Random Notes about Docker

Konstantin Burlachenko,

12-APR-2020 with update from 13-MAY-2020

How people live before and after Docker?

DevOps – it's probably people who support deployment of the products for customers.

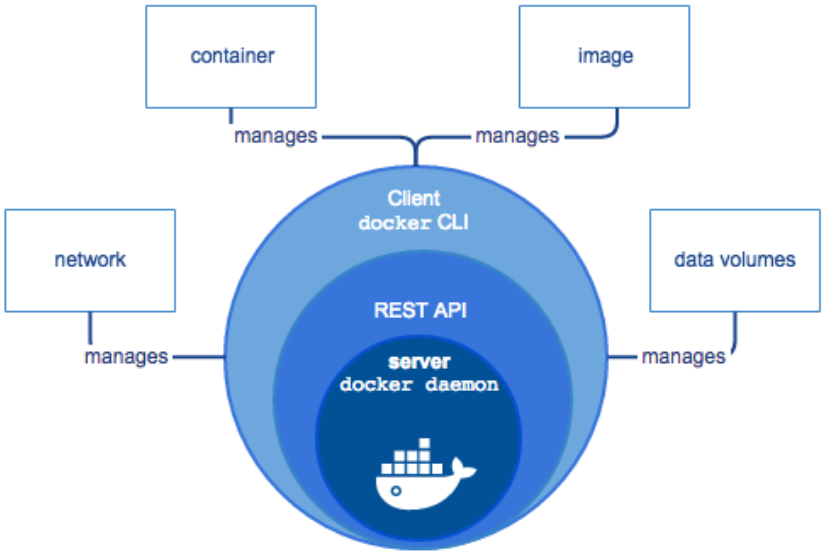
Before Docker: Developer provides guidelines and developers help dev-ops worked together in question about deployment.

After Docker: DevOps and Develop use Docker Images.

What is Docker?

Docker is a platform for developers and sysadmins to **build, run, and share** applications with containers. The use of containers to deploy applications is called *containerization*.

Main Docker concepts



The Docker daemon - The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.

The Docker client - The Docker client (docker) is the primary way that many Docker users interact with Docker. CLI interface description is mentioned here:

https://docs.docker.com/engine/reference/commandline/cli/

Docker registries - A Docker *registry* stores Docker images (will be described in next section). Docker-Hub – common repository with docker-images. For use them install docker in your local machine.

Docker Objects

Docker operate on several type of objects: images, containers, networks, volumes, plugins.

Images

Briefly: package or template used to create one or more containers.

More Details:

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. An image includes everything needed to run an application - the code or binary, runtimes, dependencies, and any other filesystem objects required. But image has not any runtime behavior.

Containers

Briefly: Container – instantiation of Images. In case of having several one each one is a separate instance.

Details: A container is a runnable instance of an image. You can:

- Create, start, stop, move, or delete a container using the Docker API or CLI
- Connect a container to one or more networks
- Attach storage to it
- Create a new image based on its current state of the container

Fundamentally, a container is nothing but a running process, with some added encapsulation features applied to it in order to keep it isolated from the host and from other containers. Each container interacts with its own private filesystem.

To really implement it *Docker* leverage into several Linux features:

1. Docker uses a technology called **Linux namespaces** to provide the isolated workspace called the *container*. When you run a container, Docker creates a set of *namespaces* for that container.
2. Docker Engine on Linux also relies on another technology called *control groups* (cgroups). A cgroup limits an application to a specific set of resources.

How to build your own images?

Todo it you should 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 something which is called “a layer” in the image.

Dockerfile commands are available here https://docs.docker.com/engine/reference/builder/

Some Dockerfile Commands

FROM – start the pre-existing image as a base.

WORKDIR /usr/src/app – specify that all subsequent actions should be taken from the Directory /usr/src/app *in your image filesystem* (never the host's filesystem).

COPY – copy files from your host to the present location

RUN – run command inside your image filesystem

CMD - directive specifying some metadata in your image that describes how to run a container based on this image.

Some Docker Commands

Command	Description
docker run -i -t ubuntu /bin/bash	1. If you do not have the ubuntu image locally, Docker pulls it from your configured registry, as though you had run **docker pull ubuntu** 2. Docker creates a new container, as though you had run a **docker container create** 3. Docker allocates a read-write filesystem to the container 4. Docker creates a network interface to connect the container to the default network 5. Docker starts the container and executes /bin/bash 6. Container is running interactively and attached to your terminal due to –i and –t flags 7. When you type exit to terminate the /bin/bash command, the container stops but is not removed. You can start it. *docker container start -i <CONTAINER_ID>* You can remove it. *docker container rm <CONTAINER_ID>* Run a command in a new container description https://docs.docker.com/engine/reference/commandline/run/

docker run –it -p 8889:8888 -v /home/dir:/ctr_dir image_name:tag	-p asks container network port 8888 maps into host's 8889 port. -v mounts local folder to folder into containers
docker run hello-world	Test that your installation works
docker image ls	List of docker images in filesystem Manage images in a container https://docs.docker.com/engine/reference/commandline/image/
docker pull ubuntu	Pull image from repository
docker container ls --all	List of containers including exiting and working https://docs.docker.com/engine/reference/commandline/container/ Manage containers
docker cp foo.txt mycontainer:/foo.txt	Copy from local filesystem into docker container filesystem
docker cp mycontainer:/foo.txt foo.txt	Copy from docker container filesystem into local filesystem
docker ps	Get list of running container name and containers id
`docker ps -a`	Get list of all container name and containers id
`docker start|stop|restart my_ctr`	start, stop or restart container

`docker attach my_ctr`	This allows you to view its ongoing output or to control it interactively, as though the commands were running directly in your terminal.
`docker exec [OPTIONS] CONTAINER COMMAND [ARG...]`	Run a command in a running container docker exec –it ctr_name /bin/bash

When you use the docker push command, your image is pushed to your configured registry.