

# Dockerized web stack application

Paul Brewer

## Objectives

The aim of the project was to develop a containerized web application which can be used to store data sent in from users who visit the website (first name, last name, location). This project allowed me to develop hands on experience setting up, configuring and developing within different libraries, within a server environment, before finally deploying them as a multi-container application

This is a similar architecture to many modern applications, known as the microservice architecture, where an application is split into many different smaller applications, where each service runs in its own environment. This minimises dependencies in large applications, making changes to individual services easier. It also makes the application more reliable, since services experience minimal downtime due to container orchestration tools having the ability to spin up copy containers automatically.

## Solution

The final solution is a multi-container application, which can be spun up using the 'docker compose up', so long as the user has 'docker' and 'docker-compose' installed and running on their machine. One container acts as a web frontend, another container runs the application code and inserts data from the form into the database, with the final container running the database.

To do this, the following libraries were used:

**NGINX** – Used for 2 reasons:

1. Act as a reverse proxy. When a user visits the NGINX reverse proxy, which is configured to listen on ports 80 and 443 of localhost, they are forwarded to port 3000 which is where the Node app listens. This information however is hidden from the user, meaning that incoming traffic can't access the Node application directly, increasing security. For applications with large amounts of traffic, this can be extended to act as a load balancer for an application running on multiple servers to ease load between servers. To do this, the 'nginx.conf' file was appended to pass traffic from NGINX to port 3000 of the application container. When creating the NGINX container, the default NGINX configuration had to be disabled to allow the new NGINX configuration to work properly. They were then sent into the container via the 'COPY' command in the NGINX Dockerfile.
2. Provides a secure connection. SSL certificates were generated to allow the app to be connected to via HTTPS, allowing for data transfer between the client and server to be encrypted, for increased security.

## NodeJS (Along with Express, Pug, Mongoose)

The application was developed using node, which allows JavaScript to run in browsers. Express was used for routing traffic between pages of the website, and Pug was used to design the forms of the website, allowing for easy development. The application listens on port 3000 and is designed to read in text from the user, validate it and store it in a database, for which MongoDB was used. The app can also retrieve information from the database which can be viewed in the /registrations page of the website. The Node library, 'Mongoose' was used for communicating between the node app and the database.

## MongoDB

Mongo was the database engine used for storing and sending data to and from the app and listens on port 27017.

## Docker (and Docker-compose)

These were installed on a Virtual Box CentOS 8 instance, intended to replicate a server environment. The 3 containers were spun up in 1 command, using a 'docker-compose.yml' file which is used to configure and spin up all 3 containers at the same time. Docker compose manages communication between docker containers using networks automatically, so that Docker containers reference each other using their service name.

### The process

An instance of the CentOS 8 distribution was downloaded in VirtualBox, in order to replicate the server environment. After this was downloaded and configured, the following port forwarding rule was set up so that I could connect to the virtual machine via SSH from elsewhere (Ubuntu 18.04 LTS) on the host machine by running the command 'ssh [pbrewer@127.0.0.1](#) -p 2222' inside the terminal.

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Rule 1 - VM	TCP	127.0.0.1	2222	10.0.2.15	22

This forwards port 22 (SSH port) on the VM to port 2222 on the host machine.

The application was then configured to run on the virtual machine, with all of the libraries being installed through the use of the 'yum' package manager apart from MongoDB. Since MongoDB wasn't available through 'yum', it was manually downloaded and added to the system repos in '/yum/repos.d' and then installed from there. I first got NGINX working to run the default page, followed by the node application. I then changed the NGINX configuration to point the node application, and then connected the node application to communicate with the MongoDB database. Finally, I began to put each service into a Dockerfile separately and configure it using docker-compose.yml, and then finally worked on making them communicate. This mainly involved copying the configuration files which I was sending through to the containers and changing references to 'localhost' to the name of the service running the application. Dockerizing the app allows each service to be reused within different stacks, for example, the NGINX frontend can be used to send users to a different app, as long as it is named 'app' within the services section within the docker-compose.yml file. Any user machine running docker can spin up the project and host it from their machine. In future, the app can be given a domain name and hosted through a hosting service.

## Challenges

The first problem I experienced was with the MongoDB inside the VM, where running 'mongo' within the terminal in order to bring up the 'mongo' terminal would bring up errors involving child processes not being able to run due to permission problems. One of the ways I attempted to overcome this by using 'sudo chmod 777' on folders which the child processes were trying to access, but as this didn't work and seemed to cause other problems, I had to restore the VM back to an earlier VirtualBox snapshot from before I downloaded and installed MongoDB onto the system.

