Chapter T:V

V. Docker Introduction

Logo credits: docker.com

Docker Introduction Architecture

Basics

Fundamentally, Docker is

- a specification for layered operating system images,
- a container runtime.

Docker Introduction Architecture

Basics

Fundamentally, Docker is

- a specification for layered operating system images,
- a container runtime.

Historically, both were proprietary technologies developed by Docker, Inc. Yet, to foster standardization, they were donated to the Open Container Initiative (OCI), a Linux Foundation project, in 2019.

Today, Docker is primarily an implementation of the OCI platform, but for the sake of simplicity, we will use both synonymously.

Docker Introduction

Container Virtualization

Containers are a lightweight, OS-level virtualization method for running multiple userspace instances under the same host kernel.

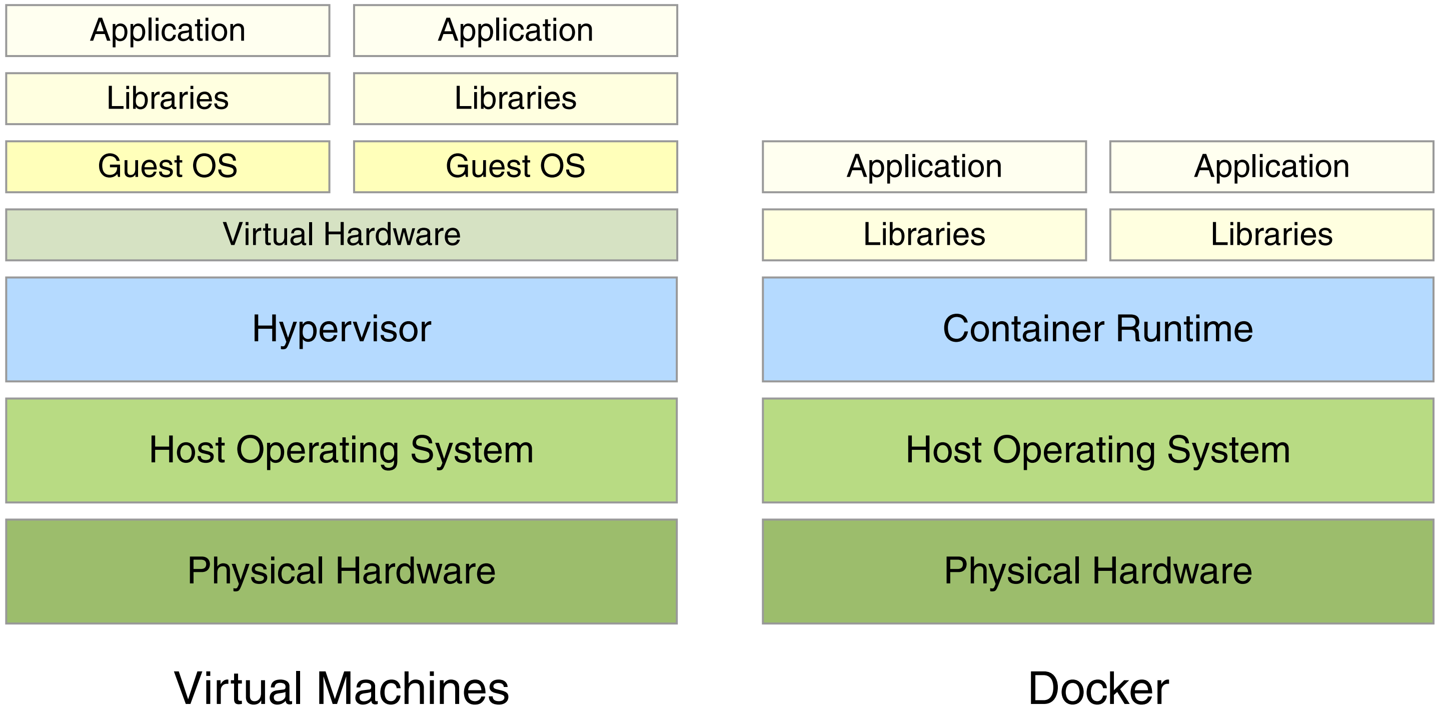
Contrary to “full” virtualization, no hardware is abstracted. This makes containers comparably lightweight and about as fast as the host system.

Hence containers serve as a suitable solution for

- sandboxing individual applications or services with different levels of access to resources of the host,
- namespacing deployments for running them in parallel without conflicts with each other or the host.

