Running a SpringBoot Application in a Docker Container

SpringBoot Application Overview

We start by creating a basic SpringBoot application with a single controller. The controller responds to a GET request by returning a simple "Hello" message.

```
@RestController
public class HelloController {
@GetMapping("/hello")
public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
return String.format("Hello %s!", name);
}
}
```

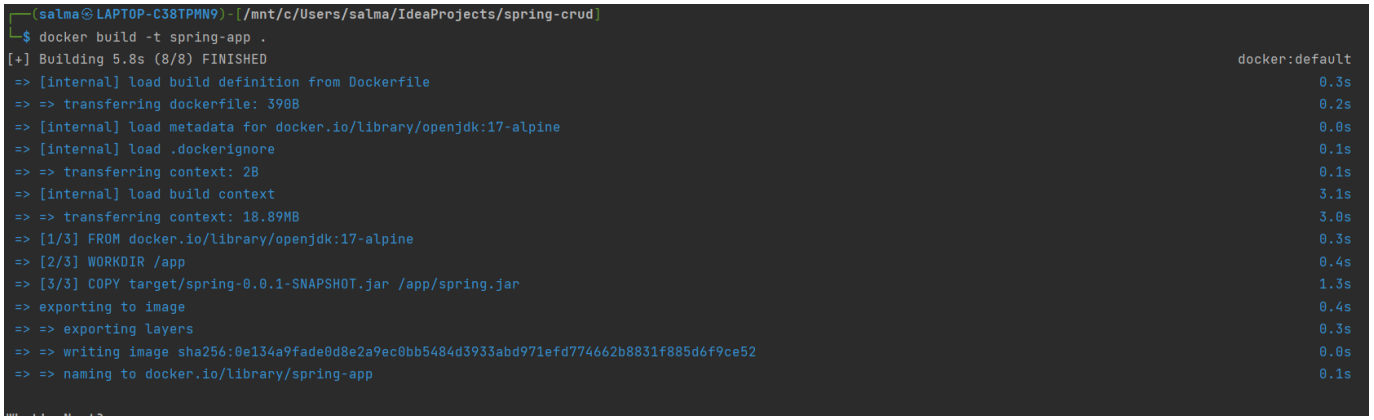
Dockerfile

To containerize the SpringBoot application, we define a Dockerfile. This file specifies the environment and commands required to package and run the application within a Docker container.

```
# Use OpenJDK 17 as base image
FROM openjdk:17-alpine

# Set the working directory in the container
WORKDIR /app

# Copy the packaged jar file into the container at /app
COPY target/spring-0.0.1-SNAPSHOT.jar /app/spring.jar

# Expose port 8080
EXPOSE 8080

# Command to run the spring boot application
CMD ["java", "-jar", "spring.jar"]
```

