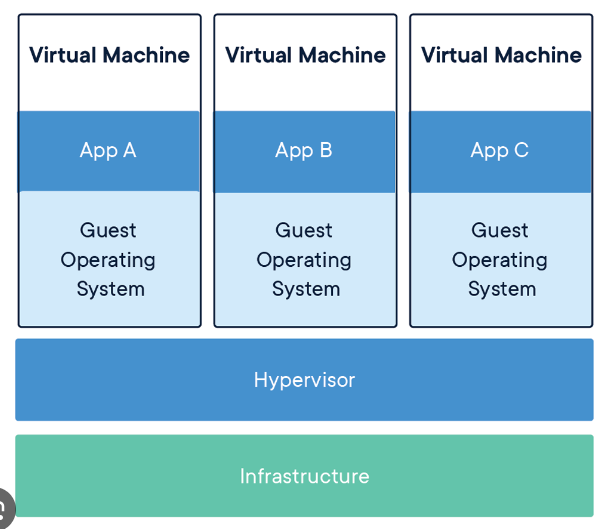
**Docker:**

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. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

Containerization is a technology that is enjoying huge popularity in the tech world – and Docker is a renowned player of it. You need to know that numerous IT giants are providing various enriching career opportunities to Docker professionals. The major reason behind such an immensely growing demand for [Docker](https://www.docker.com/)is that it actually resolves the cult problem of every development team – “**It works on my machine…!!”**.

Let us understand it with a basic example.

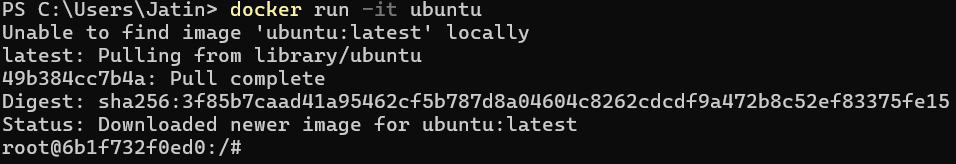
Suppose there are four developers in a team working on a single project. Meanwhile, one is having a Windows system, the second is owning a Linux system, and the third & fourth ones are working with macOS. Now, as you see, they are using the distinct environments for creating a single application or software they will be required to carry on the things in accordance with their respective machines such as the installation of different libraries & files for their system, etc. And such situations, especially on an organizational or larger level, often cause numerous conflicts and problems throughout the entire software development life cycle. However, the containerization tools such as Docker eliminates this problem.



**Commands:**

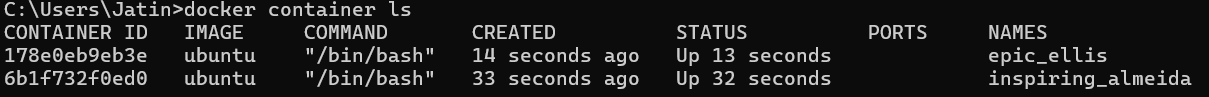
**docker run -it ubuntu**

this is use to run container with ubuntu image



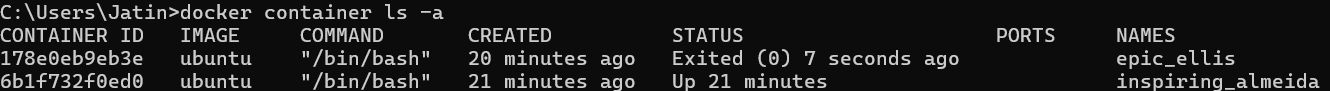
**docker container ls**

used to show all running containers



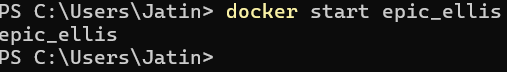
**docker container ls -a**

used to show all containers which is running or not



**docker start epic\_ellis**

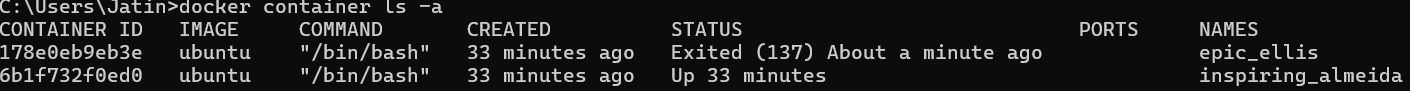
this will start container whose name is “epic\_ellis”.



**docker stop epic\_ellis**

this will stop container whose name is “epic\_ellis”.





**PortMapping:**

In docker, all the application in the container runs on particular ports when you run a container. If you want to access the particular application with the help of a port number you need to map the port number of the container with the port number of the docker host.

## Why We Use Port Mapping?

The[network services](https://www.geeksforgeeks.org/basics-of-docker-networking/)of a container are made accessible to the host or external network through port mapping in Docker. [Docker containers](https://www.geeksforgeeks.org/containerization-using-docker/)can only communicate with other containers on the same Docker network and run by default in isolation from the host and external networks. By publishing a [container’s network](https://www.geeksforgeeks.org/basics-of-docker-networking/)service to the host or external network through [port](https://www.geeksforgeeks.org/docker-managing-ports/) mapping, you can make it reachable from other networked devices.

