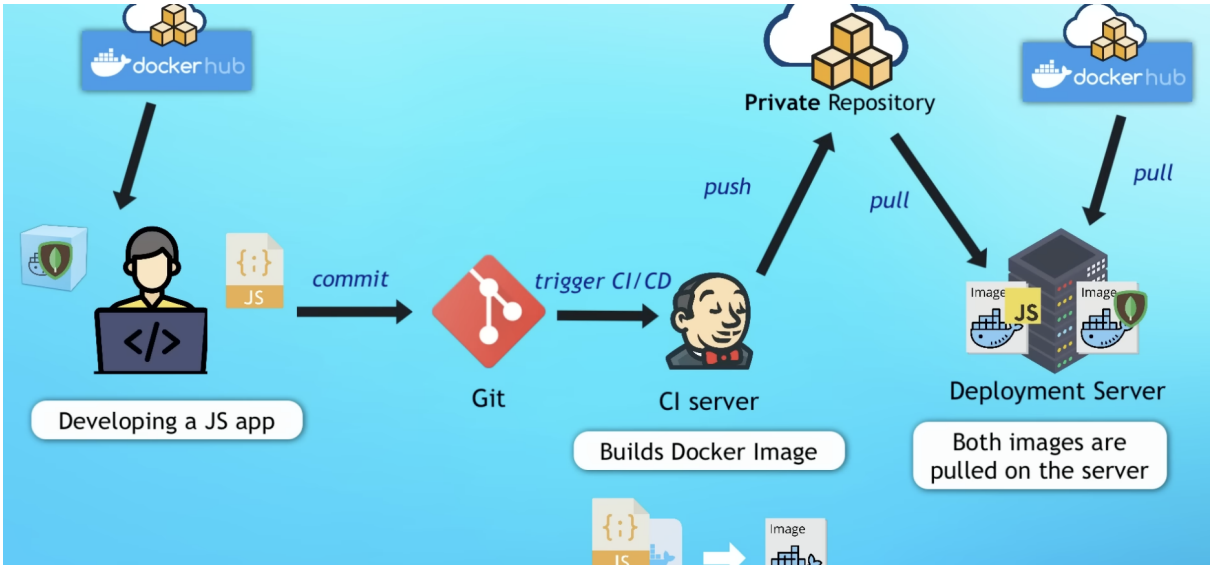


Docker for Beginners:

Docker Desktop is a software application that allows users to easily create, manage, and run Docker containers on their local computer. Docker containers are lightweight, portable environments that can run applications and services, and they provide a consistent environment across different systems.

With Docker Desktop, users can:

1. Build Docker images: Docker images are the blueprints for Docker containers. Users can use Docker Desktop to create their own images or download pre-built images from the Docker Hub.
2. Run Docker containers: Docker Desktop allows users to run one or more containers on their local machine. These containers can run different applications or services, each with its own isolated environment.
3. Manage Docker containers: Docker Desktop provides an easy-to-use interface for managing containers, including starting, stopping, and removing containers.
4. Share Docker containers: Docker Desktop also allows users to share their containers with others through the Docker Hub or by exporting and importing container images.

Docker Desktop is available for Windows and Mac operating systems, and it includes a graphical user interface (GUI) that makes it easy for users to manage Docker containers without having to use command-line tools. It also includes

tools for monitoring container resources, troubleshooting container issues, and more.

- Virtualisation Software
- Makes developing and deploying software much easier
- Packages application with all the necessary dependencies, configuration, systems tools and runtime. (It makes it easier to work)
- Portable artifact, easily shared and distributed.

> 💡 Before containers developers need to install a lot for tools/software/services depending on their OS. And sometimes it goes wrong as well as the installation and process were different on different OS.

With Container :

Development process with containers:

- Own Isolated environment
- Postgres packages with all the dependencies and configurations.
- we can start services as a Docker container using a Docker command. It will pull the config files from the internet and run on the system, command is same for all OS and all services.
- Easy to run different versions of same app without any conflicts.

> 💡 Before Docker, application teams used to send the application jar file with all the dependencies and documentation and services that were required for application. At times it can happen that there is a mis communication between Apps and Operations team and then communication carries on with blames that causes delay in deployment.

With containers, Developers create a application package that contains code, config fils and dependencies.

- No configurations needed on the server (except docker runtime)
- This way we can avoid a lot of errors and saves time.

Operations need to do few steps only.

