### 

[What is Docker?](#_i6mbowxfm0ja)

[What are some Docker commands?](#_erbb97rn298h)

[Dockerfile](#_olrybbibr02c)

[Pre-requisites](#_isv7twe0ooj3)

[Activity 1:Exploring docker run](#_65fql230rrlg)

[Activity 2: Exploring Docker Compose and Exec](#_hhizggmpp6v9)

### What is Docker?

Docker is a platform that enables developers to package, distribute, and run applications in a containerized environment. Containers are lightweight and isolated environments that package an application along with its dependencies and configuration, ensuring that the application runs consistently regardless of the environment it is deployed in.

Docker provides a way to create, manage, and run containers using a set of tools and APIs. With Docker, developers can build an application in one environment and deploy it in any other environment, without worrying about dependencies or compatibility issues.

Users can write their application in any programming language, and on top of that, they need to provide a set of instructions on how to compile their code, binaries and libraries into one lightweight container as well as how to start the container using that image.

Docker uses a client-server architecture, where the Docker client communicates with the Docker daemon to build, run, and manage containers. The Docker daemon runs on the host machine and manages the container lifecycle, including creating, starting, stopping, and deleting containers.

Docker has become very popular among developers and DevOps teams, as it simplifies the deployment and management of applications across different environments and infrastructure. It is widely used in modern application development and deployment workflows, such as microservices, continuous integration, and continuous deployment.

Look at this link to see what kind of online resources you can build with Docker:

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

### 

### What are some Docker commands?

Some popular commands that you should be familiar with include:

docker run - This command creates and starts a new container from a Docker image.

docker ps - This command lists all running containers.

docker images - This command lists all the images that are stored locally on your system.

docker stop - This command stops a running container.

docker rm - This command removes one or more containers.

docker rmi - This command removes one or more images.

docker build - This command builds a Docker image from a Dockerfile.

docker exec - This command runs a command inside a running container.

#### 

### Dockerfile

Next, let’s start exploring Dockerfile. Dockerfile is a **script that defines how to build a Docker image for your application**. It contains a set of instructions that specify the environment, dependencies, and configuration of the application or service that the Docker image will contain.

A Dockerfile starts with a FROM instruction, which specifies the base image to use. The base image provides the underlying environment and dependencies that the application or service will run on.After the FROM instruction, the Dockerfile can include a series of other instructions, such as RUN, COPY, and CMD, which are used to configure the environment, install dependencies, copy files into the image, and specify the command to run when the container is started.

The Dockerfile is used by the docker build command to create a Docker image. When the docker build command is run, it reads the Dockerfile and executes each instruction in order, building up the image layer by layer.

| FROM | The FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions. |
| --- | --- |
| WORKDIR | The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it. |
| COPY | The COPY instruction copies files from a local source location to a destination. |
| RUN | The RUN instruction will execute any commands in a new layer on top of the current image and commit the results. |
| EXPOSE | The EXPOSE instruction exposes a particular port inside a Docker container |
| CMD | There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect. The main purpose of a CMD is to provide defaults for an executing container. |

### 

### Pre-requisites

We would be setting up an EC2 as our working machine, with docker and git installed in it(Similar to lesson 3.4), just that you would be setting up and configuring the EC2 using Terraform.

Create a new public repository on Github, add a README.md and for the .gitignore template, choose Terraform. Then clone the repo to your local computer and open it in your VSCode.

Create a **main.tf** for your main infra code and **backend.tf** to specify your terraform state file. And a **provider.tf** file to specify your aws region as **“ap-southeast-1”.**

So you should have the following 5 files:

**main.tf**

**provider.tf**

**backend.tf**

**.gitignore**

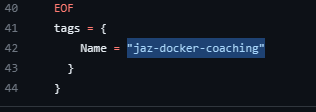
**README.md**

You can refer to this: [jaezeu/docker-ec2 (github.com)](https://github.com/jaezeu/docker-ec2)

You just have to make the following 2 changes:

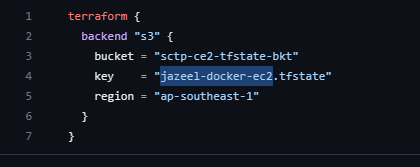
**In main.tf**

Change the value of Name to something that’s unique to you(Only change within the quotes). This would be the name of your EC2



**In backend.tf**

Change the key to a unique name as well. The key should still end with .tfstate



Once the above 2 changes have been made, run the following commands in the following order:

**terraform init**

**terraform plan**

**terraform apply**

Once the apply is complete , look for your EC2 in the AWS Console and connect to it using EC2 Instance Connect. (Right-click on your EC2 -> Connect -> Connect)

Then run these 2 commands inside the EC2:

**sudo su**

**systemctl status docker**

**For the activities below, it would be solely performed inside the EC2**

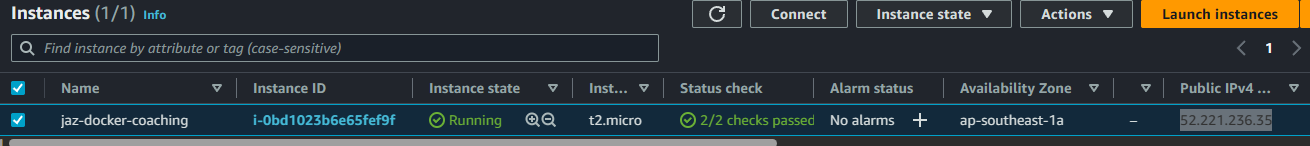
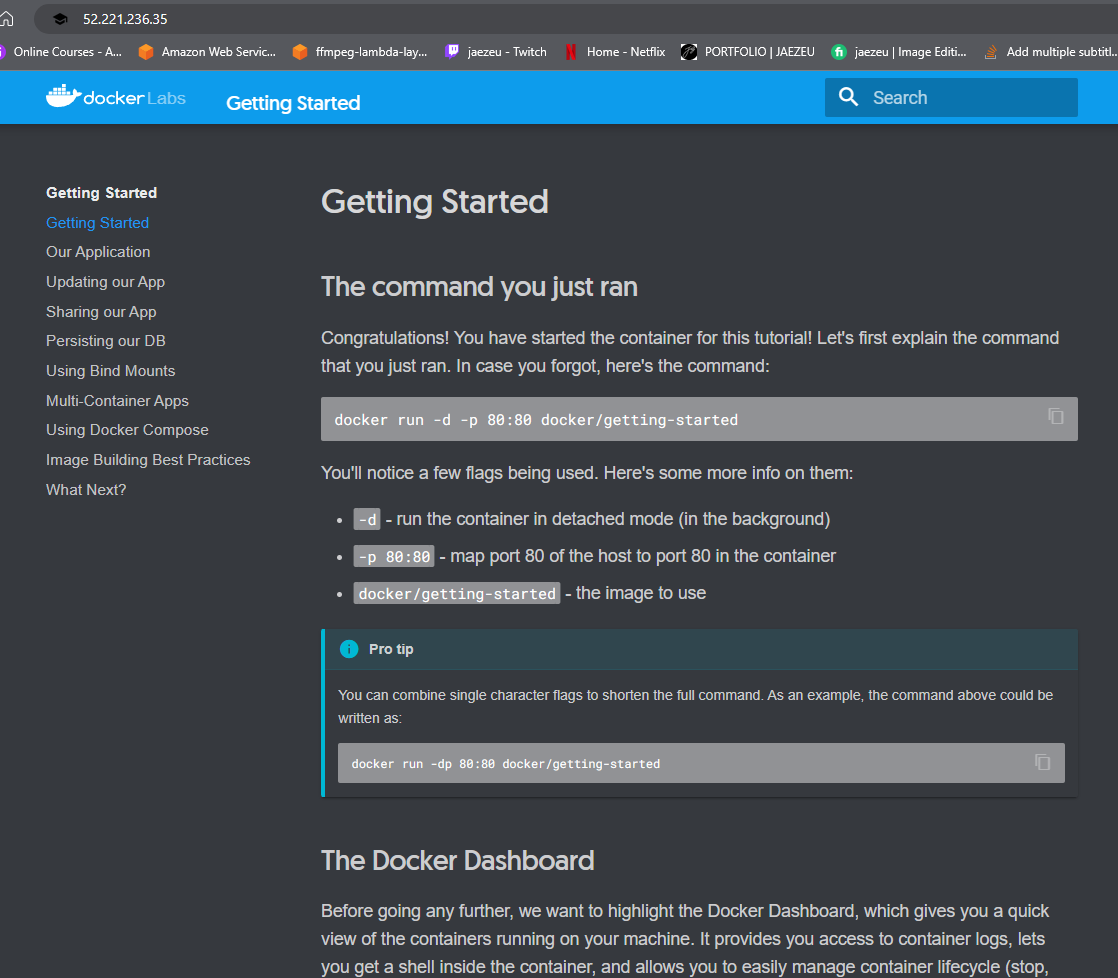
#### 

#### Activity 1:Exploring docker run

1. Run the following command to get started with a sample docker container:

docker run -d -p 80:80 docker/getting-started

* 1. docker run - Creates and starts a new container from the Docker image.
  2. -p 80:80 - Maps port 80 on the host machine to port 80 in the container. This enables the container to receive HTTP traffic on port 80.
  3. docker/getting-started - Specifies the Docker image to use for the container. In this case, the image is docker/getting-started.

