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/

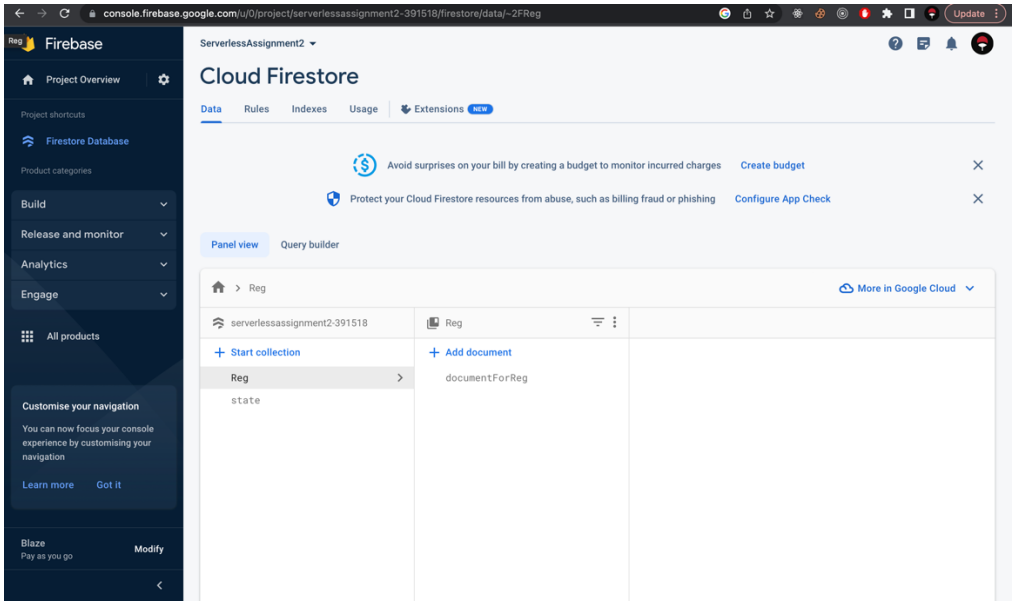

Fig 1: Unpopulated Firestore

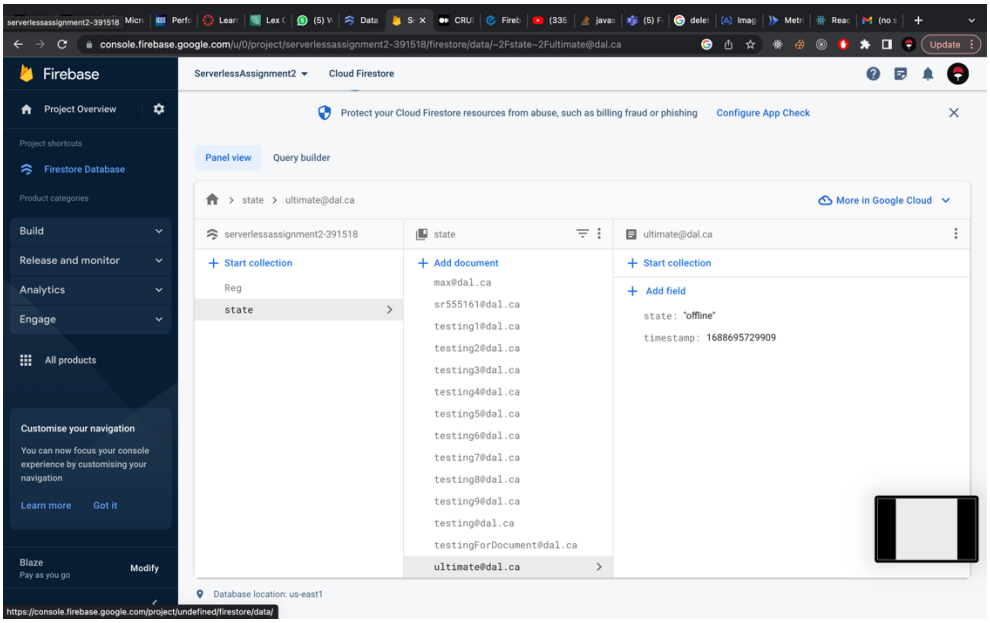

Fig 2: Populated Firestore

Container 1

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

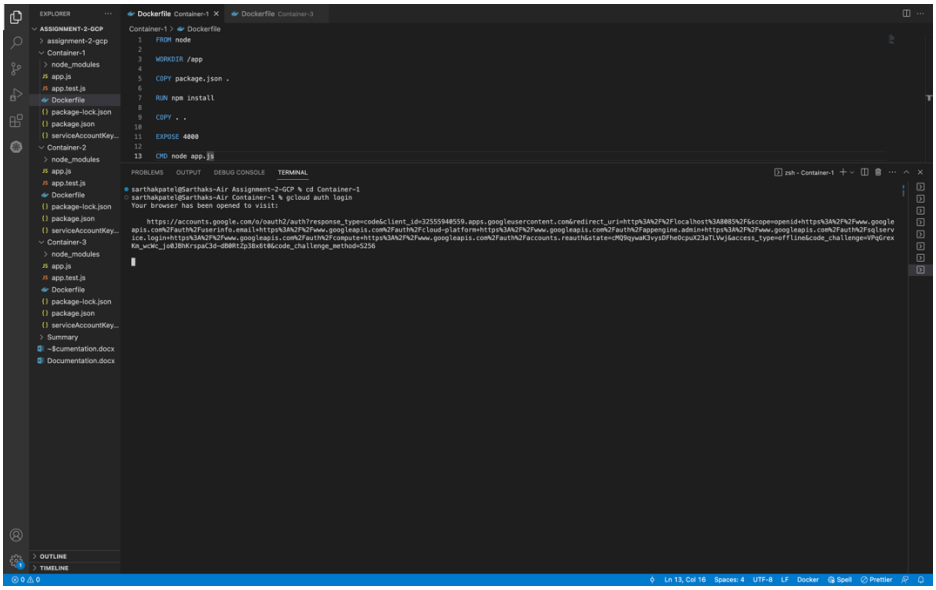

Fig 3: auth login to my Gmail account.

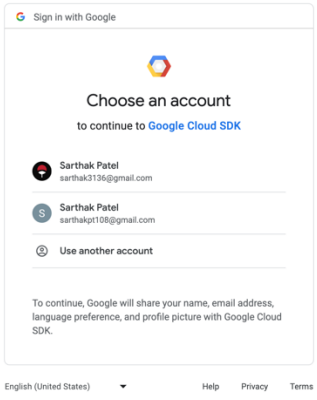

Fig 4: Selecting the google cloud account.

