**INTERNSHIP REPORT**

LING Jiarong

Student at Paris Diderot University

Internship at IBM France Lab

From 2020-03-02 To 2020-08-31

# 

# ACKNOWLEDGMENT

# 

First and foremost, special regards to the ODM department of IBM France Lab, and more particularly to my manager Mr. Nicolas Mériau and my tutor Dr. Hui Sajus, for accepting me, trusting me and giving me an opportunity on this internship as well as teaching me step by step while working on different projects. As my tutors during the internship, they taught me comprehensively and consequently. I expanded my knowledge in width and in depth. Many thanks to the entire ODM Team

Secondly, my tutor at school played a instrumental role in nurturing me and constantly giving me advice. My gratitude goes to Mrs. Sighireanu who gave me a very pertinent suggestion when I chose this internship. Therefore, I would be amiss if I did not mention them.

Last but not least, I would like to applaud the Administration and Teaching staff of EIDD for their friendly efforts in creating an enjoyable learning environment. These surroundings were conducive to learning and I am most grateful to all involved in making this a fruitful/productive experience.

Finally, I would like to extend my heartfelt gratitude to my workmates, my friends and my family for giving me the support and courage.

# 

# Table of Contents

[ACKNOWLEDGMENT 2](#_Toc49030231)

[Table of Contents 3](#_Toc49030232)

[Summary 5](#_Toc49030233)

[Résumé 5](#_Toc49030234)

[Introduction 6](#_Toc49030235)

[Context 6](#_Toc49030236)

[About IBM France Lab 6](#_Toc49030237)

[About Operational Decision Manager (ODM) 7](#_Toc49030238)

[About ODM Evolution 12](#_Toc49030239)

[About Rule Execution Server (RES) 12](#_Toc49030240)

[About software development methodologies 14](#_Toc49030241)

[Internship Content 15](#_Toc49030242)

[Problems 16](#_Toc49030243)

[Solutions 17](#_Toc49030244)

[Existing solutions 17](#_Toc49030245)

[Propositions 17](#_Toc49030246)

[Project 1. Automated Configuration of remote machine 18](#_Toc49030247)

[Conclusion of project 1: 20](#_Toc49030248)

[Project 2. Docker image simulation on local machine 20](#_Toc49030249)

[Environment of Docker image 21](#_Toc49030250)

[Network In docker 23](#_Toc49030251)

[Conclusion of project 2: 24](#_Toc49030252)

[Conclusion 25](#_Toc49030253)

[Bibliography 25](#_Toc49030254)

[Annexes 27](#_Toc49030255)

[SSH 27](#_Toc49030256)

[X protocol 27](#_Toc49030257)

[XQuartz 28](#_Toc49030258)

[rsync 29](#_Toc49030259)

[hypervisor 29](#_Toc49030260)

[Docker 29](#_Toc49030261)

[Union file systems 31](#_Toc49030262)

[Sharing files with docker 32](#_Toc49030263)

# Summary

# 

The Denis Diderot Engineering School, at the University of Paris, requires of its students to undertake an industrial training, referred to as an internship, in a technology related environment, as a requisite for the partial fulfilment of an Engineering Degree in Informatics Technology. The internship is a course unit of its own that is awarded credits upon completion.

After a short interview, the ODM Department of IBM France Lab gave me an internship opportunity at the R&D department from the 2nd of March 2020, to the 31st of August 2020, a period of approximately six months. During this internship, I was attached to some major tasks: to develop tools to facilitate the debug process and to analyze the program more efficiently; to increase the coverage of unit testing and automate tasks by developing shell scripts. Therefore, this document details these tasks during the above-mentioned six-month period.

# Résumé

L'école d'ingénieur Denis Diderot (EIDD), demande à ses étudiants de suivre une formation industrielle. Le stage est une unité de cours qui se voit attribuer des crédits à la fin d’étude.

Après un court entretien, le département ODM d'IBM France Lab m'a proposé une opportunité de stage de développeur Java du 2 mars 2020 au 31 août 2020, soit une période d'environ six mois. Pendant ce stage, j'ai eu à effectuer certaines tâches principales : développer des outils pour faciliter le processus de débogage et aider à trouver plus efficacement les bugs ; augmentez la couverture des tests unitaires et automatisez les tâches en développant des scripts. Par conséquent, ce document détaille ces tâches au cours de la période de six mois.

