Differences between Dockerized Containers and Virtual Machines

A performance analysis for hosting web-applications in a virtualized environment

Mohammad Al Burhan

18, May, 2020

Dept. Computer Science & Engineering
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the bachelor's degree in software engineering. The thesis is equivalent to 10 weeks of full-time studies.

Contact Information:
Author:
Mohammad Al Burhan
E-mail: moau17@student.bth.se

University advisor:
Kennet Henningsson
E-mail: kennet.henningsson@bth.se
Dept. Computer Science & Engineering

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

This is a bachelor thesis regarding the performance differences for hosting a web-application in a virtualized environment. We compare virtual machines against containers and observe their resource usage in categories such as CPU, RAM and disk storage in idle state and perform a range of computation experiments in which response times are measured from a series of request intervals. Response times are measured with the help of a web-application created in Python. The experiments are performed under both normal and stressed conditions to give a better indication in to which virtualized environment outperform the other during different scenarios.

The results show that virtual machines and containers remained close to each other in response times during the first request interval, but the containers outperformed virtual machines in terms of resource usages while in idle state, they had less of a burden on the host computer. They were also significantly more rapid in terms of response times. This is also most noticeable during stressed conditions in which the virtual machine almost doubled its sluggishness.

Keywords: Virtualization, containers, virtual machine, VirtualBox.

Nomenclature

ApacheBench (ab) Is a tool to stress- and load test web-applications and servers [18].

Application programming interface (API) API is an interface with definitions for interactions between multiple software intermediaries [28].

Central processing unit (CPU) The CPU, also known as a central processor, microprocessor or chip, is the unit that executes instructions to make up a computer program [24].

Container An instance of an image is also called a container, you can run multiple containers of the same image simultaneously [11].

Docker A platform for containerized applications [6].

Environment Refers to the environment of virtual machines and containers.

Host Referred to as the actual physical machine running the virtualized environment.

Hypervisor A hypervisor is referred to as a software, firmware or hardware that creates and runs virtual machines [4].

Image Is the setup of the virtual environment, a package with all the code and its dependencies required to run an application [11].

Kernel The kernel is a computer program that runs at the core of an OS with total control over the entire system [22].

Memory Swap (Swapping) When a computer runs out of physical memory, it uses virtual memory which stores the data in memory on a

disk. Reading the data from a disk is significantly slower than reading directly from memory [20].

Operating system (OS) An OS is the software that manages computer hardware and software resources allowing a user to run other applications on a computing device [21].

Oracle VirtualBox VirtualBox is a powerful virtualization platform for running a VM instance [1].

Python A programming language used to create different applications [29].

Random Access Memory (RAM) RAM is a form of computer memory where data is stored before it is being processed [23].

Stress-ng A stress testing utility tool for testing CPU, memory I/O and disk I/O [17].

Virtual machine (VM) Is referred to as an emulation of a computer system, an isolated duplicate of a real computer machine [5].

Contents

Chapter 1
Introduction

Virtualization means abstracting away an environment from the underlying architecture. The first concept of virtualization is believed to have emerged in the 1960s and early 1970s when International Business Machines Corporation, IBM devoted extensive amount of time and effort to develop vigorous and sturdy time-sharing solutions for their mainframes. Time-sharing represent a concept enabling computer resources to be shared and distributed among a vast group of users. The technology of virtualization has been in development for well over 60 years. At the present time, the IT industry is predominated by server virtualization with many companies moving towards fully virtualized cloud-managed IT ecosystems [1].

The popularity of virtualization expanded tremendously in the late 1990s because of the software company VMware Inc's release of VMware workstation which enabled virtualization of any x86/x64 architecture. The VMware workstation is a hosted hypervisor enabling users to set up virtual machines on a single physical machine and simultaneously use them alongside the host machine. With this ground-breaking technology, it was now possible to run Windows, Linux and MacOS on the same host hardware [2].

1.1 Purpose

The goal of this study is to gain knowledge about the two environments and their differences. There will be a comparison between virtual machines and virtualized containers to find the leading option within areas such as speed, scalability and the number of resources such as CPU, RAM and disk storage required to host a web-application, as well as understanding the underlying differences between the two environments. There will also be a focus on the user experience by measuring response times from the web-application while performing different scenarios explained under sub chapter 3.2.3 Setup, Scenarios.

The ones who will benefit the most from this study are software developers and IT administrators because this is generally within their field of knowledge and work tasks.

1.2 Research questions

The following research questions will be answered in this thesis:

RQ1 - How big is the demand on CPU, RAM and storage for running a web-application in a virtual machine or a containerized environment?

Finding the number of resources required for each instance is the key for providing a solid result. Having rapid response times affects the user experience which is of essence for both since they are competing against one another in the marketplace and performance is at high demand of the users [16].

RQ2 - How does the deployed environment enhance or impair the web-application running, while almost fully utilizing the given RAM and CPU capacity with respect to response times?

The value of this question comes down to understanding the underlying differences when there is little to no RAM [23] free'd and a busy CPU [24] to process the incoming request. This will give a clear indication on which of the two instances perform better under stressed conditions.

RQ3 - How does the two environments change under stressed workload while running several instances of virtual machines or containers with the same web-application?

RQ3 is about scalability which is a costly factor that many companies and users like to skimp on in their early stages when developing a service which that leads to having to buy more physical servers to meet the required demands. Results from this research question will clarify which of the two environments is more practical for running multiple instances of the same web-application therefore providing better scalability in terms of storage, RAM and CPU performance.

1.3 Scope/Delimitations

This thesis's scope is limited to answer the specific research questions in the previous section 1.2 Research questions. The main focus will be on resource usages and fundamental differences between the two environments.

In order to create a fair comparison, a web-application will be created and used to emulate a service that is going to be deployed using a virtual machine and a container. The reason for choosing a web-application is dependent on it being easy to create and used for producing the results.

Since there are many different virtual machines to pick, the chosen one for this thesis is Oracle VM VirtualBox [12], the reason being is that it is "easy to use"

and install as well as being a well-established virtual machine software. When it comes to container types the choice, fell for Docker as it is widely used and provide an easy starting process.

A more profound discussion about the reasons behind the differences is beyond the scope of this study and will not be conducted.

Chapter 2

Theory

This chapter will introduce and describe the key concepts the reader needs to understand when reading this thesis.

2.1 Hypervisor

IBM invented the hypervisor in the 1960s for its mainframe computers [1]. A hypervisor is a software layer that enables multiple operating systems to run simultaneously on the same host hardware. The hypervisor is also known as a virtual machine monitor (VMM) that manages virtual machines (VMs) while they are running alongside each other. It assigns each VM its own slice of the underlying computing power, memory, network and storage [4]. The reason is to prevent the VMs from interfering with one another; so for instance, if one VM crashes or is compromised in a way, it will not affect the other VMs that are running.

There are two types of hypervisors as explained in the following sub chapters.

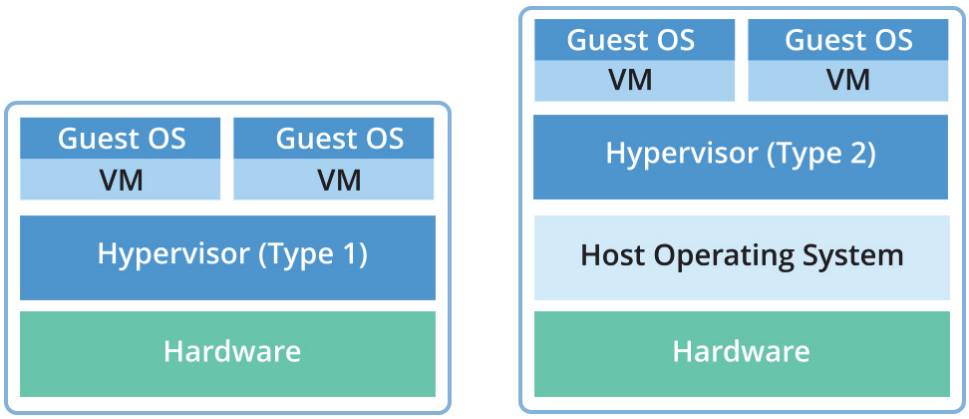

Figure 2.1: Type 1 hypervisor (bare metal) and Type 2 hypervisor (hosted) architecture.

2.1.1 Type 1 hypervisors (bare-metal)

The Type 1 hypervisors, also referred to as "bare-metal" hypervisors interact with the underlying physical resources and replaces the traditional operating system collectively, hence the reason being named bare-metal hypervisors. These hypervisors are more efficient and secure than the other type of hypervisors since they operate with access to the physical hardware and are also considered to be the best-performing hypervisors for enterprise computing [4].

Hypervisors such as Microsoft Hyper-V server, VMware ESXi and open source KVM are categorised as Type 1 hypervisors [4].

2.1.2 Type 2 hypervisors (hosted)

Type 2 hypervisors run as an application on an existing operating system. Generally referred to as "hosted" hypervisors because they require the use of a pre-existing operating system to access and coordinate the underlying hardware resources [4].

Because of their nature in operating on top of an existing OS, they also carry a performance overhead. This type of hypervisors are generally not considered applicable for the enterprise computing but are rather used by client or end-user systems where performance and security is less of an issue [4]. For instance, a software developer might utilize a Type 2 hypervisor to create and manage VMs to simulate a specific scenario or test a product prior to release. There is a broad selection of Type 2 hypervisors, including VMware's Workstation and Oracle VM VirtualBox [4, 12].

2.2 Virtual Machine

A virtual machine is a replica of a computer system that runs on top of another system, often referred to as the `host`. A VM requires its own operating system and has access to the required resources to run such as the host CPU, RAM, physical or virtual disk drives for storage and a virtual or a real network interface. These are typically the base requirements for a VM [5]. But instead of using a computer's hardware to fully function, a VM relies on software to run its processes with the help of a hypervisor.

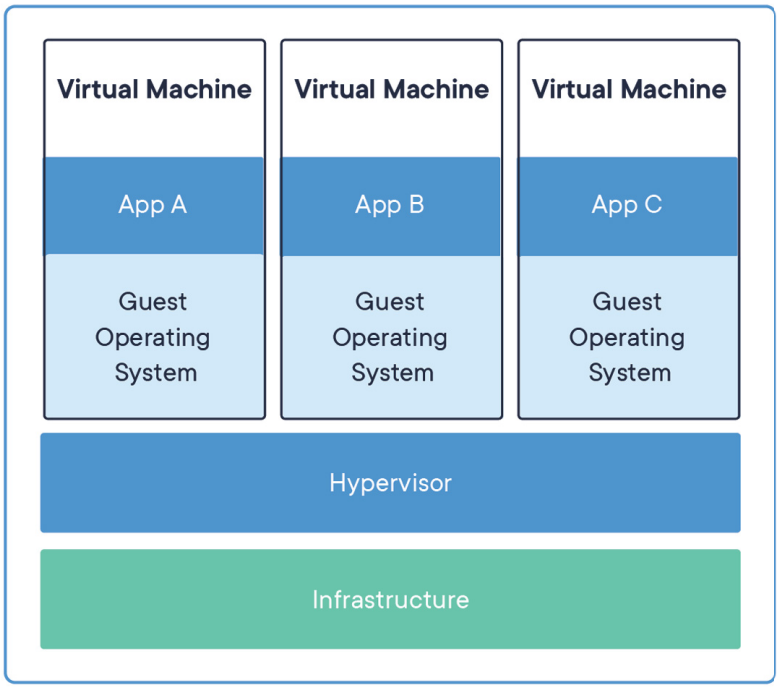

Figure 2.2: Layer structure of virtual machines.

2.3 Docker

Docker first emerged in 2013 and was released as an open-source platform in March 2013 by dotCloud, a platform-as-a-service company. In its early releasing stages Docker only supported Linux kernels, it relies on Linux kernel features to ensure resource isolation and packaging for an application with its dependencies [8]. However, that has since changed and Docker is as of now available for Windows platforms by utilizing the features of Microsoft's own hypervisor, Hyper-V [19]. Docker uses a scripting language to define an image and what should exist in a container, it is also possible to extend other images and do further configuration [13].

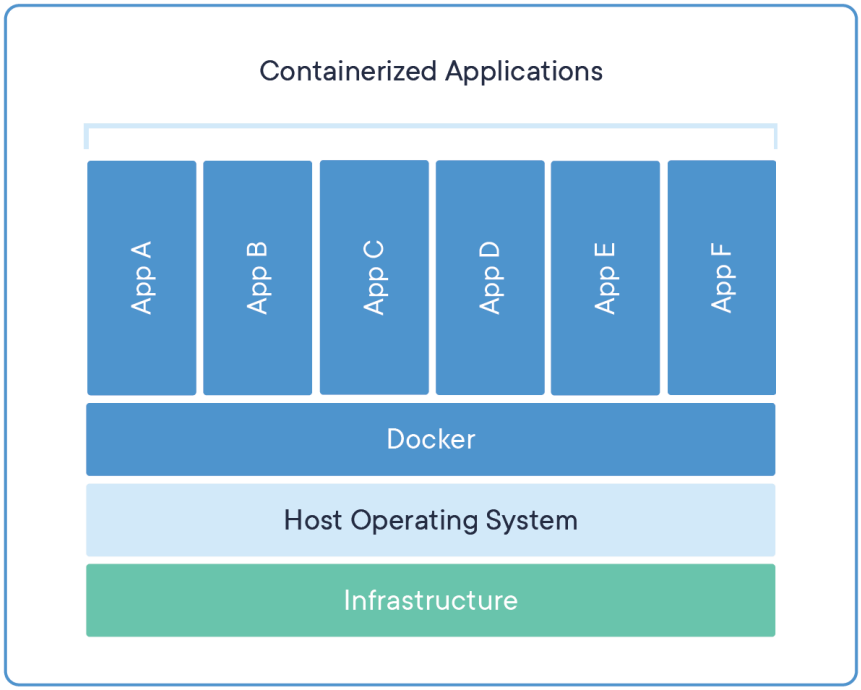

Figure 2.3: Layer structure of Docker engine.

Docker has three main components that we should know about in relation to this thesis.

Dockerfile

The Dockerfile instructs Docker to follow a specific set of commands to build an image. It can include instructions to copy files to the image, install additional packages to be a part of the same package along with a broad selection of other commands [13, 14].

Docker image

A Docker image is a file and the result of the instructions supplied by the Dockerfile. An image is comprised of multiple layers and is essentially a complete executable version of an application since it includes all the required dependencies to run an application including code, configuration files, other libraries and instructions [11].

Docker container

Docker containers are actually the running instances of Docker images. It shares the kernel with other containers and runs as an isolated process on the host OS [11].

It is also possible to have multiple containers of the same image running simultaneously.

Chapter 3

Method

This chapter contains a description of the methods and approaches used in this thesis to answer the research questions.

3.1 Theoretical

The theoretical approach for providing answers for the research questions through relevant academic literature, papers and blogs with the use of keywords such as "Docker", "virtual machine", "virtualization", "scalability", "comparison", "resource efficiency". A combination of these keywords is used to search through the literature database BTH Summon and Google Scholar to find relevant articles. The data gathered from the results will lay the base for answering the research questions. The criteria for finding relevant literature is that a comparison is made between virtual machines and containers as well as contain information related to the research questions.

3.2 Empirical

This thesis will focus on a comparison between two different application host environments through experiments using an empirical method. The experiments are conducted ten times. The acquired results from the comparison will then be analyzed and be used to give a definitive answer to each research question.

