D424 – SOFTWARE CAPSTONE
TASK 4

Capstone Proposal Project Name: Employee Management System (EMS)

Student Name: Bradley Summers

TABLE OF CONTENTS

DEPLOYMENT DOCUMENTATION

A1. CLOUD DEPLOYMENT STRATEGY AND PROVIDER JUSTIFICATION

For the deployment of the Employee Management System (EMS) application, I selected a combination of cloud service providers that best support the needs of a Minimum Viable Product (MVP) while minimizing cost. The application was deployed using Neon.tech for the database, Render.com for both the backend and frontend services, and Cloudflare for domain management and DNS configuration.

Neon.tech was chosen for its PostgreSQL hosting capabilities on the free tier, which offers sufficient performance and storage for early-stage applications. Its modern developer experience, serverless architecture, and compatibility with standard PostgreSQL clients made it an efficient and reliable choice.

The backend and frontend services were both deployed on Render.com. This platform provides a straightforward deployment experience with support for Docker images and static sites, making it ideal for full stack applications. While the backend could technically be hosted on Render's free plan, I opted for the Starter tier because the free plan spins down after periods of inactivity. This behavior would have negatively impacted the application's responsiveness during demonstrations, so maintaining uptime was prioritized. For the frontend, the free static site hosting tier was used, as it met performance requirements without any limitations that would affect user experience.

I already had the domain bradleysummers.dev registered through Cloudflare before starting this project, so I used it to deploy the application at ems.bradleysummers.dev.

This deployment strategy was intentionally designed around free-tier offerings and ease of use, making it an ideal solution for early-stage development. However, it remains flexible enough to scale or migrate to other cloud platforms as the application grows in complexity or demand.

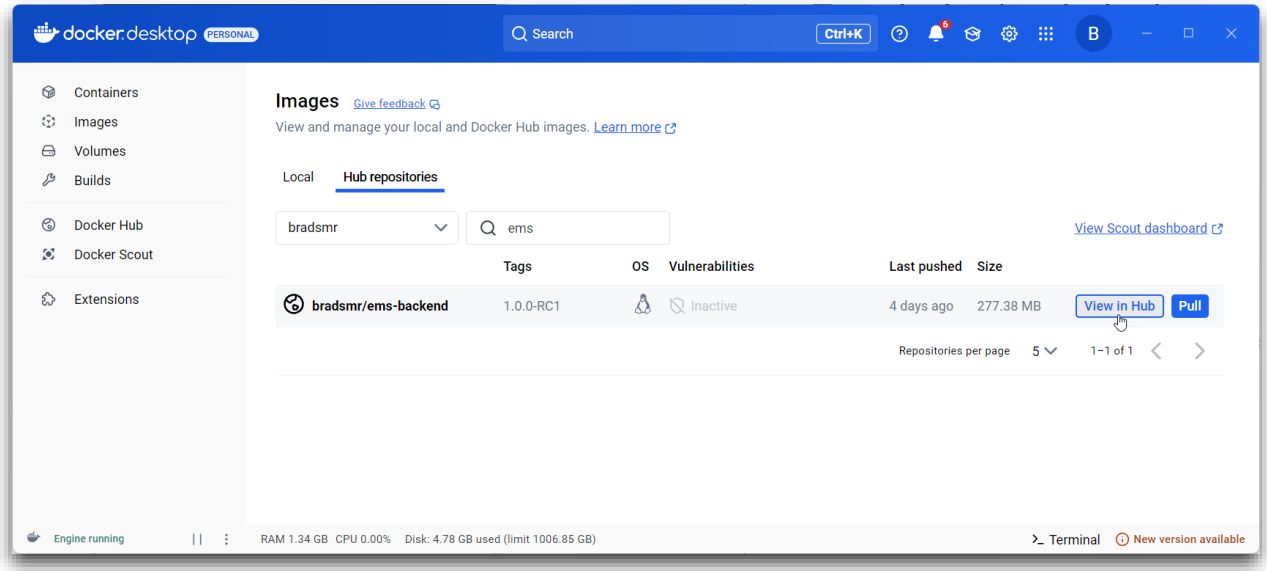
A2. CONTAINERIZATION APPROACH AND IMPLEMENTATION DETAILS

The EMS backend application was implemented using a container-based deployment strategy. I used Docker to create a consistent runtime environment for the backend code, ensuring that the application would behave identically across different environments and cloud services.

