B.TECH. PROJECT REPORT

On

Docker for SUSE Migration

BY

SUNAND AGARWAL,

160001057

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

November 2019

Docker for SUSE Migration

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degree

of
BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:
SUNAND AGARWAL,
160001057

Guided by:
Dr. Gourinath Banda,
Associate Professor,
Computer Science and Engineering,
IIT Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
November 2019

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Docker for SUSE Migration"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in '**Computer Science and Engineering**' completed under the supervision of **Dr. Gourinath Banda,** Associate Professor, Computer Science and Engineering**,** IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Sunand Agarwal

CERTIFICATE by BTP Guide(s)

It is certified that the above statement made by the student is correct to the best of my knowledge.

Dr. Gourinath Banda,
Associate Professor,
Computer Science and Engineering,
IIT Indore

Preface

This report on "**Docker for SUSE Migration**" is prepared under the guidance of **Dr. Gourinath Banda**, Associate Professor, Computer Science and Engineering, IIT Indore.

The main objective of any student is to get as much practical knowledge as possible. Being able to have practical knowledge by developing a project is a lifetime experience. As practical knowledge is as necessary as theoretical knowledge, we are thankful for working on a project.

Through this project report we have tried to give a basic overview of how the GE Senographe Pristina Mammography System works and mainly focuses on the containerization and deployment aspects of the reconstruction software application associated with the product.

We have tried to the best of our abilities and knowledge to explain the content lucidly. We have also added models and figures to make it more illustrative.

Through the development of the project we had a great insight into various approaches that can be applied in the development of the product. This project is a stepping stone for my career.

We are pleased to demonstrate this project. Proper care has been taken while organizing the project so that it can be comprehended. Also, various software engineering principles have been implemented.

Sunand Agarwal
B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Acknowledgments

The success and final result of this project required a lot of guidance and advice from many people and I am incredibly privileged to have got this all along the duration of my project. All that I have done is only due to such supervision and advice and I would not forget to thank them.

I respect and thank **Dr. Gourinath Banda**, for providing me an opportunity to do the project work and giving me the kind support and valuable guidance which made me complete the project duly. I am incredibly thankful to him for providing such excellent support and advice, although he had a busy schedule managing other projects as well. This project was done as part of my internship at **GE Healthcare**.

I would not forget to remember his name for his encouragement and moreover for his timely support and leadership until the completion of our project work. It is his help and support, due to which I became able to complete the design and technical report.

I owe my sincere gratitude to our Head of Department of CSE, **Dr. Surya Prakash**, who took a keen interest in the project work and supervised us all along, till the completion of our project work by giving all the necessary information for developing a sound system.

I am thankful for and fortunate enough to get constant encouragement, support and guidance from all the teaching staff of CSE which helped us in successfully completing our project work. Also, I would like to extend our sincere esteems to all staff in the department for their timely support.

Without their exceptional support this report would not have been possible.

Sunand Agarwal
B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Abstract

The healthcare industry is a collection and integration of divisions within the commercial system that provides goods and aids to treat patients with remedial, precautionary, rehabilitative, and palliative care. It is one of the world's largest and fastest-growing industries.

Learning that you may have cancer plunges you into uncertainty. The more you understand about your condition, the higher your sense of control. Breast cancer patients face challenges throughout the journey of examination, surgery, post-treatment, and rehabilitation. The breast cancer patient is treated by a multidisciplinary team including doctors, nurses, therapists, mentors, and psychologists.

Mammography is the method of using low-energy X-rays to test the human breast for examination and screening. The aim of mammography is the early discovery of breast cancer, typically through the exploration of specific masses or microcalcifications. Mammography has been around since the 1960s when French engineers Jean Bens and Emile Gabbay produced a breast-dedicated X-ray machine with a specialized X-ray tube that released low-energy radiation and still let doctors to see breast tissue in more comprehensive detail.

GE Healthcare announced the launch of its brand-new **Senographe Pristina** mammography system, intended to make breast screening more convenient and encouraging for patients. Senographe Pristina mammography system was designed to relieve anxiety when the patient enters the exam room. Widespread mammography systems compress the breast automatically, which can be the cause of significant trouble. Senographe Pristina highlights a self-compression tool that assists women with a sensation of control by allowing them to adjust the level of breast compression manually. Under the administration of a technologist, the patient can set compression to a level that seems right for them.

This paper aims to provide a summary of how the Senographe Pristina works, basically the reconstruction software. The project mainly works as a proof of concept to know if and up to what extent can the above software be containerized and deployed on the Senographe Pristina in the hospitals. This would help the radiologists to examine the patients' X-Ray images, and would bring down the overall cost of the system.

Table of Contents

Introduction

GE Healthcare launched the Senographe Pristina Mammography System, which is purposefully created to help diminish pain, discomfort and stress for patients to enhance the mammography experience.

Now, the System currently works on a Dual PC Setup, where one PC, called the **Mammo Host PC**, is used to take 3-Dimensional acquisitions from the Senographe Pristina. The second PC, called the **Reconstruction PC**, is used to convert those 3-Dimensional acquisitions to 3-Dimensional Volumes. These volumes are now pushed to the Radiologists' systems for further examination. Multiple sets of Dual PCs are sold to the hospitals in pairs along with the Senographe Pristina Systems. They are marketed in pairs since they are connected via Ethernet.

The problem definition states that we want to containerize the software running in the Reconstruction PC thereby removing the physical connection between both the PCs and hence deploy that software in a single system. Now we would need a single system for multiple Mammo Host PCs, thereby reducing the overall cost for the company as well as for the customers (hospitals).

The technology (or tool) used to containerize and deploy software is **Docker**. We first remove the physical connection between the two PCs, create a Docker image of the reconstruction software and deploy that image in a new system and establish a connection between the Mammo Host PC and this system.

What is the Purpose?

Currently the operating system on which both sets of PCs are running is based on **Red Hat Enterprise Linux 6.10 (RHEL 6.10).** GE wants to migrate to another operating system called **SUSE Linux Enterprise Server 15 (SLES 15)**. They want all their systems to be running on SLES 15 in the coming years, and this project is just the start of a long journey. One of the reasons the migration is happening is because software updates and releases have now been stopped for the current operating system.