In order to create a fair comparison, each instance will be observed while running resource intensive tasks using the stress testing tools in the background while performing the different scenarios with the help of a web-application explained below. There will also be an observation done of how each respective instance behaves while idling. The parameters that will be looked at are the usages of both the CPU and RAM alongside storage and response times.

3.2.1 Hardware and software

Name	Version
Operating System (OS)	Ubuntu 18.04 LTS
Processor (CPU)	AMD Ryzen 7 1700 3,9GHz
Memory (RAM)	16 GB 3000 MHz DDR4
Graphics	NVIDIA GeForce GTX 1080
Storage (Primary)	Samsung SSD 860 EVO 1TB
Storage (Secondary)	Seagate Barracuda ST3000DM007 3TB
Docker Engine	19.03.8
Oracle VM Virtualbox	6.1.8
Virtual Machine OS	Ubuntu 18.04 LTS

Table 3.1: Hardware and software specifications used when running the experiments during this thesis.

3.2.2 Setup

Configuration

In order to give each instance equal conditions, they will get assigned the same number of resources as shown in the following table.

Instance type	CPU cores	RAM	Storage	Swap memory limit
Virtual Machine	2	2 GB	10 GB	2 GB
Docker Container	2	2 GB	10 GB	2 GB

Table 3.2: Configuration overview for virtual machines and containers.

Docker

The Docker image which contains our web-application is configured to run with a certain amount of memory (See configuration table 3.2) inside of a container. By default, Docker instructs the kernel to kill its process if the container consumes all host memory [15].

However, since we aim for a fair comparison between containers and virtual machines, this option was disabled by allocating a specific amount of memory to the container and instruct Docker through the option `--oom-kill-disable` not to kill the container process if it runs out of memory [15].

This enables a similar behaviour for containers to swap memory [20] much like the virtual machines in case they run out of memory.

Virtual machine

Because we need to access our web-application from outside the VM, additional network configuration is required. Virtualbox offers multiple networking modes [27], for this experiment the networking mode `Bridged networking` was best fit to allow `ApacheBench` access the web-application from the host machine.

3.2.3 Scenarios

A range of scenarios are performed to answer specific research questions and are equally created for the two types of environments, while using stress testing tools and measuring the acquired results for a more thorough analysis which is later going to be presented using different graphs and tables. The results represent the average values from the acquired data.

RQ1 and RQ2 scenarios involve a single instance of each representative environment and lastly, RQ3 scenario increases the number of instances.

RQ1 Scenario

RQ1 scenario consists of two phases. The first phase will monitor the behaviour of each instance type and last phase is comprised of data gathering in form of response times which will be analyzed later.

Phase one

Phase one is to observe how the two types of instances behave while idling which includes the usage of system resources such as CPU, RAM and disk storage.

Phase two

Starting with 1 concurrent request (1 visitor) to establish a baseline and gradually increase the number of concurrent requests made on the web-application API with the `ApacheBench` tool in order to analyze which environment requires the CPU and RAM resources. By increasing the intervals and observing the result resource usage along with time per request, a better definitive and consistent result will give a more prominent understanding on the differences.

The intervals are defined as following and will always consist of 100 requests per interval.

Interval number	Concurrent visitor(s)	Sequential requests	Total requests
1	1	100	100
2	10	10	100
3	25	4	100
4	50	2	100

Table 3.3: Intervals overview for conducting RQ1 Scenario experiments.

RQ2 Scenario

Restricting maximum allowed usage of RAM to two gigabytes and two CPU cores and running the `stress-ng` tool to utilize 75% of their entire capacity, while measuring the response times from the web-application using `ApacheBench`.

RQ3 Scenario

Running multiple instances of each environment and simultaneously conducting the scenarios from RQ1 and RQ2 above and analyze the impact.

3.2.4 Experiment

Web-application

The web-application will be created using a well established framework in Python [29] called Flask [25] that uses SQLite3 [26] as a database engine. Flask is installed via command `pip install -r requirements.txt`. It will consist of an API [28] for retrieving data that is randomly generated and saved in the database in advance before conducting the experiments. The data consist of `1,000` rows with random user data such as (first- and last name, email and password).

The application processes incoming requests by first opening a connection to the SQLite3 database and then retrieves all the rows within the `Users` table in which the random data is stored. Lastly, the rows are collected and the connection is thereby closed since we have finished the retrieval operation and a response is sent back containing all the rows.

RQ1 Scenario

RQ1 scenario requires `ApacheBench` which is installed through the terminal with the following command `sudo apt-get install apache2-utils`. The installed version is `2.3`.

Each interval was setup according to sub chapter 3.2.3 Setup, RQ1 Scenario and configured with `ApacheBench` using `ab -l -r -n {total number of requests} -c {concurrent users} -k {website URL}`.

RQ2 Scenario

RQ2 scenario sets the base of the results in which the two types of environments get compared in regards to stressed conditions. In order to apply these conditions, we will use `stress-ng` [17] which is included in the Docker image, but had to separately be installed via the following command on the virtual machines `sudo apt-get install -y stress-ng` to aid in utilizing a set of percentage of both CPU and RAM resources. The version `0.09.25` of `stress-ng` was configured to run for 60 seconds as following:

CPU stress

```
stress-ng --cpu 2 -l {load capacity in percent} -t 60s
```

RAM stress

```
stress-ng --vm 1 --vm-bytes {load capacity in percent} --vm-method
all --verify -t 60s
```

RQ3 Scenario

Our last scenario combines the previous two scenarios and applies them for multiple instances of each environment type. This is what the scalability and performance evaluation is based upon.

Chapter 4

Results

This chapter will present the results of containers and virtual machines in two different phases. The first phase will go over the graphs and description of the idle state of the CPU, RAM and disk storage with a comparison of the results. The second part will give insight to the results of how each representative response time was affected under normal and stressed conditions. Last result is the behaviour of multiple instances of each environment in two phases.

Since the experiments are repeated 10 times, the results could deviate a bit, however, the differences are too small to even be considered as deviations.

4.1 RQ1 Scenario

4.1.1 Phase one

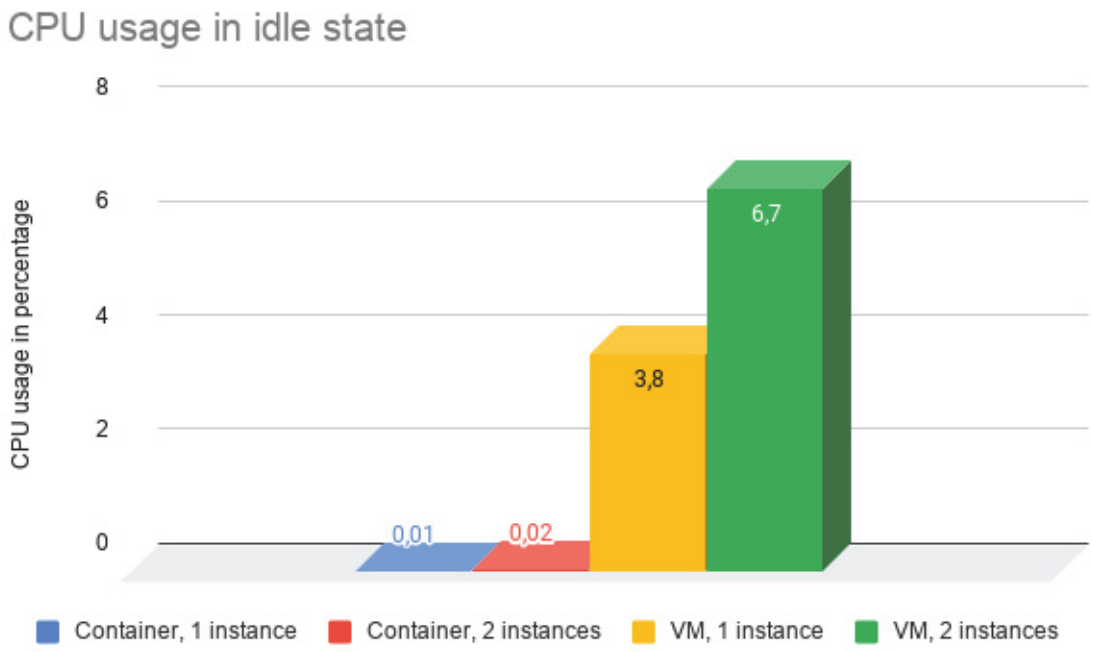

Figure 4.1: Graph overview for CPU usage in percentage, in idle state.