A Dockerfile was created in the root directory of the backend project. This file specified the base image, dependencies, and build instructions required to run the application. After defining the Dockerfile, I built the image locally to verify that the backend started correctly and performed as expected in a containerized environment.

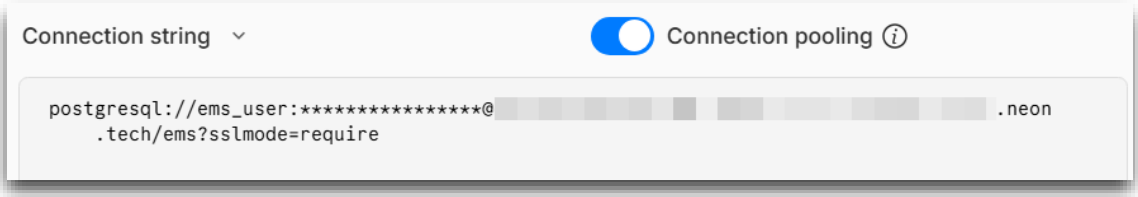
```dockerfile
# Build stage
FROM maven:3.9.5-amazoncorretto-21 AS build

WORKDIR /app

COPY pom.xml .
RUN mvn dependency:go-offline

COPY src ./src
RUN mvn clean package -DskipTests

# Runtime stage
FROM amazoncorretto:21

WORKDIR /app

COPY --from=build /app/target/ems-1.0.0-RC1.jar /app/ems.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "/app/ems.jar"]
```

Once the image was validated, I pushed it to Docker Hub using the appropriate tagging and authentication steps. This allowed me to reference the image directly from Render.com during the deployment process.