Now what would happen is we would create the Docker image of the reconstruction software based on a stripped-down version of the RHEL 6.10 operating system, and then we would deploy that docker image on a system running on SLES 15. This would seem like the software is running on SLES 15 itself.

Tools Used

What is Docker?

Docker has achieved tremendous popularity in this fast-growing IT industry. Corporations are continuously adopting Docker in their production environment.

Before containerization came into the picture, the principal way to confine, organize applications and their dependencies were to place every application in its virtual machine. These machines run various applications on the same physical hardware, and this method is nothing but virtualization.

But virtualization had few disadvantages such as the virtual machines were heavy in size, running various virtual machines lead to volatile performance, the boot-up process would typically take a long time and VM's would not solve the difficulties like portability, software updates, or continuous integration and continuous delivery.

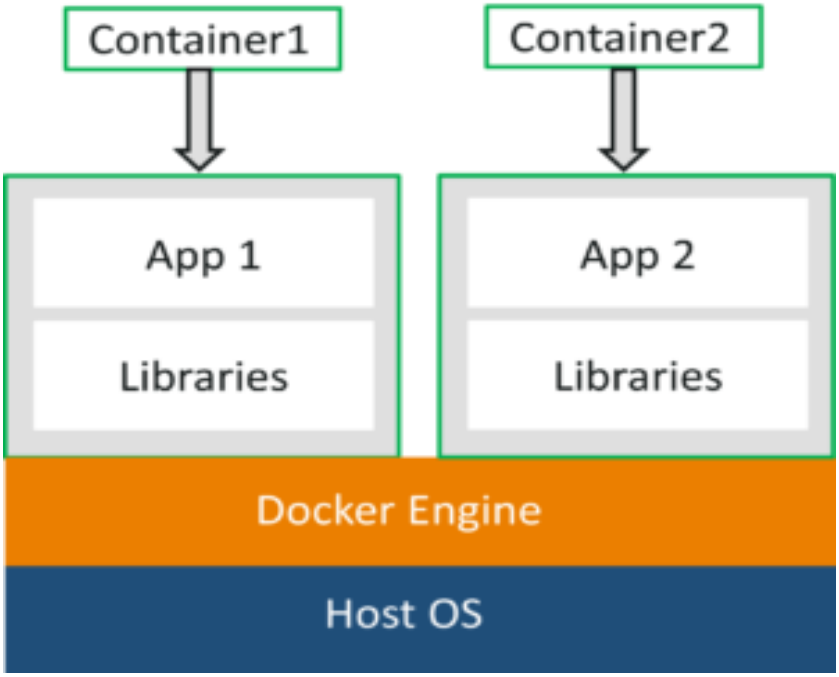
These shortcomings led to the emergence of a new technique called Containerization. Containerization is a kind of virtualization that brings virtualization to the operating system level. While the latter brings abstraction to the hardware, containerization brings abstraction to the operating system.
Hence, Docker is a platform that packages an application and all its dependencies collectively in the form of containers. This containerization character of Docker ensures that the application runs in any environment.

Basic Docker Terminology

- An **Image** is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, system dependencies, system libraries, etc.
- A **Container** is an instantiation of an image that we have brought to life. We can have multiple copies of the same image running.
- A **Dockerfile** is basically a recipe for creating an image.
- **Dockerhub / Image Registry** is a place where people can post public (or private) docker images to facilitate collaboration and sharing.

As you can see in the picture, each and every application runs on separate containers and has its own collection of dependencies and libraries. This makes certain that each application is free of other applications, giving developers a guaranty that they can build applications that will not conflict with one another.

So a developer can create a container having numerous applications installed on it and provide it to some other team. Then that team would only need to operate the container to replicate the developer's environment.



Why Docker?

So why does everyone love containers and Docker?

- **Ease of use:** A massive part of Docker's fame is how easy it is to use. Docker can be learned quickly, primarily due to the numerous resources available to learn how to create and manage containers.

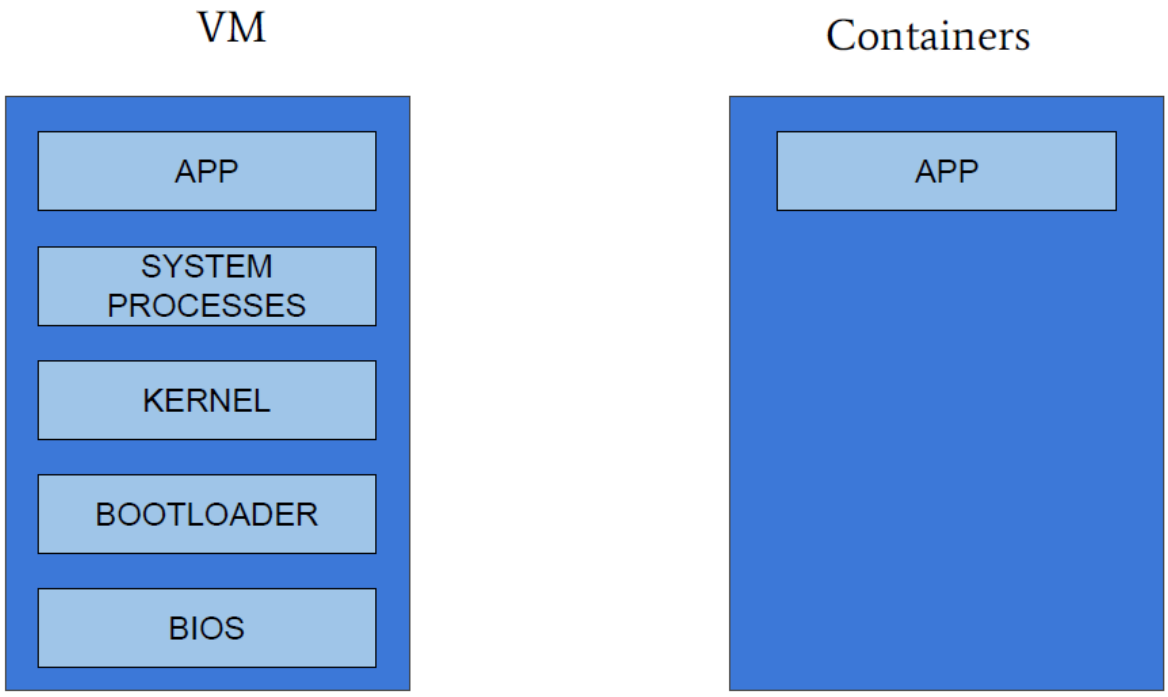
- **Reproducibility:** Reproducibility not only facilitates peer review, but ensures the model, application or analysis we build can run without friction which makes our deliverables more robust and withstands the test of time. For example, if we built a model in Python it is often not enough to just run pip-freeze and send the resulting requirements.txt file to our co-worker as that will only encapsulate Python explicit dependencies — whereas there are often dependencies that live outside Python such as operating systems, drivers, compilers, configuration files or other things that are required for our code to run successfully. Even if we can get away with sharing Python

