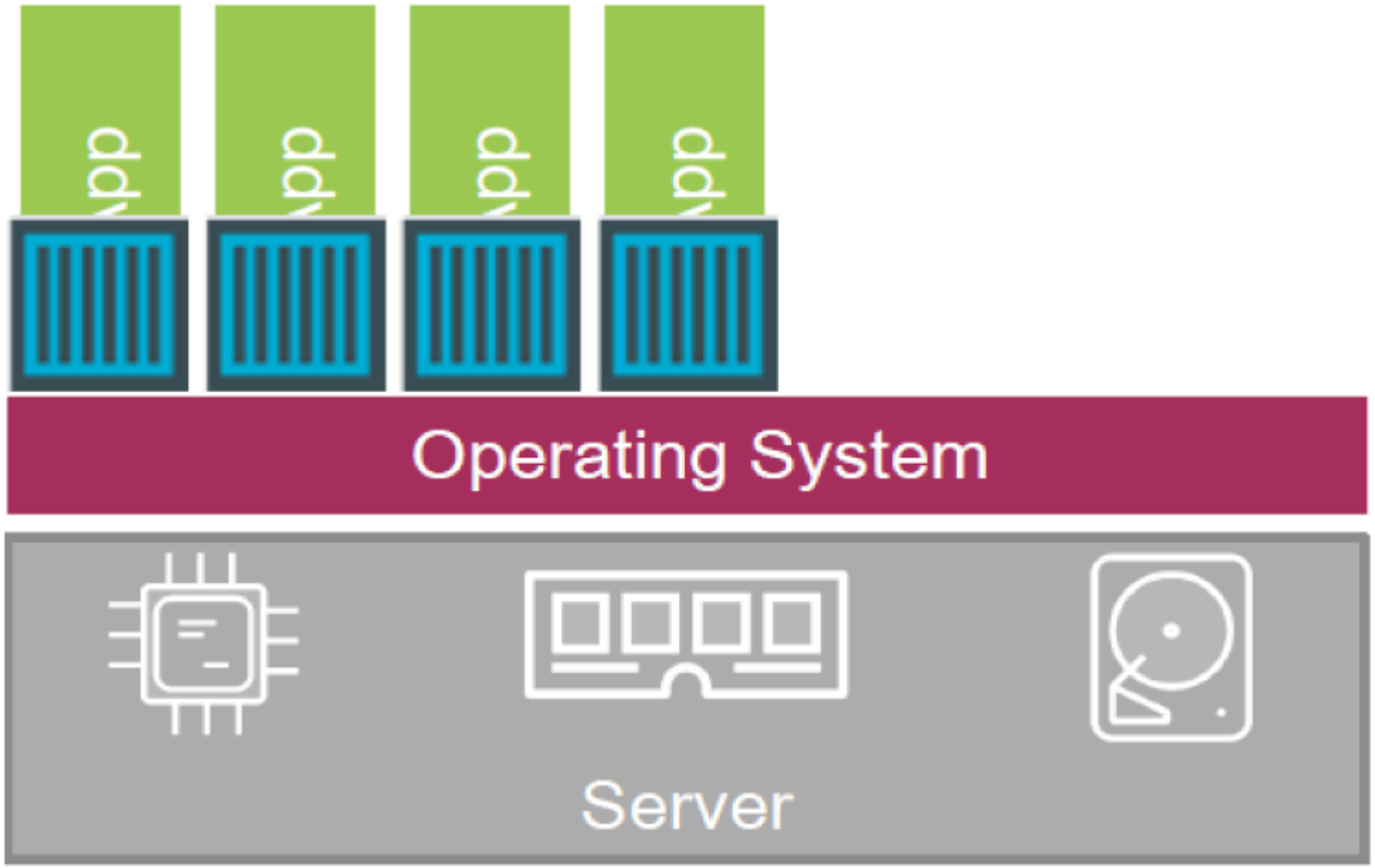
Welcome to Dockerization

What is a container ?

- A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
- A container is a bundle of Application, Application libraries required to run your application and the minimum system dependencies.

