**Why I’m running Siebel on Docker**

It can take quite a bit of effort to spin up and keep a Siebel (lab) environment running with the latest Siebel CRM updates applied. To make it easier, I’ve been experimenting with Docker. In this article I’ll explain the main reasons why I’ve been running Siebel in Docker and I’ll discuss what Docker actually is. In the upcoming articles you’ll find more details on how to set up such a (lab) environment.

**Jaap Bouwsma**

**10 februari 2020**

[**Plaats reactie**](https://www.ebicus.com/en/why-im-running-siebel-on-docker/#commentaren)

Oracle releases Siebel CRM updates on a monthly basis. The importance of these updates is sometimes [underestimated](https://www.itpro.co.uk/security/vulnerability/354562/oracle-scolds-customers-for-ignoring-critical-patches). If you don’t require the latest functional enhancements for example, you might easily decide to skip some of these updates. From a security perspective, however, it’s critical to keep up  as these updates also include security patches for critical components like Java and Tomcat.

In order to make it easier to spin up an up-to-date Siebel (lab) environment with the latest Siebel CRM updates applied, I’ve been experimenting with Docker. In this article I’ll explain the main reasons why I’ve been running Siebel on Docker and I’ll discuss what Docker actually is. In the upcoming articles you’ll find more details on how to set up such a (lab) environment.

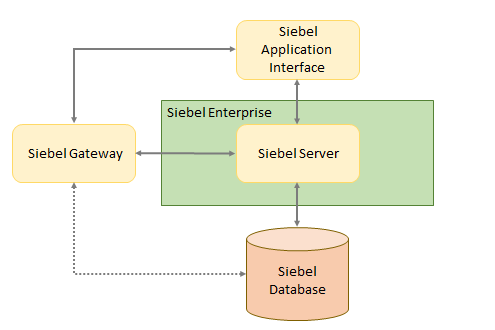
**Let’s start with why?**

Running a Siebel (lab) environment in Docker has several advantages compared to running it in traditional Virtual Machines:

* Less disk space consumption.
* Less memory and CPU utilization.
* Faster to start.
* More flexible to exchange, move and scale the environment across different platforms: on-premise, cloud and/or hybrid.
* Easier network setup.
* Easier to rebuild (parts of) an environment.
* Easier scaling of an environment.
* Easier orchestration.

**Siebel architecture**

Let me illustrate the statements above by having a closer look at the architecture of a basic Siebel CRM platform and mapping that onto a Virtual Machine and on Docker. First of all, Siebel’s basic CRM architecture consists of the following modules:

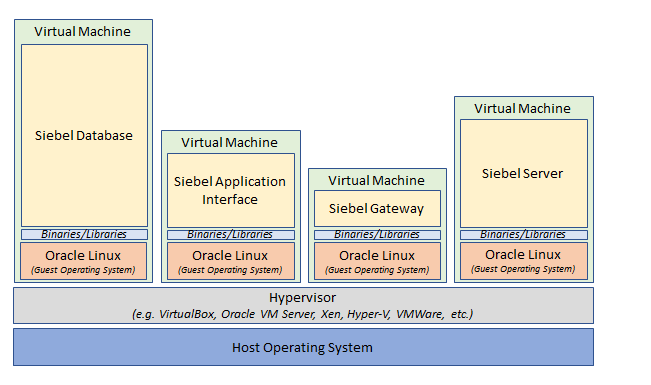


It’s good practice to install and run each module independently   from the other modules. This also allows for more flexible scaling of individual modules later on (like running multiple Siebel Servers or multiple Siebel Application Interface instances), regardless of whether you’re using Virtual Machines or Docker.

**Siebel on Virtual Machines**

In a traditional setup you would install Siebel onto Virtual Machines (or even on physical machines). In order to build and run the modules specified above, we would need at least four Virtual Machines (i.e. for the Siebel database, Siebel Application Interface, Siebel Gateway and Siebel Server).

When deploying a new environment, I prefer the “Oracle-on-Oracle” paradigm, hence I’m running the Oracle Siebel CRM modules specified above on Oracle Linux. As a result, each Virtual Machine has to be provisioned with an installation of Oracle Linux as the so-called Guest Operating System. On top of every Guest Operating System, a Siebel module has to be installed and configured:



To avoid the Virtual Machines growing too big, it’s recommended to start off by choosing “*Minimal Install*” during the installation of Oracle Linux, which only installs the basic functionality. A minimal installation, however, already consumes around 1.3 GB of diskspace regardless of the Siebel modules, which have to be installed on top of it. So, with four Virtual Machines, the (lab) environment already consumes 5.2 GB of diskspace even without having any of the Siebel modules installed.

