Docker

- Problems without Before Docker
  - Compatibility/Dependency
  - long Setup time
  - Different Dev/Test/Prod Environments

- With Docker
  - Containerize Application
  - Run each service with its own dependencies in separate containers

-Docker operate at Deployment stage

-Docker makes the process of application deployment very easy and efficient and resolves a lot of issues related to deploying application

-Docker is the world's leading software container platform

-Docker is a tool designed to make it easier to deploy and run applications by using containers

-Containers allow a developer to package up an application with all of the parts it needs such as libraries and other dependencies,and shi it all out as one package

Container

> A way to package application with all the necessary dependencies and configuration
> Portable artifact, easily shared and moved around
> Makes development and deployment more efficient
>Container live in Container Repository like:- Postgres,redis,nodejs,nginx
>Some company have private repository
>public repository for docker is "DockerHub"

-How Container improved...

1) In Application Development
   - Before Container
     >Installation process different
     On each OS environment
     >Many steps where something could go wrong
   - After Container
     >Own isolated environment
     >packaged with all needed configuration
     >one command to install the app
     >run same app with 2 different version
2) In Application Deployment
   - Before Container
     >Configuration on the server needed
     >Dependency version conflicts
     >textual guide of deployment

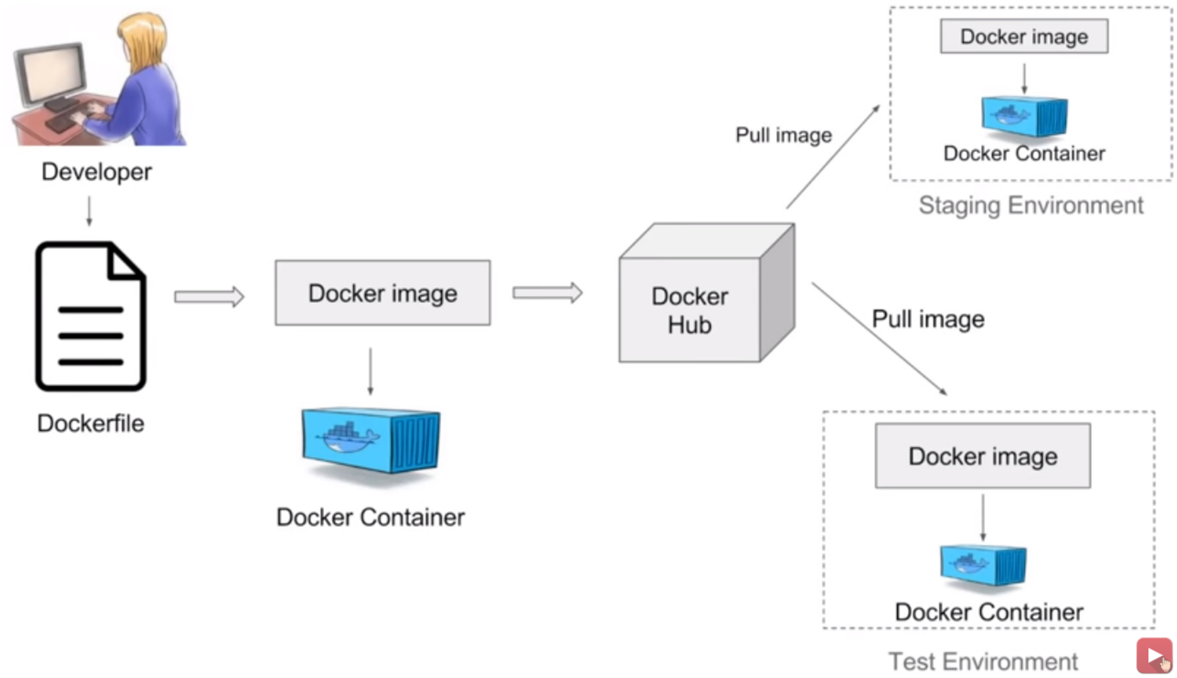
>misunderstanding

-After Container

>Developers and operation work together to package the application in a Container

>No environment configuration needed on server-except Docker runtime

Docker Workflow



Dockerfile-describes steps to create a Docker image.It's like a recipe with all ingredients and steps necessary in making your dish

Docker Image- From docker file we create docker image

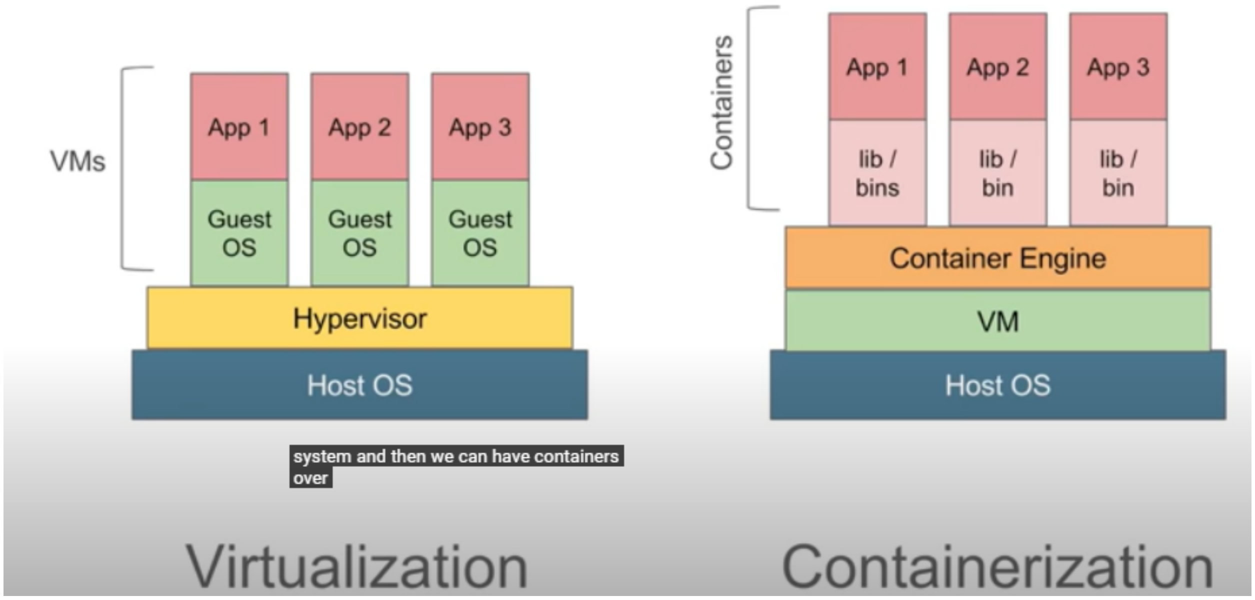
-A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform.

