


























Docker

<input checked="" type="checkbox"/> Favorite	<input type="checkbox"/>
<input checked="" type="checkbox"/> Archived	<input type="checkbox"/>
<input checked="" type="checkbox"/> Fleeting	<input type="checkbox"/>
↗ Area/Resource	
↗ Project	
▼ Type	
 Review Date	
 Image	
 URL	
🕒 Created	@February 6, 2023 11:28 AM
🕒 Updated	@February 6, 2023 3:51 PM
🔍 Root Area	
🔍 Project Area	
Σ Updated (short)	02/06/2023
↗ Pulls	
🔍 Resource Pulls	
🔍 Project Archived	
Σ URL Base	
Σ 🔍 Recipe Divider	   RECIPE BOOK PROPERTIES   
☰ 🔍 Recipe Tags	
Σ  Book Divider	   BOOK TRACKER PROPERTIES   
☰  Author	
  Date Started	

  Date Finished	
  Book Status	
  Rating	

Docker



The main idea behind docker is to provide a standardized and isolated environment for applications to run, which makes it easier to develop test and deploy applications across different environments. Meaning that the same app can run across different infrastructures without compatibility issues.

We use docker for several reasons:

- **Reproducibility:** Docker containers provide a consistent and predictable environment.
- **Scalability:** Docker containers can easily scale horizontally
- **Portability:** Docker containers can be moved from one host to another without the need to worry about dependencies and configuration
- **Isolation:** Docker containers provide isolation between different applications, which can prevent conflicts and improve security.



Reproducibility: Refers to the ability of a system to consistently produce the same results when run in the same conditions. In terms of ED it means we can run the same pipelines multiple times and expect the same results. If we can create a pipeline with a certain set of dependencies, libraries and configurations, we can package all the components into a Docker container. This container can be run on different systems and environments and the pipeline will produce the same result each time. This helps eliminate inconsistencies and variability that might arise from differences in hardware, software and configuration.



Isolation: Refers to the separation of one entity from others, such as apps, processes or systems, in order to prevent interference and maintain independence. In the context of Docker, isolation means that each container runs in its own isolated environment, separate from the host operating system and other containers.

How does Docker work



Docker works by creating containers, which are lightweight, standalone and executable packages of software that include everything needed to run the application, including the code, runtime, system tools, libraries and settings. The containers run on a host operating system, and communicate with each other and the host through a Docker engine. The Docker engine is responsible for managing the containers, including starting, stopping and monitoring them, and providing an interface for developers to interact with the containers.

1. A developer will create a Docker image, which is a blueprint for the container
 - a. The image is built from a Dockerfile, a script which specifies the base image to use, the dependencies to install and the configuration to set.
2. The Docker engine uses the image to create a container, which runs the application
 - a. Docker containers are isolated from each other and the host system, meaning that they do not depend on the host system's variables, and can be portable and moved to other environments and produce the same results

