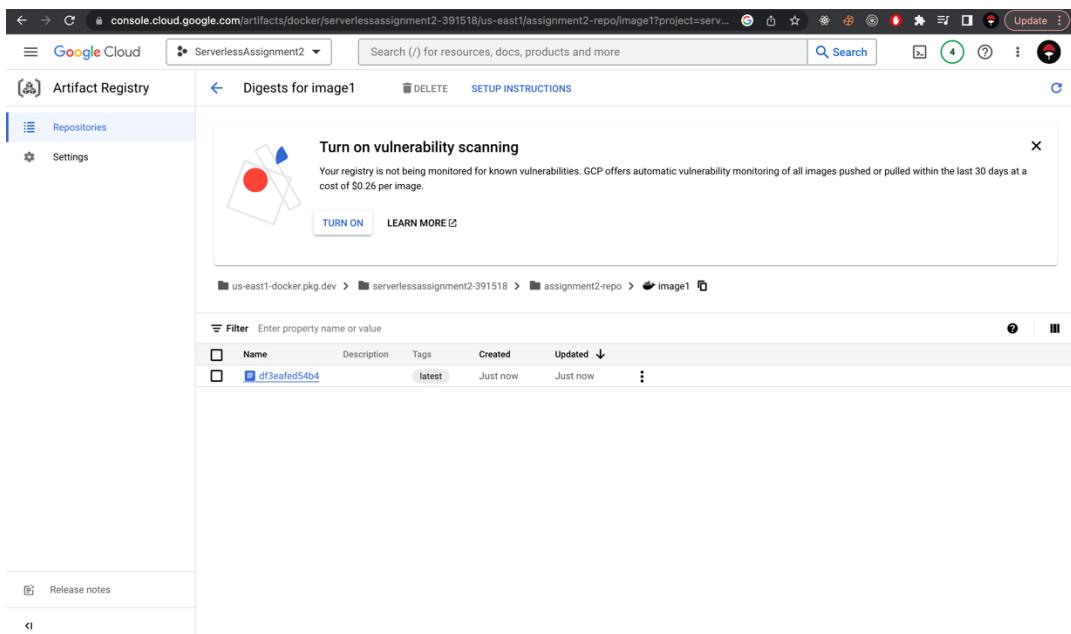
The Terminal tab shows the command being run: 'gcloud auth login'. The output of the command includes a URL for OAuth2 authorization and a code challenge method. The status bar at the bottom indicates the file is 15 lines long, has 16 columns, and is in UTF-8 encoding.

**Fig 10:** Building and submitting the docker image from the docker file to registry.

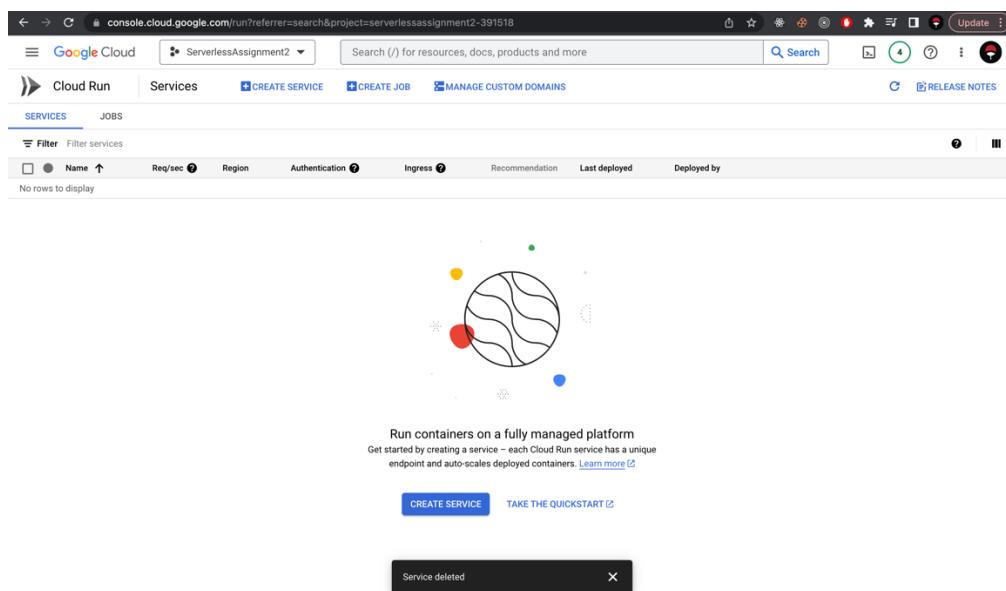
**Fig 11:** Image for container 1 code pushed in repo.



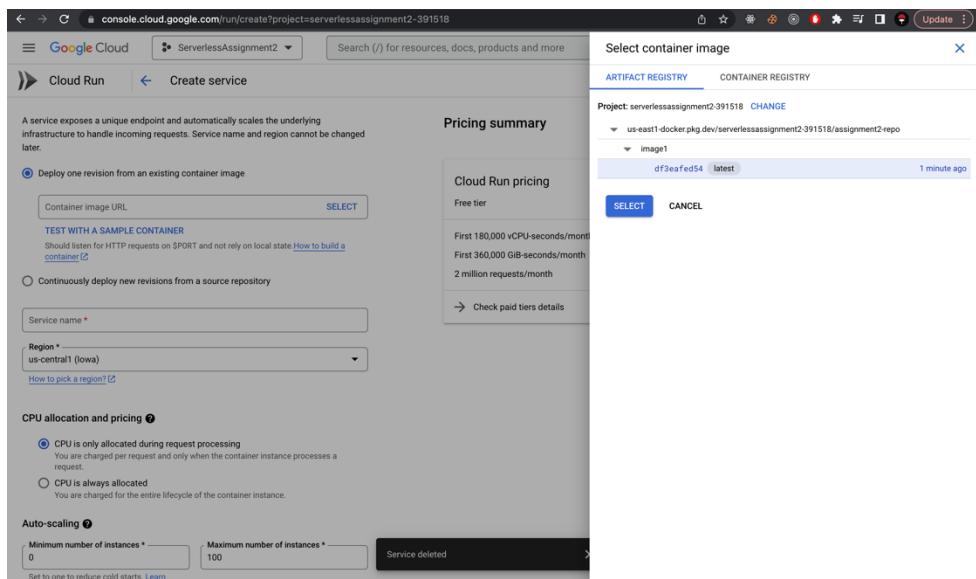
**Fig 12:** Image1 folder seen in artifact repository.



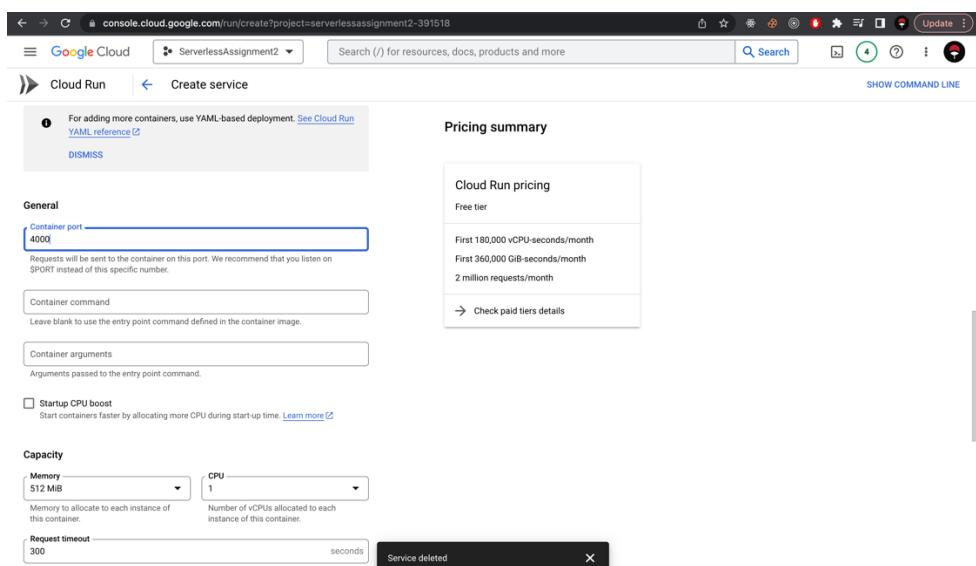
**Fig 13:** Docker image for our container



**Fig 14:** Empty Cloud Run



**Fig 15:** Selecting the image we want to deploy in cloud run.



**Fig 16:** Specifying the port as same as my node js app.

The screenshot shows the Google Cloud Run Services page. At the top, there are tabs for 'Cloud Run', 'Services', 'CREATE SERVICE', 'CREATE JOB', 'MANAGE CUSTOM DOMAINS', 'COPY', 'DELETE', 'TAGS', 'RELEASE NOTES', and 'HIDE INFO PANEL'. A search bar is present at the top right. Below the tabs, there are two main sections: 'SERVICES' and 'JOBS'. The 'SERVICES' section has a header with columns: Name, Req/sec, Region, Authentication, Ingress, Recommendation, Last deployed, Deployed by, PERMISSIONS, and LABELS. A message 'No services selected' is displayed. Under the 'SERVICES' section, there is a table with one row for 'image1'. The 'JOBS' section is currently empty. A note at the bottom right says 'Please select at least one resource.'

