

Containerizing an existing app

To complete this step-by-step guide, the following would be required.

- Docker running locally
- A Git client
- An IDE or a Text editor.

Clone the repository below.

Repository: `git clone https://github.com/spring-projects/spring-petclinic.git`

NB:

To run a java application, it is recommended to have Java OpenJDK version 15 or later installed on your machine.

But not to worry Docker got you covered if you do not have it installed, as it would be built inside our image.

If you are using the cloned repository, The Spring Pets Clinic project, cloned earlier contains an embedded version of Maven/Gradle. Therefore, we don't need to install Maven separately on your local machine.

To run and preview the application, and check if it works locally without docker.

Open your terminal and navigate to the working directory we have created and run the following command:

```
./mvnw spring-boot:run
```

This downloads the dependencies, builds the project, and starts to run it.

To test that the application is working properly, open a new browser and navigate to **`http://localhost:8080`**.

Let us continue to build and run the application in Docker.

Common Docker file instructions

Dockerfile list of instruction usually starts with RUN, ENV, FROM, MAINTAINER, ADD, and CMD, among others.

- FROM - Specifies the base image that the Dockerfile will use to build a new image. we are using openjdk:16-alpine3.13 as our base image.
- MAINTAINER - Specifies the Dockerfile Author Name and his/her email.
- RUN - Runs any UNIX command to build the image.
- ENV - Sets the environment variables. For this post, JAVA_HOME is the variable that is set.
- CMD - Provides the facility to run commands at the start of container. This can be overridden upon executing the docker run command.
- ADD - This instruction copies the new files, directories into the Docker container file system at specified destination.
- EXPOSE - This instruction exposes specified port to the host machine.

Create a Docker file for Java

Let's walk through the process of creating a Docker File for our application. In the root of your project, create a file named "Dockerfile" and open this file in your text editor and paste the code below.

```
# syntax=docker/dockerfile:1

#Use the OpenJDK 11 image as the base image
FROM openjdk:16-alpine3.13

#Set the directory for executing future commands
WORKDIR /app

#Copy the app files from the host machine to image filesystem
COPY .mvn/ .mvn

COPY mvnw pom.xml ./

RUN ./mvnw dependency:go-offline

COPY src ./src

#This starts the Docker application
CMD ["/mvnw", "spring-boot:run"]
```

Create a .dockerignore file for Java

To increase the performance of the build, and as a general best practice, we recommend that you create a .dockerignore file in the same directory as the Dockerfile. For this tutorial, your .dockerignore file should contain just one line:

Target

Build an Image

Now that we've created our Dockerfile, let's build our first Docker image, with the command "**docker build**".

docker build -t java-docker .

Let us see if the process was successful. Use the command below to preview images.

docker images

We could remove docker images if not needed any more by using the command below.

docker rmi java-docker: v1.0.0

Create a Docker Account or Sign In

Got to <https://hub.docker.com/>

Create a free account or sign into Docker Desktop to gain access to push to docker repository.

Alternatively, run docker login in terminal to sign in from terminal.

Alternatively, Push to GitHub Repo

Got to <https://github.com/>

Create a free account or sign into GitHub to gain access to push to the repository.

Scan Image (Optional)

Docker and Snyk recently entered a partnership to provide container vulnerability scanning.

When running the `docker scan` command, scans local images against the Snyk security engine, providing you with security visibility into your local Dockerfiles and local images.

The Snyk engine scans the images or Dockerfiles for Common Vulnerabilities and Exposures (CVEs) and provides recommendations for CVE remediations.

```
Docker scan java-docker
```

```
Docker scan --login
```

Tagged Image for upload to registry.

To push an image to Docker Hub, you must first name your local image using your Docker Hub username and the repository name that you created through Docker Hub on the web. You can add multiple images to a repository by adding a specific :<tag> e.g.

```
docker tag <name-of-image>:<tag> <hub-user>/<repo-name>:<tag>
```

And then run the next step: `docker push <hub-user>/<repo-name>:<tag>`

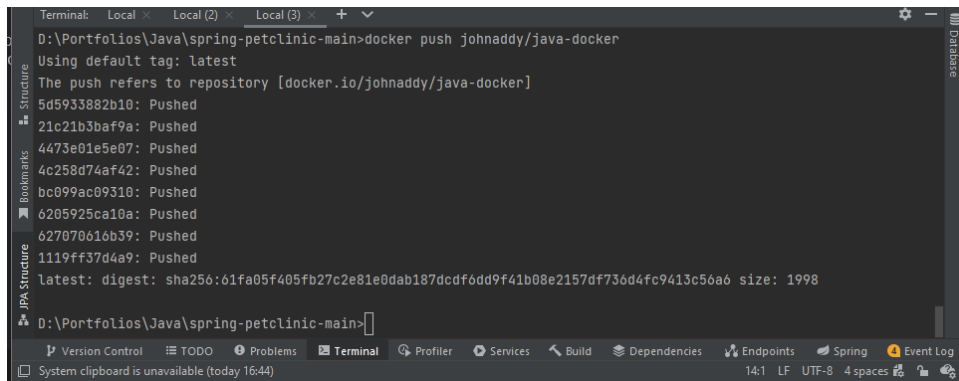
```
docker tag java-docker johnaddy/java-docker
```

Push Image to Docker Hub.

