Speeding up and automating your development routine with Docker

Andrey Hristov
CTO, DNH Soft

$ whoami

MSc in Computer Tecnologies (TU-Sofia) and Software Engineering (HFT Stuttgart)

Professional developer since year 2000

PHP core developer since 2002

Spent 11 years working at MySQL, SUN and Oracle improving the MySQL client and server side

During the past 2 years spent as a technical team leader and lately moved to a CTO position at DNH Soft

What to expect from this talk?

Linux POV

Description of technologies related to containers

Overview of Docker

L1v3 D3m0

To begin with, what is Docker?

Docker Inc. is a company, previously dotCloud

However in the past 5 years the name meant containers

Some people say dockerize when they mean containerize

Containers were not invented by Docker Inc. The company made them available to the masses.

Then, what is a container?

Containerization is **OS environment virtualization**

It feels like a VM but ain't one. Some people call them **lightweight** VMs.

“One kernel to rule them all” compared to “one hypervisor to rule them all”.

Can't boot a different OS or kernel. Can't load other modules.

Can boot different distro, however.

Examples of previous/other works : Solaris Zones, FreeBSD Jails

Diving in deep : Linux Namespaces

Linux Namespaces = **isolation**

Cgroup - Cgroup root directory

IPC - System V IPC, POSIX message queues

Network - Network devices, stacks, ports, etc.

Mount - Mount points

PID - Process IDs

User - User and group IDs

UTS - Hostname and NIS domain names

Diving in deeper: Cgroups

“Control groups, usually referred to as cgroups, are a Linux kernel feature which allow **processes** to be organized into hierarchical groups whose usage of various types of **resources** can then be **limited** and **monitored**.”

A cgroup is a collection of processes that are bound to a **set of limits** or parameters defined via the cgroup filesystem

Cgroups are found under /sys/fs/cgroup

Cgroups have their **own namespaces**

Cgroups offer **resource metering and limiting** of memory, CPU, block I/O, network

Who is running them?

Containers are managed by **runtimes**

LXC/LXD - LXD, written in Go, uses LXC

rkt - App Container compliant, deprecated, by CoreOS, now Red Hat. Natively ACI, but supports also Docker and OCI images.

runC - OCI compliant implementation in Golang by Docker Inc., a spin off from Docker Engine since Docker 1.11

containerD - works with runC for the high level details, while **runC** is low level

railcar - OCI compliant implementation in Rust by Oracle

OCI has two specs, released in July'17 : **Image** and **Runtime**

CRI-O, implementation of the Kubernetes (1.5+) Container Runtime Interface (CRI) using OCI compatible runtimes.

But there is more!

Container are managed at higher level by **orchestrators.**

Docker Compose (single host only, part of Docker Engine)

Docker Swarm (part of Docker Engine)

Marathon on Apache Mesos

Cattle, obsoleted, by Rancher. Rancher 2.0 runs k8s

Kubernetes (k8s). Recently won the **Orchestrator wars**.

Docker, where is my data?

aufs (/var/lib/docker/aufs), superseed by

overlayfs, shipped with Linux Kernel 4.0

cat /proc/filesystems to see what you have

In short, what’s in for me?

Containers are **lightweight**, or at least lighter than VMs, both in run-time resources usage and size

Containers are **immutable**

Containers can be even **read-only**

Every container contains **all** needed **dependencies** and doesn't need anything else

Implications:

Dep hell is gone. DLL hell memories resurface?

XAMPP is dead

Linux distro software choice is dead

Less software installed means **less exploit surface**

Container images hosting

Docker Inc. runs **Docker Hub**

Library of public images

Supports automated builds triggered on a **commit** in Github / BitBucket.

Storage for your images

- **free** of charge for you **public** ones
- has a **cost** for you **private** images

Alternatives are:

- Host a repo in a container on own VPS
- **Amazon** Elastic **Container Registry**, you need AWS SDK
- **Google Container Registry**, you need Google Cloud SDK

Docker Flavors

Supported OS for Docker CE:

- Linux (x86-64, ARM, ARM64, ppc64le, s390x(
- MacOS, comes bundled with k8s
- Windows, comes bundled with k8s
- AWS
- Azure

Supported Platforms for Docker EE

- CentOS (x86-64)
- OL (x86-64)
- RHEL, SUSE Linux ES, Ubuntu (x86-64 / ppc64le / s390x)
- MS Windows Server 2016 (x86-64)
- AWS
- Azure
- IBM Cloud

Docker Compose

Originally known as Fig

“Cluster” **configuration** is stored in an **YAML file**

The file is by default **./docker-compose.yml**

Features are constantly added, thus there are **many compose file versions**. Latest is 3.6 as of 18.02.

First line in the file states **minimum version**

The file is split in **3 main sections**, since 2.0 : **services**, **networks**, **volumes**

Docker Compose Entities

services - The containers = **instances** of images.

With Swarm you can have multiple instance per service - scaling up and down.

volumes / mounts - **Persistently stored data**.

Otherwise data is gone when the container get removed.

Mounts import data from the host and are shareable

Volumes are BLOBs and are shareable too

Volumes are abstracted thru plugins

networks - The actual **glue** between the services

DC creates a **default network**, if are lazy to not create one.

This network is called <projectName>_default

<projectName> is derived from **CWD**, pass **-p** to docker-compose for smth else.

Networks can be seen by other projects and they are namespaced by project name.

Network *frontend* in P1 can be attached in project P2 as external network under the foreign name P1_frontend.

Docker CLI

docker pull image[:tag|@digest], aka docker image pull

- tag can is a version, digest is a sha256 digest (like git commit hash)

docker push image:tag, aka docker image push

docker rmi image:tag, aka docker image rm

docker build, aka **docker image build**

- use --no-cache to rebuild from scratch
- use -t image:tag to add name and version

docker images, aka docker image ls

docker image inspect

docker image inspect <imageid> | jq -r '.[].RootFS'

More Docker CLI

docker run, aka docker container run

docker exec, aka docker container exec

docker rm, aka docker container rm

docker ps, aka docker container ls

docker stop, aka docker container stop (SIGTERM)

docker kill, aka docker container kill (SIGKILL)

docker kill `docker ps -q` to kill'em all

docker inspect

- inspects networks, containers, images
- gives you tons of info in JSON format. Use jq to process it.

docker network ls

docker network rm

docker network prune

docker system prune

Live Demo

Q&A / Resources

Anatomy of a container: https://bit.ly/2v0EEGj

https://github.com/andreyhristov/bws2018-docker

https://coreos.com/rkt/docs/latest/rkt-vs-other-projects.html

https://docs.docker.com/install/linux/docker-ce/ubuntu/

https://docs.docker.com/compose/install/

https://docs.docker.com/compose/compose-file/

https://docker-software-inc.scoop.it/t/docker-by-docker

https://dashtainer.com/

https://landscape.cncf.io/

https://traefik.io/

https://leanpub.com/the-devops-2-toolkit

https://leanpub.com/the-devops-2-2-toolkit

https://leanpub.com/the-devops-2-3-toolkit

https://thenewstack.io/