The graph displays the difference between containers and the virtual machines. Comparing these is almost trivial as the containers are using much less processing power in the idle state. The single container instance requires about 198.95% less resources than one VM instance.

Two container instances are 198.81% more efficient in the use of computation resources in contrast to the two VM instances.

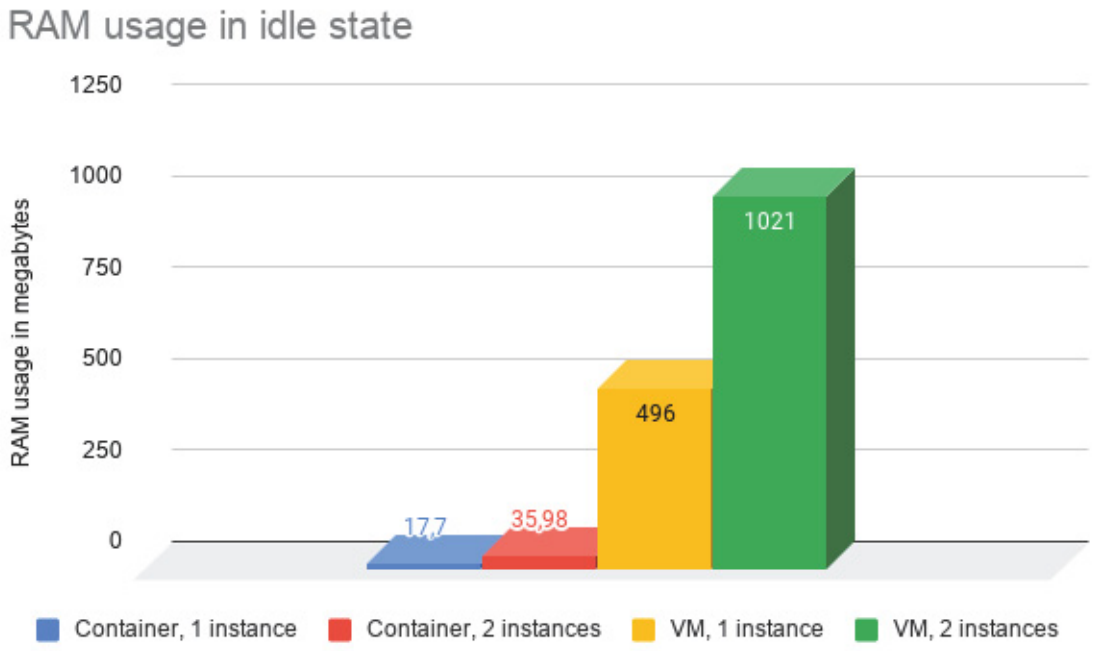

Figure 4.2: Graph overview for RAM usage in megabytes, in idle state.

This should come to no surprise as of looking back at the results from CPU usage in idle state. The containers utilization is very small in difference from the virtual machines. The single container instance is using 17,7MB while a VM instance is at 496MB. There no competition with two instances either as the containers are at 35,98MB indifference from the virtual machines that got 1021MB.

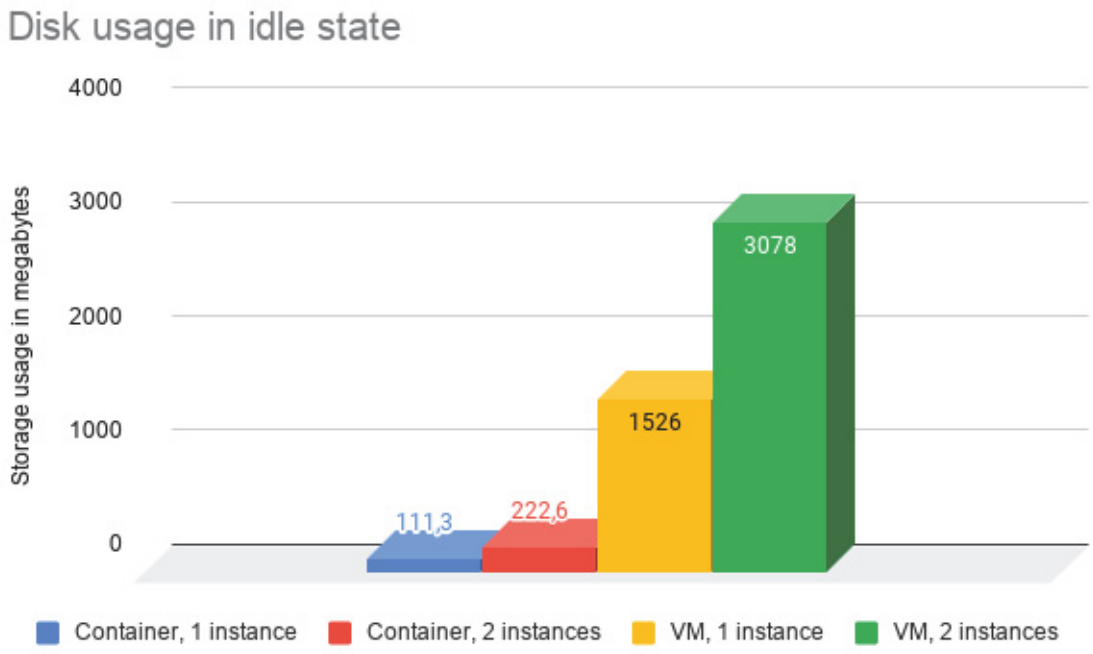

Figure 4.3: Graph showing disk storage usage in megabytes

Results of the disk storage usage are again, no different from the previous results. The containers rule this category as well. The single container instance is at 111,3MB usage which is well below the 1526MB of one VM. Having two instances shows that both the containers and the VM's nearly doubles in comparison from the single instance in usage and the gap increased vastly.

4.1.2 Phase two

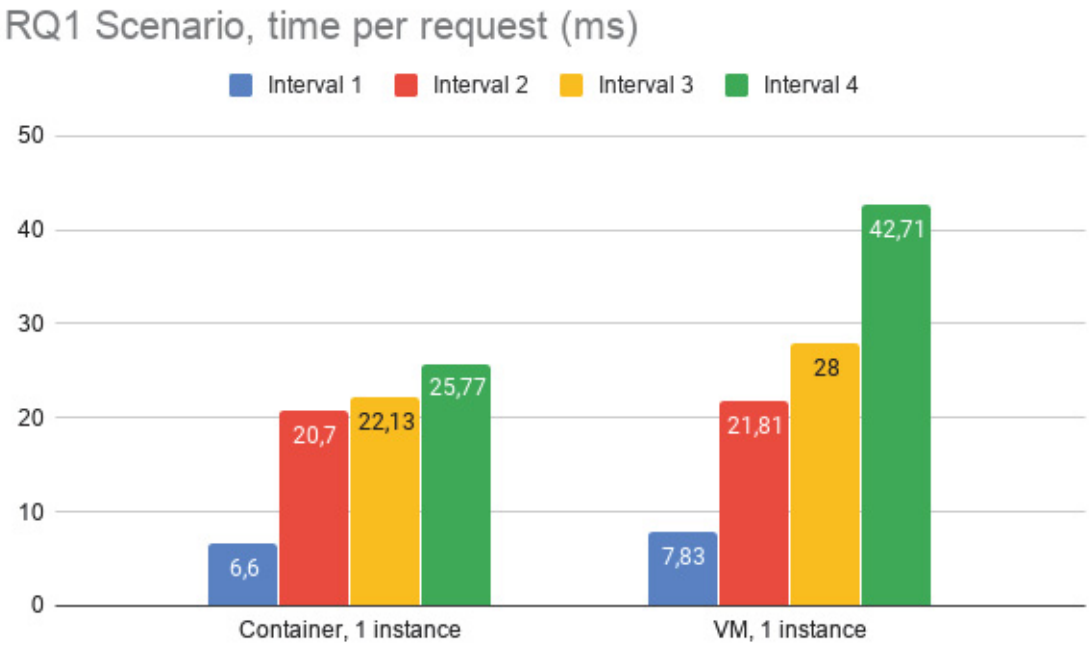

Figure 4.4: Graph showing time per request in milliseconds.