When running a Docker container, you can map a port on the container to a port on the host or external network using the **-p or —publish** options. If you use the below command it will publish[**nginx**](https://www.geeksforgeeks.org/what-is-nginx-web-server-and-how-to-install-it/), for instance, would publish**port 80** from the container to**port 8080** on the host or external network.



**docker run -p <HostPort:containerport> imagename:tag**

How to set environment variable in container?

**docker run -e env\_var=value <imageName>**

-e: It is used to set environment variable.

where:

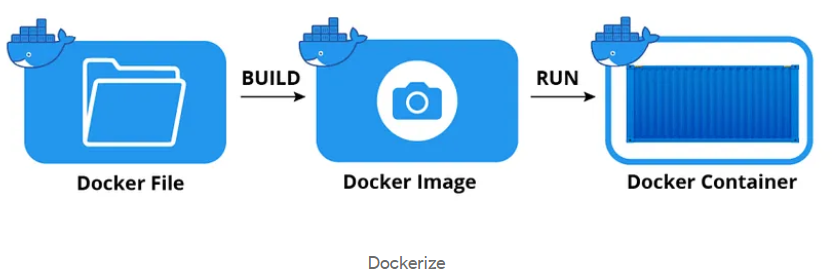
env\_var: Key

value: value

For Multiple variables:

-e k1=v1 k2 =v2 ....

**Dockerization of any application:**



# What Is Dockerize ?

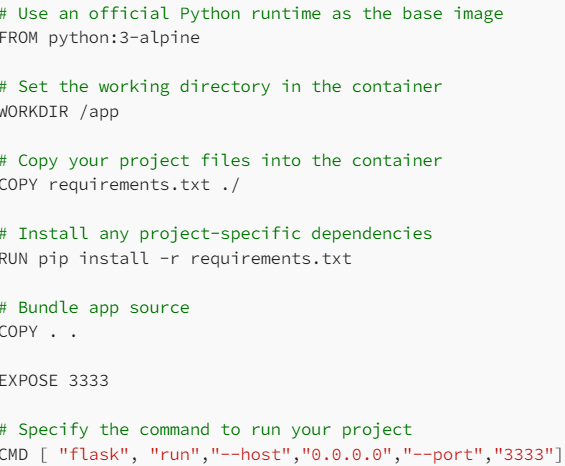
The term “Dockerize” means to adapt or configure an application or service to work within a Docker container. When you “Dockerize” an application, you are essentially preparing it for containerization using Docker, a containerization platform. The process of Dockerizing an application involves creating a Dockerfile, building a Docker image, and running the application as a Docker container.

**Create a Dockerfile**:

In your project’s directory, create a file named Dockerfile (with no file extension). This file will contain instructions for building a Docker image. Customize it to your application's requirements, specifying the base image, copying files, setting up the environment, and defining how your application should run within the container. Dockerfile is a script that provides a set of instructions for building a Docker image. Here are the typical commands you might use in a Dockerfile and what they do:

* **FROM**: This is the first and most crucial command in a Dockerfile. It specifies the base image upon which your Docker image will be built. The base image typically includes an operating system and a runtime environment (e.g., Python, Node.js, Java). We use “Python” in our example.
* **COPY and ADD**: These commands copy files and directories from your local filesystem into the Docker image. They are often used to include your application code, configuration files, and dependencies.
* **RUN**: The RUN command executes commands within the container during the image-building process. It is typically used to install packages, libraries, or dependencies that your application needs.
* **WORKDIR**: It sets the working directory for any subsequent RUN,  CMD,  ENTRYPOINT,  COPY, and ADD instructions. This is where your application will be placed in the image, and it's where the container will start by default.
* **EXPOSE**: It command informs Docker that the container listens on specific network ports. It does not actually publish the port; it is merely documentation for users.
* **CMD and ENTRYPOINT**: These commands specify the command that should be executed when the container is started. You generally use one or the other, not both. CMD provides default arguments for an executing container, while ENTRYPOINT specifies the main command.

Let’s show all the steps in order below:



For each instruction or command from the **Dockerfile**, the Docker builder generates an image layer and stacks it upon the previous ones. Therefore, the Docker image resulting from the process is simply a read-only stack of different layers.

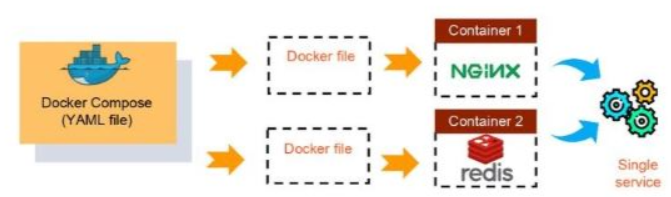
**Docker Compose:**

Suppose you have more than one container in Docker. Let’s say a web server and a database are running in separate containers, which is typical for your data structure. The building, running, and connecting the containers from separate [Docker files](https://www.simplilearn.com/tutorials/docker-tutorial/what-is-dockerfile) is difficult and can take a lot of time. This is where Docker Compose comes in. It facilitates multiple services running simultaneously.

Docker Compose is a tool that assists in defining and sharing multi-container applications. By using Compose, we can define the services in a YAML file, as well as spin them up and tear them down with one single command.

**For example:**

If you have an application that requires an NGINX server and Redis database, you can create a Docker Compose file that can run both the containers as a service without the need to start each one separately.



**Docker Networking**

**Docker Volume: store permanent data**