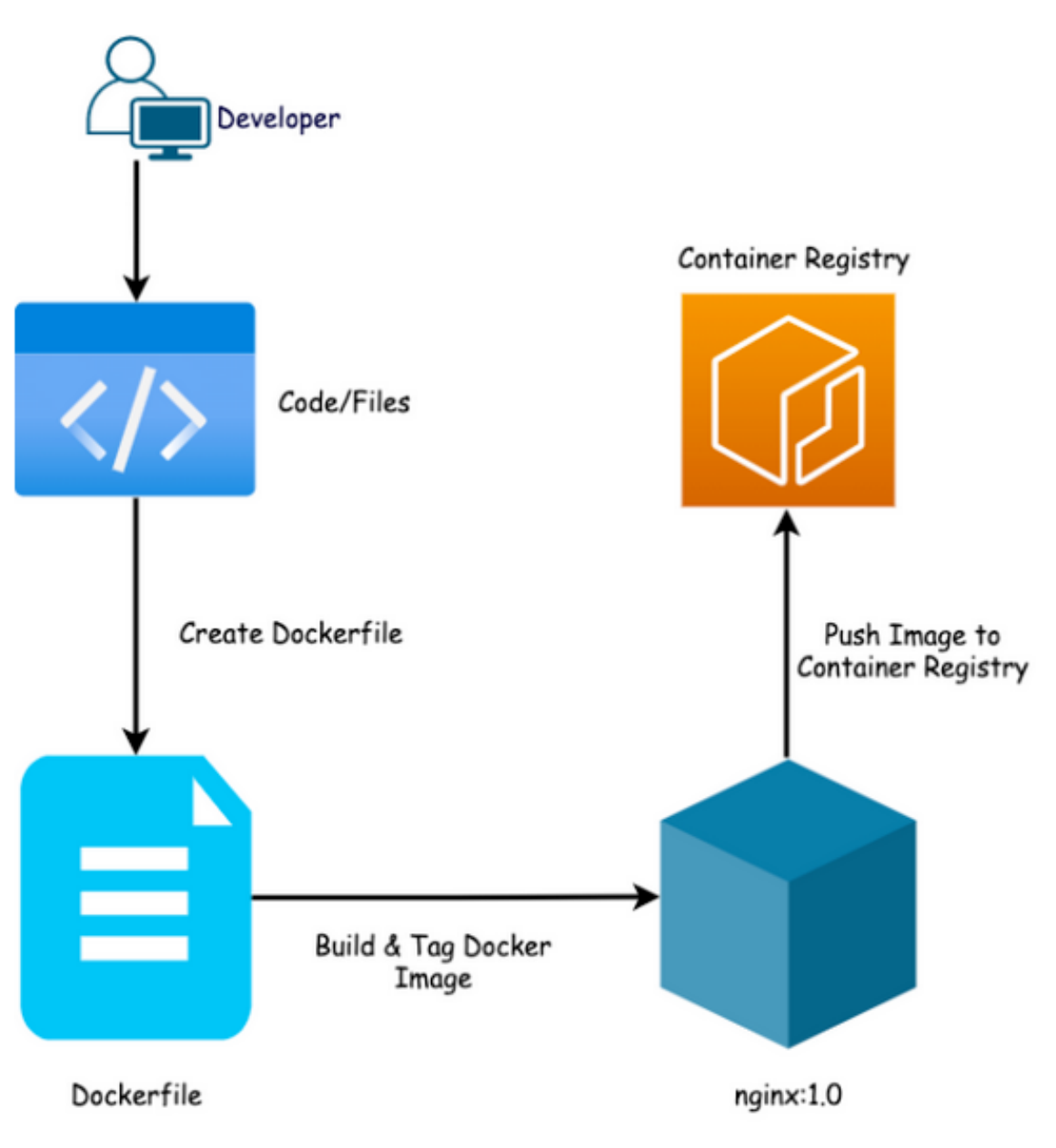
~~Dockerfile Explained~~ ✓
Build Docker Image Using Dockerfile ○
Push Docker Image To Docker Hub ○
Docker Multistage Build ○
Optimize Docker Image ○
Using heredoc With Dockerfile ○
ENTRYPOINT vs CMD ○
Dockerfile Best Practices ○
Useful Docker Commands ○
Dockerizing Applications 1 Topic ○
Real World Docker Tasks 4 Topics ○
Linting Dockerfiles 5 Topics ○
Docker Image Vulnerability Scanning 4 Topics ○
Podman
Podman Basics 9 Topics ○
Container Standards
OCI (Open Container Initiative) ○

Build Docker Image Using Dockerfile

LESSON PROGRESS

11% Complete

In this topic, you will learn to build a docker image using a real-world example. We will create an Nginx docker image from scratch with a custom index page.



Follow the steps given below to build a docker image.

> **Note:** The Dockerfile and configs used for this article is hosted on a Docker image examples Github repo. You can clone the repo for reference.

Step 1: Create the required Files and folders

Create a folder named `nginx-image` and create a folder named `files`

```
mkdir nginx-image && cd nginx-image
mkdir files
```

Create a . `dockerignore` file

```
touch .dockerignore
```

Step 2: Create a sample HTML file & Config file

When you build docker image for real-time projects, it contains code or application config files.

For demo purposes, we will create a simple HTML file & config file as our app code and package it using Docker. This is a simple `index.html` file. You can create your own if you want.

cd in to the files folder

```
cd files
```

Create an `index.html` file

```
vi index.html
```

Copy the following contents to the `index.html` and save the file.

```
<html>
  <head>
    <title>Dockerfile</title>
  </head>
  <body>
    <div class="container">
      <h1>My App</h1>
      <h2>This is my first app</h2>
      <p>Hello everyone, This is running via Docker container</p>
    </div>
  </body>
</html>
```

create a file name default

```
vi default
```

Copy the following contents to the default file.

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /usr/share/nginx/html;
    index index.html index.htm;
    server_name _;
    location / {
        try_files $uri $uri/ =404;
    }
}
```

Step 3: Choose a Base Image

We use `FROM` command in the `Dockerfile` which instructs Docker to create an image based on other images which are available on the Docker hub or any container registry configured with Docker. We call it a **base image**.

Choosing a base image depends on our application and os platform of choice. In our case, we will pick the `ubuntu:18.04` base image.

> **Note:** Always use offical base images for your applications to avoid potential vulnerabilities. Towards the end, we have added all the public registries that has verified container base images. Also, when it comes to **production use cases**, always use a minimal base image like alpine (**just 5 Mib**) or distroless images. Distroless alpine is **just 2 MiB**

Step 3: Create the Dockerfile

Create a Dockerfile in the `nginx-image` folder.

```
vi Dockerfile
```

Here's the simple `Dockerfile` for our use case.

```
# Base image
FROM ubuntu:latest

# Use LABEL for metadata as recommended
LABEL maintainer="contact@devopscube.com"

# Install nginx
RUN apt-get update -y && apt-get install -y nginx

# Copy configuration files and content
COPY files/default /etc/nginx/sites-available/default
COPY files/index.html /usr/share/nginx/html/index.html