At first glance of interval one, there is around 17% difference between the container and VM. However, in the second interval the VM is starting to close the gap with a smaller percentage of 5.2%. The difference increased as the intervals incremented.

4.2 RQ2 Scenario

In this section, the results of the second scenario are provided in graphs with an explanation of the factual differences from the stress tests.

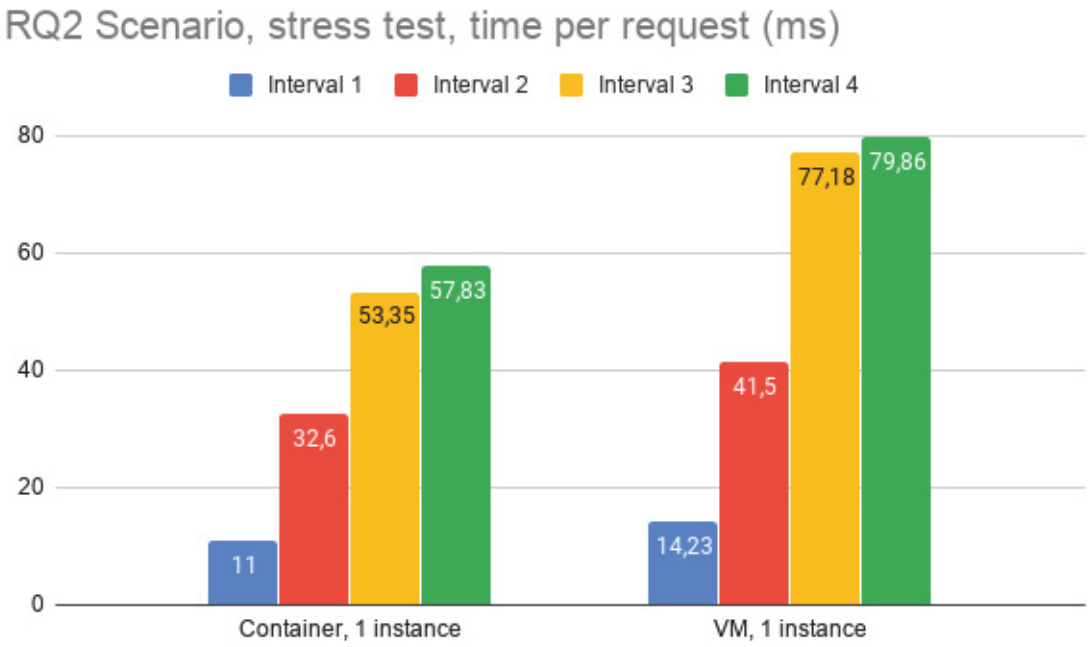

Figure 4.5: Graph showing time per request under stressed conditions.

The graph shows how the container handles the stress test in different intervals in comparison to the virtual machine. Following the intervals as they rise there is not that big difference at interval 1 and 2, but after that the VM fell behind on interval 3 and 4 with a considerably higher response time compared to the container. Interval 3 showing the container being around 0.023 second faster. The last interval displays a lower difference between the VM and the container than the previous interval.

4.3 RQ3 Scenario

4.3.1 Phase one

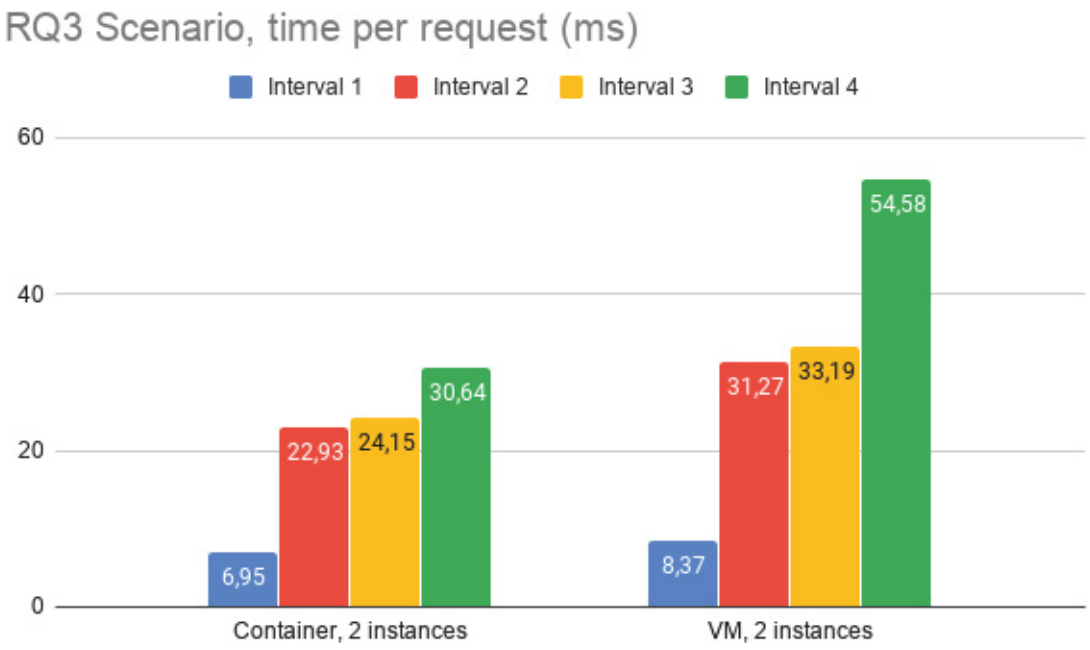

Figure 4.6: Graph showing time per request using multiple instances of each environment

Now, the requests of two instances simultaneously. The graph displays the containers being faster on all the intervals and the only time the VM's are close is at the first interval, after that the gap increases. In the fourth interval the VM's took a huge leap making them almost twice as slow compared to the containers.

4.3.2 Phase two

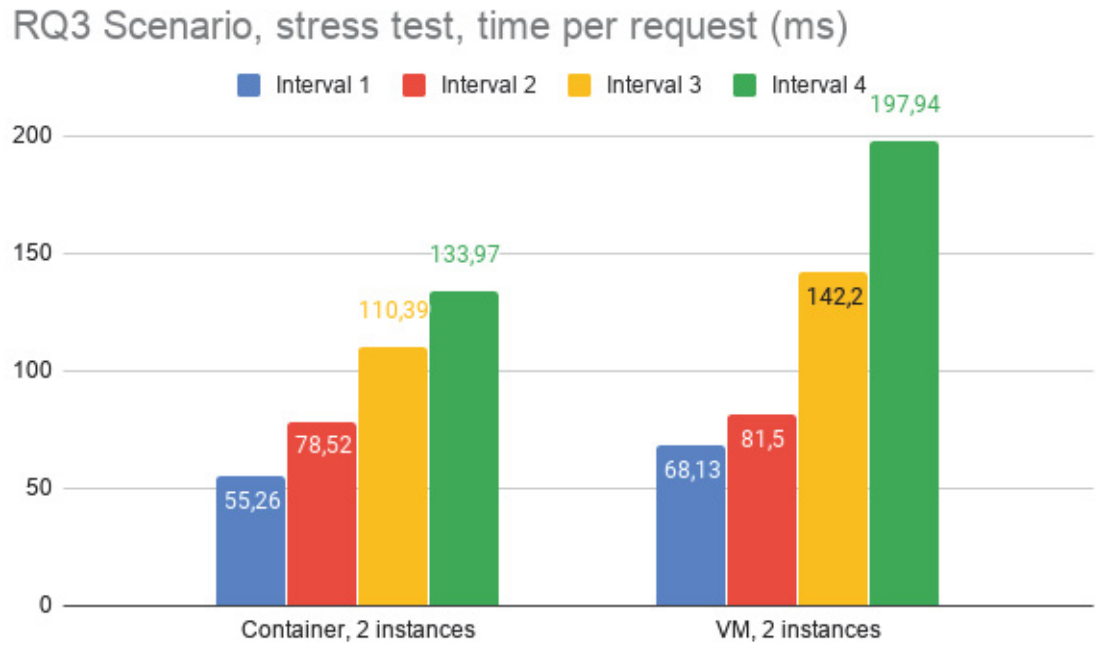

Figure 4.7: Graph showing time per request using multiple instances of each environment under stressed conditions.