1. To check if you container is running, run: docker ps
   1. This should show the container that you just ran as the output.
2. Get your EC2 public IP (You can get it from the EC2 console)
3. 
4. Open a new browser tab and paste the IP address
5. 
6. To stop the container, run the command docker stop <container ID>

#### Activity 2: Exploring Docker Compose and Exec

Another way to run docker would be via Docker Compose.

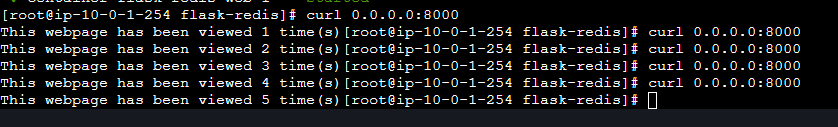
Docker Compose is a tool that allows you to **define and run multi-container Docker applications**. It uses a YAML file to define the services, networks, and volumes that make up your application stack and provides a simple command-line interface to start, stop, and manage your application.

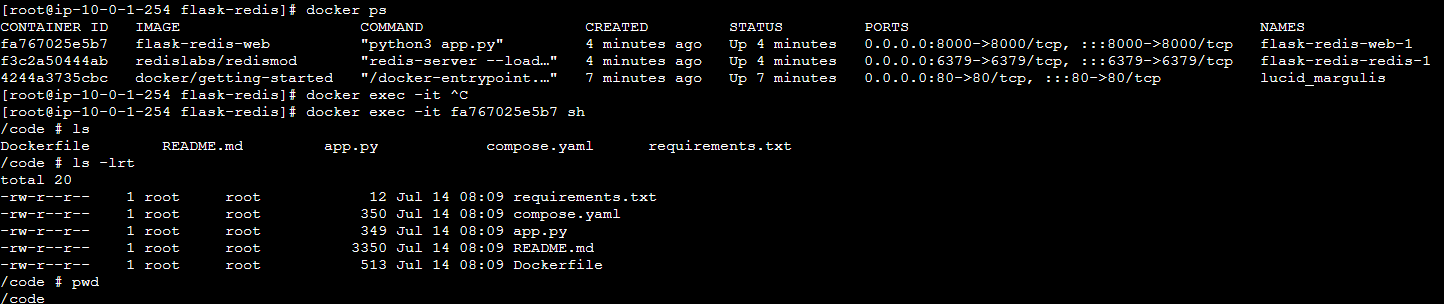
With Docker Compose, you can define your application stack as a set of interconnected services, each running in its own container. You can specify the configuration for each service, such as the container image to use, the environment variables to set, and the volumes to mount. You can also specify dependencies between services, such as a database service that is required by your web application.

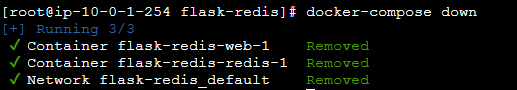
Docker Compose also provides features for managing your application's network and storage requirements. You can define custom networks to isolate your services and specify which containers can access each other. You can also define volumes to persist data between container restarts and to share data between services.

Let’s begin!

1. Clone the docker examples repo by running: git clone https://github.com/docker/awesome-compose.git. Here, you will see different examples of Docker for future use, but let’s run cd awesome-compose/flask-redis
2. Let’s look at the file compose.yaml (run **cat compose.yaml**)
3. You will notice that there are 2 services being built here:
   1. Redis running the image redislabs/redismod on port 6379
   2. Web application that is dependent on Redis running on port 8000 that will output the number of times the website has a view.
4. To run docker via docker compose, run the command docker-compose up -d
5. You will see that instead of just building the image, this command will also run the containers that you have specified.
6. Once the command has finished running, run docker ps to see your containers
7. Once done, run curl 0.0.0.0:8000. you can see the counter going up each time you run the curl command



1. Let’s try executing into the docker container via shell. To do this, run the command docker exec -it <container ID> sh. This would allow you to see the files within the container. You can run linux commands such as pwd and ls -lrt
2. To stop docker containers, run docker-compose down



1. Run docker ps to verify that all your containers have stopped running.