Hands-on Lab: Build and Deploy the Website

Estimated time needed: **30** minutes

Introduction

In this lab, you will integrate and deploy the website.

Objectives

After completing this lab, you will be able to:

- Integrate all existing functionality
- Update your GitHub repository
- Build the website in Production mode
- Deploy the website in the Skills Network lab environment

Prerequisites

- You should have completed the prerequisite courses, especially the **Developing Front-End Apps with React** course.
- You must have completed the following labs:
  - Design Website Layouts
  - Create a GitHub Repository for your Project
  - Build Static Website Layouts
  - Set up the React Environment
  - Build the Appointment Booking Component, in particular the **Appointment Booking** component must be fully functional
  - Integrate Existing Functionality
  - Build the Notification Component
  - Build the Reviews Component
  - Build the Profile Component

Exercise 1: Integrate website components

Make sure your Skills Network lab environment is properly set up with the necessary tools and services. This includes:

- Setting up a database
- Integrating all the components
- Providing other required services

  Ensure that all steps mentioned in the **Prerequisites** section are complete.

Exercise 2: Version control

1. You must push all the changes you have made in version control to manage your project's codebase.

   Refer to the **Capstone Project Reference: Git Commands** reading for details about Git command syntax and use relevant to the project.

2. Clone the entire project and test if all the functionality works as expected.

   Refer to the **Capstone Project Reference: Git Commands** reading for details about Git command syntax and use relevant to the project.

3. **Take a screenshot** of a successful appointment booking and save it as **test_success.png**.

Exercise 3: Create a Production build

1. Optimize your code for production by minifying, compiling, and bundling all necessary files.

   ▶ Click here for details on **code optimization**.

2. Set up configuration files to the Production environment using the following steps:

   - In case you exited and are re-entering the Skills Network lab environment:
     - Open a new terminal.
     - Clone your React project's GitHub repository.
   - Verify that the repository does not contain any node modules folder from the previous code.
   - Open a new terminal and execute `npm install` from the **react app root** folder.

   ▶ Click here for a **hint**.
   - Open the `package.json` file from the `root` folder and ensure the script object contents look like the json code below.

```
"scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build && rm -rdf server/build && mv build server/",
    "test": "react-scripts test",
    "eject": "react-scripts eject" }
```

Note: If there is any command that does not match, make sure to add that in the `script` object.

  - Next, open the `config.js` file and replace the value of API_URL with `<add your server-side url>`.

▶ Click here for a **hint**.

- Then, create a `build` folder for Production mode.

  1. Run `npm run build` in the **react app root** folder.
  2. It will create a build folder inside your server folder.
  3. Finally, navigate to the server folder and run `npm install` in the **server** folder. **Take a screenshot** of the build folder as **build.png**.

- Ensure the mongoDB server is active and that you have pasted the generated password in db.js. Please refer to this link for mongoDB connection.

- Replace the index.js file code with the following code.

```
const express = require('express');
const cors = require('cors');
const connectToMongo = require('./db');
const app = express();
const path = require('path');
const PORT = process.env.PORT || 8181;
// Middleware
app.use(express.json());
app.use(cors());
// Connect to MongoDB
connectToMongo();
// Routes
app.use('/api/auth', require('./routes/auth'));
app.use(express.static(path.join(__dirname, 'build')));
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'build', 'index.html'));
});
app.get('/', (req, res) => {
    res.send('Hello World!');
});
  // Start the server
app.listen(PORT, () => {
console.log(`Server is running on port http://localhost:${PORT}`);
});
```

The above code replacement for `index.js` is designed to optimize the application for Production mode. The changes made in the new code are as follows:

1. **Using the path module:** The path module joins directories, making the code more portable and platform-independent.

2. **Serving static files from the build folder:** In production mode, the application's static files are usually located in the build folder, which is the case here. The code uses `app.use(express.static(path.join(__dirname, 'build')))` to serve static files from the build folder.