Fig 17: Image 1 cloud run

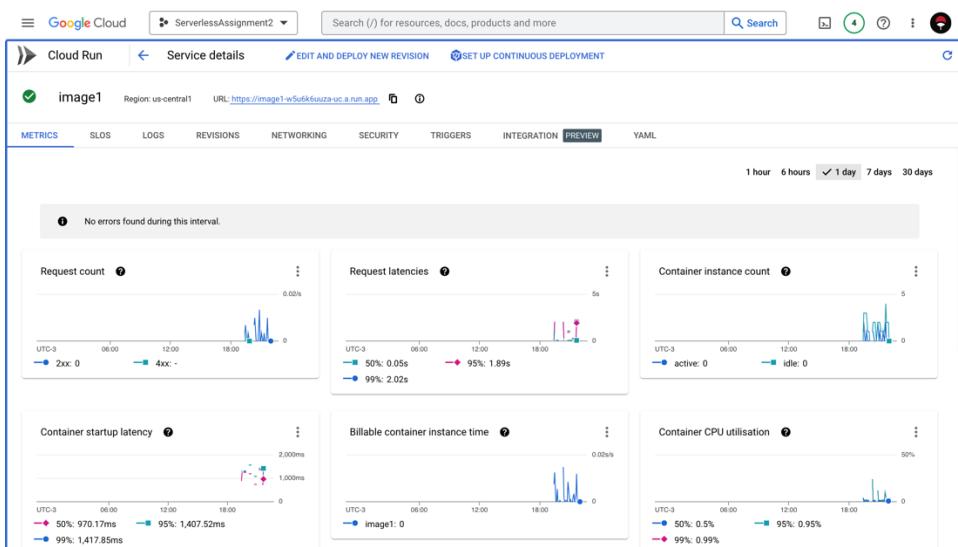
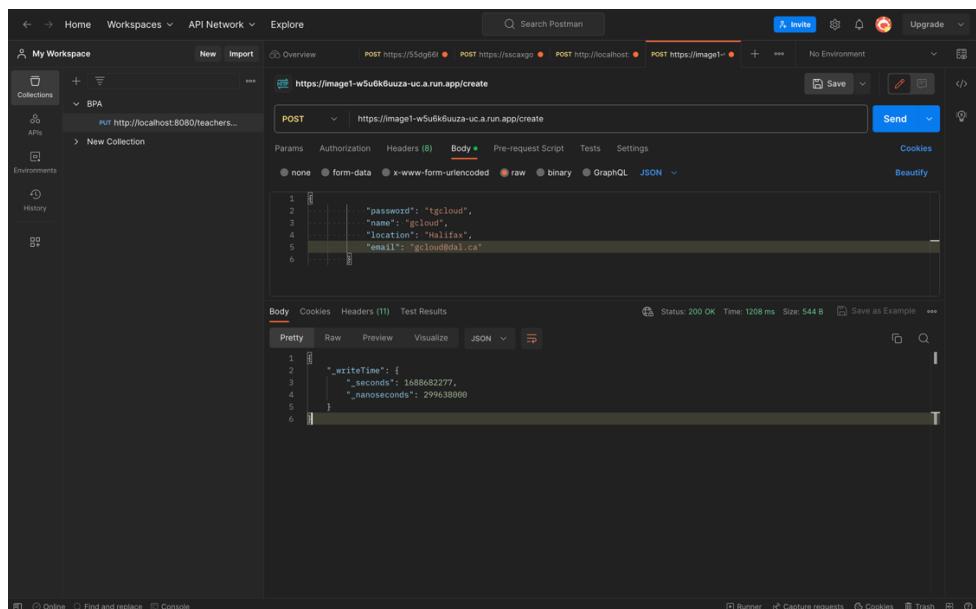


Fig 18: Successful deployment and deployment link



**Fig 19:** Testing the deployment link for the REST services in my container-1

Document ID	Fields
gcloud@dal.ca	
testing1@dal.ca	
testing2@dal.ca	
testing3@dal.ca	
testing4@dal.ca	
testing5@dal.ca	
testing6@dal.ca	
testing7@dal.ca	state: "Online", timeStamp: 168667348002
testing8@dal.ca	
testing9@dal.ca	

**Fig 20:** User getting created

The screenshot shows the VS Code interface with the Explorer, Outline, and Timeline panes visible. The Explorer pane shows a project structure for 'assignment-2-gcp' with files like App.js, index.js, Login.js, Navbar.js, Profile.js, reportWebVitals.js, setupTests.js, and Signup.js. The Signup.js file is open in the editor, displaying code for a React component that handles user sign-up or creation via a POST request to 'http://localhost:4000/create'. The code uses useState for state management and axios for making the API call. The terminal pane shows build logs for Container-1, including commands like 'npm run build' and 'node dist/index.js', along with deployment logs for Container-1 and Container-2.

**Fig 21:** Original REST API for Signup or create

This screenshot is identical to Fig 21, showing the same project structure and code for Signup.js. However, the terminal pane now displays deployment logs for Container-1 and Container-2, indicating that the application has been successfully deployed to Google Cloud Run. The logs show the command 'r.pkg.dev/serverlessassignment2-391518\_assignment2-repo/images/:1 more' followed by 'SUCCESS'.

**Fig 22:** Replacing the original API with the deployed one.

## Container 2

The screenshot shows the VS Code interface with the Dockerfile for Container 2 open. The Dockerfile contains the following code:

```

FROM node
WORKDIR /app
COPY package.json .
RUN npm install
EXPOSE 2000
CMD node app.js

```

The terminal shows the build process:

```

2f6f525410: Preparing
a723590105: Preparing
bae7700cd3e: Preparing
2f6f525410: Waiting
98a2a25288ff: Waiting
937c2864d0: Waiting
28218ec0d980: Waiting
28218ec0d980: Waiting
a723590105: Waiting
6158147931: Waiting
fd5ad48d1a: Pushed
bae7700cd3e: Pushed
98a2a25288ff: Pushed
bae7700cd3e: Pushed
937c2864d0: Pushed
937c2864d0: Pushed
6158147931: Pushed
a723590105: Pushed
bae7700cd3e: Pushed
28218ec0d980: Pushed
latest: digest: sha256:df3eafed54b4e7ef157f8ae0ac948a3354de985641a12d46844f90ecd58324 size: 2843

```

The terminal also shows the command used to build the image:

```

r-pkg.dev/serverlessassignment2-391518/build/repo/image2 (% more Dockerfile)
sarthakapate@Sarthaks-Air:~/Assignment-2-GCP$ cd Container-2
sarthakapate@Sarthaks-Air:~/Assignment-2-GCP$ gcloud builds submit --tag us-east1-docker.pkg.dev/serverlessassignment2-391518/assignment2-repo/image2

```

**Fig 23:** Building and pushing the image of container 2 to the artifact registry repo.

This screenshot is identical to Fig 23, showing the Dockerfile for Container 2 and its successful build log. The logs show the same sequence of events and commands as in Fig 23, indicating a successful build and push to the artifact registry.

