Study Case of Docker on Windows Machines

Fernando Cardoso Durier da Silva
{fernando.durier@hotmail.com, fernando.durier@uniriotec.br , fernando.durier@ibm.com}

May 4th 2018

Keywords:*[Virtualization, Container, Containerization, Hypervisor, Virtual Machine, Docker*]

Motivation:

Technology grows and evolves in a very fast pace and large scale for the sake of better development process. Therefore, we need to investigate and test those changes in order to be up to date and ready for new challenges in our lives as information technology professionals.

Objective:

This Study Case is a way of presenting the idea of docker, its main differences between its older “counterpart” (hypervisor), installation on windows and an example of running an application on a docker container.

1 Introduction:

1.1. Docker:

"Docker is a computer program that performs operating-system-level virtualization also known as containerization."[11]
Developed by Docker, Inc., it was primarily developed for Linux, using resource isolation features of the Linux kernel such as cgroups, control groups of linux that manage accounts limits over memory, CPU and other resources alike, and kernel namespaces, partitions of kernel for processes isolation,and union-capable file system to allow independent containers to run within a single Linux instance, avoiding the overhead of starting and maintaining of virtual machines.

1.2. DevOps:

"Historically, development and operations, and even testing, have been siloed operations. DevOps brings them together to improve agility and reduce the time needed to address customer feedback."[12]
So, we can understand DevOps as an alignment between development and operations teams. Sharing process, tools and responsibilities in order to fasten deliveries and generate more quality to the clients.
As agile methodologies brings development near business, the DevOps brings development near operations, thus bringing test and automation to the development process.

2 Bibliographic Review:

In this section we will discuss about the technology involved in this study case and the main concepts necessaries to its full understanding.

2.1. Virtualization:

“In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources.”[2]
Virtualization can be seen as an abstraction to make software behave like hardware, enefitting flexibility, cost, scalability, reliability, and sometimes overall capability and performance. From those simulations we get what we recognize as virtual machines, which originated from IBM mainframes in 1960s.

With virtualization the barriers of development and some of the OS restrictions were simply forgotten, as anything could run on an virtual machine, e.g. Linux simulations on Windows Machines for experimentation, education, security, etc.

The project proposed by [15], shows us that with this technique it is possible to port an entire operational system in a single server, but not only one of a kind, but instead a diverse number of operational systems at the same time.
As we could observe from that work, the main difficulties of this process is to emulate the resource handling and interfaces as they proved to be harder to emulate. However the emulation had excellent results at service offering and completion level.

2.2. Hypervisor:

Every resource management of virtual machine hardware is done by the Hypervisor or Virtual Machine Monitor. It is a software layer located in between the hardware and the operational system. Through isolation, partitioning and encapsulation, the hypervisor provides security to those virtual machines.

Virtualization is divided in two steps, being those paravirtualization and complete virtualization. On the complete one, the hypervisor emulate all hardware coming from the physical host, making the VMs running isolated from one another, in fact, creating an illusion that the machines are running in different proper hosts.
Yet in the complete virtualization scenario, we have to kinds of hypervisors, those being the bare-metal and the hosted.

The bare-metal interacts directly with the physical host hardware, being completely independent of the host's OS. On the other hand, the hosted runs on the physical host's OS, possible in any kind of OS.
This hosted type has an additional application layer on the hypervisor, both running on the same physical host OS, allowing file exchange between the virtual environment and the not

virtual one, and also allows the users to execute applications like browsers or email clients in parallel to the virtual environment. Different from bare-metal that has a full virtualization. Normally the servers that we are used to are virtualized using bare-metal, and the hosted solution is applied to local (not so serious) cases.

Now the paravirtualization makes the same thing as the complete, however, it makes a "real" hardware emulation, which means that the emulation will be exactly as good or bad as the physical host's hardware is not a software ideation, allowing a more flexible and realistic virtual machine. Given this scenario, now it is possible to perform different experiments with security, availability, storage efficiency, etc. against not only a fictional machine, but, also the hypervisor.

In the end the decision on to use a hypervisor and in which mode is oriented to your needings. It is up to you to choose between better performance through IO operations and accesses or more hardware compatibility and adaptability.

2.3. Container:

Every time developers face that problem of moving an application from its testing environment to the production one, because of dissonances between dependencies, OS, resources, etc. Causing not only a headache for them, but to the clients waiting at the other side of the screen and subsequently the organization owning that problem.

“Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. This could be from a developer's laptop to a test environment, from a staging environment into production, and perhaps from a physical machine in a data center to a virtual machine in a private or public cloud.”[5]

A container can be seen as a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment.[6]

So, their main goal is to guarantee software portability, modularity and independency through isolation and simplicity. And at the same time helping the DevOps process of an organization.

3 Comparison:

In this section we will present some discussion about the use of virtualization and containerization. How we can benefit of the new technique and how easy is to use it instead of the other one.

Containers and VMs have similar resource isolation and allocation benefits, however the main differences is in what is being virtualized. As VMs virtualize hardware, Containers virtualize operational systems, thus being more portable and efficient.

So basically we have this comparison, extracted from [6]:

Containers	Virtual Machines
Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.	Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

Table 1: Comparison

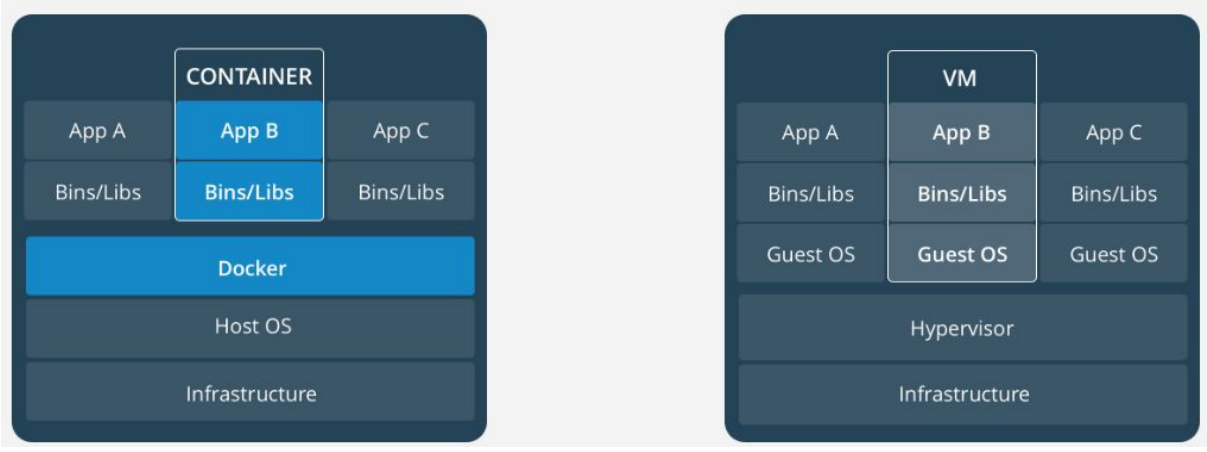

Image 1: Illustrating the comparison, again extracted from [6]

4 Installing docker on Windows:

This is the setup section of this study case. Here we will learn to configure the main Docker environment needed in Windows OS machines, a challenge for most of developers.

4.1. Initial considerations:

Since Docker was designed to operate on Linux, to run it on Windows somethings may be needed to do.
Firstly, be certified that your machine is enabled to perform virtualization. In general, it is not and it may be necessary to reboot the machine, access its BIOS, look for virtualization options and check it. Before even doing that is interesting to know if your machine runs on a 64-bit architecture as it will be necessary for virtualization.
Second, as we are going to perform experimentations for educational purposes we will use Virtual Machines hosted, not bare-metal. And so, we will need to provide an adequate environment to run this experiment, that means, be sure to have more than enough memory, as we are going to create Containers inside a Virtual Machine, ensure that we are currently running only needed process in our physical machines, etc.

4.2. Downloading the Docker for Windows or the Docker Toolbox:

After the initial considerations were checked, we will proceed to install either Docker for Windows or Docker Toolbox distributions. Being Docker for Windows ideal for Windows OS versions above 8th version and Docker Toolbox for versions below the 7th one.

4.2.1. Docker for Windows:

It is an integrated, easy-to-deploy development environment for building, debugging and testing Docker apps on a Windows PC. Being a native app deeply integrated with Hyper-V virtualization, networking and file system, making it the fastest and most reliable Docker environment for Windows. It's only restriction is that of being only available for most recent versions of Windows like Windows 10 or above. The installation of the Docker for Windows app is much like the installation of any of those in Windows system. It is simple, a set of next next guided by an intuitive GUI.

4.2.2. Docker Toolbox:

Since the study object is a Windows 7 machine, Docker for Windows won’t function as it should, in fact, will be downloadable, but, impossible to install by normal means.
As an alternative way, Docker gave us Docker Toolbox.

