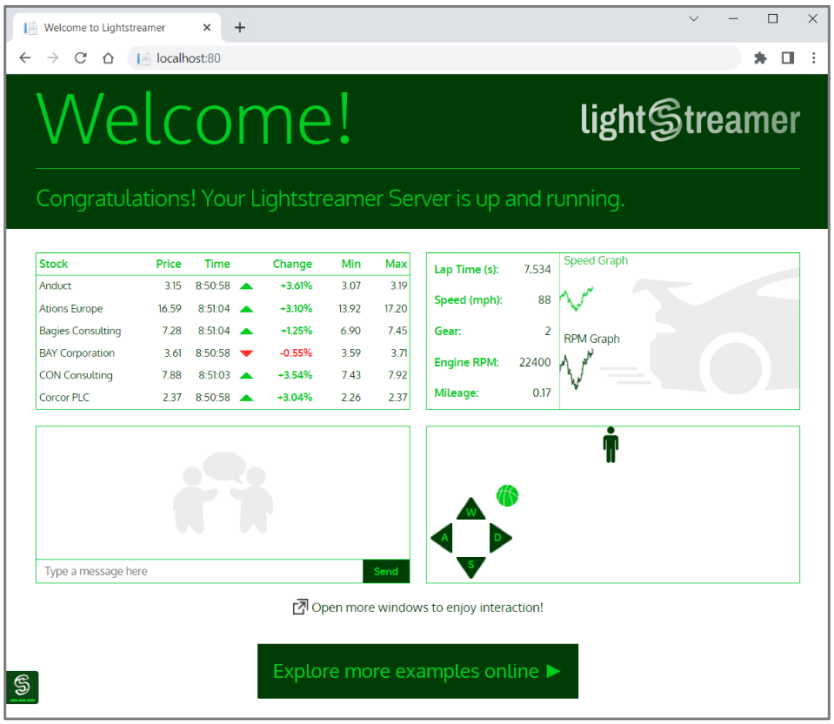
Exercises: Containers and Docker

Exercises for the "Containers and Clouds" course @ SoftUni

1. Lightstreamer Container

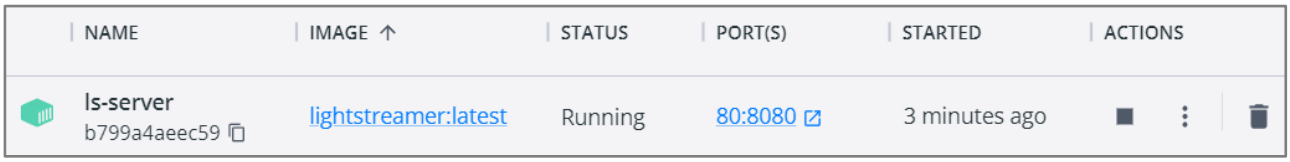
Lightstreamer (https://lightstreamer.com) is a **web-based asynchronous messaging project**.



Your task is to **run it in a Docker container**. For running the **Lightstreamer container**:

- The **image** you need is **`lightstreamer:latest`**
- Your **container's name** should be **`ls-server`**
- Server works on **port 8080**, but should be **accessed** on **`localhost:80`**
- Container should be run in **detached mode**