Let us now push our image to DockerHub. Let us run the commands below.

```
docker push johnaddy/java-docker
```

Figure 1: The image shows that docker image has been pushed to dockHub.

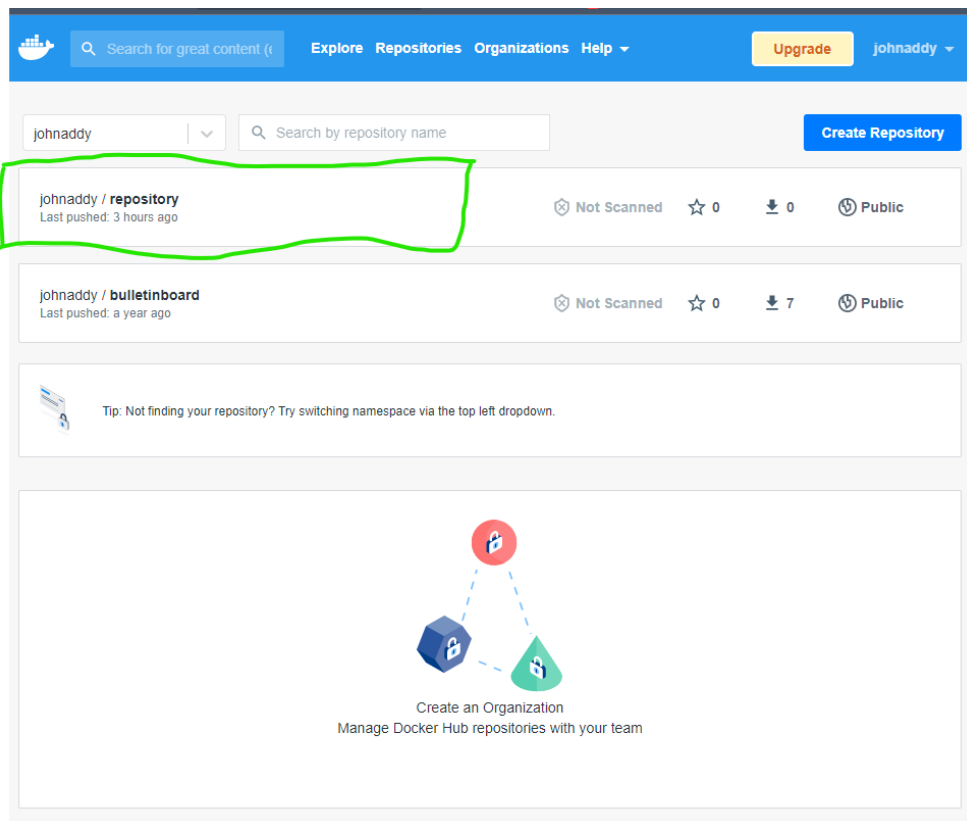


```
Terminal: Local x Local (2) x Local (3) x + v
D:\Portfolios\Java\spring-petclinic-main>docker push johnaddy/java-docker
Using default tag: latest
The push refers to repository [docker.io/johnaddy/java-docker]
5d5933882b10: Pushed
21c21b3baf9a: Pushed
4473e01e5e07: Pushed
4c258d74af42: Pushed
bc099ac09310: Pushed
6205925ca10a: Pushed
627070616b39: Pushed
1119ff37d4a9: Pushed
latest: digest: sha256:61fa05f405fb27c2e81e0dab187dcdf6dd9f41b08e2157df736d4fc9413c56a6 size: 1998
D:\Portfolios\Java\spring-petclinic-main>
```

The screenshot shows a terminal window within an IDE. The terminal output indicates that the Docker image 'johnaddy/java-docker' has been successfully pushed to Docker Hub. It lists the layers being pushed and their corresponding digests, followed by the final 'latest' tag digest and size. The IDE interface includes a sidebar with 'Structure', 'Bookmarks', and 'JPA Structure' views, and a bottom toolbar with various development tools like 'Version Control', 'TODO', 'Problems', 'Terminal', 'Profiler', 'Services', 'Build', 'Dependencies', 'Endpoints', 'Spring', and 'Event Log'.