Another challenge I faced was developing an understanding of NodeJS and the workflow / folder structure you need to follow to develop your own website. This was an exciting challenge and I'm looking forward to doing more projects with Node in my spare time.

I encountered a problem where I was unable to use the 'docker-compose up' command and learnt that this was because my user wasn't currently added to the 'docker' user group, which helped me to learn about users, groups and permissions.

I also learnt a lot regarding how docker and docker compose works during the project due to the problems I ran into. For example, when dockerizing the application, I noticed the Dockerfile 'COPY' commands failing and found out the Dockerfile can't see files outside of its own folder, had to copy them in. I also ran into problems where docker containers would automatically stop running and discovered that this was due to the daemons keeping them alive terminating due to a problem with the NGINX configuration files rather than the Docker configuration files. I had to fix this by disabling the default NGINX configuration file before starting the container, which was done in the NGINX Dockerfile.

Overall, my main challenge of the project was developing an understanding of the Linux system, in order to fix bugs. The first problem I experienced was involving the security features inside Linux, where I experienced 'firewalld' blocking user traffic from going from one page of the application to another. I eventually had to stop firewalld and disable it on startup. A similar problem occurred later on in the project, which blocked NGINX forwarding being used as a reverse proxy. After checking the logs and I found out that this was a 'permission denied' problem which led me to discovering (and disabling) SELinux.

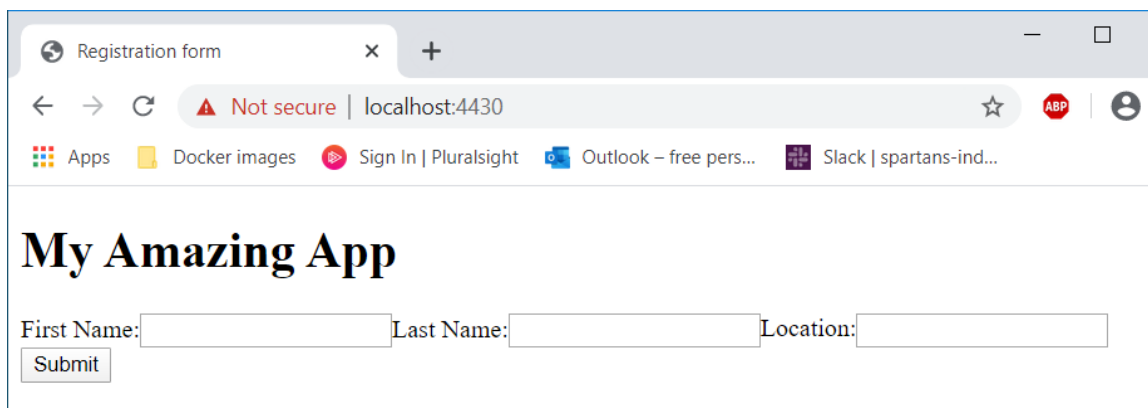
## Demonstration

```
pbrewer@localhost:~/ContainerApp
[pbrewer@localhost ContainerApp]$ ls -al
total 12
drwxrwxr-x.  6 pbrewer pbrewer 103 Apr  8 05:48 .
drwx----- 10 pbrewer pbrewer 4096 Apr  8 06:25 ..
drwxrwxr-x.  8 pbrewer pbrewer 166 Apr  8 06:24 .git
-rw-rw-r--.  1 pbrewer pbrewer  18 Apr  8 06:24 .gitignore
drwxrwxr-x.  3 pbrewer pbrewer  47 Apr  7 06:15 App
drwxrwxr-x.  2 pbrewer pbrewer  24 Apr  7 09:00 DB
drwxrwxr-x.  2 pbrewer pbrewer 150 Apr  8 05:48 ReverseProxy
-rw-rw-r--.  1 pbrewer pbrewer 367 Apr  7 11:34 docker-compose.yml
[pbrewer@localhost ContainerApp]$ docker-compose up -d
```