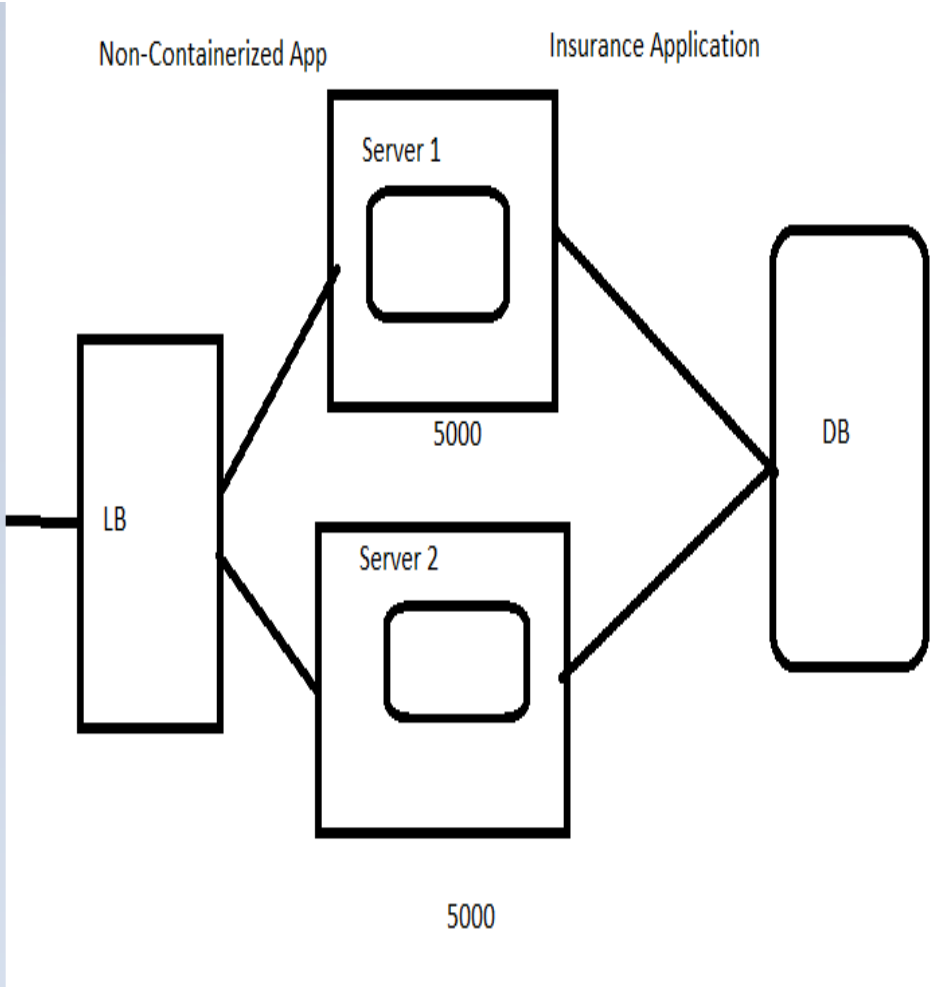
Containers

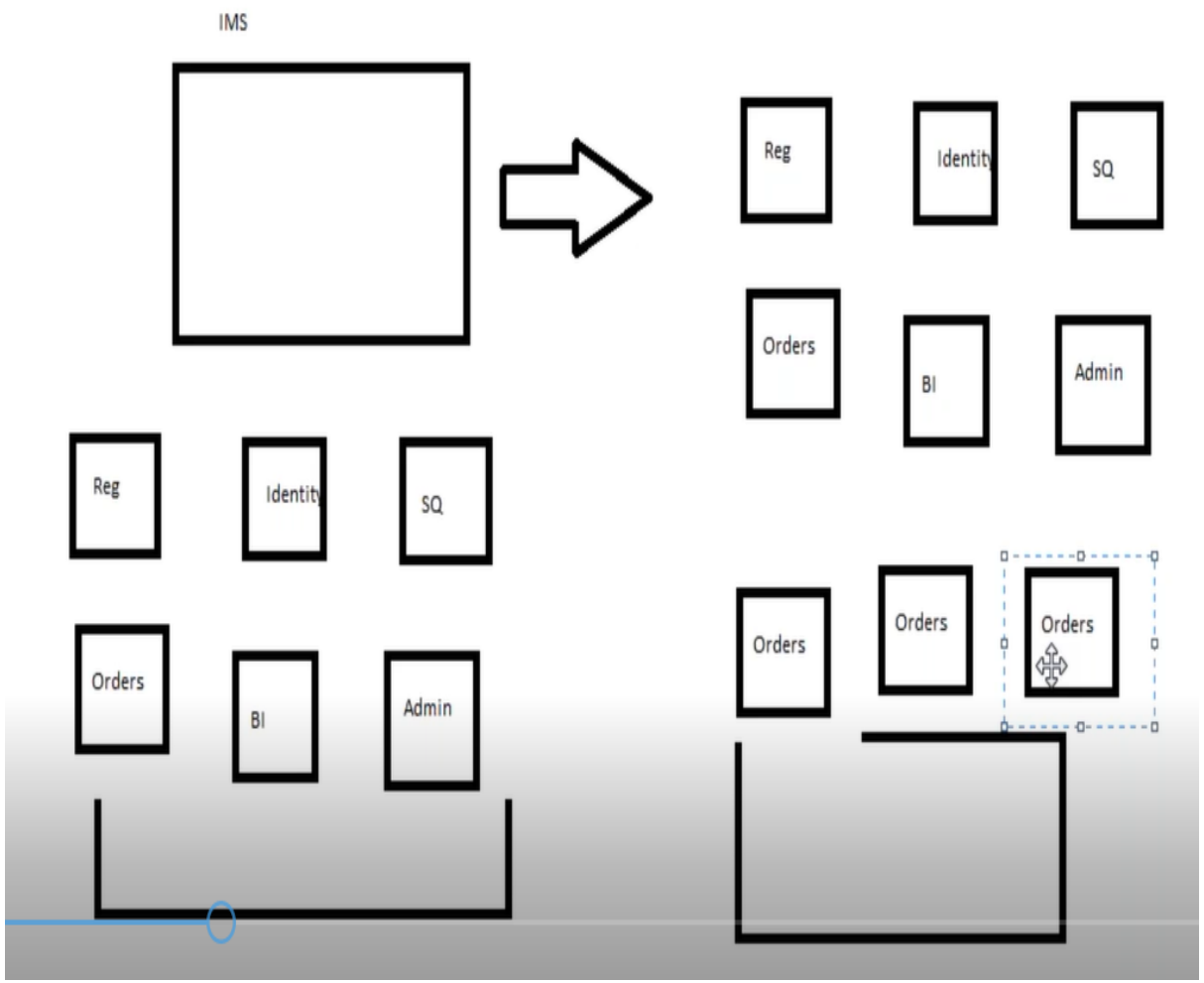


Docker Overview

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.
- Docker provides the ability to package and run an application in a loosely isolated environment called a container.
- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
- Docker provides tooling and a platform to manage the lifecycle of your containers:
- With containers scaling and portability becomes extremely simple.

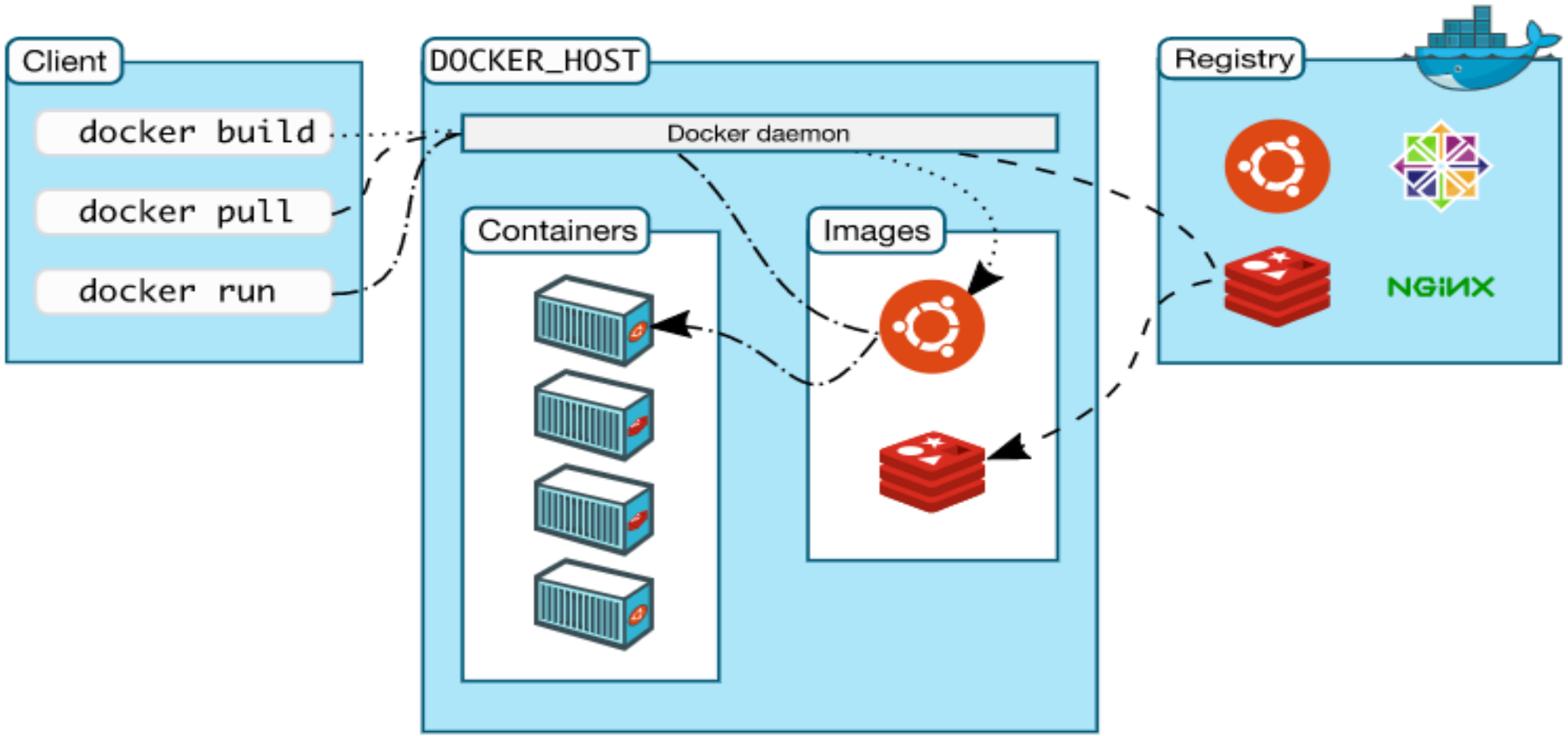
Why do we need to containerize the Application





