

login your machine

Step 1: sudo apt update

Step 2:

```
sudo apt-get install \
apt-transport-https \
ca-certificates curl \
software-properties-common
```

Step 3: `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`

Step 4: `sudo apt-key fingerprint 0EBFCD88`

Step 5: `sudo add-apt-repository \`

`"deb [arch=amd64] https://download.docker.com/linux/ubuntu \`

`$(lsb_release -cs) \`

`stable"`

Step 6: `sudo apt-get update`

Step 7: `sudo apt-get install docker-ce`

Step 8: `sudo docker -v`

Docker is installed on your machine

Some commands of Docker

1. `sudo docker pull Ubuntu` : To pull image from docker hub. It will download image from docker hub
2. `sudo docker images` : To see images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	4e2eef94cd6b	2 weeks ago	73.9MB

1.8 GB Actual Size of Ubuntu

3. `sudo docker run -it -d Ubuntu` : **This command will create container**

It will run the container daemon , in the background till we stop the container

4. `sudo docker ps` : Check running containers

5. `sudo docker stop cid` : To Stop Container
6. `sudo docker ps -a` : To see all containers, stopped or running
7. `sudo docker exec -it 5191f785dd51 bash` : Get inside the container

```
root@5191f785dd51:/# ls
```

```
bin dev home lib32 libx32 mnt proc run srv tmp var
```

```
boot etc lib lib64 media opt root sbin sys usr
```

8. `root@5191f785dd51:/# apt-get update`

```
root@5191f785dd51:/# docker
```

```
bash: docker: command not found
```

9. `root@5191f785dd51:/# exit` : To Exit out of the container

We are creating a folder inside the container

```
root@5191f785dd51:/# mkdir app
```

```
root@5191f785dd51:/# exit
```

10. `sudo docker commit 5191f785dd51 new` : Update & Commit this image

We will get this customized image now

```
anamika@UbuntuServer:~$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
new	latest	1d08d6baac2f	58 seconds ago	96.6MB
ubuntu	latest	4e2eef94cd6b	2 weeks ago	73.9MB

11. `sudo docker rm -f $(sudo docker ps -a -q)` : To delete all containers together

```
apt-get install apache2
```

```
service apache2 status
```

```
service apache2 start
```

```
service apache2 status
```

```
sudo docker commit container id anamikasawhney/apache2
```

```
apThis is a new image
```

12. `sudo docker login` : Login to docker hub

13. `sudo docker push anamikasawhney/apache` : Push this image to dockerhub

DockerFile

It's a text file / or a script file that contains all the commands a user could call on the command line to assemble an image. Using docker build command users can create an automated build that executes several command-line instructions in succession

You want your container to be customized.

1. FROM keyword is used to define the base image, on which we are building

FROM ubuntu

2. ADD is used to add files inside the container being built.

ADD <source> <destination>

ADD . /var/www/html

3. RUN is used to add layers to the base image, by installing components

RUN apt-get update

RUN apt-get install -y apache2

4. CMD : to run any commands on the start of the container

CMD apachectl -D FOREGROUND

5. ENTRYPOINT : is used to strictly run the commands the moment the container initializes. The difference between CMD and ENTRYPOINT is that ENTRYPOINT will run irrespective of the fact whether argument is specified or not

ENTRYPOINT apachectl -D FOREGROUND

6. ENV : used to define environmental variables in the container run-time

ENV name DevOps

Create this file

Create html page

FROM ubuntu

RUN apt-get update

RUN apt-get -y install apache2

```
ADD . /var/www/html

ENTRYPOINT apachectl -D FOREGROUND

ENV name Devops

-----

FROM ubuntu

ENV TZ=Europe/Kiev

RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

RUN apt-get update

RUN apt-get -y install apache2

ADD . /var/www/html

ENTRYPOINT apachectl -D FOREGROUND

ENV name Devops
```

For DotNet Program

```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build-env
WORKDIR /app
#Copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore
#Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out
#
#Build runtime image
FROM mcr.microsoft.com/dotnet/core/aspnet:3.1
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

- dotnet/sdk

The sample uses this image for building the app. The image contains the .NET SDK, which includes the Command Line Tools (CLI). The image is optimized for local development, debugging, and unit testing. The tools installed for development and compilation make the image relatively large.

dotnet/aspnet

The sample uses this image for running the app. The image contains the ASP.NET Core runtime and libraries and is optimized for running apps in production.

The **WORKDIR** command **is** used to **define** the **working directory** of a **Docker** container at any given time. The command **is** specified in the **Dockerfile**. Any **RUN** , **CMD** , **ADD** , **COPY** , or **ENTRYPOINT** command will be executed in the specified **working directory**.