-A Docker image is made up of a collection of files that bundle together all the essentials, such as installations, application code and dependencies, required to configure a fully operational container environment.

Docker Container-Container will have application with all its Dependencies

-Docker is a container platform

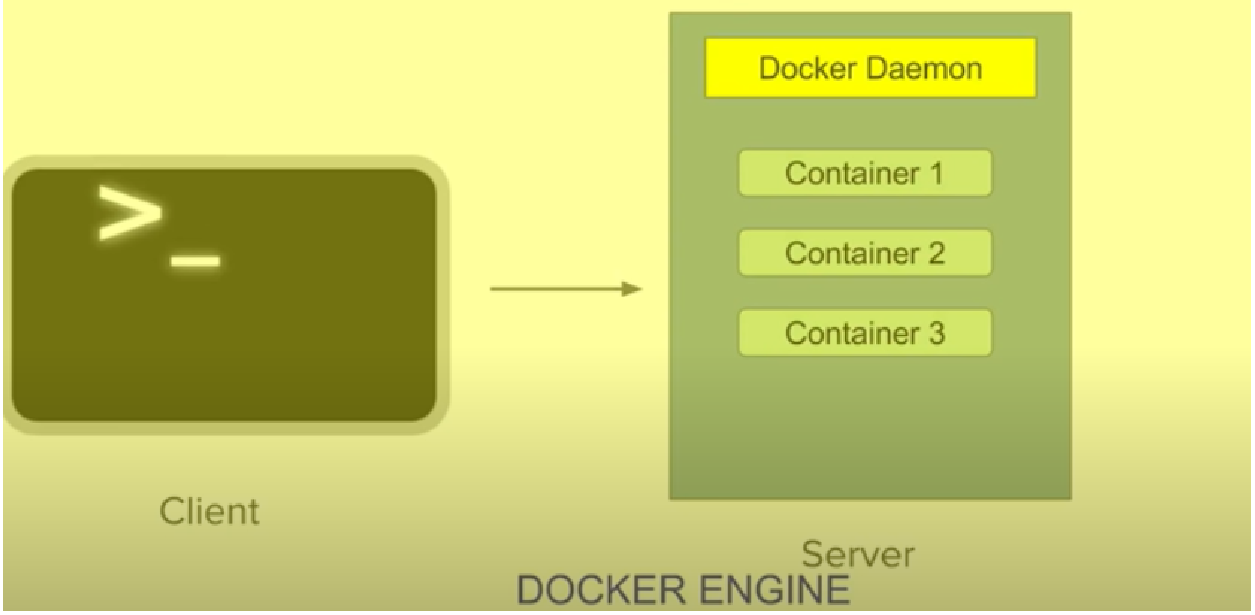
Virtualization vs Containerization



system and then we can have containers over

Container engine is docker engine

-Docker has a client-server architecture



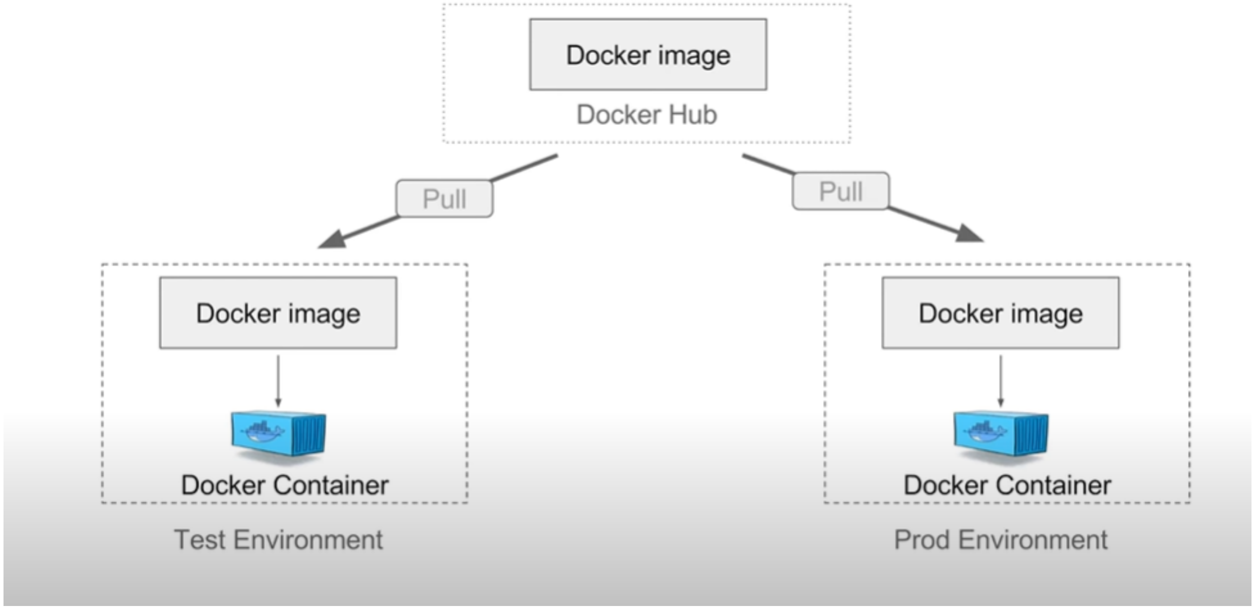
The daemon(Server) receives the commands from the docker client through CLI or REST API’s

Docker client and daemon can be present on the same host(machine) and different hosts

Uses of Docker

1.Build app only once

-An application inside a container can run on any system that has Docker installed. So there is no need to build and configure app multiple times on different platform.



2. More sleep and less worry

-With Docker you test your application inside a container and ship it inside a container. This means the environment in which you test is identical to the one on which the app will run in production.