dependencies, enveloping everything in a Docker container diminishes the burden on others of recreating our environment and makes our work more accessible.

- **Better software delivery:** Software distribution using containers can also be more productive. Containers are movable. They are also entirely independent. Containers include a separated disk volume. The volume goes with the container as it is produced and deployed to multiple environments. The software dependencies (libraries, binaries, etc.) ship with the docker container. If a container works on your device, it will run the same way in a development, staging, and production environment. Containers can eliminate the configuration variance difficulties common when deploying binaries or raw code.

- **Software-defined networking:** Docker supports software-defined networking. The Docker Engine and CLI allow administrators to define independent networks for its containers, without having to involve a single router. Developers and operators can create systems with complicated network topologies and ascertain the networks in configuration files. This is a security advantage, as well. An application's containers can run in a detached virtual network, with tightly-controlled entrance and exit paths.

Why Docker Instead of Virtual Machines? What is the big deal?



Getting down to the core, Docker allows applications to be divided into docker containers with instructions for precisely what they need to last that can be easily ported from system to system. Virtual machines also allow the exact same point, and various other accessories already exist to make reconstructing these configurations transportable and reproducible.

While Docker has a more simple structure as compared to VMs, the real area where it causes the disorder is resource performance.

If you have 20 Docker containers that you want to run, you can run them all on a single virtual machine. To run 20 virtual machines, you've got to boot 20 operating systems with at least minimum resource specifications available before factoring the Hypervisor for them to run on with the base operating system.

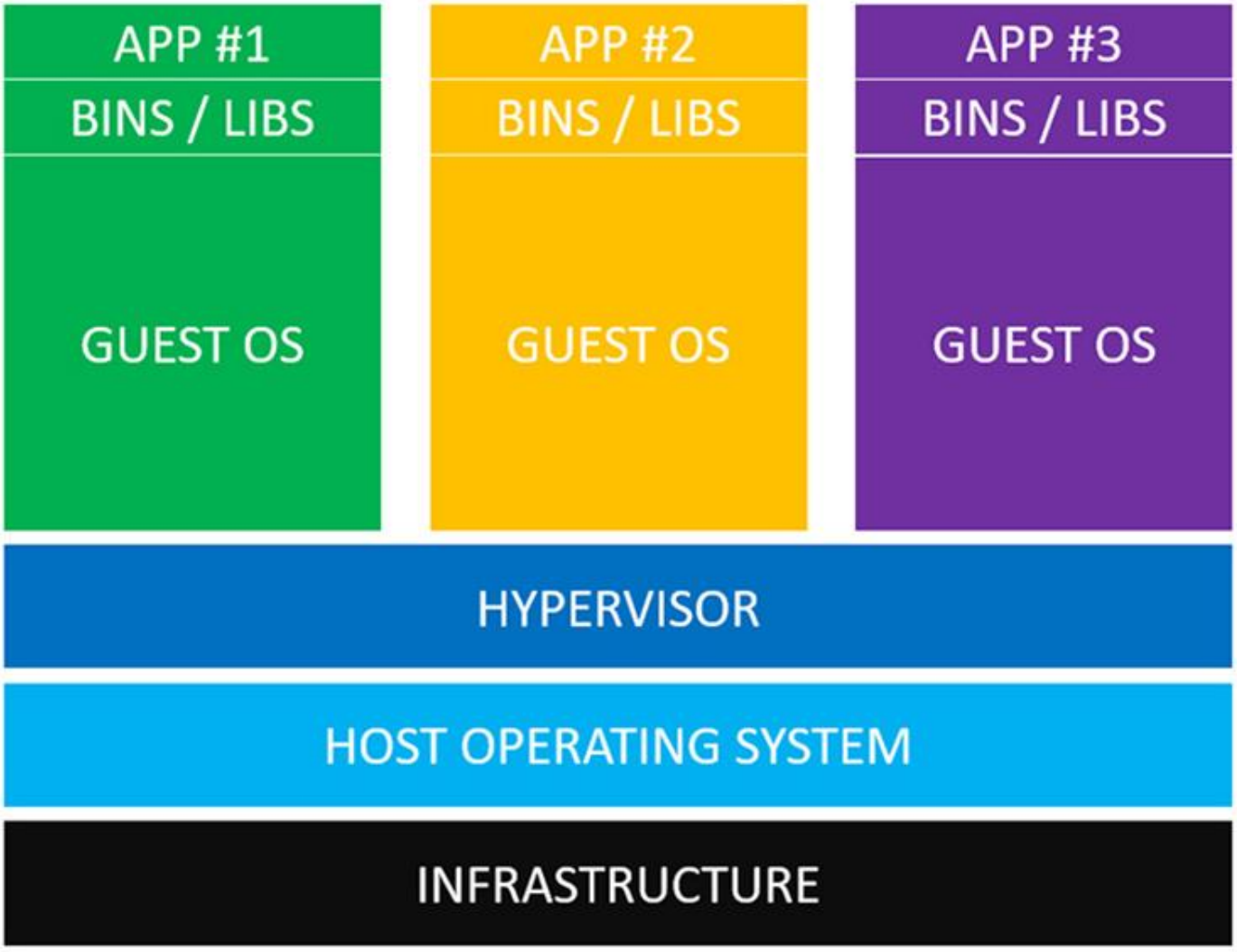
Let's say we're going to go with a minimum of 256MB VM we'd be looking at 7.5GB of RAM with 20 different OS kernels managing resources. With Docker we could allocate a chunk of RAM to one virtual machine and have a single operating system managing those competing resources… and we could do all of that on the base operating system without a costly hypervisor needing to be involved at all.
A pivotal point to keep in mind is that Docker can do what it does due to strong integration with the Linux kernel. It makes for significant performance at a low level, and as a result, Docker is not (currently) a replacement for VMs for Windows, OS X, etc. When running Docker containers on a non-Linux computer such as above, they will be run inside of a virtual machine using boot2docker.