3. **Adding a catch-all route:** The new code includes a catch-all route, `app.get('*', (req, res) => { res.sendFile(path.join(__dirname, 'build', 'index.html')); });`, which serves the index.html file from the build folder for any route that's not explicitly defined. This is especially beneficial for single-page applications (SPAs), which ensure proper routing even when the page is refreshed.

4. **Removing unnecessary code:** The new code removes the HTTP module import, which is unnecessary for production mode.

   By implementing these changes, the application is better prepared for Production mode, with optimized static file serving and a catch-all route for client-side routing. The changes ensure the application runs smoothly and efficiently in a Production environment.

Exercise 4: Deploy the website

In this exercise, you will only deploy client and server sides from a single side to make it in production.

1. Export your Skills Network Lab namespace by navigating to the root directory and printing it on the console.

```
MY_NAMESPACE=$(ibmcloud cr namespaces | grep sn-labs-)
echo $MY_NAMESPACE
```

2. Create a `Dockerfile` by running the following command:

```
touch Dockerfile
```

3. Open the `Dockerfile` in the editor and write the code to containerize the server application.

▶ Click here for a **sample Dockerfile**.

4. Now, you will build an image and push it to the IBM Cloud Image Registry (ICR). Perform a Docker build with the `Dockerfile` in the current directory.

```
docker build . -t us.icr.io/$MY_NAMESPACE/medical_app
```

5. Next, push the image to the container registry.

```
docker push us.icr.io/$MY_NAMESPACE/medical_app
```

6. Next, run the image in the container registry.

```
docker run -p 8080:3000 us.icr.io/$MY_NAMESPACE/medical_app
```

7. Then, click the **Web Application** button.

Web Application

8. **Take a screenshot** of the landing page and save it as **launch.png**.

Note: If the web application does not launch when you click the **Web Application** button, you can manually launch it through **Skills Network Toolbox –> Launch Application** and specify the port number **8080**.

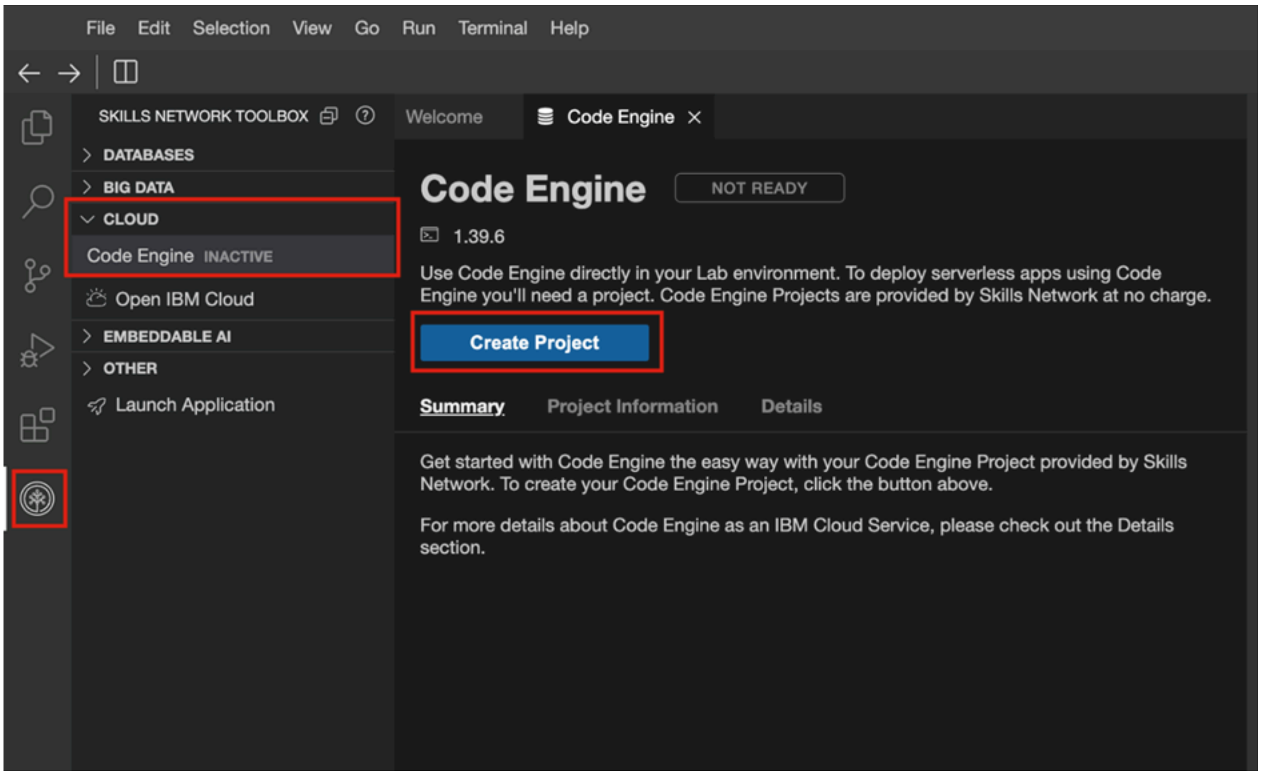
[Optional] Exercise 5: Deploy your application to Code Engine