3.Portability

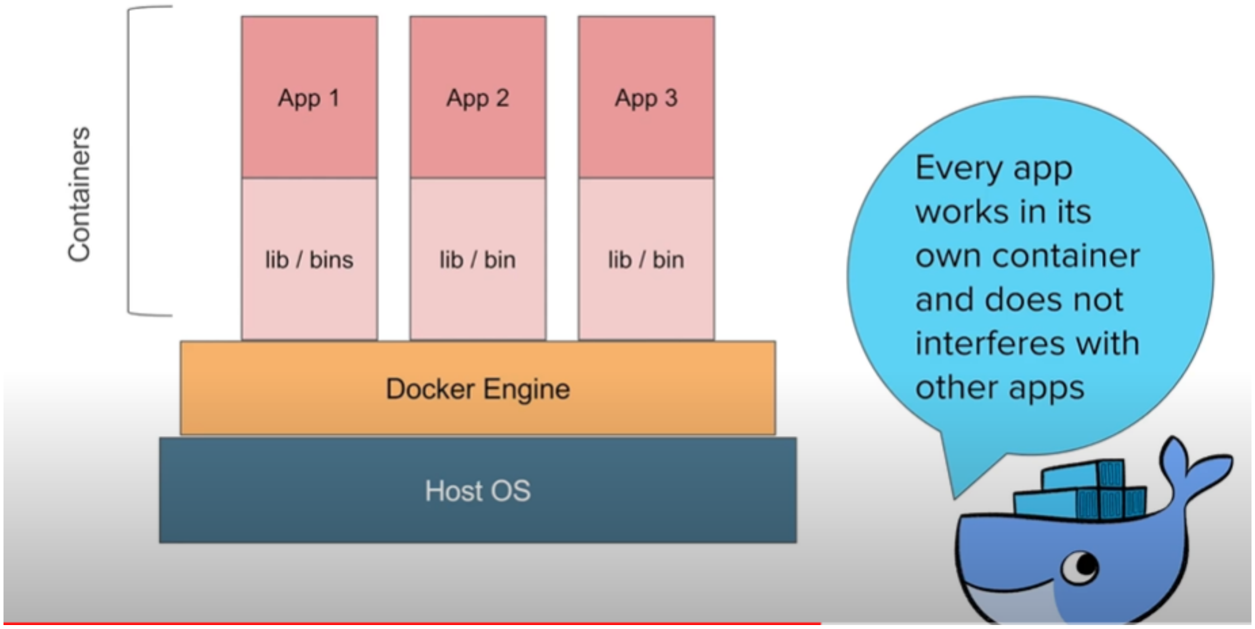
-Docker containers can run on any platform.
It can run on your local system,Amazon ec2,google cloud platform,Rackspace server,VirtualBox..etc.
A container running on AWS can easily be ported to VirtualBox

4.Version Control

-Like Git,Docker has in-built version control system

Docker containers work just like GIT repositories,allowing you to commit changes to your Docker images and version Control them

5.Isolation


Containers
App 1
App 2
App 3
lib / bins
lib / bin
lib / bin
Docker Engine
Host OS
Every app works in its own container and does not interferes with other apps

-With Docker every application works in isolation in its own container and does not interfere with other applications running on the same system.
So multiple containers can run on the same system without interface.

-For removal also you can simply delete the container and it will not leave behind any files or traces on the system

6.Productivity

-Docker allows faster and more efficient deployments without worrying about running your app on different platforms.
-it increases productivity many folds