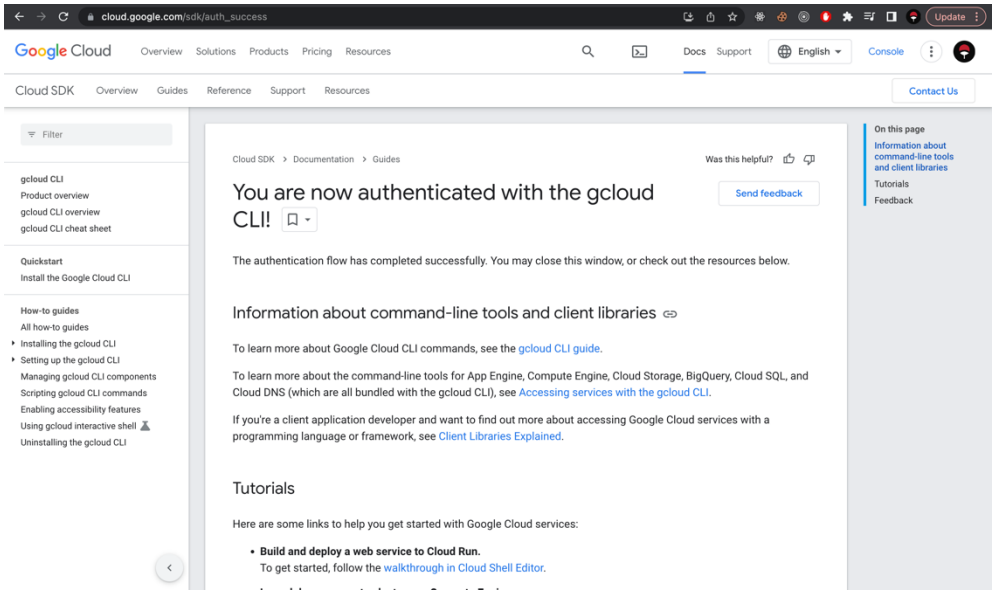

Fig 5: Successful authentication

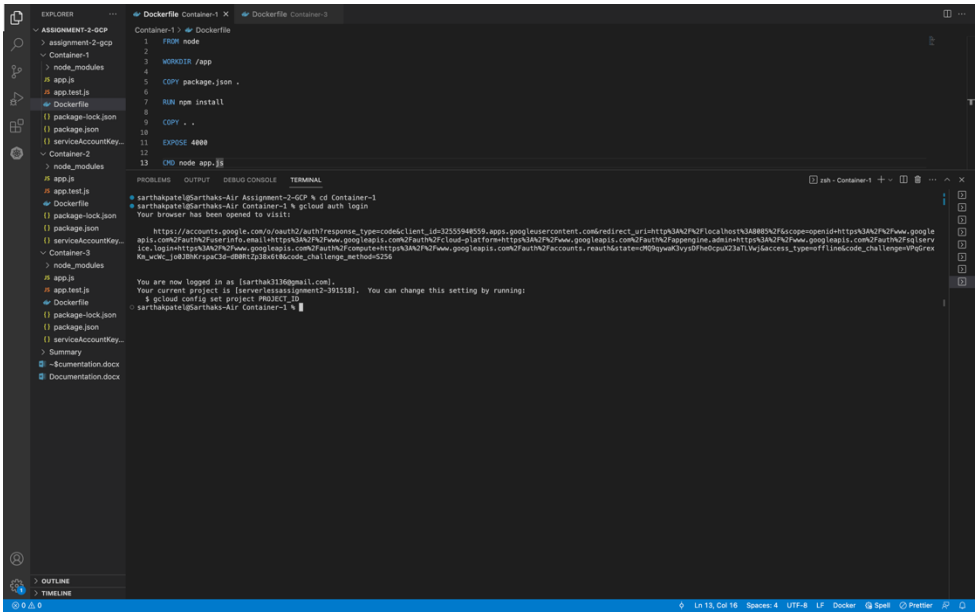

Fig 6: Successful login seen in the terminal

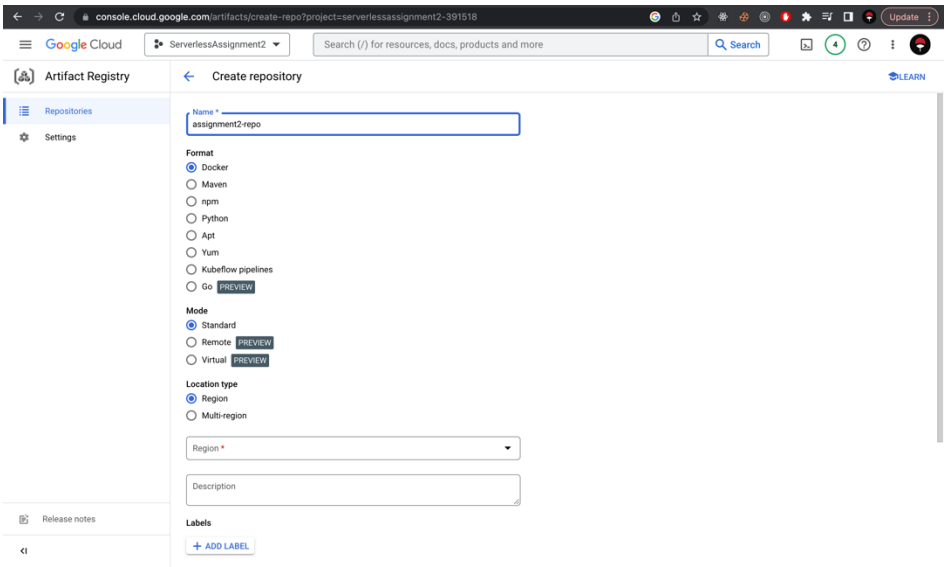

Fig 7: Creating an Artifact Registry repository

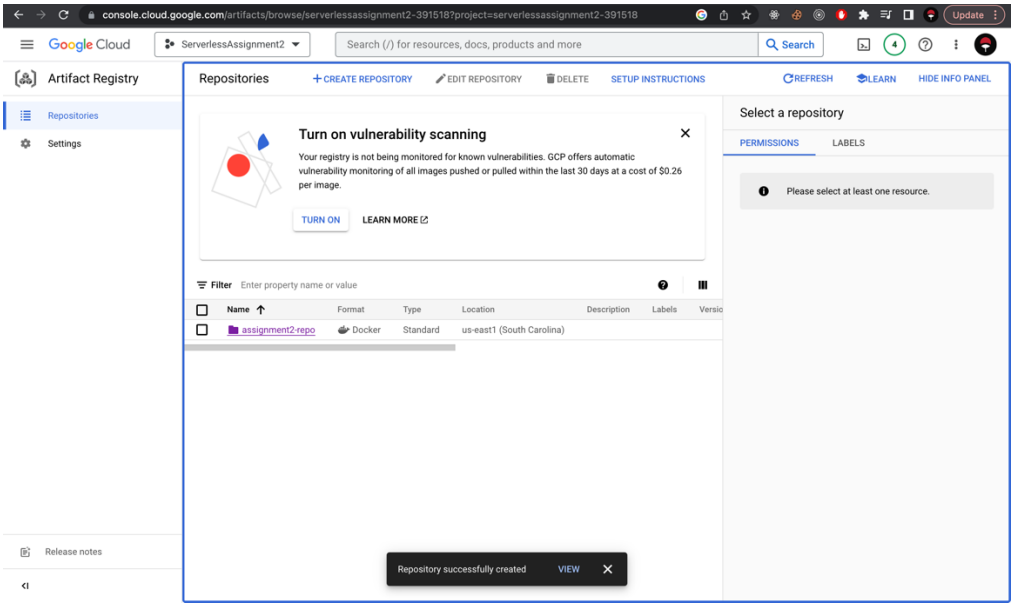

Fig 8: assignment2-repo created in artifact registry

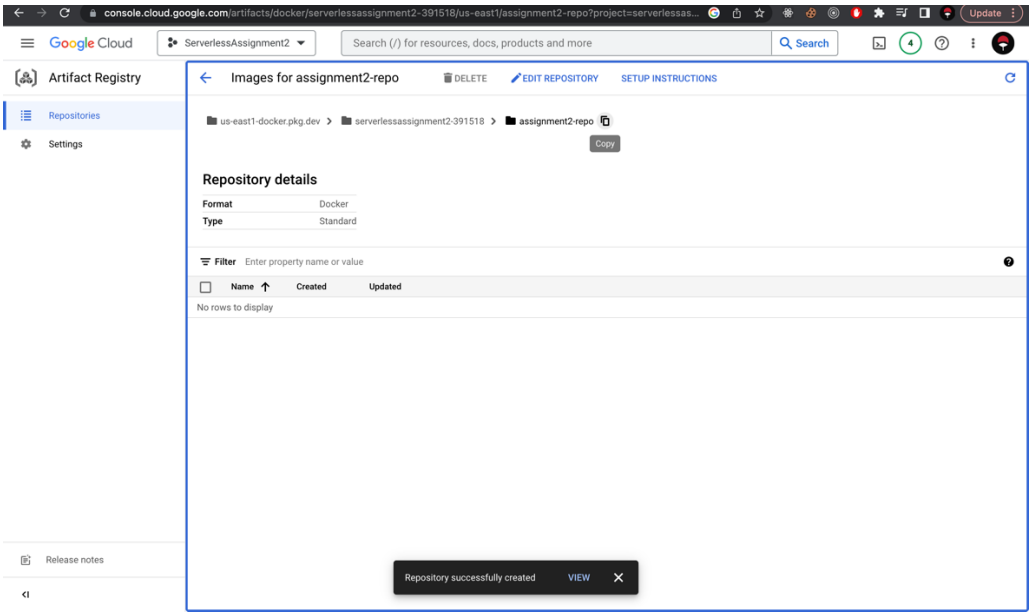

Fig 9: Initially the repository is empty

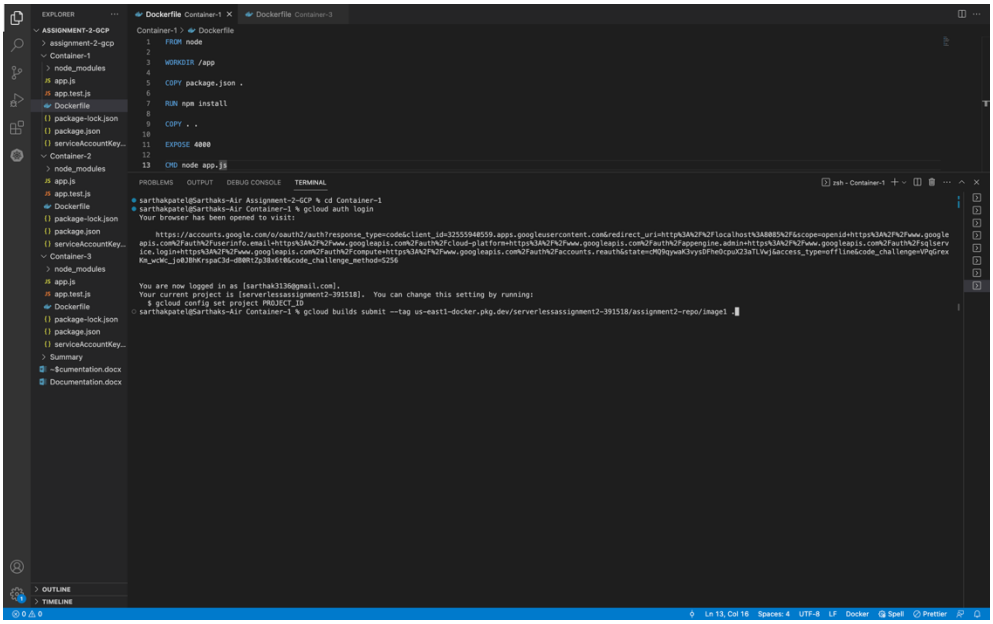

Fig 10: Building and submitting the docker image from the docker file to registry.

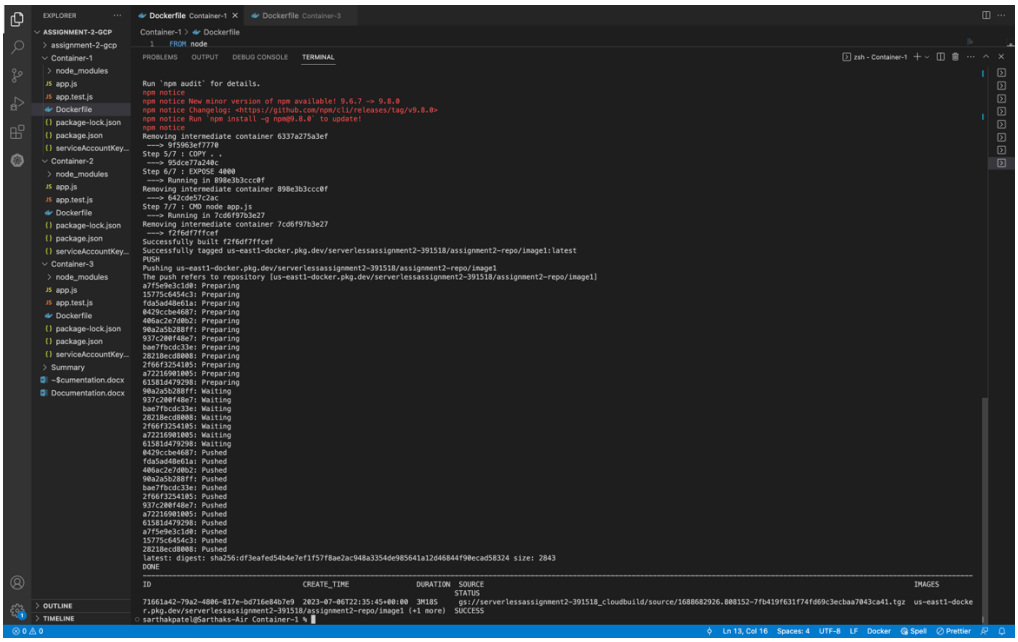

Fig 11: Image for container 1 code pushed in repo.

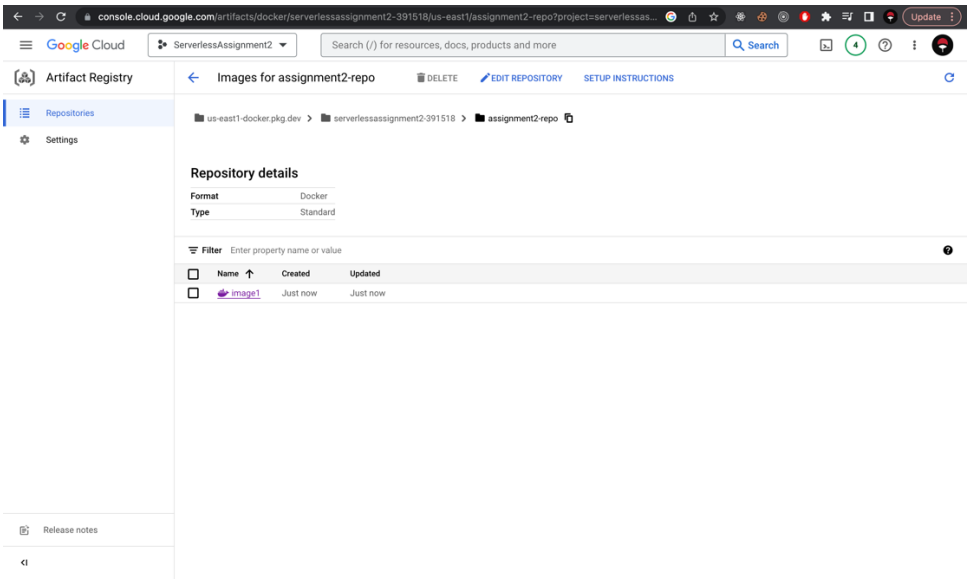

Fig 12: Image1 folder seen in artifact repository.

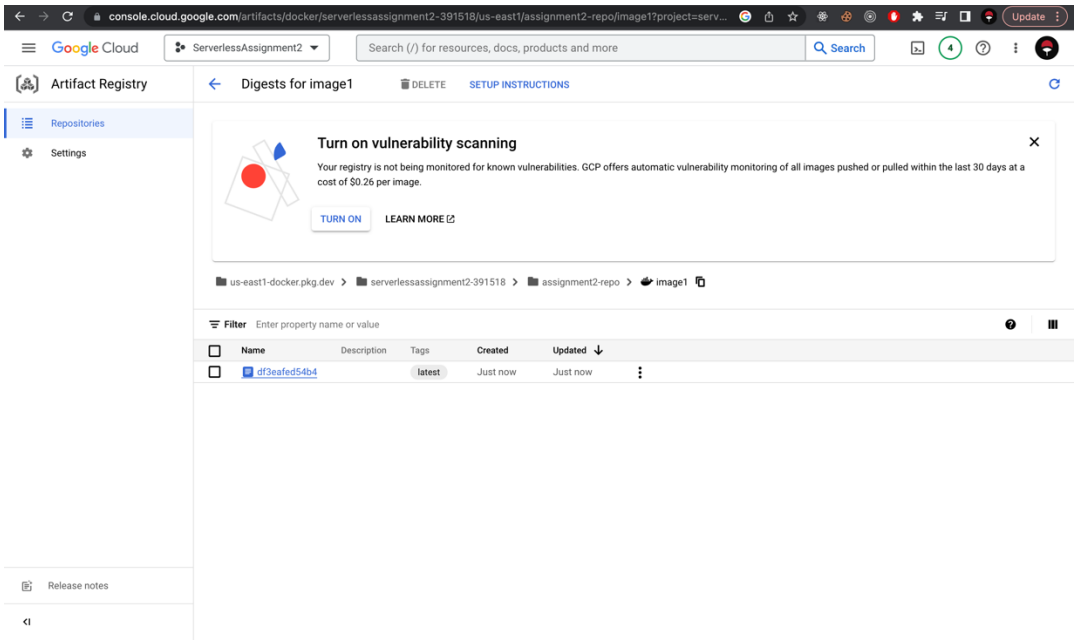

Fig 13: Docker image for our container

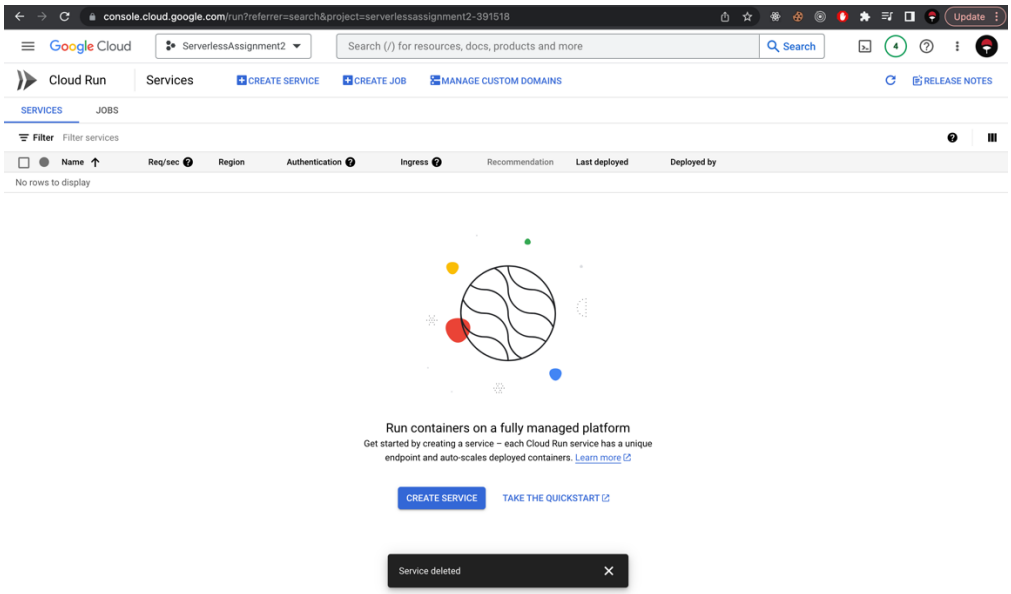

Fig 14: Empty Cloud Run

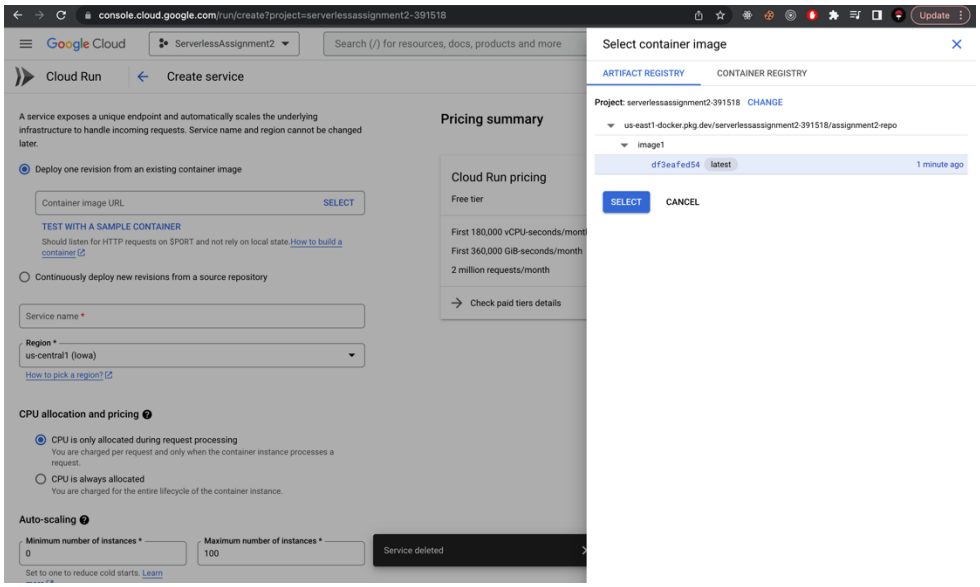

Fig 15: Selecting the image we want to deploy in cloud run.

