Docker Basics

In this lesson, we'll go over some Docker basics.

WE'LL COVER THE FOLLOWING

What is Docker?

Docker represents a lightweight alternative to virtualization. Although Docker does not provide as much isolation as a virtualization, it is practically as lightweight as a process.

Shared kernel

Instead of having a complete virtual machine of their own, Docker containers *share* the *kernel* of the operating system on the Docker host. The Docker host is the system on which the Docker containers run. The processes from the containers, therefore, appear in the process table of the operating system on which the Docker containers are running.

Isolated network of Dockers

The Docker containers have their **own network interface**. In this way, the **same port can be used in each Docker container**, and each container can use any number of ports.

The network interface is in a subnet where all Docker containers are accessible. The subnet is not accessible from the outside. This is at least the standard configuration of Docker.

The Docker network configuration offers many other alternatives. To still allow external access to a Docker container from the outside, **ports of a Docker container can be mapped to ports on the Docker host**. When mapping the ports of the Docker containers to the ports of the Docker host, be careful because each port of the Docker host can only be mapped to one port of a Docker container.

Optimized file system

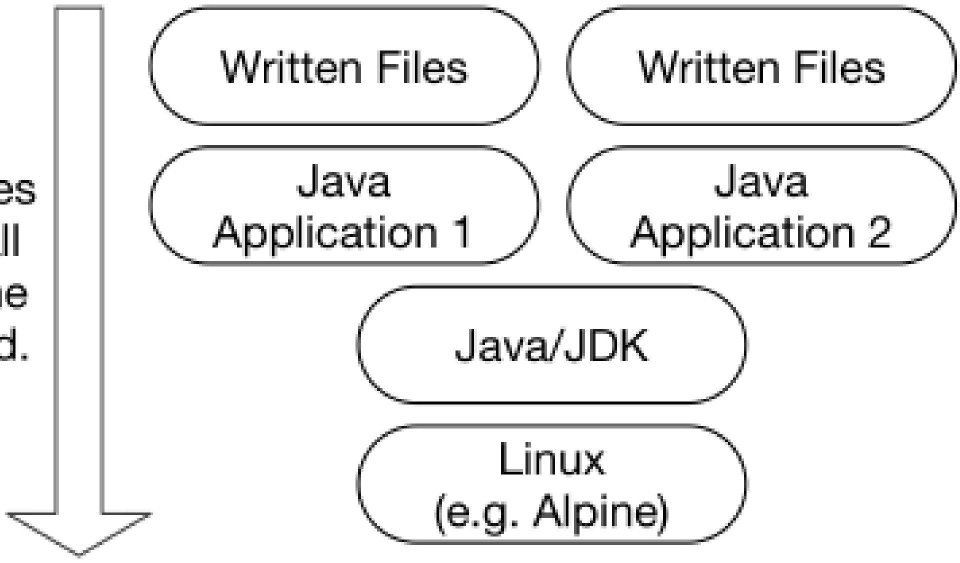
Finally, the file system is optimized. There are **layers in the file system**. When a microservice reads a file, it goes through the layers from top to bottom until it finds the data. The containers can share layers.

The drawing below shows this more precisely. The file system layer at the bottom represents a simple Linux installation with the Alpine Linux distribution. Another layer is the Java installation. Both applications share these two layers, which are stored only once on the hard disk, although both microservices use them.

Only the applications are stored in file system layers that are exclusively available to a single container. The lower layers cannot be changed. The microservices can only write to the top layer. The reuse of the layers reduces the storage requirements of the Docker containers.

It is easily **possible to start hundreds of containers on a laptop**. This is not surprising: after all, it is also possible to start hundreds of processes on a laptop. Docker has **no significant overhead** compared to a process.

Compared to virtual machines, however, the **performance benefits are outstanding**.



File System Layers in Docker

One process per container

Ultimately, Docker containers are highly isolated processes due to their own network interface and file system.

Therefore, **only one process should run in a Docker container**. Running more than one process in a Docker container contradicts the idea of separating processes by means of Docker containers. Because only one process is supposed to run in a Docker container, there should be no background services or daemons in Docker containers.

Docker image and Docker registry

File systems of Docker containers can be exported as Docker images. These images can be passed on as files or stored in a Docker registry.

Many repositories such as Nexus and Artifactory can store and provide Docker images just like compiled software and libraries. This makes it easy to exchange Docker images with a Docker registry for installation in production. The transfer of images from and to the registry is optimized. Only the updated layers are transferred.

Supported operating systems

Supported operating systems

Docker **was originally a Linux technology**. For operating systems such as **macOS and Windows, Docker installations are available**. For this purpose, a virtual machine with a Linux installation is running in the background. This is transparent for the user. It seems as if the Docker containers are running directly on a computer.

In addition, Windows, since Windows Server 2016, provides Windows Docker containers. Linux applications run in a Linux Docker container and Windows applications in a Windows Docker container.

Operating systems for Docker

Docker changes the requirements for operating systems.

One process per container

Only one process is supposed to run in one *Docker container*. This means that only as much of the operating system is required as is needed to run that process.

- For a Java application, we use the Java Virtual Machine (JVM), which requires some Linux libraries that are loaded at runtime. A shell is not necessary, for example. Therefore, distributions like Alpine Linux, which are just a few megabytes in size, only contain the most important tools, making them an ideal basis for Docker containers.
- The programming language, Go, can create statically linked programs. In that case, nothing else has to be available in the Docker container besides the program itself, and no Linux distribution is required at all.

