

**Collaboard Sizing**

On-premises sizing recommendations

|  |  |
| --- | --- |
| Title: | Sizing Recommendations |
| Abstract: | This document details the size recommendation when installing Collaboard on-premises |
| Status\* | Draft |
| Document Owner: | IBV |
| Document Author: | Gian Paolo Santopaolo |
| \*Draft, In progress, Reviewed, Authorized, Approved, Completed | |

|  |  |  |
| --- | --- | --- |
| **Document Control** | | |
| Document Distribution: | Internal: | All dev Teams |
|  | External: |  |
| Document Authorization: | Authorizer(s): | Gian Paolo Santopaolo |

|  |  |  |  |
| --- | --- | --- | --- |
| **Change History** | | | |
| Date | Ver | Changed By | Details |
| 27/1/2021 | 1.0 | Dennis Vroegop | Initial Draft and Docker chapter |
| 28/1/2021 | 1.1 | Dennis Vroegop | Added database and storage chapters |
| 29/1/2021 | 1.2 | G.P Santopaolo | Added reliability and sizing chapters. Document review |
| 23/02/2021 | 1.3 | G.P Santopaolo | Updates to several chapters |
| 10/03/2021 | 1.4 | G.P Santopaolo | Updates to sizing recommendations |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**NOTE FOR THE READER:**

**This document is a draft; at the moment, we are acquiring the necessary data to make a real sizing recommendation.**

**This is the draft with the skeleton for the final document to be done after collecting the necessary data**

Index

[Running database on a container 4](#_Toc66262118)

[SQL Server running in a container 4](#_Toc66262119)

[SQL Server running in a separate environment 4](#_Toc66262120)

[Choosing between the solutions 4](#_Toc66262121)

[Recommendation 5](#_Toc66262122)

[SQL Server Licensing 6](#_Toc66262123)

[Storage recommendations 6](#_Toc66262124)

[Storage options 6](#_Toc66262125)

[Local storage 6](#_Toc66262126)

[Network Attached Storage (NAS) 7](#_Toc66262127)

[Storage Area Network (SAN) 7](#_Toc66262128)

[Back-ups 7](#_Toc66262129)

[Reliability 8](#_Toc66262130)

[Networking - Secure protocols and bandwidth 8](#_Toc66262131)

[Using Docker in a production environment 10](#_Toc66262132)

[Limits to the single server 10](#_Toc66262133)

[Scalability 10](#_Toc66262134)

[Reliability 10](#_Toc66262135)

[Orchestration 10](#_Toc66262136)

[Docker Swarm 11](#_Toc66262137)

[Kubernetes 11](#_Toc66262138)

[Choosing between Docker Swarm and Kubernetes 11](#_Toc66262139)

[Sizing recommendations 13](#_Toc66262140)

[Base images 13](#_Toc66262141)

[Sizing recommendations 1-250 users 13](#_Toc66262142)

[Sizing recommendations 250-500 users 14](#_Toc66262143)

[Sizing recommendations 500-1000 users 14](#_Toc66262144)

[Sizing recommendations 1000-2000 users 14](#_Toc66262145)

[Sizing recommendations 2000-5000 users 14](#_Toc66262146)

[For all configurations 14](#_Toc66262147)

# Running database on a container

The heart of our system is the database. This is where we store all user-generated data; we use it to track items moving through the boards, which controls our back-end services. The database is one of the most vital parts of the system; thus it is worth paying attention to the deployment of it.

We use SQL Server, and we have a variety of options of storing the data and the location of the database engine. To make an informed decision on how to deploy SQL Server for Collaboard we need to look at the different options and the choices we must make.

## SQL Server running in a container

By default, when we deploy Collaboard on a containerized platform, we also deliver SQL Server. This instance runs in a container. The actual datafiles generated and used by the server are on a shared server. This is done to make sure the database files survive a container restart in case of an outage.

Installing SQL Server on a container allows us to choose the edition we run. Each edition of SQL server has other options that can be used. For a minimal installation, the Express edition might suffice but we recommend strongly against running that for the production system. In that case we can choose from one of the following options:

* Standard
* Enterprise
* EnterpriseCore

Although we supply the container with the database engine in the requested edition, it is the responsibility of the user to make sure the right license is available and used for this. Standard, Enterprise and Enterprise all require their own license from Microsoft. We do not supply this as part of our offering.

## SQL Server running in a separate environment

The containers need to have a connection string to connect to the database. This means we do not really care where the actual engine is. If it is reachable from the docker network or the orchestrated network, we can use it.

Thus, if there already is SQL Server running or if there is room on the available hardware or cloud space to have SQL Server, we can just change the connection string and have the system use that. In this case the shared folder for the database files is not needed: we leave that to the database administrator to take care of.

## Choosing between the solutions

Which solution is chosen depends on the needs and requirements of the user of Collaboard.