When two instances are put under stress, the results are still in favor of the containers in all intervals. There is not really any interval where the VM's are close to over taking the containers. VM's are the closest to the containers at interval 2, but began to fall behind in the interval 3 making this one of the lesser competitive results between containers and VM's.

Chapter 5
Analysis and Discussion

This chapter will start by presenting a more in-depth analysis over all research questions one by one. Lastly, a summary will be presented along with validity threats that could impact the result of this thesis.

5.1 Research questions

5.1.1 RQ1 - How big is the demand on CPU, RAM and storage for running an application in a virtual machine or a containerized environment?

The difference is huge in all of the idle states for storage, RAM and CPU. This is because the Virtualbox VM is a `type two hypervisor` [4] and operates on top of the current operating system which brings some performance overhead compared to what the container can utilize. The response times are just slightly different for the VM's, but still a bit faster on the containers.

Instance type	Interval 1	Interval 2	Interval 3	Interval 4
Container, 1 instance	6,6	20,7	22,13	25,77
VM, 1 instance	7,83	21,81	28	42,71

Table 5.1: Table showing time per request in milliseconds from RQ1 scenario.

The contributing factor is that containers can access the hardware. The extra strain for the virtual machine comes from not having that and working on top of the existing host OS [4].

5.1.2 RQ2 - How does the deployed environment enhance or impair the web-application running, while almost fully utilizing the given RAM and CPU capacity with respect to response times?

The results are showing that the containers are faster and more stable when stressed, this is related to the containers not having to operate on top of another OS which will consume extra resources as well as having direct access to the hardware [4]. The virtual machine falls behind as the intervals increments and the underlying cause for becoming more unstable is that they require more hardware resources by their design in being a type two hypervisor, thereby not having direct access to the physical hardware.

Instance type	Interval 1	Interval 2	Interval 3	Interval 4
Container, 1 instance	11	32,6	53,35	57,83
VM, 1 instance	14,23	41,5	77,18	79,86

Table 5.2: Table showing time per request in milliseconds under stressed conditions from RQ2 scenario.

5.1.3 RQ3 - How does the two environments change under stressed workload while running several instances of virtual machines or containers with the same web-application?

When trying to see if there would be any difference when running multiple instances there are no surprises to the theory in which the container is more stable and faster overall. The VM's became more unstable as the interval increase which can be seen within the last interval. This is related to having not much more hardware to use before reaching the capacity.

Instance type	Interval 1	Interval 2	Interval 3	Interval 4
Container, 2 instances	55,26	78,52	110,39	133,97
VM, 2 instances	68,13	81,5	142,2	197,94

Table 5.3: Table showing time per request in milliseconds under stressed conditions from RQ3 scenario.

Running double instances of a VM has a greater burden for the reason of having to run on top of the host OS which requires more resources [4] than the containers that are simply tapping into the hardware.

5.2 Summary

In the first scenario there were no surprises to the theory which predicted that the containers being faster and more stable compared to the VM's, but it was interesting to see how the containers handles all tasks in scenario one much more efficient, just in the idle state the VM is far behind from the containers. Being hundreds of percentages behind the containers in terms of CPU, RAM and storage usage.

In the second scenario when stress testing at the volume of one instance, they got pretty close at first with just a few milliseconds putting them apart, but as the interval increments the gap became obvious and the container instance was much faster when put under stressed conditions.

The third and last scenario has raised the number of instances to two and what can be seen here is that the time per request without stressed conditions in performance is very different in the last interval where the VM is almost twice as slow as the container. Besides that, the VM is not to far off in comparison to the container. But clearly as the interval goes up so does the margin between them. When both are put under stressed conditions, the containers are overall faster than the VM's. They are however pretty close in the second interval, but besides that the containers crush the VM's.

5.3 Validity threats

Blogs are not peer reviewed and cannot be verified, but it is also where interesting discussions regarding virtual machines and containers take place to get different perspectives.

The web-application could also affect the results if it is not setup correctly. To ensure that the web-application does not adversely affect the results, it will be created using a well-established Python framework [25] and follow the guidelines and best-practices provided by the framework on their official documentation page.

Chapter 6
Conclusions and Future Work

This thesis's main objective is finding the differences between Docker containers and virtual machines. The chosen VM software was VirtualBox which is a hypervisor type two. The hypervisor type two is working on top of the existing host operating system which brings some performance overhead that is interesting to investigate.

In order to find the differences, the experiment entails three scenarios in which the first and last scenario contain two phases while the second scenario having only one phase. The first scenario includes a first phase which is the observation of the idle state resource usages from the CPU, RAM and disk storage in a comparison between the virtual machine and container. The last phase in scenario one consist of monitoring the response times in steps of different intervals while running one instance.

In the second scenario the singular instance is put under stressed conditions and examine how the response times are reacting along the increased intervals under stress.

The final scenario involves multiple instances with one phase for measuring response times under normal conditions and the other with stressed conditions.

The method for the experiment sets the boundaries on the hardware and software which are being used in the scenarios. The web-application used is built in Flask with SQLite3 as the database engine containing one thousand rows of random generated data. To setup the interval of requests used in the scenarios, `ApacheBench` was used. In order to simulate a specific stress level in the experiment `stress-ng` was applied.

For RQ1, one can conclude that VM's have a higher demand on system resources and therefore is less applicable for hosting a web-application. But the benefit of having a VM is the entirety of having the control over its operating system rather than only relying on the host OS, which comes with the cost of an obvious performance overhead.

When it comes to RQ2, the container and VM were put under stressed conditions

the outcome was in line with the theory in which the VM with hypervisor type two had some performance overhead regardless of stressed or normal conditions. The container instance was again, more effective in handling different request intervals as the container was put under stress in contrast to the VM.

To conclude RQ3, the outcome of this research question had similar patterns as the previous two research questions. In terms of resource usage in having two instances of each environment, the container instances were less of a burden to run on the host system than two VM's. The VM's doubled their resource usages as the instances increased.

This is also reflected when looking through the response times acquired under normal conditions where the containers yielded almost twice as fast response times contrary to VM's. Same patterns are also seen through the stressed conditions test. The containers took the lead in terms of response times.

As for the future, comparing the two hypervisor types one against another to investigate the performance and if the slower one has some benefits that could be worth the reduction. It could be its user friendliness, setup process et cetera. Another interesting take can be how the resource usage scales with more instances (rather than just looking at the response times). The other thing that could be interesting would be how to optimize the virtual machine configuration to work better and be less of a strain on a computer.

References

[1] Oracle - *What is virtualization?*, *Website* (English), 2012 Accessed 2020/02/24

[2] VMware, Inc. - *Understanding Full Virtualization, Paravirtualization and Hardware Assist*, *PDF* (English), 2007 Accessed 2020/02/24

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield - *"Xen and the art of virtualization," Proceedings of the ACM Symposium on Operating Systems Principles (p 164-177)*, *Website* (English), 2003 Accessed 2020/02/24

[4] International Business Machines Corporation - *Hypervisors: An Introduction*, *Website* (English), 2019 Accessed 2020/02/24

[5] Aaron Strong - *Containerization vs. Virtualization: What's the Difference?*, *Website* (English), 2019 Accessed 2020/02/24

[6] Docker Inc - *What is Docker*, *Website* (English), 2020 Accessed 2020/02/24

[7] Scott Hanselman - *VM Performance Checklist - Before you Complain that your Virtual Machine is Slow*, *Website* (English), 2007 Accessed 2020/02/24

[8] Nick Martin - *A brief history of Docker Containers' overnight success*, *Website* (English), 2020 Accessed 2020/02/24

[9] Jared Kobos - *When and Why to Use Docker*, *Website* (English), 2020 Accessed 2020/02/24

[10] Upguard - *Docker vs VM Ware: How Do They Stack Up?*, *Website* (English), 2020 Accessed 2020/02/24

[11] Docker Inc - *What is a container*, *Website* (English), 2020 Accessed 2020/02/24

[12] VirtualBox - *Manual/Documentation*, *Website* (English), 2020 Accessed 2020/02/24

[13] Docker Inc - *Dockerfile reference*, *Website* (English), 2020 Accessed 2020/02/24

[14] Docker Inc - *Best practices for writing Dockerfiles*, *Website* (English), 2020 Accessed 2020/02/24

[15] Docker Inc - *Runtime options with Memory, CPUs, and GPUs*, *Website* (English), 2020 Accessed 2020/02/25

[16] Aaron Strong - *Containerization vs. Virtualization: What's the Difference?*, *Website* (English), 2019 Accessed 2020/02/24

[17] Colin Ian King - *stress-ng, Introduction*, *Website*, (English), 2020 Accessed 2020/02/24

[18] Apache Software Foundation - *ab - Apache HTTP server benchmarking tool*, *PDF/Paper*, (English), 2020 Accessed 2020/02/24

[19] Microsoft - *Introduction to Hyper-V on Windows 10*, *Website*, (English), 2018 Accessed 2020/02/24

[20] ARPACI-DUSSEAU - *Memory Swapping*, *PDF*, (English), 2008 Accessed 2020/02/24

[21] Technopedia - *Operating system OS*, *Website*, (English), 2020 Accessed 2020/02/24

[22] LINFO - *Kernel Definition*, *Website*, (English), 2008 Accessed 2020/02/24

[23] Technopedia - *Random Access Memory - RAM*, *Website*, (English), 2017 Accessed 2020/02/24

[24] Technopedia - *Central processing unit - CPU*, *Website*, (English), 2020 Accessed 2020/02/24

[25] Pallets - *Flask Documentation*, *Website*, (English), 2020 Accessed 2020/02/24

[26] SQLite - *About SQLite*, *Website*, (English), 2020 Accessed 2020/02/24

[27] Virtualbox - *Chapter 6. Virtual Networking*, *Website*, (English), 2020 Accessed 2020/05/02

[28] Fisher, Sharon - *OS/2 EE to Get 3270 Interface Early*, *Academic journal*, (English), 1989 Accessed 2020/02/24

[29] Python.org - *About Python ™*, *Website*, (English), 2020 Accessed 2020/05/01

Appendix A

Tables

A.1 Results

A.1.1 RQ1 Scenario, resource usage in idle state

Instance type	CPU (%)	RAM (MB)	Disk storage (MB)
Container, 1 instance	0,01	17,7	111,3
Container, 2 instances	0,02	35,98	222,6
VM, 1 instance	3,8	496	1526
VM, 2 instances	6,7	1021	3078

Table A.1: RQ1 Scenario phase one.

A.1.2 RQ1 Scenario, time per request (ms)

Instance type	Interval 1	Interval 2	Interval 3	Interval 4
Container, 1 instance	6,6	20,7	22,13	25,77
VM, 1 instance	7,83	21,81	28	42,71

Table A.2: RQ1 Scenario phase two.

A.1.3 RQ2 Scenario, stress test, time per request (ms)

Instance type	Interval 1	Interval 2	Interval 3	Interval 4
Container, 1 instance	11	32,6	53,35	57,83
VM, 1 instance	14,23	41,5	77,18	79,86

Table A.3: RQ2 Scenario, stress test single instance.

A.1.4 RQ3 Scenario, multiple instances, time per request (ms)

Instance type	Interval 1	Interval 2	Interval 3	Interval 4
Container, 2 instances	6,95	22,93	24,15	30,64
VM, 2 instances	8,37	31,27	33,19	54,58

Table A.4: RQ3 Scenario, multiple instances under normal conditions.

A.1.5 RQ3 Scenario, stress test, multiple instances, time per request (ms)

Instance type	Interval 1	Interval 2	Interval 3	Interval 4
Container, 2 instances	55,26	78,52	110,39	133,97
VM, 2 instances	68,13	81,5	142,2	197,94

Table A.5: RQ3 Scenario, stress test multiple instances.

Appendix B

Scenario 1

This chapter contains all the necessary scripts used to conduct the scenario one experiment.

A.1 Monitor resource usage commands

Resource usage monitoring is used for phase one.

A.1.1 Docker containers resource usage

Storage

```
1 | docker ps --size
```

CPU and RAM

```
1 | docker stats
```

A.1.2 Virtual machines resource usage

Storage

```
1 | df -h
```

CPU and RAM

```
1 | top
```

B.2 scenario_1.sh

Measuring time per request using `ApacheBench` for the second phase of scenario one.

```
#!/bin/sh

SERVER_PORT=${1:-6001}
SITE="http://localhost:$SERVER_PORT/"
timestamp=`date +%H-%M-%S`
OUTFILE="scenario1/$timestamp-results-${SERVER_PORT}.txt"
DIVIDER="\n======================================================================\n"

if curl --output /dev/null --silent --head --fail "$SITE"; then
  echo "Performing SCENARIO 1 on: $SITE"
else
  echo "URL does not exist: $SITE"
  echo "Aborting scenarios"
  exit 1
fi

mkdir -p scenario1
touch $OUTFILE;

echo "1 concurrent user doing 100 page hits"
echo "1 concurrent user doing 100 page hits" > $OUTFILE
echo "This shows you how well the web-server will handle a simple load of 1
↪  user doing a number of page loads." >> $OUTFILE
echo "\n" >> $OUTFILE
ab -l -r -n 100 -c 1 -k $SITE >> $OUTFILE

sleep 5

echo $DIVIDER >> $OUTFILE
echo "10 concurrent users each doing 10 page hits"
echo "10 concurrent users each doing 10 page hits" >> $OUTFILE
echo "This is 100 page loads by 10 different concurrent users, each user is
↪  doing 10 sequential pages loads." >> $OUTFILE
echo "\n" >> $OUTFILE
ab -l -r -n 100 -c 10 -k $SITE >> $OUTFILE

sleep 5

echo $DIVIDER >> $OUTFILE
echo "25 concurrent users each doing 4 page hits"
echo "25 concurrent users each doing 4 page hits" >> $OUTFILE
echo "This is 100 page loads by 10 different concurrent users, each user is
↪  doing 4 sequential pages loads." >> $OUTFILE
echo "\n" >> $OUTFILE
ab -l -r -n 100 -c 25 -k $SITE >> $OUTFILE

sleep 5

echo $DIVIDER >> $OUTFILE
echo "50 concurrent users each doing 2 page hits"
echo "50 concurrent users each doing 2 page hits" >> $OUTFILE
```

```
49 echo "This is 100 page loads by 50 different concurrent users, each user is
   ↪  doing 2 sequential pages loads." >> $OUTFILE
50 echo "\n" >> $OUTFILE
51 ab -l -r -n 100 -c 50 -k $SITE >> $OUTFILE
52
53 sleep 5
54
55 echo $DIVIDER >> $OUTFILE
56 echo "SCENARIO 1 Complete."
```

Appendix C

Scenario 2

Scenario two is about putting the two environments under stressed conditions, the tool `stress-ng` was configured according to the following scripts.