Docker Introduction Architecture

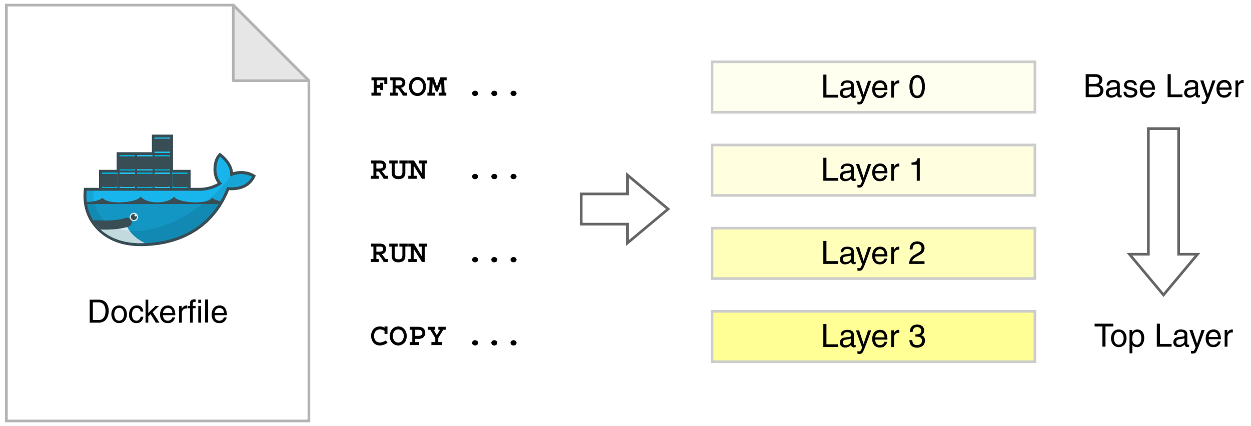
Container Virtualization



- The Docker container runtime runs as a daemon on the host system.
- Containers load an image containing a root file system.
- The kernel is shared between host and container, removing the need for full virtualization.

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.

Docker Introduction Architecture

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`.

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
```

Docker Introduction Basic Commands

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`

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`.

Docker Introduction Basic Commands

Building and Pulling Images

In the following, we will use two images built from these `Dockerfile` snippets:

- Image `my-image`:

  ```
  FROM alpine
  CMD ["sh", "-c", "echo Hello World!"]
  ```

- Image `my-server-image`:

  ```
  FROM httpd:2.4
  RUN echo "Hello World" > /usr/local/apache2/htdocs/index.html
  CMD ["httpd-foreground"]
  ```

Docker Introduction

Exercise: Running Containers

- Build and run `my-image`:

```
$ docker build -t my-image my-image-src-dir
$ docker run --rm my-image
Hello World!
```

- Build and run `my-server-image` (stop the container with CTRL+C):

```
$ docker build -t my-server-image my-server-image-src-dir
$ docker run --rm --name my-server-container \
    -p 8001:80 my-server-image
```

Test: http://localhost:8001/

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

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

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

Docker Introduction Basic Commands

Exercise: Containers are Persistent

- Run the server image in the background:

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

Test: http://localhost:8001/

Show running containers: `$ docker ps`

- Connect to the container and change the file contents:

```
$ docker exec -it my-server-container bash
# echo "Hello Docker" > htdocs/index.html
# exit
```

Test again: http://localhost:8001/

- Stop, restart, kill, and delete the container:

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

Docker Introduction Basic Commands

Exercise: Working with Docker Hub

If an image should be run that is not available locally, it is fetched from an online registry. The default registry is Docker Hub, yet there are many others. [webis repository]

- ❑ Authenticate with Docker Hub:

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

- ❑ Update image name and push it to your Docker Hub namespace:

```
$ docker tag my-image yourusername/my-image:1.0
$ docker push yourusername/my-image:1.0
```

Docker Introduction Dockerfile Best Practices

Introduction

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

The 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`)

Docker Introduction Dockerfile Best Practices

Introduction

- `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]

Docker Introduction Dockerfile Best Practices

BP I: Reduce Image Size

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

Common options are:

- `ubuntu:20.04|focal` (~72 MB)
- `centos:8` (~230 MB)
- `debian:11|bullseye` (~125 MB)
- `alpine:3|3.15` (~5 MB)

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

Docker Introduction Dockerfile Best Practices

BP I: Reduce Image Size (continued)

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

- ❑ Use them sparingly
- ❑ Combine shell commands

Docker Introduction Dockerfile Best Practices

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
```

Docker Introduction Dockerfile Best Practices

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]

Docker Introduction Dockerfile Best Practices

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/*
```

Docker Introduction Dockerfile Best Practices

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]

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 "$@"
```

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
```

Docker Introduction Dockerfile Best Practices

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"]
```

Docker Introduction Dockerfile Best Practices

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`

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)

Docker Introduction Debugging

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).

Docker Introduction References

- Official Docker Documentation
  https://docs.docker.com/
- Open Container Initiative
  https://opencontainers.org/
- 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