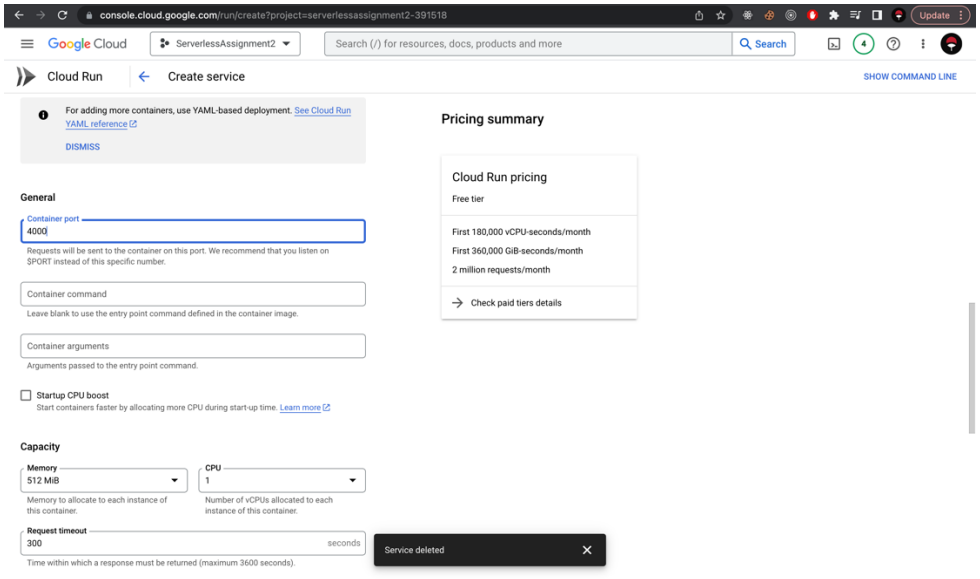

Fig 16: Specifying the port as same as my node js app.

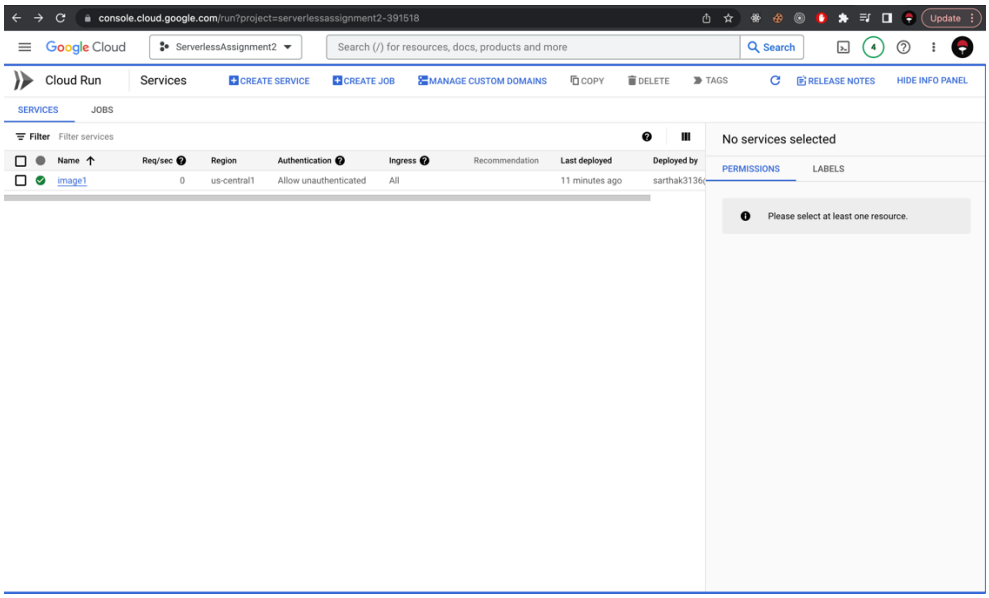

Fig 17: Image 1 cloud run

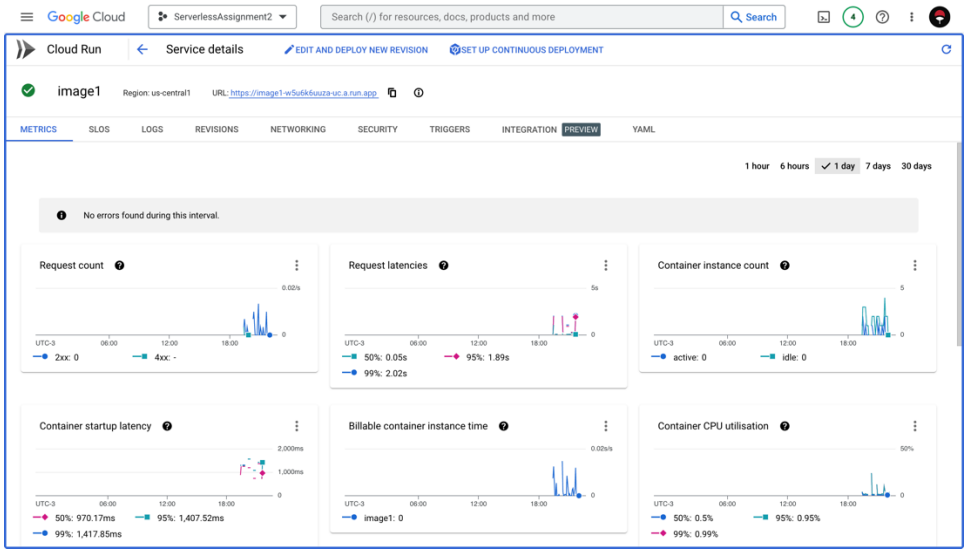

Fig 18: Successful deployment and deployment link

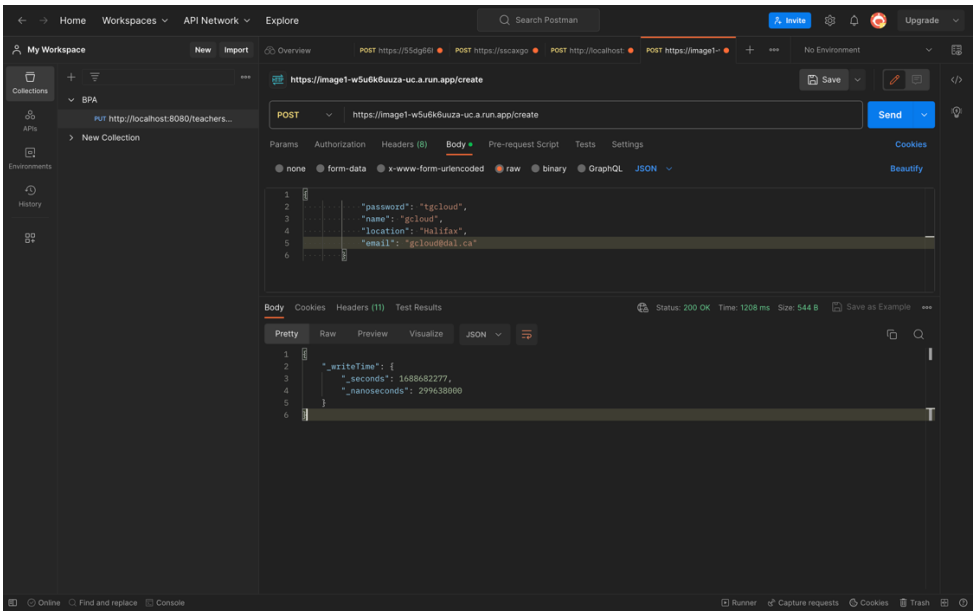

Fig 19: Testing the deployment link for the REST services in my container-1

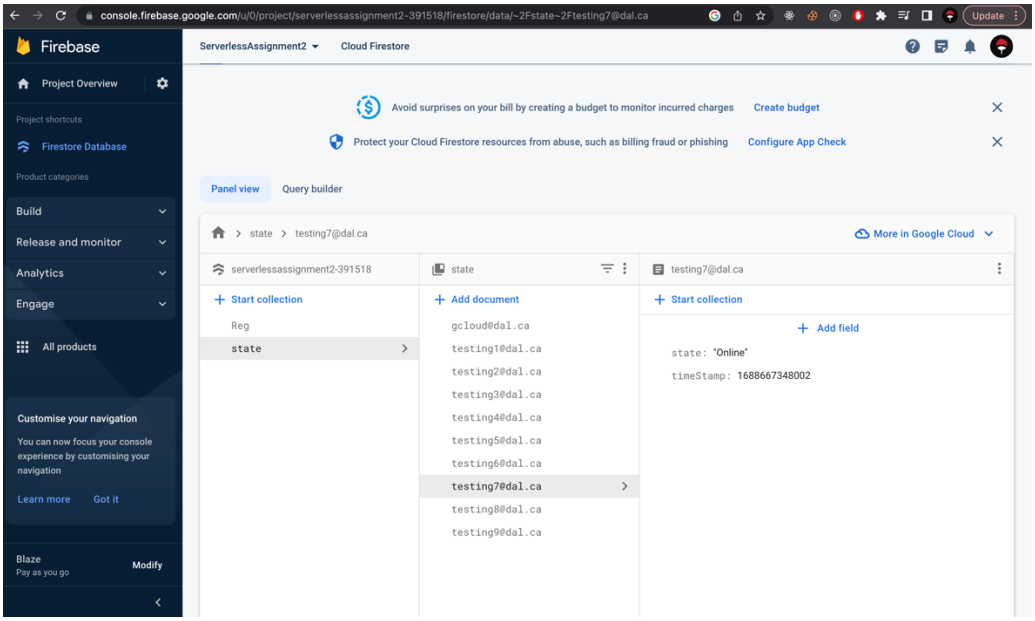

Fig 20: User getting created

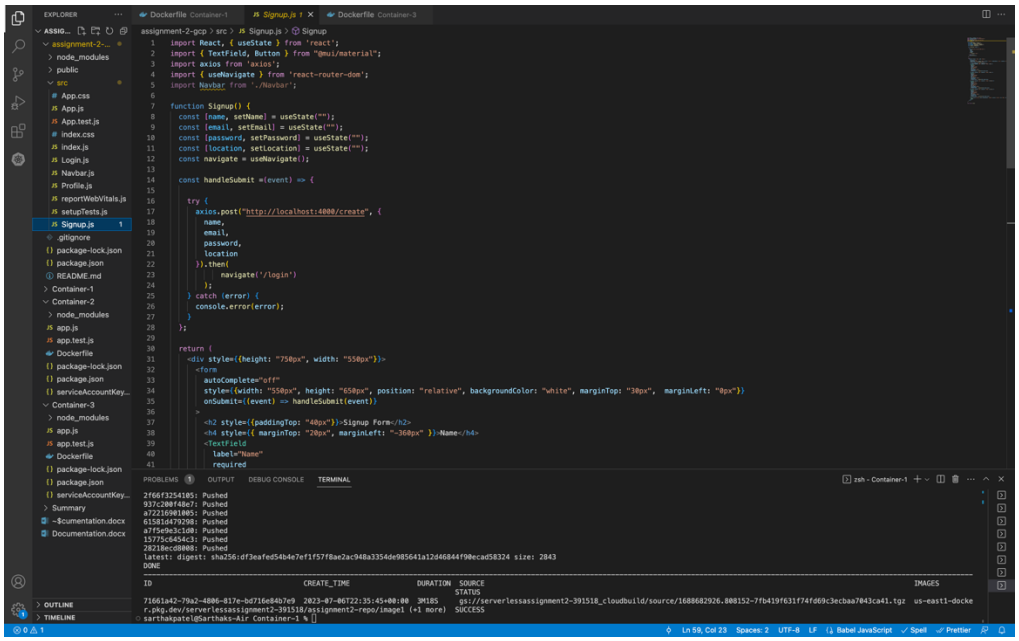

Fig 21: Original REST API for Signup or create

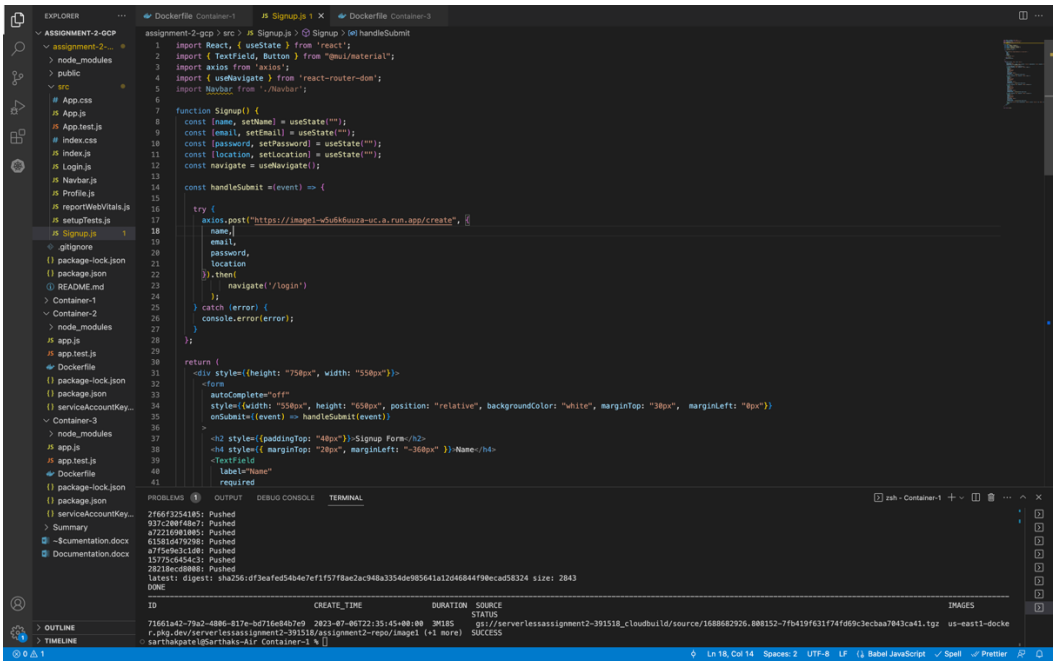

Fig 22: Replacing the original API with the deployed one.