Code Engine is a cloud-based service that allows you to run and manage your applications and workloads in a scalable and efficient manner. It includes a mechanism for creating container images from your source code, called a build, and a feature for one-time execution of your code, called a job. These entities and applications are grouped in a project, providing a namespace and access control for its entities. This allows you to easily organize, manage, and monitor your resources in Code Engine. It provides several benefits, including:

- **Easy deployment:** Code Engine makes it easy to deploy and manage your applications, regardless of their size or complexity.
- **Scalability:** Code Engine automatically scales your applications up or down based on demand, ensuring they are always available and responsive.
- **Integration:** Code Engine integrates with other IBM Cloud services, making building and deploying complex, multiservice applications easy.

Deploy an app using Code Engine

1. Start Code Engine by creating a project.



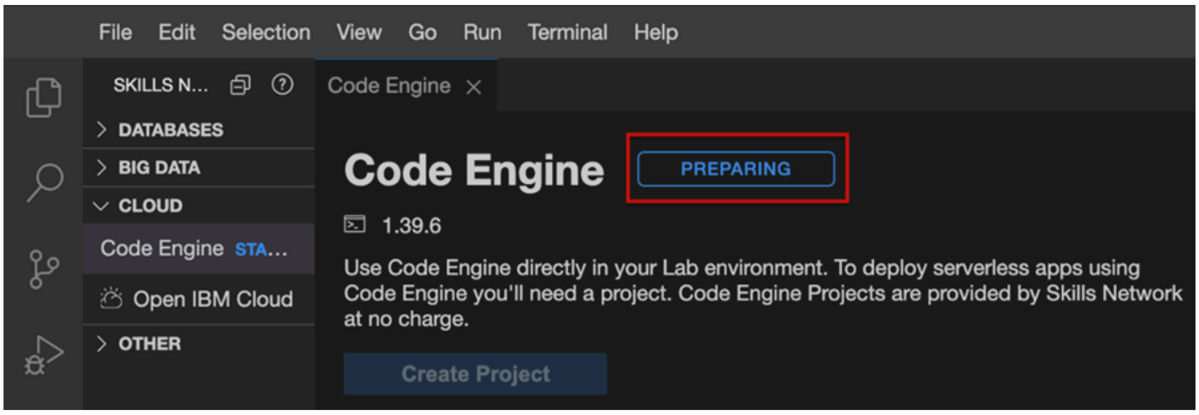
Code Engine

Use Code Engine directly in your Lab environment. To deploy serverless apps using Code Engine you'll need a project. Code Engine Projects are provided by Skills Network at no charge.

Get started with Code Engine the easy way with your Code Engine Project provided by Skills Network. To create your Code Engine Project, click the button above.

For more details about Code Engine as an IBM Cloud Service, please check out the Details section.