Building and Running the Docker Image

```
docker build -t .
```

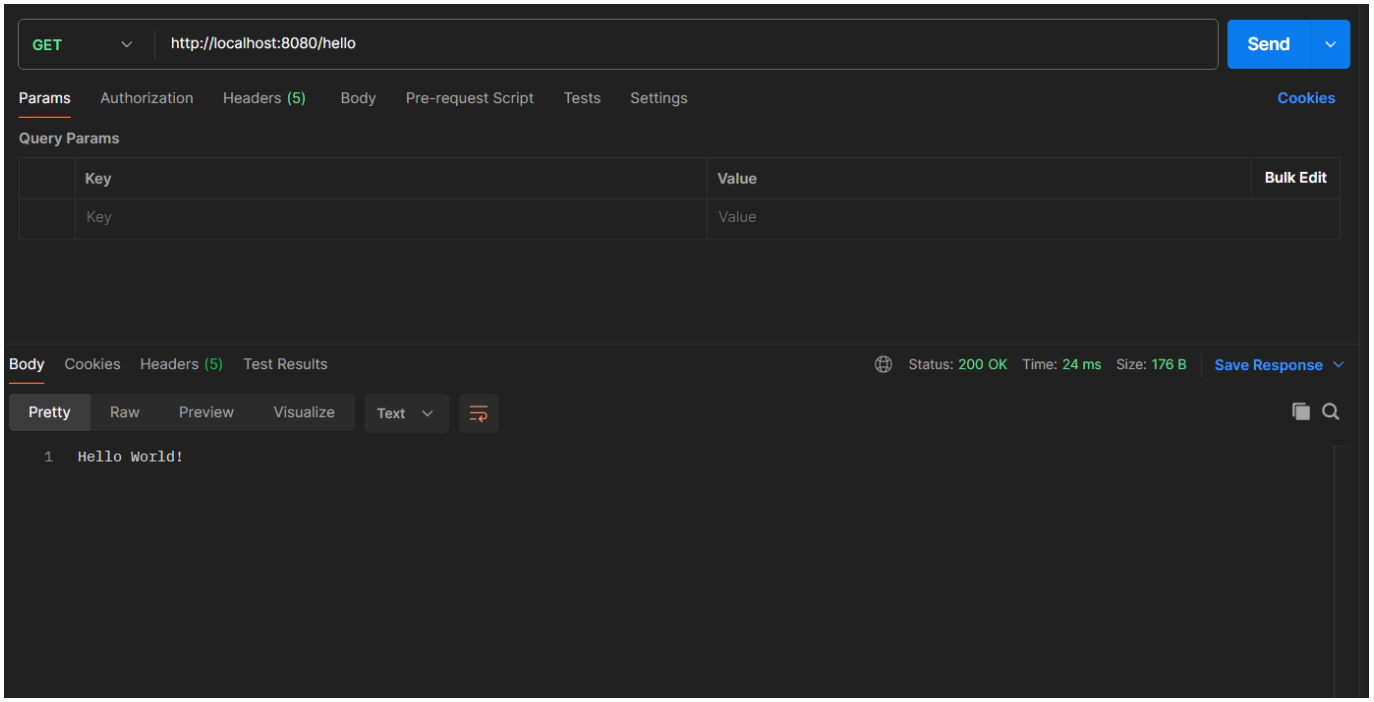


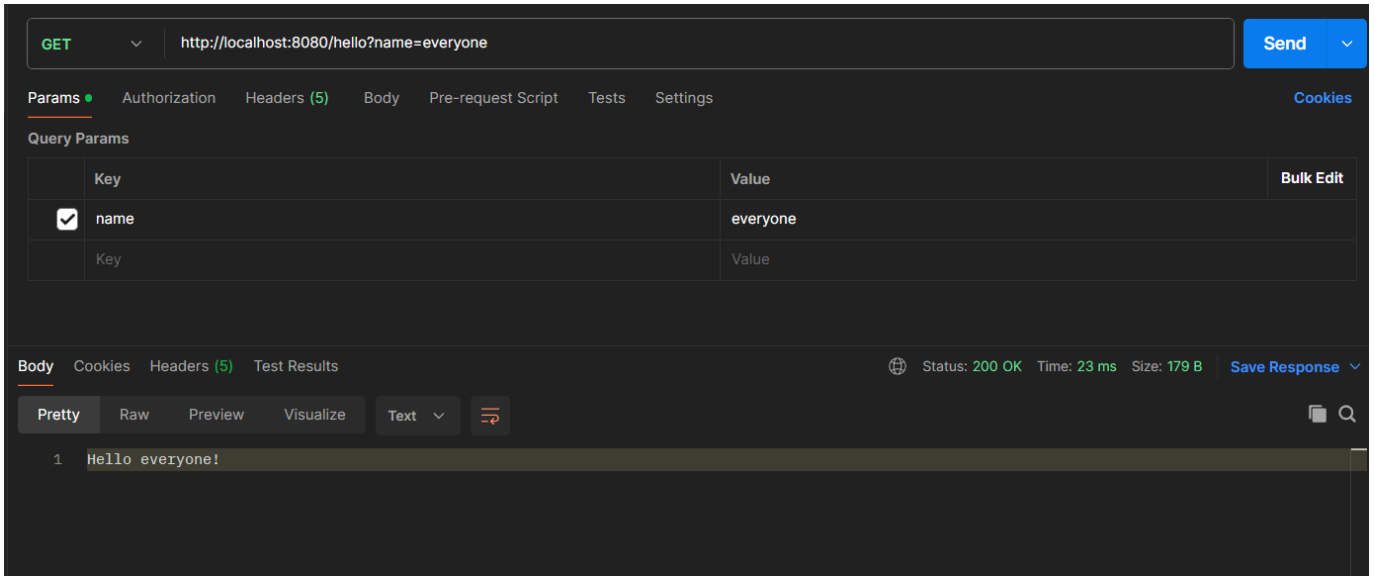
Then, we launch the Docker container while exposing port 8080 to allow access to the SpringBoot application.

```
docker run -p 8080:8080 spring-app
```

```
(salma㉿LAPTOP-C38TPMN9)-[/mnt/c/Users/salma/IdeaProjects/spring-crud]
$ docker run -p 8080:8080 spring-app

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.1.0)

2024-02-19T10:55:33.890Z  INFO 1 --- [           main] com.start.spring.Application            : Starting Application v0.0.1-SNAPSHOT using Java 17-ea with PID 1 (/app/spring.jar started by root in /app)
2024-02-19T10:55:33.913Z  INFO 1 --- [           main] com.start.spring.Application            : No active profile set, falling back to 1 default profile: "default"
2024-02-19T10:55:38.005Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2024-02-19T10:55:38.071Z  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-02-19T10:55:38.073Z  INFO 1 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.8]
2024-02-19T10:55:38.413Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-02-19T10:55:38.424Z  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4121 ms
2024-02-19T10:55:39.948Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2024-02-19T10:55:40.025Z  INFO 1 --- [           main] com.start.spring.Application            : Started Application in 8.35 seconds (process running for 10.675)
aaslema.......
```