Getting Started

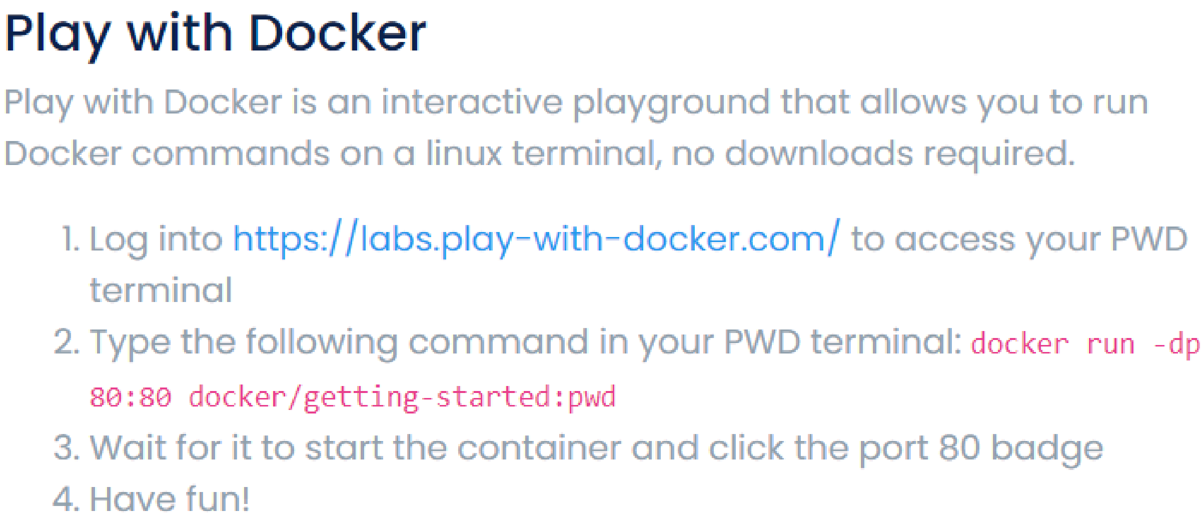
Steps:-

1.Go to https://www.docker.com/
2.Click on Get Started

3.Scroll down click on Play with Docker

4.Scroll down and click on Docker 101 for developers

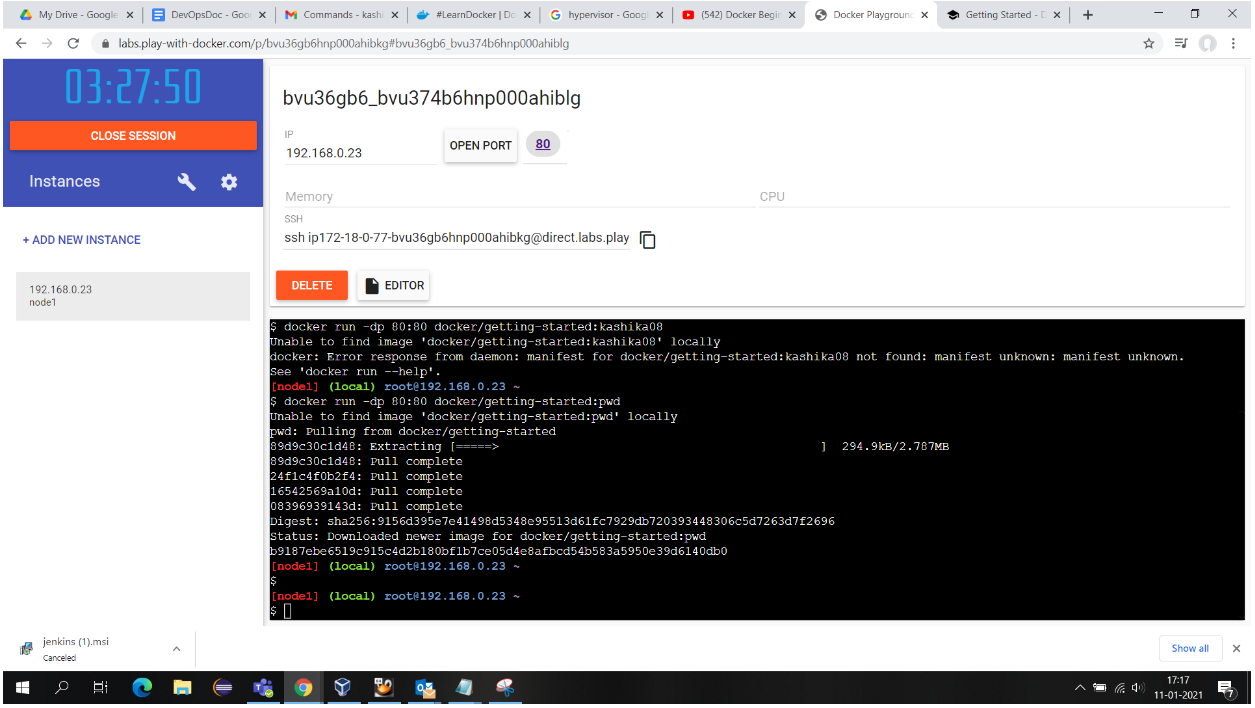
5.go to play with docker and follow the steps written in it
 Sign In if not already



Play with Docker

Play with Docker is an interactive playground that allows you to run Docker commands on a linux terminal, no downloads required.

1. Log into https://labs.play-with-docker.com/ to access your PWD terminal
2. Type the following command in your PWD terminal: `docker run -dp 80:80 docker/getting-started:pwd`
3. Wait for it to start the container and click the port 80 badge
4. Have fun!



labs.play-with-docker.com/p/bvu36gb6hnp000ahibkg#bvu36gb6_bvu374b6hnp000ahiblg

192.168.0.23
node1

bvu36gb6_bvu374b6hnp000ahiblg

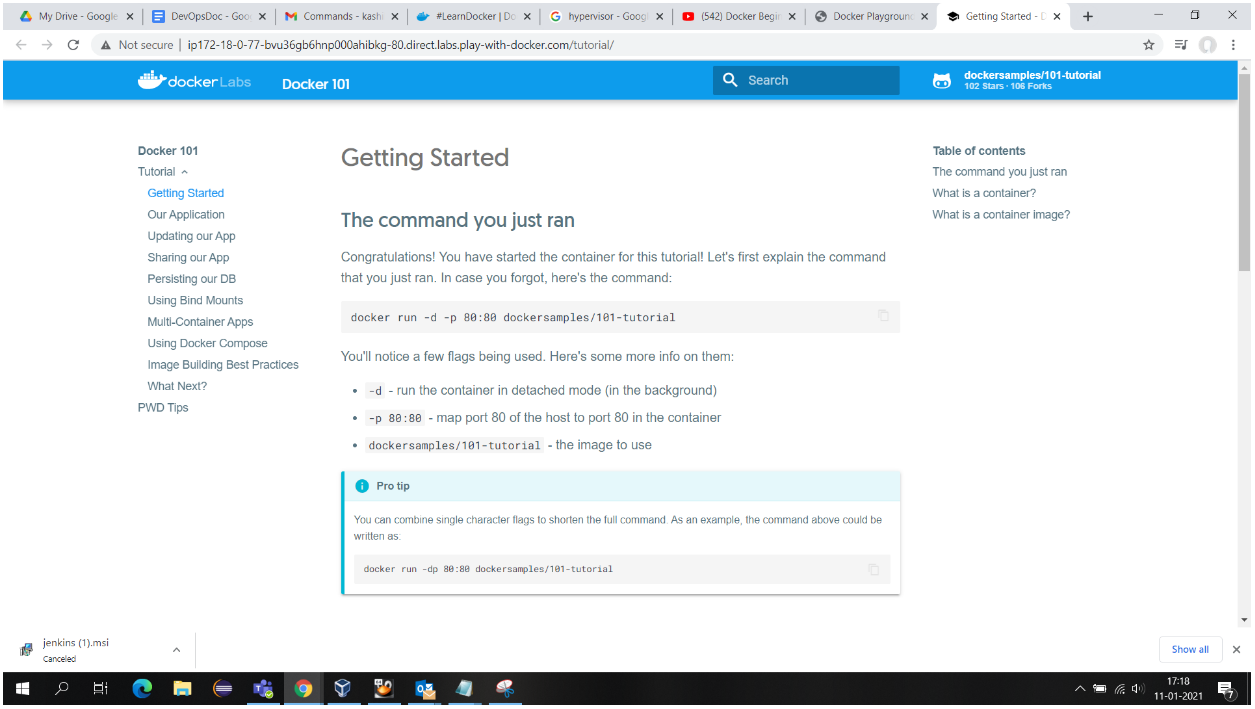
IP: 192.168.0.23

Memory

CPU

SSH: ssh ip172-18-0-77-bvu36gb6hnp000ahibkg@direct.labs.play

```
$ docker run -dp 80:80 docker/getting-started:kashika08
Unable to find image 'docker/getting-started:kashika08' locally
docker: Error response from daemon: manifest for docker/getting-started:kashika08 not found: manifest unknown: manifest unknown.
See 'docker run --help'.
[node1] (local) root@192.168.0.23 ~
$ docker run -dp 80:80 docker/getting-started:pwd
Unable to find image 'docker/getting-started:pwd' locally
pwd: Pulling from docker/getting-started
89d9c30c1d48: Extracting [=====>                                             ]  294.9kB/2.787MB
89d9c30c1d48: Pull complete
24f1c4f0b2f4: Pull complete
16542569a10d: Pull complete
08396939143d: Pull complete
Digest: sha256:9156d395e7e41498d5348e95513d61fc7929db720393448306c5d7263d7f2696
Status: Downloaded newer image for docker/getting-started:pwd
b9187ebe6519c915c4d2b180bf1b7ce05d4e8afbcd54b583a5950e39d6140db0
[node1] (local) root@192.168.0.23 ~
$
[node1] (local) root@192.168.0.23 ~
$ 
```