Those types of performance improvements are on par with cloud provider services like Amazon and others getting a 26 to 1 performance gain on the VMs that they sell per hour. It's a massive enabler for their markets because we're suddenly capable of doing a lot more for the same amount. Instead of needing to buy 2 virtual machines (for load balancing) for every isolated application that you need to deploy, we could cluster three larger VMs and deploy all of them to it, actual processor limits aside.

Docker vs VM

Understanding VMs

Here's how it looks like to run a few applications on a system using virtual machines:



- **It all begins with some infrastructure.** This could be our laptop, a server running in a data center, or a private server that we're using in the cloud.

- **On top of that host runs a base operating system.** On our laptop this will likely be Windows, MacOS or some division of Linux. When we're discussing VMs, this is generally designated as the host OS.

- **Then we have something called a Hypervisor.** We can think of virtual machines as a self-contained system gathered into a single file, but something needs to be able to operate that file. Modern type 1 hypervisors are HyperKit for MacOS, KVM for Linux and Hyper-V for Windows. Popular type 2 hypervisors are VMWare and VirtualBox.

- **The next layer is our guest operating systems.** Let's say we wanted to run 3 applications on our server in total isolation. That would require spinning up three guest operating systems, which are all controlled by our hypervisor.

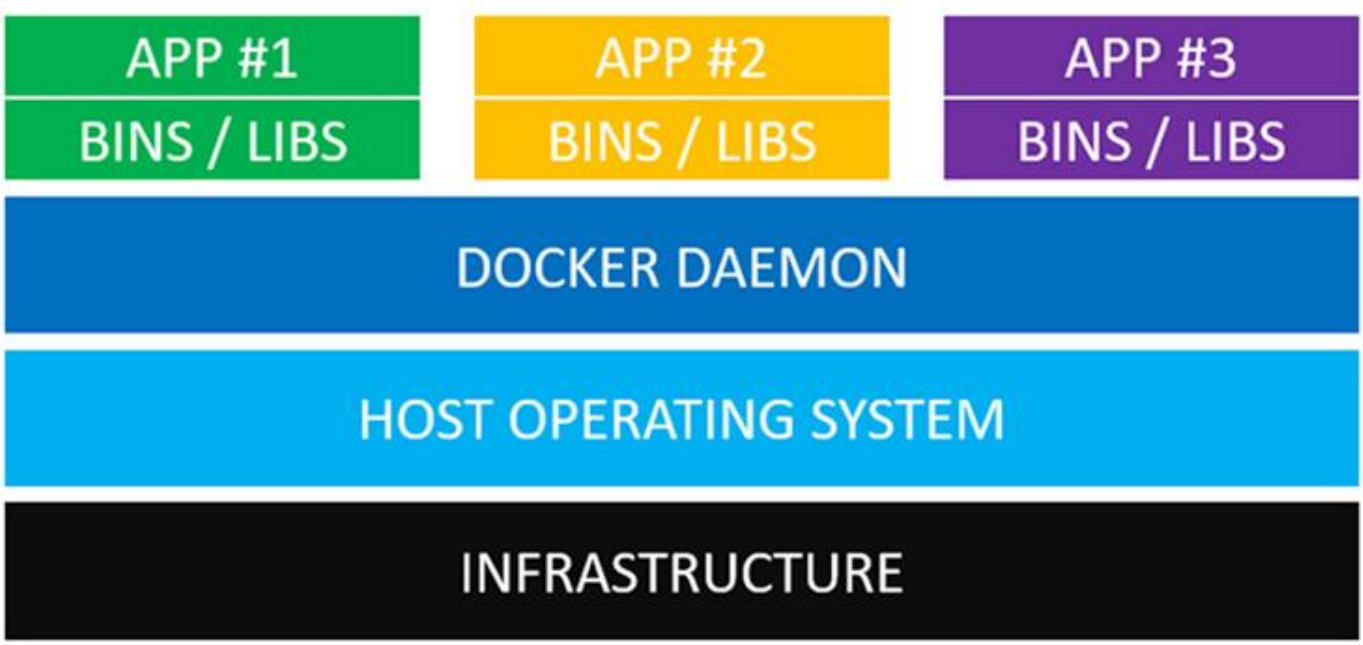
  Virtual machines come with a lot of equipment. Each guest OS in itself might be 700MB. That means we're using 2.1 GB of disk space just for our guest operating systems. It gets more serious too because each guest OS needs its own CPU and memory resources also.

- **Then on top of that, each guest operating system needs its own copy of many binaries and libraries** to lay the groundwork down for whatever our application needs to run. For example, we might need libpq-dev installed so that our web application's library for connecting to PostgreSQL can connect to our PostgreSQL database. If we're using something like Ruby, then we would need to install our gems. Likewise with Python or NodeJS we would install our packages. Just about every primary programming language has its own package manager. Since each application is diverse, it is assumed that each application would have its own collection of library requirements.

- **Finally, we have our app.** This is actually the source code for the awesome application we've created. If we want each app to be isolated, we will need to run each one inside of its own guest operating system.

Analyzing Docker Containers

Here's how the same set up looks like using docker containers instead:



- **We still need some infrastructure to run Docker containers.** Like VMs, this could be our laptop or a server system somewhere out there in the cloud.

- **Then, we have our host operating system.** This could be anything we want that is capable of operating Docker. All significant divisions of Linux are supported and there are alternatives to run Docker on MacOS and Windows too.

- **The Docker daemon/engine replaces the hypervisor in this case.** The Docker daemon is a service that operates in the background on our host operating system and manages everything needed to run and communicate with Docker containers.

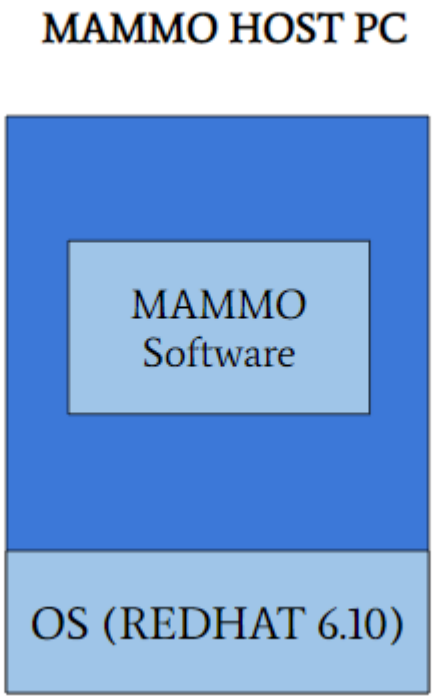
- **Next up we have our executables and libraries**, just like we do on virtual machines. Instead of them running on a guest OS, they get built into individual units called Docker images. Then the Docker daemon runs those Docker Images.

- **The last bit of the puzzle is our applications.** Each one would end up residing in its own Docker image and will be handled separately by the Docker daemon. Typically each application and its libraries and dependencies get collected into the same Docker image. As we can see, each application is still separated.

Docker Adoption and Product

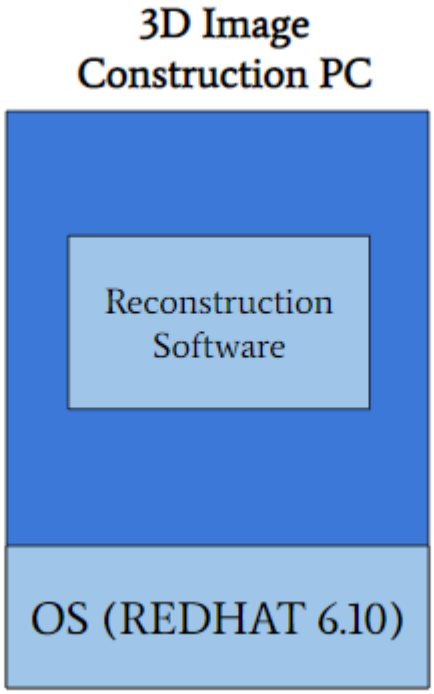
There are basically two PCs behind the working of the Senographe Pristina, called the **Dual PC Setup**.

1. **Mammo Host PC**



The first system is the **Mammo Host PC**, on which the 2D images are stored after taking the patient's exam. The underlying operating system this machine runs on is **RHEL 6.10**.

2. **Reconstruction PC**



The second system is the **3D Image Reconstruction PC**, on which a reconstruction software runs and converts those 2D X-Ray images on Mammo machine to 3D Volumes. The underlying operating system on which this system runs on is also **RHEL 6.10**.

How does the Reconstruction PC receive those 2D images?

The answer is simple, via a network. Following the network protocol, the Reconstruction PC acts as a Server and the Mammo Host PC as a client.

This client/server application lets a computer user show and optionally save and update files on a remote machine as though they were part of the user's computer. This protocol allows the user or system administrator to mount all or a portion of a file system on a server. The portion of the filesystem that is mounted can be accessed by clients with whatever privileges are allocated to each file (read-only or read-write). It uses Remote Procedure Calls (RPCs) to map requests between servers and clients.

So, what is the whole procedure of reconstruction?

We take a patient's exam, i.e., 2D X-Ray images which are stored in some directory of the Mammo Host PC. These images are then transferred via a network protocol to the Reconstruction PC, whereby a reconstruction algorithm runs and converts those 2D images to

3D volumes. These 3D volumes are stored in some other directory of this Reconstruction PC and then pushed to the radiologists' system for further examination of the patient.

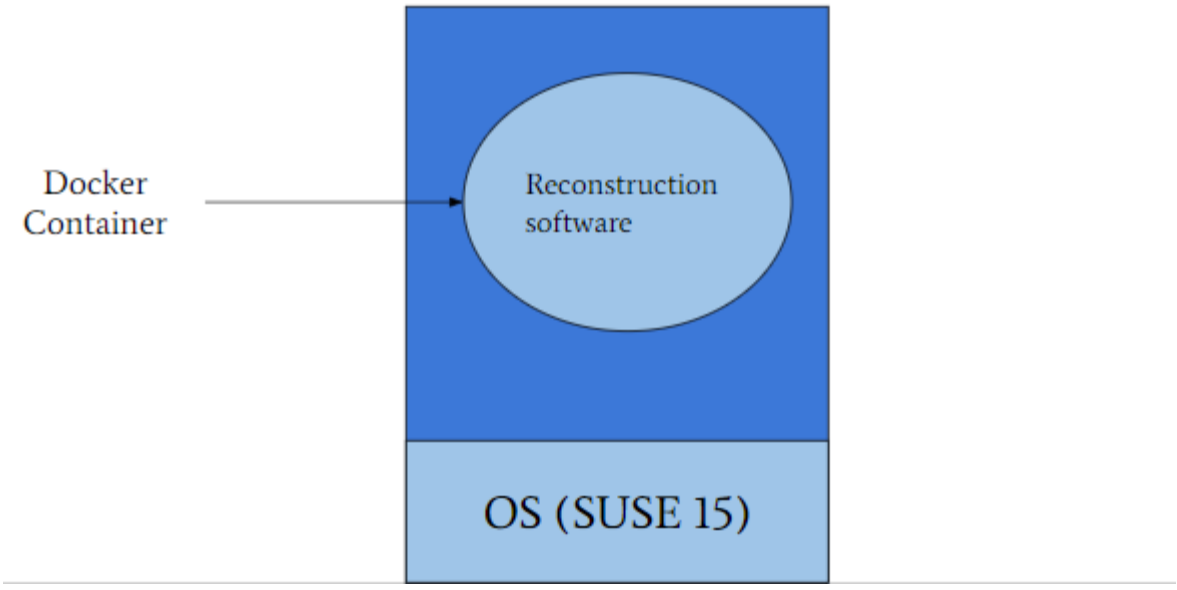
What is the proposal?