I have 3 subfolders in my project, each containing their own Dockerfile and with the necessary resources to be sent to the containers, such as source code for the App and configuration files for the Reverse Proxy. The 'docker-compose.yml' file allows the 3 containers to be spun up using 'docker compose up'

```
total 12
drwxrwxr-x.  6 pbrewer pbrewer 103 Apr  8 05:48 .
drwx----- 10 pbrewer pbrewer 4096 Apr  8 06:25 ..
drwxrwxr-x.  8 pbrewer pbrewer 166 Apr  8 06:24 .git
-rw-rw-r--.  1 pbrewer pbrewer  18 Apr  8 06:24 .gitignore
drwxrwxr-x.  3 pbrewer pbrewer  47 Apr  7 06:15 App
drwxrwxr-x.  2 pbrewer pbrewer  24 Apr  7 09:00 DB
drwxrwxr-x.  2 pbrewer pbrewer 150 Apr  8 05:48 ReverseProxy
-rw-rw-r--.  1 pbrewer pbrewer 367 Apr  7 11:34 docker-compose.yml
[pbrewer@localhost ContainerApp]$ docker-compose up -d
Creating network "containerapp_default" with the default driver
Creating containerapp_app_1      ... done
Creating containerapp_db_1       ... done
Creating containerapp_reverse-proxy_1 ... done
[pbrewer@localhost ContainerApp]$
```

Visiting 'localhost:4430' allows me to see the website, seen below.

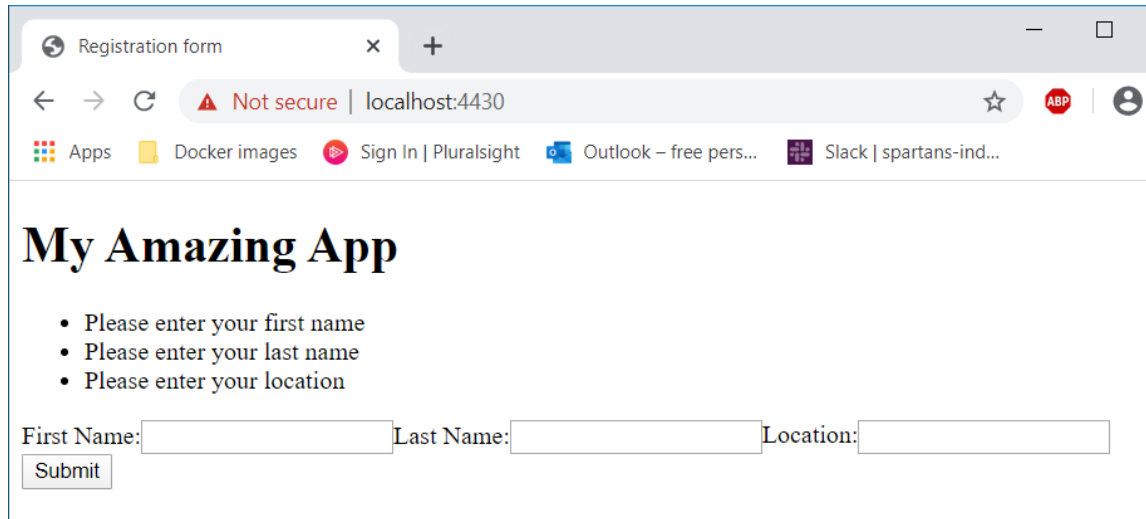


The screenshot shows a web browser window with a single tab titled 'Registration form'. The address bar indicates the URL is 'localhost:4430' and the connection is 'Not secure'. Below the address bar, there are several bookmarked sites: 'Apps', 'Docker images', 'Sign In | Pluralsight', 'Outlook - free pers...', and 'Slack | spartans-ind...'. The main content of the page features a large heading 'My Amazing App'. Below this heading is a registration form with three input fields labeled 'First Name:', 'Last Name:', and 'Location:'. A 'Submit' button is positioned below the 'First Name' field.

This is because I have set up the following port forwarding rule between my host machine and VM.

Rule 5 - DockerNGINX SSL	TCP	127.0.0.1	4430	10.0.2.15	443
--------------------------	-----	-----------	------	-----------	-----

When the submit button is pressed but a box is left empty, the page is reloaded, and the user is prompted to enter the information which they left out.



