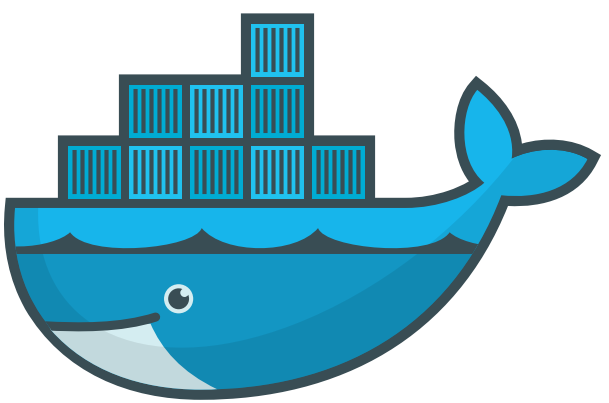
Chapter T:V

V. Docker Introduction

- Architecture
- Basic Commands
- Dockerfile Best Practices
- Debugging
- References



Logo credits: docker.com

© Bevendorff/Kiesel 2019

What is Docker?

- A virtualization technique that runs guest systems as containers
- A means of shipping and running micro services as portable images
- A handy tool for exploring and experimenting with new technologies
- An encapsulation mechanism to deploy applications in parallel without conflicts

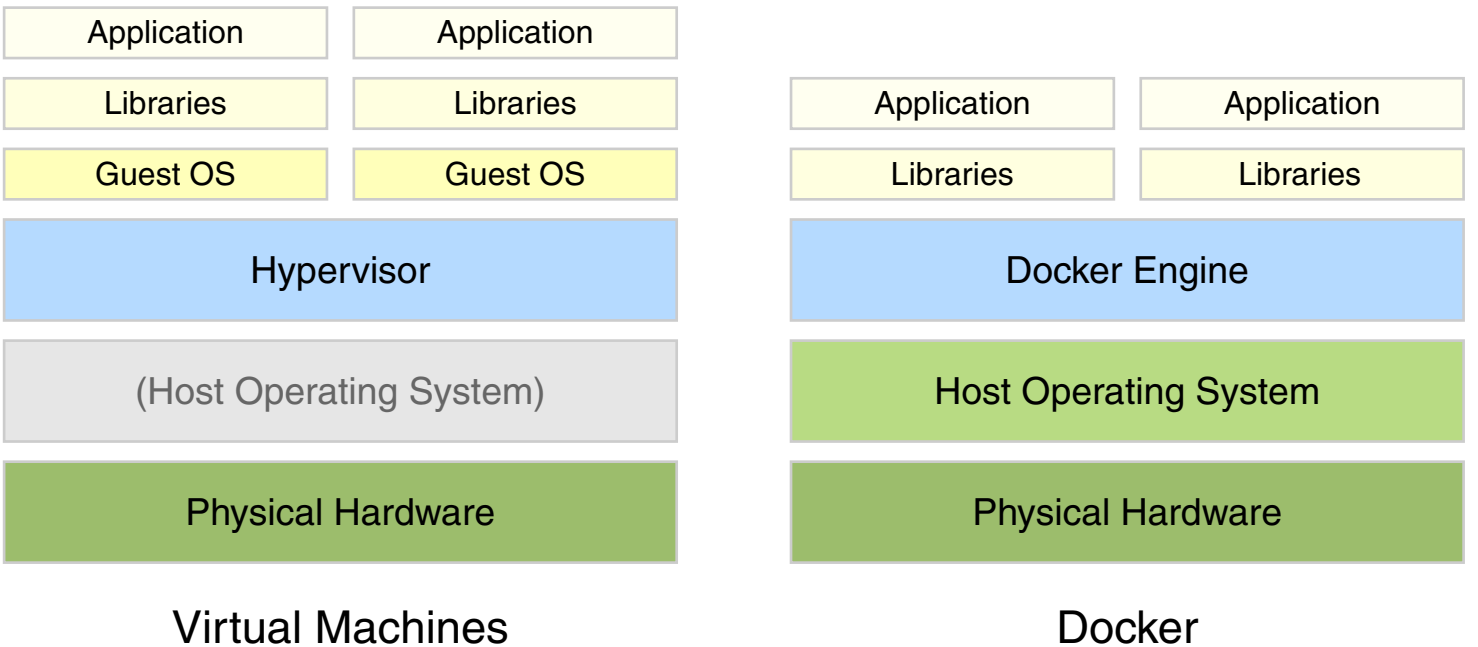
What is Docker not?

