# Using Docker in a production environment.

Docker is a system that allows us to abstract away the underlying platform and infrastructure. This leads to a predictable and stable deployment of Collaboard on any platform.

However, Docker itself is nothing more than a way of packaging software and running it somewhere. All parts of the installation must talk to one another through some means. That can be:

* Shared files
* Database content
* Underlying network topology

The shared files solution is used to share the content the users create and that different parts of the system need to access. The database content is shared since all containers have access to the running database instance. However, that last one requires the database to be accessible through the network topology.

Docker by default runs on a single machine. It uses all resources from the host machine. It will create a virtual layer with its own virtual network inside it. This way we can have a secure environment: only the containers running inside this virtual network have access to the resources inside. That is, unless we mark items otherwise. We do that for the shared files.

## Limits to the single server

Docker relies on the operating system it runs on to handle certain tasks. Most of those tasks have to do with supplying means for communication between the different containers that are running. This works extremely well but there can be limitations to what we can achieve running on one machine.

### Scalability

The virtual network runs on the local system and thus the performance of all containers is dependent on the power of the machine running them. The better performant the host machine is, the better performant Collaboard is. This means that to serve a larger number of users, we need to have a bigger machine running Docker.

There is a physical limitation to this. Machines can only get a certain size. If we reach the limits of the hardware available, we need to scale out to more hardware.

### Reliability

Docker is fault tolerant. If one of the containers fail, Docker will notice and restart them so that the service in that container remains available, be it after a slight delay. Of course, this requires Docker to be running all the time. If there is an issue on the Docker level itself, or even on the hardware level there is a bigger issue. If the host machine fails, the whole Collaboard system will be unavailable. It would be good to have a redundant solution so we can have a fallback scenario.

Both issues require to have more than one machine available. We can use two or more host systems to run Docker and have parts replicated and distributed across hardware boundaries.

## Orchestration

To do that, we need some additional piece of software that handles all cross-machine communications. In other words: we need something that monitors all containers and handles all network across physical machines so that the internal docker system behaves as if it is all running on one machine.

This is done through something we call the orchestrator.

There are several software solutions for this, but the two most uses are Docker Swarm and Kubernetes. Both offer a solution to the orchestration issue and are comparable. They do have differences and their up- and downsides.

### Docker Swarm

Docker Swarm is the native solution from Docker itself. It enables the system to be running on multiple machines. By installing Docker Swarm on multiple machines that are interconnected through a network we can use the native Docker Commands to deploy all containers to more than one machine. The network itself can also consist of a cluster of networks that can reach each other: it does not need to be one network of one subnet.

We need to make sure we assign some machines the role of Manager, and the other machines will be nodes. The Manager machines handle all the health-checking and load balancing. Setting this up is extremely easy: all we need to do is issue a couple of commands and then tell the system to distribute itself.

Docker Swarm is built in Docker, so there is no additional software needed. Collaboard comes with a default DockerCompose file that tells Docker how to deploy and run the system; that same DockerCompose file can be used to instruct Swarm how to operate, with the only change being that we tell some services to deploy more than once.

Docker Swarm is extremely simple to use and to deploy and it works very well in systems with less than 100 users. Although Docker Swarm can service limitless users and machines, the lack of built-in monitoring and control software makes it a challenging to use if the system grows beyond a certain point.

If that happens, it is best to move to the current market leader in orchestration Kubernetes.

### Kubernetes

Kubernetes is an open-source solution originating from Google. It does fulfill the same roles as Docker Swarm but is more geared towards bigger installations. Kubernetes is a native offering on many cloud providers, making this the better choice if the installation needs to be done on such an environment. Azure, Amazon, and Google all offer Kubernetes enabled environments that are ready to use.

Kubernetes has a more complex system in place to deploy. We cannot use the current relatively simple DockerCompose file. We supply a file that instructs Kubernetes how to deploy.

Kubernetes has a strong support from the community and a large group of people are available to help with issues. It is the most popular container orchestration platform. Reason for this is that it is extremely well suited to handle large complex environments and large number of users.

## Choosing between Docker Swarm and Kubernetes

Giving a good recommendation on when to use which platform is not easy: it depends on a lot of factors. Yet we can give some sort of recommendation when to use what under what circumstances and why this choice is made.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Use Docker when.. | User Docker Swarm when… | User Kubernetes when… |
| Scale | Small local installation for non-business critical or testing purposes | Local installation without Kubernetes knowledge available for < 250 users and enough local on-premises hardware available | Larger installation or cloud-based installation. Kubernetes knowledge is required to monitor and finetune systems performance and usage |
| Available knowledge | No container knowledge available | Some Docker knowledge available | Access to Kubernetes knowledge |
| Monitoring needs | No monitoring necessary | Little or manual reporting on system usage needed | Extensive reporting needed on system behavior, performance, and usage |
| Tooling | Command line-based tooling | Command line-based tooling, or 3rd party tooling | GUI / Web based tooling and monitoring tools |
| Easy of deployment | Simple and quick | Simple and quick | Less simple and more intensive |
| Required availability | No requirements | Higher availability, but becoming more complex when number of user increases | High availability from the start |