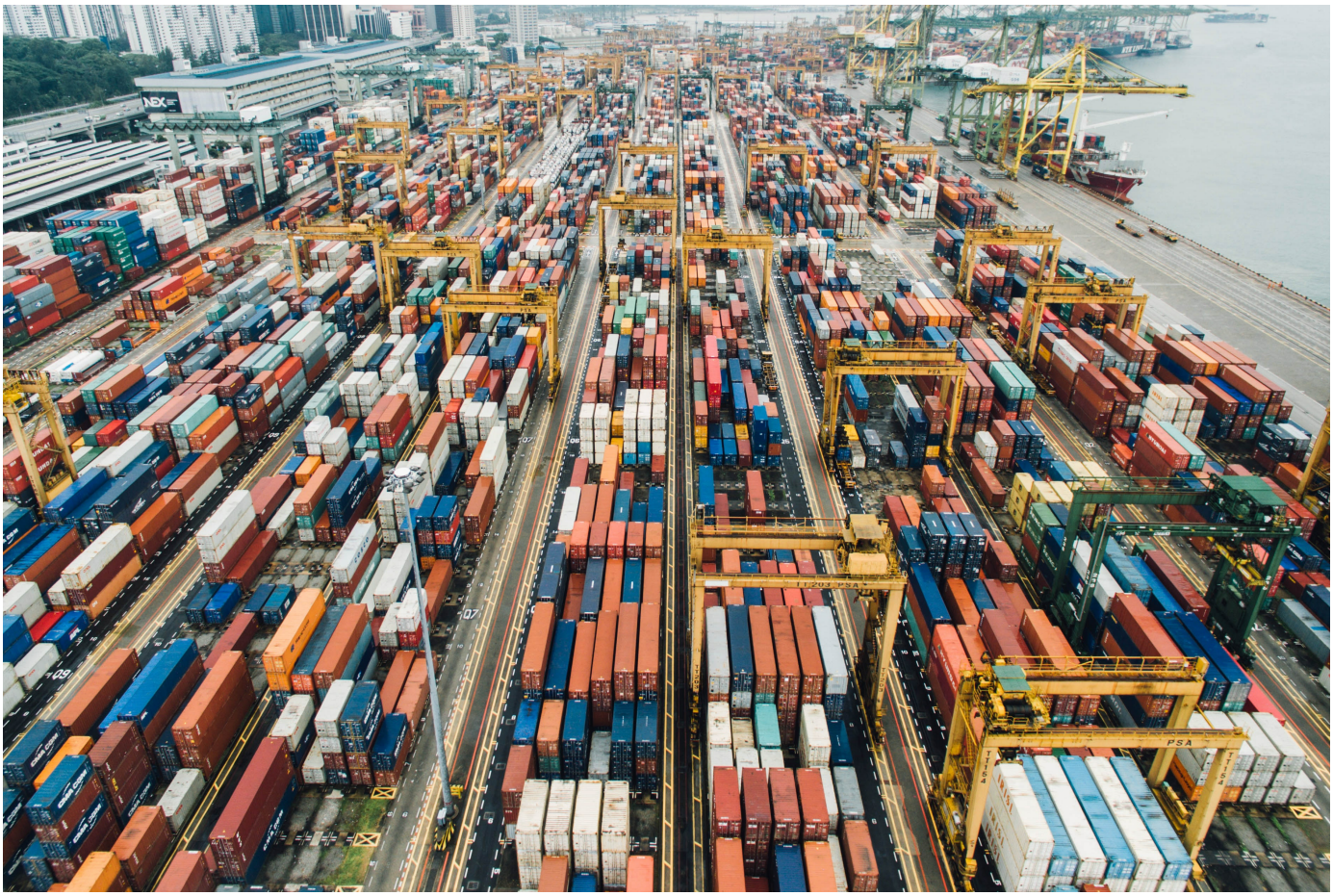
Docker for Beginners

Introduction

Docker is a tool that makes application deployment easier with the use of containers. Containers allows a developer to package an application along with its required libraries and system dependencies in an isolated environment, containers can run on a variety of operating system enviroment with varying configurations which help in limiting the “It works on my Machine” Problem developers experiance.

