Dockers

Developers are likely familiar with Docker from their work experience. And we probably already know how crucial it is for any application developer to understand this technology. To understand in detail about this topic, lets first understand about virtualization.

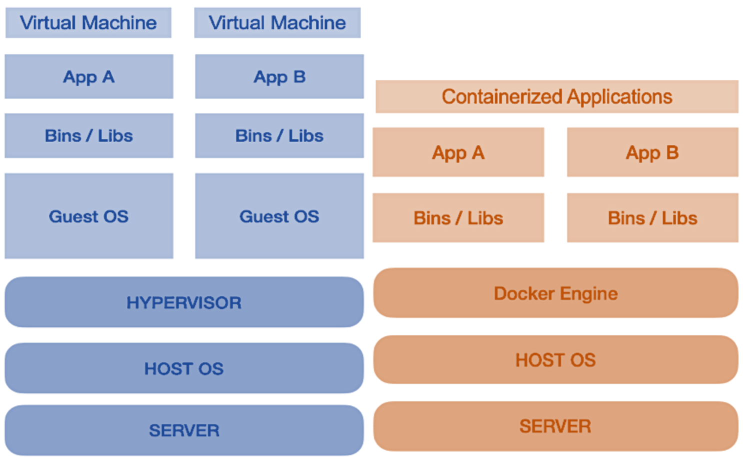
Need for Virtualization Technology

Every programme in an IT environment, with several versions must be executed in an isolated environment to avoid conflicts with hardware requirements and software dependencies. The aforementioned scenario's requirement is met using virtualization technology.

Virtual Machine (VM) VS Docker

- A Virtual Machine (VM) is a virtual environment that performs as a virtual computer system with its own CPU, memory, network interface, and storage (located off- or on-premises) on a physical hardware system.
- In contrast to virtual machines, a container can share the operating system's kernel while merely loading its various binaries and libraries.
- A container runtime is called Docker. Many people mistakenly believe that Docker was the first of its kind, yet Linux containers have been around since the 1970s.
- Containers help isolate specific kernel processes which make them believe they are the only ones running on a brand-new computer.
- In other words, you don't need to install a completely other operating system inside of your host OS. It is possible to run several containers on a single OS without installing multiple guest operating systems.

*In general, a **container** is an abstraction at the application layer that packages code and dependencies together. Instead of virtualizing the entire physical machine, containers virtualize only the host operating system.*



Hypervisor is a software that simulates hardware such as CPU , MEM etc so that guest OS thinks its running on physical system

Docker Engine is the core of the open source Docker platform. It is a lightweight container runtime that builds and runs Docker containers. Similar to java runtime required to run java applications, it is docker runtime which is used for running docker containers.

Docker Overview

- It is an open-source platform for creating, distributing, and running programmes.
- We can divide the apps from infrastructure with the help of Docker, allowing for rapid software delivery.

- We can manage the infrastructure using Docker in the same manner that manages applications.
- We can drastically shorten the time between writing code and executing it in production by utilising Docker's methodology for shipping, testing, and deploying code quickly.
- Things that are created and saved locally can be easily used in servers(cloud platform).
- It is OS independent
- To create docker images we need an isolated environment called as containers(running instance of an image)

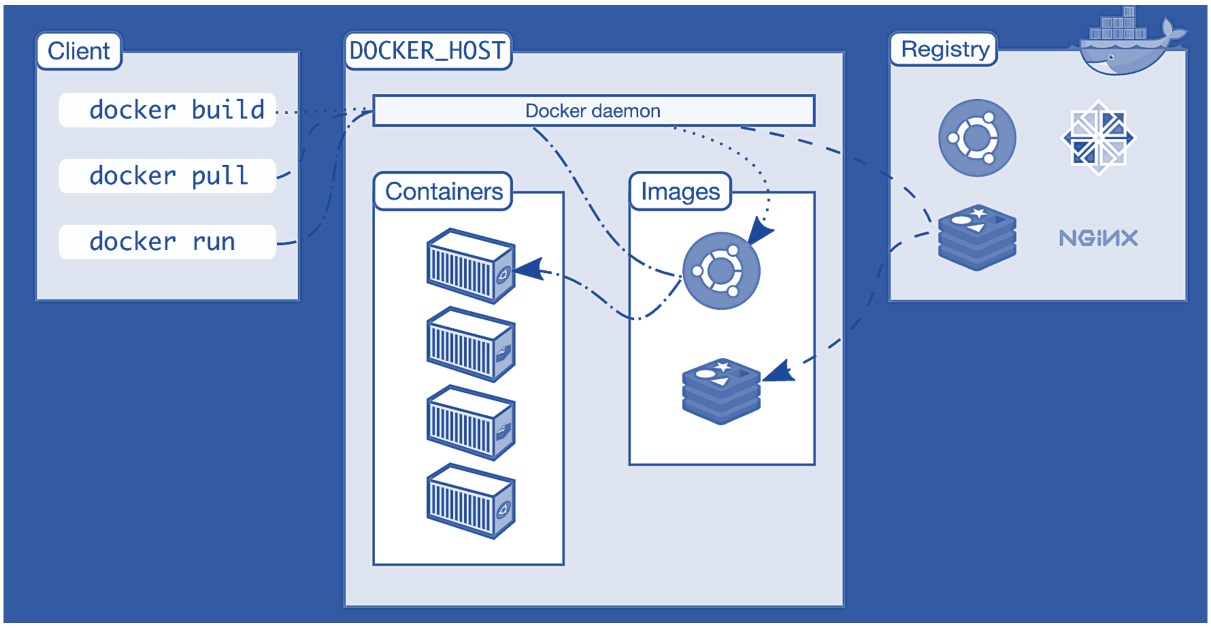
Why Dockers are popular?

- Virtual Machines (VMs) need Guest OS
- VMs are slow and take lot of time to boot
- Docker containers share the Host OS for its requirement of system related function calls such as Disk I/O, Memory etc. Hence it has no kernel of its own and its Lighter and Faster
- As Dockers host the operating system and share the necessary libraries, containers are quick to start up.
- Contrary to virtual computers, containers do not consume or obstruct host resources.
- Libraries and binaries isolated to the programme being run are available in containers.

Benefits of using docker:

- It runs container in seconds instead of minutes
- It utilizes less resource which results in less disk space
- It occupies less memory
- It does not need full OS
- Easy Deployment and Testing

Docker Architecture



The Docker Daemon

The Docker daemon manages Docker objects like images, containers, networks, and volumes while listening for requests made over the Docker API. To manage Docker services, a daemon can also talk to other daemons.

The Docker client

Many Docker users interact with Docker primarily through the Docker client (docker). When you issue commands like "docker run," the client delivers these instructions to the dockerd service, which executes them. The Docker API is used by the docker command. The Docker client can speak with many daemons.

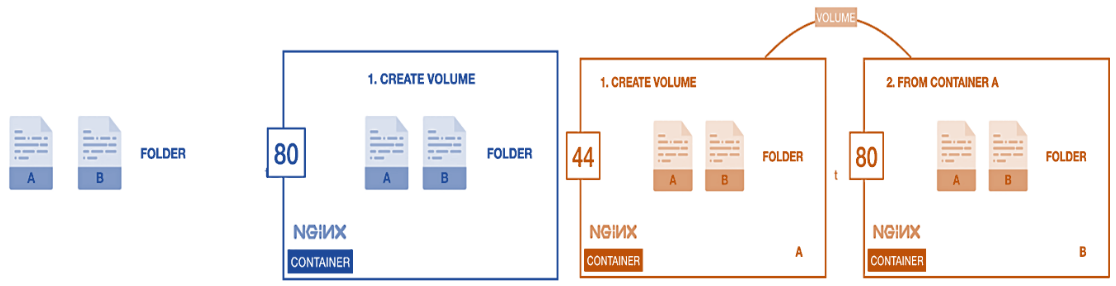
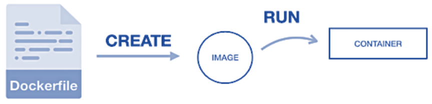
Docker registries

Docker images are kept in a registry. Anyone can utilise Docker Hub, a public registry, and by default, Docker is set up to search there for images.

For detailed understanding of dockers, please refer to the official docker documentation *https://docs.docker.com/get-started/overview/*

Docker Commands:

Command	**Explanation**
docker	- To check what are all the commands
docker —version	- To check the version
docker ps	- To see whether any container is running. If so we can see the details of the container
docker pull nginx	- 'pull' helps us to download the images. Here the image name is ngnix.
docker images	- To check whether the image is downloaded
docker run nginx:latest	- To run a container, use this command along with the tag.
docker container ls	- To check whether the container is running or not and at which port it is running
docker run -d nginx:latest	- To run a container in detached mode, use this command along with the tag.
docker stop container_id	- To stop the container from running when used in detached mode
docker run -d -p 8080:80 nginx:latest	- To run the container in detached mode with mapping the ports to 8080:80. We can change the ports as per our wish.
docker run -d -p 8080:80 -p 3000:80 nginx:latest	- To map two ports simultaneously
docker start container_id or container_name	- It would start the container back as it is not deleted
docker ps -a	- Lists all the containers that are available, even the ones which are not running
docker ps —help	- It will show all the commands associated with the containers
docker rm container_id or name	- It deletes that specific container
docker ps -a -q	- Displays all the containers ids alone. -q represents quiet mode, displays only the ids
docker rm $(docker ps -aq)	- Deleting all the containers at once. But this will not delete the containers that are running.
docker rm -f $(docker ps -aq)	- This helps to close all the containers and even running container using -f (forced command)
Creating custom name for the container	
docker --name container_name run -d -p 8080:80 -p 3000:80 nginx:latest docker --name container_name run -d nginx:latest docker --name container_name run nginx:latest	- Under container_name we can use whatever the name we want. This command creates a name for the container then spins it at port 8080 & 3000 under nginx:latest docker image. The later part after the container name is just an example with two ports mapped together, it can be done with one port mapping or without any port mapping
export FORMAT="ID\t{{.ID}}\nNAME\t{{.Names}}\nIMAGE\t{{.Image}}\nPORTS\t{{.Ports}}\nCOMMAND\t{{.Command}}\nCREATED\t{{.CreatedAt}}\nSTATUS\t{{.Status}}\n" docker ps --format=$FORMAT	- Formatting the container information for better view. After setting this we can check the container information in a much better formatted way. Here FORMAT is the variable name that has been set for formatting the container. - Then next command helps to see the container information in the formatted way.
To change the docker image content	
docker run --name website -v $(pwd):/usr/share/nginx/html:ro -d -p 8080:80 nginx	- Create a new html file and save it in local system - Map the created image to the actual file path (pwd - represents the local file path where the html is saved, /usr/share/nginx/html:ro – The path represents the path of the original image, which can be seen in the link. - :ro represents the read-only mode

docker exec -it website bash	- To execute the same command in interactive mode (i.e inside the container)
ls -al	- List all the files inside the container(we can see linux file structure)
cd /usr/share/nginx/html/	- Navigate to the path where html file is stored inside the container
ls -al	- We can find the index.html file inside the directory
touch about.html	- To see whether we can edit the files inside the container. It will not allow us to edit/write anything, it is just a read only file, as we used only (:ro) command. - If we navigate inside a container, we can only read the files, we cannot write anything
docker run --name website -v $(pwd):/usr/share/nginx/html -d -p 8080:80 nginx	- To edit the files inside the container, use this command. Removing ':ro' part alone in the above command to make it editable.
docker exec -it website bash cd /usr/share/nginx/html/ ls touch about.html	- Now as we can edit the files inside the container, try creating a new html file inside that directory using touch command (touch command is for creating a file)
To create a volume between containers Docker allows sharing of data(files & folders) - Between host and container - Between containers  FOLDER A B 80 1. CREATE VOLUME NGINX CONTAINER VOLUME 44 2. FROM CONTAINER A	
docker run --name website-copy --volumes-from website -d -p 8081:80 nginx	- We can create a new container with the volumes of the existing container. - Here website-copy is the name of the new container (we can have any name). It is created by using volume from existing container(website). - The port has been changed to 8081. Whatever the localhost:8080 displays, the same will be displayed in 8081
Creating a new docker image with dockerfile 	
FROM nginx:latest ADD . /usr/share/nginx/html/	- Create a docker file (Dockerfile) inside the same folder where the html file is created - FROM command will extract the image from docker hub. nginx:latest – is the name of the image, we can use any image name of our choice. This will help us to utilize the image features for our work. - usr/share/…/html/ —> is the destination location(can be referred from docker hub website), we can specify it as per the image used. - Add command will add the all the files in the local folder to the destination mentioned.
Creating a new docker image: - Docker image is a template for creating an environment of our choice - It is a snapshot that has everything needed to run the Apps - OS, Software and App Code	

docker build --tag website:latest . docker build -t website:latest .	- Here -t or --tag both denotes the same meaning. '.' denotes that all the files needed to be build are present in the same working folder.
docker images ls docker images	- This command is to check whether the image is created or not
docker run --name website -p 8080:80 -d website:latest	- website:latest is the name of the image created under the created container name website. - Before running any container at port 8080, try to stop containers running at port 8080 – if any container is already running at that port.
docker ps --format=$FORMAT	Use this command to check the ids and other information about the containers.

Docker Ignore File: Adding docker ignore file to ignore the files/folders that are not needed during docker build command. This can avoid unnecessary overwriting and ignore unwanted files.

Dockerfile	- It does not add any meaning to the image, it is just for creating the image
.git	- .git - If git folder/file is added for pushing the files to repo, it is not needed while creating an image - Similarly, any files that doesn't add any meaning to the image can be added to the .dockerignore file. The .dockerignore file will leave them out during building of the image.
foldername/**	- If any folder to be ignored and multiple subfolders inside the folder to be removed, then it can be done with **

Layers and caching from building images from our docker file:
With the help of caching creation of docker image becomes much faster. Caching helps to store the information while creating the image at the first time. So, when the image is re-created with few modifications, it will be created faster with the help of caching.
Considering an example: While utilizing docker for creating an image for an api, we might see few files like package.json or package-lock.json which can be utilised for this process. We can modify the docker file with the utilisation of caching.