# Introduction

The purpose of the report is to present the work I did during the internship. It will be a document for a scholarly evaluation, for the future review by IBM France Lab and to keep a clearly referenced record of what I have learned and achieved during this internship.

This document is written during the internship and will be reviewed by my instructors at IBM France company and submitted before 2020-08-24. The document contains information about the organization and the responsibilities performed throughout the period between March and August 2020. More than a plain account of tasks, the objective of this text is to reflect upon the experiences collected during the internship from the perspective of an engineering student.

The first part of the report offers an overview of the organization, followed by an outline of all the duties carried out during my internship. It then goes on to describe in some detail the most relevant projects carried out and their respective analysis. Finally, the report wraps up with a few closing remarks and conclusions from the experience.

# Context

## About IBM France Lab

On June 15, 1911, the Computing Tabulating Recording Company (CTR) was born in the United States, in Endicott, New York, from the merger of several companies that produce balances, calculators, and electronic accounting machines. In 1924, the CTR became IBM (International Business Machines). Historically, IBM's main activity has been the construction of computer hardware, a market in which IBM has become a leader.

IBM is one of the largest IT, infrastructure and IT consultancy companies in the world. For more than 100 years, IBM has continuously worked to create a smarter planet through the integration of systems and technologies. Nowadays, IBMers continue to develop and apply advanced technologies, such as Watson artificial intelligence, the Cloud, IoT and Blockchain, in many domains to transform all industries, all over the world, reinventing the experience of passengers in transport, improving data security or even revolutionizing the health care field.

The French division of IBM was created in 1914 under the name of “Société Internationale de Machines Commerciales (SIMC)” before taking the name of IBM France in 1948. Since November 2009, the new IBM France head office has welcomed 4,000 employees in an HQE (high environmental quality) building located at Avenue de l’Europe in Bois-Colombes, in the Bécon-les-Bruyères district, a few kilometers from Paris.

IBM is positioned as a leader in information technology and the preferred partner for companies that believe in innovation as an engine of growth. By aligning information technology with business processes in this way, companies improve and streamline operations, but also benefit from a greater return on their technology investments. IBM provides businesses with the most comprehensive set of resources : skills, systems, software, services, finance, technology to help them and enable them to become companies of innovation.

## About Operational Decision Manager (ODM)

Operational decision manager (ODM) is a set of technologies and methods for clients to automate their day-to-day operational decisions without compromising their accuracy and effectiveness. It combines decision making and change detection tools to provide a business rule management system that is easy to evolve, trace, audit, and test. ODM allows clients extracting decisions from their application code, managing decisions and Putting decision management decisions in the hands of business users.

**Business policies** are statements that are utilized to make decisions. These decisions can determine pricing for insurance or loan underwriting, eligibility approval for social or health services, or product recommendations for online purchases. Business policies are typically found inside application code, in the form of if-then statements. However, they can also be stored elsewhere for documentation purposes, such as in procedural manuals or other documents.

A **business policy** can be expressed as several business rules. Here is an example of the kind of business policy that might be familiar: Customers who spend a lot of money in a single transaction need an upgrade. The process of capturing rules happens in two steps. The first step consists in formalizing the vocabulary that is required to express the policy as a conceptual object model. The second step consists in representing the logic of the business policy as if-then statements.

After the **vocabulary** is created, the business policy can be implemented with the following business rule:

if

the customer's category is Gold

and the value of the customer’s shopping cart is more than $1500

then

change the customer's category to Platinum

When a business policy also has an IT policy or security policy that is embedded in it, you can combine business rule management with capabilities to handle the business policy aspects. For example, the following business policies can be handled as rules: customers who spend large amount of money should be routed to a preferential service or may require additional security on their transactions.

In the form of business rules, the business logic can be packaged and called from the application code as a decision service. Therefore, changes to the business policy do not require changes to the application or process code.

A picture containing clock

Description automatically generated

Figure Scenario of ODM Application

A screenshot of a cell phone

Description automatically generated

Figure Example of Application powered by ODM

A screenshot of a social media post

Description automatically generated

Figure An example of Rules

Operational Decision Manager includes two main components: **Decision Server** for managing decisions and **Decision Center** for putting decision management in the hands of those who drive the business.

