Variations

In this lesson, we'll discuss some variations to the docker approaches we've already discussed.

WE'LL COVER THE FOLLOWING

There are not really any fundamental alternatives to Docker:

- **Virtualization** has too much overhead.
- **Processes** are not sufficiently isolated. The required libraries and runtime environments for all microservices must be installed in the operating system. This can be difficult because each process occupies one port, and therefore the allocation of ports must be coordinated.
- Other **container solutions** such as rkt are far less common.

Clusters

For production, **applications should run in a cluster**. Only in this way can the system be scaled across several servers and secured against the failure of individual servers.

Docker Compose can use Docker Swarm Mode for **cluster management**. Docker Swarm Mode is built into Docker.

Kubernetes is widely used for operating Docker containers in a cluster.

There are also other systems like **Mesos**. Mesos is actually a system for managing batches for data analysis in a cluster but it also supports Docker

containers. Offers in the cloud are also available, such as ECS (EC2 Container Service).

Docker without Scheduler

An alternative is to install Docker containers **directly on a server**. In this scenario, the servers are provided with classic mechanisms – for example, with virtualization. Linux or Windows is installed on the server. The microservice runs in a Docker container.

The only difference to the procedure without Docker is that Docker containers are now used for deployment. But this already makes it much easier to keep production and test environments identical. Concepts such as *immutable servers* are also easier to implement, as is the technology freedom for microservices.

Traditional virtualization is still responsible for high availability, scaling, and distribution to the servers.

This approach offers some of the advantages of Docker and reduces the effort.

These approaches have **one thing in common**: *how* the load is distributed in the cluster is decided by the scheduler – that is, by Kubernetes or Docker Swarm Mode. This means that the **scheduler is of crucial importance** for fail-safety and load balancing.

If a container fails, a new container must be started. Likewise, additional containers must be started at times of high loads, ideally on machines that are not too busy.

not too busy.

Schedulers such as Kubernetes solve many challenges, especially with synchronous microservices. It is highly recommended to use such a platform for operating microservices.

But the introduction of microservices requires many changes. The architecture needs to be adapted, and developers need to learn new approaches and technologies, as well as having adopted the deployment pipeline and the tests.

Implementing an additional technology such as a Docker scheduler should, therefore, be well thought out because many other changes also have to be made.

PaaS

Another alternative is a **PaaS**. A PaaS has a higher degree of abstraction than Docker schedulers because only the application needs to be provided.

The PaaS creates the Docker images. Therefore, a PaaS can be the simpler and therefore better solution.

Experiments

- Docker Machine can use clouds like Amazon Cloud or Microsoft Azure with the appropriate drivers. Create an account with one of the cloud providers. Most cloud providers offer free capacity to a new user.
- Create an account in the Docker Hub. Build a Docker image, for example, based on one of the microservices examples of the following chapters and place it in the Docker hub with `docker push`.
- Use the tutorials to familiarize yourself with the Docker Swarm Mode, which can be used to run Docker in a cluster.

QUIZ

1 Why should applications run in a cluster?

COMPLETED 0%

1 of 2

In the next lesson, we'll look at a quick conclusion of this chapter.