Getting Started

The command you just ran

Congratulations! You have started the container for this tutorial! Let's first explain the command that you just ran. In case you forgot, here's the command:

```
docker run -d -p 80:80 dockersamples/101-tutorial
```

You'll notice a few flags being used. Here's some more info on them:

- `-d` - run the container in detached mode (in the background)
- `-p 80:80` - map port 80 of the host to port 80 in the container
- `dockersamples/101-tutorial` - the image to use

Pro tip

You can combine single character flags to shorten the full command. As an example, the command above could be written as:

```
docker run -dp 80:80 dockersamples/101-tutorial
```

Command explanation

```
docker run -d -p 80:80 dockersamples/101-tutorial
```

You'll notice a few flags being used. Here's some more info on them:

- `-d` - run the container in detached mode (in the background)
- `-p 80:80` - map port 80 of the host to port 80 in the container
- `dockersamples/101-tutorial` - the image to use

For get the application source code into the Play with Docker environment.
>For real projects, you can clone the repo. But, in this case, you will upload a ZIP archive.

1. Download the zip and upload it to Play with Docker. As a tip, you can drag and drop the zip (or any other file) on to the terminal in PWD.
2. In the PWD terminal, extract the zip file.

```
unzip app.zip
```

3.Change your current working directory into the new 'app' folder.

```
cd app/
```

4.In this directory, you should see a simple Node-based application.

```
ls
package.json  spec          src           yarn.lock
```

Building the App's Container Image

In order to build the application, we need to use a `Dockerfile`. A Dockerfile is simply a text-based script of instructions that is used to create a container image.

1. Create a file named Dockerfile with the following contents.

```
FROM node:10-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "/app/src/index.js"]
(make a file of name dockerfile and just drag and drop into pwd terminal)
```

Each instruction creates one layer:

1) The `FROM` instruction sets the container image that will be used during the new image creation process.

   FROM <image>

2) The `WORKDIR` instruction sets a working directory for other Dockerfile instructions, such as `RUN`, `CMD`, and also the working directory for running instances of the container image.
3) `COPY` adds files from your Docker client's current directory.

   The `COPY` instruction copies files and directories to the container's file system. The files and directories must be in a path relative to the Dockerfile.

   The `COPY` instruction's format goes like this:

   Dockerfile

Copy

```
COPY <source> <destination>
```

4) The RUN instruction specifies commands to be run, and captured into the new container image. These commands can include items such as installing software, creating files and directories, and creating environment configuration.

   The RUN instruction goes like this:

   Dockerfile

   Copy

```
 # exec form
RUN ["<executable>", "<param 1>", "<param 2>"]
# shell form
RUN <command>
```

5) CMD specifies what command to run within the container.

   The CMD instruction sets the default command to be run when deploying an instance of the container image.

For more information visit https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-docker/manage-windows-dockerfile#:~:text=A%20Dockerfile%20must%20be%20created,%22%20(including%20the%20quotes).&text=For%20additional%20examples%20of%20Dockerfiles,the%20Dockerfile%20for%20Windows%20repository.

2. Build the container image using the `docker build` command.

```
docker build -t docker-101 .
```

This command used the Dockerfile to build a new container image. You might have noticed that a lot of "layers" were downloaded. This is because we instructed the builder that we wanted to start from the `node:10-alpine` image. But, since we didn't have that on our machine, that image needed to be downloaded.
After that, we copied in our application and used `yarn` to install our application's dependencies. The `CMD` directive specifies the default command to run when starting a container from this image.

Starting an App Container

Now that we have an image, let's run the application! To do so, we will use the `docker run` command (remember that from earlier?).

1. Start your container using the `docker run` command:
2. `docker run -dp 3000:3000 docker-101`

Updating Our App

Steps:-

1. Through editor go to file in which you want to change
2. build our updated version of the image, using the same command we used before.

   ```
   docker build -t docker-101 .
   ```

3. Now for running we will use the run command but We aren't able to start the new container because our old container is still running. The reason this is a problem is because that container is using the host's port 3000 and only one process (containers included) can listen to a specific port. To fix this, we need to remove the old container.

Replacing our Old Container

To remove a container, it first needs to be stopped. Then, it can be removed.

1. Get the ID of the container by using the `docker ps` command.

   ```
   docker ps
   ```

2. Use the docker stop command to stop the container.

```
# Swap out <the-container-id> with the ID from docker ps

docker stop <the-container-id>
```

3. Once the container has stopped, you can remove it by using the `docker rm` command.
   ```
   docker rm <the-container-id>
   ```