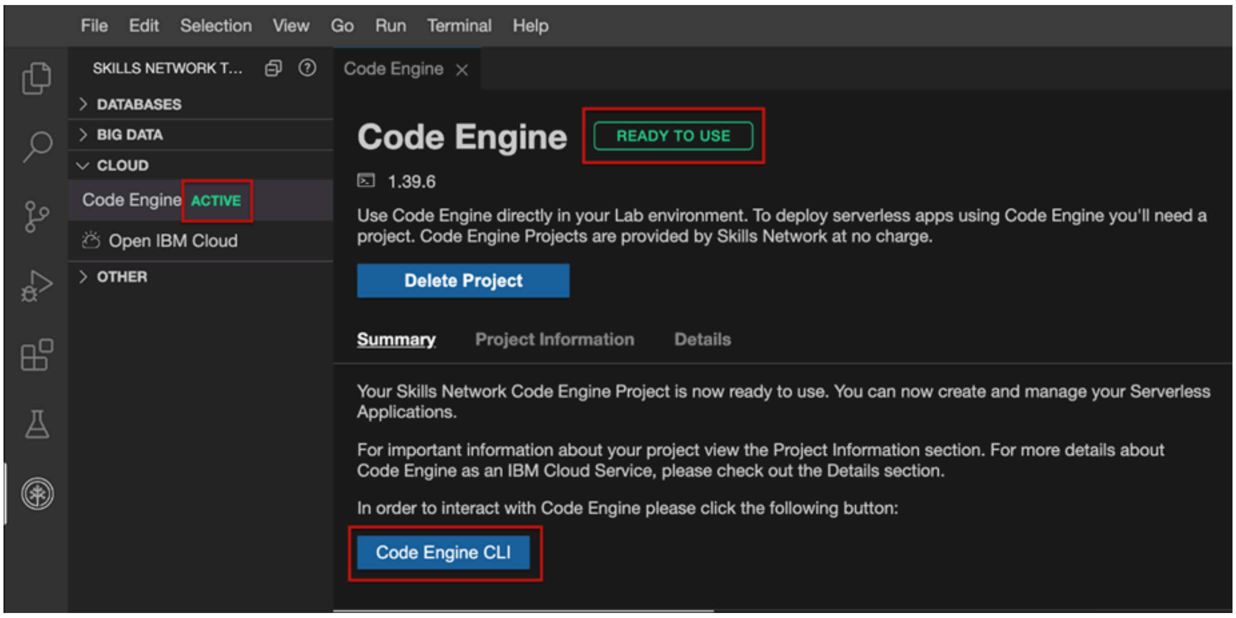
2. The Code Engine environment takes a while to prepare. You will see the progress status indicated in the set up panel.



Code Engine

Use Code Engine directly in your Lab environment. To deploy serverless apps using Code Engine you'll need a project. Code Engine Projects are provided by Skills Network at no charge.

3. Once the Code Engine setup is complete, it is active. Click `Code Engine CLI` to begin the preconfigured CLI in the following terminal.



Code Engine

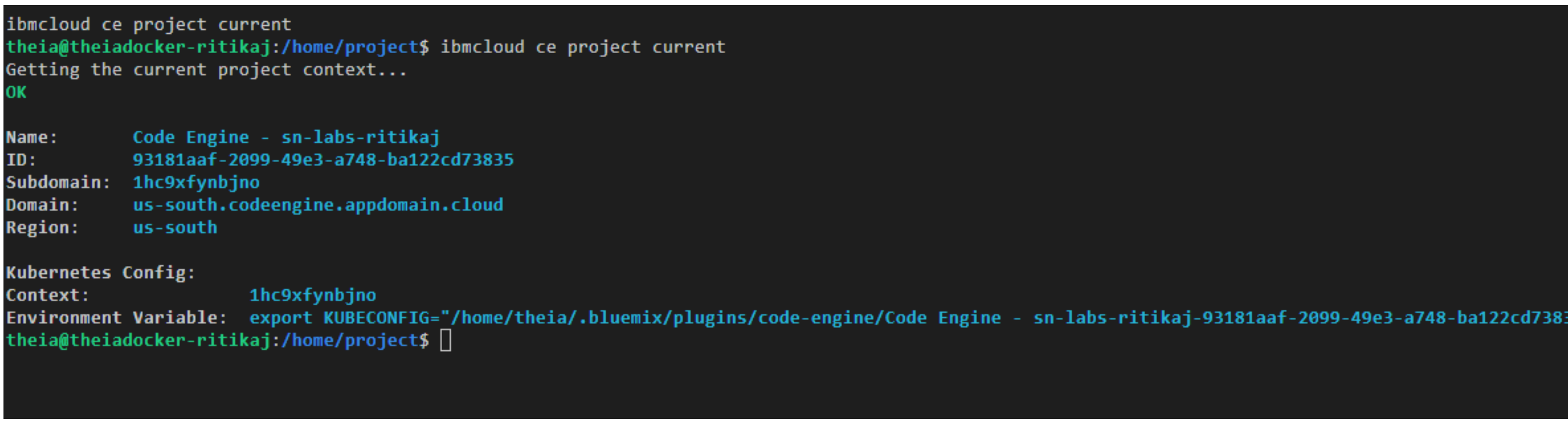
Use Code Engine directly in your Lab environment. To deploy serverless apps using Code Engine you'll need a project. Code Engine Projects are provided by Skills Network at no charge.

