CS3204 | Cloud Infrastructure and Services

Lab1: Virtual Machines and Containers

Content

- Goals & Objectives.
- Benefits of VMs and Containers.

Lab Work Content

- VM and containers (and a bit on KVM & Hyper-V).
- Testing Method
- Applications used
- Who is fastest: The Host, VM or Mr Container?
- Compare & Analyse Results - show which is the best result and why.
- Experience gained from using, testing and debugging new software.

Goals & Objectives

My goal for this 3 week lab is to put into practice the theoretical knowledge I have acquired from lectures and to further understand the advantages of Virtual Machines and Container technologies that are used in the cloud and the services they can provide to us.

I will accomplish this by documenting the runtime of a Fibonacci application that I deployed on my Host Computer, inside a Ubuntu Virtual Machine and inside a container powered by Docker.

By the end of this Lab work, I hope to have an enhanced understanding of container & VM software and when it is best to use each one.

Benefits of VMs & Containers

In a nutshell:

VMs solve infrastructure problems by letting organisations get more out of their servers and facilitate limited workload portability.

Containers solve application problems by improving DevOps, enabling microservices, increasing portability, and further improving resource utilisation.

- Source VMware

In my project, I am to get acquainted with the basics of VMs & Containers so that I could further understand their benefits and where I can best save costs in their deployment in the real world!

In the modern world, cloud resources (including software) are provided to clients through a pay-as-you-go approach. Virtual machine and container technologies enhance the utilisation of cloud hardware resources, promote energy efficiency, and maintain load equilibrium, all the while establishing a secure framework for executing monolithic or distributed applications.

Nevertheless, these technologies introduce an additional software layer that can affect system performance. The sole exception to this scenario arises when bare metal servers are employed, and the hypervisor operates as a lean operating system.

In my case, I used a Lenovo Thinkpad with Windows 11. Using MyComputerDetails: I collected some basic information about my laptop specs:

Operating System	
Size	64-bit
Operating System	Windows 11
Version	10.0.22621
Locale	0809

BIOS	
BIOS	R17ET31W (1.14)
Manufacturer	LENOVO

Sound	
Audio 1	
Sound Device	Microsoft Streaming Service Proxy
Audio 2	
Sound Device	Intel(R) Display Audio
Audio 3	
Sound Device	Realtek(R) Audio
Driver	6.0.9158.1

Drives	
Free Memory	13 GB

Software	
DirectX	11.0
.NET	4.8
Internet Explorer	11.1.22621.0
Chrome	0.0

Processor	
Processor	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz
Number of Cores	8
Speed	4.2 GHz
Stepping	C
Family	06
Model	8E
CPU ID	BFEBFBFF000806EC

Memory	
RAM	8.0 GB

Video Card	
Video Card	Intel(R) UHD Graphics
Chipset	Intel(R) UHD Graphics
Manufacturer	Intel
Hardware T&L	Yes
Total Memory	4.0 GB
Dedicated Memory	128 MB
Driver Version	30.0.101.1122
Vertex Shader Version	5.1
Pixel Shader Version	5.1
Plug and Play ID	VEN_8086&DEV_9B41&SUBSYS_507C17AA&REV_02
Device	9B41
Vendor ID	8086

Knowing the limits of my hardware, I could not afford to run a resource heavy application to test VMs & containers to their very extremes!

As advised by my lecturer (if you are reading this right now, then hello hello!), I ran a simple program that was easy to redeploy on a VM & Container software without too much struggle or harsh waiting times!

Now.. I keep talking about VM & Containers, but what actually are they?

Introduction to VMs & Containers (and a bit on KVM & Hyper -V)

Virtual Machines (VMs) and containers are two fundamental technologies in the world of virtualization and application deployment. They both serve critical roles in modern computing, enabling efficient resource utilisation and application isolation. Let's take a look into each one in greater detail!