Then we check if it works





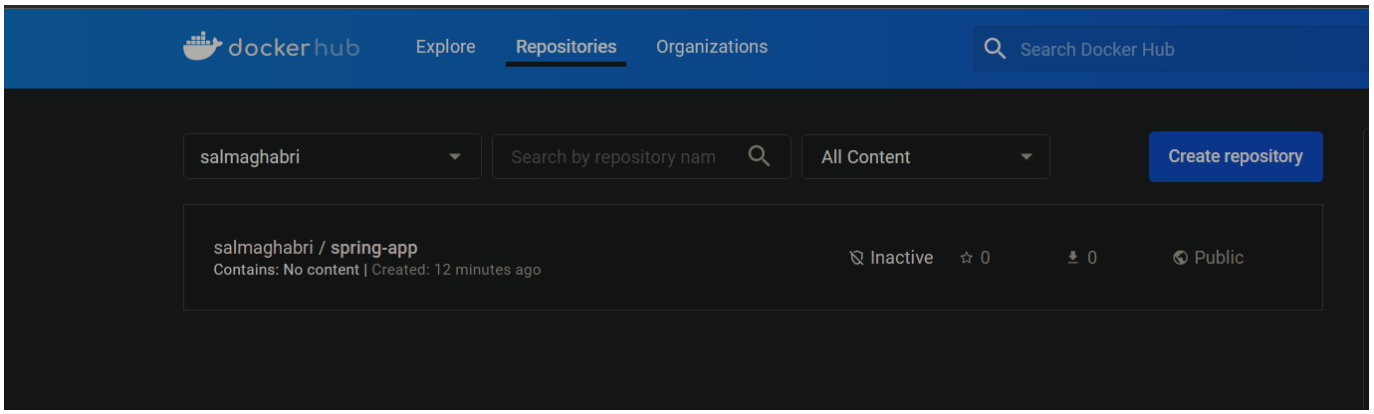
Pushing to DockerHub

login

```
docker login
```

```
┌──(salma㉿LAPTOP-C38TPMN9)-[~]
└─$ docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head
 over to https://hub.docker.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and
 is required for organizations using SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: salmaghabri
Password:
Login Succeeded
```

create a new repo in dockerhub



change the local image's tag

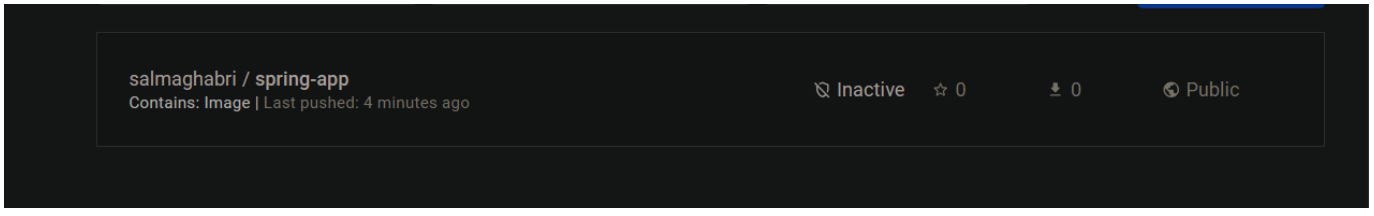
```
docker tag local_image_name:tag_name dockerhub_username/repository_name:tag_name
```

```
┌──(salma㉿LAPTOP-C38TPMN9)-[~]
└─$ docker tag spring-app salmaghabri/spring-app
```