It is an installer for quick setup and launch of a Docker environment on older Mac and Windows systems that do not meet the requirements of new apps.
The toolbox comes with a set of Docker tools and auxiliary tools for virtualization, those being: Docker Machine for running "docker-machine" commands, a Docker Engine (like the

Docker Daemon) for running "docker" commands, Docker Compose for running the docker-compose commands, Kitematic being the Docker GUI (although, never felt the necessity to use it), a shell preconfigured for a Docker command-line environment and the virtualization software (our hypervisor), Oracle VirtualBox.

By accessing https://download.docker.com/win/stable/DockerToolbox.exe, the download of docker toolbox shall get started.
When over, click on the icon on the browser or go directly to the downloads folder and execute the recent downloaded file. The installation process will begin and a set of "next button" windows will appear. Click next till the installation ends, and only change the default settings if you really know what are you doing.
Done that, in the Program Files folder there will be the docker toolbox folder and within there will be all the aforementioned features included by Docker Toolbox inside it. And the docker engine, docker-machine, docker gui and virtualization software will be there as well.
Although, Docker Toolbox has a gui and a terminal of its own, we will check if it our installation was a success using an alternative to the official Docker guide. But it will be shown in configuration section.

4.3. Installing the VirtualBox:

We can use any Hypervisor software we want, however, due to documentation and community recommendations I chose to use the VirtualBox as my hypervisor. Hyper-V from Windows 8 and Windows 10 could also be used, but, since the machine used in this example was a Windows 7 one. By using Docker Toolbox, it already installs the VirtualBox, as it assumes you doesn't have another Hypervisor already installed like Hyper-v on Windows more recent versions. It can be found on program files folder either on Oracle or docker-machine folders.

Its installations is very simplistic and doesn't require much attention, only if you really need to alter its core settings for ultimate performance, but if in an educational scenario, just follow the next sequence filling whichever information is required by the installer.

Done that, we can begin the configuration of the Docker environment.

4.4. Configuring the Docker environment:

Given that Docker Toolbox finished its installation, we will begin configuring the docker-machine environment. This is a very important process, as it will be repeatedly used during Docker assisted Development.

1. Open powershell as an admin (that means, with elevated privileges).

2. Issue against the Powershell the command “docker-machine version”.
   It should return the version and build of the docker-machine executable in your machine. If something went wrong till now, return to the installation step.

3. Now that we are sure that docker-machine is well installed and allocated in our machine/host, we will begin to its environment configuration.
   Issue the command “*docker-machine ls*”. The PowerShell should return an empty response with headers of NAME, ACTIVE, DRIVER, STATE, URL, SWARM, DOCKER and ERRORS. These are the main status informed by CLI when operating the docker-machine environment.
   It is expected to them to be empty as we hadn’t yet created any docker-machine.

4. Now to create our first docker-machine, we will use its syntax that is very intuitive and relatively simple.
   Issue the command “ *docker-machine --native-ssh create -d virtualbox --engine-insecure-registry "localhost:5000" default* ”.
   But, what that huge string does ? Well it creates a docker-machine called default, which will have a native ssh for minimal security, will run using VirtualBox as its drive (remember that we need that a Windows have a kernel from a Linux), this docker-machine also will have a insecure-registry allocated in localhost:5000, what that means is that this docker environment will be able to access this localhost:5000 without security issues. That last specification will be useful for later purposes of local operations of versioning, resource sharing, etc.

5. Ok, now we have a docker-machine environment up, but yet, not running. Issue the command of “*docker-machine ls*” once again and we will see that the command line shows us the default machine created, but not active. So, in order to make it active, some settings must be done like environment variables setting, ip definition, etc. Calm down, we don’t need to manually configure those, we just need to issue the command “*docker-machine env*”. It will show us which are the environment variables and their values to be set, and will show a simpler command to do everything and set our default machine running. In PowerShell that simple command is “*& '<in your machine, the location of docker-machine exec file>\docker-machine.exe' env | Invoke-Expression*”. For cmd users, the command is quite different and bigger.

6. Now, by issuing the "*docker-machine ls*" command once again, we will be able to see not only the state of RUNNING and marking of active, but also , the IP of docker-machine as well. Well, we are ready to start developing with Docker containers.

7. Disclaimer: The docker-machine may get some errors from installation, in any case where this happens it is necessary to redownload just it. So for this, I would recommend to use chocolatey (the apt-get for Windows) in order make its download.