Virtual Machines are a form of hardware virtualization that replicates an entire physical computer, including the operating system, on a single physical server. They operate through a hypervisor, which is a software or hardware layer that manages multiple VMs on the same physical hardware.

Containers, on the other hand, are a lightweight form of virtualization that packages an application and its dependencies into a single unit. They share the same OS kernel with the host system but remain isolated from each other.

Here are some key features provided by VMs and containers:

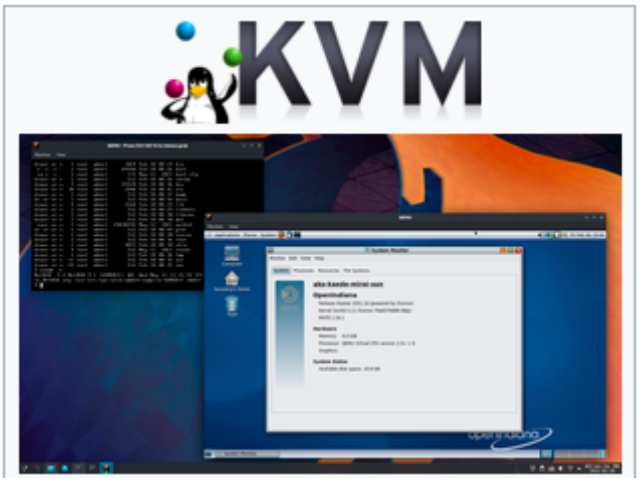
Isolation	Isolation
Resource Overhead	Resource Efficiency
Flexibility	Consistency
Snapshots	Orchestration
Migration	Portability

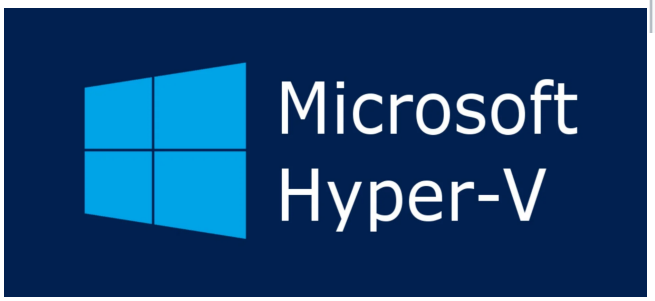
In summary, both VMs and containers have their own use cases. VMs are suitable when strong isolation is needed (for example hacker protection or testing malicious software), or when running applications with different operating systems. Containers excel in resource efficiency, scalability, and consistency, making them the preferred choice for modern, cloud-native applications and microservices. In many cases, organisations use both VMs and containers to create a versatile and efficient infrastructure that meets their various needs.

A touch on KVM & Hyper-V

KVM (Kernel-based Virtual Machine) and Hyper-V are both virtualization technologies that enable the creation and management of virtual machines (VMs) on a physical host system. However, they are developed and used in different environments, with KVM being primarily associated with Linux-based systems and Hyper-V being a Microsoft virtualization technology. Let's looked into each of these in a bit more detail (again):

KVM is an open-source virtualization technology that is tightly integrated with the Linux kernel. It allows you to run multiple virtual machines on a single physical host, each with its own operating system. KVM provides hardware-assisted virtualization, which means it takes advantage of virtualization extensions in modern CPUs to improve performance and security.





As for Hyper-V, it is a virtualization technology developed by Microsoft. It is available on Windows Server editions and Windows 10/11 Pro and Enterprise editions. Hyper-V enables the creation and management of virtual machines on Windows-based systems.

Biggest advantages of these 2 software are their speeds as they have direct access to the computer's physical hardware. However, they both have slightly different uses in industry.

KVM is used in cloud computing and enterprise environments (so Linux machines mainly), whereas Hyper-V is more for Integration with Windows Ecosystems & Containers (so Windows machines mainly).

In summary, KVM is an open-source virtualization solution tightly integrated with the Linux kernel, while Hyper-V is a Microsoft-developed virtualization technology primarily used in Windows environments. Both are powerful virtualization platforms, and the choice between them often depends on the specific needs and existing infrastructure of the organisation or individual users.