Container 2

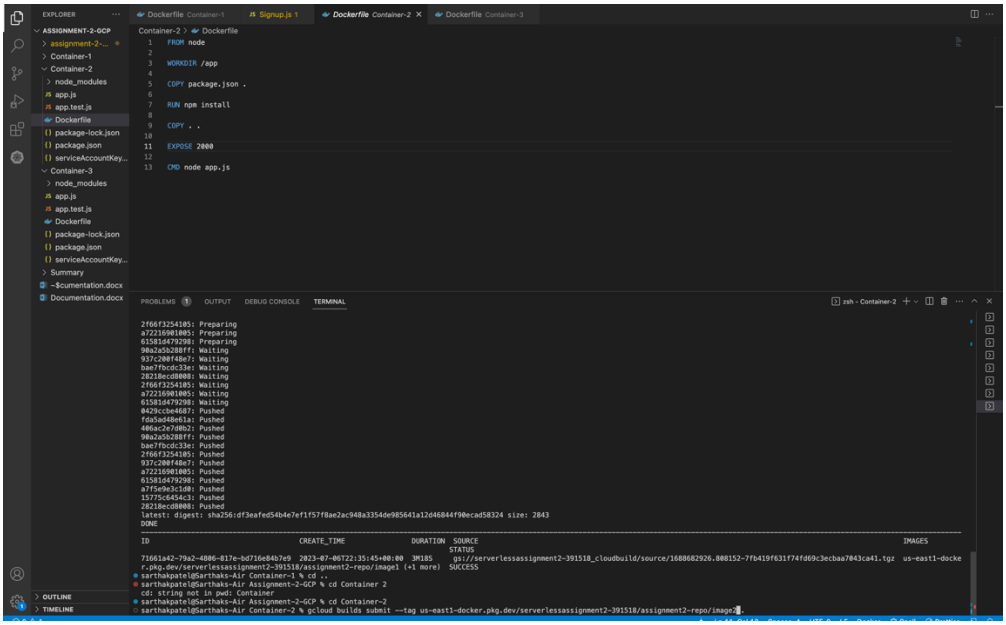

Fig 23: Building and pushing the image of container 2 to the artifact registry repo.

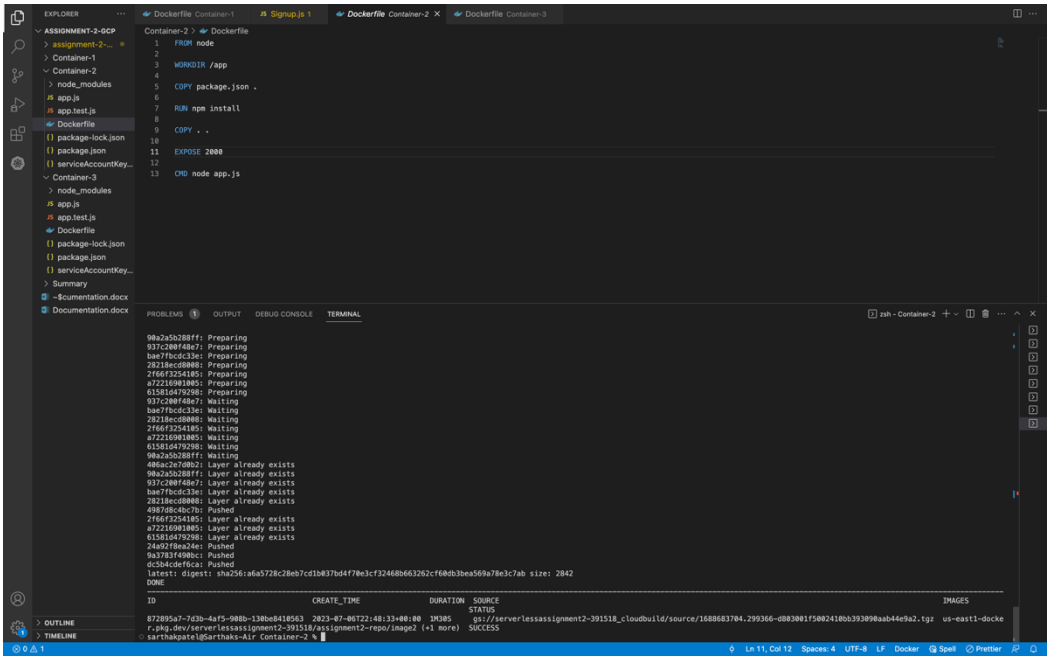

Fig 24: Image successfully pushed

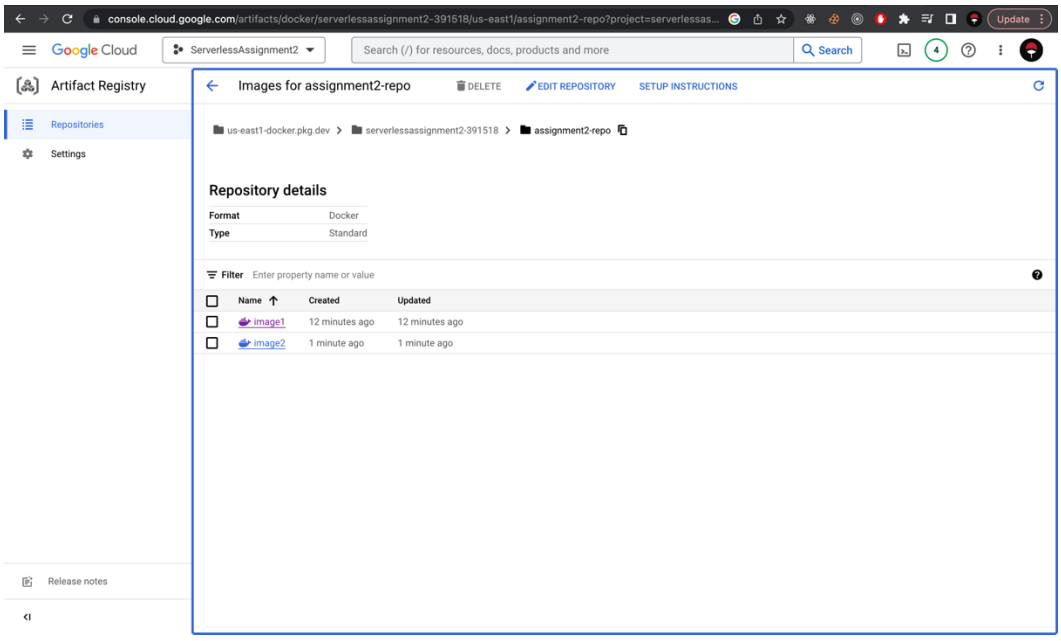

Fig 25: Image 2 for the container 2

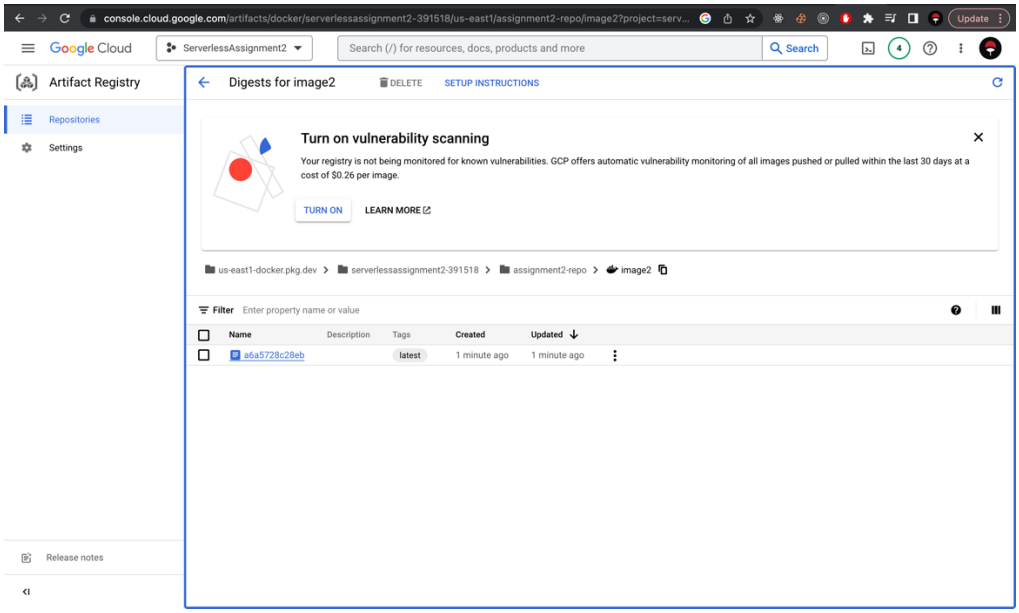

Fig 26: Image pushed in Image 2 repo.

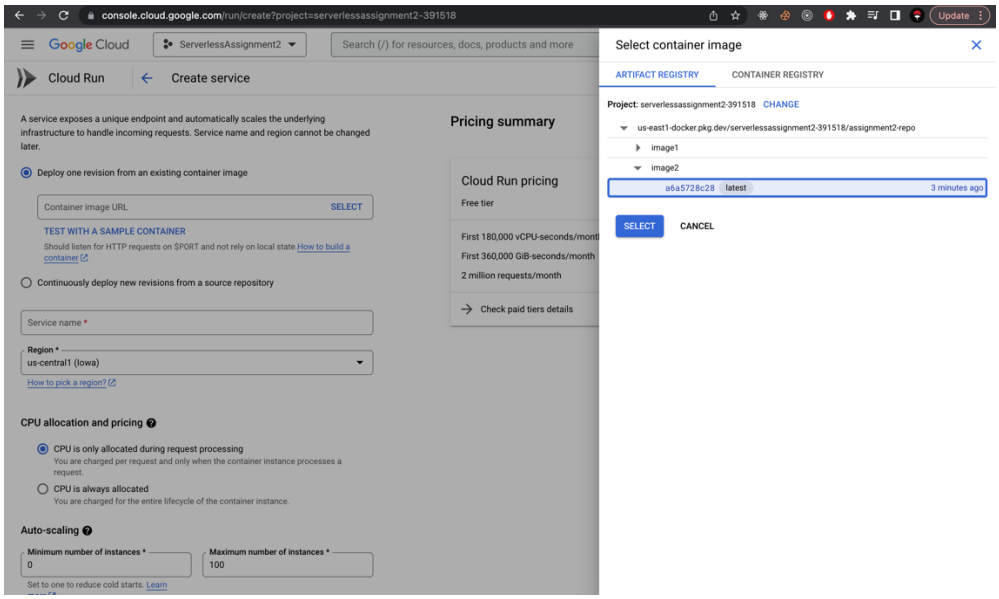

Fig 27: Selecting image from image 2 as deployment.

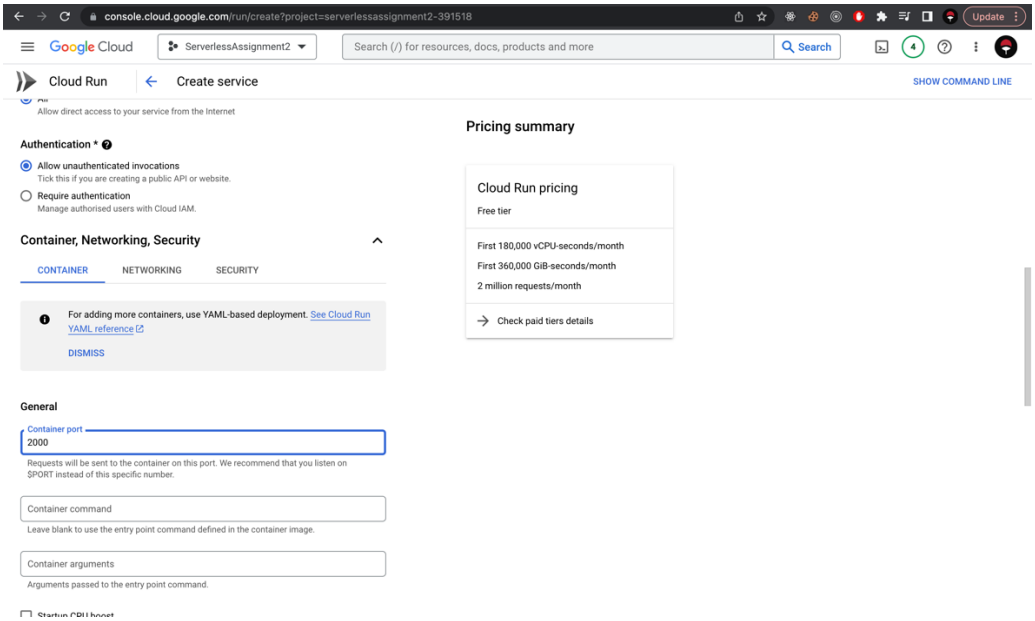

Fig 28: Port same as container 2 port of app.js

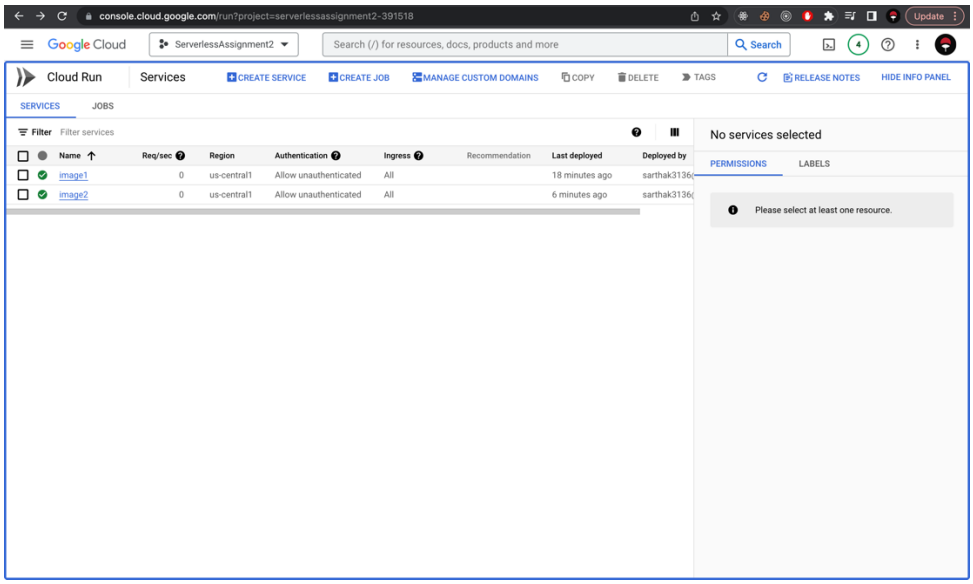

Fig 29: Cloud run for images 1 and 2.