What we want to prove using Docker is that the reconstruction software can be made into a Docker Image, thereby removing the need for a whole extra set of Reconstruction PC.

Since we have to migrate from RHEL 6.10 to SLES 15, what is done in this project is that we have a new PC, running on SLES 15, on which we will deploy our docker image. We will establish a connection between the Mammo Host PC and this docker container so that the reconstruction happens as before.

The difference now is that the software seems to run on a SLES 15 machine, though the underlying base docker image of the container is still RHEL 6.10. This new PC would serve as a reconstruction PC for multiple Mammo Host PCs, by running multiple instances of the docker image. This would also save a lot of money for the company as well as the customers.

So, the new machine would look like this:

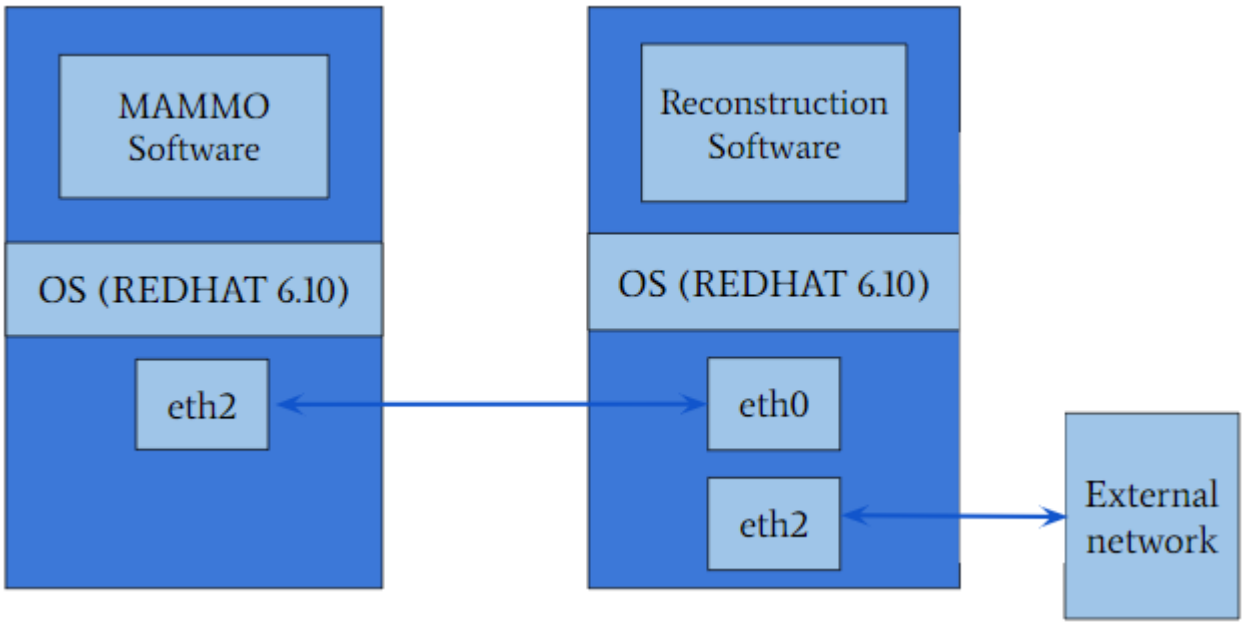


Current Dual PC Setup

As of now, the Mammo Host PC, as well as the Reconstruction PC, are running on RHEL 6.10. The networking part is as follows:

- The **eth2** port of Mammo Host PC is connected via Ethernet to the **eth0** port of the Reconstruction PC.
- The **eth2** port of the Reconstruction PC is connected to the external LAN, hence it is available in the local network.

This means that the Mammo Host PC is not connected to the outside world and can only be accessed using the corresponding Reconstruction PC. This is achieved using static IPs. Therefore, this inseparable pair of PCs together is sold to the hospitals because of the intermediate connection. This is very costly.



Containerization and Deployment

Containerization

Now we have to containerize the reconstruction software running on the Reconstruction PC. This is done by taking an underlying base image of RedHat Linux 6.10 and installing all the necessary packages to install the reconstruction software. So, we pull a rhel6.10 docker base image and install the packages. Now we run the installation scripts of the software which will install all the necessary RPMs, like software packages, antivirus, etc. Now we will commit the docker image to save our changes and then push it to the Docker Hub for future deployment.

Docker hosts public repositories called the Docker Registry (Hub) where we can find a list of public Docker images for our use. While the Docker Hub plays an essential role in providing public visibility to our Docker images and for we to utilize quality Docker images put up by others, there is a clear need to set up our private registry too for our team/organization.

We can use Docker Hub or host a local Docker registry for storing our image.

Image Deployment

For the deployment of the image on our SLES 15 machine, we follow the steps below;

1. **Pulling the Image**

The first step in deployment is to pull the docker image to our system, which we've created earlier. To download a specific image, or set of images (i.e., a repository), use **docker pull**.

