## screenshot.pngBuilding the docker images

In this Lab, we’ll run a full-stack application built using Spring Boot, React, and MySQL.

It is a Polling app where users can login, create a Poll, and vote for a Poll.

Please clone the application locally before proceeding further.

### Dockerfile for the Spring Boot application

The backend of the application ([polling-app-server](https://github.com/callicoder/spring-security-react-ant-design-polls-app/tree/master/polling-app-server)) is written in Spring Boot. Here is the Dockerfile for the spring boot app:

#### Stage 1: Build the application

FROM openjdk:8-jdk-alpine as build

# Set the current working directory inside the image

WORKDIR /app

# Copy maven executable to the image

COPY mvnw .

COPY .mvn .mvn

# Copy the pom.xml file

COPY pom.xml .

# Build all the dependencies in preparation to go offline.

# This is a separate step so the dependencies will be cached unless

# the pom.xml file has changed.

RUN ./mvnw dependency:go-offline -B

# Copy the project source

COPY src src

# Package the application

RUN ./mvnw package -DskipTests

RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../\*.jar)

#### Stage 2: A minimal docker image with command to run the app

FROM openjdk:8-jre-alpine

ARG DEPENDENCY=/app/target/dependency

# Copy project dependencies from the build stage

COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib

COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF

COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app

ENTRYPOINT ["java","-cp","app:app/lib/\*","com.example.polls.PollsApplication"]

### Dockerfile for the React application

The frontend of the application ([polling-app-client](https://github.com/callicoder/spring-security-react-ant-design-polls-app/tree/master/polling-app-client)) is written in React. We’ll deploy the React app behind an nginx server.

Following is the Dockerfile for the React app (It uses [nginx.conf](https://github.com/callicoder/spring-security-react-ant-design-polls-app/blob/master/polling-app-client/nginx.conf) available in the same directory.)

#### Stage 1: Build the react application

FROM node:12.4.0-alpine as build

# Configure the main working directory inside the docker image.

# This is the base directory used in any further RUN, COPY, and ENTRYPOINT

# commands.

WORKDIR /app

# Copy the package.json as well as the package-lock.json and install

# the dependencies. This is a separate step so the dependencies

# will be cached unless changes to one of those two files

# are made.

COPY package.json package-lock.json ./

RUN npm install

# Copy the main application

COPY . ./

# Arguments

ARG REACT\_APP\_API\_BASE\_URL

ENV REACT\_APP\_API\_BASE\_URL=${REACT\_APP\_API\_BASE\_URL}

# Build the application

RUN npm run build

#### Stage 2: Serve the React application from Nginx

FROM nginx:1.17.0-alpine

# Copy the react build from Stage 1

COPY --from=build /app/build /var/www

# Copy our custom nginx config

COPY nginx.conf /etc/nginx/nginx.conf

# Expose port 80 to the Docker host, so we can access it

# from the outside.

EXPOSE 80

ENTRYPOINT ["nginx","-g","daemon off;"]

## Creating the docker-compose.yml configuration

To deploy your application using docker compose, you need to create a docker-compose.yml file that contains configuration for all the services in your entire stack.

Following is the docker-compose.yml file for running our Polls app. It has three services: app-server, app-client, and db. The app-server contains configuration for the backend app, app-client contains configuration for the react app, and db is for the mysql database.

# Docker Compose file Reference (<https://docs.docker.com/compose/compose-file/>)

version: '3.7'

# Define services

services:

# App backend service

app-server:

# Configuration for building the docker image for the backend service

build:

context: polling-app-server # Use an image built from the specified dockerfile in the `polling-app-server` directory.

dockerfile: Dockerfile

ports:

- "8080:8080" # Forward the exposed port 8080 on the container to port 8080 on the host machine

restart: always

depends\_on:

- db # This service depends on mysql. Start that first.

environment: # Pass environment variables to the service

SPRING\_DATASOURCE\_URL: jdbc:mysql://db:3306/polls?useSSL=false**&serverTimezone**=UTC**&useLegacyDatetimeCode**=false