Registration form

← → ↻ Not secure | localhost:4430

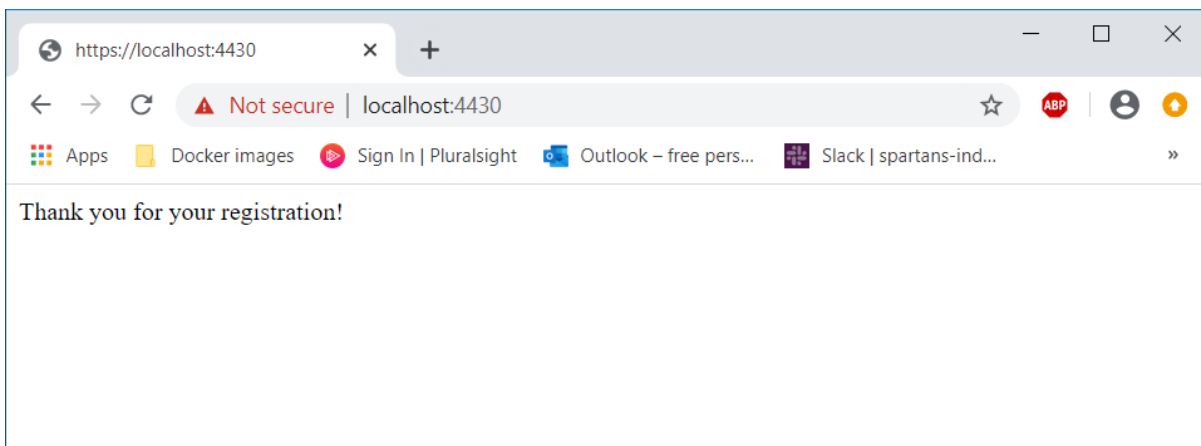
Apps Docker images Sign In | Pluralsight Outlook – free pers... Slack | spartans-ind...

## My Amazing App

- Please enter your first name
- Please enter your last name
- Please enter your location

First Name:  Last Name:  Location:

After the user submits valid data, they are taken to the following page



https://localhost:4430

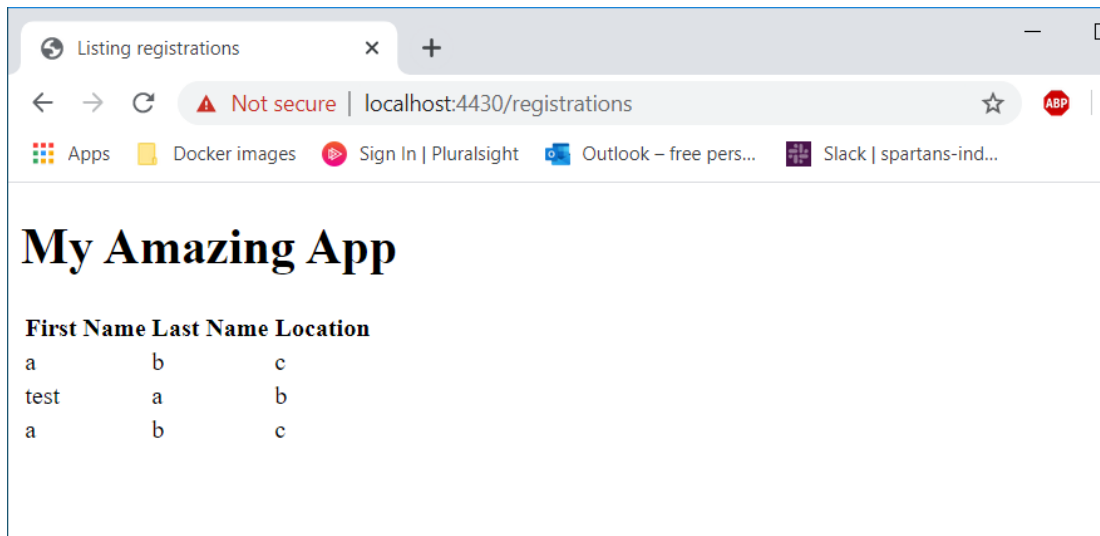
← → ↻ Not secure | localhost:4430

Apps Docker images Sign In | Pluralsight Outlook – free pers... Slack | spartans-ind...