4.5. Final considerations:

Well, now that we are ready to build even more containers and better our development process, it is needed to say that, as we are using much resources present in our physical host machine, it may be necessary to manually manage some idle data, idle container, idle project in order to not exhaust the physical host. Remember that you are not only virtualizing a operational system but an entire hardware as well, even if the hypervisor is in hosted mode.

5 Example:

In this section we will illustrate how to run an image to our container environment managed by Docker and its commands.

5.1. The application:

Lets try to make a container able to host any application as our physical machine does.
First we will need to understand the Dockerfile, that is a set of instructions arranged in a script that will build our container.
Go to a folder of your choice and create a file named Dockerfile, that won't have any proper termination, so it won't be a txt, nor yml, etc.

Open it on your favorite editor and type the below commands:

```
FROM ubuntu:14.04
# Exposing Ports
EXPOSE 5035
RUN apt-get update -y
# Install packages
RUN apt-get install -y curl
RUN apt-get install -y postgresql
RUN apt-get install -y postgresql-client
# Remove apt cache to make the image smaller
RUN rm -rf /var/lib/apt/lists/*
CMD bash
```

Basically this will emulate the ubuntu OS and its functions, a very general development environment. It will be exposed on port 5035 of our docker-machine environment. And will have already the needed dependencies to access postgresql.

Now to build your docker issue the command "*docker build -t docker-example ./*", it means build a container named docker-example based on the Dockerfile located in this folder. Then we go for docker run in "*docker run -td docker-example*", meaning that it will run on background the ubuntu emulation you just created.

5.2. The adequate docker image:

The concept of a container is to be simple, minimalistic and portable. So there are some set of simplistic images to choose from the DockerHub, like nodejs and alpine that offer ideal runtime for their languages and are very small in size. There are other ways to minimize the size of your container.

5.3. The dockerfile and .dockerignore:

Like github we can make docker push only what is really necessary to the docker daemon/engine. We do this by creating a dockerignore file, much similar of what we do with gitignore and inform what we want to push or not.

6 Conclusions:

Although hard, the use of containers in our day-by-day development routine has proved to be very handy, as helped with understanding and applying of the DevOps process, gave a more elegant appearance to our projects and helped us to further understand what is happening post developing time.

7 Main Difficulties:

The main difficulties found in the process of making of this study case were the series of incompatibilities between older Windows systems and Docker, as many errors were provoked either by excessive computational resource depletion or by limitations of Windows itself like the necessity of a virtual machine for older versions. However, the community is very present to help in such moments of doubt and uncertainty.

References:

[1] Container docker: o que é e quais são as vantagens de usar? - https://www.meupositivo.com.br/panoramapositivo/container-docker/
[2] Virtualization Concept and Definition - https://pt.wikipedia.org/wiki/Virtualiza%C3%A7%C3%A3o
[3] Docker Concept and Definition - https://pt.wikipedia.org/wiki/Docker_(software)
[4] What is Docker ? - https://www.mundodocker.com.br/o-que-e-docker

[5] What are containers and why do you need them ? - https://www.cio.com/article/2924995/software/what-are-containers-and-why-do-you-need-them.html
[6] What is a container ? - https://www.docker.com/what-container
[7] O que é virtualização ? - https://www.tecmundo.com.br/web/1624-o-que-e-virtualizacao-.html
[8] Docker Toolbox Installation - https://docs.docker.com/toolbox/toolbox_install_windows/
[9] DevOps Concept and Definition - https://pt.wikipedia.org/wiki/DevOps
[10] O que é DevOps - https://www.ibm.com/developerworks/community/blogs/rationalbrasil/entry/o_que_devops?lang=ens
[11] O'Gara, Maureen (26 July 2013). "Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud". SYS-CON Media. Retrieved 2013-08-09.
[12] DevOps - https://www.ibm.com/cloud/devops
[13] Virtualization - https://www.techopedia.com/definition/719/virtualization
[14] What is Virtualization - https://www.networkworld.com/article/3234795/virtualization/what-is-virtualization-definition-virtual-machine-hypervisor.html
[15] BARHAM, Paul et al. Xen and the art of virtualization. In: **ACM SIGOPS operating systems review**. ACM, 2003. p. 164-177.
[16] Hypervisor segurança em ambientes virtualizados - https://www.devmedia.com.br/hypervisor-seguranca-em-ambientes-virtualizados/30993
[17] Docker for Windows - https://www.docker.com/docker-windows
[18] Docker Toolbox Overview - https://docs.docker.com/toolbox/overview/