Cloud application platform - Virtualization vs Containerization

A comparison between application containers and virtual machines

Simon Vestman

Faculty of Computing
Blekinge Institute of Technology
SE-37179 Karlskrona Sweden
This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Bachelor in Software Engineering. The thesis is equivalent to 10 weeks of full time studies.

Contact Information:
Author:
Simon Vestman
E-mail: sive14@student.bth.se

External advisor:
Johan Nohave
Malvacom
E-mail: johan.nohave@malvacom.com

University advisor:
Per Olof Bengtsson
Department of Software Engineering (DIPT)
E-mail: per-olof.bengtsson@bth.se

Faculty of Computing
Blekinge Institute of Technology
SE-37179 Karlskrona Sweden
Internet : ww.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Abstract

Context. As the number of organizations using cloud application platforms to host their applications increases, the priority of distributing physical resources within those platforms is increasing simultaneously. The goal is to host a higher quantity of applications per physical server, while at the same time retain a satisfying rate of performance combined with certain scalability. The modern needs of customers occasionally also imply an assurance of certain privacy for their applications.

Objectives. In this study two types of instances for hosting applications in cloud application platforms, virtual machines and application containers, are comparatively analyzed. This investigation has the goal to expose advantages and disadvantages between the instances in order to determine which is more appropriate for being used in cloud application platforms, in terms of performance, scalability and user isolation.

Methods. The comparison is done on a server running Linux Ubuntu 16.04. The virtual machine is created using Devstack, a development environment of Openstack, while the application container is hosted by Docker. Each instance is running an apache web server for handling HTTP requests. The comparison is done by using different benchmark tools for different key usage scenarios and simultaneously observing the resource usage in respective instance.

Results. The results are produced by investigating the user isolation and resource occupation of respective instance, by examining the file system, active process handling and resource allocation after creation. Benchmark tools are executed locally on respective instance, for a performance comparison of the usage of physical resources. The amount of CPU operations executed within a given time is measured in order determine the processor performance, while the speed of read and write operations to the main memory is measured in order to determine the RAM performance. A file is also transmitted between host server and application in order to compare the network performance between respective instance, by examining the transfer speed of the file. Lastly a set of benchmark tools are executed on the host server to measure the HTTP server request handling performance and scalability of each instance. The amount of requests handled per second is observed, but also the resource usage for the request handling at an increasing rate of served requests and clients.

Conclusions. The virtual machine is a better choice for applications where privacy is a higher priority, due to the complete isolation and abstraction from the rest of the physical server. Virtual machines perform better in handling a higher quantity of requests per second, while application containers is faster in transferring files through network. The container requires a significantly lower amount of resources than the virtual machine in order to run and execute tasks, such as responding to HTTP requests. When it comes to scalability the preferred type of instance depends on the priority of key usage scenarios. Virtual machines have quicker response time for HTTP requests but application containers occupy less physical resources, which makes it logically possible to run a higher quantity of containers than virtual machines simultaneously on the same physical server.

Keywords: Cloud application platform, Openstack, Docker, Application container, Virtual machine
# Table of Contents

Abstract  2

1 Introduction  5
   1.1 Openstack vs. Docker  6
   1.2 Background  7
      1.2.1 Cloud application platform  7
      1.2.2 Openstack  7
      1.2.3 Docker  8

2 Research questions  10
   2.1 Reasoning for research questions  10

3 Method  11
   3.1 Theoretical Study  11
   3.2 Empirical Study  11
      3.2.1 Goal  11
      3.2.2 Setup  12
      3.2.3 Configuration  14
      3.2.4 Methodology  15

4 Literature review  18
   4.1 User space isolation against resource allocation  18
   4.2 Performance evaluation  19
   4.3 Scalability evaluation  19
   4.4 Summary from literature review  20

5 Results  21
   5.1 User space isolation and resource allocation  21
   5.2 Performance experiment  26
   5.3 Scalability experiment  31