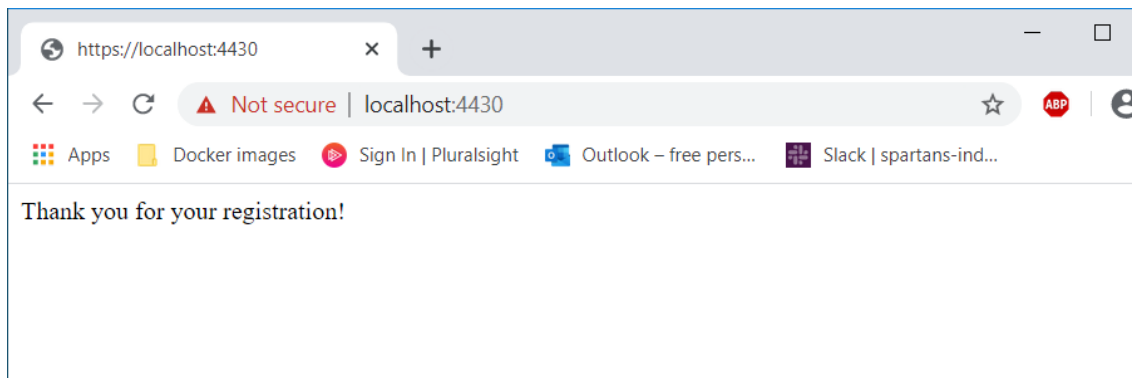
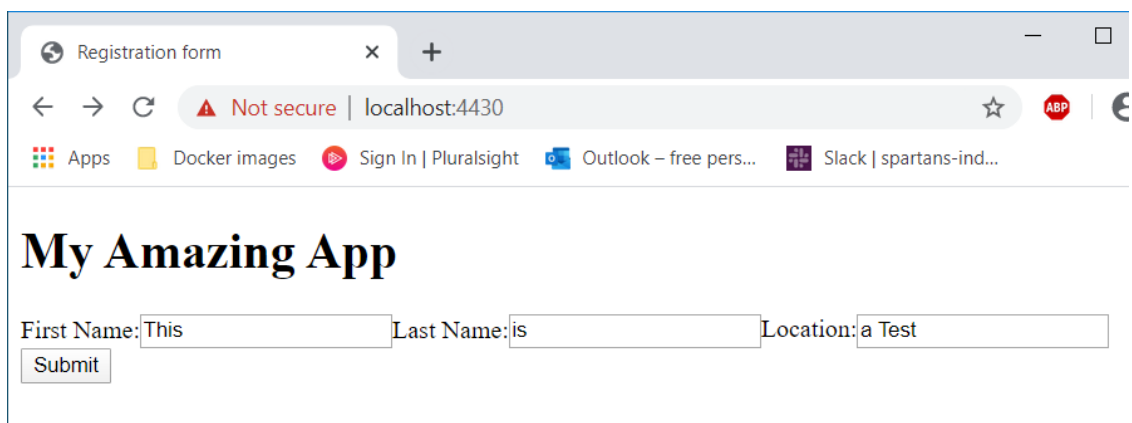
Thank you for your registration!

As you can see below, data is sent into the database when the submit button is pressed, which can be retrieved from the /registrations page

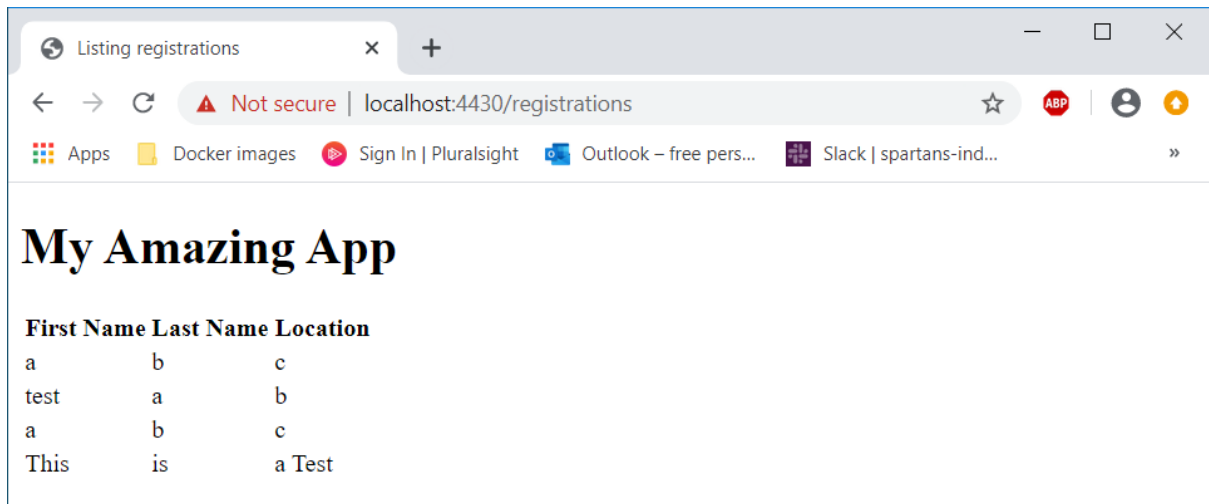
**(Before)**



Data is entered



After



Note: A docker volume was declared in the 'docker-compose.yml' file, allowing data from the database to persist on the server's file system after the containers are stopped. This initialises a folder called 'appdatabase' which receives data from the /data/db folder from inside the mongo container.

```
- 27017:27017
volumes:
- /home/pbrewer/appdatabase:/data/db
```