**Fig 24:** Image successfully pushed

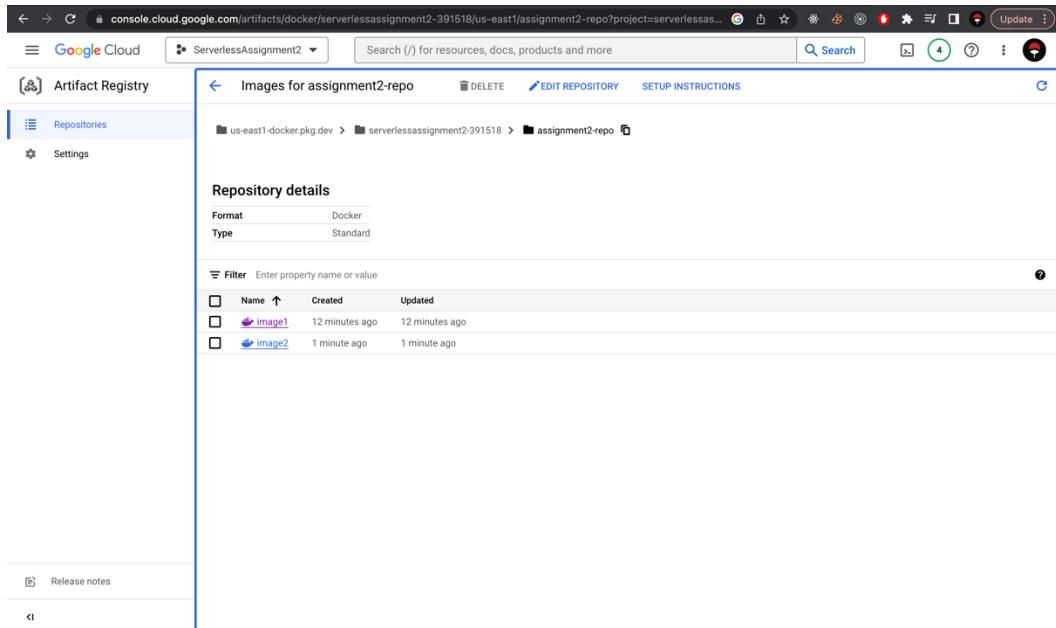


Fig 25: Image 2 for the container 2

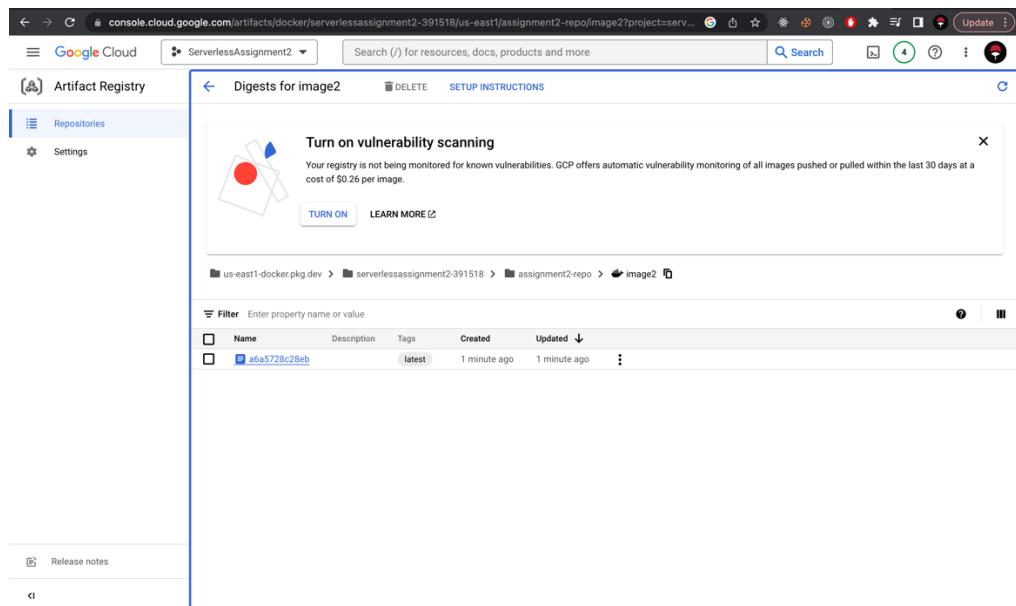
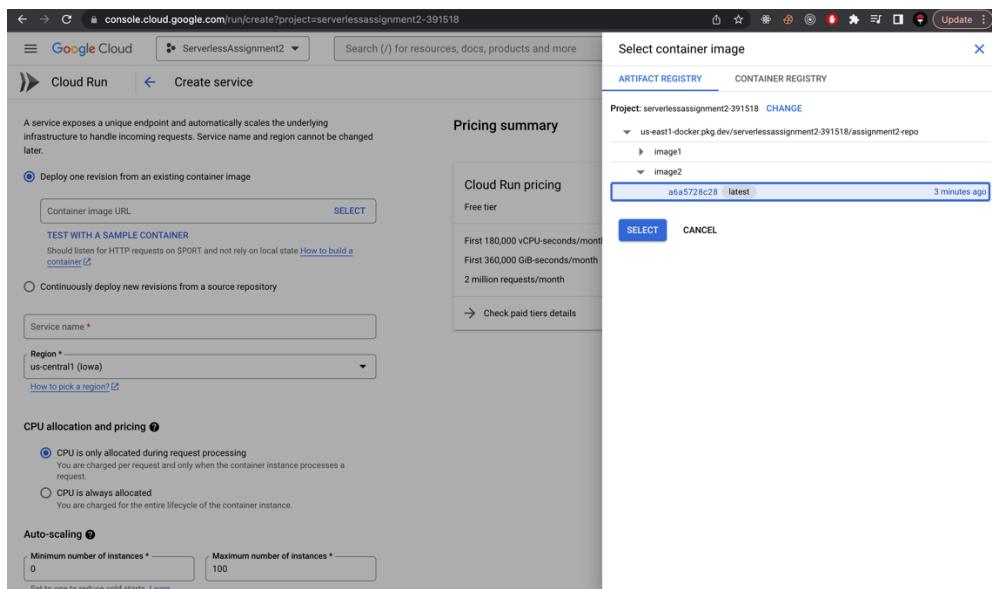
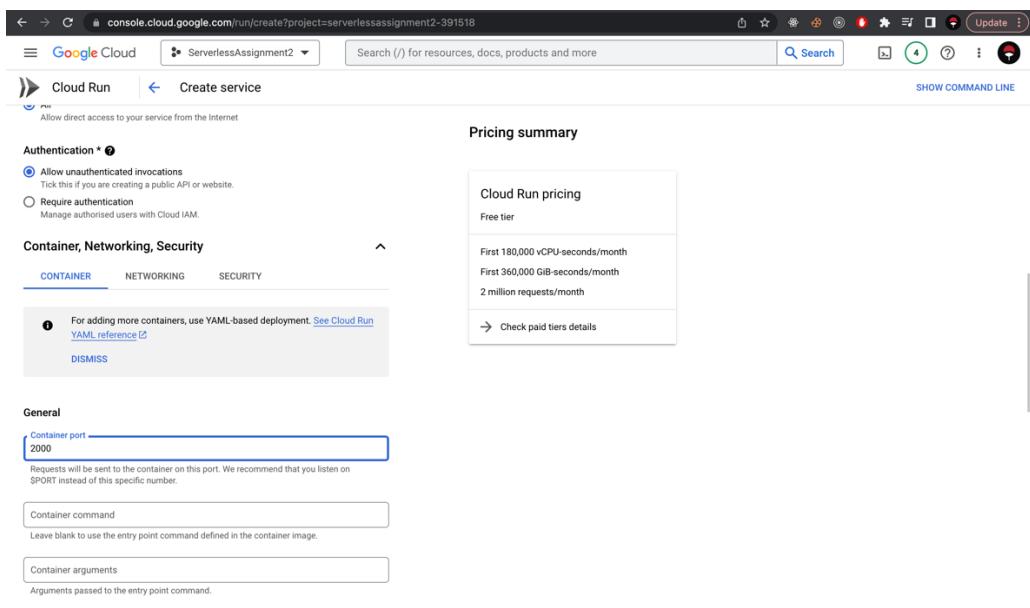


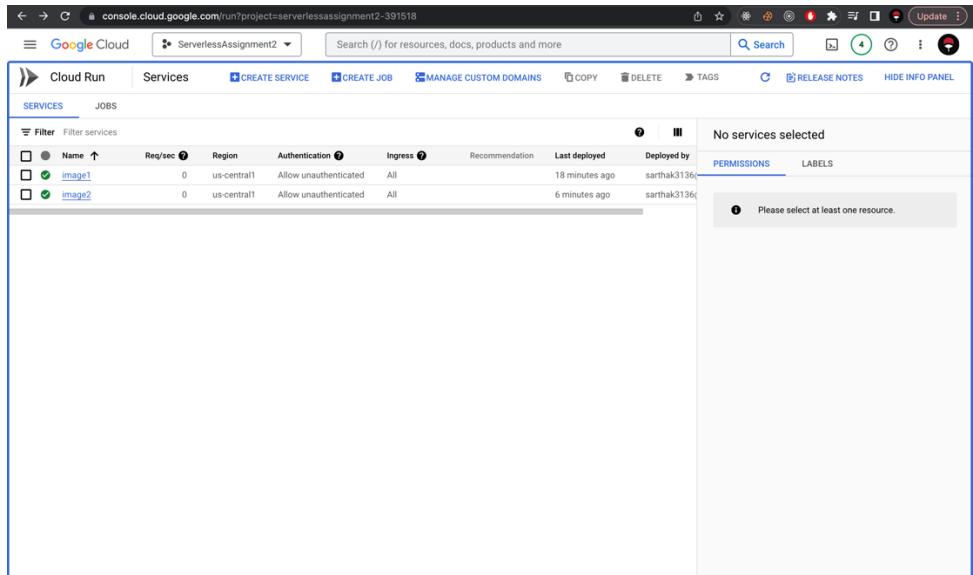
Fig 26: Image pushed in Image 2 repo.