4. Now, start your updated app.
   ```
   docker run -dp 3000:3000 docker-101
   ```

5. Open the app and you should see your updated help text!

Sharing Our App

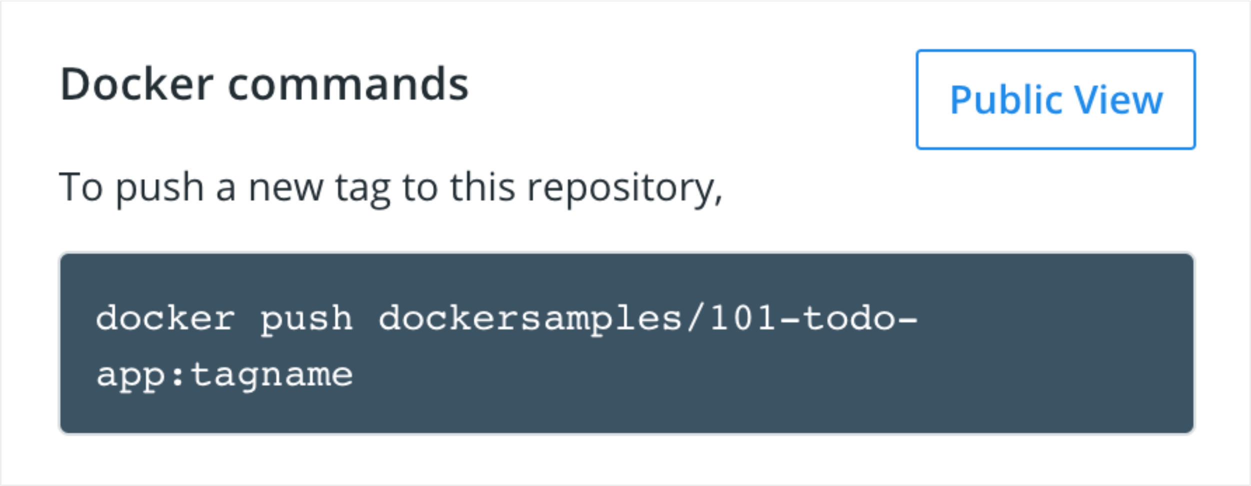
Now that we've built an image, let's share it! To share Docker images, you have to use a Docker registry. The default registry is Docker Hub and is where all of the images we've used have come from.

Create a Repo

To push an image, we first need to create a repo on Docker Hub.

1. Go to Docker Hub and log in if you need to.
2. Click the **Create Repository** button.
3. For the repo name, use `101-todo-app`. Make sure the Visibility is `Public`.
4. Click the **Create** button!

If you look on the right-side of the page, you'll see a section named **Docker commands**. This gives an example command that you will need to run to push to this repo.



Docker commands

To push a new tag to this repository,

```
docker push dockersamples/101-todo-app:tagname
```

Pushing our Image

1. Back in your PWD instance, try running the command. You should get an error that looks something like this:

```
$ docker push dockersamples/101-todo-app
The push refers to repository [docker.io/dockersamples/101-todo-app]
An image does not exist locally with the tag:
dockersamples/101-todo-app
```

Why did it fail? The push command was looking for an image named dockersamples/101-todo-app, but didn't find one. If you run `docker image ls`, you won't see one either.
To fix this, we need to "tag" our image, which basically means give it another name.

2. Login to the Docker Hub using the command

```
docker login -u YOUR-USER-NAME.
```

3. Use the `docker tag` command to give the `docker-101` image a new name. Be sure to swap out `YOUR-USER-NAME` with your Docker ID.

```
docker tag docker-101 YOUR-USER-NAME/101-todo-app
```

4. Now try your push command again. If you're copying the value from Docker Hub, you can drop the `tagname` portion, as we didn't add a tag to the image name.

```
docker push YOUR-USER-NAME/101-todo-app
```

Running our Image on a New Instance

Now that our image has been built and pushed into a registry, let's try running our app on a brand instance that has never seen this container!

Back in PWD, click on **Add New Instance** to create a new instance.

1. In the new instance, start your freshly pushed app.

```
docker run -dp 3000:3000 YOUR-USER-NAME/101-todo-app
```

2. You should see the image get pulled down and eventually start up!

3. Click on the 3000 badge when it comes up and you should see the app with your modifications! Hooray!

Docker Commands

1. Run-start a container

   docker run <Image name>

   Example:- docker run nginx

   -This command will run an instance of the nginx application from the docker Host if it already exists

   - if the image is not present on the host it will go out to docker hub and pull the image down. But this only done for the first time for subsequent Execution the same image will be reused

2. ps-List Containers

   docker ps

   -ps command list all running containers and some basic information about them Such as container id, the name of the image we used to run the containers ,the current status and the name of the container

   -each container automatically gets random ID and name created for it by docker

   -for checking all containers are running or not use

   docker ps -a

   -to stop a running container use the command

   docker stop <container name/Id>

3. Rm-Remove a Container

   docker rm <Container name/Id>

4. images-List images

   docker images

   -use this command to see list of available images and Size

   -for remove an image use commands

   Docker rmi <image name>

   For deleting an image first ensure you deleted or stop and remove all container of that image

5. Pull-download an image

   docker pull <image name>

-its only pull the image not run the container

-if the image inside the container have no process then container exit

Example

```
docker run ubuntu
```

When you will type docker ps command you won’t get any running container

But when you type docker ps -a you will see the ubuntu image container in exit position

Why?

Because ubuntu have no process running into it so Container exit immediately

-using run command we can assign process to container

Example:- docker run ubuntu sleep 5

6. Exec-execute a command

   ```
   docker exec <name of container> cat /etc/hosts
   ```

   -when we want to run a process on a running container we use this command

7. Run - attach and detach

   -for running a container in background(detach)

