Docker Images, Containers, Commands, and Dockerfiles

Figure 1: Docker Logo

From the Docker site …

> Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

What Is A Docker Image

Again, from the Docker site.

> A Docker container image is a lightweight, standalone, **executable** package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

The main point is that images once started in Docker engine and running in a Docker container will always run the same no matter what hardware they are run on. The container runs it's own file system, and this provides the greatest source of isolation to ensure this constant operation.

AND if you run multiple containers, they are all isolated from one another UNLESS you design a way for them to interact.

What Is A Docker Container

Also from the Docker Site

> A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

and

> Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine.

Docker Commands

Some common docker commands that you'd want to keep handy are:

Command	Explanation
`docker -version`	reports the currently installed version of docker
`docker pull <image_name>`	pulls image_name from docker repository such as hub.docker.com

Command	Explanation
`docker build -t image_tagname dockerfile_dir`	build a docker container named `tagname` with `-t image_tagname`using file named `Dockerfile` in directory `dockerfile_dir`Example: `docker build -t my_container_name .`, where `.` specifies the current directoryThe `image_tagname` is handy for having slightly different versions using the same `Dockerfile`
`docker ps`	returns a list of the running docker containers. Add `-a` to show all running and non-running containers
`docker run -it <container_name> first_command`	run container_name, interactively, and run first_command in itNOTE: You do NOT need to use `-it` or `first_command`. You could just run `docker run <container_name>`
`docker exec -it <container_name> first_command`	much like the run previous run command, but used to access an already running container
`docker image ls`	returns a list of all Docker images on your computer
`docker stop <container_id>`	stops a running container specified by the container's id
`docker kill <container_id>`	kills a running container when you don't want to wait for a typical shutdown process
`docker commit <container_id> <user_name/image_name>`	creates a new image of an edited container on your local computer
`docker login`	login to your account on the docker hub repository; you can create a free account if you do not have one
`docker push <user_name/image_name>`	used to push an image of yours to your docker hub repository
`docker images`	lists all the locally stored docker images
`docker rm <container_id>`	used to delete **stopped** containers
`docker rmi <image_id>`	used to delete images from your local computer storage

See also Top 15 Docker Commands – Docker Commands Tutorial.

Dockerfiles

A Dockerfile contain instructions for building images.

From Dockerfile reference,

> Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using `docker build` users can create an automated build that executes several command-line instructions in succession.

Dockerfile Commands

The Dockerfile commands in the table below are very common and will serve most of your Dockerfile needs. There are more Dockerfile commands that these though.

Command	Explanation
`FROM docker_image_name`	creates an initial layer FROM an existing image
`WORKDIR directory_on_image`	changes the specified directory__on__image to be the working directory
`COPY client_file(s) image_file(s)`	COPies files from the client that docker is running on into the image

| `ADD source image_destination` | Copy files 3 ways:

from client storage into image

moving tarball from client and extracting in image

from URL into image

| | `RUN command` | runs Linux commands on the image's command line | the Linux commands are run | | `ENV environment_variable_name=environment_variable_value` | | | EXPOSE port__number | tells Docker the port our container will start on | | USER username | specifies the user that should run the application | | `ENTRYPOINT command_in_image [options]` | command(s) that will always run when this image launches into a container | | `CMD command_in_image [options]` | command(s) that are passed to ENTRYPOINT unless overridden during `docker run` on the command line |

For understanding ENTRYPOINT and CMD better and how they relate, I like THIS StackOverflow answer.