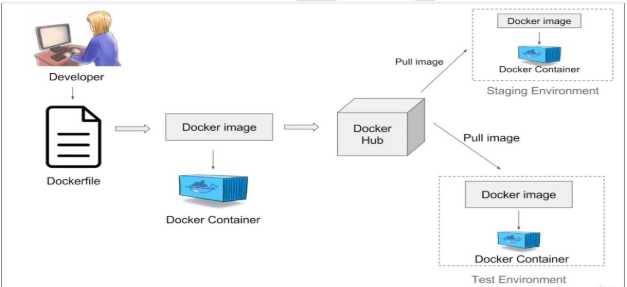
 **DOCKER TUTORIAL – 3**

**What you will Learn in this Tutorial:**

* **Dockerfile Concepts**
* **Dockerfile Keywords**
* **How to write your first Dockerfile**
* **Docker images**
* **Docker container**
* **Docker Hub Overview**
* **Docker Hub account integration login**
* **Push Image to Docker hub**
* **Pull images**

**Introduction**  we will cover the basics of Docker, including Docker architecture, Docker commands, and how to create and containerize your first application.

****

**Understanding Docker Architecture** Before we dive into the tutorial, let's take a look at the Docker architecture. Docker is a containerization platform that allows you to run applications in isolated environments called containers. Containers are lightweight and portable, making it easy to deploy applications across different environments.

**Lab Setup** For this tutorial, we will be using a Linux machine with Docker installed. If you don't have Docker installed, you can download it from the official Docker website.

**Docker Commands** Let's start with some basic Docker commands. To list all running containers, use the following command:

docker ps

**Dockerfile  
Docker file  
FROM : Node11 , and mysql can be multiple base images**

**MAINTAINER : Aseem**

**COPY : . .**

**Target/app.jar /usr/app/tomcat/webapp.war**

**Source destinated source**

**Host mc container mc**

**RUN : docker images**

**RUN : ‘apt update -y’**

**RUN : ‘git clone : url ‘**

**RUN : ‘sudo apt install git’**

**CMD : During Container   
  
CMD ‘npm’ ‘start’  
CMD ‘npm’ ‘install’**

**EXPOSE : 3002 knowleged & docs purpose**

**WORKDIR : path of work dir**

**/opt/app**

**/var/www/app**

**ENTRYPOINT : ‘apt update -y’**

**‘npm install’**

**‘npm build’**

Let me explain what each line does:

* **FROM python:3.9-slim**: This line sets the base image for our Docker image. In this case, we're using the official Python 3.9 image with the **slim** tag, which is a smaller version of the image.
* **MAINTAINER Your Name <your.email@example.com>**: This line sets the maintainer of the Docker image. You should replace **Your Name** and **your.email@example.com** with your own information.
* **WORKDIR /app**: This line sets the working directory in the container to **/app**.
* **COPY requirements.txt .**: This line copies the **requirements.txt** file from the current directory (i.e., the directory containing the Dockerfile) into the container at the current working directory (**/app**).
* **RUN pip install -r requirements.txt**: This line installs the dependencies specified in **requirements.txt** using pip.
* **COPY . .**: This line copies the application code from the current directory into the container at the current working directory (**/app**).
* **EXPOSE 8000**: This line exposes port 8000 from the container to the host machine.
* **CMD ["python", "app.py"]**: This line sets the default command to run when the container starts. In this case, it runs the **app.py** file using Python.
* **ENTRYPOINT ["python"]**: This line sets the entry point for the container. In this case, it sets the entry point to the Python interpreter.

ENTRYPOINT is a Dockerfile instruction that allows you to configure a container to run a specific command when it starts. It's similar to the CMD instruction, but with some key differences.

You should use ENTRYPOINT when:

You want to ensure that a specific command is always executed when a container starts, and you don't want users to be able to override it.

You want to create a container that runs a specific application or process, and you want to ensure that it's always executed in a consistent way.

# Dockerfile

# Set the base image

FROM python:3.9-slim

# Set the maintainer

MAINTAINER Your Name <your.email@example.com>

# Set the working directory

WORKDIR /app

# Copy the requirements file

COPY requirements.txt .

# Install the dependencies

RUN pip install -r requirements.txt

# Copy the application code

COPY . .

# Expose the port

EXPOSE 8000

# Set the command to run when the container starts

CMD ["python", "app.py"]

# Set the entry point

ENTRYPOINT ["python"]

To list all containers, including stopped containers, use the following command:

To start a new container, use the following command:

docker run -it ubuntu /bin/bash

This command will start a new container from the Ubuntu image and open a bash shell.

**Create and Containerize Your First Application** Let's create a simple web application using Node.js and containerize it using Docker. First, create a new directory for your project and create a file called **app.js** with the following code:

const http = require('http');

http.createServer((req, res) => {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Hello World\n');

}).listen(3000, 'localhost');

console.log('Server running at <http://localhost:3000/>');

Next, create a file called **Dockerfile** with the following code:

FROM node:14

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN npm run build

EXPOSE 3000

CMD ["npm", "start"]

This Dockerfile will create a new image from the Node.js 14 image, copy the **package.json** file and install the dependencies, copy the application code, build the application, expose port 3000, and set the default command to start the application.

**Build and Run the Docker Image** To build the Docker image, use the following command:

docker build -t myapp .

This command will build a new image with the name myapp. To run the image, use the following command:

This command will start a new container from the **myapp** image and map port 3000 on the host machine to port 3000 in the container.

**Overview of Docker Hub** Docker Hub is a registry of Docker images. You can push your images to Docker Hub and share them with others. To push an image to Docker Hub, use the following command:

docker tag myapp:latest <your-username>/myapp:latest

docker push <your-username>/myapp:latest

This command will tag the **myapp** image with your username and push it to Docker Hub.

That's it for this tutorial! I hope you learned something new about Docker and how to create and containerize your first application. Thanks for watching!

Tag Your Image

Next, tag your image with your Docker Hub username and the desired repository name. Assuming the image you want to push is named newimage, you should tag it as follows:

docker tag newimage aseemakram19/newimage

Push the Tagged Image

Now, push the tagged image to your Docker Hub repository:

sh

Copy code

docker push aseemakram19/newimage

Complete Example

Here is a complete example with all steps combined: