**DESCRIPTION**

**Docker Architecture**

**References**

* [Docker Architecture from the Docker Getting Started Guide](https://docs.docker.com/get-started/overview/#docker-architecture)

**Overview**

Docker adheres to a Client-Server Architecture. The client allows you to run commands and interact with the docker objects managed by the docker daemon, which is in effect the server.

This architecture, consequently, translates into these *primary* components that allow you to run and interact with Dockerized applications (applications that have been packaged up and run in Docker containers.)

* [Docker Daemon](https://app.revature.com/admin-v2/)
* [Rest API](https://app.revature.com/admin-v2/)
* [Docker CLI client](https://app.revature.com/admin-v2/)
* [Docker Registries](https://app.revature.com/admin-v2/)
* [Docker objects](https://app.revature.com/admin-v2/)

As you can see from the diagram the *client* includes the commands that manage docker containers and images. The *docker host* is the computer on which the *docker daemon* runs, and the *docker daemon* is responsible for managing the *docker objects* i.e. containers and images. The *registry* is an additional/optional component that provides a centralized place to share docker images. Finally, while the *REST API* is not labeled in the diagram it's effectively the lines connecting the *client* and *docker daemon*.

**Docker CLI (Command Line Interface) client**

The command line interface is what Docker developers typically use for interactions with the Docker daemon.

It's prefaced by the docker command.

The CLI client may be on the same host machine as the Docker daemon with which it communicates, but it doesn't have to be.

All the commands outlined in the [docker workflow module](https://app.revature.com/admin-v2/) notes are effectively part of the client.

[**Docker Daemon**](https://app.revature.com/admin-v2/)

The docker daemon is the long running process on the docker host that does all the heavy lifting of managing Docker objects- containers, images, etc. The docker daemon is at the core of running dockerized applications.

**Rest API**

These are the underlying commands used by the CLI and other applications to interact with the Docker Daemon.

**Docker registries**

Docker registries provide a centralized place to store images, allowing you to easily share images between docker hosts. The Docker Hub is a public registry managed by Docker that docker hosts can pull images from and push images to by default.

For more info see [these notes.](https://app.revature.com/admin-v2/)

**Docker objects**

Docker objects are the building blocks that are managed by the docker daemon. The most fundamental docker objects are images and containers. Images are the templates that outline all dependencies for a particular container and it's primary process. Meanwhile the container is the runnable instance of a set of processes and their dependencies.

For more info on some of the key docker objects see the notes on [docker containers](https://app.revature.com/admin-v2/), [docker images](https://app.revature.com/admin-v2/), and [docker volumes](https://app.revature.com/admin-v2/).

**Typical Flow**

1. Using CLI commands (in your command line, Terminal, Command Prompt etc.) such as docker build ... or docker pull ..., acquire an image. This image lays out everything needed to create and run a container.
2. Behind the scenes the Docker daemon either pulls your image from a registry(repository of images) or it creates the image. (Depending on the command you gave.)
3. Then you use the CLI again with a command such as docker run .... (Note if you don't already have the image you were trying to run then the command will include that first step of pulling the image implicitly.) If you didn't already do this during the installation process, try the following command.

docker run hello-world

1. Thus, the CLI once again instructs the daemon. This time it tells the daemon to spin up a container from the image.
   * The application or set of processes in the container are tied to its life cycle and have started with its instantiation.
2. Finally, it's just a matter of using additional CLI commands to manage the now running container, unless the primary process of the container has already finished. In that case the container will exit on it's own.