- A full replacement for virtual machines
- An out-of-the-box sandbox for untrusted code [Best Practices]
- A one-size-fits-all solution (not everything wants to be a Docker container)

© Bevendorff/Kiesel 2019

Docker Introduction Architecture

Virtual Machines and Docker

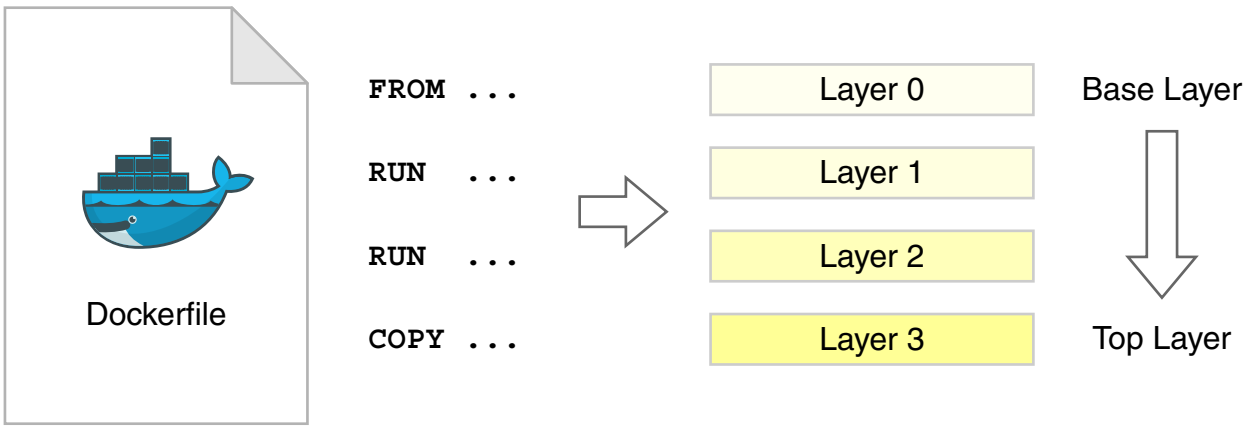


- Docker runs as a daemon on the host system.
- Containers share the host kernel, removing the need to virtualize a guest OS.

© Bevendorff/Kiesel 2019

Docker Introduction Architecture

Docker Images



- Docker containers are created from pre-compiled images.
- Images are built from `Dockerfile` recipes and have multiple layers.
- Images can use other images as base layer.
- Layers allow reuse of identical image parts and efficient build caching.
- Layers are not free and their size and number should be kept to a minimum.
- At runtime, a copy-on-write layer is added on top to allow in-memory modifications.

© Bevendorff/Kiesel 2019

Docker Images (continued)

- Ready-to-use images can be loaded from [Docker Hub].

  Docker pulls images automatically from Docker Hub first time they are started.

- A number of “official” OSS images are maintained by Docker, Inc. [Docker Hub]

- Application authors can build their own image with a custom `Dockerfile`.

© Bevendorff/Kiesel 2019

Docker Introduction Basic Commands

Running Containers

Start a container: [Docs]

```
$ docker run [--rm] [-ti] [--name CONT_NAME] \
             [-v HOST_PATH:CONT_PATH ...]  \
             [-p [HOST_IFACE:]HOST_PORT:CONT_PORT ...]  \
    IMG_NAME[:TAG] [CMD]
```

- `-t` and `-i` required for an interactive shell, `--rm` removes the container after use
- `-v` mounts host paths into container, `-p` forwards host ports to container

Execute a command inside an already running container: [Docs]

```
$ docker exec [-ti] CONT_NAME CMD
```


© Bevendorff/Kiesel 2019

Stopping Containers

Stop a container gracefully (`SIGTERM`): [Docs]

```
$ docker stop [-t TIMEOUT] CONT_NAME
```

Brutally murder it (`SIGKILL`): [Docs]

```
$ docker kill [-s SIGNAL] CONT_NAME
```

- `-s` also allows sending other signals such as `SIGHUP`

© Bevendorff/Kiesel 2019

Docker Introduction Basic Commands

Building and Pulling Images

Build an image from a `Dockerfile`: [Docs]

```
$ docker build [--no-cache] [-t IMG_NAME[:TAG]] PATH
```

`PATH` is the directory containing the `Dockerfile` (usually just `.`)

Pull or update an image explicitly: [Docs]

```
$ docker pull IMG_NAME[:TAG]
```

The suffix `TAG` designates the image version and defaults to `latest`.

© Bevendorff/Kiesel 2019

Docker Introduction Basic Commands

Exercise: Running Containers

- Run an image (create a container):
  Files: `code-kubernetes-hackathon-2019/examples/my-first-image`

```
$ sudo docker run --rm webis/my-first-image
```

- Run a server image (stop the container with CTRL+C):
  Files: `code-kubernetes-hackathon-2019/examples/my-first-server-image`

```
$ sudo docker run --rm --name my-first-server-image \
    -p 8001:80 webis/my-first-server-image
```

  Test: http://localhost:8001/not-my-first-file.txt

- Run the same image, but serving your current directory:

```
$ sudo docker run --rm --name my-first-server-image \
    -v "$PWD":/usr/local/apache2/htdocs/ \
    -p 8001:80 webis/my-first-server-image
```

(`sudo` is required only if your user is not part of the `docker` group)

© Bevendorff/Kiesel 2019

Docker Introduction Basic Commands

Exercise: Containers are Persistent

- Run the server image in the background (create a container):

```
$ sudo docker run -d --name my-first-server-container \
    -p 8001:80 webis/my-first-server-image
```

Test: http://localhost:8001/not-my-first-file.txt

Test: `$ sudo docker ps -a`

- Connect to the container and change the content:

```
$ sudo docker exec -it my-first-server-container bash
# echo "Hello Kubernetes" > htdocs/not-my-first-file.txt
# exit
```

- Stop, restart, kill, and delete the container
(reload http://localhost:8001/not-my-first-file.txt before each step):

```
$ sudo docker stop my-first-server-container
$ sudo docker start my-first-server-container
$ sudo docker kill my-first-server-container
$ sudo docker rm my-first-server-container
```


© Bevendorff/Kiesel 2019

Docker Introduction Basic Commands

Exercise: Working with Docker Hub

If an image should be run that is not available locally, it is fetched from online.

The default service is Docker Hub. [webis repository]

- Authenticate with Docker Hub:

```
$ sudo docker login
Username:  yourusername
Password:  yourpassword
```

- Update and push your local image:

```
$ cd code-kubernetes-hackathon-2019/examples/my-first-image
```

(now update the `Dockerfile` to show a personal message)

```
$ sudo docker build -t webis/my-first-image:yourname .
$ sudo docker push webis/my-first-image:yourname
```

Test: https://cloud.docker.com/u/webis/repository/docker/webis/my-first-image/tags

- Run the image of someone else:

```
$ sudo docker run --rm webis/my-first-image:johanneskiesel
```


© Bevendorff/Kiesel 2019

A `Dockerfile` is a sequential recipe for building an image. [Docs]

Most important commands are:

- `FROM` define the base image (e.g., `ubuntu:18.04`, `alpine:3.10`)
- `RUN` run a shell command (e.g., install packages)
- `ENV` set environment variables
- `COPY` copy files from the build context into the image
- `ADD` same as `COPY`, but also supports URLs (avoid if possible)
- `WORKDIR` default working directory inside the container
- `ENTRYPOINT` executable to run as PID 1 inside the container
- `CMD` command passed to `ENTRYPOINT` (if none given to `docker run`)

© Bevendorff/Kiesel 2019

- `Dockerfile` best practices have been devised to ensure images are...
  - ...as reusable as possible
  - ...as lightweight as possible
  - ...as secure as possible
- In the following, the three most important ones are listed. [Docs]
- A realistic example following these guidelines can be found in:

  `code-kubernetes-hackathon-2019/examples/flask-server`

© Bevendorff/Kiesel 2019

BP I: Reduce Image Size

Use the correct base image. Ubuntu is convenient, but not the smallest.

Common options are:

- `ubuntu:16.04|18.04` (∼190 MB)
- `centos:6|7` (∼170 MB)
- `debian:8|9` (∼125 MB)
- `alpine:3.9|3.10` (∼5 MB)

More specialized images are available also (e.g., `openjdk`, `python`).

© Bevendorff/Kiesel 2019

BP I: Reduce Image Size (continued)

`RUN`, `COPY`, `ADD` all create new layers.

- Use them sparingly
- Combine shell commands

© Bevendorff/Kiesel 2019

BP I: Reduce Image Size (continued)

`RUN`, `COPY`, `ADD` all create new layers.

- Use them sparingly
- Combine shell commands

Example:

```
RUN apt-get update \
    && apt-get install -y \
        build-essential \
        curl \
        gosu
```


© Bevendorff/Kiesel 2019

BP I: Reduce Image Size (continued)

Clean up as many files as you can, but make sure you do it on the same layer.

- Clean up temporary build files and package manager caches
- Use `--no-install-recommends` for installation via `apt-get`
- Run `apt-get autoremove` (if needed)
- Use `.dockerignore` to exclude unwanted files from `COPY` and `ADD` [Docs]

© Bevendorff/Kiesel 2019

BP I: Reduce Image Size (continued)

Clean up as many files as you can, but make sure you do it on the same layer.

- Clean up temporary build files and package manager caches
- Use `--no-install-recommends` for installation via `apt-get`
- Run `apt-get autoremove` (if needed)
- Use `.dockerignore` to exclude unwanted files from `COPY` and `ADD` [Docs]

Example:

```
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
        build-essential \
        curl \
        gosu
    && apt-get autoremove
    && rm -rf /var/lib/apt/lists/*
```


© Bevendorff/Kiesel 2019

BP II: Write Proper Entrypoints

Custom `ENTRYPOINT` scripts let you run your app with lowest possible privileges.

- Use `gosu` or `su-exec` for dropping privileges
- Do not use `su`, do not use `sudo` [Here's why]

© Bevendorff/Kiesel 2019

BP II: Write Proper Entrypoints

Custom `ENTRYPOINT` scripts let you run your app with lowest possible privileges.

- Use `gosu` or `su-exec` for dropping privileges
- Do not use `su`, do not use `sudo` [Here's why]

`docker-entrypoint.sh:`

```
#!/bin/sh
set -e
if [ "$1" = "postgres" ]; then
    exec gosu postgres "$@"
fi
exec "$@"
```


© Bevendorff/Kiesel 2019

BP II: Write Proper Entrypoints

Custom `ENTRYPOINT` scripts let you run your app with lowest possible privileges.

- Use `gosu` or `su-exec` for dropping privileges
- Do not use `su`, do not use `sudo` [Here's why]

`docker-entrypoint.sh`:

```
#!/bin/sh
set -e                                  # Fail if subcommand errors
if [ "$1" = "postgres" ]; then          # Check if CMD is postgres
    exec gosu postgres "$@"             # Exec CMD as postgres user
fi
exec "$@"                               # Exec all other CMDs as root
```


© Bevendorff/Kiesel 2019

BP II: Write Proper Entrypoints

Custom `ENTRYPOINT` scripts let you run your app with lowest possible privileges.

- Use `gosu` or `su-exec` for dropping privileges
- Do not use `su`, do not use `sudo` [Here's why]

`docker-entrypoint.sh:`

```
#!/bin/sh
set -e                              # Fail if subcommand errors
if [ "$1" = "postgres" ]; then      # Check if CMD is postgres
    exec gosu postgres "$@"         # Exec CMD as postgres user
fi
exec "$@"                           # Exec all other CMDs as root
```

`Dockerfile:`

```
COPY ./docker-entrypoint.sh /
ENTRYPOINT ["/docker-entrypoint.sh"]
CMD ["postgres"]
```


© Bevendorff/Kiesel 2019

BP II: Write Proper Entrypoints (continued)

Avoid the shell form of `ENTRYPOINT` and `CMD`.

Both are possible:

```
ENTRYPOINT ["/docker-entrypoint.sh"]
```

```
ENTRYPOINT "/docker-entrypoint.sh"
```

Avoid the second form:

- The value of `CMD` will be ignored
- Your entrypoint will be wrapped in a `/bin/sh` call and will not be PID 1
- Your entrypoint will not receive UNIX signals from `docker stop`

© Bevendorff/Kiesel 2019

BP III: Leverage Build Cache

Building images takes time. Leverage the build cache by...

- ...using the most specific base image that makes sense
- ...ordering commands from least to most frequently updated

Putting `COPY` or `ADD` last avoids many accidental rebuilds.

Make sure each layer is consistent in itself.

(e.g., always run `apt-get update` on same layer as package installations)

© Bevendorff/Kiesel 2019

Useful Debugging Guidelines

If a `Dockerfile` is not working as expected, consider the following steps:

- Re-run build with `--no-cache`. If that helps, your layers are inconsistent.
- Check execution rights of all script files (particularly `docker-entrypoint.sh`).
- Prefix `RUN` commands with `set -x` to print commands after shell expansion:

```
RUN set -x \
    && apt-get update ...
```

- When combining shell commands, it is easy to forget `\` or `&&`.
- Make sure you have no silent shell command failures. `set -e` may help.
- Check if all needed packages are installed.

  `--no-install-recommends` or `autoremove` can be surprising at times.
- Ensure that all commands run non-interactively (e.g., use `-y` for all `apt-get` commands).

© Bevendorff/Kiesel 2019

Docker Introduction References

- Official Docker Documentation
  https://docs.docker.com/
- Getting Started Guide
  https://docs.docker.com/get-started/
- Dockerfile Reference
  https://docs.docker.com/engine/reference/builder/
- Dockerfile Best Practices
  https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
- Docker Hub Browser
  https://hub.docker.com/search?q=&type=image
- Docker Hub Browser: “Official” Images
  https://hub.docker.com/search?q=&type=image&image_filter=official

© Bevendorff/Kiesel 2019