We have the factors that influence this decision captured in the table below.

|  |  |  |
| --- | --- | --- |
|  | **Container** | **Outside container** |
| Installation type | Trying out, prototyping, small user base | Larger userbase (> 25 users), production environments |
| Need for backup | Backup means shutting down SQL Server, thus shutting down Collaboard | Regular backups of SQL Server can be used, minimizing downtime |
| Scalability | Small installation (<25 users with little activities). Not scalable besides increasing the hardware the container runs on. | Larger installation (>25 users). SQL Server can scale both on premises and in the cloud, including clustering servers. Scaling is virtually limitless and not bound to region |
| Stability | Local database running on one process, thus prone to downtime.  Recovery after calamity is hard to do | With failover options, stability can be in the 99.999% area. Recovery can be done if backups are taken care of and can be done without causing downtime |

Recommendation

For larger production usage, we recommend using a separate instance of SQL Server. This way the backup, scaling and disaster recovery can be handled the same way all other databases are done.

For smaller instances or test installations, the containerized solution will be adequate if the database files are backed-up regularly.

# SQL Server Licensing

When planning a Collaboard installation, it is crucial to keep in consideration that Collaboard needs to store part of its data in a SQL Server 2019 database/

There is a description on this page https://www.microsoft.com/en-us/sql-server/sql-server-2019-pricing

(make sure you download the two pdf and scroll down to pricing).

For Collaboard to be fully functional, for us is enough:

* One SQL Standard – server license ($899,00)
* One Standard – CAL ($209,00) per each server connecting to the SQL plus two more that shall be used for maintenance / administrative purpose

Of course, all the reliability/disaster recovery/fault tolerance (for both database and files) is at the customer's discretion. Depending on how valuable the information stored in Collaborad, they may opt for a higher licensing level which may dramatically affect the cost.

# Storage recommendations

Collaboard enables users to share notes, ink, and shapes. But it also allows for sharing images, videos, and documents. That last category of shareable items needs to be stored somewhere so we can retrieve it again later.

Since containers lose their contents every time they get recycled, which could happen due to a system reset or an unexpected termination of the service, we need to store the files outside of the containers.

Having the files outside of the containers also enables us to create and restore backups of these files.

## Storage options

Collaboard needs fast read and writes access to the files. Multiple services and users will need to access those files, so a fast storage medium is required. Next to that, when the usage of Collaboard grows, more storage space is needed as well.

Unfortunately, it is hard to predict the amount of storage needed. Some users generate little content in the form of videos, images, or documents and use sticky notes and ink a lot. Other users do the opposite.

But by monitoring the growth of the shared storage, the total future needs can be extrapolated from the current size of the current storage size.

### Local storage

When it comes to speed, faster is always better. But as a minimum, we recommend having a drive that can read/write at around 500 MB/s. But this is just the minimum and only applies when the shared drives are on the same physical device as the container runs, or at least is near it to reduce network latency.

A good and modern SSD can deliver these speeds easily and are very reliable. Still, the user is responsible for making sure the files in the shared folders are regularly backed up and that the backup is tested. Backing up can be done without shutting down Collaboard. Restoring lost files might require a Collaboard shut down, but that depends on the situation.

### Network Attached Storage (NAS)

NAS is a way of sharing storage space that is well suited for smaller installations. It is not intended to be used for large amounts of data and large amounts of users at the same time.

It is a better solution than having a local drive for storing Collaboard files: it is more scalable and easier to use in a fault-tolerant RAID environment. The speed of the storage depends on the speed of the network, so a stable and fast network connection is recommended.

### Storage Area Network (SAN)

A SAN is a more enterprise-oriented storage solution. Although the price is generally higher, stability, scalability and fault tolerance are also much higher. If Collaboard is serving a large group of users, or if the users have lots of images, videos, or documents on their boards (or both!) SAN is the preferred way to go. This will ensure Collaboard runs at its fastest without being blocked by file and network latency.

### Back-ups

Backing up, restoring and disaster recovery are always the responsibility of the users. This needs to be considered when determining the size and sort of storage.

# Reliability

It is essential to specify that our solution and the way we work is by scaling the first instance on the code. Thus, our code is highly optimized, and we only use talented senior developers, most of the time, they are Microsoft MVP as well.

Because of the architecture that we develop, the solution can run on a single layer and different tiers.

Of course, there is something that cannot be reached only by developing the best code possible, and this is when it comes to very high volume, and the solution needs to scale up the hardware way or when it comes to reliability.

That's all said; we came to concepts such as load balancer or balancer of balancer that they can add the level of reliability that the customers want to reach.

The same is when it comes to SQL server; our application can run on a single instance database. It's on the customer to understand which level of reliability/security is needed for the database server.