Varying the size of Docker Image:
While downloading the docker images we can download a reduced image size using alpine-linux version. To see the difference, we can try downloading the images with and without alpine.

docker pull node:latest	- Size of the image is 991 MB
docker pull node:alpine	- Size of the image is 123 MB

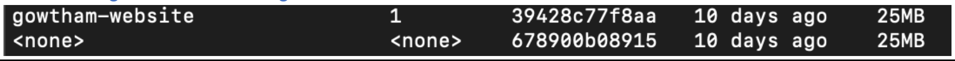
**** Pls check the docker hub website with the name of the images for alpine version if there are any doubts**

Tags and Versions:
While installing a docker image which going to be utilised for many purposes for a long time, it is preferred to install it with the tags and version number. So, when we create a docker image with some other docker image, it will be easy for the other person to install and utilize the same version.

- It allows you to control image version
- It avoids breaking changes
- Safe

docker pull node:16.17.1-alpine3.15	For node the alpine version represented here is 16.17.1 and the supported alpine that is available is alpine3.15. Please check for other versions in docker-hub website for clarification. The above command helps to pull the alpine version of node with version number – 16.17.1

Tagging Override:
While creating a docker image, if we use the same name with tag multiple times, then the newly created image will have the name and all the old images will be replaced with <none> tag.
Eg: In the below picture, docker image has been created with same name. The latest one has the name and the older is changed to <none> tag. We can see this by using the command
docker images or docker image ls



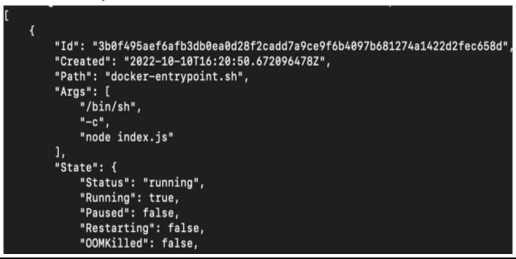
```
gowtham-website                1         39428c77f8aa   10 days ago    25MB
<none>                         <none>    678900b08915   10 days ago    25MB
```

Docker Registry - Pushing images to docker-hub:

- Docker Registry is a highly scalable server side application that stores and lets you distribute Docker images.
- It is used in our CI/CD pipeline
- It helps run our applications

Some of the docker registries that are available,

- Docker Hub
- quay.io
- Amazon ECR

Here I have given a demo with respect to docker hub. To utilize the services of docker registry, Create a new repo in the docker hub website under your login. Let the created repo be private or public according to your choice. Tag the created images to suitable names for pushing it to the repo.	
docker tag gowtham-website gowtham12591/website-api	This is converted to the format of docker hub(we can refer to the docker hub login page for the format)
docker push gowtham12591/website-api	Push the tagged images to docker hub
Pulling Own Images: We can pull our own pushed images from docker-hub.	
docker pull gowtham12591/website-api	If tag is not mentioned, it will look for default 'latest' tag, but if it is not there then it will throw an error. So please mention the tag if it is used.
docker run -- name gowtham-website-api -d -p 8000:80 gowtham12591/website-api	Test the pulled image by spinning a container. Here the container name is gowtham-website.
Docker Inspect: Docker inspect provides detailed information on constructs controlled by Docker. By default, docker inspect will render results in a JSON array. We can inspect a running container with its id	
docker inspect 54eecdab48ec	54eecdab48ec is the container id, we can also put the name instead of container id.
Docker Logs: We can see the logs of a running container	
docker logs container_id	This will show what the container is listening to
docker logs --help	This will show different options that can be used in logs
Docker Exec: Spin a container and then with the container id we can enter the inside the docker and check its root files	
docker exec -it 7f5e7be5dd75 /bin/sh	Here 7f5e7be5dd75 is the container id and /bin/sh is the command for navigating inside the container /bin/sh is not common for all the containers. To check which command to navigate inside the container, we can use the container id and use inspect command.
docker inspect 7f5e7be5dd75 	This will show all the information about the container. The image shows some part of the information. Under "Args" – We can see /bin/sh – This helps to navigate inside the container, similarly we can find information about other containers as well using inspect command.

```
[
    {
        "Id": "3b0f495aef6afb3db0ea0d28f2cadd7a9ce9f6b4097b681274a1422d2fec658d",
        "Created": "2022-10-10T16:20:50.672096478Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "/bin/sh",
            "-c",
            "node index.js"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
```

Once navigated inside the container, we can use ls -al command see all the root files. The 'exec' command helps to see the root files inside the container.	