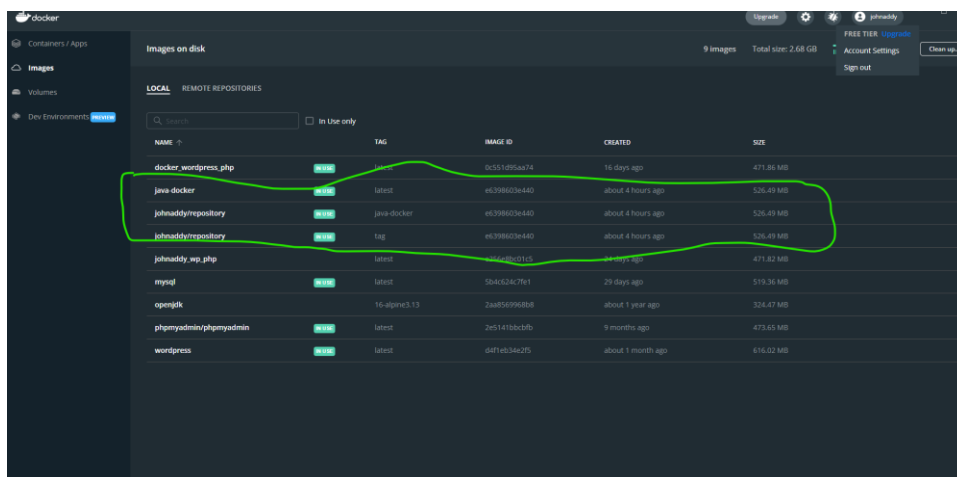
Congrats! Now you have successfully pushed to docker hub and should see something that looks like this image.

Figure 2: The image shows the docker image on docker hub.



NB: Also, should you see the image on your Docker Desktop, and it should look like the image below.

Figure 3: Docker images inside the docker desktop.



Let a friend test your Image.

Step 1: Let us pull docker the image by using the command

```
docker pull johnnaddy/java-docker
```

Step 2: Run your image as a container locally

Start the container and expose port 8080 to port 8080 on the host.

```
docker run --publish 8080:8080 johnnaddy/java-docker
```

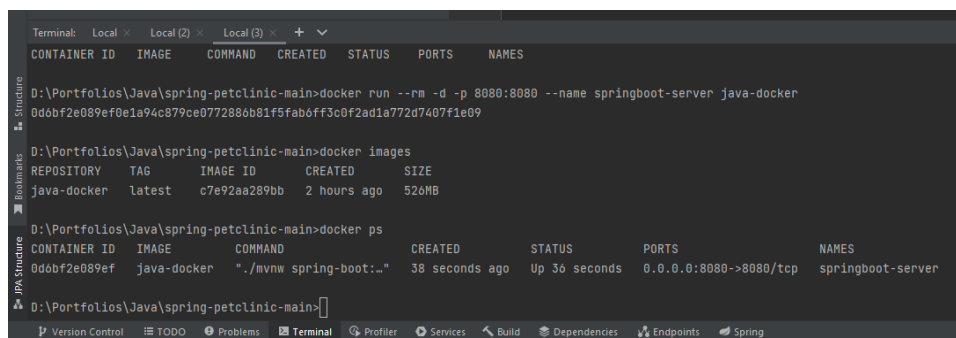
We could also alternatively run our container locally in a detached mode, which means the container will run in the background and most importantly name our container for easy identification.

```
docker run --rm -d -p 8080:8080 --name springboot-server johnnaddy/java-docker
```

Only Run in detached mode

```
docker run -d -p 8080:8080 johnnaddy/java-docker
```

Figure 4: The terminal shows that the application has been renamed successfully.



```
Terminal: Local x Local (2) x Local (3) x + v
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
0d6bf2e089ef0e1a94c879ce0772886b81f5fab6ff3c0f2ad1a772d7407f1e09

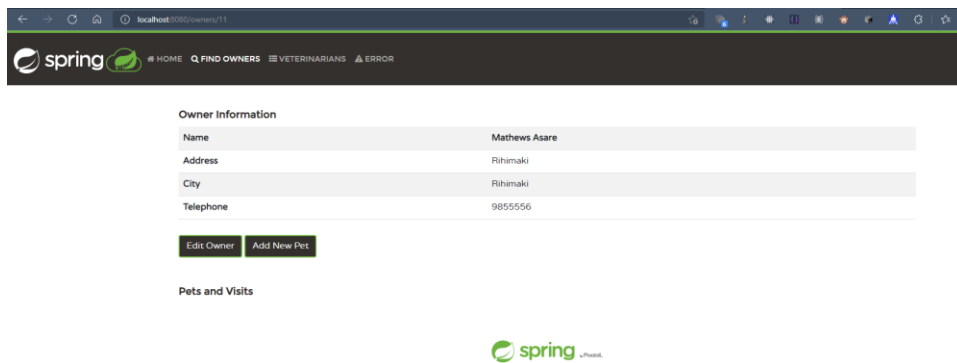
D:\Portfolios\Java\spring-petclinic-main>docker run --rm -d -p 8080:8080 --name springboot-server java-docker
0d6bf2e089ef0e1a94c879ce0772886b81f5fab6ff3c0f2ad1a772d7407f1e09

D:\Portfolios\Java\spring-petclinic-main>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
java-docker   latest   c7e92aa289bb   2 hours ago    526MB

D:\Portfolios\Java\spring-petclinic-main>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
0d6bf2e089ef   java-docker    "./mvnw spring-boot:..." 38 seconds ago Up 36 seconds 0.0.0.0:8080->8080/tcp springboot-server

D:\Portfolios\Java\spring-petclinic-main>
```

Figure 5. The below is a containerized application that runs in the browser.



References

[Build your Java image | Docker Documentation](#)