# Expose port 80
EXPOSE 80

# Start nginx
CMD ["/usr/sbin/nginx", "-g", "daemon off;"]
```

Here's the explanation of each step:

1. `FROM` instruction will pull the Ubuntu latest version Image from the Docker hub.
2. With `LABEL` instruction, we are adding metadata about the maintainer. It is not a mandatory instruction.
3. The we're installing Nginx using the RUN instruction
4. Then we're copying the Nginx default config file from the local `files` directory to the target image directory.
5. Next, we're copying our `index.html` file from the local `files` directory into the target image directory. It will overwrite the default index.html file created during Nginx installation.
6. We're exposing port 80 as the Nginx service listens on port 80.
7. At last, we're running the Nginx server using `CMD` instruction when the Docker image launches.

For Docker containers, the `daemon off;` directive tells Nginx to stay in the foreground. It disables the self-daemonizing behavior of Nginx. the `-g` option specifies a directive to Nginx.

The reason we run the process in the foreground is to attach the console process to standard input, output, and error.

> Note: In this Dockerfile, the entrypoint is not explicitly defined using the `ENTRYPOINT` instruction. However, the `CMD` instruction specifies the command to be executed when a container is started using the image.

Step 4: Build your first Docker Image

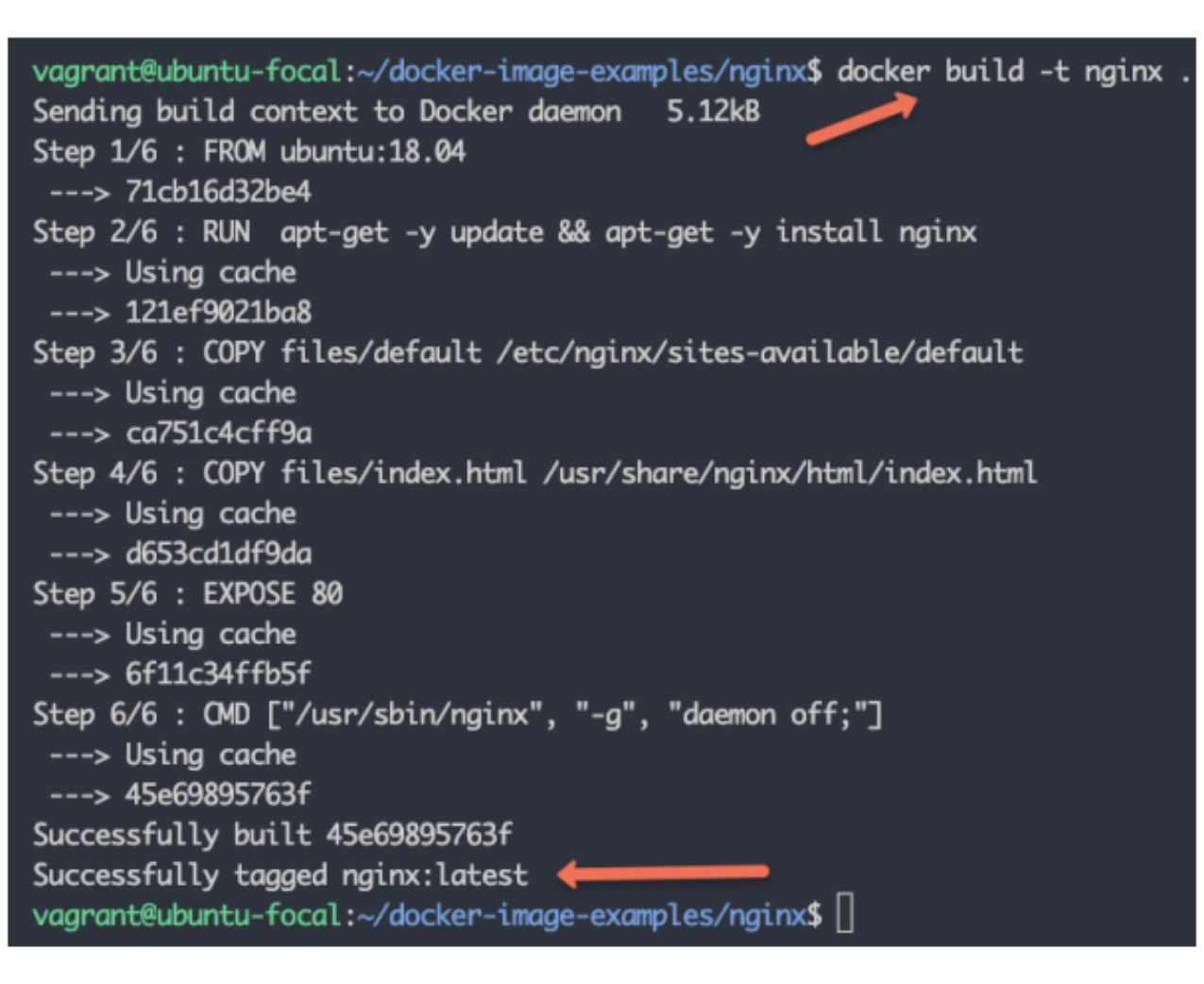
The folder & file structure would look like the following.

```
nginx-image
├── Dockerfile
└── files
    ├── default
    └── index.html
```

Now, we will build our image using the Docker command. The below command will build the image using `Dockerfile` from the same directory.