A.1 stress_vm.sh

This section apply stressed conditions to the virtual machines.

```
#!/bin/sh

./stress_cpu.sh host &
./stress_ram.sh host &
```

B.2 stress_containers.sh

This script targets the containers and puts them under stress.

```
#!/bin/sh

./stress_cpu.sh &
./stress_ram.sh &
```

C.3 stress_ram.sh

The script for stressing RAM.

```
#!/bin/sh

image_name="stresstest_flask_app_benchmark"
stress_load_capacity=75
stress_time_seconds=60
stress_cmd="stress-ng --vm 1 --vm-bytes ${stress_load_capacity}% --vm-method
↪  all --verify -t ${stress_time_seconds}s"

echo "-------------------------------------------------------"
echo "Stress testing RAM at ${stress_load_capacity}% load capacity for
↪  $stress_time_seconds seconds"
```

```
10 echo "------------------------------------------------------"
11
12 if [ $# -eq 0 ]; then
13   echo "No arguments given, using default option to stress test Docker
     ↪  containers"
14   echo ""
15   echo "Executing command (${stress_cmd}) in each container"
16   echo ""
17
18   for container in `docker ps -q --filter ancestor=$image_name`; do
19     echo "Stressing container: $(docker inspect --format='{{.Name}}'
       ↪  $container)"
20     docker exec $container $stress_cmd &
21   done
22 else
23   echo "Executing command (${stress_cmd}) on host computer"
24   echo ""
25   eval $stress_cmd
26 fi
27
28 exit 0
```

D.4 stress_cpu.sh

The script for stressing CPU.

```
1  #!/bin/sh
2
3  image_name="stresstest_flask_app_benchmark"
4  stress_load_capacity=75
5  stress_time_seconds=60
6  stress_cmd="stress-ng --cpu 2 -l ${stress_load_capacity} -t
   ↪  ${stress_time_seconds}s"
7
8  echo "------------------------------------------------------"
9  echo "Stress testing CPU at ${stress_load_capacity}% load capacity for
   ↪  $stress_time_seconds seconds"
10 echo "------------------------------------------------------"
11
12 if [ $# -eq 0 ]; then
13   echo "No arguments given, using default option to stress test Docker
     ↪  containers"
14   echo ""
15   echo "Executing command (${stress_cmd}) in each container"
16   echo ""
17
18   for container in `docker ps -q --filter ancestor=$image_name`; do
19     echo "Stressing container: $(docker inspect --format='{{.Name}}'
       ↪  $container)"
20     docker exec $container $stress_cmd &
```

```
21    done
22  else
23    echo "Executing command (${stress_cmd}) on host computer"
24    echo ""
25    eval $stress_cmd
26  fi
27
28  exit 0
```

Appendix D

Scenario 3

This scenario is about having multiple instances running simultaneously.

The virtual machine was cloned in order to spawn multiple instances of it, this was done through the following simple steps:

1. Open VirtualBox.
2. Select the virtual machine to be cloned.
3. Right-click the virtual machine to be cloned and press `Clone` from the popup menu.
4. When prompted, give the clone a name, and press `Next`.
5. Select `Full clone` from the Clone type window, and press `Clone`.

A.1 start_multiple.sh

This scripts spawns two instances of the Docker containers.

```
1  #!/bin/sh
2
3  image_name="stresstest_flask_app_benchmark"
4  ram_limit="2g"
5  cpu_core_limit=2
6
7  docker run -d -p 6001:5000 --memory=$ram_limit --oom-kill-disable --cpus
   ↪  $cpu_core_limit $image_name
8  echo "Exposed container 1 via port 6001"
9
10 docker run -d -p 6002:5000 --memory=$ram_limit --oom-kill-disable --cpus
   ↪  $cpu_core_limit $image_name
11 echo "Exposed container 2 via port 6002"
```

Appendix E

Python code

This chapter contains all the Python code required to run the web-application used for the experiments.

A.1 app.py

```
1  import time
2  import sqlite3
3  from flask import Flask, jsonify, g
4  from flask_restful import Resource, Api
5
6  app = Flask(__name__)
7  api = Api(app)
8
9  @app.before_request
10 def before_request():
11     g.start = time.time()
12
13 def get_users():
14     db = sqlite3.connect('Users.db')
15     cur = db.cursor()
16     cur.execute('SELECT * FROM Users')
17     rows = cur.fetchall()
18     db.close()
19     return rows
20
21 class UserController(Resource):
22     def get(self):
23         users = get_users()
24         executionTime = time.time() - g.start
25         stats = {
26             "raw": executionTime,
27             "millis": int(executionTime * 1000)
28         }
29         return jsonify(a_execution_time=stats, data=users)
30
31 api.add_resource(UserController, '/')
32
```

```
33 if __name__ == '__main__':
34     app.run(debug=False, host='0.0.0.0')
```

B.2 requirements.txt

The additional packages including Flask had to be installed separately in the virtual machine through the command `pip install -r requirements.txt`. The contents of the requirements.txt are:

```
1 flask
2 flask_restful
```

Appendix F

Docker

A.1 Dockerfile

This chapter contains all the scripts and configuration files to build a Docker image and run a container.

```
FROM python:3.6-alpine
RUN apk update
RUN apk add --upgrade stress-ng

WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

B.2 build.sh

```
#!/bin/sh

image_name="stresstest_flask_app_benchmark"
docker build --no-cache -t $image_name:latest .
```

C.3 start.sh

```
#!/bin/sh

image_name="stresstest_flask_app_benchmark"
ram_limit="2g"
cpu_core_limit=2

docker run -d -p 6001:5000 --memory=$ram_limit --oom-kill-disable --cpus
↪  $cpu_core_limit $image_name
echo "Exposed container 1 via port 6001"
```

D.4 stop.sh

```
#!/bin/sh

image_name="stresstest_flask_app_benchmark"

docker stop $(docker ps -q --filter ancestor=$image_name)
```

Appendix G

SQLite database

The last chapter of the appendix contains the raw SQL code to create the database used for the web-application. However, not every row is included here simply because it is rather repeating having a thousand of rows displayed here.

The full version can however be found via the following ***reference***.

A.1 Users.sql

```
1  create table Users (
2          id VARCHAR(40),
3          first_name VARCHAR(50),
4          last_name VARCHAR(50),
5          email VARCHAR(50),
6          password VARCHAR(50)
7  );
8  ---
9  --- THIS IS RANDOMLY GENERATED DATA THROUGH https://www.mockaroo.com/
10 ---
11 insert into Users (id, first_name, last_name, email, password) values
   ↪  ('e2a96ce9-aaba-4528-8214-8e4b87f7563d', 'Glynda', 'Caplen',
   ↪  'gcaplen0@japanpost.jp', 'IerDCzkLe');
12 insert into Users (id, first_name, last_name, email, password) values
   ↪  ('2354a2bb-56d3-49b0-9190-36a77bd30501', 'Lynnette', 'Estabrook',
   ↪  'lestabrook1@elegantthemes.com', 'QyrmYp8');
13 insert into Users (id, first_name, last_name, email, password) values
   ↪  ('c83bdb71-212d-4617-930b-ae247c6933e9', 'Leah', 'Hymas',
   ↪  'lhymas2@nytimes.com', '4jnSwRiz8Jk');
14 insert into Users (id, first_name, last_name, email, password) values
   ↪  ('ce40991a-3205-4471-8c7f-86cdcf15d43c', 'Lillian', 'Brosini',
   ↪  'lbrosini3@latimes.com', '5dEDcY3q0XA');
15
16 --- Additional rows have been omitted, total number of rows is 1,000 ---
```