push to dockerhub

```
docker push dockerhub_username/repository_name:tag_name
```

```
┌──(salma㉿LAPTOP-C38TPMN9)-[~]
└─$ docker push salmaghabri/spring-app
Using default tag: latest
The push refers to repository [docker.io/salmaghabri/spring-app]
25f0fadc014e: Pushed
913bd0375e21: Pushed
34f7184834b2: Mounted from library/openjdk
5836ece05bfd: Mounted from library/openjdk
72e830a4dff5: Mounted from library/openjdk
latest: digest: sha256:2e26382bb7d3bf16fa30091fbc8a6d6193f5de107438f1e2a2d577c542d024a8 size: 1369
```



Understanding Docker Image Layers and Reusability

Docker Image Layers

Docker images are composed of multiple layers, each representing a discrete instruction in the Dockerfile. These layers are stacked on top of each other, with each layer caching the results of the corresponding build step. This approach enables efficient image building and sharing.

Reusability of Images

One of the key advantages of Docker is the reusability of images. By utilizing existing images as base layers in the Dockerfile, developers can build upon well-tested and optimized environments. This reduces the need to recreate the entire environment from scratch, saving time and resources.

Docker Image Layers

Docker images are composed of multiple layers, each representing a discrete instruction in the Dockerfile. These layers are stacked on top of each other, with each layer caching the results of the corresponding build step. This approach enables efficient image building and sharing.

Reusability of Images

One of the key advantages of Docker is the reusability of images. By utilizing existing images as base layers in the Dockerfile, developers can build upon well-tested and optimized environments. This reduces the need to recreate the entire environment from scratch, saving time and resources.

Running a Multi-Container Application

Without Docker Compose

Build Docker image for a spring boot app

Launch the container for a Spring Boot application and verify communication:

```
FROM openjdk:17-alpine

WORKDIR /app

COPY target/CAT.jar /app

EXPOSE 8080

# entrypoint khir
ENTRYPOINT ["java", "-jar", "CAT.jar"]
```

```
┌──(salma㉿LAPTOP-C38TPMN9)-[/mnt/c/Users/salma/Downloads/elyadata-java-springboot-stub/elyadata-java-springboot-stub]
└─$ docker build -t cat-back .
[+] Building 163.5s (5/7)                                                                              docker:default
 => [internal] load build definition from Dockerfile                                                            0.1s
 => => transferring dockerfile: 190B                                                                            0.1s
 => [internal] load metadata for docker.io/library/eclipse-temurin:17-jdk-focal                                 4.6s
 => [internal] load .dockerignore                                                                               0.1s
 => => transferring context: 2B                                                                                 0.0s
 => [internal] load build context                                                                               6.5s
 => => transferring context: 53.15MB                                                                            6.5s
 => CANCELED [1/3] FROM docker.io/library/eclipse-temurin:17-jdk-focal@sha256:8bd16a6366f6e6cf981f796cf07696de2e4a7654260244418d93bbc645595737  158.4s
 => => resolve docker.io/library/eclipse-temurin:17-jdk-focal@sha256:8bd16a6366f6e6cf981f796cf07696de2e4a7654260244418d93bbc645595737  0.1s
 => => sha256:8bd16a6366f6e6cf981f796cf07696de2e4a7654260244418d93bbc645595737 1.21kB / 1.21kB                  0.0s
 => => sha256:dcdf4ce432f13324d7ae9738dc374de01fd8783b6bd9af91162e7478631ef296 1.37kB / 1.37kB                  0.0s
 => => sha256:2dc82430b3a5d16f2ebeb5d4e77196d2d47d96d6f6bce48d8a106bc10277a899 6.78kB / 6.78kB                  0.0s
 => => sha256:276661600bf82016e796ef3c730ed0615c29dc62c3d76898043d1c302939275f 17.83MB / 144.90MB             158.3s
 => => sha256:3c67549075b6db92af85c8649f848d697b5bb1f448b436c4b4d6ee6834ab45f7 11.53MB / 28.58MB              158.3s
 => => sha256:b3cda026f111795ebd512080c1256f21ecd05b7ae0d155c104abe0f60fc168ff 18.87MB / 20.67MB              158.3s
```