**Fig 27:** Selecting image from image 2 as deployment.



**Fig 28:** Port same as container 2 port of app.js



**Fig 29:** Cloud run for images 1 and 2.

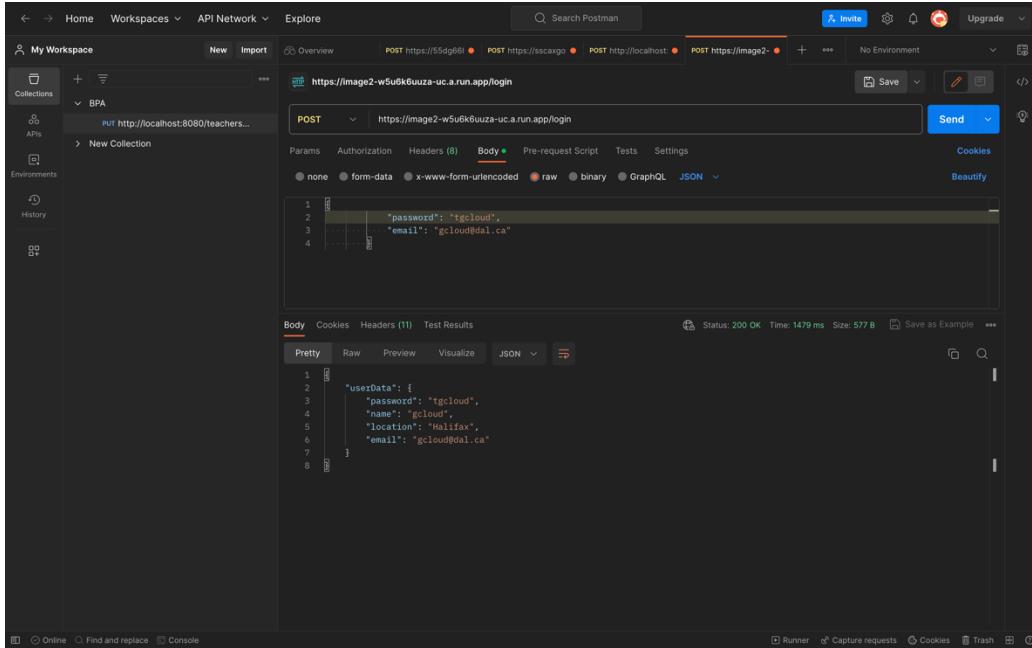
The screenshot shows the Google Cloud Run Service details page for 'image2'. At the top, there are buttons for 'EDIT AND DEPLOY NEW REVISION' and 'SET UP CONTINUOUS DEPLOYMENT'. Below this is a table with columns: METRICS, SLOs, LOGS, REVISIONS, NETWORKING, SECURITY, TRIGGERS, INTEGRATION, PREVIEW, and YAML. The 'REVISIONS' tab is selected, showing a list of revisions:

Name	Traffic	Deployed	Revision URLs (tags)	Actions
image2-00001-2kt	100% (to latest)	Just now	+	⋮

To the right, detailed configuration for the 'image2-00001-2kt' revision is shown under 'CONTAINERS' tab:

- General**
  - CPU allocation: CPU is only allocated during request processing
  - Startup CPU boost: Disabled
  - Concurrency: 80
  - Request timeout: 300 seconds
  - Execution environment: First generation (default)
- Auto-scaling**
  - Max. instances: 10
- Image URL**: us-east1-docker.pkg.dev/serverlessassignment2-391...
- Port**: 2000
- Build**: (no build information available)
- Source**: (no source information available)
- Command and arguments**: (container endpoint)
- CPU limit**: 1
- Memory limit**: 512MB

**Fig 30:** Cloud run for image 2 with the deployment link



**Fig 31:** Testing the login REST API with the deployed URL.

```

Dockerfile Container-1
# Stage 1: Build
FROM node:14 AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 2000
CMD ["node", "app.js"]

# Stage 2: Run
FROM nginx:1.19.0-alpine
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

```

Login.js
const express = require('express');
const app = express();
const port = 2000;
const cors = require('cors');

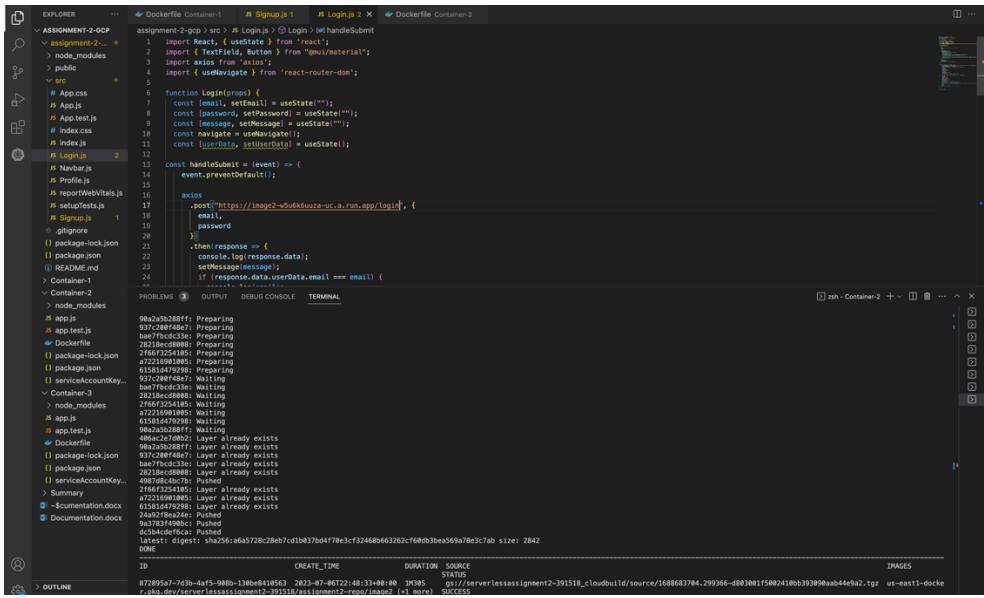
app.use(cors());
app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.post('/login', (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    return res.status(400).send('Email and password are required');
  }
  // Mock user data
  const user = {
    id: 1,
    name: 'tgcloud',
    email: 'gcloud@dal.ca',
    password: 'tgcloud'
  };
  if (user.email === email && user.password === password) {
    // Set session storage
    const storedUser = JSON.stringify(user);
    const sessionStorage = sessionStorage.getItem('userData');
    if (!sessionStorage) {
      sessionStorage.setItem('userData', storedUser);
    } else {
      sessionStorage.removeItem('userData');
      sessionStorage.setItem('userData', storedUser);
    }
    // Redirect to profile page
    res.redirect('/profile');
  } else {
    res.status(401).send('Invalid credentials');
  }
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});

```

**Fig 32:** Old REST API for the login



**Fig 33:** Updated REST API for the login

## Important Methods

```

app.post("/login", async (req, res) => {
  try{
    const email = req.body.email;
    const password = req.body.password;

    const checkUser = await usersDb.doc(email).get();
    console.log(checkUser);

    if(checkUser.exists){
      const userData = checkUser.data();

      if(userData.password === password){
        const response = await stateDb.doc(email).set({
          state: "Online",
          timeStamp: Date.now()
        });
        res.json({ userData: userData });
      }
      else{
        res.status(401).send("Incorrect password");
      }
    }
    else{
      res.status(404).send("User not found");
    }
  }
}

```

```
        catch(error){
            console.log(error);
        }
    });
}


```