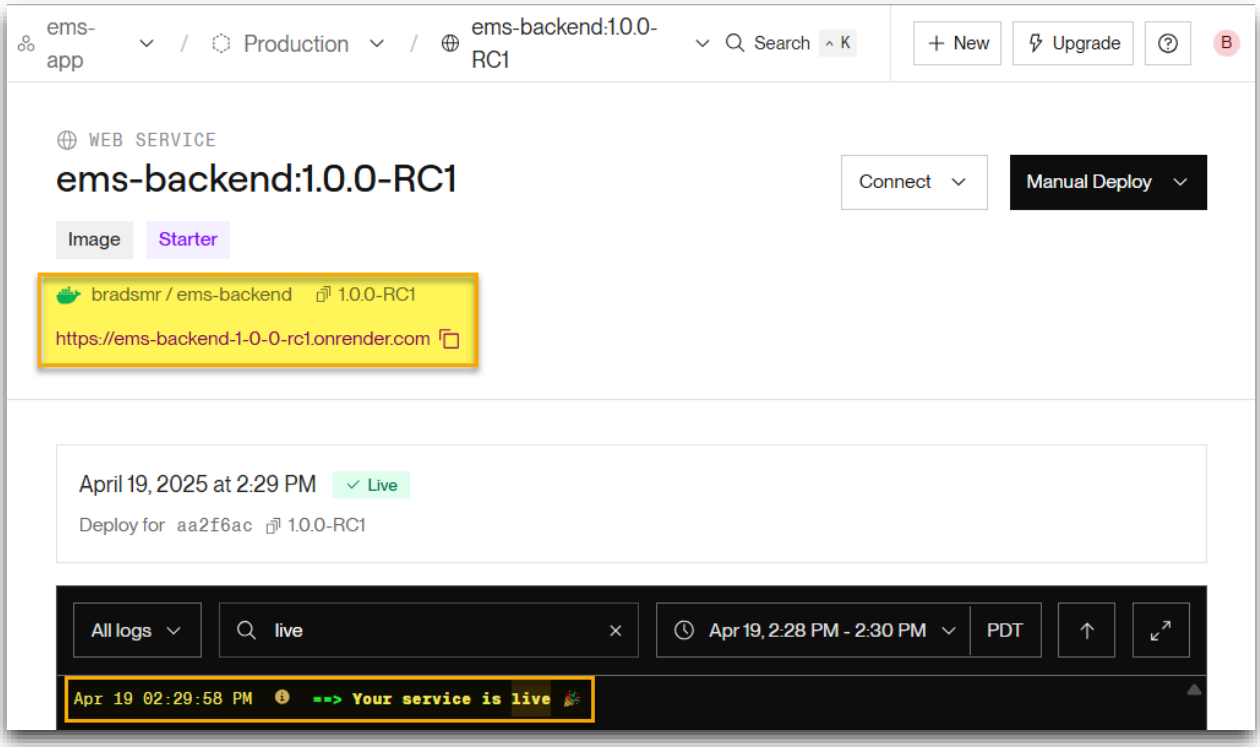


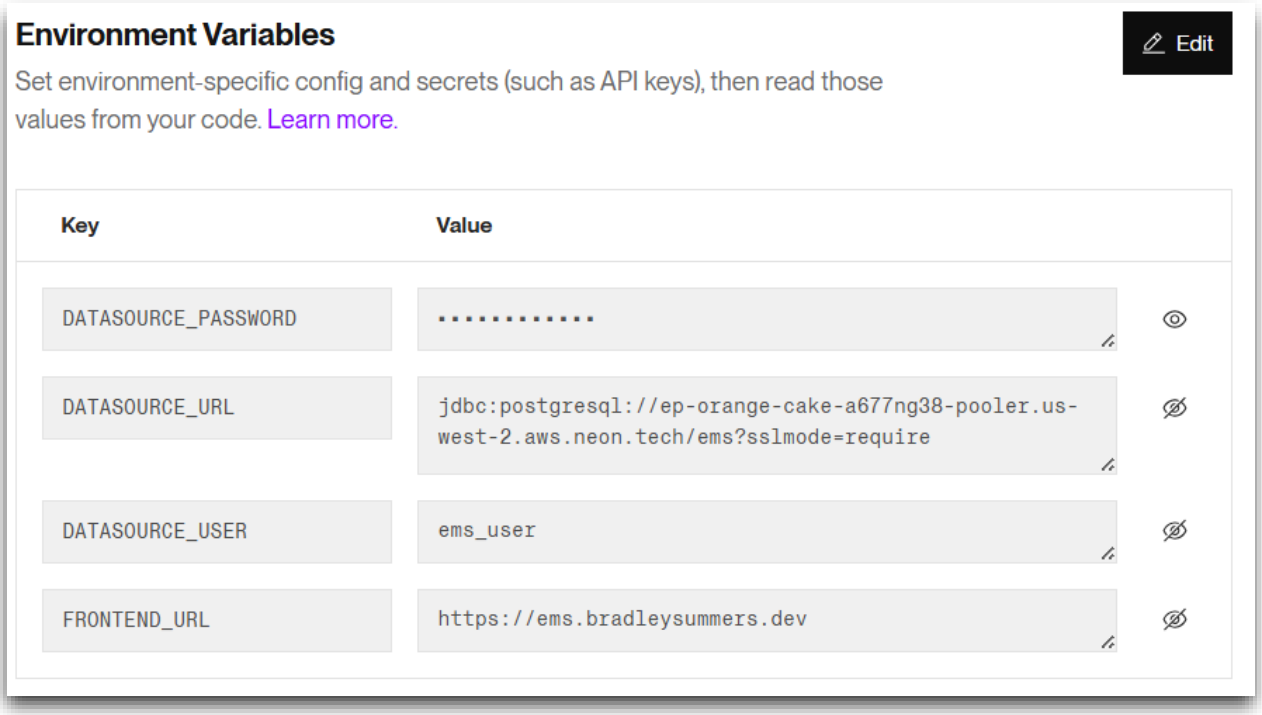
Before deploying the backend from this container image, I deployed the cloud database using the free-tier PostgreSQL offering from Neon.tech. I created a new project and database instance, then set up a dedicated user with appropriate privileges. Afterward, I generated a connection string, which was later used as an environment variable in the backend service to enable secure and persistent communication between the application and the database.

```
GRANT ALL ON SCHEMA public TO ems_user;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO ems_user;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO ems_user;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL PRIVILEGES ON TABLES TO ems_user;
```