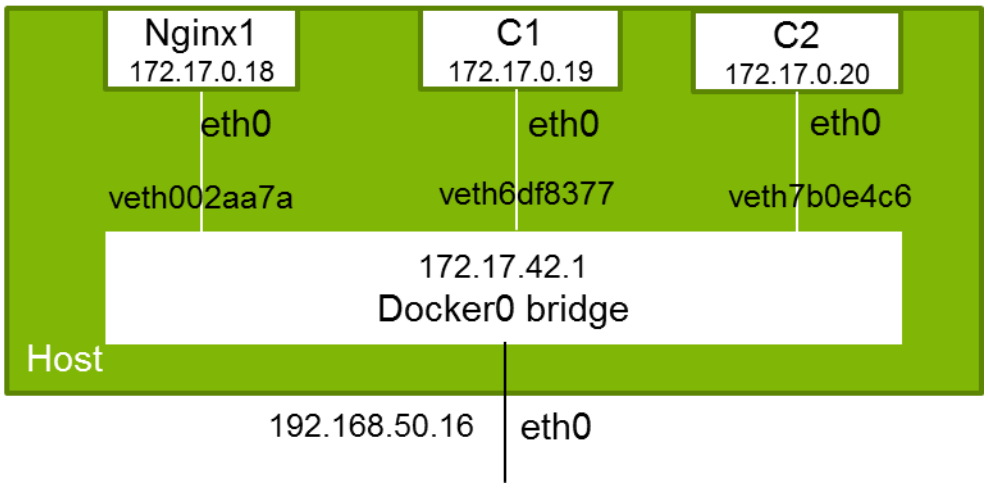
For example,

```
$ docker pull <image_name>
```

2. Docker Network

Since the Reconstruction PC requires two network interfaces, eth0 and eth2, our docker container also needs to be connected to these interfaces. Docker takes care of the networking aspects by itself so that the containers can communicate with other containers and even with the Docker Host.

By default, Docker creates a virtual bridge docker0, and all container networks are linked to docker0 initially. The docker0 bridge is the core of default networking. When the Docker service is started, a default Linux bridge is created on the host machine. The interfaces on the containers talk to the bridge, and the bridge speaks to the external world. Multiple containers on the same host machine can talk to each other through the same bridge.



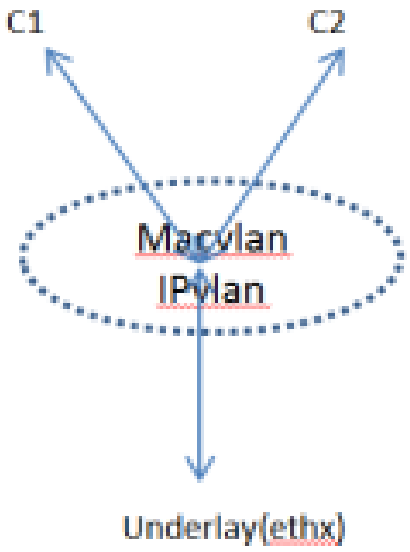
As illustrated above, each container (Nginx1, C1, and C2) has its own virtual network interface (eth0) which are in turn connected to the default docker0 bridge network, which is connected to the host network interface (eth0). This means that using default network drivers, containers are allocated static IPs and are not connected directly to the host network.

We have built a Docker application which is expected to be directly connected to the underlying local network. In this type of situation, we can use the Macvlan network driver to assign a MAC address to each container's virtual network interface, making it appear to be a physical network interface directly connected to the physical network.

To create a Macvlan network,

```
$ docker network create -d macvlan \
  --subnet=192.168.32.0/24 \
  --ip-range=192.168.32.128/25 \
  --gateway=192.168.32.254 \
  --aux-address="my-router=192.168.32.129" \
  -o parent=eth0 macnet32
```

The following picture shows two Containers using Macvlan Bridge to communicate with each other as well as to the outside world. Both Containers will directly get exposed to the underlying network using Macvlan sub-interfaces.



3. Create container

After downloading the image and creating the required networks, it is now time to create the container. For example,

```
docker create -t -i <container_name> <image_name>
```

4. **Connect container to networks**

While creating the container, we didn't provide any network driver which we created in previous steps. So the container is still connected to its virtual network interface. In this step, we connect it to the two macvlan networks corresponding to eth0 and eth2 ports of the host SLES 15 machine. Now the container has its network interfaces and is connected to the local network.

5. **Start container**

This is a straightforward step. After creating the container and connecting it to the networks, it still isn't running. We start the container using

```
docker start -a -i <container_ID>
```

6. **Configuration Steps**

After we get inside the container, we still have loads of configuration steps. These steps include changing the hostname, setting the IPs of the Mammo Host PC inside the container so a connection can be established, etc.

All these configuration steps are to be done manually for now. We'll optimize this step in the coming chapter so that these steps are performed automatically.

7. **Network Mount**

After doing the configuration, we have to run a network mount script on the Mammo Host PC so that the 2D images can be transferred to the filesystem of this container.

8. **Start the services**

This is the final step. We start the necessary services inside the container, after which the connection is established between the container and the Mammo Host PC. Now the systems are ready for the reconstruction process.

After all these steps, we have our systems ready and our container is running and performing the reconstruction successfully. But we have to optimize this as a radiologist would not know much about Docker and configuring IPs.

Optimizing Above Steps

1. **Dockerfile**

A Dockerfile is a text file that contains all the commands a user could call on the command line to assemble an image. Docker can build images automatically by executing the instructions from a Dockerfile. For example,

```
FROM ImageName
RUN apt-get update && \
    <other instructions>
ENTRYPOINT <file_name>
CMD <services>
```

This is a sample dockerfile. We are merely using some image as our base container image, installing the updates and other packages, defining our ENTRYPOINT as some file and then defining that the command for this container is the service name.

An ENTRYPOINT allows us to configure a container that will run as an executable. We write all the commands and configuration scripts inside the entrypoint filename, which will automatically execute every time we start our container. This would get rid of the manual configuration and the services steps. For this, we have to build the image using the dockerfile, by

```
docker build . <image_name>
```

After building the image, we would then push it to the Docker Hub for future deployment. This step is needed only once, as we have created a new image out of our previous one. We'll use this new image now for deployment.

2. **Removing intermediate connection**

This is a crucial step. We remove the intermediate Ethernet between our SLES 15 machine (eth0) and Mammo Host PC (eth2) and connect the latter PC to LAN. This would mean that now both the systems are connected to the External LAN and are visible in the local network. Now our container needs to be connected to one Macvlan network driver.
Both the systems would now have dynamic IP instead of static IP, which would have to be updated in the dockerfile entrypoint of the container.