```
app.post("/create", async (req, res)=> {
    try{
        const id = req.body.email;
        console.log(req.body);
        const user = {
            name: req.body.name,
            email: req.body.email,
            password: req.body.password,
            location: req.body.location
        }
        const response = await usersDb.doc(id).set(user);
        await stateDb.doc(id).set({
            state: "Offline",
            timeStamp: Date.now()
        })
        res.send(response);
        console.log(response);
    }
    catch(error){
        res.send(error)
    }
}
);


```

```
app.post("/logout", async (req, res)=> {
    try {
        const email = req.body.email;
        await stateDb.doc(email).set({ state: 'offline', timestamp: Date.now() });
        res.send({ status: 'Logout successful' });
    } catch (error) {
        res.status(500).send({ error: 'An error occurred' });
    }
};

app.get("/online-users", async (req, res)=> {
    try {
        const onlineEmails = [];
        const onlineUsers = [];

        const data = await stateDb.where("state", "==", "Online").get();

        data.forEach((doc) => {
            onlineEmails.push(doc._ref._path.segments[1]);
        });
    }
});
```

```
    });

    await Promise.all(
      onlineEmails.map(async (email) => {
        const userDataOnEmail = await usersDb.where("email", "==", email).get();

        userDataOnEmail.forEach((doc) => {
          const userData = doc.data();
          onlineUsers.push(userData);
        });
      })
    );
  }

  res.send({ onlineUsers: onlineUsers });
} catch (error) {
  console.log(error);
  res.status(500).send({ error: "An error occurred." });
}

});
```

```
useEffect(()=> {
  axios.get("https://image3-w5u6k6uuza-uc.a.run.app/online-users").then(response=> {
    console.log(response);
    setOnlineUsers(response.data.onlineUsers);
  }).catch(error => {
    console.log(error)
  });
}, []);

window.addEventListener("unload", (event) => {
  axios.post("https://image3-w5u6k6uuza-uc.a.run.app/logout", { email
}).then(response => {
  console.log(response);
  localStorage.removeItem("userData")
  navigate("/login")
}).catch(error => {
  console.log(error)
})
});
```

```
const handleSubmit = (event) => {
  event.preventDefault();

  axios
    .post("https://image2-w5u6k6uuza-uc.a.run.app/login", {
```

```
        email,
        password
    })
    .then(response => {
        console.log(response.data);
        setMessage(message);
        if (response.data.userData.email === email) {
            console.log(email);
            console.log(response.data);
            sessionStorage.setItem('userData', JSON.stringify(response.data.userData));
            const storedUser = JSON.parse(sessionStorage.getItem('userData'));
            console.log(storedUser)
            // navigate('/profile', { state: { email: response.data.userData.email,
name: response.data.userData.name } });
            navigate('/profile', {state: { email: storedUser.email, name:
storedUser.name, userData: storedUser.userData}})
            //navigate("/profile")
        }
    })
    .catch(error => {
        console.log(error);
        setMessage("Error occurred during login");
    });
};

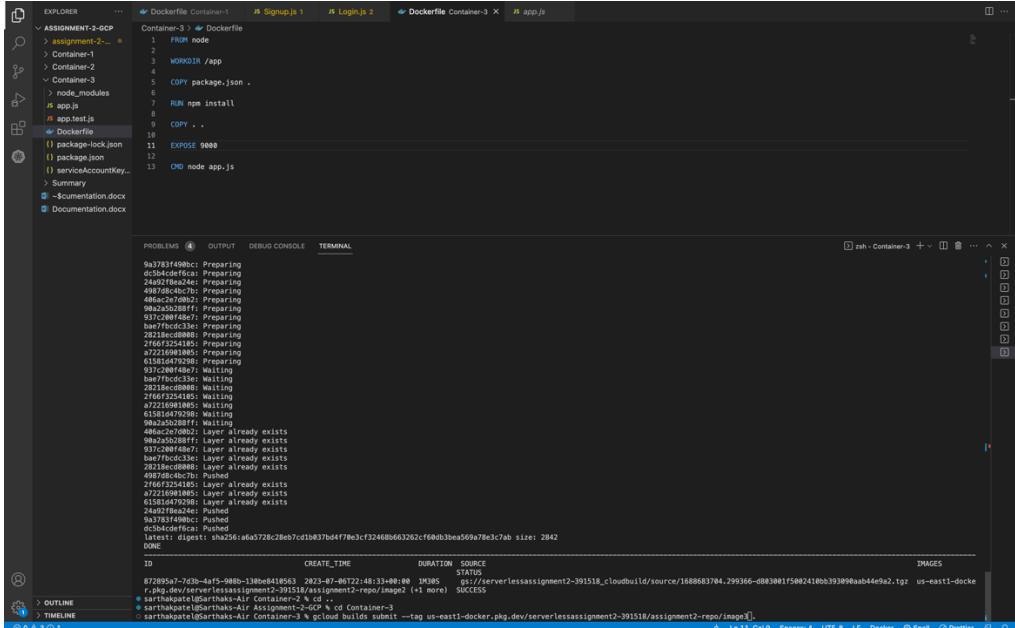

```

```
const handleSubmit =(event) => {

try {
    axios.post("https://image1-w5u6k6uuza-uc.a.run.app/create", {
        name,
        email,
        password,
        location
    }).then(
        navigate('/login')
    );
} catch (error) {
    console.error(error);
}
};


```

## Container 3



**Fig 34:** Building and pushing container 3 image in artifact repo

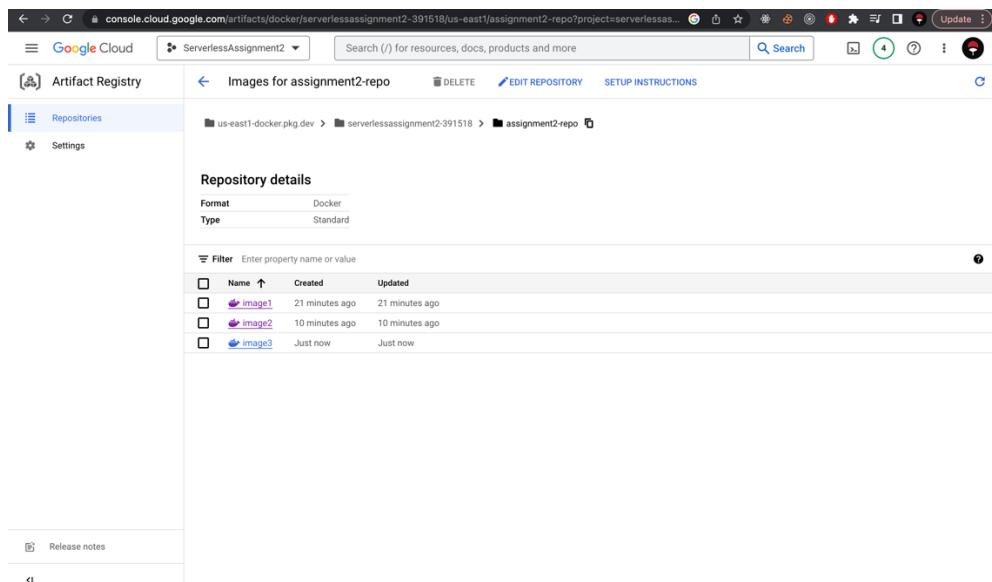
```

Pushing us-east1-docker.pkg.dev/serverlessassignment2-391518/assignment2-repo/app
The push refers to repository [us-east1-docker.pkg.dev/serverlessassignment2-391518/assignment2-repo/app]
f02f2e448f22: Preparing
22380bd1c6c62: Preparing
486ac2790821: Preparing
98a25a28ff: Preparing
baef7b0cdcc35c: Preparing
28218ec08080: Preparing
28218ec08080: Preparing
a7216901085: Preparing
937298f4d67: Waiting
baef7b0cdcc35c: Preparing
2760f3254105: Waiting
61581d792398: Preparing
98a25a28ff: Waiting
98a25a28ff: Layer already exists
98a25a28ff: Layer already exists
937298f4d67: Layer already exists
28218ec08080: Layer already exists
22380bd1c6c62: Pushed
2760f3254105: Layer already exists
61581d792398: Layer already exists
f02f2e448f22: Pushed
latest: digest: sha256:768b303c7cff02646fe3801379c10722e39d163e57891ee06e4c842c1e281cd size: 2842
DONE