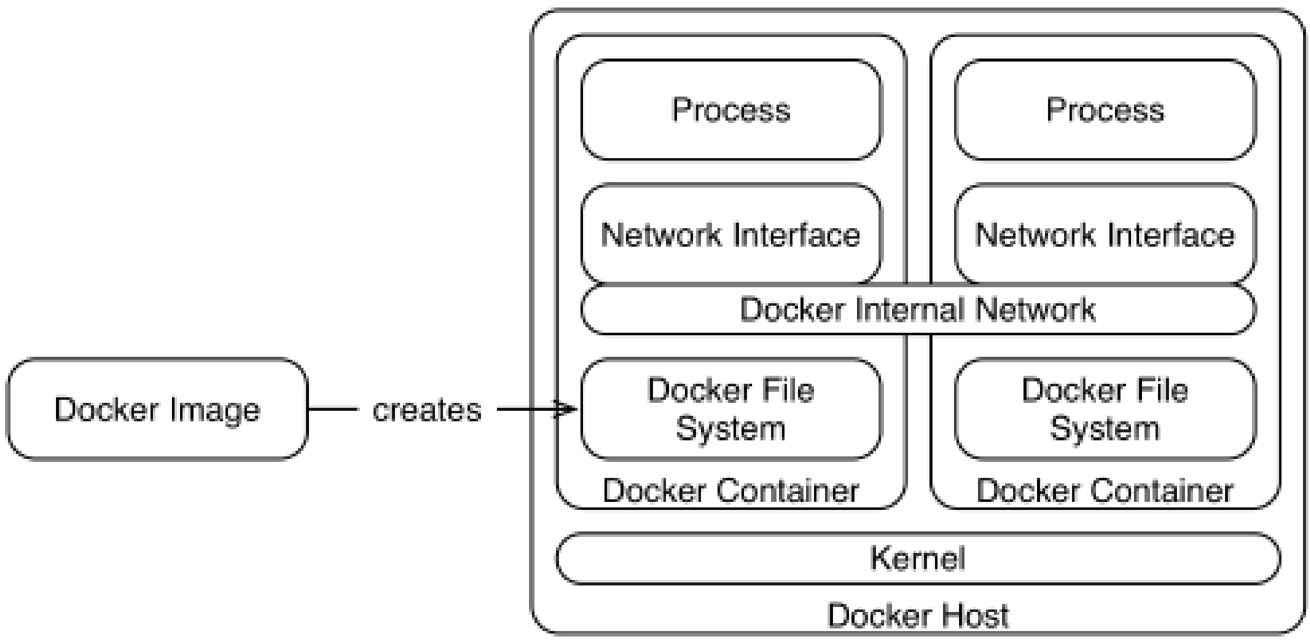
Host does not have to have tools needed in the containers

The *Docker host* on which the Docker containers run **only has to run Docker containers**. Many Linux tools are therefore superfluous.

- CoreOS is a Linux distribution that can run little more than Docker containers and, for example, considerably simplifies operating system updates of an entire cluster.
- CoreOS can also serve as a basis for Kubernetes.

Overview

The drawing below shows an overview of the Docker concepts.



Overview of Docker

- The **Docker host** is the machine on which the Docker containers run. It can be a *virtual* machine or a *physical* machine.

- **Docker containers** run on the Docker host.

- The containers typically contain a **process**.

- Each container has its own **network interface** with its own IP address. This network interface is only accessible from the Docker internal network. However, there are also ways to allow access from outside this network.

- In addition, each container has its own **file system**.

- When a container is started, the **Docker image** creates the first version of the Docker file system. When the container has been started, the image is extended by another layer into which the container can write its own data.

- All Docker containers share the **kernel** of the Docker host.

Does it always have to be docker?

Does it always have to be docker?

Docker is a very popular option for deploying microservices. However, there are **alternatives**. Two alternatives were already mentioned in this lesson: **virtual machines** or **processes**.

Microservices as WARs in Java application servers

However, you can also deploy several microservices as **WAR files** in a Java application server or Java web server. WARs contain a Java web application, and can be deployed separately.

However, the deployment of a WAR may require the server to be restarted. Because microservices should only be deployable separately as per chapter 2, **microservices can be implemented as WARs**.

Disadvantages

- **Compromises are made with regards to robustness**. A memory leak in a microservice can lead to failure of all microservices. The microservice would allocate more and more memory until an `OutOfMemoryError` is thrown and the entire Java application server crashes.
- **Separate scalability is also difficult to implement** because each server contains all microservices and, therefore, only all microservices can be scaled together.
  - This makes the scaling more complex than in cases where each server contains only one microservice as unneeded microservices are also scaled.
  - Of course, it would be possible to run each WAR on a separate Java web server and have multiple instances of each of these Java web servers. But then the WARs would no longer run together on one Java web server.
- And finally, all microservices run in one operating system process, which is a **compromise in terms of security**. When a hacker can take over the process, he or she has access to the entire functionality and data of all microservices.

Advantages

- In return, these approaches **consume less resources**. An application server with several web applications requires only one JVM (Java Virtual Machine), only one process, and only one operating system instance.
- Furthermore, there is **no need to introduce a new infrastructure** if application servers are already in use, which can reduce the workload for operations.

QUIZ

1

Consider an application where all the microservices have to be individually externally accessible from the network. Does Docker solve the issue of keeping track of unused ports that the host machine has?

COMPLETED 0%

1 of 3 ‹ ›

In the next lesson, we'll study `Dockerfiles`!