6 Analysis  34
   6.1 User space isolation in comparison to resource allocation  34
      6.1.1 Analysis of the results  34
      6.1.2 Comparison to the literature review  35
      6.1.3 Answering the first research question  36
   6.2 Performance with respect to request handling  37
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.1 Analysis of the results</td>
<td>37</td>
</tr>
<tr>
<td>6.2.2 Comparison to the literature review</td>
<td>38</td>
</tr>
<tr>
<td>6.2.3 Answering the second research question</td>
<td>38</td>
</tr>
<tr>
<td>6.3 Scalability in terms of request handling speed</td>
<td>38</td>
</tr>
<tr>
<td>6.3.1 Analysis of the results</td>
<td>38</td>
</tr>
<tr>
<td>6.3.2 Comparison to the literature review</td>
<td>39</td>
</tr>
<tr>
<td>6.3.3 Answering the third research question</td>
<td>39</td>
</tr>
<tr>
<td>7 Conclusion</td>
<td>40</td>
</tr>
<tr>
<td>8 Future work</td>
<td>41</td>
</tr>
<tr>
<td>References</td>
<td>42</td>
</tr>
</tbody>
</table>
1 Introduction

This study investigates and exposes strengths and weaknesses of two type of instances for hosting applications in cloud application platforms, application containers and virtual machines. Motivations, for which of them fits better for different usage scenarios within cloud application platforms, are presented.

In recent years the popularity of cloud application platforms has increased, mostly due to services having higher accessibility. The priority lies in allowing a higher quantity of users to access the same software through internet connections of high speed without the need of providing computer software or hardware. The goal is to save money by reducing cost of time and hardware for deployment, while increasing scalability among applications intended to be used by a number of clients exceeding the normal rate of users [1].

As a consumer of a Platform as a Service provider, most of the times there is a need of certain privacy. It is important to retain a trust relationship between service provider and consumer, where the provider can assure the consumer that the data will not be exposed to third parties, sometimes not even to the provider [2]. This implies an abstraction so that the consumer is unable to access the underlying system or other applications from one application and that the underlying system is unable to access the individual applications without authentication.

Over the last years the amount of cloud computing infrastructures have increased, and with it the number of its customers. Virtualization has become an important part of cloud computing since virtualized resources are more efficiently manageable. Unfortunately virtual machines have the possibility to include reduced performance due to the extra levels of abstraction, therefore cloud computing providers sometimes prefer to make use of containerization, which on the contrary have the possibility to provide performance improvement by reducing the amount of resources required to execute tasks [11]. One of the goals in this study is to determine whether virtualization or containerization provides better performance at certain usage scenarios within a cloud application platform.

Scalability is defined by a system being able to handle workload in both growing and shrinking manner. The focus in cloud applications platforms is the behaviour of an application at increasing workload [16]. In modern companies scalability has been more prioritized over the last years, due to more people using cloud services as common providers. The need of quick responses while having the ability to serve high quantities of users is increasing incessantly, where poor load balancing is a well known bottleneck that can decrease the overall performance drastically. Also, well structured management of physical resources is as important and a key feature for high scalability [3].
The main purpose of this investigation is to gain knowledge about two different types of environments that can be used as Platform as a Service, Openstack using virtual machines and Docker using application containers. Companies that can benefit from this study are organizations providing cloud application platforms. They will have the chance to receive new, updated information about instances able to host applications. But also companies that in general host applications for any kind of customer have the ability to find valuable information in this paper, regarding how to host an application providing the best conditions for their customers [23].

1.1 Openstack vs. Docker

Openstack has increased significantly in private cloud areas. As it consists of several different parts configured to work together as one platform it allows users to manage multiple instances through one graphical user interface. Modern organizations consider it a feasible solution for private clouds, because the Openstack environment provides an effective resource management with high scalability [17]. There is a high amount of different virtual machines currently on the market [26], however because of the huge growth in popularity [27], Openstack has been chosen for this particular study.