**Decision Server** provides the runtime and development components to automate the response of highly variable decisions based on the specific context of a process, transaction, or interaction. You can process the information against hundreds or even thousands of business rules in order to determine how to respond within front-end and back-end systems.

**Decision Center**, business users can manage decisions directly based on organizational knowledge and best practices, with limited dependence on the IT department. The degree of dependence can range from a limited review by business users of the business logic implemented by developers, to complete control over the specification, creation, testing, and deployment of the business logic by business users.

Business and IT functions can work in collaboration, aligning the entire organization in the implementation of automated decisions and accelerating the maintenance lifecycle as they evolve based on new external and internal requirements.

Two categories of users are involved in developing and maintaining a decision management solution.

**IT users:**

Architects, developers, and administrators develop and maintain the decision services.

**Business users :**

Business analysts, policy managers, and rule authors develop and maintain the decision logic.

A screenshot of a social media post

Description automatically generated

Figure Decision Center & Decision Server

## About ODM Evolution

The ODM Evolution Team is responsible for product maintenance, testing, building and delivery of ODM. It consists of a RES sub-team (responsible for Rule Execution Server module of ODM), a DC sub-team (responsible for Decision Center module), a QA & Software Factory sub-team (responsible for the product building and Quality Assurance) ...

## About Rule Execution Server (RES)

Decision Server provides development and runtime components for a rule-based solution that automates the response of highly variable decisions required by client applications.

* **Rule Designer** is an Eclipse-based development environment in which you design, author, test, and deploy decision services.
* **Rule Execution Server** provides the runtime environment for running and monitoring decision services.

At the core of a rule-based solution, you have a client application requesting a decision from a **decision service**. There can be many decision points required of the decision service by the client application. At each decision point, business rules packaged as **rulesets** are used to express policies for how decisions are made. A decision service is deployed to **Rule Execution Server** as a **RuleApp**. Each **RuleApp** contains one or more **rulesets**, each corresponding to a decision. It has an execution and management model, Java™ EE and Java SE execution components, the **Execution Unit** (XU), a persistence layer.

A **RuleApp** is a deployment and management unit for Rule Execution Server. A RuleApp contains one or more rulesets. You deploy your RuleApps in to Rule Execution Server order to make the ruleset available to a client application. RuleApps can be deployed from a decision service deployment configuration. **Rule Designer** does not explicitly show **RuleApps**. You use the decision service deployment configuration to hold information to make a RuleApp.

Architects, administrators, developers, and testers work together to create, deploy, tune, and administer RuleApps and client applications that run on **Rule Execution Server** instances on a production enterprise cell.

A screenshot of a cell phone

Description automatically generated

Figure 5 Role of RES

**Rule Execution Server** is an environment for executing rules. It provides management, performance, security, and logging capabilities.With Rule Execution Server, you can change the business logic dynamically in both Java™ SE and Java EE environments.

## About software development methodologies

The **Agile** approach as the development methodology is integrated: there is a department meeting every week to discuss the process of work of each other, new tasks with different priorities, new events and some management decisions ; a briefer meeting for each sub-teams every week to discuss the process, the challenge and the target to achieve weekly. It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages flexible responses to changes.

**Continuous integration** (CI) is used that helps us merge our code changes back to a shared branch, called “trunk” daily. When a developer’s changes to an application are merged, those changes are validated by automatically building the application and running the test automation, typically unit and integration tests, to ensure the changes have not damaged the program. This means testing everything from classes and functions to the different modules that comprise the entire product. If the test automation discovers a conflict between new and existing code, CI makes it easier to fix those bugs frequently and quickly. The runner of test automation is called **Versatile Testing Tool (VTT)**, a Testing and Reporting system that can help Dev/QA Teams to distribute and execute test campaigns, and then report the test results in a web dashboard application. Anyone can command the powerful remote machines to run the test automation daily or even more frequently by using **VTT**.

When there is an error, we can easily detect which merging causes the error. In the next example, there are some failures for some unit testing classes so we know that we need to relocate and correct the error.

A screenshot of a cell phone

Description automatically generated

Figure 6 Details of Trunk RES Test

We can see the failing tests, but we cannot easily figure out easily the reasons why they failed. To help analyzing the program, I need to develop some tools to run test automation on a local machine.

