Docker Introduction

Ondrej Sika
ondrej@ondrejsika.com
@ondrejsika

https://sika.link/docker

Agenda

- What is a Docker?
- Docker Installation
- Basic Commands
- Containers
- Images
- Docker Registry
- Volumes

What is Docker?

What is Docker?

Docker is an open-source project that automates the deployment of applications inside software containers. ...

Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server.

Containers vs virtualization

Virtualization

A VM is an abstraction of physical hardware. Each VM has a full server hardware stack from virtualized BIOS to virtualized network adapters, storage, and CPU.

That stack allows run any OS on your host but it takes some power.

Containers

Containers are abstraction in linux kernel, just proces, memory, network, … namespaces.

Containers run in same kernel as host - it is not possible use different OS or kernel version, but containers are much more faster than VMs.

Advantages

- Performance
- Management
- Containers distribution

Disadvantages

- Security
- One kernel / "Linux only"

Usage

- Almost everywhere
- Production, Testing, Development
- Better deployment process
- Separates running applications

Work with

Cluster managements

- Kubernetes
- Swarm

Installation

—

https://docs.docker.com/engine/installation/

https://ondrej-sika.cz/docker/instalace/ (CS)

Test the installation

docker run hello-world

...

Hello from Docker!

This message shows that your installation appears to be working correctly.

Basic Usage

Image and Container

An **image** is an inert, immutable, file that's essentially a snapshot of a **container**. **Images** are created with the build command, and they'll produce a **container** when started with run. Images are stored in a Docker registry..

Help

```
docker

docker help

docker help <command>

# eg. docker help run, docker help help
```

System wide info

```
docker info  # system vide information

docker system df  # docker disk usage

docker system prune  # cleanup unused data

docker version  # print version
```

Docker Images

```
docker image ls  # list all images

docker image ls -q  # quiet output, just IDs

docker image save …  # save image to file

docker image load …  # load image from file

docker image rm <image>  # remove image
```

Docker Run

```
docker run <image> [<command>]

# Eg.:

docker run hello-world

docker run debian date

docker run -ti debian
```

Containers

docker ps - list containers
docker start <container>
docker stop <container>
docker restart <container>
docker logs <container> - show STDOUT & STDERR
docker rm <container> - remove container

Common Docker Run Params

--name <name>
--rm - remove container after stop
-d - run in detached mode
-ti - map TTY a STDIN (for bash eg.)
-e <variable>=<value> - set ENV variable
-h <hostname> - set hostname
-u <user> - run command by specific user

List containers

docker ps - list running containers
docker ps -a - list all containers
docker ps -a -q - list IDs of all containers

Eg.:

```
docker rm -f $(docker ps -a -q)
```

Docker Exec

docker exec <container> <command>

-d - run in detached mode
-e <variable>=<value> - set ENV variable
-ti - map TTY a STDIN (for bash eg.)
-u <user> - run command by specific user

```
# Eg.:
docker run --name db -d postgres
docker exec -ti -u postgres db psql
```

Docker Logs

```
docker logs [-f] <container>

# Eg.:

docker logs my-debian

docker logs -f mydebian  # following
```

Docker Inspect

Get lots of information about container in JSON.

docker inspect <container>

Docker Volumes

Volumes are persistent data storage for containers.

Volumes can be shared between containers and data are written directly to host.

Volumes

```
docker run -ti -v /data debian

docker run -ti -v my-volume:/data debian

docker run -ti -v $(pwd)/my-data:/data
debian
```

Port Forwarding

Docker can forward specific port from container to host

Port forwarding

```
docker run -p <host port>:<cont. port> nginx


# eg.:

docker run -ti -p 8080:80 nginx
```

Own Images

Dockerfile

Dockerfile is preferred way to create **images**.

Dockerfile defines each layer of **image** by some command.

To make **image** use command **docker build**

Dockerfile

FROM <image> - define base image
MAINTAINER <maintainer> - set maintainers name & email
RUN <command> - run command and save as layer
COPY <local path> <image path> - copy file or directory to image layer
ADD <source> <image path> - instead of copy, archives added by add are extracted

Dockerfile

ENV <variable> <value> - set ENV variable
USER <user> - switch user
WORKDIR <path> - change working directory
VOLUME <path> - define volume
ENTRYPOINT <command> - executable
CMD <command> - parameters for entrypoint

.dockerignore

Ignore files for docker build process.