Resources

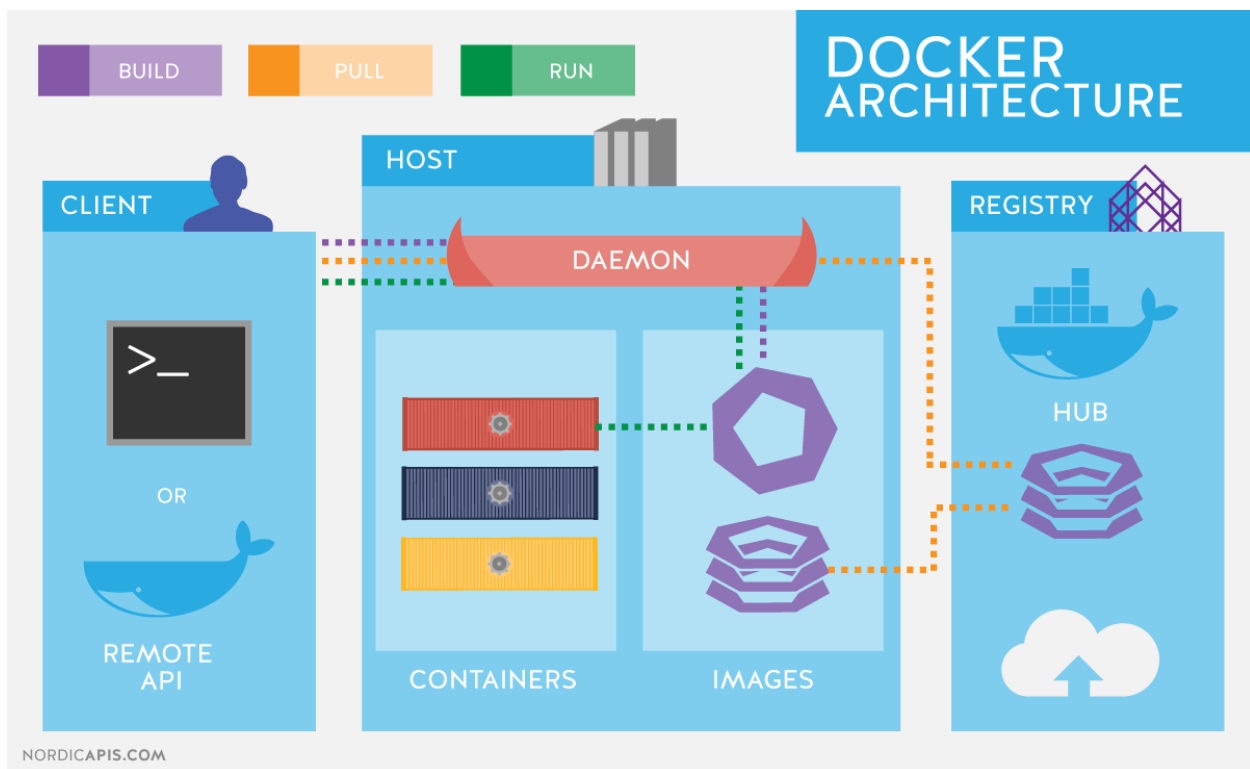


Docker containers have their own isolated resources, such as CPU, memory, network and storage. These resources are allocated to containers by Docker and can be limited or unrestricted based on the requirements of the individual containers. This isolation allows containers to run independently.

Docker can allocate resources two ways:

1. Manually, the user will specify how many resources are needed based on the use of flags and options in the docker run command.
2. Dynamic resource allocation, which adjusts the resources allocated to containers in real-time based on the usage patterns of the containers

Architecture



Docker architecture consists of multiple components that work together to create a platform for creating, deploying and running containers.

Docker client: The CLI that allows users to interact with the docker daemon. Users can use the client to build images, run and manage containers.

Docker Daemon: The core component of the architecture and runs on the host machine. The Docker daemon is responsible for building images, running containers and managing the containers.

Docker registry: The place where images are stored and can be retrieved by the docker client. There are public and private registries such as Docker Hub, or the Google container Registry, which allows users to store images privately and interact with other GCP services.

Docker Engine: The software that runs on the host machine and provides an interface between the Docker Daemon and the client. The docker engine includes additional tools such as the Docker CLI and REST API

Docker Images: The building blocks of containers. An image is a pre-configured environment that includes the necessary files, libraries and settings required to run an app

Docker containers: Containers are instances of Docker images that run as isolated processes on the host machine. Each container has it's own file system networking and resources, and can be started, stopped and managed independently



Daemon: A background process in computer systems that independently of a user's session. Daemons are often used to perform system-level tasks such as managing network connections, scheduling tasks, and monitoring the status of other processes. They run continuously in the background, even when there is no user session active, and are typically started automatically at system boot. In the context of Docker, the daemon is the core component that runs on the host machine and responsible for building images.

Dockerfile



The Dockerfile details how Docker should build the image and includes the commands that must be run when the image is being built on the underlying infrastructure.

```
FROM python:3.9 -- base image we want to use

RUN pip install pandas -- What to run when you run the docker image

ENTRYPOINT ['BASH']
```

Environmental variables



Env variables in docker dictate how the environment in Docker will be set up.

Volumes



Volumes are a way to store data outside of a container's filesystem. Meaning that even if the container is deleted or recreated, the data in the volume remains persistently stored on the host file system. Mapping folders on host machine to be used in Docker is called **mounting**.

```
VOLUME /data --will create this record on the underlying file system.
```