**Docker Basics**

**Docker, the Company**

1. **Docker, Inc.** is the name of the company behind the Docker project.
2. It started as a **Platform as a Service (PaaS)** company called **dotCloud**, but pivoted and rebranded to Docker, Inc. in **2013**.
3. Their main focus is **container technologies**.

### ****Docker, the Technology****

1. Docker is an **open-source technology** that Docker, Inc. developed.
2. It's designed to make it easier for developers to **create**, **deploy**, and **run applications** by using containers.
3. **Containers package an application and its dependencies together**, making the application easily runnable on any system that has Docker installed.
4. We can install Docker on any O.S but the Docker engine runs natively on Linux distribution.
5. Docker is written in the ‘GO’ language.

### ****Containerization Before Docker****

1. Although the concept of containerization wasn't new, **Docker popularized it** with a simpler, more efficient approach.
2. Previous technologies like **Linux Containers (LXC)** laid the groundwork, but **Docker made containers more accessible and easier to use**.

which problems solves by docker ?

### ✅ 1. ****“It works on my machine” Problem****

* **Issue**: Code runs on one machine but fails on another due to different environments.
* **Docker Solution**: Docker containers package the application with all its dependencies, ensuring consistent behavior across environments (development, testing, production).

### ✅ 2. ****Environment Configuration****

* **Issue**: Setting up development environments is time-consuming and error-prone.
* **Docker Solution**: Docker allows you to define the environment in a single Dockerfile, making setup easy and replicable.

### ✅ 3. ****Dependency Management****

* **Issue**: Conflicts due to different versions of libraries, runtimes, or tools.
* **Docker Solution**: Containers include their own dependencies, avoiding conflicts with other applications or host OS.

### ✅ 4. ****Portability****

* **Issue**: Applications behave differently on different platforms.
* **Docker Solution**: Docker containers run the same way on Linux, Windows, macOS, or cloud, making apps truly portable.

### ✅ 5. ****Resource Isolation****

* **Issue**: Running multiple apps on the same system causes interference.
* **Docker Solution**: Containers are isolated from each other and the host system, preventing resource conflicts.

### ✅ 6. ****Scalability & Microservices****

* **Issue**: Monolithic apps are hard to scale and maintain.
* **Docker Solution**: Docker encourages microservices architecture—breaking apps into smaller services that can be scaled independently.

### ✅ 7. ****Faster Development and Deployment****

* **Issue**: Setting up environments, testing, and deployment takes time.
* **Docker Solution**: Containers can be built and deployed quickly, speeding up CI/CD pipelines.

### ✅ 8. ****Version Control for Environments****

* **Issue**: No way to track environment changes like code.
* **Docker Solution**: Dockerfiles and images act like version-controlled snapshots of environments.

### ✅ 9. ****Easy Rollback****

* **Issue**: Rolling back to a previous working version can be hard.
* **Docker Solution**: You can maintain multiple versions of images and revert easily.

### ✅ 10. ****Security & Isolation****

* **Issue**: Applications may expose the host OS to vulnerabilities.
* **Docker Solution**: Containers run in isolated environments, reducing the attack surface.

### ✅ 11. ****Simplified CI/CD****

* **Issue**: Different CI/CD environments can cause failures.
* **Docker Solution**: Docker integrates seamlessly with CI/CD tools (like Jenkins, GitHub Actions, GitLab CI) for consistent builds and tests.

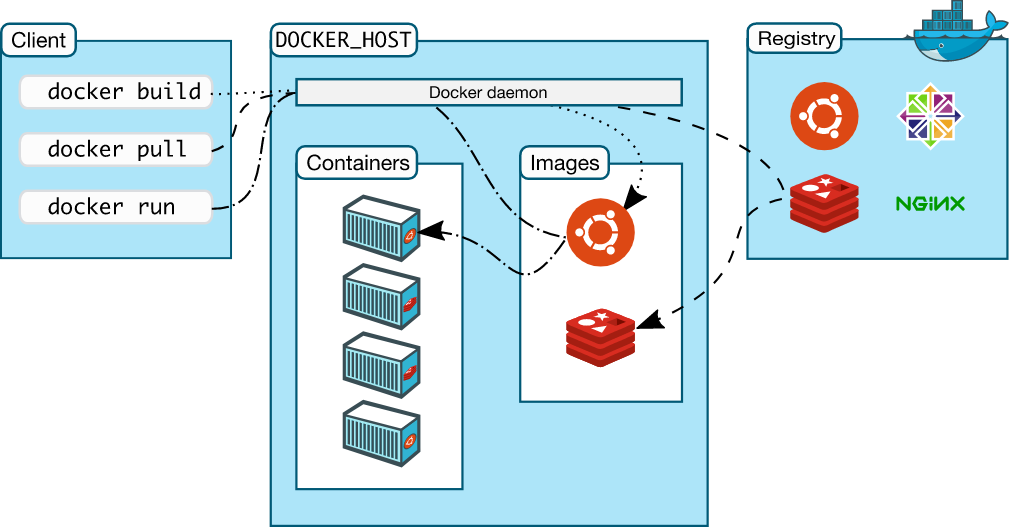
### ✅ 12. ****Infrastructure as Code****

* **Issue**: Manual infrastructure setup is error-prone.
* **Docker Solution**: Docker Compose allows infrastructure setup using YAML files, making it automated and reproducible.

### 🚫 ****Disadvantages of Docker (Short Notes)****

1. **No GUI Support**  
   Not suitable for applications with rich graphical interfaces.
2. **Hard to Manage Many Containers**  
   Becomes complex without tools like Kubernetes.
3. **Limited Cross-Platform**  
   Windows containers don’t run on Linux and vice versa.
4. **Needs VM on Windows/macOS**  
   Docker runs slower because it uses a VM on non-Linux systems.
5. **No Built-in Backup**  
   Data recovery and backup must be done manually.
6. **Not a Full OS**  
   Can’t replace VMs if different OS kernels are needed.
7. **Security Risks**  
   Containers share the same OS kernel, increasing risk if misconfigured.
8. **Persistent Storage is Tricky**  
   Data can be lost if volumes are not properly managed.

Architecture of Docker



Docker Host

Docker Host=a machine in which we are installing Docker daemon it can be =Laptop

=Server

=Vm in your data center or =a cloud service provider

Docker engine/Daemon:How you can convert your normal systems into the host.

runs on the Host O.S. It is responsible for running containers, and managing docker services. Docker daemons can communicate with other daemons.

Docker CLI: It same in all os Windows or linux (same commands in all OS)

Docker Registries:hub.docker.com

Docker registry manages and stores the docker images. There are two types of registries in the docker. 1) Public Registry → Public registry also called Docker Hub. Docker Images 2) Private Registry → It is used to share images within the enterprise.

Docker images

Docker images are the read-only binary templates used to create docker containers. or, a single file with all dependencies and configurations required to run a program. Train Train Shubham With With Shubham → Ways to create an images 1. Take an image from Docker Hub. 2.Create an image from Docker File. 3.Create an image from existing docker containers.