Docker for Microservices: Reasons

In this lesson, we'll look at some reasons for using Docker for microservices.

WE'LL COVER THE FOLLOWING

Chapter 2 defined microservices as separately deployable units. The separate deployment not only results in a decoupling at the architectural level, but also in regard to technology choice, robustness, security, and scalability.

OS processes for microservices

If microservices are supposed to have all these characteristics, the question arises as to **how they can be implemented**. Microservices must be scalable independently of each other. In the event of a crash, a microservice must not be allowed to make other microservices unavailable, too, and thus endanger the robustness of the whole system. Therefore, **microservices must at least be separate processes**.

Scalability can be guaranteed by multiple instances of a process. When an application is started, the operating system generates a process and allocates resources such as CPU or memory to it. Therefore, more processes can use more resources.

But processes are limited concerning scaling. If the processes all run on one server, then only a limited amount of hardware resources are available. Instead, the microservices should run in a cluster. Kubernetes and Cloud Foundry support running microservices in a cluster.

With processes, robustness is guaranteed to a certain extent because the

crash of one process does not affect the other processes. However, a server failure still causes a large number of processes, and thus microservices, to fail.

But there are also other problems:

- **All processes share one operating system**. It must provide the libraries and tools for all microservices. Each microservice must be compatible with the operating system version. It is difficult to configure the operating system to support all microservices.
- In addition, the processes must coordinate in such a way that each process **has its own network port**. If you have a large number of processes, it becomes increasingly harder to find unused ports. Also, it's hard to figure out which ports are used by which process.

Virtual machines: too heavy-weight for microservices

Instead of a process, **each microservice can run in its own virtual** machine.

> **Virtual machines** are simulated computers that run on the same physical hardware. For the operating system and application, virtual machines look exactly like a physical server.

Through virtualization, the microservice has its own operating system installation. Thus, the **configuration of the operating system can be adapted to the specific microservice**, and there is also **complete freedom in choosing the network port**.

Overhead

However, a virtual machine has a **substantial overhead**:

- The virtual machine must give the operating system the illusion of running directly on the hardware. This leads to overhead. Therefore, **performance is poorer** than with physical hardware.
- Each microservice has its own instance of the operating system. This **consumes a lot of memory** in the *RAM*.
- Finally, the virtual machine has virtual disks with a complete operating

system installation. This means that the microservice **occupies a lot of hard disk space**.

So virtual machines have an overhead, making their operation expensive. In addition, operations will have to **manage a large number of virtual servers**. This is **complicated and time-consuming**.

The **ideal solution** would be a **lightweight alternative to virtualization**, which possesses the isolation of virtual machines, but consumes as little resources as processes do and is similarly easy to operate.

QUIZ

1 Which of the following is NOT a reason why microservices must at least be implemented as separate processes?

COMPLETED 0%

1 of 4

In the next lesson, we'll discuss some Docker basics.