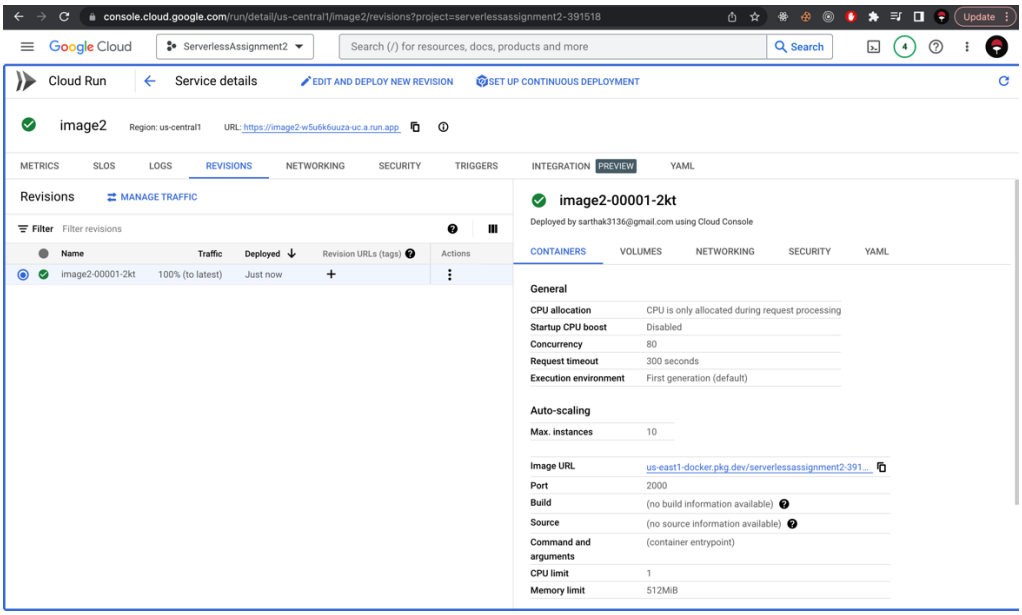

Fig 30: Cloud run for image 2 with the deployment link

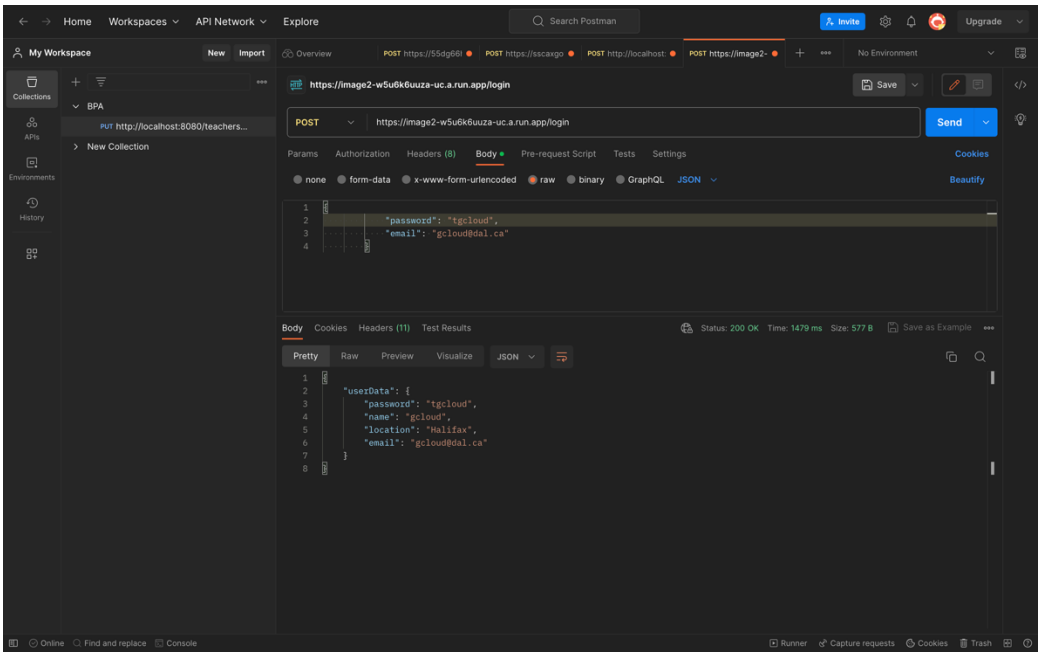

Fig 31: Testing the login REST API with the deployed URL.