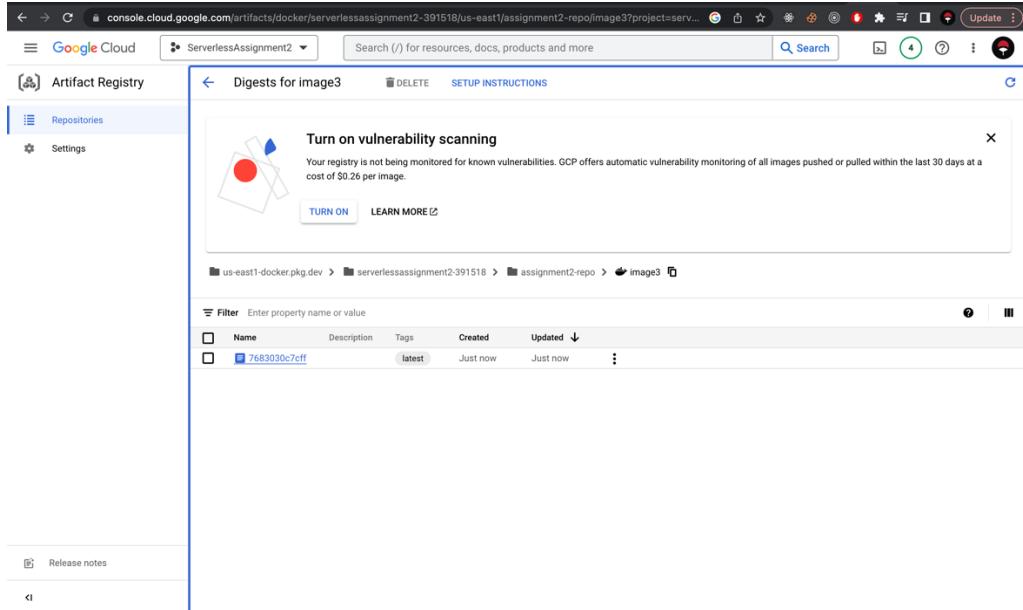
```

The terminal output shows the process of pushing a Docker image named 'app' from a local repository to a Google Cloud Artifact Registry. The command used was 'docker push us-east1-docker.pkg.dev/serverlessassignment2-391518/assignment2-repo/app'. The output details the layers being pushed, including their IDs and creation times, and concludes with the message 'latest: digest: sha256:768b303c7cff02646fe3801379c10722e39d163e57891ee06e4c842c1e281cd size: 2842'.

**Fig 35:** Successfully pushed in artifact repo



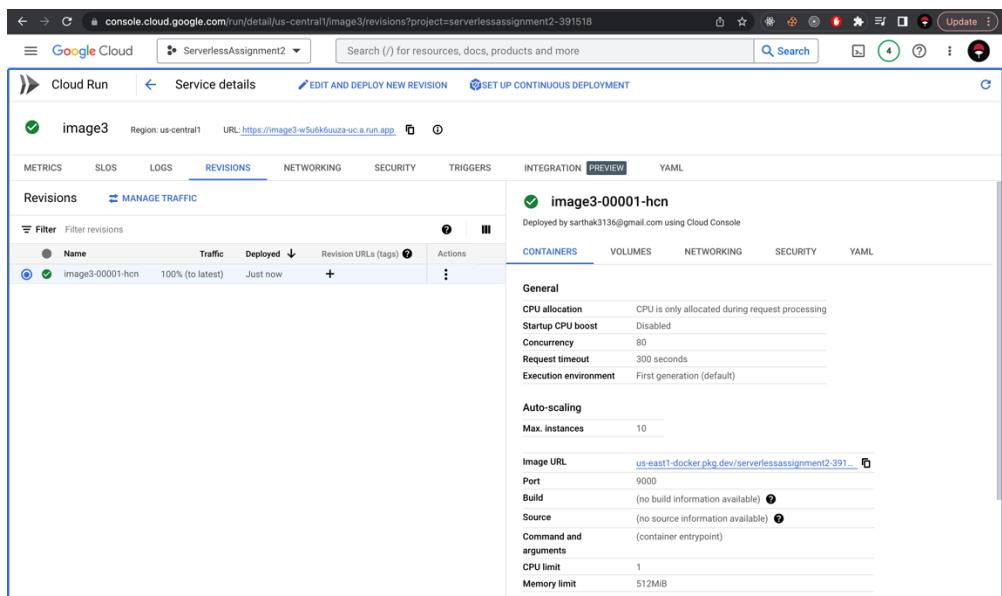
**Fig 36:** Images 3 folder created in artifact repo



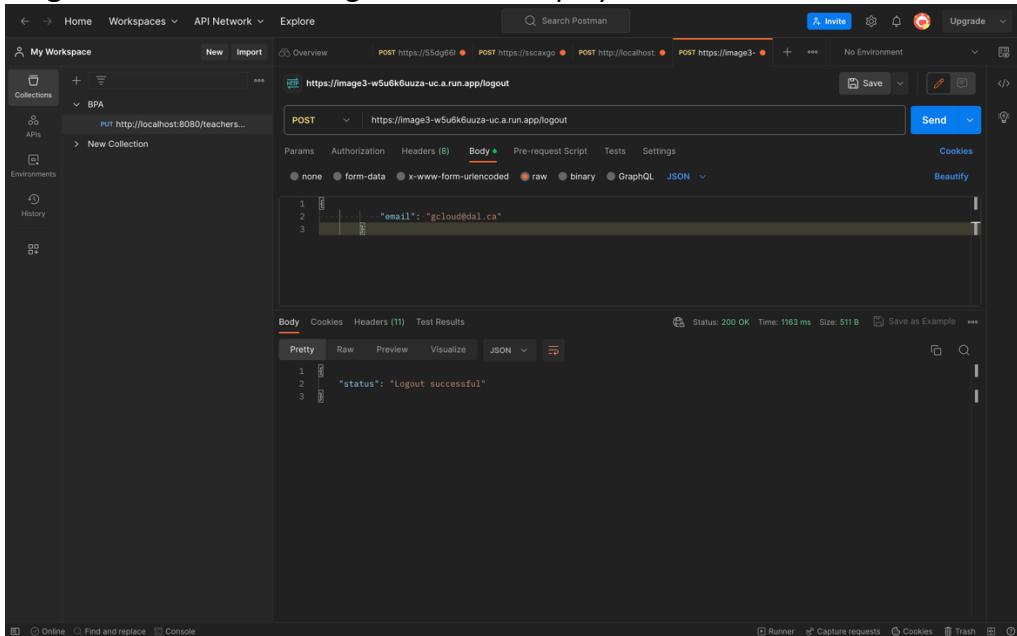
**Fig 37:** Image for the container 3

The screenshot shows the Google Cloud Cloud Run 'Create service' page. The main form includes fields for 'Container image URL' (with a 'SELECT' button), 'Service name' (empty), 'Region' (set to 'us-central1 (Iowa)'), and 'Auto-scaling' (with 'Minimum number of instances' at 0 and 'Maximum number of instances' at 100). To the right, a modal titled 'Select container image' lists images from the project 'serverlessassignment2-391518': 'image1', 'image2', and 'image3'. Under 'image3', the item '7683030c7cff latest' is selected, with a timestamp of '1 minute ago'. There are 'SELECT' and 'CANCEL' buttons at the bottom of the modal.

**Fig 38:** Creatin a cloud run deployment for image 3.



**Fig 39:** Cloud run for image 3 with the deployment link for the microservice



**Fig 40:** Testing the logout API for the deployed microservice

The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation bar includes 'Project Overview', 'Cloud Firestore', and other project-related links. The main area displays a collection named 'state'. Inside this collection, there is a document with the key 'gcloud@dal.ca'. The data within this document is as follows:

```

state: "offline"
timestamp: 1688684597731
  
```

Below this, a list of other documents under the 'state' collection is shown:

- testing1@dal.ca
- testing2@dal.ca
- testing3@dal.ca
- testing4@dal.ca
- testing5@dal.ca
- testing6@dal.ca
- testing7@dal.ca
- testing8@dal.ca
- testing9@dal.ca

**Fig 41:** User getting logged out

The screenshot shows the Postman application interface. The left sidebar lists 'My Workspace' with collections like 'BPA' and 'APIs'. The main workspace shows a collection named 'BPA' with a single item: 'New Collection'. A specific request is selected for viewing, with the URL being <https://image3-w5u6k6uza-uc.a.run.app/online-users>. The request method is 'GET'. The response body is displayed in JSON format:

```

{
  "onlineUsers": [
    {
      "password": "testing3",
      "name": "testing3",
      "location": "Tokyo",
      "email": "testing3@dal.ca"
    },
    {
      "password": "testing7",
      "name": "testing7",
      "location": "Osaka",
      "email": "testing7@dal.ca"
    },
    {
      "password": "testing8",
      "name": "testing8",
      "location": "Toronto",
      "email": "testing8@dal.ca"
    }
  ]
}
  
```

The status bar at the bottom indicates a 200 OK response with a time of 337 ms and a size of 851 B.

**Fig 42:** Testing the online-users API for the deployed microservice

The terminal window displays the following log output:

```
98a2a5b28ff: Waiting
98a2a5b28ff: Layer already exists
98a2a5b28ff: Layer already exists
937208f48c71: Layer already exists
bae7f0cdcc33: Layer already exists
20230617d656: Layer already exists
0786dbd67e61: Pushed
2236015c6262: Pushed
2365f13a3333: Layer already exists
a72216961085: Layer already exists
615814793985: Layer already exists
f2c2a2a2a2a2: Pushed
da881ccb1319: Pushed
testset digest: sha256:676889367cf10248fbef001379c18723a99d16a57991e0de4c4021c16281d size: 2842
Date: Sun, 10 Jul 2023 10:45:22 +0000
ID          CREATE_TIME      DURATION SOURCE    STATUS          IMAGES
6d1f1151-c049-4765-a553-ef16c0272023-07-10T07:58:13+00:00 1m27s  gcr://serverlessassignment-2-391518._cloudbuild/source/1688804303.42615-9a6f543ab2e45951b2791b5ff7a:tgr us-east-1 docker
: pkg.dev/serverlessassignment-2-391518._cloudbuild/repo1Image3 [1+ more] SUCCESS
:: sarthakpate@Sarthaks-Air Container-3 %
```

**Fig 43:** Old APIs in React

The screenshot shows a browser-based code editor interface with several tabs open. The tabs include 'EXPLORER', 'CONTAINER-1', 'CONTAINER-2', 'CONTAINER-3', and 'Profile.js'. The 'Profile.js' tab is active, displaying a large block of JavaScript code. The code includes imports from 'react', 'react-dom', 'react-router-dom', and 'material-ui'. It defines a 'Profile' component with a state object containing 'name', 'email', and 'onlineUsers'. The component uses 'useEffect' to handle user login and logout, making API calls to 'https://image3-w5udk6uza-uc.a.run.app/online-users' and 'https://image3-w5udk6uza-uc.a.run.app/logout'. It also handles local storage removal and navigation. The code ends with a 'const StyledTableCell = styled(TableCell)(({ theme }) => ({ ...)).

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

zsh - Container-3

LOCATIONS

IMAGES

ID	CREATE_TIME	DURATION	SOURCE	STATUS
0f7fbef0-21-11ed-937-7770e0c4b4d4	2023-07-07T08:21:00+00:00	14m0s	git://server/lessassignment2-391518_cloudbuild/source/1688806450_911966-88894e4082344f5bd5e3580561d43a5.tgz	us-east-1
r-pkg/devServer/lessassignment2-391518_cdn/zipfile/repo/image3-w5udk6uza-uc.a				
sarthakpathak@SARTHAK-ALI:Container-3 %				

**Fig 44:** Updated REST API in React

## Deploying the React Application

```

version: '3'
services:
  app:
    build: ./assignment-2-gcp
    ports:
      - "3000:3000"

```

```

FROM node:14 as react-build
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 3000
RUN npm run build
# server environment
FROM nginx:alpine
COPY nginx.conf /etc/nginx/conf.d/configfile.template
COPY --from=react-build /app/build /usr/share/nginx/html
ENV PORT 3000
ENV HOST 0.0.0.0
EXPOSE 3000
CMD sh -c "envsubst '$PORT' < /etc/nginx/conf.d/configfile.template > /etc/nginx/conf.d/default.conf && nginx -g 'daemon off;'"

```

**Fig 45:** Docker file for the frontend REACT deployment.

```

gcloud builds submit --tag us-east1-docker.pkg.dev/serverlessassignment2-391518/assignment2-repo/image4 .

```

```

Uploading [...] to [git@github.com:serverlessassignment2-391518-cloudbuild/source/1688689392-24576-5b69959217a980:0b5a2d564e42f90f.tgz]
Created https://cloudbuild.googleapis.com/v1/projects/serverlessassignment2-391518/locations/global/builds/1bdc9fd-e8ec-a69-6754fe43e1el.
Logs are available at https://console.cloud.google.com/CloudBuild/builds/1bdc9fd-e8ec-a69-6754fe43e1el?project=serverlessassignment2-391518&tab=logs.

```

```

Step 1/14 : FROM node:14 as react-build
Step 1/14 : WORKDIR /app
Step 1/14 : COPY package.json .
Step 1/14 : RUN npm install
Step 1/14 : COPY . .
Step 1/14 : EXPOSE 3000
Step 1/14 : RUN npm run build
Step 1/14 : # server environment
Step 1/14 : FROM nginx:alpine
Step 1/14 : COPY nginx.conf /etc/nginx/conf.d/configfile.template
Step 1/14 : COPY --from=react-build /app/build /usr/share/nginx/html
Step 1/14 : ENV PORT 3000
Step 1/14 : ENV HOST 0.0.0.0
Step 1/14 : EXPOSE 3000
Step 1/14 : CMD sh -c "envsubst '$PORT' < /etc/nginx/conf.d/configfile.template > /etc/nginx/conf.d/default.conf && nginx -g 'daemon off;'"

```

**Fig 46:** Building and Pushing image for the frontend React

The screenshot shows a terminal window titled "zsh - Container-3" displaying the output of a Docker build command. The logs indicate the creation of multiple layers from a Dockerfile, with some layers being skipped due to already existing files. The Dockerfile itself is visible at the top of the terminal window.

```
assignment-2-gcp
  assignment-2-gcp
    build
      node_modules
        src
          App.css
          App.js
          App.test.js
          index.css
          index.js
          Login.js
          Navbar.js
          Profile.js
          reportWebVitals.js
          setupTests.js
          Signup.js
          ignore
        Dockerfile
        nginx.conf
        package-lock.json
        package.json
        README.md
      Container-1
      Container-2
      Container-3
        build
          node_modules
            app.js
            app.test.js
          Dockerfile
          package-lock.json
          package.json
          serviceaccountKey.json
        Summary
        documentation.docx
        Documentation.docx

assignment-2-gcp
assignment-2-gcp> ↵ Dockerfile
1  FROM node:14 as react-build
2
3   WORKDIR /app
4
5   COPY package*.json .
6
7   RUN npm install
8
9   COPY . .
10
11  EXPOSE 3000
12
13  RUN npm run build
14
15  # server environment
16  FROM nginx:alpine
17  COPY nginx.conf /etc/nginx/conf.d/configfile.template
18
19  COPY --from=react-build /app/build /usr/share/nginx/html
20
21  ENV PORT 3000
22  ENV HOST 0.0.0.0
23  EXPOSE 3000
24  CMD sh -c "env PORT=$PORT < /etc/nginx/conf.d/configfile.template > /etc/nginx/conf.d/default.conf && nginx -g 'daemon off;'"