But to answer simply: does your company use Linux or Windows (mainly)?

Applications used

For this lab we were allowed to make use of any Host operating system, any Virtualization software & any Container software. I did however follow provided guides & sources to assist me, most notably, the installation of VM software.

Host Operating system:	Windows 11
Integrated Development Environment:	Visual Studio Code
Virtualization Software:	Oracle VM VirtualBox
VM Operating System:	Linux - Ubuntu
Container Software:	Docker
Integrated Development Environment:	Visual Studio Code

Anytime I was confused on how exactly to implement something, I researched on my own, asked peers or people present in the lab.

Fibonacci: A Recipe for Disaster?

I built a simple program designed to calculate the n-th Fibonacci Number using Recursion programming. Due to it being recursive the time complexity became O(n^2). I wrote a python file containing this code (I have attached it at the bottom of the report for further viewing as well, as well as the base file & a link to my GitHub repository).

Due to the ever increasing runtime of my program, I was forced to limit it to 40 iterations if I wanted my results back reasonably fast (and without burning my laptop)!

When testing the execution time of the 3 environments, to keep all conditions equal, I made sure not to run any unnecessary background processes that could affect the calculation time.

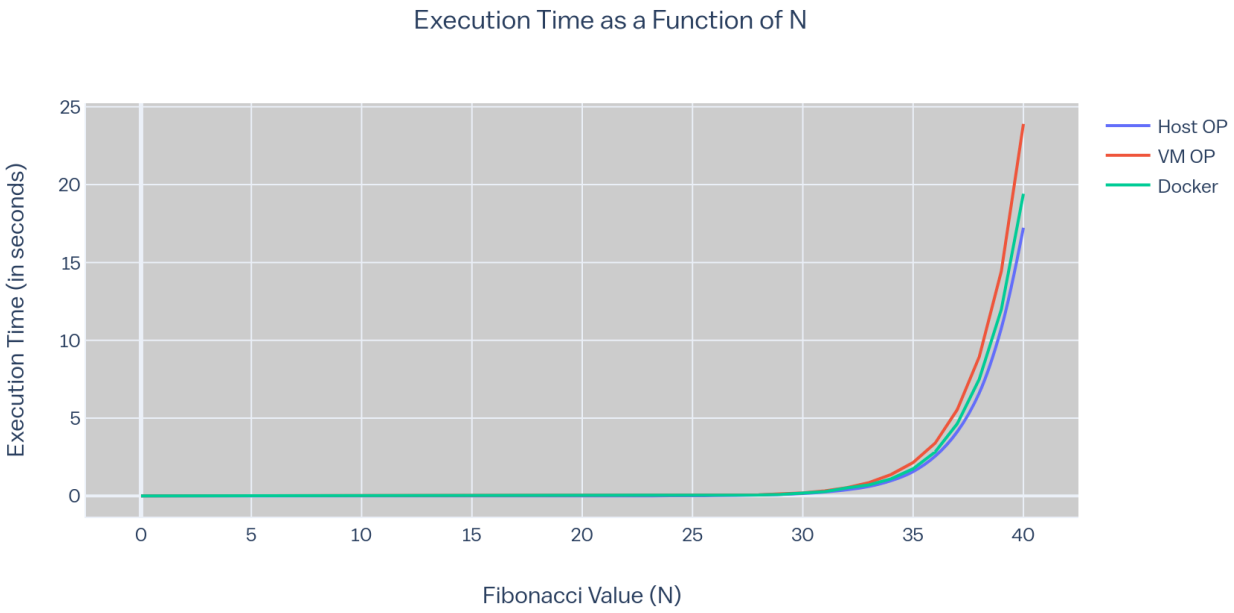
Collection of data was done via 3 CSV files - one per environment. (can be found attached to result & inside GitHub repository)

Afterwards, I used plotting software to draw a comparison graph between the 3 environments.