   ```
   Docker run -d <name of file>
   ```

   -for attach to the back running container use

   ```
   docker attach <ContainerId>
   ```

8. Run-STDIN

   For giving input from command prompt into docker

   ```
   Docker run -i nameoffile
   ```

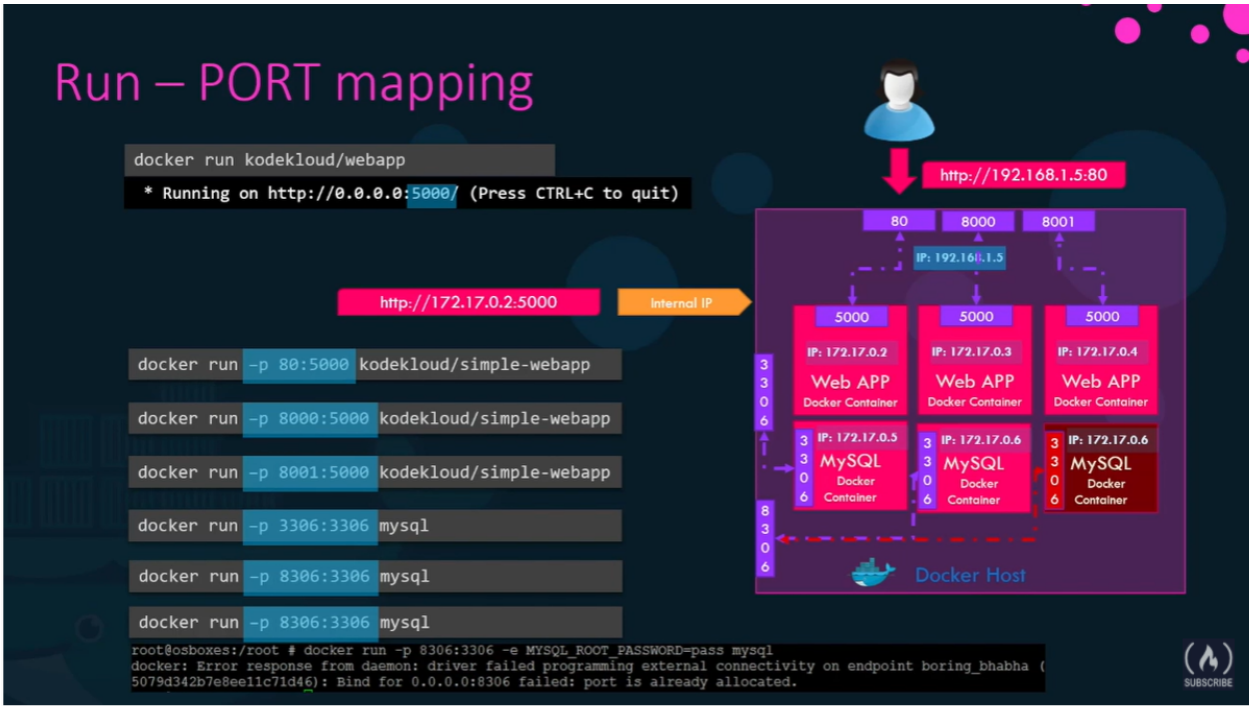
   Example:-

   ```
   ~/prompt-application$./app.sh
   Docker run kodecloud/simple-prompt-docker
   Docker run -i kodekloud/simple-prompt-docker
   Docker run -it kodekloud/simple-prompt-docker
   ```

   t=terminal

   i=input

9. Run-port mapping



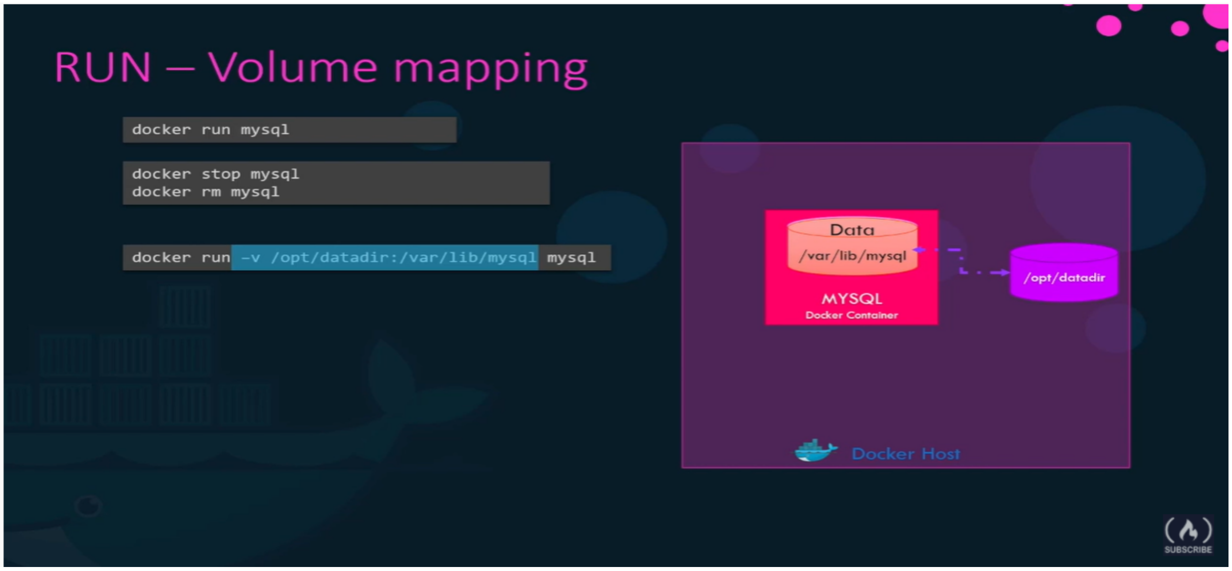
Run – PORT mapping

```
docker run kodekloud/webapp
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

```
docker run -p 80:5000 kodekloud/simple-webapp
docker run -p 8000:5000 kodekloud/simple-webapp
docker run -p 8001:5000 kodekloud/simple-webapp
docker run -p 3306:3306 mysql
docker run -p 8306:3306 mysql
docker run -p 8306:3306 mysql
```

```
root@osboxes:/root # docker run -p 8306:3306 -e MYSQL_ROOT_PASSWORD=pass mysql
docker: Error response from daemon: driver failed programming external connectivity on endpoint boring_bhabha (
5079d342b7e8ee11c71d46): Bind for 0.0.0.0:8306 failed: port is already allocated.
```

10. Run -volume mapping



RUN – Volume mapping