PROBLEMS ⚠️ OUTPUT DEBUG CONSOLE TERMINAL
```

The terminal also displays the Dockerfile content and the resulting layers:

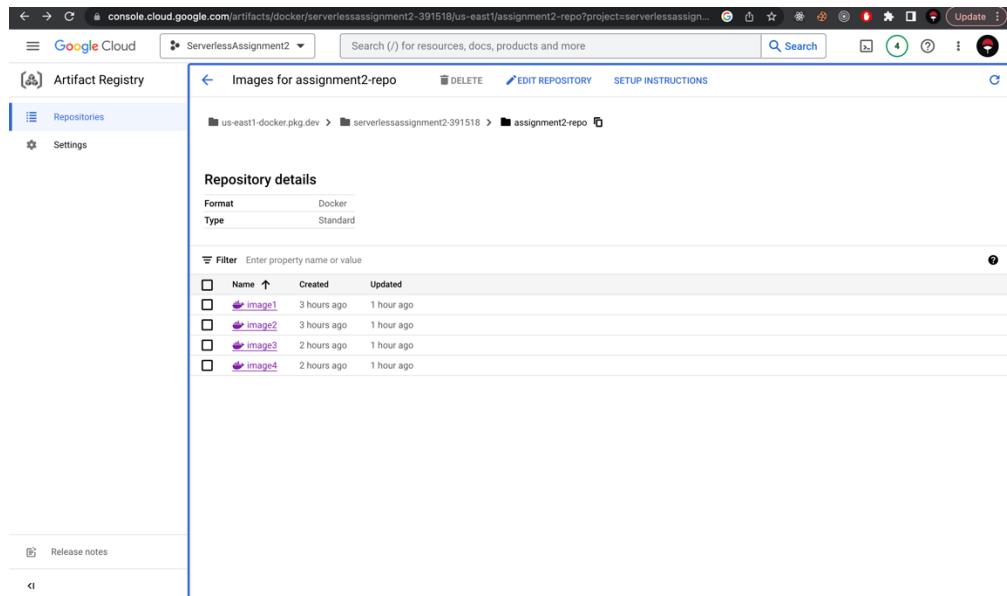
```
PUSHING us-east1-docker.pkg.dev/serverlessassignment2-391518/assignment2-repo/image4
The push refers to repository [us-east1-docker.pkg.dev/serverlessassignment2-391518/assignment2-repo] (Digest: sha256:83dff3a00538687ade8533ee42447f7cceab641947ed16585a4ea7d4857) size: 2486
76997a65bde: Pushed
60319a033e6: Layer already exists
bde9a7c663e8: Layer already exists
15228761504: Preparing
d9f9201374b: Preparing
2538717f0ab: Preparing
e7766bc308a: Preparing
c819071374b: Preparing
bc897201374b: Preparing
3da09f8072d2: Preparing
e7766bc308a: Waiting
c819071374b: Waiting
bc897201374b: Waiting
3da09f8072d2: Waiting
15228761504: Layer already exists
bde9a7c663e8: Layer already exists
d9f9201374b: Layer already exists
2538717f0ab: Layer already exists
e7766bc308a: Layer already exists
c819071374b: Layer already exists
3da09f8072d2: Layer already exists
76997a65bde: Pushed
latest digest: sha256:83dff3a00538687ade8533ee42447f7cceab641947ed16585a4ea7d4857 size: 2486
NONE
```

At the bottom, there is a table showing the status of the layers:

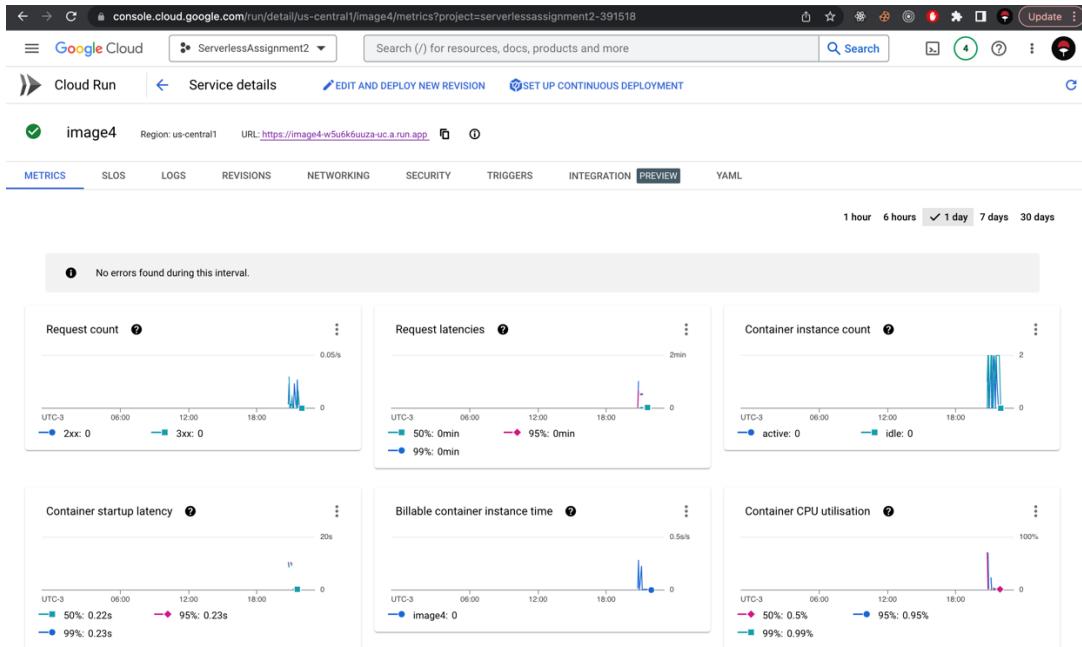
ID	CREATE_TIME	DURATION	SOURCE	IMAGES
1b60e0f15a-ebbe-486d-a009-67514e1e3-2a1	2023-07-07T08:23:13+00:00	3H55M	g1t/kerverlessassignment2-391518_c1oudBuild/d/source/1688689392/2576-5609559217/a98-285/x2564e42f109	uv-wx-11-dckr
-	-	-	-	-

Below the table, there are navigation links: OUTLINE, TIMELINE, and a search bar with placeholder text "Ln 23, Col 12 Spaces: 4 UTF-8 LF Docker G Shell".

**Fig 47:** Image 4 pushed in artifact registry



**Fig 48:** Image pushed in artifact registry



**Fig 49:** Image 4 deployed in cloud run as same as other containers.

## Testing the deployed website

The screenshot shows a web browser displaying a 'Signup Form' for 'Assignment 2'. The form fields are: Name (ultimateTesting), Email (ultimate@dal.ca), Password (\*\*\*\*\*), and Location (Halifax). A 'SUBMIT' button is at the bottom. The browser's address bar shows the URL <https://image4-w5u6k6uuza-uc.a.run.app>.

**Fig 50:** Signup form

The screenshot shows a browser window with a red header bar containing the text "Assignment 2" and navigation links for "Home" and "Login". The main content area displays a "Login Form" with two input fields: "Email" and "Password". The "Email" field contains "ultimate@dal.ca" and the "Password" field contains "\*\*\*\*\*". Below the form is a blue "SUBMIT" button. At the bottom of the page, a message reads "Error occurred during login".

**Fig 51:** Login Page being redirected to from signup page

The screenshot shows a browser window with a red header bar containing the text "Assignment 2" and navigation links for "Home" and "Login". The main content area displays a message "Hi, ultimateTesting you are logged in." followed by a table titled "Here are other users who are online". The table has three columns: "Name", "Email", and "Location". The data in the table is as follows:

Name	Email	Location
testing8	testing8@dal.ca	Toronto
fenil patel	fenil@dal.ca	halifax
Max Verstappen	max@dal.ca	Netherlands
testing4	testing4@dal.ca	Delhi
testing3	testing3@dal.ca	Tokyo
testing7	testing7@dal.ca	Dallas
Sarthak Patel	sr555161@dal.ca	Halifax

At the bottom of the page is a blue "LOGOUT" button.

**Fig 52:** On successful login display the online users.

The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar includes 'Project Overview', 'Firestore Database' (selected), 'Analytics', 'Engage', and 'Blaze'. Under 'Customise your navigation', there are 'Learn more' and 'Got it' buttons. The main area displays the 'Cloud Firestore' section for project 'ServerlessAssignment2'. It shows a 'Reg' collection containing a document named 'ultimate@dal.ca'. This document has fields: email ('ultimate@dal.ca'), location ('Halifax'), name ('UltimateTesting'), and password ('ultimate'). A 'state' collection is also visible.

Fig 53: User stored in Firestore database

This screenshot is identical to Fig 53, showing the same user document 'ultimate@dal.ca' in the 'Reg' collection. However, the 'state' collection now contains a single document with the field 'state' set to 'Online' and a timestamp value of '1688695693575'.

Fig 54: After logging the user is online

The screenshot shows the Firebase Cloud Firestore interface. On the left, a sidebar lists various services: Micro, Perf, Learn, Lex C, (S) V, Data, CRUI, Fire, (335), Java, (5) F, delete, (A) Imag, (P) Metr, (M) (no s). The main area is titled 'Cloud Firestore' and shows a collection named 'state'. The collection contains documents for email addresses: max@dal.ca, sr55161@dal.ca, testing1@dal.ca, testing2@dal.ca, testing3@dal.ca, testing4@dal.ca, testing5@dal.ca, testing6@dal.ca, testing7@dal.ca, testing8@dal.ca, testing9@dal.ca, testing@dal.ca, and testingForDocument@dal.ca. A modal dialog is open at the top right, titled 'Protect your Cloud Firestore resources from abuse, such as billing fraud or phishing', with a 'Configure App Check' button.

Fig 55: After logout the user is offline