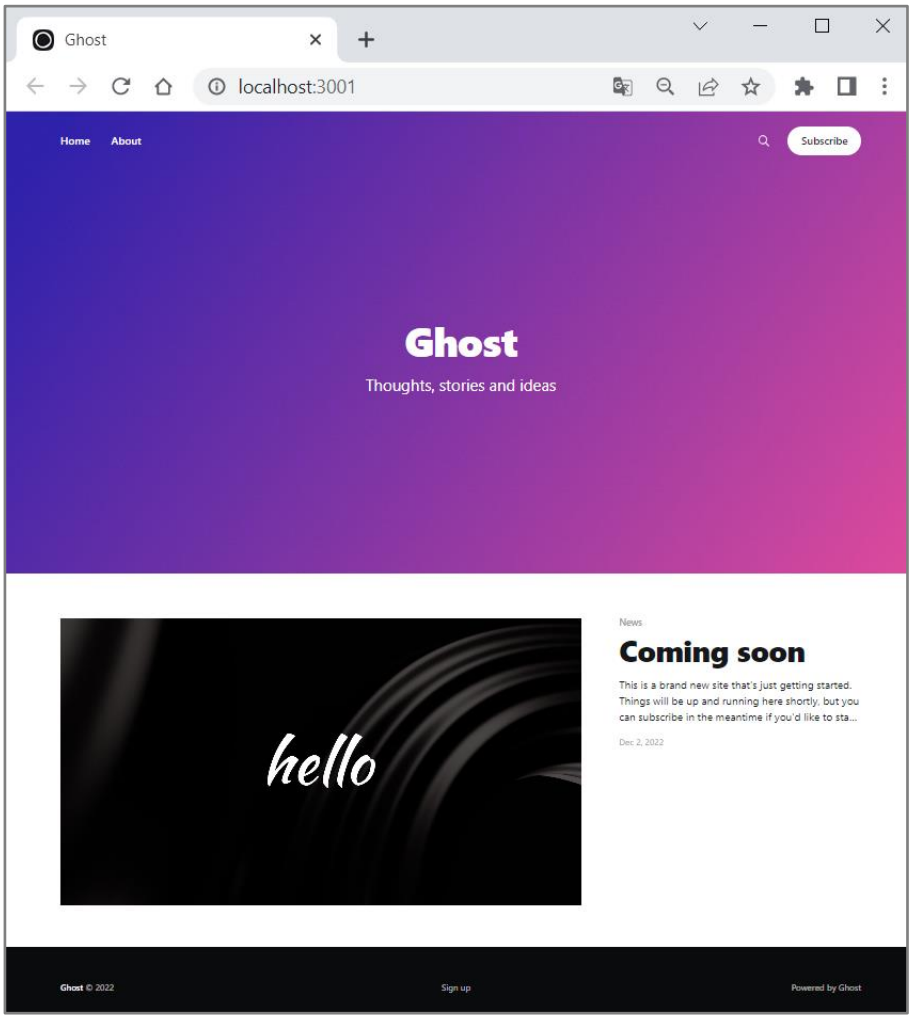
Your **container** should look like this:



Make sure your **container is created** and **Lightstreamer works in the browser**. Then you can **delete the container** and the **image**.

2. Ghost Container

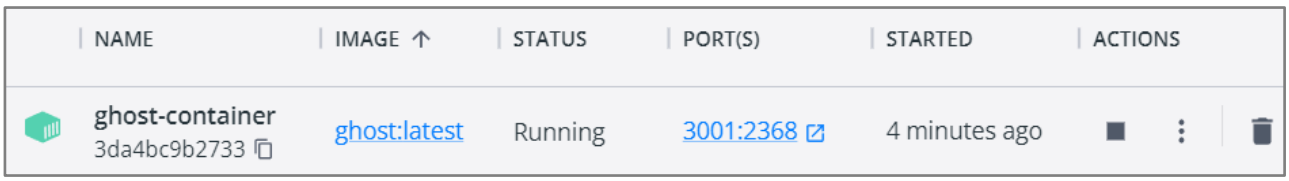
Ghost (https://en.wikipedia.org/wiki/Ghost_%28blogging_platform%29) is a free and open-source **blogging platform**, written in **JavaScript**. When **run in** a **Docker container** and **accessed in the browser**, it looks like this:



For running your **Ghost container**, follow these **requirements**:

- The **image** you need is **`ghost:latest`**
- Your **container's name** should be **`ghost-container`**
- Server works on **port 2368**, but should be **accessed** on **`localhost:3001`**
- You should set **`NODE_ENV=development`** as an **environment variable** with the **`-e` option**
- Container should be run in **detached mode**

Your **container** should look like this:



Note: if a "**`We'll be right back`**" **message** appears in the browser, it means that **Ghost is still loading**, so **refresh the browser** and everything should be alright.

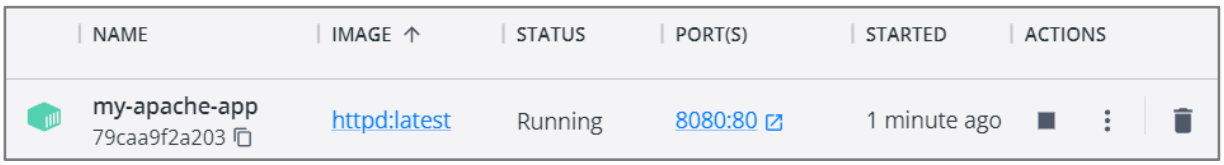
3. Apache HTTP Server Container

Now you should run **Apache HTTP Server** in a **Docker container**.

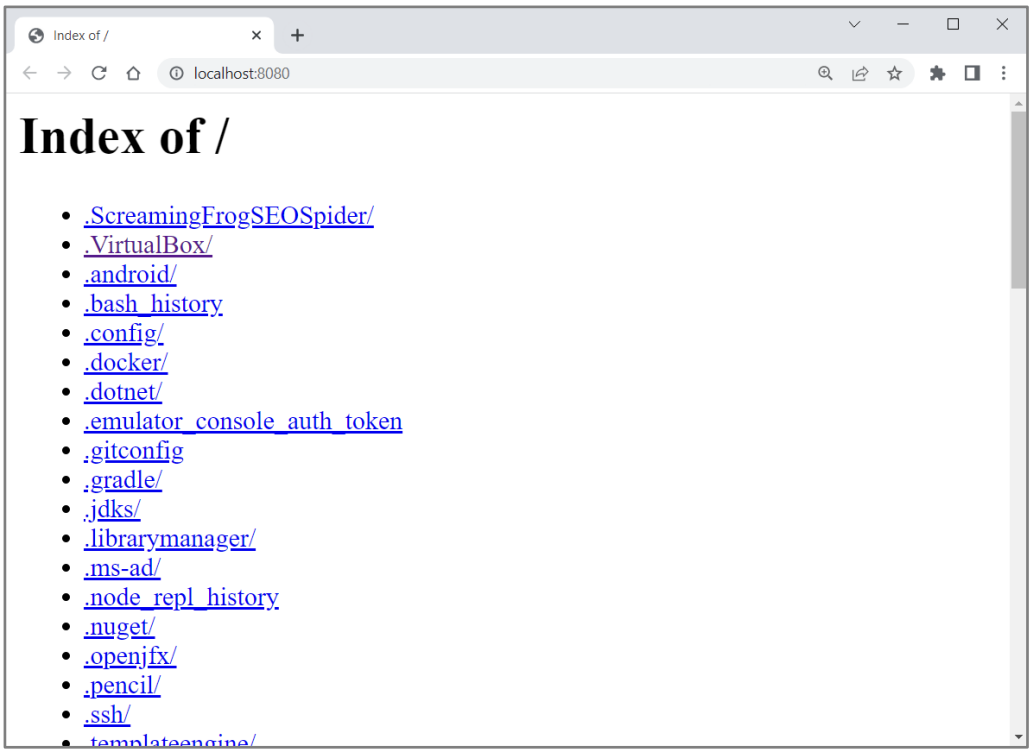
- Use the **latest image: `httpd:latest`**

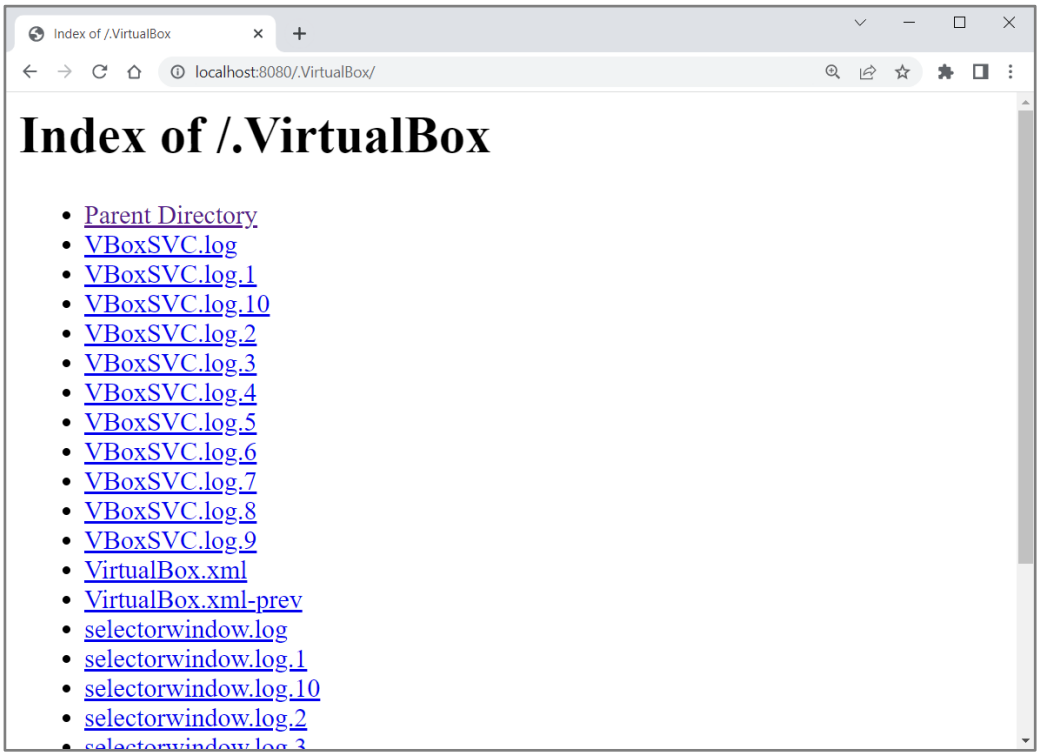
- Your **container's name** should be **`my-apache-app`**
- **Server** works on **port 80**, but should be **accessed** on **`localhost:8080`**
- Container should be run in **detached mode**
- You should create a **volume** – map **current PowerShell** (or another) **directory** to the **container's directory** **`/usr/local/apache2/htdocs/`**