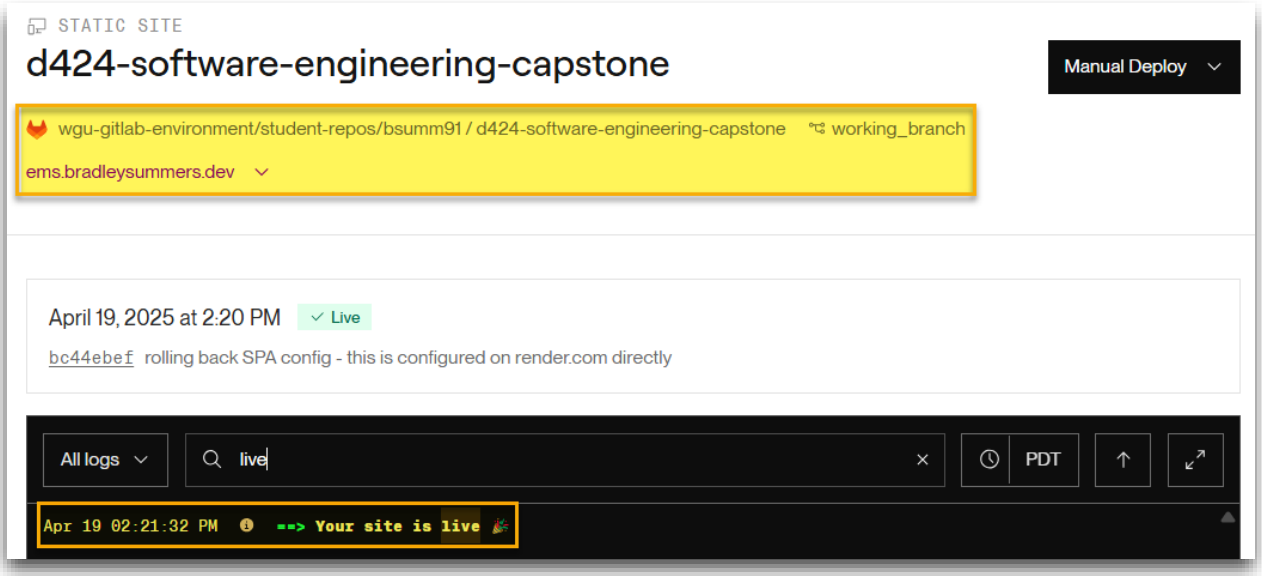


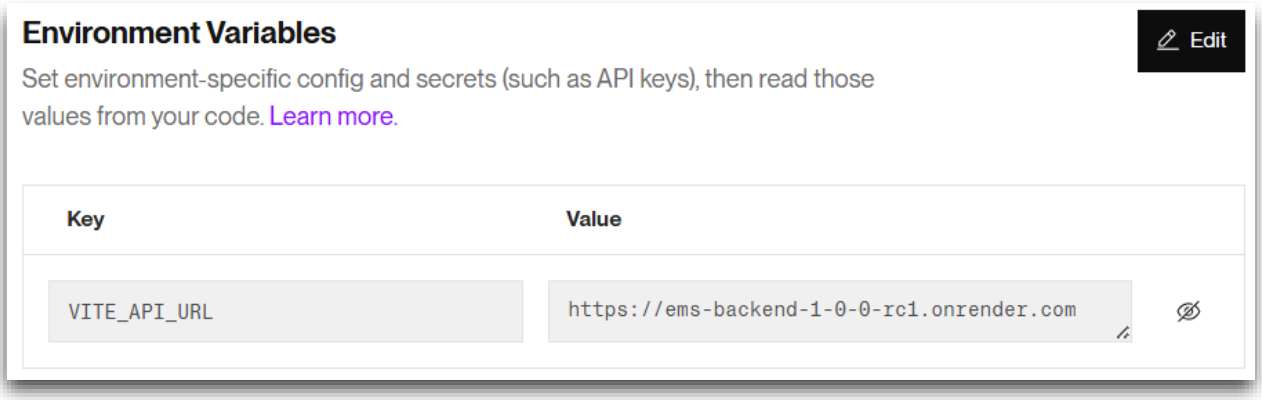
Render.com supports deploying backend services directly from container registries. I created a new Web Service on Render using the “Deploy an existing image from a registry” option. The service was named ems-backend, and I selected a deployment region close to the database to reduce latency. I also specified the image URL from Docker Hub and configured the necessary environment variables, including the database connection string noted earlier.

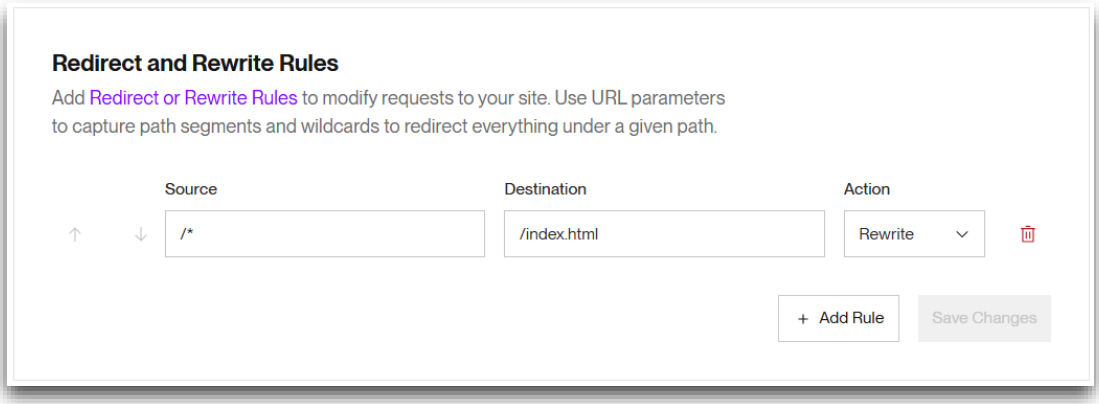




After the backend was successfully deployed, I used Render.com again to deploy the frontend. The frontend project was configured as a static site linked to the GitLab repository's working_branch. Build commands and output directories were set to generate the production version of the site from the /frontend directory, and the previously noted backend URL was added as an environment variable so the frontend could connect to the backend. I also configured a rewrite rule to support proper routing for the single-page application.







To complete the deployment, I added a custom domain and pointed it to the frontend using Cloudflare. A CNAME record was created in the DNS settings, and Render's dashboard was used to verify and apply the domain. Once verified, HTTPS was automatically enabled through Render's TLS support.

```
ems.bradleysummers.dev.  1  IN  CNAME  d424-software-engineering-capstone-tzo0.onrender.com. ;
cf_tags=cf-proxied:true
```

After deployment, the application presents a setup screen to the first user who accesses the site. This setup process guides the user through creating an administrator account, after which the system transitions to a standard login page for all future users. This initial setup only occurs once and ensures that administrative access is securely established from the beginning.