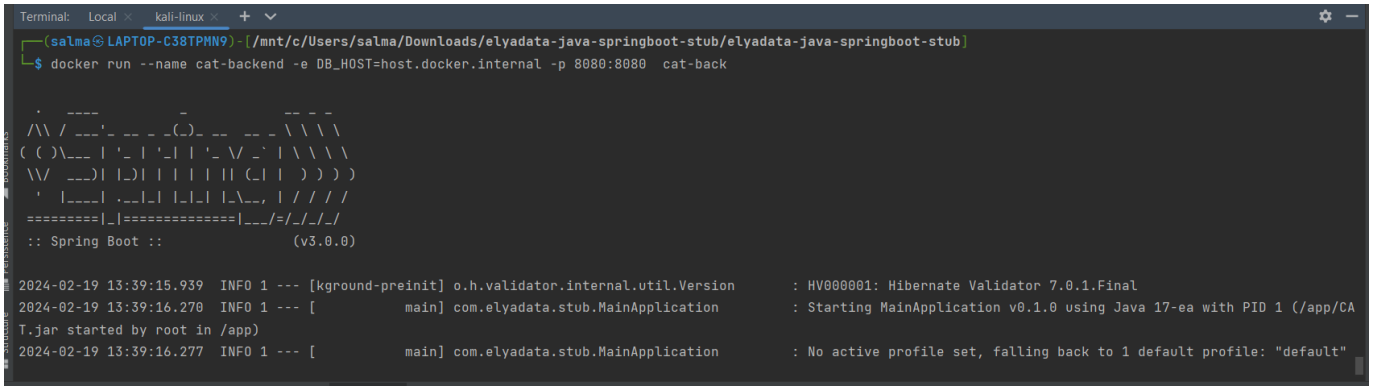
postgres-container

```
docker pull postgres
```

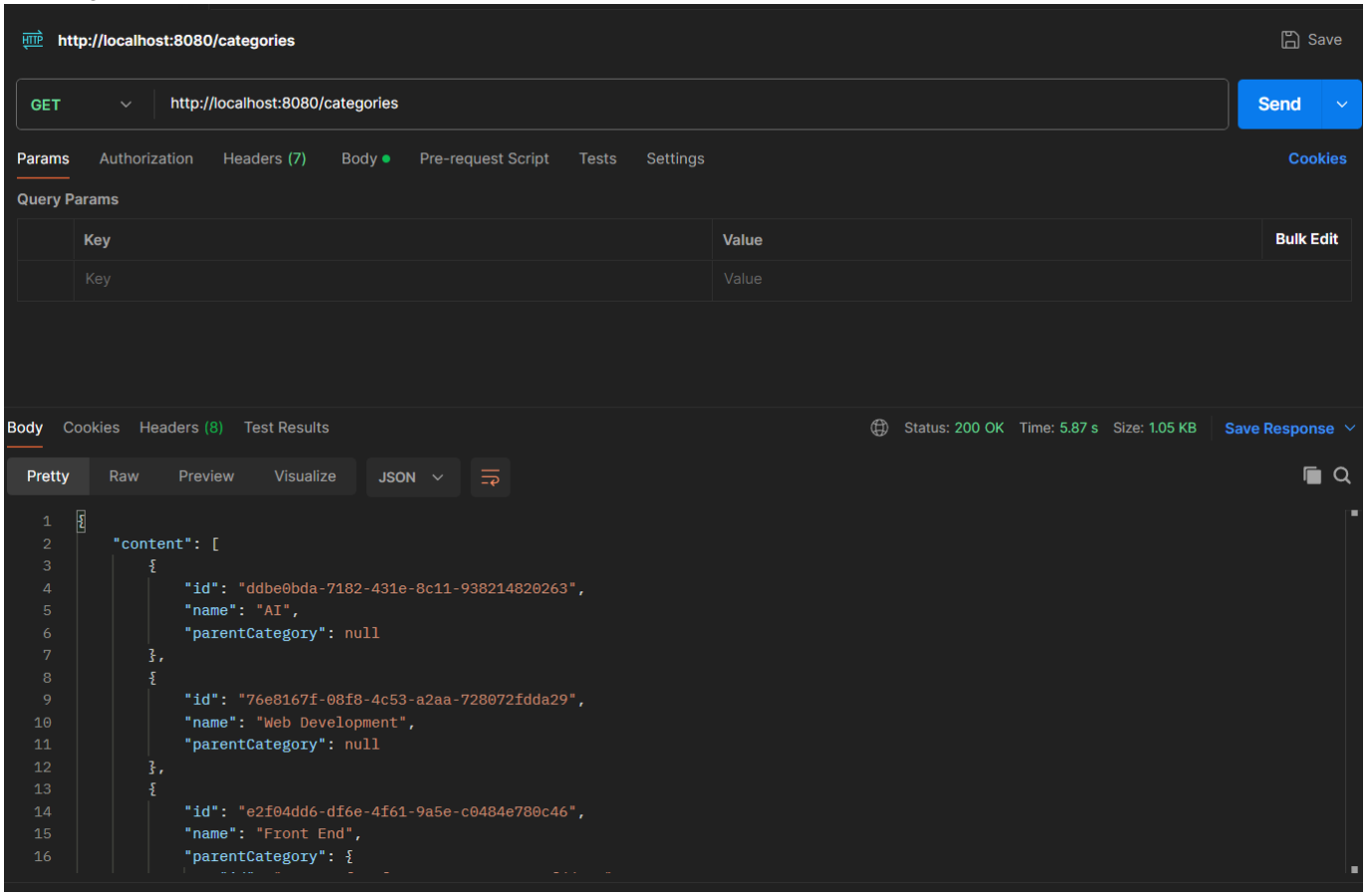
run containers spereately

```
docker run --name postgres-container -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 -d postgres
        ```

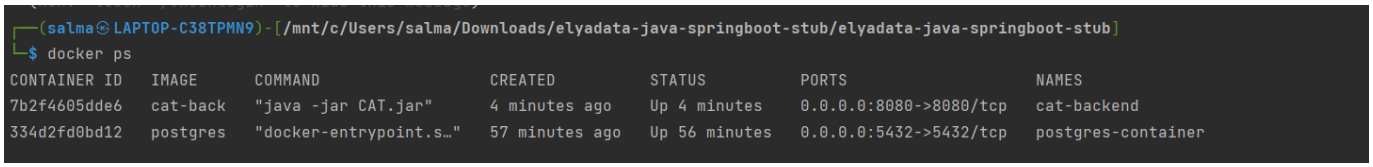
``` shell
docker run --name cat-backend -e DB_HOST=host.docker.internal -p 8080:8080  cat-back
```



checking if it works