Docker Architecture

- Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.



Docker Architecture

- A docker image basically consists of several layers.
- If you look at the Dockerfile, each instruction in the Dockerfile corresponds to a layer in the resulting image.
- More instructions you have in the Dockerfile, more layers in the resulting image.

Docker Components

- **The Docker Daemon**

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

- **The Docker Client**

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon

Docker Components

- **Docker Registries**

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker Installation

- Create EC2 Instance with Ubuntu AMI and install Docker
- curl -sSL https://get.docker.com/ | sh
- sudo usermod -aG docker ubuntu
- exit
- login back
- Execute docker info
- Execute docker –version

OR

- sudo apt-get update
- sudo apt-get install docker.io –y
- sudo usermod -aG docker ubuntu
- exit
- login back
- Execute docker info
- Execute docker –version

Docker Hub

- Docker Hub is a Cloud hosted service provided by Docker. Like GitHub repository for Docker images
- Docker Hub is a service provided by Docker for finding and sharing container images with your team. It provides the following major features:
  - Repositories:
  - Push and pull container images. ... Builds: Automatically build container images from GitHub and Bitbucket and push them to Docker Hub.
- Docker images are pushed to Docker Hub through the docker push command. A single Docker Hub repository can hold many Docker images (stored as tags).
- https://hub.docker.com/
- Public Images – Created By Vendors or anybody
- Private images
- Better to use Official public images published by Vendors

