

TP1 Devops

Work done by: Mohamed Aziz Bellaj, Louay Badri and Salma Ghabri

I. Dockerizing a Springboot Application

1. 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.

2. Unit tests of the controller with Junit

Test 1: `shouldReturnDefaultMessage()`: This test checks if calling the `/hello` endpoint without passing any parameters returns "Hello World!".

Test 2: `shouldReturnCustomMessage()`: This test checks if calling the `/hello` endpoint with the `name` parameter returns the correct personalized message.

```
@Test
void shouldReturnDefaultMessage() throws Exception {
    this.mockMvc.perform(get("/hello"))
        .andExpect(status().isOk())
        .andExpect(content().string("Hello World!"));
}

@Test
void shouldReturnCustomMessage() throws Exception {
    this.mockMvc.perform(get("/hello").param("name", "GL5"))
        .andExpect(status().isOk())
        .andExpect(content().string("Hello GL5!"));
}
```

Dockerfile

To containerize the SpringBoot application, we define this Dockerfile.

```
# 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"]
```

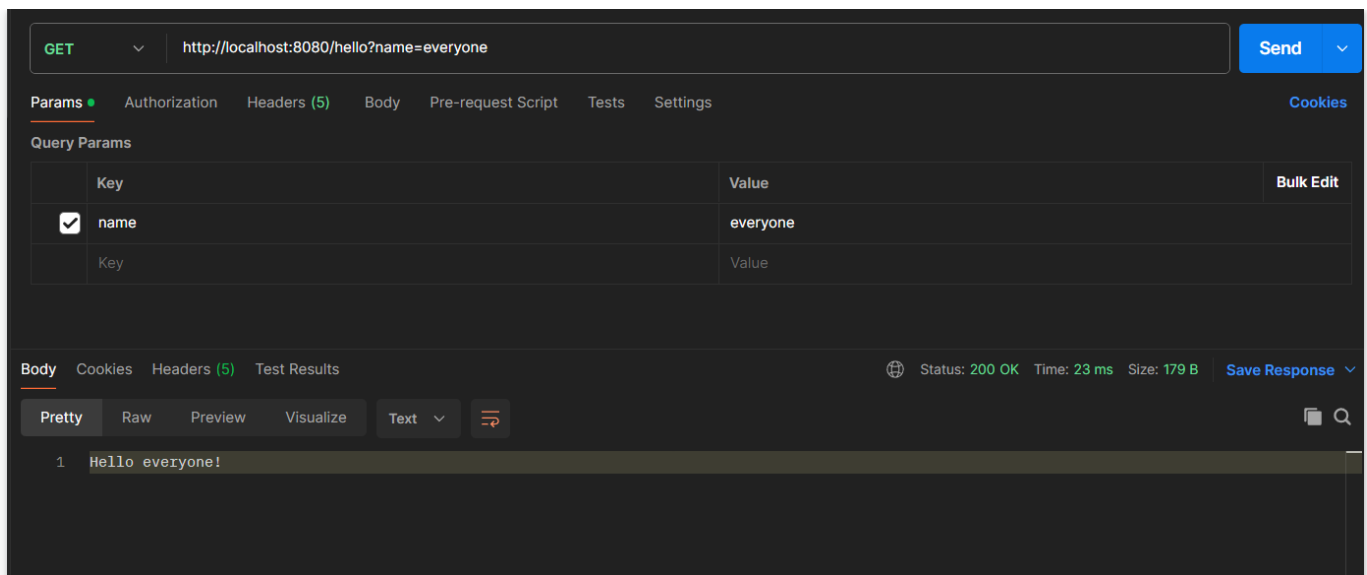
Building and Running the Docker Image

```
docker build -t .
```

```
(salma@LAPTOP-C38TPHN9) - [ /mnt/c/Users/salma/IdeaProjects/spring-crud ]
$ docker build -t spring-app .
[+] Building 5.8s (8/8) FINISHED
=> [internal] load build definition from Dockerfile                                docker:default
=> => transferring dockerfile: 390B                                              0.3s
=> [internal] load metadata for docker.io/library/openjdk:17-alpine              0.2s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 28                                                    0.1s
=> [internal] load build context                                                  3.1s
=> => transferring context: 18.89MB                                              3.0s
=> [1/3] FROM docker.io/library/openjdk:17-alpine                              0.3s
=> [2/3] WORKDIR /app                                                            0.4s
=> [3/3] COPY target/spring-0.0.1-SNAPSHOT.jar /app/spring.jar                 1.3s
=> exporting to image                                                            0.4s
=> => exporting layers                                                            0.3s
=> => writing image sha256:0e134a9fade0d8e2a9ec8bb5484d3933abd971efd774662b8831f885d6f9ce52 0.0s
=> => naming to docker.io/library/spring-app                                    0.1s
```