```
Docker ps
```

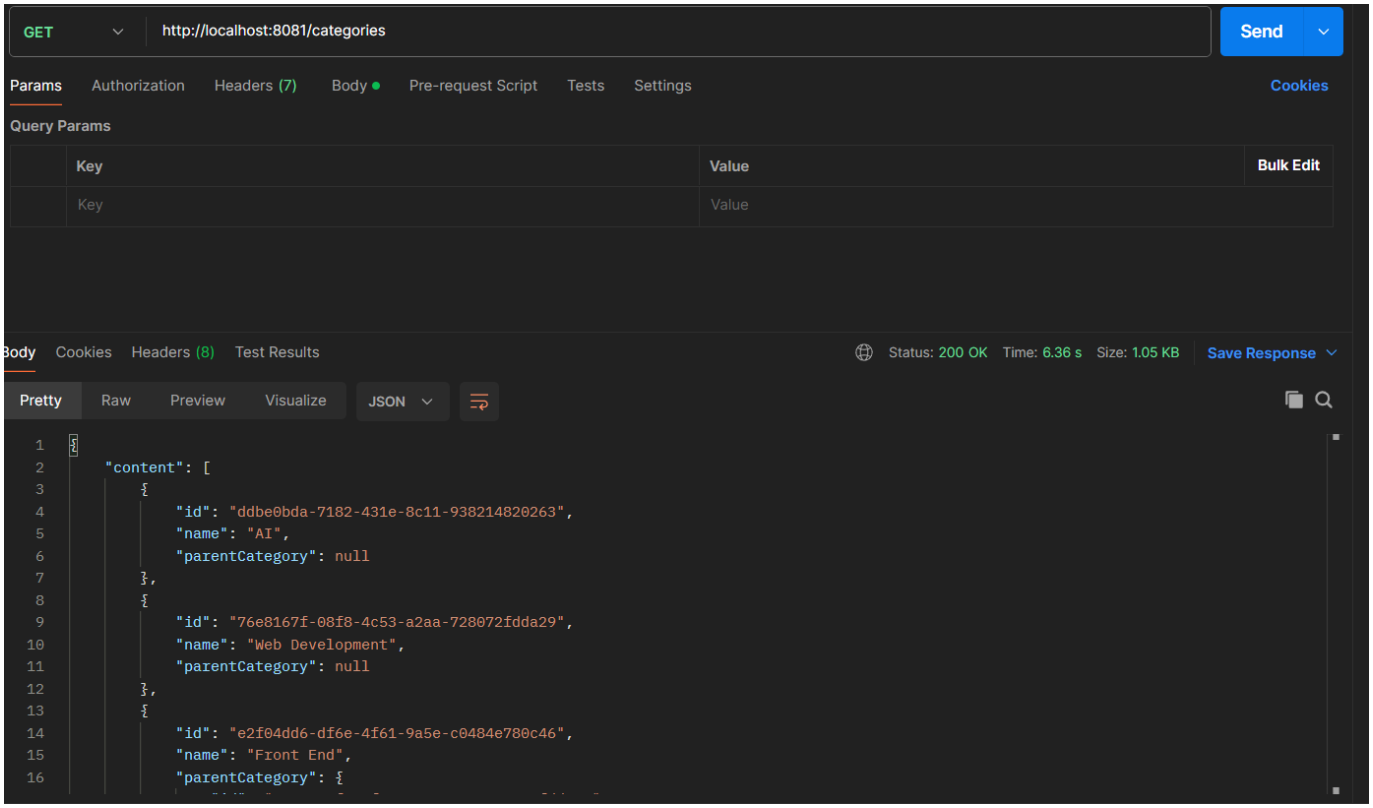


same application different ports