We can check it

```
sudo docker exec -it e3a5b3c270b3 bash
```

where this id is container id

The **dotnet restore** command uses NuGet to **restore** dependencies as well as project-specific tools that are specified in the project file

Commands after Dockerfile is created :

```
docker build . -t new_dockerfile
```

```
docker images
```

```
docker run -it -p 84:80 -d new_dockerfile
```

```
docker ps
```

check in browser

ip/1.html

Go inside container

Docker exec –it contid

ls

It will show 1.html Dockerfile

If you do not want to have Dockerfile inside this

Cd /var/www/html

ls

You should see this html page

./1.html will not copy Dockerfile in the container

Echo \$NAME

Types of Docker Storage

By default, all the data of a container is stored on a writable container layer.

Data only exists while container is active. If the container no longer exists, the data is also deleted along with the container. The Writable container layer is tightly coupled with the host machine, hence not portable.

The data on the writable layer in the container is written using a storage driver.

For data to persist, even after end of life cycle of container, we should use Docker storage

When we delete container, nothing is stored on the host machine, without affecting anything on host machine. We should make containers stateless , But in case if your application has to store data , we can do that too

To persist data inside the container , even after it is deleted , we have two options

Docker Volumes

Bind Mounts

Docker volume is a virtual entity or virtual software which mimics like a virtual hard disk

So, it's a physical hardware component attached to your machine We don't add a hardware , but it creates a volume which can be attached and detached from the container

Go to your Ubuntu host machine

```
docker volume create demo-vol
```

Volume is created

Now attach it to a container

```
docker run -it --mount source=demo-vol ,destination=/app -d Ubuntu
```

which folder in your container u wanted it to be mounted on.

Remove all the containers

Check all volumes

```
sudo docker ls
```

```
docker run -it --mount source=demo-vol ,destination=/app -d Ubuntu
```

Container is created and running

```
docker ps
```

```
docker exec -it containerid
```

```
ls
```

You will see app folder here

```
cd app
```

```
echo "Hello" > hello.txt
```

```
cat hello.txt
```

This file is stored in app folder

Exit

Remove this container now

```
Docker rm -f containerid
```

```
Docker run -it --mount source=demo-vol , destination=/app -d Ubuntu
```

Go inside this container and you will see the app folder

YOU CAN ATTACH THIS VOLUME WITH ANY CONTAINER THAT WE ARE CREATING

LINKING DOCKER CONTAINERS

You do not create network or use docker volume , but you use Docker linking

With linking, containers communicate with each other by using IP addresses.

/etc/hosts file

Docker run -it --link <name of container> -d imagename

Docker run -it --name c1 -d Ubuntu

Docker run -it --name c2 --link c1 -d Ubuntu

It creates 2 containers

Go inside 2nd container

sudo docker exec -it cid bash

Cat /etc/hosts

MICROSERVICES

Opposite of a Microservice is Monolithic application

Is a single-tiered software application in which different components are combined into a single program which resides in a single platform

Like Mail, Payments , Notifications, Customer Service , Passenger Management

Microservices are a software development architectural style that structures an application as a collection of loosely coupled services.

Each component is in a different service now, There is no or less dependency among the components.

To scale up our applications we use microservices

Docker Compose

Helps In deploying multiple containers together

Compose is a tool for defining and running multi-container Docker Applications. With compose, we use a YAML file to configure your application's services. Then with a single command, we create and start all the services from the configuration.

Docker-compose up will start and run your entire app.

For deploying containers using Docker compose, we use YAML files.

YAML is a superset of JSON file.

There are two types of structures in YAML file

Maps : key value pair in YAML file

Name:HR

Manager:Ajay

Lists : A Sequence of objects

-

Args

- arg1

- arg2

Create a Docker compose file

To create two services

Httpd

Nginx

Version: '3'

Services:

Sample1:

Image: httpd

Ports:

- -"80:80"

Sample2

Image: nginx

Sudo docker-compose up -d

Wordpress comprise of two things

First worpress application

Second mysql database on which wordpress will be storing all information

Version of file that wordpress is using is 3.3

Database will create some data and that data cannot be lost

We want to persist data

Mount volume inside /var/lib /mysql

Depends_on

: dependency, ensures that is service is up and running , then only it will run

Networking is now enabled between these two containers

version: '3.3'

services:

db:

```
image: mysql:5.7
volumes:
- db_data:/var/lib/mysql
restart: always
environment:
MYSQL_ROOT_PASSWORD: somewordpress
MYSQL_DATABASE: wordpress
MYSQL_USER: wordpress
MYSQL_PASSWORD: wordpress
```

```
wordpress:
depends_on:
- db
image: wordpress:latest
ports:
- "8000:80"
restart: always
environment:
WORDPRESS_DB_HOST: db:3306
WORDPRESS_DB_USER: wordpress
WORDPRESS_DB_PASSWORD: wordpress
WORDPRESS_DB_NAME: wordpress
volumes:
db_data: {}
```