## Internship Content

When I arrived, in order to become familiar with this product, I first began by following the tutorial of ODM, implemented some automated tests for the first month, and then immediately entered the subject of my internship. It took me several months to develop a program to resolve the problem, and then I spent the rest of my time improving this program and writing some technical documents as well as helping with product maintenance.

# Problems

For the ODM evolution department, we have automated tests running on the remote machines (Linux System) every day. Our test automation process is generated by a script: Start the server on the remote machine, then run the test program, count the pass rate of these tests, and display our test results on the VTT platform. Consequently, we only need to check whether there are problems after the testing has finished running, and then we need to debug if there is a problem.

Given that the operating system of our local machines for development are MacOS or Windows, it is possible that the difference of operating system makes the result of the same testing different. In fact, this actually happens: Some tests pass on the local machine but fail on the remote machine. In this case, it is hard to locate the problem because we cannot turn the debug mode on our developing machine.

In addition, our product supports running on many different web servers (tomcat, WAS, liberty …), compatibles with different databases (mysql, db2, oracle...). The testing should run with the use of different browsers (Firefox, Chrome).We also need to support different versions of Jdk( openJdk, IBM Java, Oracle) and different OS (Redhat 7, Redhat 6). In the development environment, it is difficult for us to execute testing with various combinations, but on the remote machine, we can easily run in all required combinations.

The main task for me is to develop some tools in order to simplify the debugging procedure, which requires it to be stable and easy to use. The tool should display the exact error message and developers can analyze easily why tests fail.

# Solutions

## Existing solutions

Suppose that VTT tells us a test fails, to fix the problem we need to first locate the bug. Here are some possible solutions.

1. The first naïve solution: By ignoring the difference of OS, we follow the same process on local machine. More specially, we launch the server on debug mode and execute the automated testing. If we are able to reproduce the same error, we can debug directly on our local machine. But it happens that the test passes with no problem on local machine. In this case, we are not able to use the debug mode to locate the bug easily.
2. We can improve the first solution by moving the server to the remote machine: Start the server on the remote machine and allow local machines to connect onto the server, then launch automated testing on a local machine. If the bug is in the testing part, we can do debug directly on the local machine. But if the bug is on the server side, we are not able to use the debug mode.
3. We can do even better by using the debugger: Start the server on the remote machine with debug mode on and allow local machines to connect onto the server, then connect to the debugger and launch automated testing on local machine. It resolves most of cases, but it happens that the frameworks, software act differently on different OS. In this case nothing can be done on a local machine.

## Propositions

To exactly simulate the same environment Linux, we need to operate directly on the Linux system:

1. We can consider the remote machine as a workstation and connect it to the machine. There are many types of remote desktop software like VNC Viewer (based on VNC protocol) and SPICE-Client (based on Spice protocol). We can also use ssh if we don’t need the remote desktop. We chose ssh + X protocol if the bandwidth is low and VNC Viewer if the bandwidth is high enough.
2. We can use a virtual machine based on the Hypervisor technique. Several VM Software can be used but there is not a stable version for MacOS. So, we chose to use Docker.

After discussing with my tutor, I started to implement these two solutions. For the first solution, the idea is to use scripts to avoid installing the necessary tools and software every time we get a remote machine. Because the remote machine could be reset anytime and could be shared by others. For the second solution, I need to build a docker image with the necessary tools and software on it.

## Project 1. Automated Configuration of remote machine

The goal is to move the source code to a remote machine, configure the development environment and construct the project structure. Developers can directly develop, test and tune in the released environment by using this project.

For this project, given that we already have the Linux system. All we need is to install our developing tools and download the source code.

To connect to the server, I usesecure shell([*ssh*](#_SSH.)) and [*XQuartz*](#_XQuartz)to grant access to the remote machine and Secure Copy(*scp*) in order to transfer the source code.

To make this project simpleto use, I wrote twobash scripts, one to install eclipseon the remote machine and the other to build the project and copy the source codeto theremote machine. The script for installing eclipse is to use ***wget*** to select the most suitable image from the eclipse official website to download, and then decompress it. Another script uses ***sshpass*** to connect to the target machine, build the proper directory, and then copy the source codefrom local machine to the target.