I have taken some screenshots to showcase my collection of data as well, if those are needed!

Analysing the Results of my Work

Firstly, we have a comparison graph of the 3 different environment execution times:



Secondly, to make the results a bit more clear, I made a table of the values - for easier comparison!

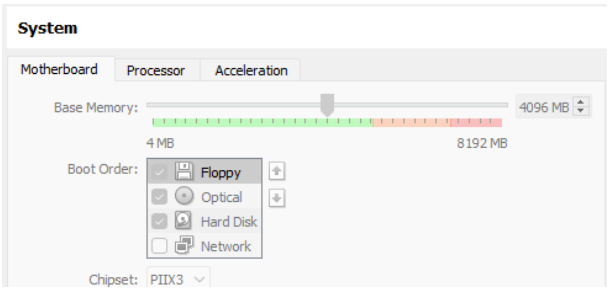
Fibonacci Value	Host Op	VM OP	Docker
35	1.5	2.1	1.7
36	2.6	3.4	2.8
37	4.1	5.5	4.6
38	6.6	8.9	7.5
39	10.8	14.4	12.0
40	17.2	23.9	19.4

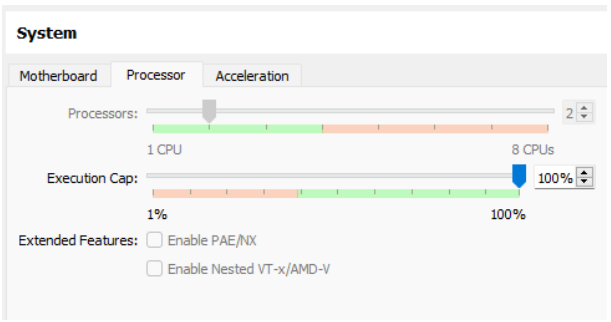
(rounded to 1 decimal place)

From looking at my results, I can see that the program runs fastest on the Host Operating System, followed by Docker Containers & then the Virtual Machine.

This result is completely expected as the code I am using is not parallelized hence Docker Containers can't take full advantage of the task provided to them. I also did not code my program to be parallelised.

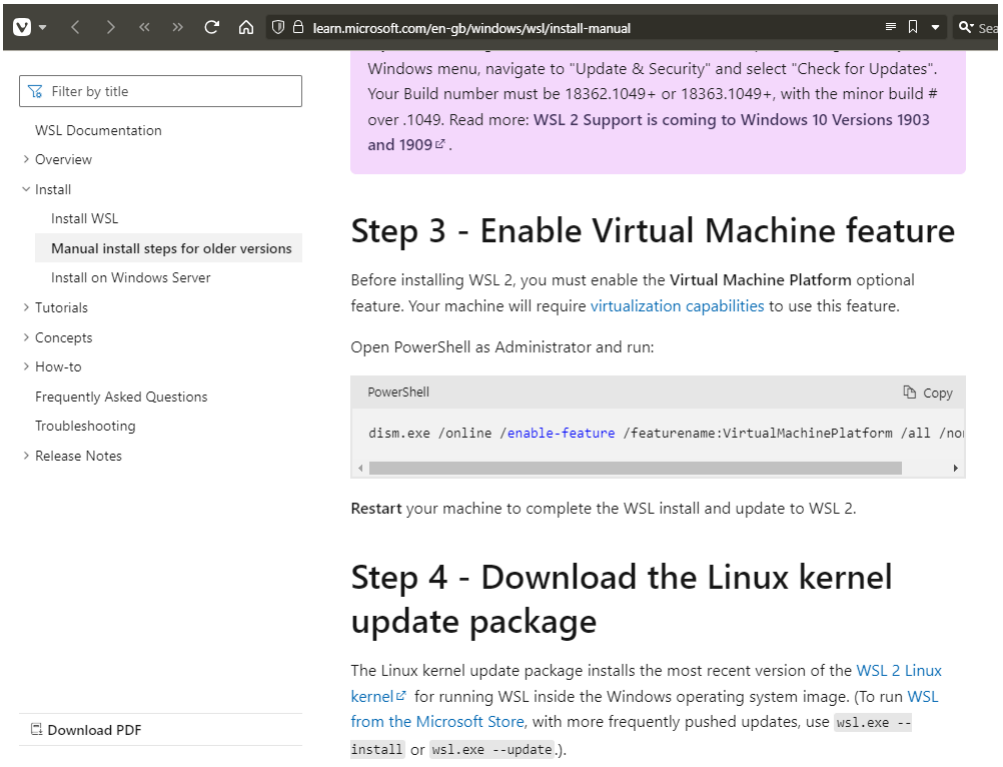
Another point to note is that the Virtual Machine was only allocated 4GB of RAM & only 1 CPU, whereas my native laptop had access to everything when running the program code.