```
docker run mysql
```

```
docker stop mysql
docker rm mysql
```

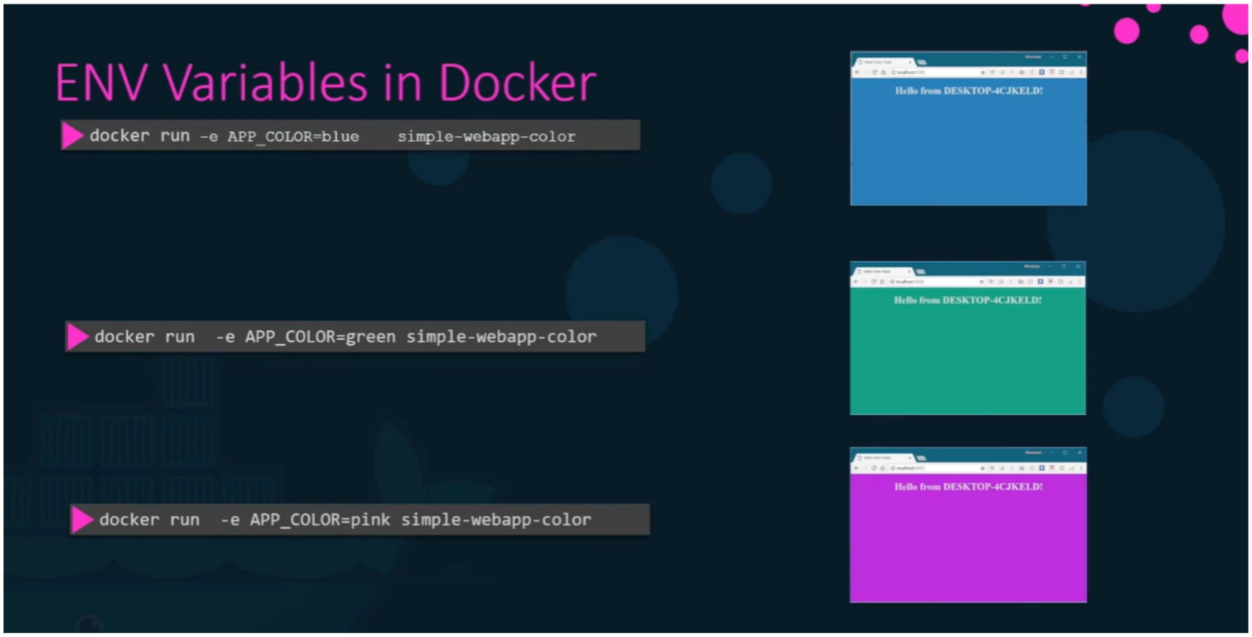
```
docker run -v /opt/datadir:/var/lib/mysql mysql
```

11.Inspect container
  -use when need to details about container
  -docker inspect <docker name>

12. Container Logs
  -for view the logs which happens to be the contents written to the standard out of the container use the docker logs command and specify the container id or name

Environment variable in docker

-for giving a value to environment variable of docker image



ENV Variables in Docker

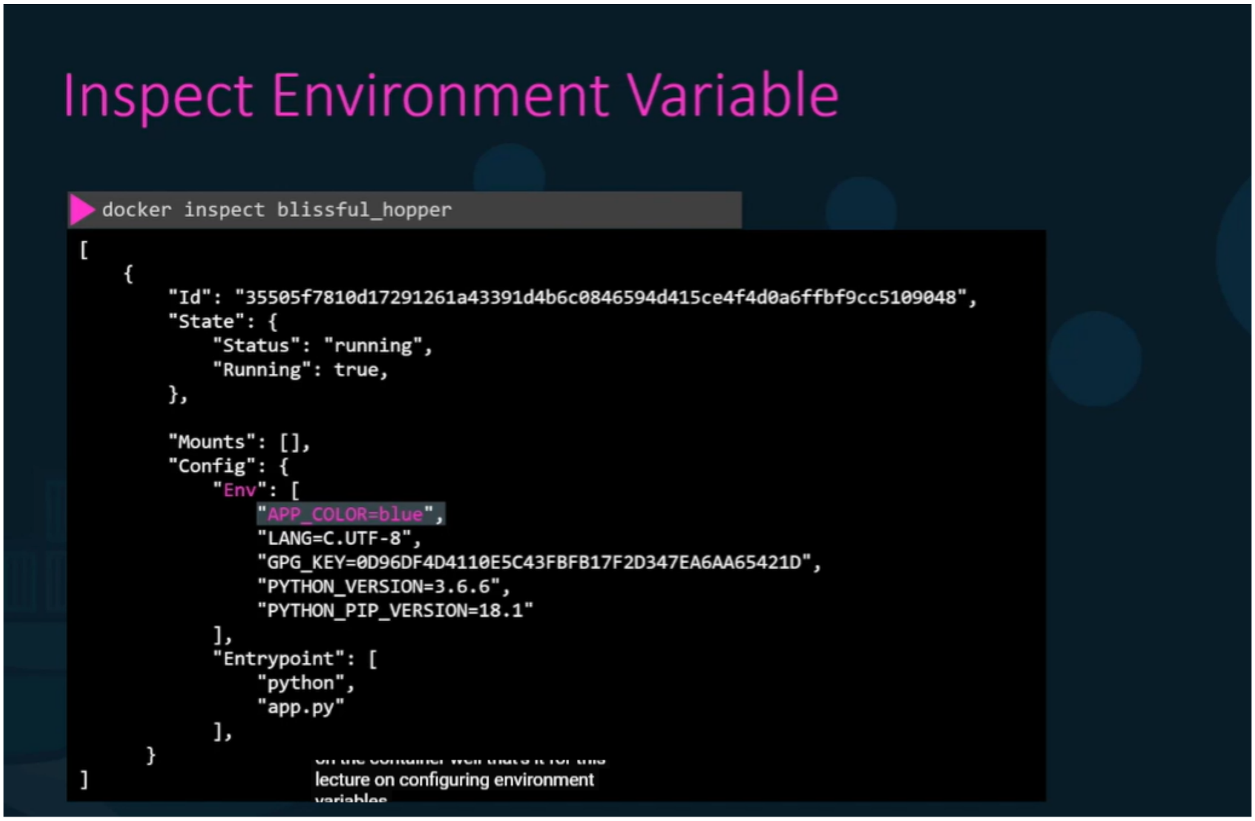
```
docker run -e APP_COLOR=blue    simple-webapp-color
```

```
docker run  -e APP_COLOR=green simple-webapp-color
```

```
docker run  -e APP_COLOR=pink simple-webapp-color
```

Hello from DESKTOP-4CJKELD!
Hello from DESKTOP-4CJKELD!
Hello from DESKTOP-4CJKELD!

`-for inspect the all env variable in image`



Inspect Environment Variable

```
docker inspect blissful_hopper
```

```
[
    {
        "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",
        "State": {
            "Status": "running",
            "Running": true,
        },

        "Mounts": [],
        "Config": {
            "Env": [
                "APP_COLOR=blue",
                "LANG=C.UTF-8",
                "GPG_KEY=0D96DF4D4110E5C43FBFB17F2D347EA6AA65421D",
                "PYTHON_VERSION=3.6.6",
                "PYTHON_PIP_VERSION=18.1"
            ],
            "Entrypoint": [
                "python",
                "app.py"
            ],
      }
]
```

on the container well that's it for this
lecture on configuring environment
variables