Over the past years Docker has received a lot of attention from Google, Microsoft and Amazon. Docker containers share the operating system and important resources, such as depending libraries, drivers or binaries, with its host and therefore they occupy less physical resources. In cloud application platforms hosting a larger amount of applications per physical server this feature can be very useful [18]. The popularity and successful development is the reason, in this study, for hosting the application container in Docker, and comparing it to Openstack, but also to gain knowledge about different environments that can be used in cloud application platforms.
1.2 Background

The word “cloud” is often used in computer science to describe a pool of resources accessible from a distance. The concept of cloud computing started in the early 1970s with time-sharing used in Remote Job Entry. The idea was to allow programmers to use shared hardware resources for computing different tasks, where each user could connect to a specific socket and send requests to a job scheduler. The scheduler used a simple system for distributing requests and preventing the hardware from being monopolized by a single user [4]. In the early 1990s telecommunications companies began to use virtual private network services, allowing a certain optimization and distribution of available resources. It was not until the late 1990s and early 2000s that companies began to implement web-based retail services, where customers with internet access were able to use these applications from anywhere [5].

1.2.1 Cloud application platform

Cloud applications platforms fall into the category of Platform as a Service (PaaS), meaning that they offer space on a physical server for customers to use freely. Often the platform servers run environments with tools adapted to provide high scalability, also called application containers. In that way one server can host multiple applications while sharing the physical resources among the running containers, where the resources are distributed between applications in an effective way. The workload distribution system varies between different platforms with the same goal, to provide performance as promised to the customers [6][7].

1.2.2 Openstack

Openstack appeared for the first time in the early months of 2010, given the goal to provide a cloud computing platform that is free to use for anyone and fits both private and public networks, regardless of their size. The platform is built and compiled of separate programs intended for different tasks, which provides an infrastructure over physical computer resources, data partitioning, scaling and more for subscribers [8]. Figure 1 below describes how the key parts of Openstack work together to form one project. Each component plays an important role and is needed for the system to work as initially intended.
1.2.3 Docker

The first time Docker appeared was in 2013, by a company called dotCloud. Back then it was only supported on Linux kernels, but nowadays it is also available for Windows with Hyper-V capabilities. The aim was to create a certain level of portability so that an application could easily be moved between different Linux operating systems. The Docker engine creates an application layer that abstracts each application from the underlying system while it still shares the operating system of the host computer. In that way it provides portability between systems using the same kernel combined with a certain level of user space isolation and low resource usage. In figure 2 the layer structure of the Docker engine is presented and compared to the structure used by virtual machines [19][20].
Figure 2: Docker engine layer structure compared to virtualized layer structure
2 Research questions

The focus of the research is to find out if either application containers or virtual machines are more preferable for hosting applications in a Platform as a Service, by comparing the two instances in particular, evaluating user space isolation, performance and scalability.

The research questions of this study are:

- **RQ1**: What, if any, significant differences, in the aspects of user space isolation and resource allocation, can be observed from the comparison between application containers and virtual machines?
- **RQ2**: Which of the two types of instances, application containers or virtual machines, provides better CPU and RAM efficiency, with respect to request handling and file transferring speed?
- **RQ3**: Which of the two types of instances, application containers or virtual machines, provides better scalability in cloud application platforms, in terms of CPU and RAM usage in comparison to requests per second?

2.1 Reasoning for research questions

The research questions are designed to help understanding the main focus of this study and what kind of information to expect as a result. For answering the first research question, the two instances, application container and virtual machine, will be examined, where the importance lies in gaining knowledge about how they are placed and deployed in cloud application platforms, and, if any, what kind of user space isolation they provide. Also the resource occupation of each type of instance plays an important role as it is connected to the scalability of respective instance.

The goal of answering the second research question is to gain knowledge about how and with what efficiency the different types of instances are using the available physical resources. In this chapter, advantages and disadvantages will be exposed regarding the performance, observing mostly the CPU and RAM.

The third research question will be answered by running experiments evaluating the CPU and RAM usage in conjunction with the amount of requests handled per second. An important feature for applications is to be able to run consistent independent of the amount of workload, and therefore the third task will be an important investigation within cloud application platforms.