A close up of a map

Description automatically generated

Figure Project Diagram

Developers can launch two scripts and the environment on the remote machine is prepared.

Although this project is simple, there are some things topay attention to. Firstly, since we only need src (\*.java,\*.jar…) to be copied to the remote machine, log files and dependencies should be filtered out. To exclude some files in a directory, [***rsync***](#_rsync)is the perfect choice. Then, in order to allow users to use my script directly without installing the prerequisite tools, the script will automatically install all the dependencies if they have not already been installed yet. I also decided to define a configuration file (format ***ini***) in which we can easily define the URL, user, password and destination of the developing tool of the remote machine.Through the above design, users only need to run the script to build the environmenteasily.

After a number of tests and improvements, this project worked well and will be used when necessary. We need to use a remote-control method to work on the remote machine. *ssh* (with X Protocol), **VNC Viewer** and **TeamViewer** could be used but only *ssh* works if the bandwidth is low.

### Conclusion of project 1:

This is a solution: we move the debugging process from the local machine to the remote machine. The development environment is exactly the same as the VTT release environment. This project helps to configure the developing environment and construct the project structure on the remote machine.

But the disadvantage is also obvious: when we don’t have a high bandwidth, we can only use **ssh** to connect to the remote machine. More importantly, the remote machine often fails and there is a limited number of available machines. It happens that we won’t be able to use a remote machine when the platform is busy. We need to continue our work even when we do not have a remote machine available!

## Project 2. Docker image simulation on local machine

We still want to simulate the test automation even when the platform VTT is not available. To achieve this goal, we decide to build a Docker image on a local machine.

If we want to work without the remote machine, we need to use a technique called [*hypervisor*.](#_hypervisor) Virtual machine and [Docker](#_Docker) are two different options in this case. The operating-system-level virtualization allows us to run a Linux VM on the macOS machine. Given that there is not a stable version of Virtual Machine Software for MacOS. We decided to build our own Docker image.

Docker can build images automatically by reading the instructions from a **Dockerfile**. A Dockerfile is a text that contains all the commands a user could call on the command line to assemble an image. By using “*docker build”* users can create an automated build image that executes several command-line instructions consecutively. To accelerate the speed of network transmission and build an extensive docker image, we decided to use Dockerfile in which we specified all the configuration and installation.

A picture containing text, map

Description automatically generated

Figure The composition of Docker

### Environment of Docker image

According to the release environment, several composition of web servers and jdk should be supported. In order to simulate more environment in the docker container, the following tools should be installed and configured:

To make the project easier to use, I encapsulate a script to build the image and also run the container with the correct [binding](#_Sharing_files_with) and network.

We need to modify the src inside the container and also on the host, so we chose to use bind mount over volume. In this case, to use the files on the host, we mount the src directory on a host machine directly with the docker container. When we modify the src within the docker container, the actual src will be modified. So, do not need to worry about the transmission of files like in the first solution.

To construct the correct environment on the local machine, we need to:

1. Pull the Redhat image. The remote machines are under Redhat OS, we need a Redhat image in docker. There is already a Redhat image in docker hub, so we can use it directly.
2. Installation of necessary developing tools such as wget unzip, ibm-jdk, eclipse... in docker.
3. Configure the system environment in docker.
4. Build the docker image, share files with host (in order to change the src in docker).
5. Execute a docker container using the docker image and configure the X protocol (for displaying GUI applications).

Steps 1,2,3 are encapsulated in the dockerfile, and Steps 4,5 are encapsulated in a script. Since docker is based on a [union file system](#_Union_file_systems), the docker image will be built by layer.

A screenshot of a cell phone

Description automatically generated

Figure The docker image

Through this project, we directly established a virtual environment on the local machine that is almost the same as the release environment as well as being convenient to use. Developers only need to run a script to pre-install docker-related software, and then run another script to build the relevant image, wait a few minutes and then enter the Red Hat environment where directly simulate related tests, debugged and tuned. Because of the binding technology, the src in the docker is directly shared with the local machine: local modifications are visible to the docker container, and the code modified inside the docker container is also visible by the local machine, which greatly improves the development efficiency. However, in order to support running multiple test processes at the same time, I refactored the project and changed to a server-client architecture. The advantage of the separation of the two container instances is that multiple tests can be run on a server, and the container becomes more lightweight. However we need to consider about how different containers communicate with each other.