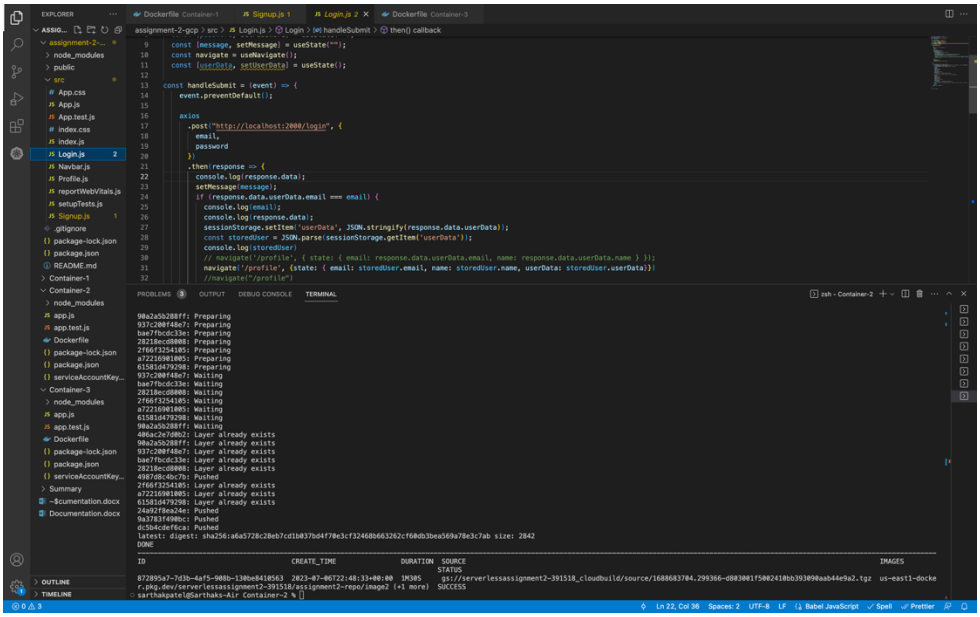

Fig 32: Old REST API for the login

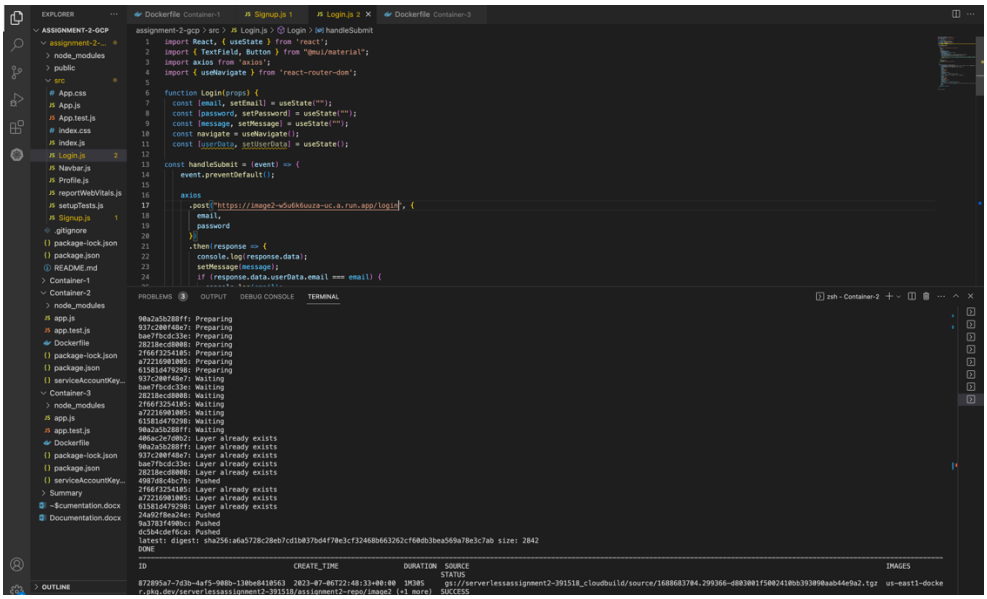

Fig 33: Updated REST API for the login

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

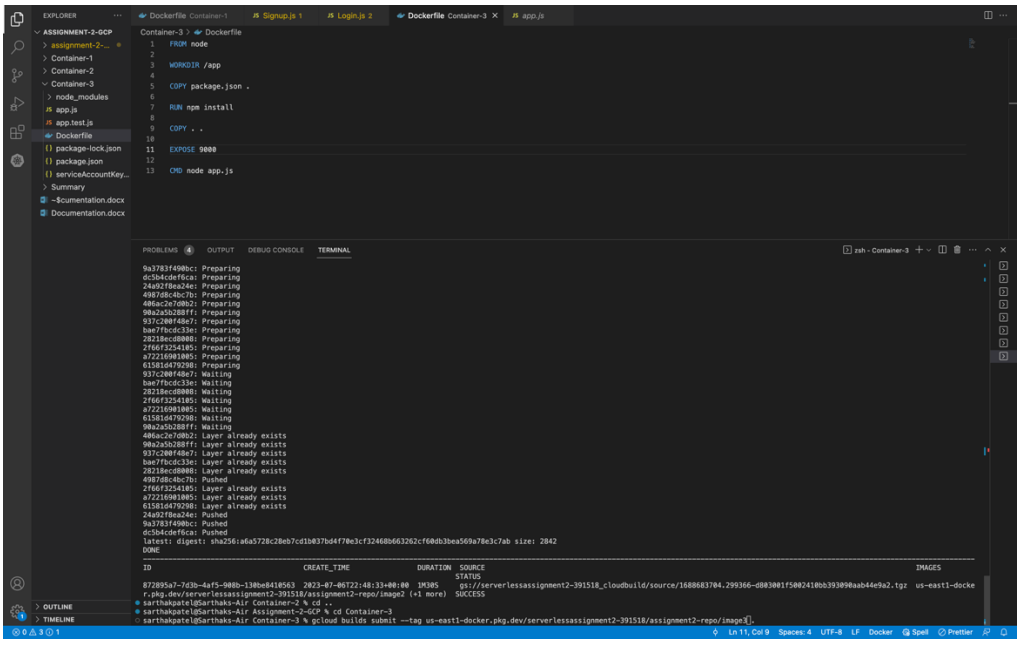

Fig 34: Building and pushing container 3 image in artifact repo

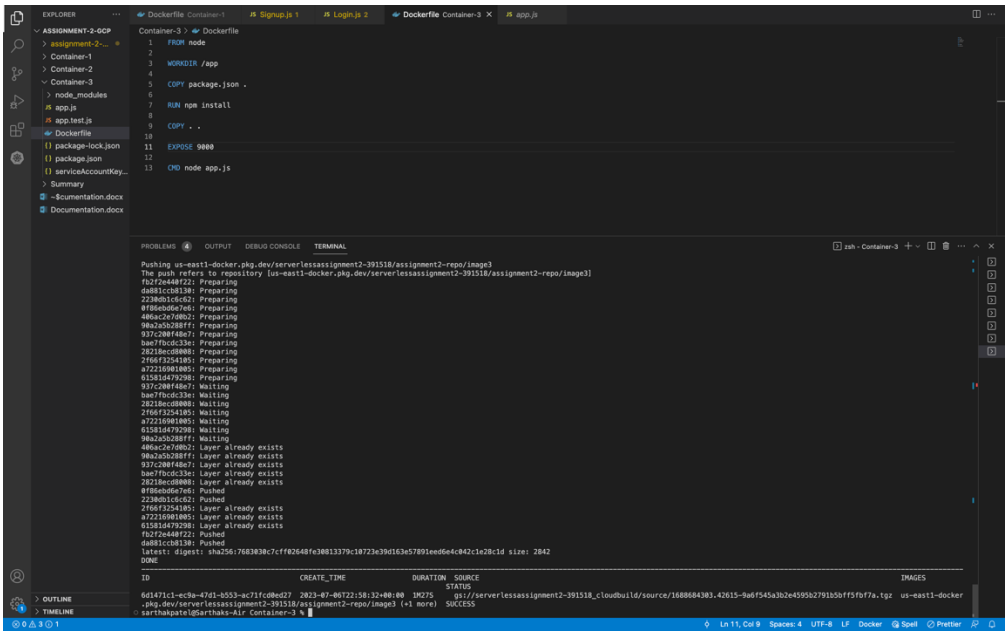

Fig 35: Successfully pushed in artifact repo

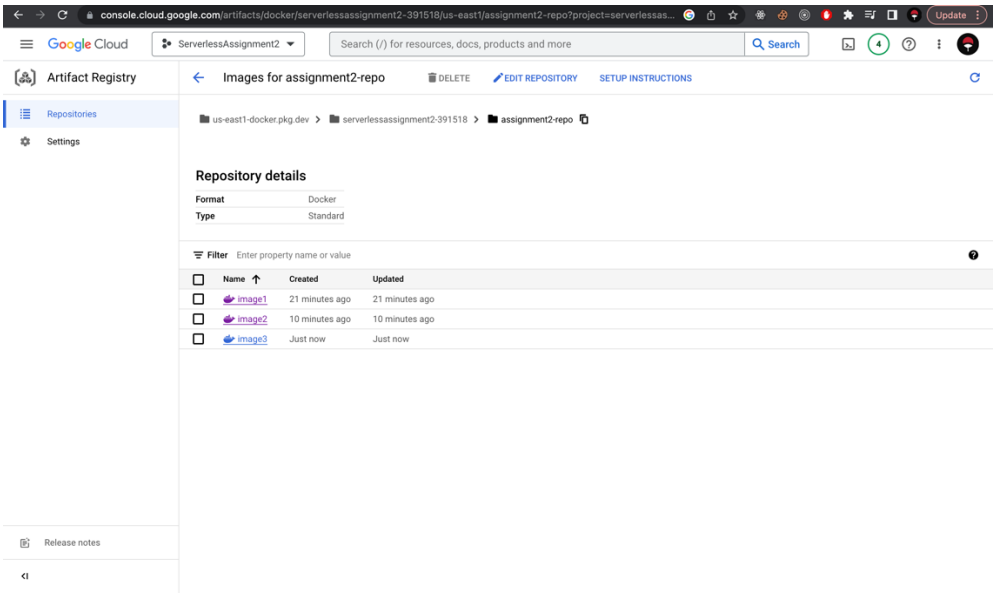

Fig 36: Images 3 folder created in artifact repo

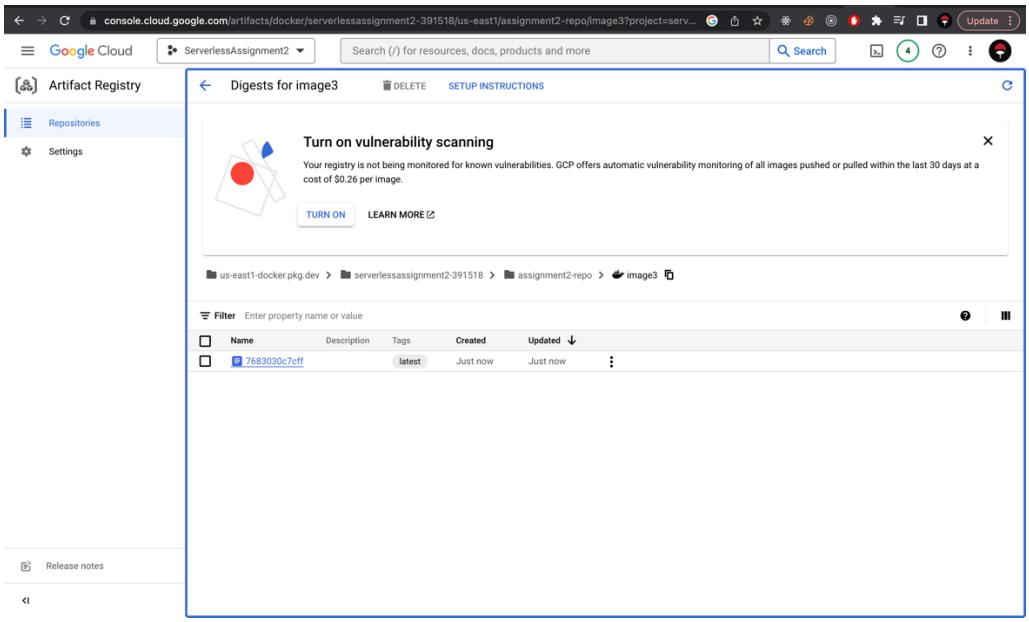

Fig 37: Image for the container 3

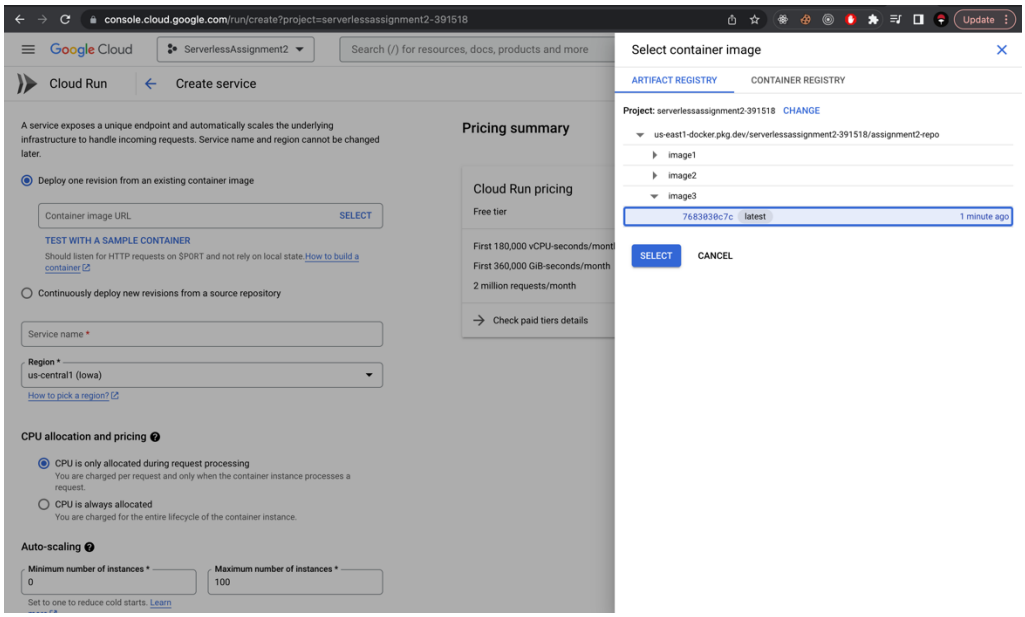

Fig 38: Creatin a cloud run deployment for image 3.