Docker Containers



Containers allow you to package any application with the right configurations. You could add the dependencies of the application in the container itself. This way you could save developers the time and frustration involved in finding and installing dependencies of your application. You could open ports on the container to allow the application to communicate with the applications on the host or communicate with other running containers.

Containers are very useful in deploying microservices, containers are designed to package and run a single service. Each container has its own independent Libraries and Dependencies, in case another container goes down it does not affect other containers, because they can all run independently.

Docker Images

A Docker Image is a file that contains a series of instruction that describes how a container should be built, it specifies the libraries and dependecies required by an application to run. An intance of a docker image is a container, and multiple containers can be created from a single image. Images are usually stored in a registry like Docker hub.

> ***Note*** *
*
> *An instance of an image is called a container. You have an image, which is a set of layers as you describe. If you start this image, you have a running container of this image. You can have many running containers of the same image.*

Step 1 - Building our Hello World App

The application we will be building will print “Hello World” along with the current time using PHP. To get started, create a new directory named `/public` in your preferred workspace, then create a new file titled `index.php` in the `/public` directory and paste the following code snippet in the file:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
  </head>
```

```
  <body>
    Hello World. The time is <?php echo date("h:i:sa"); ?>
  </body>
</html>
```

Step 2 - Creating a Docker container

In this step, we will create a new file `Dockerfile` while will be our image that contains the instructions needed to package our application in a container.

```
FROM php:7.2-apache
COPY public/ /var/www/html/
```

Let us break down what these lines in our `Dockerfile` will do when executed. `FROM php:7.2-apache` pulls the APACHE server image from Docker Hub with PHP 7.2 pre-configured, `COPY public/ /var/www/html/` copies the contents in `/public` directory into `/var/www/html/` directory which is a virtual directory in our container that APACHE will try to serve files from.

Step 3 - Building our Dockerfile

To build our Dockerfile we need to run the following in the root directory of out project:

```
docker build -t docker-php-hello-world .
```

The `docker build` command builds the `Dockerfile` and the argument `-t` tags the name of the image as `docker-php-hello-world` while the `.` indicates that the Dockerfile is in the current directory along with other files required in the `Dockerfile` context.

```
Sending build context to Docker daemon  104.4kB
Step 1/2 : FROM php:7.2-apache
 ---> 5093fdd5bd47
Step 2/2 : COPY public/ /var/www/html/
 ---> Using cache
 ---> c1be6ef449fc
Successfully built c1be6ef449fc
Successfully tagged docker-php-hello-world:latest
```

Step 4 - Building and running our container

To run our container, we must execute the following command:

```
docker run --name=docker-php-hello-world -d --rm -p 90:80
docker-php-hello-world
```

The `docker run` command tries to run the container created from the `Dockerfile` image in that directory, `--name=docker-php-hello-world` gives the container a name; in this case it is named `docker-php-hello-world`, `-d` argument runs the container in detached mode. In other words, the container will run in he background, `--rm` argument will ensure that the container is destroyed when it is stopped, this is

good to save disk memory, `-p 90:80` argument binds the internal APACHE port of `80` to an external port of `90` so that our container can be accessed for port `90` and finally `docker-php-hello-world` which is the Image we are building our container from.

This will generate a container from the `Dockerfile` and the name of the container will be `docker-php-hello-world`. To view the container details run:

```
docker ps
```

This will return a table view of the container attributes.

Step 5 - Accessing our application

At this point our application is running in a docker container that can be accessed from port `90`, by visiting 0.0.0.0:90 on a web browser, we should be able to view our application.

As shown in the Image above, our application is accessible via port `90`.

Step 6 - Stopping the container

We can stop the running container by running a command in this format:

```
docker stop [CONTAINER HASH OR CONTAINER NAME]
```

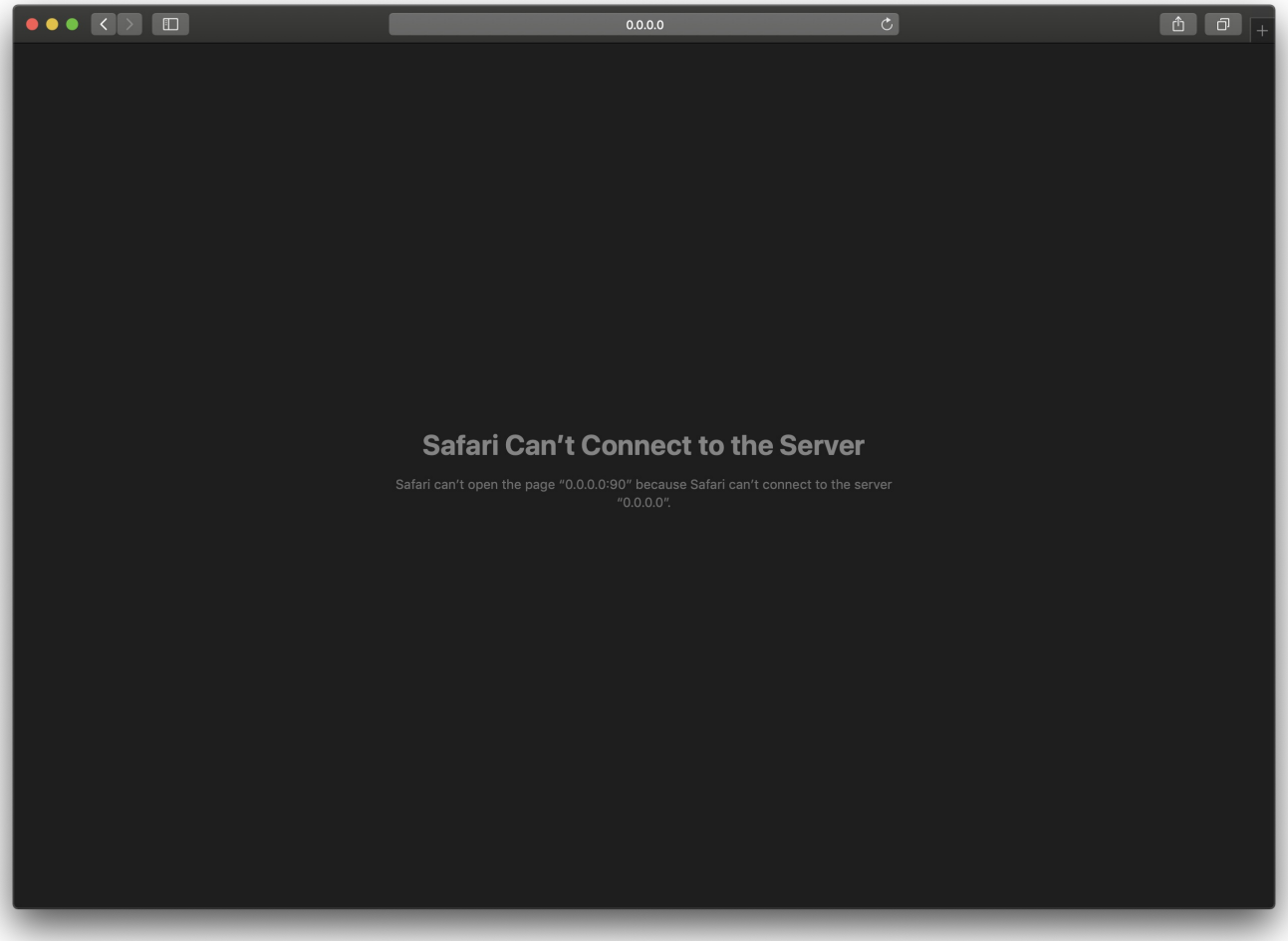
Therefore, by running:

```
docker stop docker-php-hello-world
```

Our application container will be stopped and removed because in Step 4 we specified that the container should be destroyed when it is stopped. So if we run `docker ps` we will not find any container listed.

```
▲  docker-php-hello-world git:(master) docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
```

If we try visiting our applicatio on port `90` it will be unreachable because the container has been stopped and destroyed.



At this point you have learnt the very basics of Docker.

Conclusion

Docker allows super elegant deployment of your applications with the use of containers. Applications can be packaged and distributed across multiple enviroment without any worries and removes setup time from installing dependecies for a projec on development and in production.