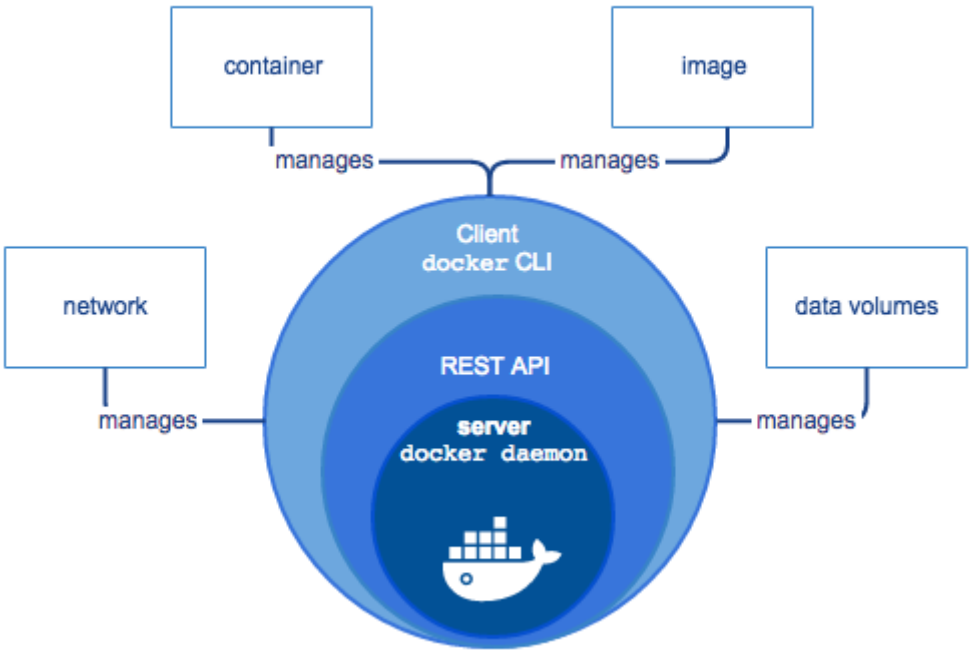
#004 Docker Architecture

Introduction

this is part 4 from the journey it's a long journey(360 day) so go please check previous parts , and if you need to walk in the journey with me please make sure to follow because I may post more than once in 1 Day but surely I will post daily at least one 😍.

And I will cover lot of tools as we move on.

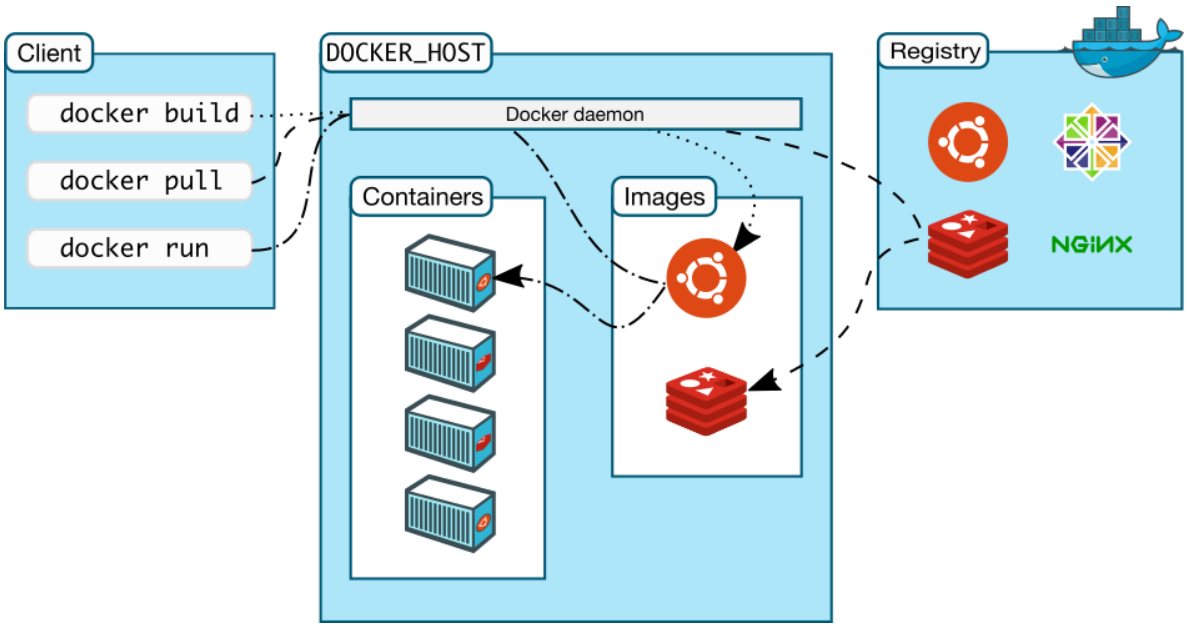
Docker Engine



Docker Engine is a client-server application , he use an REST API to communicate between Server and client , same as back-end communicate with front-end , you send an API request to the server from the client and the server get back a response .

Since it's a server-client and REST API based it's mean you can have a docker daemon running on server , and client running on local machine . So you can remote control the server !

A bigger picture



The Docker daemon

The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (`docker`) is the primary way that many Docker users interact with Docker.

When you type a command the client send it to the API and then API interact with the server

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

go to Postgres Docker Hub , this is an example for postgres database you can run database with 1 command!

Images

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the `ubuntu` image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

reference