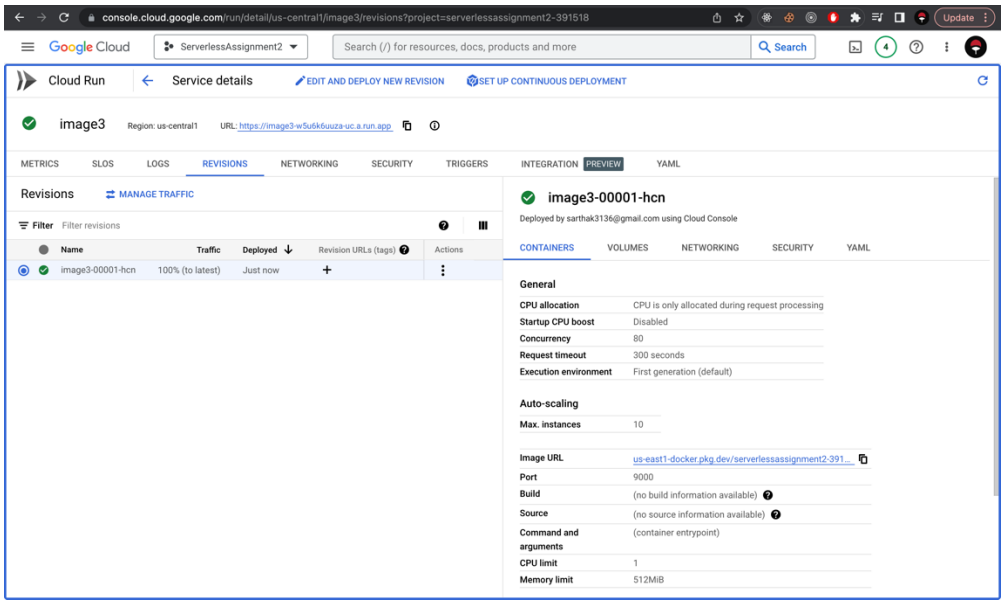

Fig 39: Cloud run for image 3 with the deployment link for the microservice

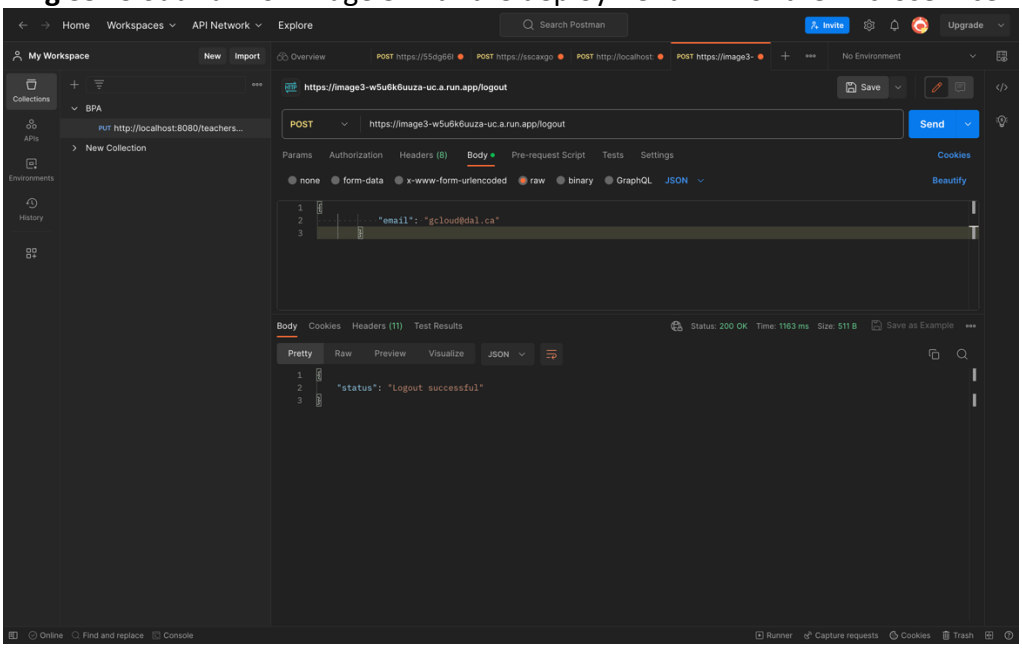

Fig 40: Testing the logout API for the deployed microservice

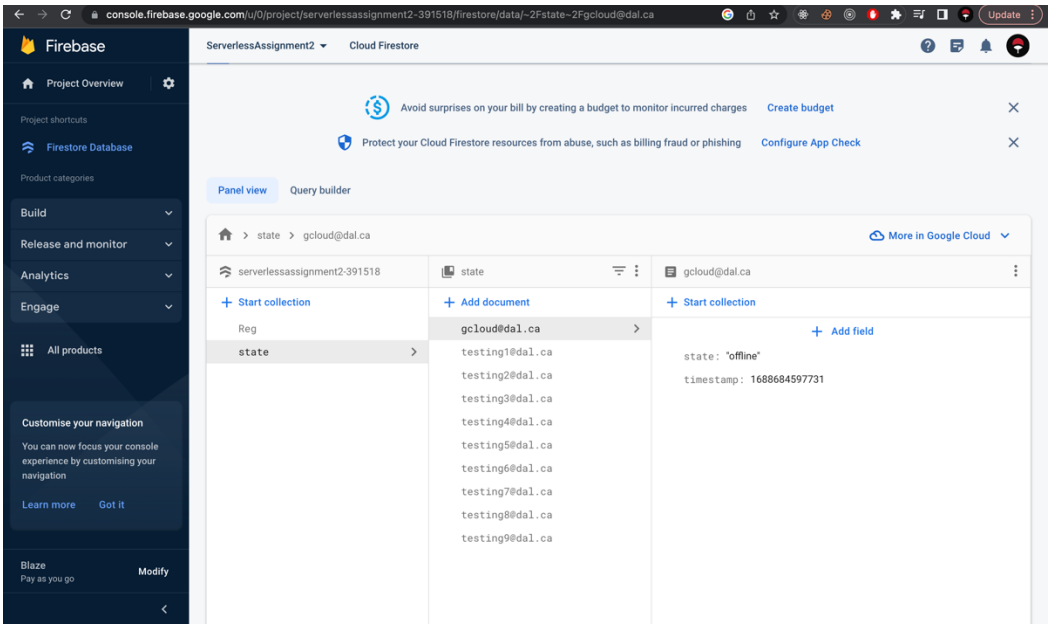

Fig 41: User getting logged out

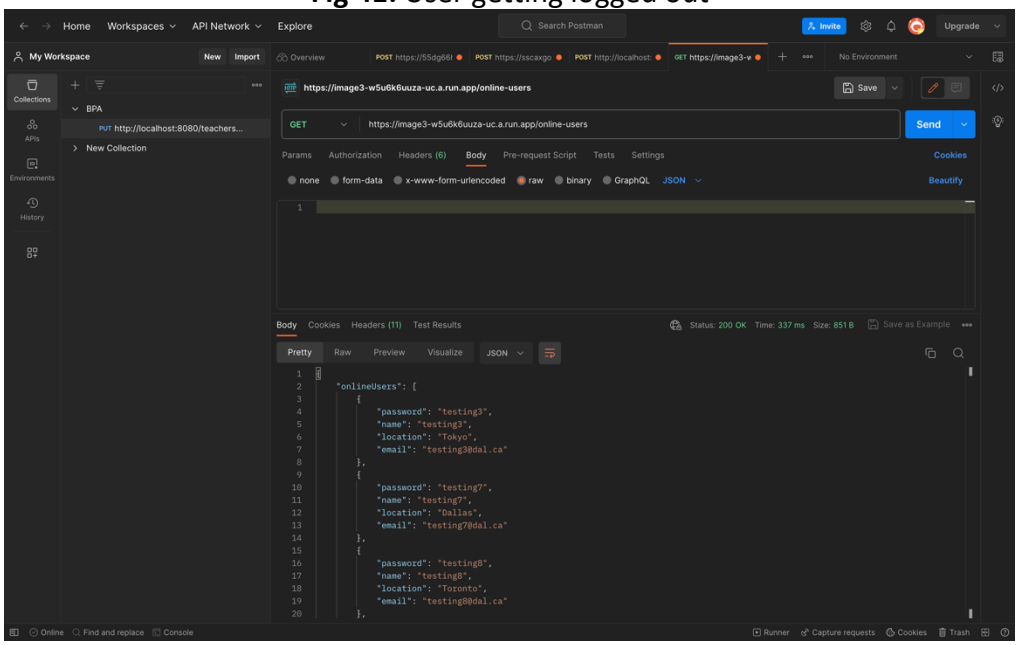

Fig 42: Testing the online-users API for the deployed microservice

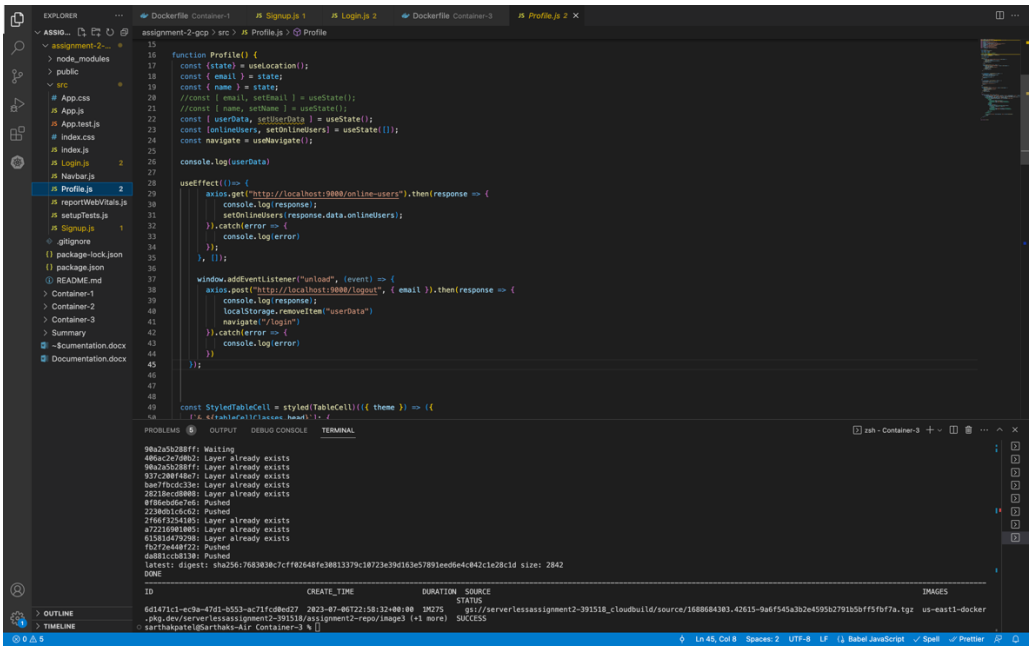

Fig 43: Old APIs in React

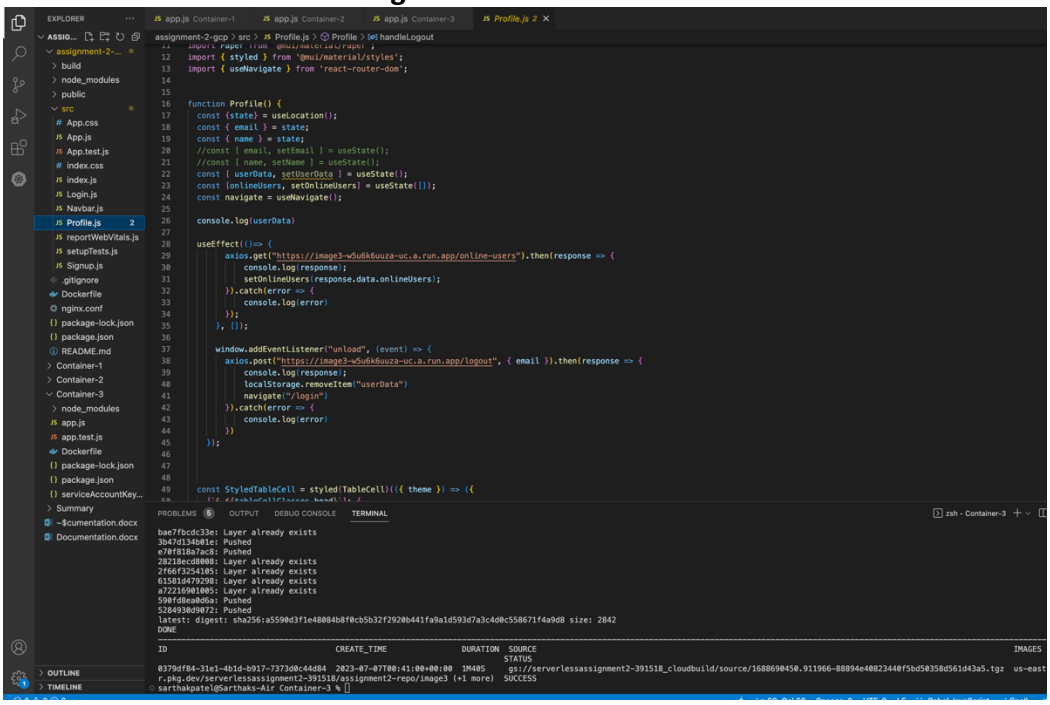

Fig 44: Updated REST API in React

Deploying the React Application

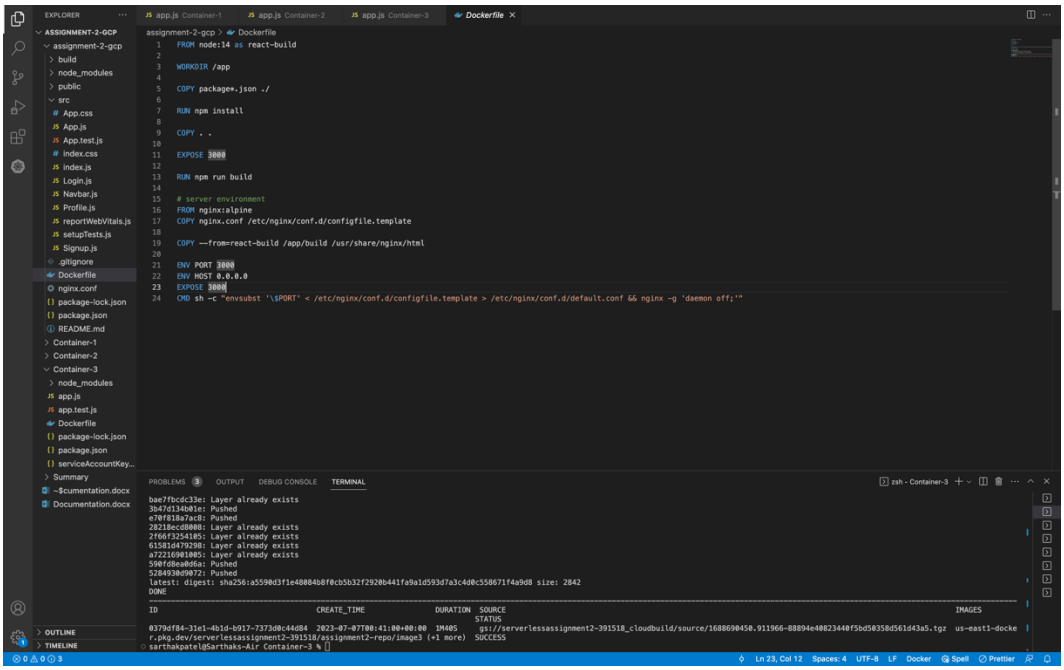

Fig 45: Docker file for the frontend REACT deployment.

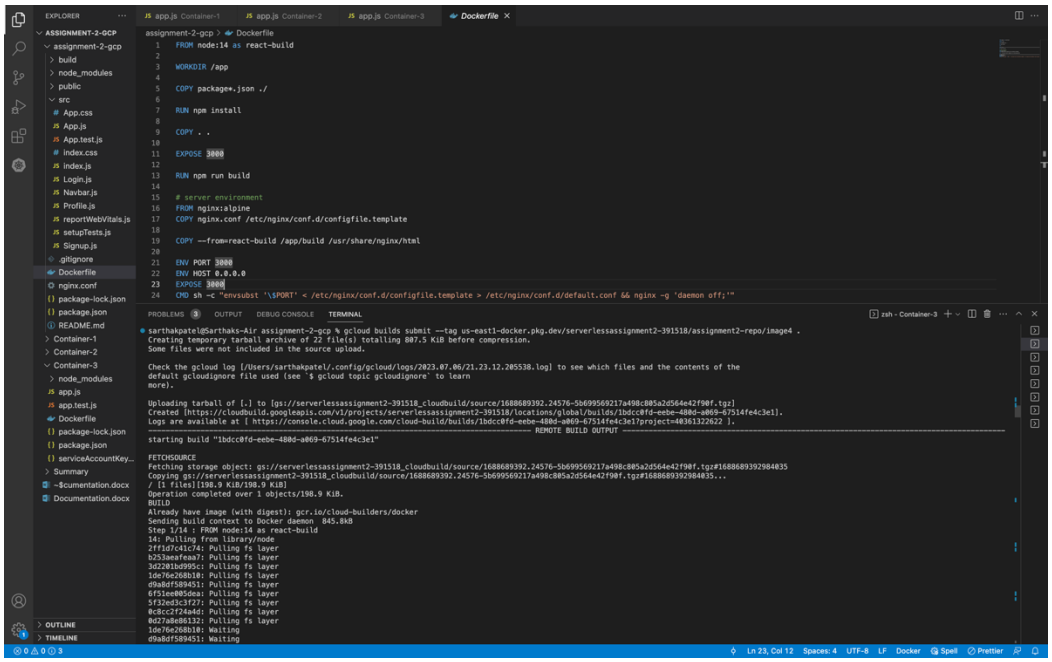

Fig 46: Building and Pushing image for the frontend React

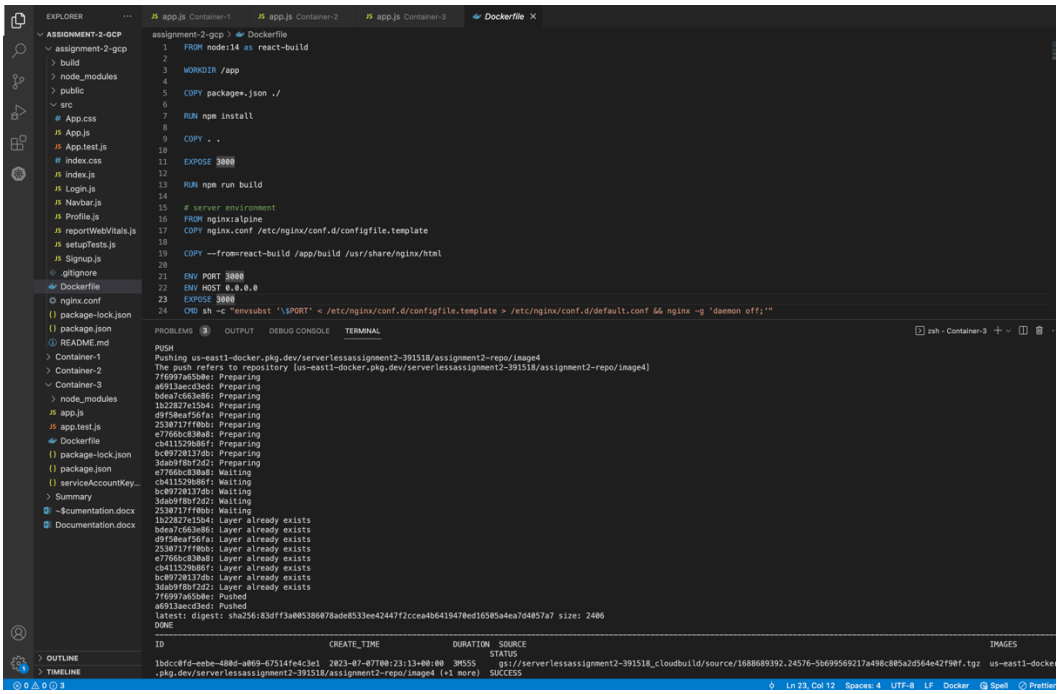

Fig 47: Image 4 pushed in artifact registry

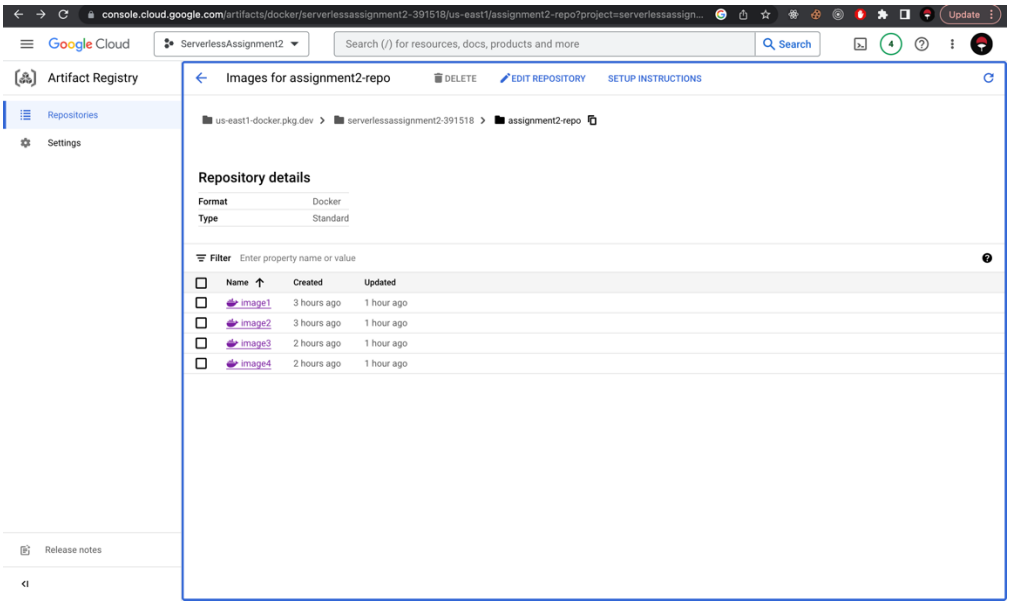

Fig 48: Image pushed in artifact registry

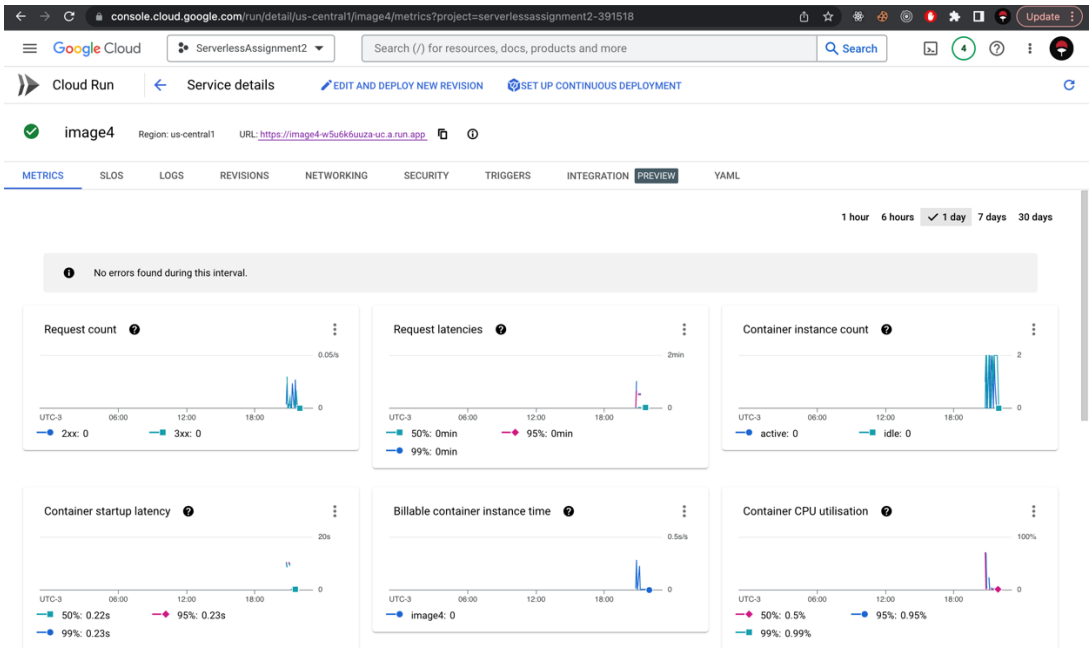

Fig 49: Image 4 deployed in cloud run as same as other containers.