Your Skills Network Code Engine Project is now ready to use. You can now create and manage your Serverless Applications.

For important information about your project view the Project Information section. For more details about Code Engine as an IBM Cloud Service, please check out the Details section.

In order to interact with Code Engine please click the following button:

4. You will observe that the preconfigured CLI startup and the home directory are set to the current directory. The project and Kubeconfig have been set up as part of the configuration. The details are shown on the terminal.

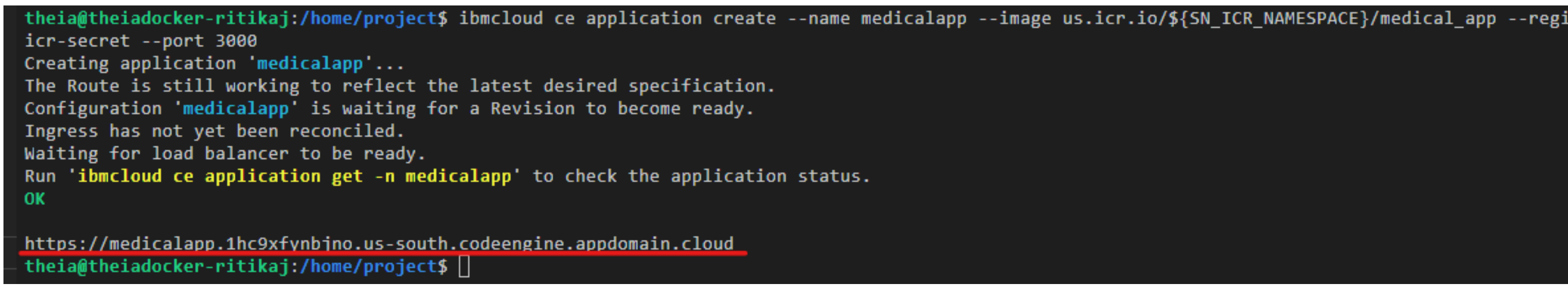


```
ibmcloud ce project current
theia@theiadocker-ritikaj:/home/project$ ibmcloud ce project current
Getting the current project context...
OK

Name:        Code Engine - sn-labs-ritikaj
ID:          93181aaf-2099-49e3-a748-ba122cd73835
Subdomain:   1hc9xfynbjno
Domain:      us-south.codeengine.appdomain.cloud
Region:      us-south

Kubernetes Config:
Context:               1hc9xfynbjno
Environment Variable:  export KUBECONFIG="/home/theia/.bluemix/plugins/code-engine/Code Engine - sn-labs-ritikaj-93181aaf-2099-49e3-a748-ba122cd738
theia@theiadocker-ritikaj:/home/project$ 
```