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

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



II. Building and pushing docker image with github actions

1. GitHub Actions workflow

This GitHub Actions workflow file automates the process of building a Spring Boot application, packaging and testing it with Maven, building a Docker image, and pushing it to Docker Hub.

```
name: Build, Test, 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:
      # Checkout the code
      - name: Checkout code
        uses: actions/checkout@v2
      # Set up JDK 17
      - name: Set up JDK 17
        uses: actions/setup-java@v2
        with:
          java-version: 17
          distribution: "adopt"
```

```

# Build and run unit tests with Maven
- name: Build and run tests with Maven
  run: ./mvnw clean verify
# 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

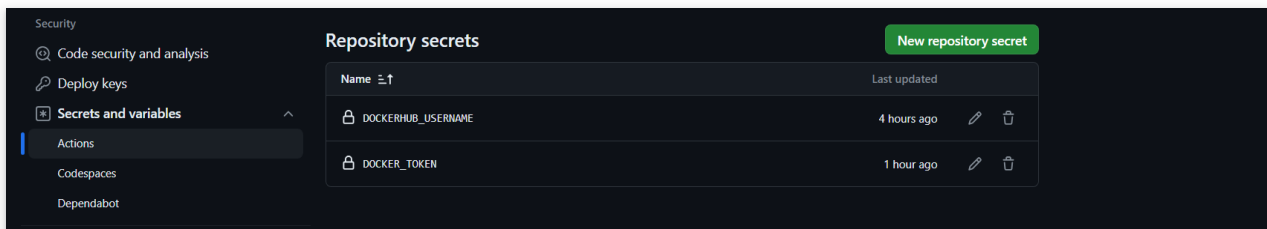
```

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 and run tests with Maven:** Runs the Maven wrapper script (`mvnw`) to clean the project, compile the code, and execute unit tests using JUnit. This step ensures that all tests pass before proceeding to the packaging and Docker build stages. If any test fails, the workflow stops here to prevent further actions.
 - **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`.

2. Build finished

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

The screenshot shows the GitHub Actions interface for a workflow named "update ga workflow #12". The left sidebar contains a navigation menu with "Summary", "Jobs", "Run details", "Usage", and "Workflow file". The "Jobs" section is active, showing a list of jobs: "build" (green checkmark), "Set up JDK 17", "Build and run tests with Maven", "Login to DockerHub", "Build and push Docker image", "Post Set up JDK 17", "Post Checkout code", and "Complete job". The "build" job is selected, showing its details. The job status is "succeeded 31 minutes ago in 28s". The job steps are listed with their durations: "Set up job" (1s), "Checkout code" (0s), "Set up JDK 17" (4s), "Build and run tests with Maven" (10s), "Login to DockerHub" (0s), "Build and push Docker image" (8s), "Post Set up JDK 17" (0s), "Post Checkout code" (0s), and "Complete job" (0s). The "Annotations" section shows "2 warnings".

The screenshot shows the Docker Hub repository page for "salmaghabri / with-ga". The repository is "Inactive" (greyed out), has 0 stars, 0 downloads, and is "Public". The repository description is "Contains: Image | Last pushed: 5 minutes ago".