

Docker

Isolated environment that run the application in a container.

- It not like the virtual machine. It run many application inside the one platform.
- In vm that run every application run in separate operating system. but in docker all application run in one platform.
- Use less memory and not need full os and less resource, less disk space.

`docker ps` - It shows the container that running currently in the daemon.

Docker image

- It is the base for anything and the snapshot for the application.
- It have everything that application need to run.
- More images store in **docker registry**.
- This is build by layers.
 - ex : `f03b40093957: Downloading 22.82MB/31.4MB`
`eed12bbd6494: Downloading 23.35MB/25.77MB`
`fa7eb8c8eee8: Download complete`
`7ff3b2b12318: Download complete`
`0f67c7de5f2c: Download complete`
`831f51541d38: Download complete`
 - This is the basic layers for the nginx image if this layers is used in another images it check first in the daemon then download that not present in the daemon.
 - It help to re-download the same layers. avoid net and storage wastage.



`$ docker images` → To see the images that have local in the machine.



`$ docker pull [image_name]` → It download the image from the registry.



`$ docker run [image_name:tag]` → It help to run the image inside the container.



`$ docker run -d [image_name:tag]` → It run the image in background.



`$ docker rmi [image_id]` → It delete the image completely. If want back use `$ docker pull [image_name]` command.

Docker container

It is the instance of the image.

- Can run many version of the container using image.
- Every time run the image new container and container id will created.

Managing container using commands



`$ docker container ls` → It gives the container that running currently. `[docker ps]` → It also same as this.



`$ docker stop [container_id]` → It stop the container immediately.



`$ docker start [container_id_or_name]` → It re-run the container.

Every time run the image new container and container id will created.



`$ docker rm [container_id_or_name]` → It delete the container permanently.



`$ docker ps -a -q` → It show all the containers id.



`$ docker rm $(docker ps -aq)` → It remove all the container that created.



We can't remove the running container to remove the running container this is usable

`$ docker rm -f $(docker ps -aq)`.

Naming container



`$ docker run --name [container_name]` → It put the name for the container.

It help to easily take the container and manage the container.

Exposing Ports

We can request the container to map to the localhost.

To map the container to the localhost this command will help.



`$ docker run -d -p 8080:80 nginx:latest` → This is the example to map the localhost to the container.



`$ docker run -d -p 8080:80 -p 3000:80 nginx:latest` → This is help to map the multiple host to the container.

It run on two hosts in the local machine localhost:8080 and localhost:3000

docker ps Format

In this `$ docker ps` this command is not formatted it bit hard to read while use this. So some steps to make this in readable format

- `export`

`FORMAT="ID\t{{.ID}}\nNAME\t{{.Names}}\nIMAGE\t{{.Image}}\nPORTS\t{{.Ports}}\nCOMMAND\t{{.Command}}\nCREATED\t{{.CreatedAt}}\nSTATUS\t{{.Status}}`

→ This is gives the default style to the showing containers command and it stored in the variable name `FORMAT` in default.

- Then run the `$ docker ps --format=$FORMAT` it will show the formatted view of the `$ docker ps` command.

docker volumes

Volumes between host and container:


It allow to share the data between the **host and container and between containers**.

1. Create the volume.
2. `$ docker run --name some_name -v /local_directory:/usr/share/nginx/html -d nginx` → In some content that paste the current directory.
3. It run the image nginx in some_name named container and it hosted to the the local_directory.
4. If we change the **local_directory** it will reflect in default nginx storage **/usr/share/nginx/html**