Hopefully the results to these questions will provide enough information for Platform as a Service providers to decide whether they should use application containers or virtual machines in future cloud application platforms.
3 Method

3.1 Theoretical Study

The theoretical method for answering the research questions contains intensively research through the literature database BTH Summon. This database was chosen because it provides search results from different popular scientific document providers, such as IEEE Xplore and Springer Nature. By using keywords related to the questions, the goal was to find relevant work and articles. The relevance, to this study, of found articles was determined by understanding the titles and reading the abstracts. The search was followed by snowballing through the references of found articles in order to gather more information about the subject and to determine if the sources were trustworthy. Criteria for identifying if literature is relevant was that a comparison between application containers and virtual machines must exist and contain information related to the requested information in the research questions.

Keywords used: Application container, virtual machine, performance, scalability, comparison, user isolation, abstraction, resource allocation, containerization, virtualization, cloud application platform, platform as a service, CPU, RAM, efficiency, request handling.

The theoretical study will mostly contain overall information about application containers and virtual machines so that a comparison between the ones used in the empirical study and other popular environments can take place.

3.2 Empirical Study

3.2.1 Goal

The empirical study is focusing on comparing two different types of application hosts deployed in two different environments. A comparison between virtual machine and application container is analyzed at different perspectives. The main focus is the usage and management of physical resources, such as the CPU and main memory. Respective instance is being observed while simultaneously receiving load in order to examine the behaviours in scenarios similar to a production environment. One of the goals is to compare these results to the gathered information from the theoretical study in order to expose possible differences between virtual machines in Openstack and virtual machines in other, similar environments. The same goes for application containers in Docker.
3.2.2 Setup

The experiments are implemented on a single computer, running the host operating system Linux Ubuntu 16.04 server. The Openstack and Docker environments are deployed side by side onto the server, running individually while executing the experiments.

The Openstack environment is a fresh install of Devstack, version 3.9.0. For this study an image of Linux Ubuntu 16.04 cloud is uploaded onto Glance, for which a volume is created in Cinder. Neutron is used to create a private network, in which two ports are published. One of the ports is used as internal interface in the router of the public network in order to gain external network access to the virtual machine. The other port is later assigned to the instance which provides an identity within its private network. In Nova an instance is created using the volume from Cinder and the private network in Neutron. Lastly a new floating IP is created in the public network and assigned to the virtual machine for external access. Glance, Cinder and Neutron are parts of Openstack and all manageable through the dashboard, Horizon. Figure 3 demonstrates how the different parts are set together after the virtual machine has been created.

![Figure 3: The different parts of Openstack used in the empirical study](image)

The Docker environment, on the other hand, is installed through the Linux Advanced Packaging Tool (apt). For this study version 17.04.0-ce, build 4845c56 is used. In order to be able to start an application container in Docker an image is created using a custom Dockerfile. As figure 4 describes, the new image specifies that the container should have SSH installed and available.
Figure 4: Dockerfile for application container

It is important that port 22 and 80 are being exposed in this study, while port 21 is optional for FTP connections. In the experiments port 22 will be used for connecting to the application container through SSH and port 80 will be used for sending and receiving HTTP requests. The last command in figure 4 specifies that as soon as the container is created the SSH service should be started. The image is forked from a standard Linux Ubuntu 16.04 image, since it is also used as host operating system. After the image is created the application container is started and a specific port in the external network of the host server is assigned to each port of the container. In that way it is possible to SSH into port 22 of the container by connecting to the assigned port in the external network, owned by the host operating system. The final structure is demonstrated in figure 5.
For the performance and scalability experiments each instance has an Apache web server installed for handling HTTP requests. The server contains one single page, which in the case of a received request determines the request method and sends a response accordingly.

### 3.2.3 Configuration

For a fair comparison respective instance is assigned the same amount of CPU cores and main memory. However the maximum amount of available storage space differs, but is insignificant for the experiments. In Openstack a new volume is created with a fixed size, while in Docker the complete hard drive partition of the host operating system is indicated as available for use.