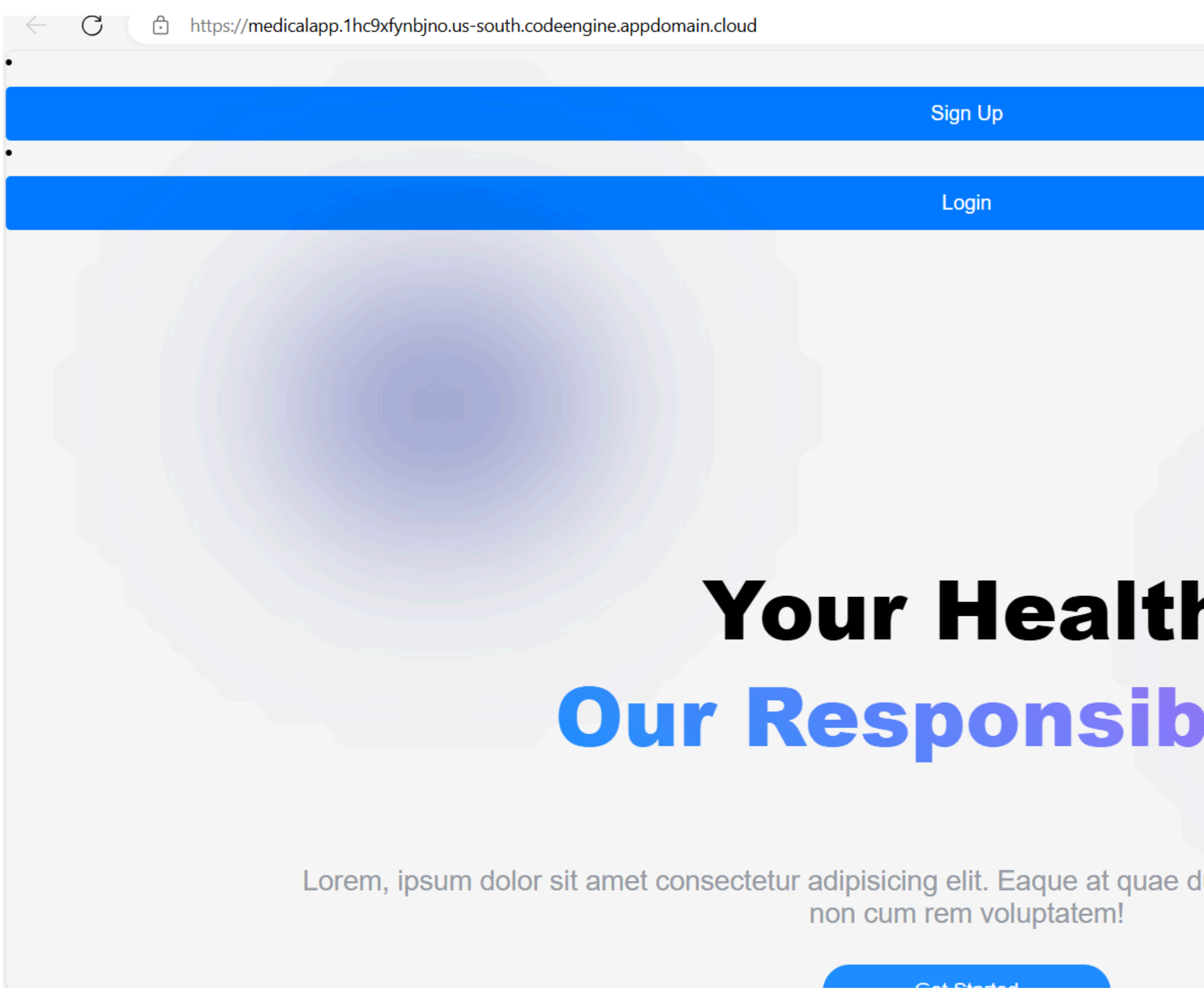
5. Deploy the `medicalapp` application on Code Engine (This will take a while.)

```
ibmcloud ce application create --name medicalapp --image us.icr.io/${SN_ICR_NAMESPACE}/medical_app --registry-secret icr-secret --port 3000
```