Make a website inside the container

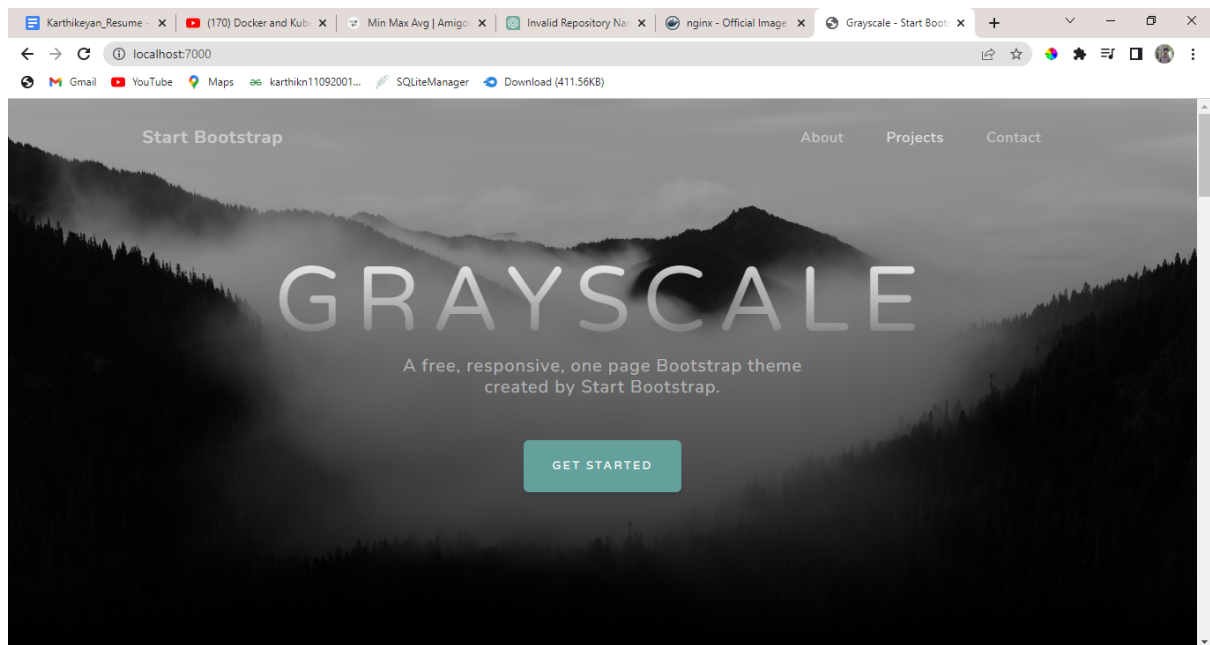
Steps:

1. Go to the website called Bootstrap Landing Page and select the desired website with source code.
2. Now paste the source code to the local_directory that mapped to the container.

 `$ docker exec -it [container_name] bash` → To execute the container with bash shell interactive.

3. That's all now refresh the host website you can see the web page that hosted.

This is the sample image that ran from localhost:7000



- In URL box that you can see localhost:7000 running.

Volumes between Containers

It done by simple one command

 `$ docker run --name [new_container_name] --volumes-from [container_that_want_to_be_volume] -d -p host_number:80`

Dockerfile

To create the new image for our own usage.

- Steps to build the image.

- we need some commands to build a image from this file.
- `FROM nginx:latest` → this is help to build a image from base image.
 - we can't build a entire base image from scratch.
- `ADD . /usr/share/nginx/html` → this mount the current directory files to the static content of the base image.
- `WORKDIR /some_name /container_dir` → This is create the directory inside the container.
- `RUN command` → This run the command to install the dependencies or some other opearions.
- `CMD cmd_to_run_the_app` → It run the application inside the container.

To build the image

Image contain everything that application needs to run like (os,software,app code).



`$ docker build --tag [image_name]:[tag_version] [dockerfile_location]` → It will help to build a own image.

- In a **image_name** name that you want to your image and the **tag** mention the version of your image.



`$ docker run --name [container_name] -p 8080:80 -d [image_name]:[tag_version]` → It run the container based our image in the port 8080.

First API

Node js

- Download and installed node js on windows.
 - This the guide to install the nodejs on windows : [link](#)
- Now using **express js** to build a api and run an localhost.
 - This is the guide to run an sample hello program in api.
 - [Helloworld_api](#)
- Then dockerize the api using docker file.

After installed

- Open the terminal and create the directory for woking.



```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(Example app listening on port ${port})
})
```

Paste the above code with js extension.

- `npm init` → It initialize the package.json file for the application.
- `npm install express` → It install the express in the save the dependencies default. `npm install express --no-save` → It is temporary.
- `node --verison` → To check the version.

- `node [file_name.js]` → It connect the api to the port number that mentioned in the js file.