### Network In docker

The **bridge** network is useful in this case. The bridge driver creates a private network internal to the host so containers on this network can communicate with each other. External access is granted by exposing ports to containers. Docker secures the network by managing rules that block connectivity between different Docker networks.

Behind the scenes, the Docker Engine creates the necessary Linux bridges, internal interfaces, iptables rules, and host routes to make this connectivity possible. In the example highlighted below, a Docker bridge network is created, and two containers are attached to it. With no extra configuration the Docker Engine does the necessary wiring, provides service discovery for the containers, and configures security rules to prevent communication to other networks.

A screenshot of a cell phone

Description automatically generated

Figure Structure Of bridge network

### Conclusion of project 2:

Through this project, we can directly build a virtual environment on the local machine that is almost the same as the release environment. It is quite simple to use and we do not need a remote machine. However, the driver for browser Google-Chrome “*chromedriver”* does not work with the X protocol. So it is a pity that we cannot test with Google-Chrome using this project.

# Conclusion

This internship was very fruitful to me because I had to cover many different fields. In terms of programming skills, I learnt programming in Java, developing tests with Selenium and scripting with shell, which is very useful for further software development. I had a chance to experience the deployment of a web project with several web servers on the Linux server, the developing of debug tools and techniques of virtualization. On the other hand, I also experienced the whole software development process, which taught me to follow some structures (Separation of Client-server, Separation of configuration and execution )in order to make the code readable and extensible. In terms of management. I experienced the Agile approaches development which advocates adaptive planning, evolutionary development, early delivery, continual improvement and flexible responses to change. Working at IBM is an amazing opportunity for learning, whether it is soft skills or hard skills. What I appreciated was the atmosphere of trust and respect between the management and the employees. In a word, it was an extremely valuable experience because I gained professional expertise for my CS career

# Bibliography

* IBM official site, https://www.ibm.com/
* IBM Knowledge Center of IBM product documentation, <https://www.ibm.com/support/knowledgecenter/>
* IBM Wikipedia, <https://fr.wikipedia.org/wiki/IBM>
* IBM France, <https://fr.wikipedia.org/wiki/>
* Scrum Wikipedia, <https://en.wikipedia.org/wiki/Scrum_(software_development)>
* IBM Rational Team Concert, <https://www.ibm.com/fr-fr/marketplace/workflow-management>
* Maven official site, <https://maven.apache.org/>
* Selenium <https://www.javatpoint.com/selenium-tutorial>
* JUnit Wikipedia, <https://en.wikipedia.org/wiki/JUnit>
* JavaResourceBundle tutorial, <http://tutorials.jenkov.com/java-internationalization/resourcebundle.html>
* Webdriver, <https://www.techbeamers.com/selenium-components-web-test-automation/#8211-selenium-webdriver>
* Docker EcoSystem, <https://washraf.gitbooks.io/the-docker-ecosystem/content/Chapter%201/Section%203/union_file_system.html>
* Docker documentation, <https://docs.docker.com/network/>
* DerBy, <https://db.apache.org/derby/>
* Apache Tomcat, http://tomcat.apache.org/
* Linux man Page, <https://linux.die.net/man/1/sshpass>
* Linux CGroup, <https://www.linuxembedded.fr/2011/03/bien-utiliser-les-cgroups/>
* Linux NameSpace, <https://man7.org/linux/man-pages/man7/namespaces.7.html>

# Annexes

## SSH

SSH is a secure protocol and the most common way of safely administering remote servers. Using a number of encryption technologies, SSH provides a mechanism for establishing a cryptographically secured connection between two parties, authenticating each side to the other, and passing commands and output back and forth.

**A screen shot of a computer

Description automatically generated**

Figure The process of SSH

## X protocol

To see the **GUI (**graphical user interface), we must configure the **X protocol.**The X Window System core protocol is the base protocol of the X Window System, which is a networked windowing system for bitmap displays used to build graphical user interfaces on Unix, Unix-like, and other operating systems. The X Window System is based on a client–server model: a single server controls the input/output hardware, such as the screen, the keyboard, and the mouse; all application programs act as clients, interacting with the user and with the other clients via the server. This interaction is regulated by the X Window System core protocol. Other protocols related to the X Window System exist, both built at the top of the X Window System core protocol or as separate protocols.