- Install Docker runtime on the server.
- Run Docker commands to fetch and run the Docker artifacts.

Virtual Machines vs Docker :

Virtual Machines:

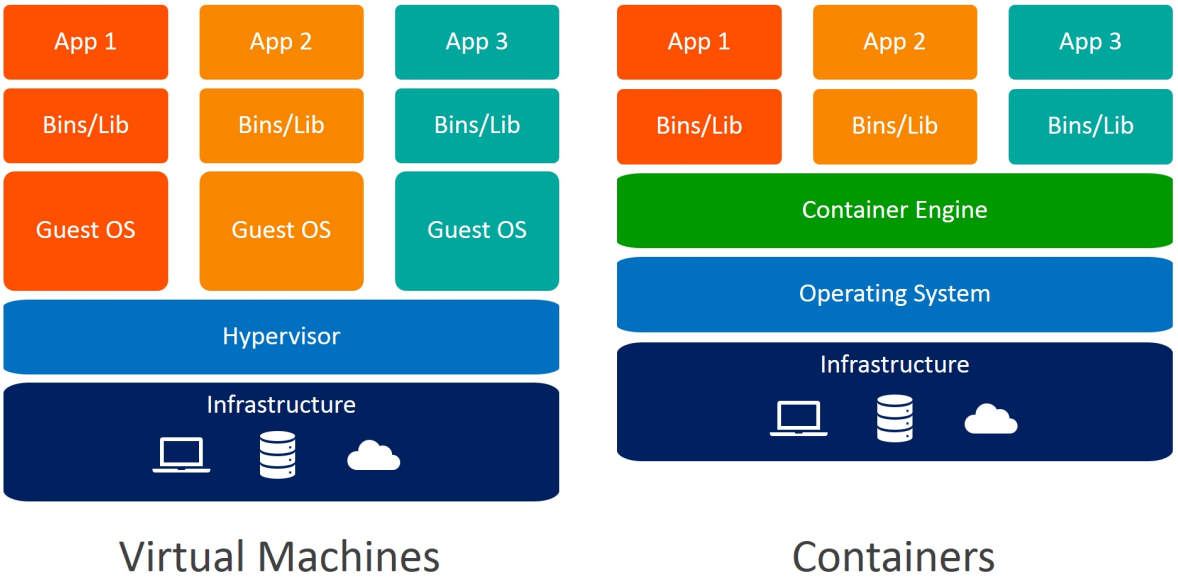
- Emulate a complete hardware environment
- Allow running multiple operating systems and applications on a single physical server
- Require a significant amount of memory and storage space to run
- Provide strong isolation and security between applications
- Can be resource-intensive and may have longer startup times

Docker:

- Use containerization to share a single operating system kernel
- Allow lightweight deployment and running of applications
- Share the underlying OS kernel with other containers on the same host
- May not provide the same level of isolation as VMs
- Require less memory and storage space to run
- Have shorter startup times
- Provide a consistent environment for development, testing, and deployment

Note:

```
- Docker Virtualises the application layer.
- Contains the OS application layer.
- Services and apps installed on top that layer.
- It uses the Kernel of the host.
- VM, virtualises the complete OS as it virtualises the appli
- Docker images are much smaller in size, docker images varie
- Containers takes seconds to run while VMs takes hours to ru
- Vm is compatible with all the OS but Docker is only compati
```



Docker Desktop :

Docker Desktop is a software application that allows users to easily create, manage, and run Docker containers on their local computer. Docker containers are lightweight, portable environments that can run applications and services, and they provide a consistent environment across different systems.

With Docker Desktop, users can:

1. Build Docker images: Docker images are the blueprints for Docker containers. Users can use Docker Desktop to create their own images or download pre-built images from the Docker Hub.
2. Run Docker containers: Docker Desktop allows users to run one or more containers on their local machine. These containers can run different applications or services, each with its own isolated environment.
3. Manage Docker containers: Docker Desktop provides an easy-to-use interface for managing containers, including starting, stopping, and removing containers.
4. Share Docker containers: Docker Desktop also allows users to share their containers with others through the Docker Hub or by exporting and importing container images.

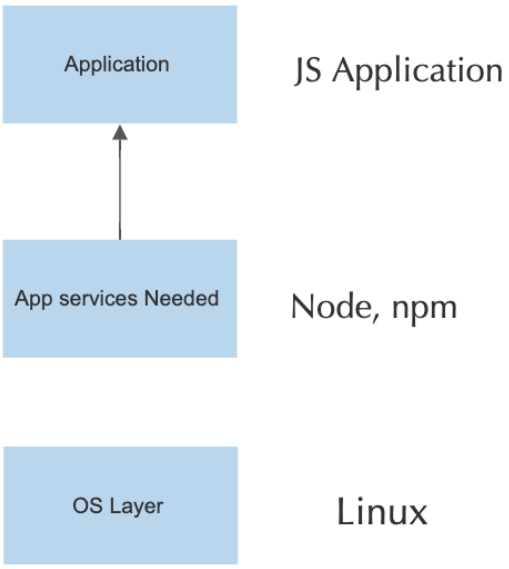
Docker Desktop is available for Windows and Mac operating systems, and it includes a graphical user interface (GUI) that makes it easy for users to manage Docker containers without having to use command-line tools. It also includes tools for monitoring container resources, troubleshooting container issues, and more.

Docker Images :

They are executable packages(bundled with application code & dependencies, software packages, etc.) for the purpose of creating containers. Docker images can be deployed to any docker environment and the containers can be spun up there to run the application.

- An executable application artifact.
- Includes app source code, but also complete environment configuration.
- Add environment variables, create directories etc.

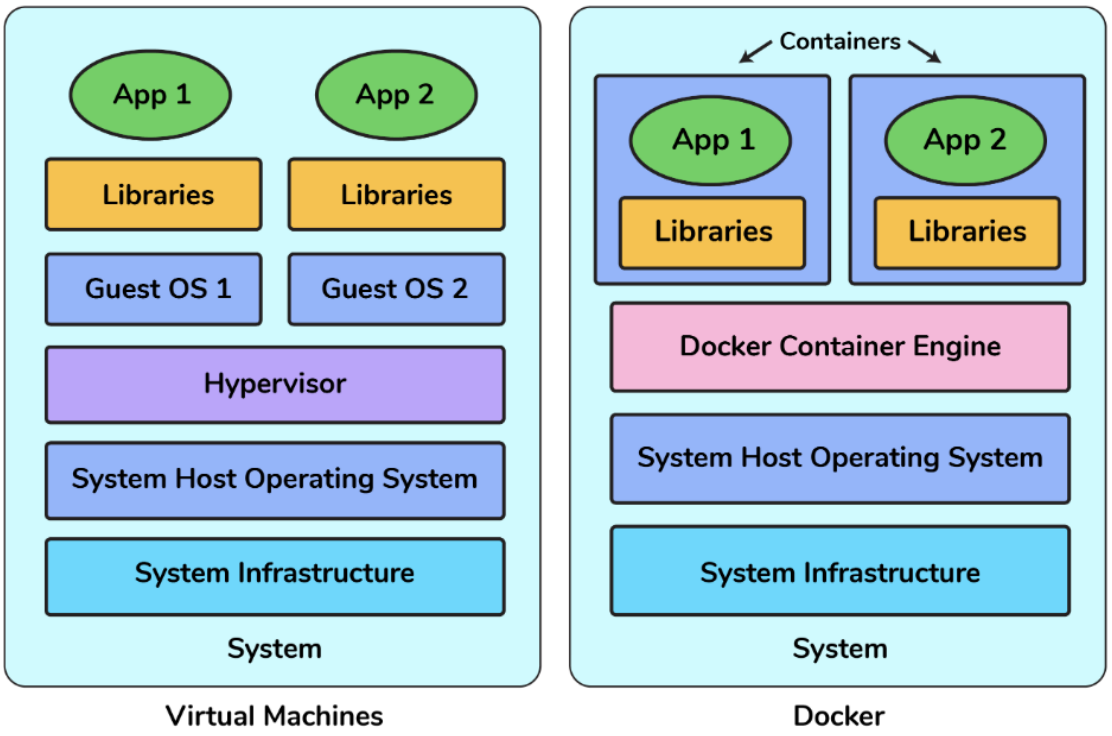
All the information are packed in the docker image with the application code.



Docket Container:

A container is a standard unit of software bundled with dependencies so that applications can be deployed fast and reliably b/w different computing platforms.

- Docker can be visualized as a big ship (docker) carrying huge boxes of products (containers).
- Docker container doesn’t require the installation of a separate operating system. Docker just relies or makes use of the kernel’s resources and its functionality to allocate them for the CPU and memory it relies on the kernel’s functionality and uses resource isolation for CPU and memory, and separate namespaces to isolate the application’s view of the OS (operating system).



- A running instance of an image is called as container.
- We can run multiple containers with one image or different images with separate environment.

Docker Registry:

- A storage and distribution system for Docker images.
- Official images available from application like Redis, Mongo, Postgres etc.
- Official images are maintained by the software authors or in collaboration with Docker community.
- Docker hosts one of the biggest Docker Registry, called “Docker hub”. Where companied and individuals developers has uploaded their official images verified by docker and any one download it from their.
- Docker images are versioned as well, as the technology changes with time or requirement.
- Docker versions are identified by tags.

Docker Basic commands:

1. `docker pull`: This command is used to download a Docker image from a registry, such as the Docker Hub.

```
docker pull -d nginx:latest: It will run the pull process in
```

```
amansrivastava@Amans-MacBook-Air ~ % docker pull nginx:latest
latest: Pulling from library/nginx
Digest: sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

```
amansrivastava@Amans-MacBook-Air ~ % docker pull  mongo:6.0.4
6.0.4: Pulling from library/mongo
986c4f6be307: Downloading [==================================>                ]  19.14MB/28.39MB
41dd2f9a5253: Download complete
8aa47abefa19: Download complete
061fd5097347: Download complete
130857d0db26: Download complete
22ed9280970a: Download complete
7936c45da7e5: Download complete
d921cae36514: Downloading [====>                                              ]  17.78MB/194.6MB
98c4148c49d4: Download complete
```

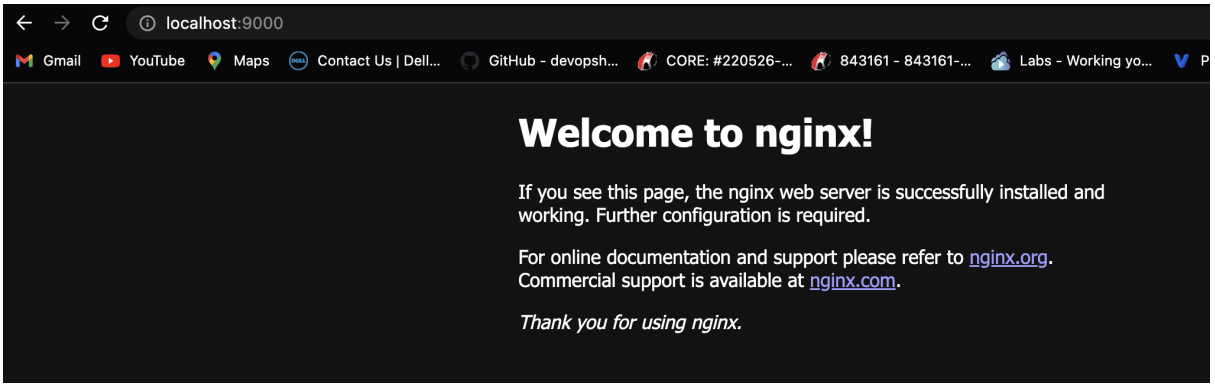
2. `docker ps`: This command is used to list all running Docker containers on the host machine.

```
amansrivastava@Amans-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS                    NAMES
c8348758306a   nginx:1.23    "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes   0.0.0.0:9000->80/tcp     sharp_jemison
```

3. `docker run -d -p 9000:80 nginc:1.23`: This command will run the docker container in background and it will bind the port 80 with port 9000. The local host will look like below output.

```
docker run --name webapp -d -p 9000:80 nginc:1.23: This will
This commands always creates a new container it does'nt uses
```

```
amansrivastava@Amans-MacBook-Air ~ %    docker run -d -p 9000:80 nginx:1.23
c8348758306a0975998b05afa486fbc1e59a42b5db285f2b0a7f1b3dee19700f
amansrivastava@Amans-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS                    NAMES
c8348758306a   nginx:1.23    "/docker-entrypoint.…"   8 seconds ago   Up 7 seconds   0.0.0.0:9000->80/tcp     sharp_jemison
```



4. 1. `docker logs (container name)`: This command is used to show the container logs.

```
amansrivastava@Amans-MacBook-Air ~ % docker logs 8c62d7e17072
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/03/05 17:33:31 [notice] 1#1: using the "epoll" event method
2023/03/05 17:33:31 [notice] 1#1: nginx/1.23.3
2023/03/05 17:33:31 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/03/05 17:33:31 [notice] 1#1: OS: Linux 5.15.49-linuxkit
```

5. `docker stop`: This command is used to stop a running Docker container. It sends a signal to the container to stop gracefully.

```
amansrivastava@Amans-MacBook-Air ~ % docker stop c8348758306a
c8348758306a
amansrivastava@Amans-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
amansrivastava@Amans-MacBook-Air ~ %
```

6. `docker images`: This command is used to list all Docker images on the host machine.

```
amansrivastava@Amans-MacBook-Air ~ % docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
mongo        6.0.4     bd066eec4864   42 hours ago    617MB
nginx        1.23      114aa6a9f203   4 days ago      135MB
nginx        latest    114aa6a9f203   4 days ago      135MB
amansrivastava@Amans-MacBook-Air ~ %
```

7. `docker rmi`: This command is used to remove a Docker image from the host machine.

```
amansrivastava@Amans-MacBook-Air ~ % docker rmi nginx:latest
Untagged: nginx:latest
amansrivastava@Amans-MacBook-Air ~ % docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
mongo        6.0.4     bd066eec4864   42 hours ago    617MB
nginx        1.23      114aa6a9f203   4 days ago      135MB
amansrivastava@Amans-MacBook-Air ~ %
```

8. `docker rm`: This command is used to remove a stopped Docker container from the host machine.

```
amansrivastava@Amans-MacBook-Air ~ % docker rm c8348758306a
c8348758306a
amansrivastava@Amans-MacBook-Air ~ %
```

9. `docker ps -a:` It will list all the containers whether they are running or not.

```
amansrivastava@Amans-MacBook-Air ~ % docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                      PORTS     NAMES
3c62d7e17072   114aa6a9f203   "/docker-entrypoint.…"   48 minutes ago   Exited (0) 39 minutes ago             stoic_shamir
amansrivastava@Amans-MacBook-Air ~ %
```

10. `docker push`: This command is used to upload a Docker image to a registry, such as the Docker Hub.
11. `docker exec`: This command is used to execute a command inside a running Docker container.

12. `docker start containr_Id`: This command will start the container again

```
amansrivastava@Amans-MacBook-Air ~ % docker start 8c62d7e17072
8c62d7e17072
amansrivastava@Amans-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS         PORTS     NAMES
8c62d7e17072   114aa6a9f203   "/docker-entrypoint.…"   53 minutes ago   Up 4 seconds   80/tcp    stoic_shamir
amansrivastava@Amans-MacBook-Air ~ %
```

13. `docker stop containr_Id`: This command will stop the container

```
amansrivastava@Amans-MacBook-Air ~ % docker stop 8c62d7e17072
8c62d7e17072
amansrivastava@Amans-MacBook-Air ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
amansrivastava@Amans-MacBook-Air ~ %
```

Port Binding:

Bind the container's port to the host's port to make the service available to the outside world.

- Application inside a container runs in an isolated Docker network.
- We need to expose the container port to the host (the machine on which container runs on)
- Only one service can run on a specific port on the host.
- As per the standard we should use same port on the host machine same as docker container using.

Docker Registries: Public and Private Docker Registries

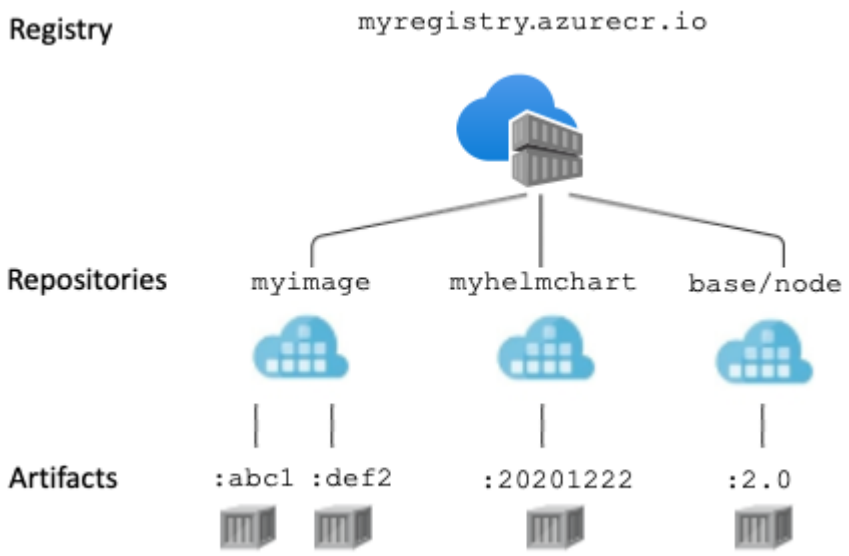
- Docker hub is the public Docker Registries.
- We need to authenticate before we access the Private repositories.
- Almost all Cloud providers have their private repositories, name AWS, GCP, Azure.
- Docker also has its own private repositories under Docker hub.

Docker Registry vs Repositories:

A Docker Registry is a central place where Docker images are stored, managed, and distributed. It serves as a repository for Docker images and is responsible for maintaining the integrity and security of the images. A Docker Registry can be either public or private, and it can be hosted on-premises or in the cloud. Docker Hub is a popular public Docker Registry that allows developers to share and distribute their Docker images.

A Docker Repository, on the other hand, is a collection of related Docker images with a common name. It can be seen as a logical grouping of Docker images that belong to the same application, service, or project. A Docker Repository can be hosted in a Docker Registry or on a local machine.

In summary, Docker Registry is the platform that stores and distributes Docker images, while a Docker Repository is a collection of related Docker images that can be hosted in a Docker Registry or on a local machine.



In the same way Docker Hub is a registry, On Docker Hub you can host private or public repositories for your applications.

Docker File:

- Docker file is a text document that contains commands to assemble an image
- Docker can then build an image by reading the instructions.
- To create a Docker image for an application, we have to create a docker file for the application.

Here's an example Docker file for a Node.js application:

```
# Use an official Node.js runtime as a parent image
FROM node:14-alpine

# Set the working directory to /app
WORKDIR /app

# Copy the package.json and package-lock.json files to the co
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the application files to the container
COPY . .

# Expose port 3000
EXPOSE 3000

# Set the command to start the application
CMD [ "npm", "start" ]
```

Here's a breakdown of what each line does:

- `FROM node:14-alpine`: Use the official Node.js 14 runtime as the parent image.
- `WORKDIR /app`: Set the working directory to `/app`.
- `COPY package*.json ./`: Copy the `package.json` and `package-lock.json` files to the container.

- `RUN npm install`: Install the dependencies using `npm`.
- `COPY . .`: Copy the rest of the application files to the container.
- `EXPOSE 3000`: Expose port 3000, which is the port that the application will listen on.
- `CMD [ "npm", "start" ]`: Set the command to start the application using `npm start`.
-

To build the Docker image, save the above code in a file called `Dockerfile` and run the following command from the directory where the Docker file is located:

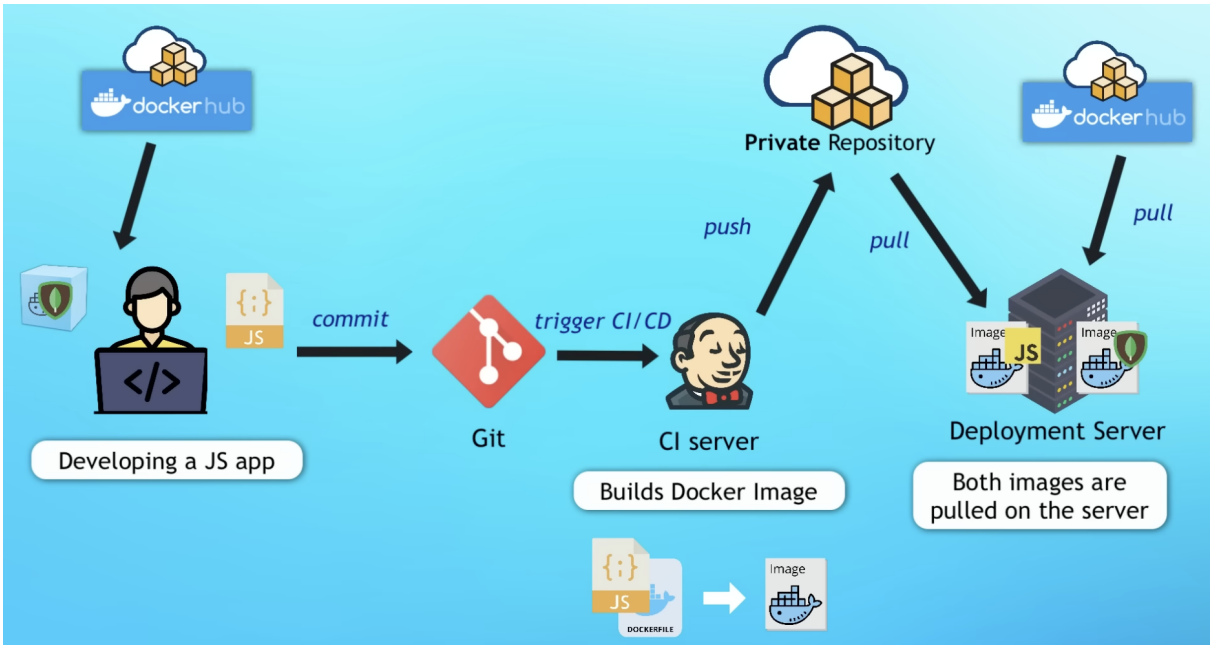
```
docker build -t my-node-app .
```

This will create a Docker image with the tag `my-node-app`. Once the image is built, you can run a container from the image using the following command:

```
docker run -p 3000:3000 my-node-app
```

This will start a container from the `my-node-app` image and map port 3000 on the host system to port 3000 inside the container. The application should now be accessible at `http://localhost:3000`.

Docker in complete software development lifecycle



A container is a way to package applications with everything they need inside the package including the dependencies and all the configuration.

Package is portable, it can be easily shared and moved around. It makes development and deployment process more efficient.

In the context of software development, a container is a lightweight, portable, and self-contained environment that encapsulates an application, along with its dependencies and configuration, in a standardized package. Containers provide a way to run software applications consistently across different environments, from development to production, without worrying about compatibility issues or infrastructure dependencies.

A container typically consists of the following layers:

1. **Application layer**: This layer contains the software application that needs to be run in the container. The application can be any software program or service, such as a web application, database, or messaging system.
2. **Dependency layer**: This layer contains all the dependencies required to run the application, such as libraries, frameworks, and runtime environments. The dependencies are typically installed and managed using a package manager or a container orchestration tool.

3. **Operating system layer**: This layer contains the operating system components required to run the application and its dependencies. Containers can run on any operating system, but the most commonly used operating systems for containers are Linux-based systems, due to their lightweight and scalable nature.
4. **Hardware layer**: This layer represents the physical or virtual hardware on which the container runs. This layer is usually abstracted away by the containerization platform, which provides a virtualized environment that emulates the necessary hardware components.

The role of containers is to provide a standardized and isolated environment for running applications, without the need for complex infrastructure configurations or management. Containers offer several benefits, including:

- Portability: Containers can run on any infrastructure that supports the container runtime, making it easy to move applications between different environments, such as from a developer's laptop to a production server.
- Scalability: Containers can be easily scaled up or down to meet changing application demands, without the need for manual intervention.
- Consistency: Containers provide a consistent and reproducible environment for running applications, which helps to avoid compatibility issues and ensures that the application behaves predictably across different environments.
- Security: Containers provide a level of isolation between applications and the underlying infrastructure, which helps to reduce the risk of security breaches and minimize the impact of any security vulnerabilities.

A Docker image is a read-only template that contains the instructions for creating a Docker container. It's essentially a snapshot of the application and its dependencies at a particular point in time. Docker images are created using a Dockerfile, which is a script that specifies the steps required to build the image.

A Docker container, on the other hand, is a runnable instance of a Docker image. It's created by running a Docker image, which creates a writable layer on top of the read-only image. The container has its own file system, network, and process space, which are isolated from the host system and other containers.

In summary, the main differences between Docker images and Docker containers are:

- Docker images are read-only templates that define the application and its dependencies, while Docker containers are runnable instances of Docker images.
- Docker images are created using a Dockerfile, while Docker containers are created by running a Docker image.
- Docker images are designed to be immutable and can be shared and reused across multiple containers, while Docker containers are designed to be ephemeral and can be created and destroyed as needed.
- Docker images are typically smaller in size than Docker containers, as they don't contain any runtime state or data.

Overall, Docker images and Docker containers are two fundamental concepts in Docker that work together to provide a powerful containerization platform for building and deploying applications.

docker run -d --name=grafana -p 3001:3000 -v grafana_config:/etc/grafana -v grafana_data:/var/lib/grafana -v grafana_logs:/var/log/grafana grafana/grafana

docker run -d --name=grafana -p 3001:3000 -v grafana_config:/etc/grafana grafana/grafana