Dockerize the

- Build the docker file inside the directory.
 - `$ docker build -t image_name [dir_doc_fie]`
- Run the image to create the container and run the application inside the container.
 - `$ docker run --name container_name -d -p port_number:default_port image_name:tag_version`

.dockerignore file

It file created where the dockerfile is located and it ignore the file that we don't want to add to the docker image.

Using cache

In dockerfile we need to add the dependencies to run the application there is command that used `RUN npm install` this is comes after the `ADD` the source code command.

- It would cause some problems.
 - Every time change the source code it take time to add the dependency.

```
FROM node:latest
WORKDIR /app
ADD . .
RUN npm install
CMD node index.js
```

This is a example of the above problem. To solve this there is way to install the dependency before add the source code get added.

```
FROM node:latest
WORKDIR /app
ADD package*.json ./
RUN npm install
ADD . .
CMD node index.js
```

Above problem solve the installation dependency every time, instead it using cache to take the dependencies.

Alpine

Always the pulling and building image took long time and massive size of storage to reduce the size **linux-alpine** is used.

what it is?

- It is distrubution of linux for security and reduce the size of the image.
- To learn more about it read this : <https://www.alpinelinux.org/about/>
- This is used when the image building is out of hand.

This is more much reduce the size of the image.



`$ docker pull nginx:alpine` → It pull the image from alpine version



`$ docker pull node:alpine` → It pull the latest alpine version of the node image.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
node	latest	304294f4bd8f	21 minutes ago	1GB
own	latest	35f75cbdd4fd	19 hours ago	145MB
nginx	alpine	fe7edaf8a8dc	2 days ago	41.4MB
nginx	latest	f9c14fe76d50	2 days ago	143MB
node	lts-alpine	6f44d13dd258	6 weeks ago	175MB

see the size column.

Change the Own image using alpine

It start from Dockerfile change the FROM line and add the tag is alpine.

`FROM nginx:latest` to `FROM nginx:alpine`

`FROM node:latest` to `FROM node:alpine`

Tag Version Control

It is strongly recommended to use the tag version of the image.

- It will safe to see the latest changes made in the project or image.
- It is easy to maintain the project.

To create the own tag version.



`$ docker tag image_name:tag_version [new_image_name:new_tag_version]`

- Build the latest version every time and tagged to the new_tag_version. like this example.



`$ docker build -t website:latest .` → It build the latest version of the website.



`$ docker tag website:latest website:1` → It tagged to the version : 1 image and created the new image.

Imagine someone changed the source code in some part.

then create the latest version of the website again.



`$ docker build -t website:latest .` → It build the latest version of the website after changed the code.



`$ docker tag website:latest website:2` → It tagged to the version : 2 image and created the new image.

Now we can run the container of this images on different ports to see the changes.

- `$ docker run --name old_web -d -p 8080:80 website:1` → It run the old-version of the website on port 8080.
- `$ docker run --name new-web -d -p 8081:80 website:2` → It run the new version of the website on port 8081.

Docker Registry

- It is server side application to store and distribute images.
- Can used in CD/CI pipeline.
- It help to push the **host side** application to the **Server side**.

Steps to push images:

1. Create the docker hub account on docker official website here is the link : <https://hub.docker.com/> .
2. Open the terminal and type `$ docker login` .
3. Now Enter your credentials and you will logged in successfully.
 - a. In docker hub website create a repository first.
4. Now you can push the images to the repository in docker hub.



```
$ docker push [user_name/repository_name:tag_version]
```

Change the image name similar to the repository name is help to easily push the image.

Once the image is pushed it will show in repository.

Steps to pull:



```
$ docker pull [user_name/repository_name:tag_version]
```

If the tag version is not mentioned it will pull the latest version of the image.

Docker Inspect



```
$ docker inspect image_id
```

 → It will give the detail explanation of the image.

Docker Logs



```
$ docker logs image_id
```

 → It help to see the network traffic of the image.

```
$ docker logs -f image_id
```

 → It help to follow the image with network requests made to the image.

Docker execute



```
$ docker exec -it [container_id] /bin/bash_or_sh
```

 → It help to get into the container.