This step is necessary as we want to connect multiple Mammo Host PCs to our single SLES 15 machine by running numerous containers.

3. **Image compression**

This step reduces the size of the Docker Image. We can do so by downloading only the necessary packages for the reconstruction software to run and not installing the unnecessary ones as the container is just used to run that software. We are not using the container for anything else, so as long as we can compress the size of our image, we do so.

New Image Deployment

Now we have a new image and the deployment steps have been reduced:

1. **Pulling the Image**

This step doesn't change. We still have to download our new docker image from Docker Hub or private registry, wherever it had been pushed.

2. **Create network**

This time we have to create one macvlan network corresponding to the eth2 port of the host machine so that the container is connected to the underlying local network.

3. **Run container**

In this step, we need to run the container using the image and the network we created. As soon as we run the container, all the commands which we wrote in the dockerfile entrypoint automatically execute by themselves. This saves us from the manual configuration.

For example,

```
docker run --network=my-net -itd --privileged --name=reconstruction --ip=<some_ip> <image_name>
```

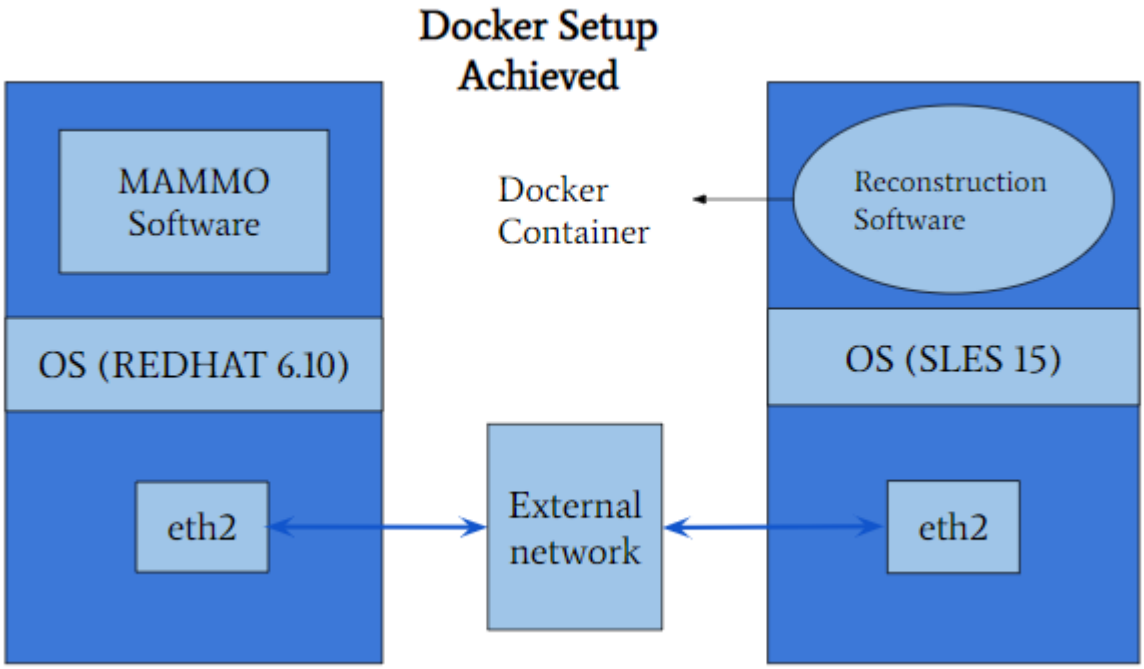
4. **Network Mount**

We still have to run a network mount script on the Mammo Host PC as we didn't disturb the configuration of this PC.

These are the steps required for the new image. The number has reduced to half of the original, which is more comfortable for the radiologist. In short, he has to do the following:

- Pull Image
- Create a network
- Run Container
- Run Script File

New Dual PC Setup



In the above diagram, we can see that the container is running on our SLES 15 machine and both the systems are now connected to the LAN.

Business Aspect

The current Dual PC setup has to be sold in pairs, as explained in the previous chapters. Now let us assume the cost of one Reconstruction PC for the customers is around 3000$.

Let's say that the company sells around 6 to 7 Senographe Pristina Mammography Systems to the customers, who are none other than hospitals themselves.

Case1: Using the current Dual PC setup

In this case, the company sells 6 to 7 pairs of Mammo Host PC and the Reconstruction PCs. The total cost of the Reconstruction PCs alone would be nearly **18000$ - 21000$**.

Case 2: Using the new Dual PC Setup

If our new Dual PC setup is implemented and sold, the company sells 6 to 7 Mammo Host machines just as before. So this cost is the same. Now instead of 6 Reconstruction PCs, we only have one single system running on SLES 15 used to run the docker containers. These containers would be connected to their corresponding Mammo Host PCs as required. This means that we will have a docker container for every Mammo Host PC. This would be a very high-end machine than the original Reconstruction PC, as it is used to run several instances of the reconstruction software. Let's say the cost of this new machine is as high as **5000$**.

Money saved per hospital = approx. **15000$.**

Now GE supplies products to thousands of hospitals all across the globe. Hence, this implementation would save a lot of money (in millions of dollars) for the customers as well as the company.

Conclusion

We calculated the time taken for reconstruction of 2D X-Ray images for the following cases:

Case1: Current Dual PC Setup

Case2: New Dual PC Setup

After running the tests for various instances of 2D images, we found that **the time taken in Case 2 is nearly 25% faster than in Case 1**.

This is merely because of the fact that our new SLES 15 machine is a very high-end machine as compared to the original Reconstruction PC. Theoretically, the duration should have been slower in Case 2 if we use a similar kind of PC in both the cases, just because in Case 2, the PCs are not connected via Ethernet and everything is happening over a network.

References

- https://opensource.com/resources/what-docker
- https://www.edureka.co/blog/docker-explained/
- https://towardsdatascience.com/how-docker-can-help-you-become-a-more-effective-data-scientist-7fc048ef91d5
- https://www.itnonline.com/content/ge-healthcare-launches-senographe-pristina-mammography-system
- https://nickjanetakis.com/blog/comparing-virtual-machines-vs-docker-containers