Docker Images

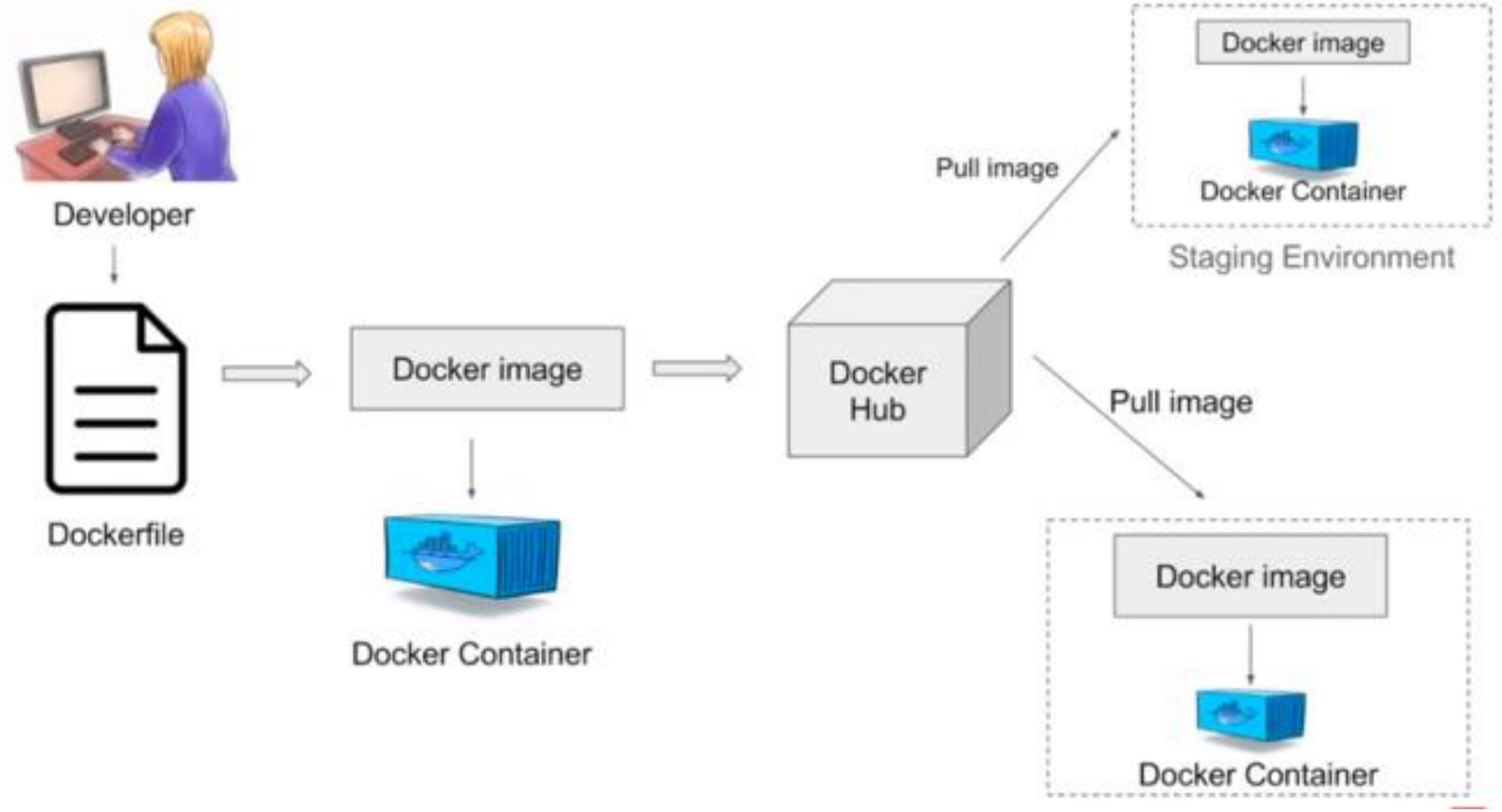
- An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization.
- You might create your own images or you might only use those created by others and published in a registry.
- To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it.
- Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.
- Build by Docker users
- Stored in Docker Hub or Your Local Registry

Docker Container

- A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.
- You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
- By default, a container is relatively well isolated from other containers and its host machine.
- You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.
- A container is defined by its image as well as any configuration options you provide to it when you create or start it.
- When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Docker Workflow

- Docker file builds a Docker image and that image contains all the project's code
- You can run that image to create as many Docker containers as you want
- Then this image can be uploaded on Docker hub, from Docker hub any one can pull the image and build a container



Docker File

- We have seen the various Image files such as Centos which get downloaded from Docker hub from which you can spin up containers.
- If we use the **Docker images** command, we can see the existing images in our system.
- But Docker also gives you the capability to create your own Docker images, and it can be done with the help of Docker Files. A Docker File is a simple text file with instructions on how to build your images.

The following steps explain how you should go about creating a Docker File.

- **Step 1** − Create a file called Docker File and edit it using vim. Please note that the name of the file has to be "Dockerfile" with "D" as capital.

- **Step 2 −** Build your Docker File using the following instructions.

Docker File

```
#This is a sample Image
FROM ubuntu
MAINTAINER demousr@gmail.com

RUN apt-get update
RUN apt-get install –y nginx
CMD [“echo”,”Image created”]
```

- The first line "#This is a sample Image" is a comment. You can add comments to the Docker File with the help of the # command
- The next line has to start with the **FROM** keyword. It tells docker, from which base image you want to base your image from. In our example, we are creating an image from the **ubuntu** image.
- The next command is the person who is going to maintain this image. Here you specify the **MAINTAINER** keyword and just mention the **email ID**.

Docker File

- The **RUN** command is used to run instructions against the image. In our case, we first update our Ubuntu system and then install the nginx server on our ubuntu image.
- The last command is used to display a message to the user.

- **Step 3 −** Save the file.

- The Docker File can be built with the following command −

  docker build -t ImageName:TagName dir

Options

- **-t** − is to mention a tag to the image
- **ImageName** − This is the name you want to give to your image.
- **TagName** − This is the tag you want to give to your image.
- **Dir** − The directory where the Docker File is present.

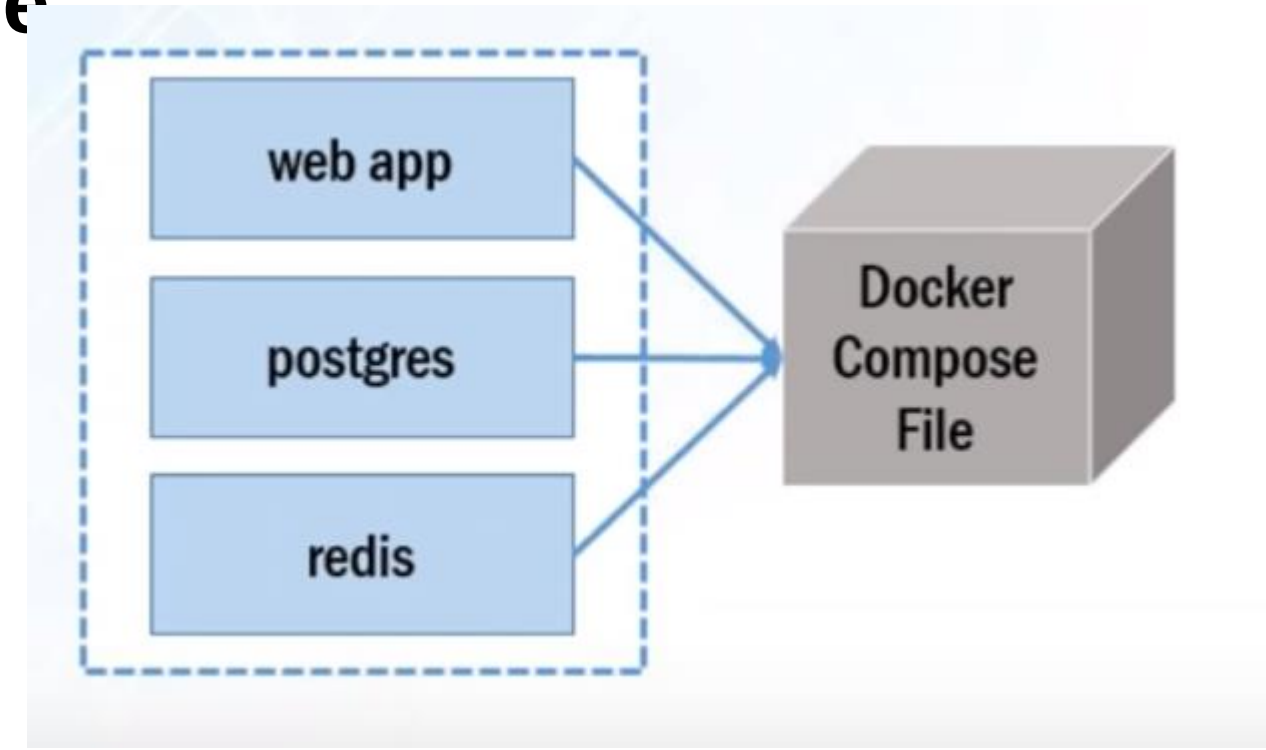
Docker File

- From the output, you will first see that the Ubuntu Image will be downloaded from Docker Hub, because there is no image available locally on the machine.
- Finally, when the build is complete, all the necessary commands would have run on the image.
- You can now build containers from your new Image.

Docker Compose

- Docker Compose is used to run multiple containers as a single service. For example, suppose you had an application which required NGNIX and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately.
- With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.
- For example: imagine being able to define three containers – one running a web app, another running postgres, and third running redis – all in one YAML file and then running those three connected containers with a single command.

Docker Compose



- Docker Compose Installation

  sudo apt install docker-compose

  docker-compose --version

Docker Compose

- Using Compose is basically a three-step process:
  - Define your app's environment with a Dockerfile so it can be reproduced anywhere.
  - Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
  - Run **docker-compose up -d** and Compose starts and runs your entire app.
  - Can stop all services with a single command (**docker-compose down**)
  - Can scale up selected services when required

  docker-compose up -d --scale database=4

  - If the file name is other than docker-compose.yaml

  docker-compose -f docker-compose1.yaml up -d

Q & A