SPRING\_DATASOURCE\_USERNAME: callicoder

SPRING\_DATASOURCE\_PASSWORD: callicoder

networks: # Networks to join (Services on the same network can communicate with each other using their name)

- backend

- frontend

# Frontend Service

app-client:

build:

context: polling-app-client # Use an image built from the specified dockerfile in the `polling-app-client` directory.

dockerfile: Dockerfile

args:

REACT\_APP\_API\_BASE\_URL: http://127.0.0.1:8080/api

ports:

- "9090:80" # Map the exposed port 80 on the container to port 9090 on the host machine

restart: always

depends\_on:

- app-server

networks:

- frontend

# Database Service (Mysql)

db:

image: mysql:5.7

ports:

- "3306:3306"

restart: always

environment:

MYSQL\_DATABASE: polls

MYSQL\_USER: callicoder

MYSQL\_PASSWORD: callicoder

MYSQL\_ROOT\_PASSWORD: root

volumes:

- db-data:/var/lib/mysql

networks:

- backend

# Volumes

volumes:

db-data:

# Networks to be created to facilitate communication between containers

networks:

backend:

frontend:

Notice the depends\_on keyword in the compose file. Docker compose will make sure that all the dependencies of a service is started first before starting the service itself.

## Running the complete stack with docker compose

Well, you can now bring up the entire application with just one command:

$ docker-compose up

Starting spring-security-react-ant-design-polls-app\_db\_1 ... done

Starting spring-security-react-ant-design-polls-app\_app-server\_1 ... done

Starting spring-security-react-ant-design-polls-app\_app-client\_1 ... done

## ...

You see, it’s super easy to just run all the services using docker compose. Any new developer in your team can just run this command and start playing with your application without any setup whatsoever.

You can also stop all the services at once using docker compose using the following command:

$ docker-compose down

**Building the docker images**

In this article, we’ll run a full-stack application built using Spring Boot, React, and MySQL.

You can check out the complete code of the application on this [github repository](https://github.com/callicoder/spring-security-react-ant-design-polls-app) ). It is a Polling app where users can login, create a Poll, and vote for a Poll.

Please clone the application locally before proceeding further.

**Dockerfile for the Spring Boot application**

The backend of the application ([polling-app-server](https://github.com/callicoder/spring-security-react-ant-design-polls-app/tree/master/polling-app-server)) is written in Spring Boot. Here is the Dockerfile for the spring boot app:

#### Stage 1: Build the application

FROM openjdk:8-jdk-alpine as build

# Set the current working directory inside the image

WORKDIR /app

# Copy maven executable to the image

COPY mvnw .

COPY .mvn .mvn

# Copy the pom.xml file

COPY pom.xml .

# Build all the dependencies in preparation to go offline.

# This is a separate step so the dependencies will be cached unless

# the pom.xml file has changed.

RUN ./mvnw dependency:go-offline -B

# Copy the project source

COPY src src

# Package the application

RUN ./mvnw package -DskipTests

RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../\*.jar)

#### Stage 2: A minimal docker image with command to run the app

FROM openjdk:8-jre-alpine

ARG DEPENDENCY=/app/target/dependency

# Copy project dependencies from the build stage

COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib

COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF

COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app

ENTRYPOINT ["java","-cp","app:app/lib/\*","com.example.polls.PollsApplication"]

**Dockerfile for the React application**

The frontend of the application ([polling-app-client](https://github.com/callicoder/spring-security-react-ant-design-polls-app/tree/master/polling-app-client)) is written in React. We’ll deploy the React app behind an nginx server.

Following is the Dockerfile for the React app (It uses [nginx.conf](https://github.com/callicoder/spring-security-react-ant-design-polls-app/blob/master/polling-app-client/nginx.conf) available in the same directory.)