```
┌──(salma㉿LAPTOP-C38TPMN9)-[/mnt/c/Users/salma/Downloads/elyadata-java-springboot-stub/elyadata-java-springboot-stub]
└─$ docker run  -e DB_HOST=host.docker.internal -p 8081:8080 -d  cat-back
4487a0e08216a2dfe9cf704ca0c98a49977cec00765b4a73daff56c7198c3a57
```

```
┌──(salma㉿LAPTOP-C38TPMN9)-[/mnt/c/Users/salma/Downloads/elyadata-java-springboot-stub/elyadata-java-springboot-stub]
└─$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED             STATUS             PORTS                    NAMES
4487a0e08216   cat-back   "java -jar CAT.jar"      2 minutes ago       Up 2 minutes       0.0.0.0:8081->8080/tcp   adoring_heisenberg
3b7b1c6f28d8   cat-back   "java -jar CAT.jar"      2 minutes ago       Up 2 minutes       0.0.0.0:8080->8080/tcp   elastic_khorana
334d2fd0bd12   postgres   "docker-entrypoint.s…"   About an hour ago   Up About an hour   0.0.0.0:5432->5432/tcp   postgres-container
```

if we check with the new port



docker compose

Compose a Docker Compose file to set the interconnected services:

```yaml
version: '3.8'

services:
postgres:
        image: postgres:latest
                container_name: postgres
        environment:
                POSTGRES_DB: gps2
                POSTGRES_USER: ${POSTGRES_USER}
                POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
                ports:
                - "5432:5432"
                volumes:
                - postgres:/var/lib/postgresql/data

back-cat:
        build: .
        container_name: back-cat
        ports:
        - "8080:8080"
        environment:
                SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/gps2
```

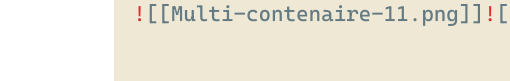
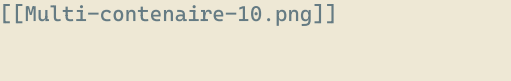
```
                        SPRING_DATASOURCE_USERNAME: ${POSTGRES_USER}
                        SPRING_DATASOURCE_PASSWORD: ${POSTGRES_PASSWORD}

        depends_on:
        - postgres

        volumes:
                postgres:
                        driver: local
        ```
Postgres Service:

- **Image**: Pulls the latest PostgreSQL image from Docker Hub.
- **Container Name**: Specifies the name of the container as "postgres".
- **Environment Variables**:
    - `POSTGRES_DB`: Sets the name of the database to "gps2".
    - `POSTGRES_USER`: Uses the value of the `POSTGRES_USER` environment variable.
    - `POSTGRES_PASSWORD`: Uses the value of the `POSTGRES_PASSWORD` environment variable.
- **Ports**: Maps port 5432 of the host to port 5432 of the container, allowing external access to the PostgreSQL service.
- **Volumes**: Mounts a volume named "postgres" to persist data at "/var/lib/postgresql/data" in the container.

