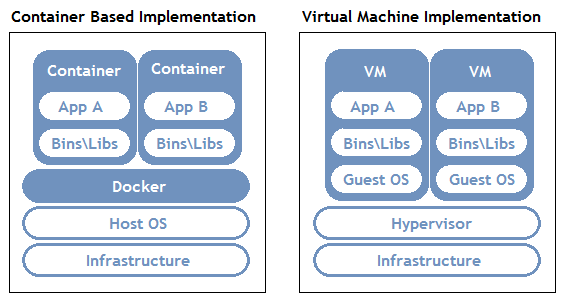
What is Docker?

You probably heard of the statement ‘Write once, run anywhere’, a catchphrase that SUN Microsystems came out with to capture Java’s ubiquitous nature. This is a great paradigm except that, if you have a java application, for example, in order to run it anywhere you need platform-specific implementations of the Java Virtual Machine. On the other end of the ‘run anywhere’ spectrum, we have Virtual Machines. This approach, while versatile, comes at the cost of large image sizes, high IO overhead, and maintenance costs.   
What if there is something that is light in terms of storage, abstracted enough to be run anywhere, and independent of the language used for development?

This is where [Docker](https://www.aquasec.com/wiki/display/containers/Docker+Containers) comes in! Docker is a technology that allows you to incorporate and store your code and its dependencies into a neat little package - an image. This image can then be used to spawn an instance of your application - a container. The fundamental difference between containers and Virtual Machines is that containers don’t contain a hardware hypervisor.



This approach takes care of several issues:

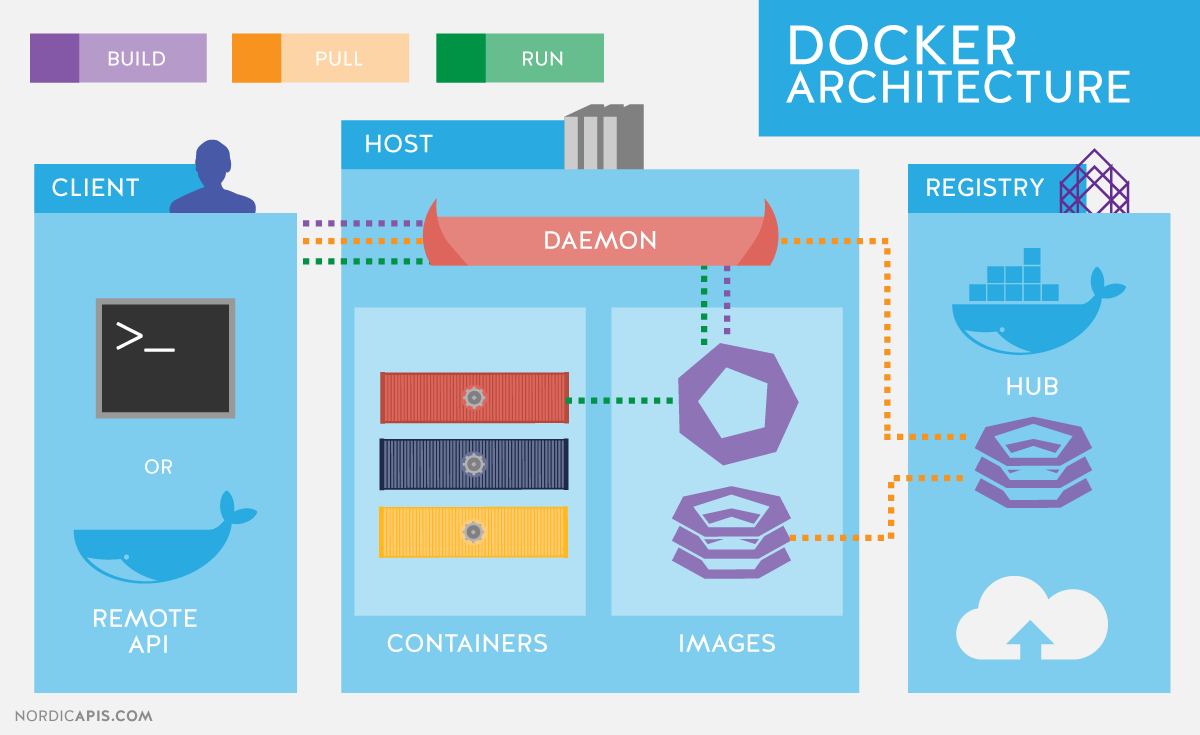
* No platform specific, IDE, or programming language restrictions.
* Small image sizes, making it easier to ship and store.
* No compatibility issues relating to the dependencies/versions/setup.
* Quick and easy application instance deployment.
* Applications and their resources are isolated, leading to better modularity and security.