**Server and instance configuration:**

- **Host server**
  - Motherboard
    - Product: MSI B85M-E45
  - CPU
    - Product: Intel Core i3-4150
    - Speed: 3.50 GHz
    - Cores: 2
    - Threads: 4
  - RAM
    - Type: DDR3 Synchronous
    - Speed: 1600 MHz
- Size: 12 GB
  - Network
    - Product: RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller
  - Storage
    - Product: Seagate ATA Disk
    - Speed: 7200 RPM
    - Size: 1 TB
  - Operating system
    - Product: Linux Ubuntu 16.04.2 LTS
    - Kernel: 4.4.0-72-generic
    - Architecture: x86_64 (64 Bit)

- **Virtual machine**
  - CPU: 4 virtual CPUs
  - RAM: 4 GB
  - Storage: 20 GB
  - Operating system: Linux Ubuntu 16.04.2 LTS

- **Application container**
  - CPU: 2 cores, 4 threads
  - RAM: 4 GB
  - Storage: 100 GB (shares main partition with host server)
  - Operating system: Linux Ubuntu 16.04.2 LTS

### 3.2.4 Methodology

Performance and scalability play important roles in cloud application platforms [11][16], and in many cases also the privacy of the hosted applications [2]. Therefore the empirical study is focusing on these attributes, where the central point is the usage of physical resources.

The following parts describe the approach for answering the research questions as defined in section 2 *Research questions*. The methods and tools used in the empirical study are presented in detail, with motivations for choices that were made.

**User space isolation and resource allocation**

For answering the first research question the deployment of respective instance is examined, as well as the resources required for creating them. The goal is to find out how isolated each instance is within a cloud application platform and how it shares the resources with its host.
The results for the user space isolation are produced by investigating the physical location of instances within the file system of the host operating system, but also by observing system information, such as active processes, in order to determine if the processes run by an instance are abstracted from the processes run by the host server. These are parts of the basic isolation contained in a standard Linux container [28]. Therefore these particular attributes were chosen to be investigated in the compared types of instances.

The resource allocation is determined by creating and firing up respective instance, followed by investigating the current resource usage in an idle state after creation. This helps understanding how much of the physical resources are required for spawning either virtual machines or application containers, but also provides first inputs for the scalability experiments.

**Performance**

The second research question is answered through running experiments using benchmark tools in order to determine the performance of the CPU and main memory in respective instance, followed by evaluation of requests handled per second through the Apache web server and the transfer speed of a large file through network.

For the CPU benchmarking a tool called stress-ng is executed with different time intervals, which allows examination of executed operations within given time interval. The periods are 10 to 60 seconds with 10 seconds leaps, and the executed operations are calculations where two 128 x 128 matrices of double float are multiplied by each other.

The main memory is benchmarked by using the tool mbw, which measures the RAM bandwidth by copying a certain size of data into the main memory and calculates the transfer speed in MB per second. In this experiment a size of 1024 MB is used, and the test is performed 10 times, in order to get a better computation of the overall transfer speed. In combination with the CPU experiment the results provide information about the usage of physical resources by respective instance.

The comparison of requests handled per second is done by benchmarking the Apache web server installed on each instance, by using the tool ab. A large amount of requests are sent from the host server to each instance, and the tool is calculating the average request handling speed in terms of requests handled per second. The amount of requests starts at 100.000 and continues up to 1.000.000 in total, where the average is calculated after each 100.000 requests. These requests are sent from 100 different connection simultaneously in order to experience a scenario more similar to a production environment.

In order to also gain knowledge about the network performance, a large file is transferred between respective instance and the host server. The goal is to measure the speed in ten seconds with one second intervals, in order to determine an average transferring speed in MB per
seconds. In this case a 5 GB large file is used. The file is transmitted by using rsync, a tool allowing remote file-copying.

This part of the empirical study is aimed for gaining knowledge about performance differences and similarities between application containers and virtual machines.

**Scalability**

In solving the third research question the focus lies in finding out how respective instance behaves at increasing load, observing the amount of requests handled per second and usage of CPU and RAM.