Back-cat Service:

- **Build**: Specifies to build the Dockerfile found in the current directory (`.`).
- **Container Name**: Specifies the name of the container as "back-cat".
- **Ports**: Maps port 8080 of the host to port 8080 of the container, allowing external access to the Spring Boot application.
- **Environment Variables**:
    - `SPRING_DATASOURCE_URL`: Sets the URL for the Spring Boot application to connect to the PostgreSQL database.
    - `SPRING_DATASOURCE_USERNAME`: Uses the value of the `POSTGRES_USER` environment variable.
    - `SPRING_DATASOURCE_PASSWORD`: Uses the value of the `POSTGRES_PASSWORD` environment variable.
- **Depends On**: Specifies that the `back-cat` service depends on the `postgres` service.

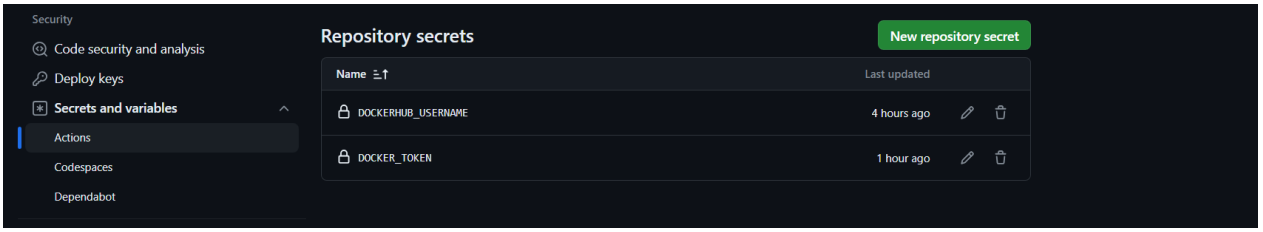
Volumes:
- **postgres**: Defines a volume named "postgres" with the "local" driver to persist PostgreSQL data.

Checking communication



Building and pushing docker image with github actions
create a GitHub Actions workflow
This GitHub Actions workflow file automates the process of building a Spring Boot application, packaging it with Maven, building a Docker image, and pushing it to Docker Hub.
```yaml
name: Build and Push Docker Image

  on:
  push:
  branches:
  - master
  env:
  USERNAME: ${{ secrets.DOCKERHUB_USERNAME }}
  DOCKER_TOKEN: ${{ secrets.DOCKER_TOKEN }}

  jobs:
  build:
  runs-on: ubuntu-latest

  steps:
```

```
- name: Checkout code
uses: actions/checkout@v2
- name: Set up JDK 17
uses: actions/setup-java@v2
with:
java-version: 17
distribution: 'adopt'

- name: Build with Maven
run: ./mvnw clean package

# Login to DockerHub
- name: Login to DockerHub
run: docker login -u $USERNAME --password $DOCKER_TOKEN

# Build and push Docker image
- name: Build and push Docker image
run: \
docker build -t salmaghabri/with-ga:latest .
docker push salmaghabri/with-ga:latest
```

1. **Workflow Triggers**:
   - The workflow triggers on `push` events to the `master` branch.
2. **Environment Variables**:
   - It sets up environment variables `USERNAME` and `DOCKER_TOKEN` to store Docker Hub credentials. These are sourced from GitHub secrets.



3. **Jobs**:
   - **build**: This job runs on an `ubuntu-latest` virtual machine.
     - **Steps**:
       - `Checkout code`: Checks out the source code from the repository.
       - `Set up JDK 17`: Sets up JDK 17 using the `actions/setup-java@v2` action.
       - `Build with Maven`: Runs the Maven wrapper script (`mvnw`) to clean the project and package it into an executable JAR file. But first, we need to ensure that git has the execution permission on mnvw by running `git update-index --chmod=+x ./mvnw`
       - `Login to DockerHub`: Logs in to Docker Hub using the provided credentials.
       - `Build and push Docker image`: Builds a Docker image from the Dockerfile in the repository and pushes it to Docker Hub. The Docker image is tagged as `salmaghabri/with-ga:latest`.
4. **Docker Hub Credentials**:
   - The workflow uses the `DOCKERHUB_USERNAME` and `DOCKER_TOKEN` secrets to authenticate with Docker Hub.
5. **Docker Image Tagging**:
   - The Docker image is tagged with `salmaghabri/with-ga:latest`.

Build finished

After running run the jobs of the workflow, we can see that the build succeded and the image was pushed.