Docker Architecture

To allow for an application to be self-contained the Docker approach moves up the abstraction of resources from the hardware level to the Operating System level.

To further understand Docker, let us look at its architecture. It uses a client-server model and comprises of the following components:

* **Docker daemon:** The daemon is responsible for all container related actions and receives commands via the CLI or the REST API.
* **Docker Client:** A Docker client is how users interact with Docker. The Docker client can reside on the same host as the daemon or a remote host.
* **Docker Objects:** Objects are used to assemble an application. Apart from networks, volumes, services, and other objects the two main requisite objects are:
* **Images:** The read-only template used to build containers. Images are used to store and ship applications.
* **Containers:** Containers are encapsulated environments in which applications are run. A container is defined by the image and configuration options. At a lower level, you have containerd, which is a core container runtime that initiates, and supervises container performance.
* **Docker Registries:** Registries are locations from where we store and download (or “pull”) images.



BASIC DOCKER OPERATIONS...

* [Docker Image Repositories](https://www.aquasec.com/wiki/display/containers/Docker+Image+Repositories) — A Docker Image repository is a place where Docker Images are actually stored, compared to the image registry which is a collection of pointers to this images. This page gathers resources about public repositories like the Docker hub and private repositories and how to set up and manage Docker repositories.
* [Working With Dockerfiles](https://www.aquasec.com/wiki/display/containers/Working+With+Dockerfiles) — The Dockerfile is essentially the build instructions to build the Docker image. The advantage of a Dockerfile over just storing the binary image is that the automatic builds will ensure you have the latest version available. This page gathers resources about working with Dockerfiles including best practices, Dockerfile commands, how to create Docker images with a Dockerfile and more.
* [Running Docker Containers](https://www.aquasec.com/wiki/display/containers/Running+Docker+Containers) — All docker containers run one main process. After that process is complete the container stops running. This page gathers resources about how to run docker containers on different operating systems, including useful docker commands.
* [Working With Docker Hub](https://www.aquasec.com/wiki/display/containers/Working+With+Docker+Hub) — Docker Hub is a cloud-based repository in which Docker users and partners create, test, store and distribute container images. Through Docker Hub, a user can access public, open source image repositories, as well as use a space to create their own private repositories, automated build functions, and work groups. This page gathers resources about Docker Hub and how to push and pull container images to and from Docker Hub.
* [Docker Container Management](https://www.aquasec.com/wiki/display/containers/Docker+Container+Management) — The true power of Docker container technology lies in its ability to perform complex tasks with minimal resources. If not managed properly they will bloat, bogging down the environment and reducing the capabilities they were designed to deliver. This page gathers resources about how to effectively manage Docker, how to pick the right management tool including a list of recomended tools.
* [Storing Data Within Containers](https://www.aquasec.com/wiki/display/containers/Storing+Data+Within+Containers) — It is possible to store data within the writable layer of a container. Docker offers three different ways to mount data into a container from the Docker host: volumes, bind mounts, or tmpfs volumes. This page gathers resources about various to store data with containers, the downsides like the persistent storage and information on how to manage data in Docker.

What is a Docker Image?

A [Docker](https://www.aquasec.com/wiki/display/containers/Docker+Containers)image is a snapshot, or template, from which new containers can be started. It’s a representation of a filesystem plus libraries for a given OS. A new image can be created by executing a set of commands contained in a Dockerfile (it’s also possible but not recommended to take a snapshot from a running container). For example, this [Dockerfile](https://www.aquasec.com/wiki/display/containers/Working+With+Dockerfiles) would take a base Ubuntu 16.06 image and install mongoDB, resulting in a new image:

|  |
| --- |
| FROM ubuntu:16.04  RUN apt-get install -y mongodb-10gen |

From a physical perspective, an image is composed of a set of read-only layers. Image layers function as follows:

* Each image layer is the outcome of one command in the image’s Dockerfile—an image is then a compressed (tar) file containing the series of layers.
* Each additional image layer only includes the set of differences from the previous layer (try running  docker history  for a given image to list all its layers and what created them).

Running Images as Containers

Images and containers are not the same—a container is a running instance of an image. A single image can be used to start any number of containers. Images are read-only, while containers can be modified, Also, changes to a container will be lost once it gets removed, unless changes are committed into a new image.

Follow these steps to run an image as container:

* First, note that you can run containers specifying either the image name or image ID (reference).
* Run the  docker images  command to view the images you have pulled locally or, alternatively, [explore the Docker Hub](https://hub.docker.com/explore/) repositories for the image you want to run the container from.

Once you know the name or ID of the image, you can start a container with the   docker run   command. For example, to download the Ubuntu 16.04 image (if not available locally yet), start a container and run a bash shell:

|  |
| --- |
| docker run -it ubuntu:16.04 /bin/bash |