Your **container** should look like this:



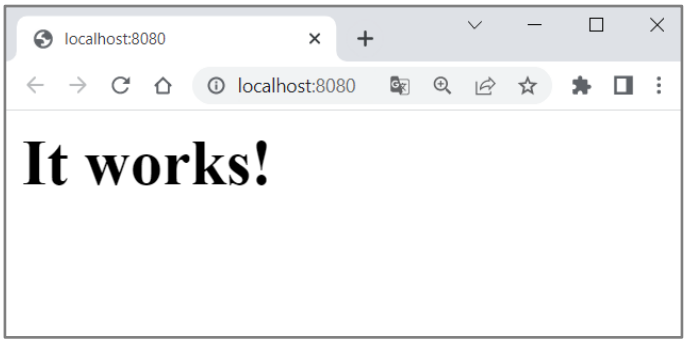
When **accessed** from the **browser**, it should list the files and folders from your local file system in the PowerShell directory you provided the server with, as well as in child directories:





The **local file system** is accessed by the **container** because of the **volume**.

However, if the browser only shows you the "**`It works!`**" **message** (see below), then you **didn't succeed in running the container properly** and you should **fix your command and try again**:



4. SQL Server Container

Our task is to **run a container** with an **SQL Server database** in it. To do this, we will need the **following image** from **Docker Hub**: https://hub.docker.com/_/microsoft-mssql-server.

You can look at the "**How to use this Image**" **section** to learn how to **run the database container**. However, we will also **show and explain** this step by step.

Create the Container

Start **writing the multi-line run command** for the **Docker container**:

```
PS C:\Users\PC> docker run `
```

Let's first take care of the **environment variables** needed for the **SQL Server container**. We should **confirm the acceptance to licensing agreement** with **ACCEPT_EULA=Y**:

```
>> -e ACCEPT_EULA=Y `
```

We should also **set a password** for the **database system administrator** (**`sa`**) to **connect to SQL Server** once the container is running:

```
>> -e MSSQL_SA_PASSWORD=yourStrongPassword12# `
```

Note: your **password** should follow the **requirements from the documentation**: "This password needs to include at least 8 characters of at least three of these four categories: uppercase letters, lowercase letters, numbers and non-alphanumeric symbols".

Next, we should **expose a port for the container**. The server works on port **1433** and we will start it **locally on the same** one:

```
>> -p 1433:1433 `
```

Then, we should **create a volume**, otherwise **data will be lost** when container is stopped, which is bad for a database container. We will name our **volume `sqldata`** and map it to the **`/var/opt/mssql` directory**, where **SQL Server data is stored**:

```
>> -v sqldata:/var/opt/mssql `
```

At the end, we will use the **`-d` option** to run the container in **detached mode** and will use the **`mcr.microsoft.com/mssql/server` image**:

```
>> -d mcr.microsoft.com/mssql/server
```

Note: we **didn't pull the image in advance** but don't worry – it will be **pulled automatically** when the **`docker run`** command is executed.

Execute the above command and the **container should be created**:

```
PS C:\Users\PC> docker run `
>> -e ACCEPT_EULA=Y `
>> -e MSSQL_SA_PASSWORD=yourStrongPassword12# `
>> -p 1433:1433 `
>> -v sqldata:/var/opt/mssql `
>> -d mcr.microsoft.com/mssql/server
Unable to find image 'mcr.microsoft.com/mssql/server:latest' locally
latest: Pulling from mssql/server
342d87d17479: Pull complete
112c1458d0bd: Pull complete
04016b3a8e25: Pull complete
Digest: sha256:7c61aeefa1c8eb55bccfa8d536a283ec922c486c7688e51f193b84c5f0aa3768
Status: Downloaded newer image for mcr.microsoft.com/mssql/server:latest
a7b7d5ddcf99b35974ecee1251e3c51df1e33e6578837bb420c6aebd146cbcbd
```