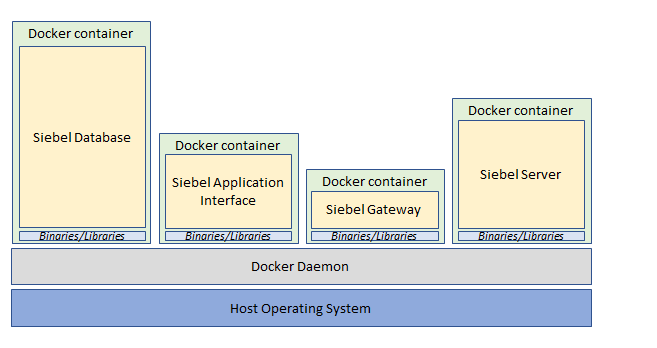
While running, each Virtual Machine also consumes a certain amount of memory and CPU utilization. This is not only consumed by the application components like the database and Siebel modules, but also by the Guest Operating System running inside each Virtual Machine. For just a small environment, there is already quite a lot of overhead.

**Siebel on Docker**

You might wonder, how is Docker different from Virtual Machines? A Virtual Machine talks to a virtual hardware layer which is being used by each Guest Operating System. Docker, on the other hand, is a container engine that doesn’t virtualize any hardware, as it uses the Linux Kernel features of the Host Operating System. So, the main difference between Docker Containers and Virtual Machines is that Virtual Machines are isolating systems, while containers are isolating applications.

On the official Docker website there is nice blog post from Mike Coleman titled “*Containers are not VMs*”, where he uses a great analogy by comparing houses (Virtual Machines) and an apartment building (Containers). In short, in his analogy, houses (the VMs) are fully self-contained and possess their own infrastructure like plumbing, heating, electrical, etc. (i.e. the virtualized hardware components), while the apartments (containers) inside the apartment building (Docker Host) share plumbing, heating, electrical, etc. (i.e. the shared resources from the Host Operating System).

As the application inside the container uses the Kernel of the Host Operating System, there is no need to load and boot a full operating system inside a container as is the case with a Virtual Machine. This allows a container to start up almost instantly and also results in lower memory and CPU utilization compared to a Virtual Machine. Without the need to include a full-blown Guest Operating system, the Docker container images can also be smaller, as illustrated below:



For very small microservice applications, this will make a huge difference comparing Virtual Machines and Docker images, where a Docker image could be even smaller than 100 MB. It should be noted, however, that the installation of a huge enterprise application like Siebel CRM still makes Docker images very bulky and unfortunately still results in Docker images of several Gigabytes. The illustrations above also show this by indicating the sizes of the different Siebel modules.

**Platform agnostic**

Nevertheless, Docker images are easier to exchange compared to VMs. While exchanging Virtual Machines, you have to take the appropriate format into account. For example, A VirtualBox image cannot directly run on a VMWare host or on the platform of a cloud provider without any additional conversions. Docker images, on the other hand, are supported by a wide range of both on-premise and cloud platforms, like:

* Oracle Container Engine for Kubernetes on Oracle’s Cloud Infrastructure (OCI)
* Azure Kubernetes Service (AKS)
* Amazon Elastic Container Service (Amazon ECS)
* Amazon Elastic Kubernetes Services (Amazon EKS)
* Google Kubernetes Engine (GKE)

This makes running a Siebel (lab) environment in Docker flexible and enables “*Lift and shift*”: the containers can easily be exchanged and run on-premise, in the cloud or in a hybrid solution.

**Easier network setup**

When using several Virtual Machines, a network and DNS configuration should be set up to allow the VMs to contact each other. With Docker, each container in a Docker network can immediately communicate with other containers in the same Docker network. Also, Docker’s embedded DNS server provides built-in service discovery for containers.

**Easier to rebuild (parts of) an environment**

Docker supports a special file format that contains all commands to assemble a Docker image, which is called a Dockerfile. With Dockerfiles, it is simple to (re)build a complete Siebel installation, which is very useful when Oracle releases a new Siebel CRM update.

**Easier scaling**

If additional instances of a Siebel Server have to be created, for example, Docker allows us to create more containers within seconds. The Docker Image with the Siebel Server installation simply serves as a blueprint, which can be used to instantly create as many containers from this blueprint as desired, each serving as a new instance. If we compare this to Virtual Machines, we would have to clone them first and have to take additional network configuration into account.

**Easier orchestration**

The orchestration of the different Siebel modules can easily be achieved by using a utility called Docker Compose. Using Docker Compose, the Docker containers running the different Siebel modules can easily be started in the right order. Also, more advanced orchestration tools like Docker Swarm or Kubernetes could be used to further enhance scaling, failover and high availability.

**Point of attention: containers should be disposable**

Apart from the many advantages listed above, there is one last important aspect you have to consider: Docker containers have an ephemeral character and should be considered disposable. For lab environments this is not such a big deal, but it is something to take into account when an environment evolves over time. On the other hand, this enforces the need to properly think about automating the provisioning and configuration of a Siebel environment, by building a complete CI/CD pipeline for example.

This concludes this article about why you would run a Siebel (lab) environment in Docker. The following articles in this series will focus on how to set up such a (lab) environment.