6. Connect to the generated URL to access the website and check if the deployment is successful.



```
theia@theiadocker-ritikaj:/home/project$ ibmcloud ce application create --name medicalapp --image us.icr.io/${SN_ICR_NAMESPACE}/medical_app --regi
icr-secret --port 3000
Creating application 'medicalapp'...
The Route is still working to reflect the latest desired specification.
Configuration 'medicalapp' is waiting for a Revision to become ready.
Ingress has not yet been reconciled.
Waiting for load balancer to be ready.
Run 'ibmcloud ce application get -n medicalapp' to check the application status.
OK

https://medicalapp.1hc9xfynbjno.us-south.codeengine.appdomain.cloud
theia@theiadocker-ritikaj:/home/project$ 
```



https://medicalapp.1hc9xfynbjno.us-south.codeengine.appdomain.cloud

- Sign Up
- Login

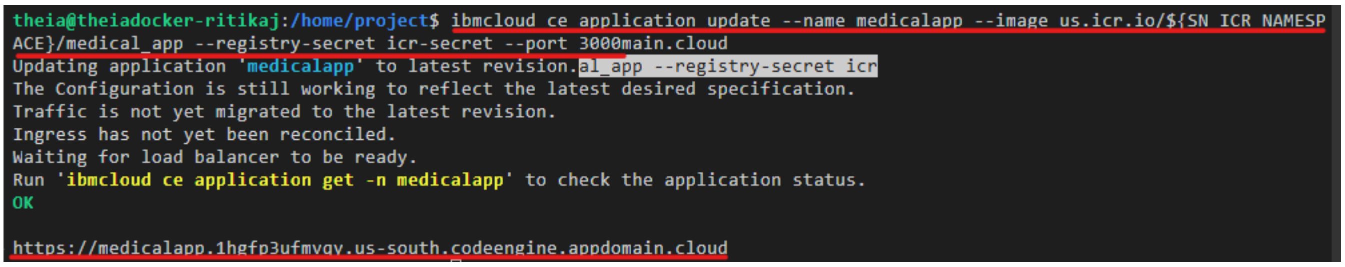
Your Health

Our Responsib

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Eaque at quae d
non cum rem voluptatem!

7. Similarly, you can also update the application using the command `ibmcloud ce application update` whenever you build a Docker image again.

```
ibmcloud ce application update --name medicalapp --image us.icr.io/${SN_ICR_NAMESPACE}/medical_app --registry-secret icr-secret --port 3000
```



```
theia@theiadocker-ritikaj:/home/project$ ibmcloud ce application update --name medicalapp --image us.icr.io/${SN_ICR_NAMESP
ACE}/medical_app --registry-secret icr-secret --port 3000main.cloud
Updating application 'medicalapp' to latest revision.al_app --registry-secret icr
The Configuration is still working to reflect the latest desired specification.
Traffic is not yet migrated to the latest revision.
Ingress has not yet been reconciled.
Waiting for load balancer to be ready.
Run 'ibmcloud ce application get -n medicalapp' to check the application status.
OK

https://medicalapp.1hgfp3ufmvqv.us-south.codeengine.appdomain.cloud
```

Screenshot checklist

You should have taken the following screenshots as part of this lab:

- *test_success.png*
- *build.png*
- *launch.png*

Note about data management and persistence

To ensure the proper management and persistence of your data in a GitHub repository, it is crucial to follow a It is crucial to follow a few essential steps to ensure the proper management and persistence of your data in a GitHub repository:

- **Regular updates:** Whenever you make changes or add new components to your project, adding, committing, and pushing the updates to your GitHub repository is essential. This ensures that your latest work is safely stored and accessible to collaborators.

- **Session persistence:** During an active session, your data remains accessible. However, it's important to note that if your session expires or you log out, you must clone the repository again to resume work.

- **Ignoring node modules:** When pushing data to GitHub, it's best practice to exclude the node modules folder from both your server and client directories. This folder contains external dependencies and can be large, making the repository heavy and slowing down the process. Adding the folder to the .gitignore file prevents it from being pushed to the repository, keeping your commits cleaner and more focused.

By adhering to these guidelines, you can maintain a well-organized and efficient GitHub repository, ensuring your work is securely stored and easily accessible to you and your collaborators.

Author(s)

Richa Arora

Other Contributor(s)

Ritika Joshi