Experience gained from using, testing and debugging new software.

Dockers was definitely one of the biggest culprits for time wasted during this lab work. The initial set-up of Docker Hub was plagued with additional instals, in-particular I had to reinstall python & even download a new Disk Driver!



It was a mess! But once it got working! It was very satisfying to observe all the downloads occurring inside of the terminal!

```
Build an image from a Dockerfile
PS C:\Users\Admin\Documents\UNI-Docker> docker build -t python-imdb .
[+] Building 0.7s (7/7) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 31B                                                                                 0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3.11                                                      0.5s
 => [internal] load build context                                                                                   0.0s
 => => transferring context: 1.05kB                                                                                 0.0s
 => CACHED [1/2] FROM docker.io/library/python:3.11@sha256:832cdc43284d265c5a5626893fbd0192c2878ef8381c9cc255fc26b93f6a28fd  0.0s
 => [2/2] ADD appcode.py .                                                                                          0.0s
 => exporting to image                                                                                              0.0s
 => => exporting layers                                                                                             0.0s
 => => writing image sha256:a389682ef3bd5287b5756f3b1935fa87f1d274ac68406fff06a6078d320c8704                        0.0s
 => => naming to docker.io/library/python-imdb                                                                      0.0s
```

```
PS C:\Users\Admin\Documents\UNI-Docker> docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS
f75a382fe1f9   python-imdb   "python ./appcode.py"   32 seconds ago   Up 31 seconds
PS C:\Users\Admin\Documents\UNI-Docker> docker stop ef1a41454d90^C
PS C:\Users\Admin\Documents\UNI-Docker> docker stop f75a382fe1f9
f75a382fe1f9
PS C:\Users\Admin\Documents\UNI-Docker> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\Admin\Documents\UNI-Docker> docker build -t python-imdb
"docker build" requires exactly 1 argument.
See 'docker build --help'.
```

The most difficult part of this full assignment was definitely installation of Docker!

But I learned a lot and enjoyed the overall experience!

Applications Code

```
# Student Number: 121464124
# Author: Eugene .Z

# dependacies/imports
import time

# application code
def Fibonacci(n):
        """
        Fibonacci program done using Recursion
        """
```

```
        # Check if input is 0 => print incorrect input
        if n < 0:
                print("Incorrect input, must be greater than 0")

        # Check if n is 0 => return 0
        elif n == 0:
                return 0

        # Check if n is 1,2 => return 1
        elif n == 1 or n == 2:
                return 1

        else:
                return Fibonacci(n-1) + Fibonacci(n-2)

# ========================================== #

# Time Testing Function
def timeofFibonacci(n):
    start_time = time.perf_counter()
    Fibonacci(n)
    end_time = time.perf_counter()
    return end_time - start_time

# Result up to a certain 'n' value
def resultofTest(n):
        print("\tFib Number\t|\tTime Taken")
        for i in range(n+1):
                # print("\t", i, "\t\t", timeofFibonacci(i))
                print(str(i) + "," + str(timeofFibonacci(i)))
                # print(timeofFibonacci(i))
        print()

# Driver Program
resultofTest(50)
```