```
docker build -t nginx:1.0 .
```

1. -t is for tagging the image.
2. `nginx` is the name of the image.
3. `1.0` is the tag name. If you don't add any tag, it defaults to the tag named latest.
4. . means, we are referring to the Dockerfile location as the docker build context.



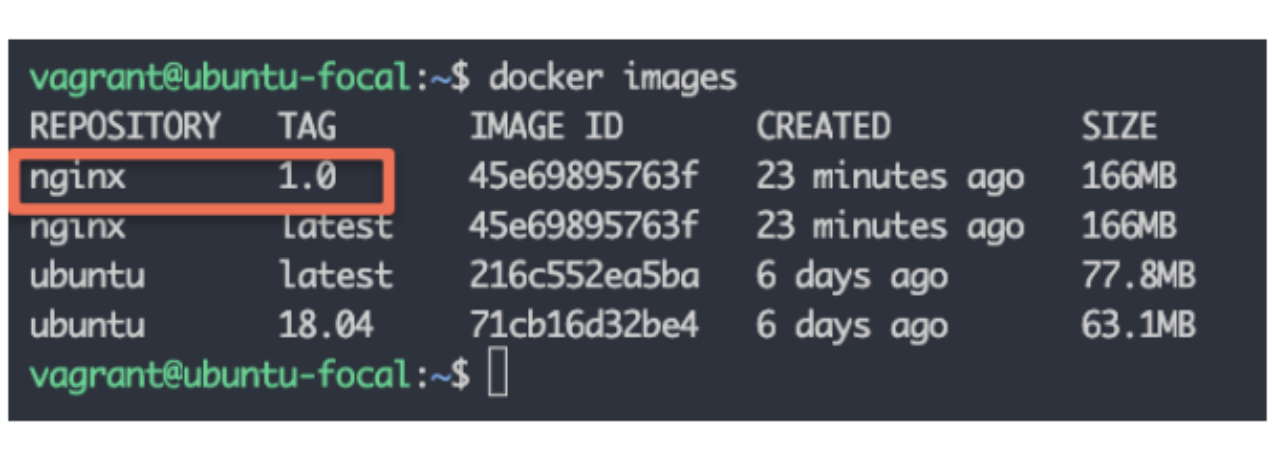
```
vagrant@ubuntu-focal:~/docker-image-examples/nginx$ docker build -t nginx .
Sending build context to Docker daemon   5.12kB
Step 1/6 : FROM ubuntu:18.04
 ---> 71cb16d32be4
Step 2/6 : RUN  apt-get -y update && apt-get -y install nginx
 ---> Using cache
 ---> 121ef9021ba8
Step 3/6 : COPY files/default /etc/nginx/sites-available/default
 ---> Using cache
 ---> ca751c4cff9a
Step 4/6 : COPY files/index.html /usr/share/nginx/html/index.html
 ---> Using cache
 ---> d653cd1df9da
Step 5/6 : EXPOSE 80
 ---> Using cache
 ---> 6f11c34ffb5f
Step 6/6 : CMD ["/usr/sbin/nginx", "-g", "daemon off;"]
 ---> Using cache
 ---> 45e69895763f
Successfully built 45e69895763f
Successfully tagged nginx:latest
vagrant@ubuntu-focal:~/docker-image-examples/nginx$ 
```

If the `Dockerfile` is in another folder then you need to specify it explicitly.

```
docker build -t nginx /path/to/folder
```

Now, we can list the images by using this command.

```
docker images
```



```
vagrant@ubuntu-focal:~$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
nginx        1.0       45e69895763f   23 minutes ago   166MB
nginx        latest    45e69895763f   23 minutes ago   166MB
ubuntu       latest    216c552ea5ba   6 days ago       77.8MB
ubuntu       18.04     71cb16d32be4   6 days ago       63.1MB
vagrant@ubuntu-focal:~$ 
```

We can see the tag is `1.0` here. If we want to put a specific tag we can put it like this `image-name:<tag>` . If you don't specify any tag, it defaults to `latest` tag.