Environments

For the deployment to be successful, our suggestion is to have at last, but then it depends on specific customer rules, two different environments:

* One for staging
* One for production

## Networking - Secure protocols and bandwidth

The formula below is usually used to measure performance on an SOA architecture when it comes to measuring the response time of a SOA service.

|  |  |
| --- | --- |
| Variable | Definition |
| RT | Response time. The total time from the user requesting a REST API (by clicking a button, and so on) to when the message come back and the UI shows desidered content with information coming from REST response. Typically measured in seconds. |
| Payload | Total bytes sent to the SOA service, RESTfull object + http(s) header and so on |
| Ecr | Encryption for the request, if absent 1 |
| Ect | Encryption for the transport channel eg. https, if absent 1 |
| Bandwidth | Rate of transfer to and from the server. This may be asymmetrical and might represent multiple speeds if a given page is generated from multiple sources. Usually, it is averaged together to create a single bandwidth expressed in bytes per second. |
| RTT | The time it takes to round-trip, regardless of bytes transferred. Every request pays a minimum of one RTT for the page itself. Typically measured in milliseconds |
| Concurrent Requests | Number of simultaneous requests a browser will make for resource files. By default, Internet Explorer performs two concurrent requests. This setting can be adjusted but rarely is |
| Cs | Compute time on the server. This is the time it takes for code to run, retrieve data from the database, and compose the response to be sent to the browser. Measured in milliseconds |
| Cc | Compute time on the client. This is the time it takes for a client for code to run, analyze the response and render the UI |

The main challenge in calculating this formula is by getting an exact measurement of each element.

This is challenging but not impossible, there are tools, and maybe some performance counters would help measure compute times.

This chapter aims not to dig into the formula itself but to look at two very important variables: Ecr and Ect so about security.

The higher these values will be, the more secure your service will be, but as you can clearly see from the formula, these two variables will exponentially impact the response time.

When deploying in production, our recommendation is to deeply impact the security level you choose to have the right benefit on both sides: Security and performance.

# 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 users' content, 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 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 used 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 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 in 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 on how to deploy.

Kubernetes has strong support from the community, and many people are available to help with issues. It is the most popular container orchestration platform. This is because 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 |

# Sizing recommendations

We have done a series of performance tests against our application to determine the bottleneck and environment limits in terms of resources.

We also analyzed real-world scenarios and application usages based on our production environment.

For specific detail on the matter, please see the "Load Test Collaboard" document.

Comparing performance test with real-world usage

On the performance test (run 2 and 3), we got 200 users performing one SignalR request every second, translating into 200 requests per second.

In the real world, we got an average of 20 actions per minute. Since one action is made of several requests, we can safely say that we got 200 requests per minute or 3.4 requests per second.

Based on these assumptions, we can say that an environment with the following sizing

## Base images

|  |  |  |
| --- | --- | --- |
| **Container name** | **vCPU** | **RAM (GB)** |
| cb\_proxy | 2 | 2 |
| cb\_frontend | 1 | 0.5 |
| cb\_api | 1 | 1.5 |
| cb\_auth | 1 | 0.5 |
| mft\_web | 1 | 1.5 |
| cb\_licensing | 1 | 0.5 |
| cb\_copyworker | 1 | 0.5 |
| cb\_worker1 | 1 | 0.5 |
| cb\_worker2 | 1 | 0.5 |
| mft\_cleanerservice | 1 | 0.5 |
| mft\_mergeservice | 1 | 1.5 |
| cb\_canvasshotter | 1 | 2 |
| cb\_fileconverter | 1 | 2 |
| mft\_storageoperationsservice | 1 | 0.5 |
| **Total** | 15 | 14.5 |

Note about the images: We are continually improving our product and also adding new features. The list of the images listed above may vary at the moment ot the product installation.

For the sizing recommendation below, you can find some hardware recommendations for the total system on which our application will run.

Hardware resources needed to the orchestrator are out of this document's scope; we will only report the capacity required by our system to support a determined number of users.

## Sizing recommendations 1-250 users

CPU: Single AMD EPYC 7302P (16 cores - 32 threads) or higher

RAM: 32GB or higher

## Sizing recommendations 250-500 users

CPU: Single AMD EPYC 7402P (24 cores - 48 threads) or higher

RAM: 64GB or higher

## Sizing recommendations 500-1000 users

CPU: Single AMD EPYC 7402P (24 cores - 48 threads) or higher

RAM: 128GB or higher

## Sizing recommendations 1000-2000 users

CPU: Single AMD EPYC 7542P (32 cores - 64 threads) or higher

RAM: 128 GB or higher

## Sizing recommendations 2000-5000 users

CPU: Dual AMD EPYC 7542P (32 cores - 64 threads) or higher

RAM: 128 GB or higher

## For all configurations

Network Interface: 10Gbps higher with appropriate internet connection speed

Storage speed: SATA Read Intensive 6Gbps or higher (for both the SQL Server data files and storage)

SQL Server minimum version: 2019 Standard with 10 CALL (device)