## Overview

Docker is a tool that makes it easy to run our application as a container.

In this lab we will:

* Utilize separate stages in our Dockerfile to build our application and run it
* Use docker-compose to deploy our application

[Overview](#_4l6y9k9n7exo)

[Instructions](#_hxr977vi6sdt)

[Task 0: Examine the App](#_hszxfr2ntf23)

[Task 1: Create a Simple Dockerfile](#_c0i533prk1n7)

[Step 1: Create the Dockerfile](#_y2xjad3y77zo)

[Step 2: Build the image](#_stcldscuqytw)

[Step 3: Run and test the application](#_y8iqsoq879p7)

[Task 2: Compile the app in separate images](#_gjg0saj5jfxe)

[Step 1: Update Dockerfile](#_r30mh0fkva2)

[Step 2: Build image](#_ysv4078yt1m6)

[Step 3: Test the app](#_xni8u8rpim9l)

[Task 3: Composify it](#_lp7wyam4f1)

[Step 1: Make a compose file](#_cv4h48awd8vl)

[Step 2: Deploy the app](#_9g89q5shkvg2)

[Step 3: Cleanup](#_fdag8dgrzsd6)

## 

## Instructions

Read this lab like a book, all text is there for a reason!

"→" denotes an action you must take

Use your favorite editor to edit files within the console. I suggest VI, nano, or emacs.

|  |
| --- |
| White boxes with black text denote commands and file contents |

|  |
| --- |
| Black boxes with green text denote example output |

### Task 0: Examine the App

We will be building an image for a Flight Tracker application created by Mohammed Aboullaite in his [blog post.](https://aboullaite.me/building-real-time-aircraft-tracker-using-reactor-and-spring-boot/) This application uses Java, spring-boot, and reactor. All of the dependencies are specified in a pom.xml file that the Maven build tool will use to install and build our application.

→ View the command usage

|  |
| --- |
| cd ~  git clone <https://github.com/jrrickerson/spring-boot-reactive>  cd spring-boot-reactive |

### Task 1: Create a Simple Dockerfile

#### Step 1: Create the Dockerfile

→ Open ***Dockerfile*** in an editor

|  |
| --- |
| FROM maven:3-jdk-8-alpine  RUN apk add ca-certificates  COPY src /usr/src/app/src  COPY pom.xml /usr/src/app  RUN mvn -f /usr/src/app/pom.xml clean package  EXPOSE 8080  ENTRYPOINT ["java","-jar","/usr/src/app/target/flighttracker-1.0.0-SNAPSHOT.jar"] |

#### Step 2: Build the image

→

|  |
| --- |
| docker build -t flighttracker . |

|  |
| --- |
| Sending build context to Docker daemon 85.75MB  Step 1/6 : FROM maven:3.6.1-jdk-8-alpine  ...  Successfully tagged flighttracker:latest |

→ View images

|  |
| --- |
| docker images |

|  |
| --- |
| REPOSITORY TAG IMAGE ID CREATED SIZE  flighttracker latest 13aa72ace76f 5 seconds ago 356MB |

#### Step 3: Run and test the application

→ run our database

|  |
| --- |
| docker run --name mongodb --network=host -d mongo:latest |

|  |
| --- |
| f698d9356cbd9b64edb2d38be0adba25a5bf4a425f5d5536f63fbc71d965e376 |

→ run the app

|  |
| --- |
| docker run -d --network host --name flighttracker flighttracker |

|  |
| --- |
| 53e3b9185e354b63c70146deda0a595ac10dc0044da56e5045006df4706f2e44 |

→ Test the app by opening http://<VM IP>:8080 in a browser window.

It may take the app a minute to initially load in flight data. After data loads, you can click planes on the map to see their information!

→ Terminate the app

|  |
| --- |
| docker stop flighttracker mongodb  docker rm flighttracker mongodb |

### Task 2: Compile the app in separate images

Using multiple stages in our Dockerfile allows us to perform actions in one image from which we can copy various artifacts. You may use one stage that uses an image containing compilers, build libraries, etc to build your application. Then, copy the application binary into another build stage where you run unittest, functional tests, static analysis, etc. Then use a final stage to build your runtime image.

#### Step 1: Update Dockerfile

For more information, see [Dockerfile Reference](https://docs.docker.com/engine/reference/builder/)

**TODO: Makes sure you fill in the dockerfile TODOs below**

→ Open ***Dockerfile****, insert the following*

|  |
| --- |
| FROM maven:3-jdk-8-alpine AS build  RUN apk add ca-certificates  COPY src /usr/src/app/src  COPY pom.xml /usr/src/app  RUN mvn -f /usr/src/app/pom.xml clean package  FROM openjdk:8-alpine  **# TODO: Copy our jar file from the first stage image**  **COPY <TODO>**  EXPOSE 8080  ENTRYPOINT ["java","-jar","/usr/app/flighttracker-1.0.0-SNAPSHOT.jar"] |

#### Step 2: Build image

→ Build the new image

|  |
| --- |
| docker build -t flighttracker . |

→ View images

|  |
| --- |
| docker images |

You should see output similar to the following.

|  |
| --- |
| REPOSITORY TAG IMAGE ID CREATED SIZE  <none> <none> 13aa72ace76f 5 minutes ago 356MB  flighttracker latest 5aa81d3bfc9b 5 seconds ago 145MB |

#### Step 3: Test the app

→ Run the container

|  |
| --- |
| docker run --name mongodb --network=host -d mongo:latest  docker run -d --network=host --name flighttracker flighttracker |

→ Test the app by opening http://<VM IP>:8080 in a browser window.

→ Terminate the app

|  |
| --- |
| docker stop flighttracker mongodb  docker rm flighttracker mongodb |

### Task 3: Composify it

#### Step 1: Make a compose file

Use the [Compose file reference docs](https://docs.docker.com/compose/compose-file/) and any examples in the course materials to make a compose file that defines two services, both our database and application. We want compose to build our flighttracker image for us and be available on port 8080.

→ Open ***docker-compose.yaml***, insert the following

|  |
| --- |
| # TODO: If you would like, update this to the latest version, which may require  # some syntactic changes  version: '3.3'  services:  db:  image: mongo:latest  container\_name: mongodb  ports:  - "27017:27017"    flighttracker:  **# TODO** |

#### Step 2: Deploy the app

→ Run containers

|  |
| --- |
| docker-compose up |

Note that all of the app logs will be dumped to our screen, this is because we did not use the '-d' flag. If we Ctrl-C this or detach from our terminal session, all of the containers will stop executing.

→ Test the app by opening http://<VM IP>:8080 in a browser window.

→ **Ctrl-C** the docker-compose command

#### Step 3: Cleanup

→ Delete containers

|  |
| --- |
| docker-compose down --volumes  docker stop flighttracker mongodb  docker rm flighttracker mongodb |