```
docker build -t nginx:2.0 .
```

A single image can have multiple tags. There are two approaches we generally follow to tag the image:

1. **Stable Tags** – We can continue to pull a specific tag, which continues to get updates. Our tags are always constant, but the image content is changed.
2. **Unique Tags** – We use a different and unique tag for each image. There are different ways to provide unique tags for example date-time stamp, build number, commit id, etc.

> **Note**: When it comes to prodcution, a recomended docker image tagging method is semantic versioning (Semver).

Docker caches the build steps. So if we build the image again the process will move a little faster. For example, it will not download the ubuntu 18.04 image again.

Using large images slows down the build and deployment time of containers. If you want to learn more about optimizing Docker images, check out reduce docker image guide.

Step 5: Test the Docker Image

Now after building the image we will run the Docker image. The command will be

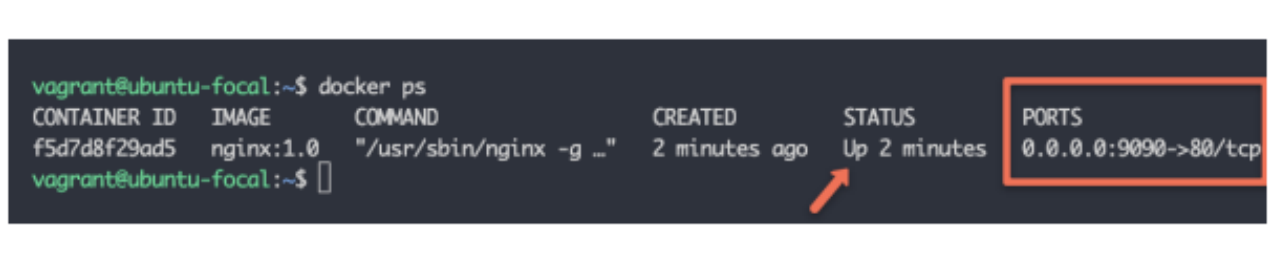
```
docker run -d -p 9090:80 --name webserver nginx:1.0
```

Here,

1. `-d` flag is for running the container in detached mode
2. `-p` flag for the port number, the format is local-port:container-port
3. `--name` for the container name, webserver in our case

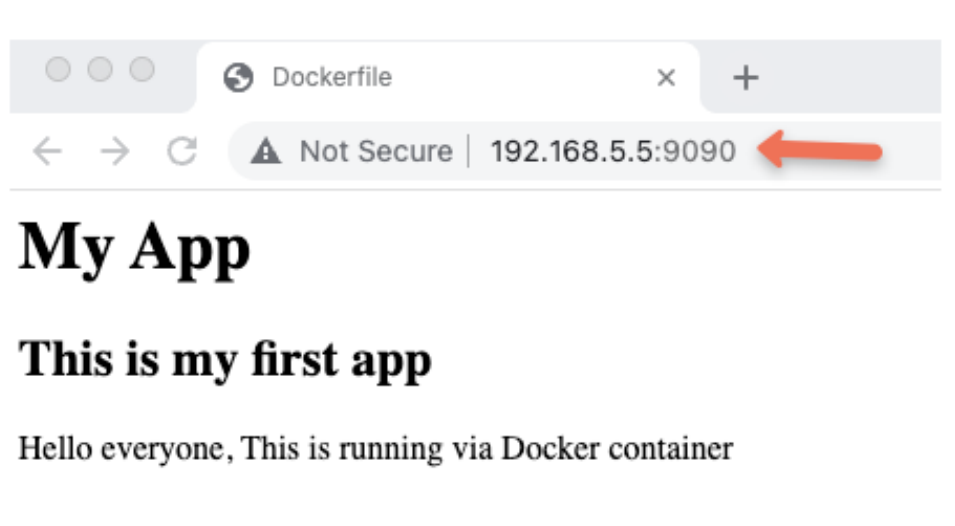
We can check the container by using the below command

```
docker ps
```



```
vagrant@ubuntu-focal:~$ docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS
f5d7d8f29ad5   nginx:1.0   "/usr/sbin/nginx -g …"   2 minutes ago   Up 2 minutes   0.0.0.0:9090->80/tcp
vagrant@ubuntu-focal:~$ 
```

Now in the browser, if you go to `http://<host-ip>:9090` , you can see the index page which displays the content in the custom HTML page we added to the docker image.



My App

This is my first app

Hello everyone, This is running via Docker container

Mark Complete