#### Stage 1: Build the react application

FROM node:12.4.0-alpine as build

# Configure the main working directory inside the docker image.

# This is the base directory used in any further RUN, COPY, and ENTRYPOINT

# commands.

WORKDIR /app

# Copy the package.json as well as the package-lock.json and install

# the dependencies. This is a separate step so the dependencies

# will be cached unless changes to one of those two files

# are made.

COPY package.json package-lock.json ./

RUN npm install

# Copy the main application

COPY . ./

# Arguments

ARG REACT\_APP\_API\_BASE\_URL

ENV REACT\_APP\_API\_BASE\_URL=${REACT\_APP\_API\_BASE\_URL}

# Build the application

RUN npm run build

#### Stage 2: Serve the React application from Nginx

FROM nginx:1.17.0-alpine

# Copy the react build from Stage 1

COPY --from=build /app/build /var/www

# Copy our custom nginx config

COPY nginx.conf /etc/nginx/nginx.conf

# Expose port 80 to the Docker host, so we can access it

# from the outside.

EXPOSE 80

ENTRYPOINT ["nginx","-g","daemon off;"]

**Creating the docker-compose.yml configuration**

To deploy your application using docker compose, you need to create a docker-compose.yml file that contains configuration for all the services in your entire stack.

Following is the docker-compose.yml file for running our Polls app. It has three services: app-server, app-client, and db. The app-server contains configuration for the backend app, app-client contains configuration for the react app, and db is for the mysql database.

# Docker Compose file Reference (<https://docs.docker.com/compose/compose-file/>)

version: '3.7'

# Define services

services:

# App backend service

app-server:

# Configuration for building the docker image for the backend service

build:

context: polling-app-server # Use an image built from the specified dockerfile in the `polling-app-server` directory.

dockerfile: Dockerfile

ports:

- "8080:8080" # Forward the exposed port 8080 on the container to port 8080 on the host machine

restart: always

depends\_on:

- db # This service depends on mysql. Start that first.

environment: # Pass environment variables to the service

SPRING\_DATASOURCE\_URL: jdbc:mysql://db:3306/polls?useSSL=false**&serverTimezone**=UTC**&useLegacyDatetimeCode**=false

SPRING\_DATASOURCE\_USERNAME: callicoder

SPRING\_DATASOURCE\_PASSWORD: callicoder

networks: # Networks to join (Services on the same network can communicate with each other using their name)

- backend

- frontend

# Frontend Service

app-client:

build:

context: polling-app-client # Use an image built from the specified dockerfile in the `polling-app-client` directory.

dockerfile: Dockerfile

args:

REACT\_APP\_API\_BASE\_URL: http://127.0.0.1:8080/api

ports:

- "9090:80" # Map the exposed port 80 on the container to port 9090 on the host machine

restart: always

depends\_on:

- app-server

networks:

- frontend

# Database Service (Mysql)

db:

image: mysql:5.7

ports:

- "3306:3306"

restart: always

environment:

MYSQL\_DATABASE: polls

MYSQL\_USER: callicoder

MYSQL\_PASSWORD: callicoder

MYSQL\_ROOT\_PASSWORD: root

volumes:

- db-data:/var/lib/mysql

networks:

- backend

# Volumes

volumes:

db-data:

# Networks to be created to facilitate communication between containers

networks:

backend:

frontend:

Notice the depends\_on keyword in the compose file. Docker compose will make sure that all the dependencies of a service is started first before starting the service itself.

**Running the complete stack with docker compose**

Well, you can now bring up the entire application with just one command:

$ docker-compose up

Starting spring-security-react-ant-design-polls-app\_db\_1 ... done

Starting spring-security-react-ant-design-polls-app\_app-server\_1 ... done

Starting spring-security-react-ant-design-polls-app\_app-client\_1 ... done

## ...

You see, it’s super easy to just run all the services using docker compose. Any new developer in your team can just run this command and start playing with your application without any setup whatsoever.

You can also stop all the services at once using docker compose using the following command:

$ docker-compose down