Similar to **.gitignore**

.dockerignore

```
# comment
*/temp**
*/*/temp*
temp?
```

Build Image from Dockerfile

```
docker build <path> -t <image>

docker build <path> -f <dockerfile> -t <image>

docker tag <source image> <target image>
```

Practice

app.py

```
import os
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Hello from %s" % os.environ.get('HOSTNAME')

if __name__ == "__main__":
    app.run(host='0.0.0.0', port='80')
```

requirements.txt

```
flask
```

Dockerfile

```
FROM python:3.7-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD [ "python", "app.py" ]
```

Build Arguments

Build Args

```
ARG FROM_IMAGE=debian:9
FROM $FROM_IMAGE
```

```
FROM debian
ARG $PYTHON_VERSION=3.7
RUN apt update && \
    apt install python==$PYTHON_VERSION
```

Build with Args

```
docker build \
  --build-arg FROM_IMAGE=python .

docker build .
```

```
docker build \
  --build-arg PYTHON_VERSION=3.6 .
```

Multi Stage Builds

Dockerfile for multistage builds

```
FROM java-jdk:... as build
RUN maven build

FROM java-jre:... as run
COPY --from=build /build/demo.jar .

FROM run as devel
RUN apt install ...
```

Build Image from Dockerfile

```
# By default, last stage is used

docker build -t <image> <path>


# Select output stage

docker build -t <image> --target <stage> <path>
```

Practice

app.go

```
package main
import "fmt"
import "net/http"

func index(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello World from Go!\n")
}

func main() {
    http.HandleFunc("/", index)
    fmt.Println("Server startded.")
    http.ListenAndServe(":80", nil)
}
```

Dockerfile

```
FROM golang as build
WORKDIR /build
COPY app.go .
ENV CGO_ENABLED=0
RUN go build -a -ldflags \
    '-extldflags "-static"' app.go

FROM scratch
COPY --from=build /build/app .
CMD ["/app"]
```

Docker Registry

What is Docker Registry?

A service responsible for hosting and distributing images.

The default registry is the **Docker Hub**.

GitLab contains Docker Registry from version 8.

What is Docker Hub?

Docker Hub is default public docker registry.

You can host unlimited free images.

Docker Hub is source of our base images.

Docker registry

docker login - Login to Docker Registry

docker logout - Logout from Docker Registry

docker pull <image> - download image from registry

docker push <image> - upload image to registry

Run own registry

```
docker run -d -p 5000:5000 \
    --restart=always --name registry \
    registry:2
```

More at: https://docs.docker.com/registry/deploying/

Networks

Default networks

```
docker run debian ip a

docker run --net none debian ip a

docker run --net host debian ip a
```

List and create networks

```
docker network ls
```

```
docker network create -d bridge my_bridge
```

Inspect a network

docker network inspect my_bridge

```
...
"IPAM": {
    "Driver": "default",
    "Options": {},
    "Config": [{
         "Subnet": "172.19.0.0/16",
         "Gateway": "172.19.0.1"
    }]
}
...
```

Add container to network

```
# Run on network

docker run -d --net=my_bridge --name nginx nginx
docker run -d --net=my_bridge --name apache httpd

# Connect to network

docker run -d --name nginx2 nginx
docker network connect my_bridge nginx2
```

Test network

```
docker run -ti --net my_bridge ondrejsika/host nginx

docker run -ti --net my_bridge ondrejsika/host apache

docker run -ti --net my_bridge ondrejsika/curl nginx

docker run -ti --net my_bridge ondrejsika/curl apache
```

Thank you & Questions

Ondrej Sika

email: ondrej@ondrejsika.com
twitter: @ondrejsika
linkedin: /in/ondrejsika/

Slides: https://sika.link/docker
https://github.com/ondrejsika/docker-training-examples

Did you enjoy the course?

Tweet about it!

@ondrejsika @rootcz