A close up of a sign

Description automatically generated

Figure The structure of X Protocol

## XQuartz

**XQuartz** is an open-source version of the X server, a component of the X Window System that runs on macOS. The name "XQuartz" derives from Quartz, part of the macOS Core Graphics framework, to which XQuartz connects these applications. XQuartz allows cross-platform applications using X11 for the GUI to run on macOS, many of which are not specifically designed for macOS. This includes numerous scientific and academic software projects. With XQuartz, the configuration of X protocol would be easy to use.

## rsync

***rsync*** is autility for efficiently transferring and synchronizing files between a computer and an external hard drive and across networked computers by comparing the modification times and sizes of files

## hypervisor

A ***hypervisor*** is computer software, firmware or hardware that creates and runs virtual machines. A computer on which a hypervisor runs one or more virtual machines is called a host machine, and each virtual machine is called a guest machine. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources: for example, Linux, Windows, and macOS instances can all run on a single physical x86 machine. This contrasts with operating-system-level virtualization, where all instances (usually called containers) must share a single kernel, though the guest operating systems can differ in user space, such as different Linux distributions with the same kernel.”

### 

## Docker

The operating-system-level virtualization allows us to run a Linux VM on the macOS machine. Given that there is not a stable version of Virtual Machine Software for MacOS. We decided to build our own Docker image. Docker is a set of platforms as a service (**PaaS**) product that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines.

As shown in figure 9 Docker is container-based technology and containers are just user space of the operating system. At the low level, a container is just a set of processes that are isolated from the rest of the system, running from a distinct image that provides all files necessary to support the processes. It is built for running applications. In Docker, the containers running share the host OS kernel. A Virtual Machine, on the other hand, is not based on container technology. Virtual Machines are made up of user space plus kernel space of an operating system. Under VMs, server hardware is virtualized. Each VM has an Operating system (OS) and apps. VM shares hardware resources with the host.

A picture containing clock

Description automatically generated

Figure The comparison between Docker and Virtual machine

Docker is written in Go and takes advantage of several features of the Linux kernel to deliver its functionality. Docker uses a technology called **namespaces** to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container. These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace. Docker Engine uses namespaces such as the following on Linux:

* The *pid* namespace: Process isolation (PID: Process ID).
* The *net* namespace: Managing network interfaces (NET: Networking).
* The *ipc* namespace: Managing access to IPC resources (IPC: InterProcess Communication).
* The *mnt* namespace: Managing filesystem mount points (MNT: Mount).
* The *uts* namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Docker Engine on Linux also relies on another technology called **control groups** (cgroups). A ***cgroup*** limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources with containers and optionally enforces limits and constraints. For example, you can limit the memory available to a specific container.

## Union file systems

**Union file systems,** or **UnionFS**, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS and Device Mapper.

Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a **container format**. For our project, we can customize our own docker image and launch the container using the docker image. Then we can work on the docker container to continue the debugging process and analyze our program.

Docker names each of these file system **images**. Images can be layered on top of one another. The image below is called the parent image and you can traverse each layer until you reach the bottom of the image stack where the final image is named the base image. Finally, when a container is launched from an image, Docker mounts a read-write filesystem on top of any layers below. This is where whatever processes we want our Docker **container** to run will execute. When Docker first starts a container, the initial read-write layer is empty. As changes happen, they are applied to this layer; for example, if you want to change a file, the file will be copied from the read-only layer below into the read-write layer. The read-only version of the file will still exist but is now hidden underneath the copy.

## Sharing files with docker

Volumes and Bind mounts are the mechanism for persisting data generated by and used by Docker containers. Bind mounts are dependent on the directory structure of the host machine, while volumes are totally managed by Docker. A bind mount is a file or folder stored somewhere on the host file system, mounted into a running container. The main difference a bind mount has from a volume is that since it can exist on the host filesystem, processes outside of Docker can also modify it. We need to modify the src inside the container and also on the host, so we chose to use bind mount over volume.

A screenshot of a cell phone

Description automatically generated

Figure sharing file structure