Testing the deployed website

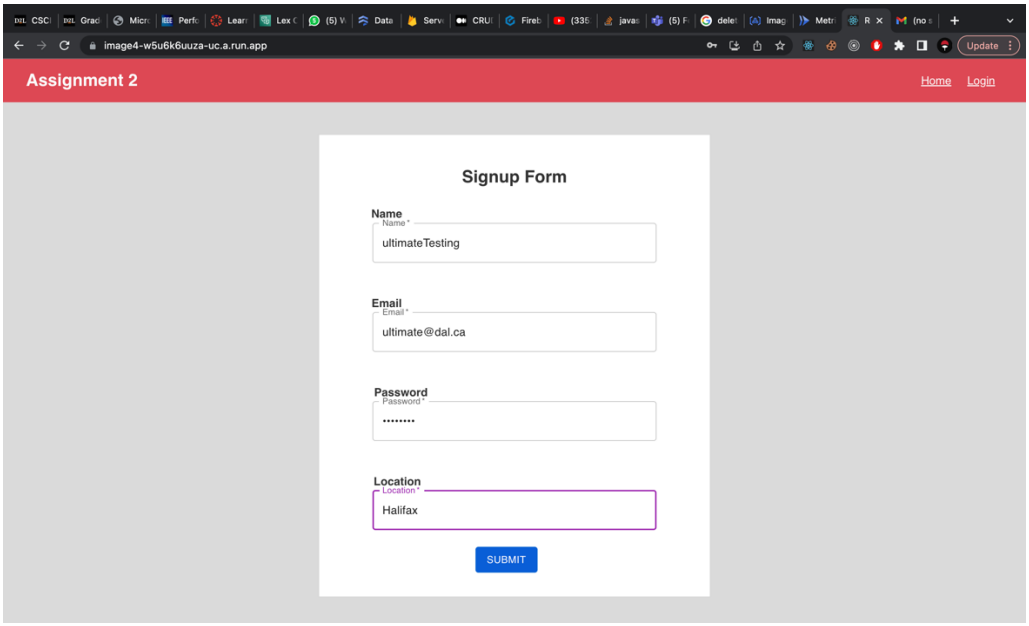

Fig 50: Signup form

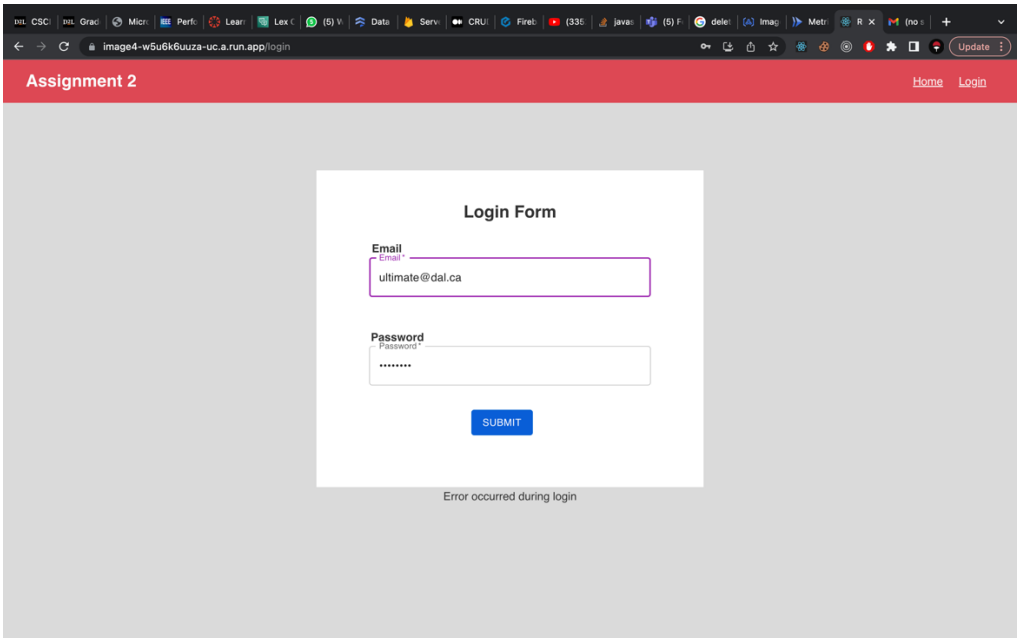

Fig 51: Login Page being redirected to from signup page

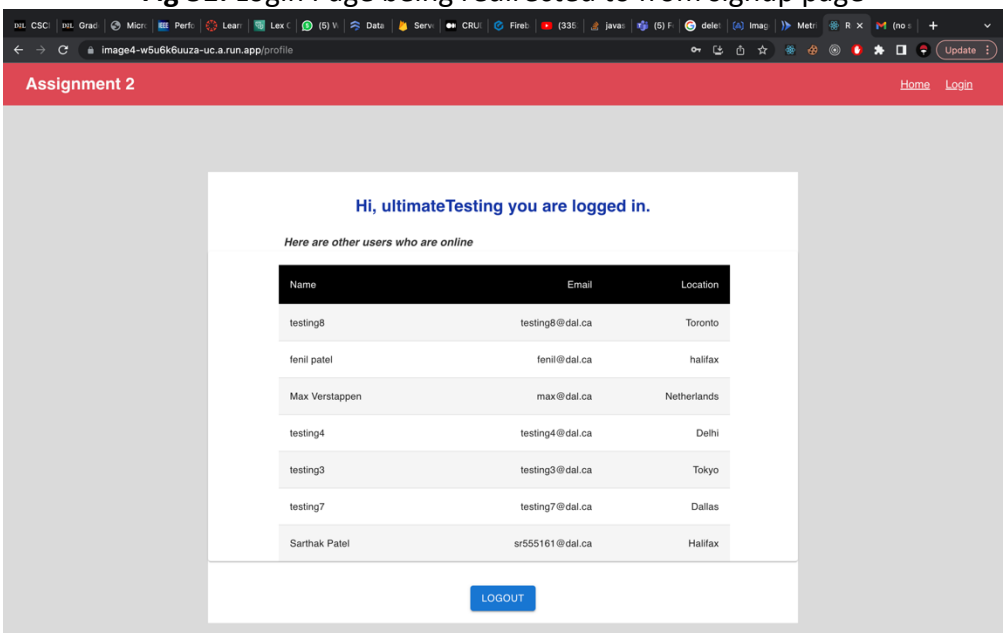

Fig 52: On successful login display the online users.

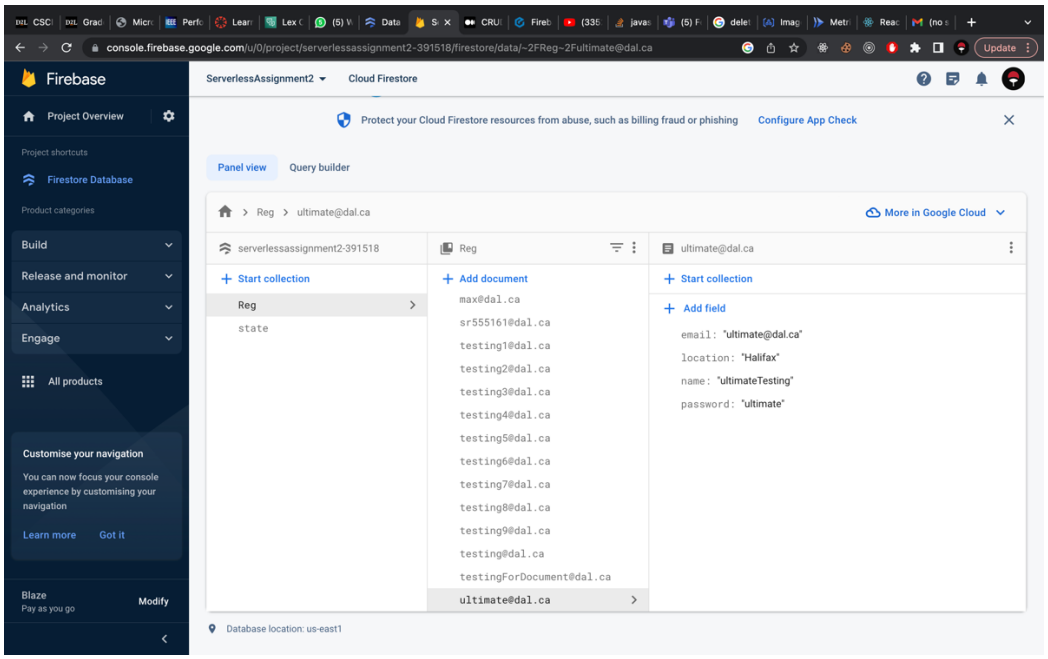

Fig 53: User stored in Firestore database

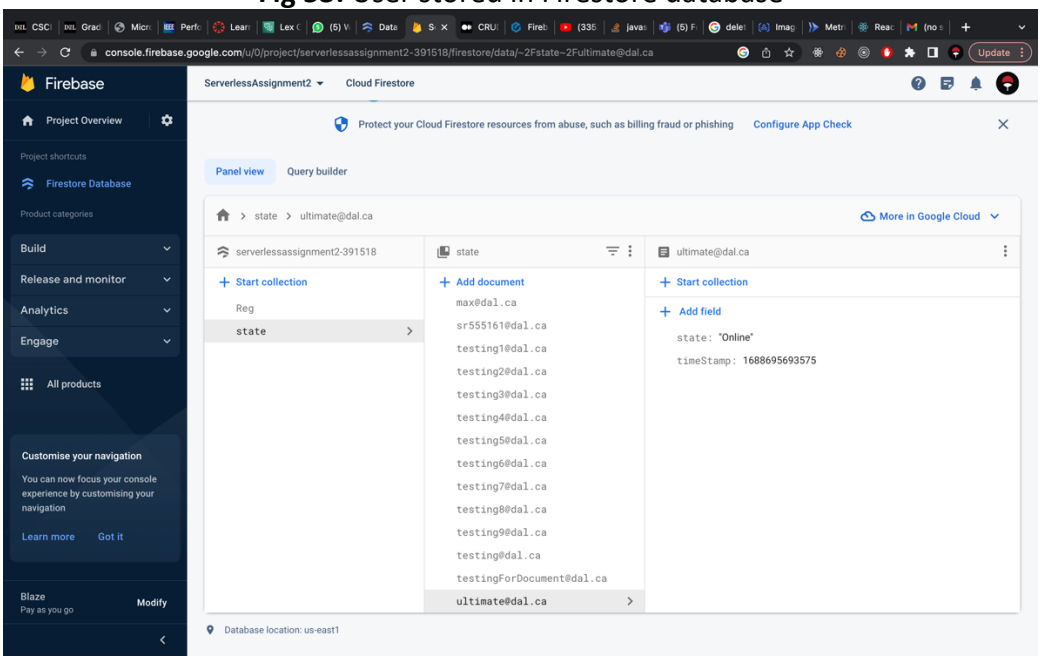

Fig 54: After logging the user is online

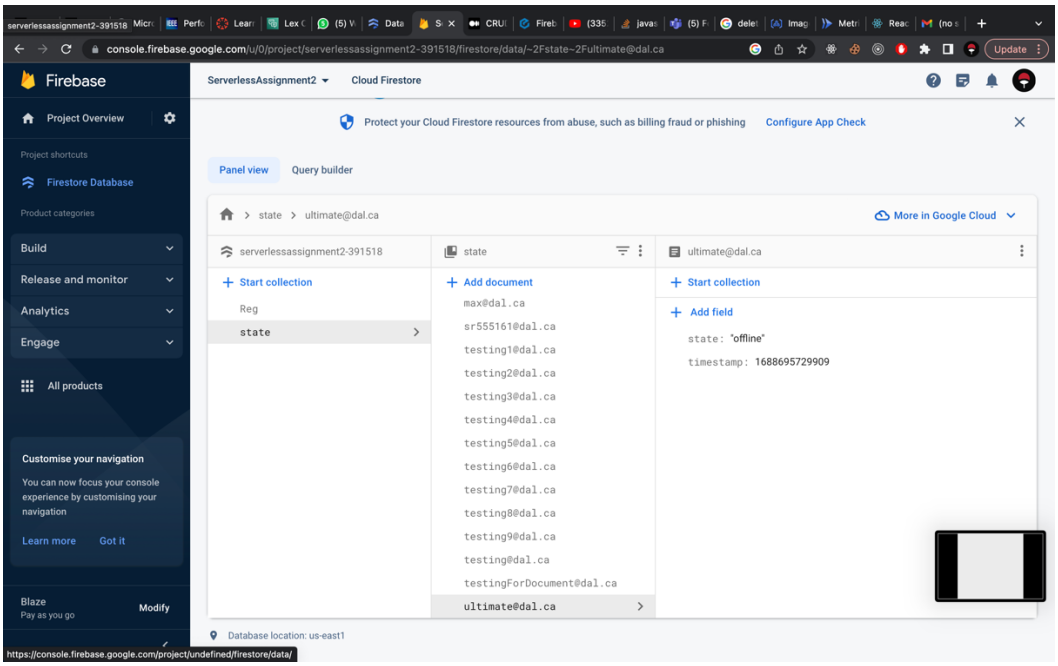

Fig 55: After logout the user is offline