The experiment in this part of the study is done by sending HTTP requests, with the tool ab, to respective instance for the Apache web servers to handle. The requests are sent multiple times with different amount of connections, starting at 1,000 connections and increasing to 10,000 with an interval of 1,000. The amount of requests is always 100,000. At each interval the CPU and RAM usage is observed and the requests handled per second is measured. Subsequently the results are compared in order to determine which type of instance is performing better in scenarios of heavy load.
4 Literature review

4.1 User space isolation against resource allocation

Cloud application platforms generally have three facilities on how to host applications:

- **Version 1**: Host each application using a container running directly on the operating system run by the physical machine. Each container is a layer that contains the needed code and dependency for its application to run, where it shares the host operating system kernel with other containers running on the same machine.

- **Version 2**: Host each application inside its own virtual machine. Each virtual machine runs an operating system on their own, sharing physical resources with the operating system run by the physical computer.

- **Version 3**: A combination of version 1 and version 2, where each application is hosted inside a container running on a virtual machine, sharing the virtual machine with other containers.

Platform as a Service providers commonly choose version 3, as it combines some of the requirements of user space isolation and resource sharing [10], but generally implement the architecture dependent on the expected requirements of their customers.

In virtual machines, the running operating system is completely abstracted from the underlying system by virtualizing the physical resources and implementing their own drivers, binaries and libraries, resulting in high user space isolation. Processes run by the individual virtual machine are invisible for the host operating system, however a scheduler is used for resource reservation and to share execution time fairly with its host. The file system of a virtual machine is completely abstracted from the host computer by creating a new partition or volume for each instance. It is not possible to share libraries or files between an instance and the host [10][11].

For the reason of running an operating system of their own, virtual machines require a high amount of physical resources for deployment. In the matter of storage they need space for a complete operating system kernel and its dependencies, while in the matter of task execution they need execution time and main memory not only for the executed tasks, but also for the operating system to be kept active [12].

Application containers share the operating system and kernel with the host computer. Instead of creating a partition separately and using a different kernel, containers use kernel interfaces of the host system for user space isolation in terms of file system and resource management. This allows each instance to be isolated, so that from within a container only files
and processes owned by the instance are visible, however subdirectories can also be shared between host and guest [13][14].

Containers tend to consume only a small amount of physical resources. By sharing the operating system with the host computer including drivers, binaries and libraries, these instances themselves solely need resources for executing the assigned tasks [11].

### 4.2 Performance evaluation

In a study, where different types of containers were compared to a virtual machine in performance tests [15], it appears that application containers require less CPU time than virtual machines for executing a defined amount of operations. The virtual machine experiences a CPU efficiency of 1% below the containers, meaning that the CPU performance is almost the same between the two different types of instances, only deviating a minimal amount. The network I/O performance was also examined [15], where TCP data was sent from a server to respective instance. In the results it is clearly visible that application containers send and receive TCP data faster than virtual machines.

Another study shows that application containers are faster in RAM operations [11]. The test is accessing the main memory while measuring the memory speed in MB/s. The virtual machine turns out to be about 100 MB/s slower than the application container in accessing the main memory. The same study investigates the Apache performance of respective type of instance [11]. In this test HTTP requests were sent to respective instance in order to measure the number of requests handled per second. The container is handling almost 9,000 requests more per second than the virtual machine and therefore it performs better in Apache web server performance.

### 4.3 Scalability evaluation

A scalability comparison between virtual machines and Linux containers [16] demonstrates that containers outperform virtual machines, by being able to scale and process a service request much faster while experiencing a heavy load. In the test the CPU of respective instance was utilized with a load above 80% so that a scaling process would be triggered. As fast as the CPU reached 100% a new instance was scaled up in order to handle subsequent requests. While a new container was scaled up in just eight seconds the virtual machine took three minutes.

An experiment [21] shows that virtual machines are increasing in response time corresponding to the message size. In the executed test messages were sent to a virtual machine, where the size of the messages was increasing exponential by doubling the capacity after each reply. The response time of the instance was raising almost equally to the size of the messages,
revealing that the latency is increasing linear corresponding to the amount of data sent. A similar study [22], comparing different virtualized and containerized environments, exposes that the response speed in containers relate to virtual machines at increasing message sizes. However the container-based environments are responding faster to requests, and the examined paper explains that the network isolation of virtual machines may be one possible reason.

### 4.4 Summary from literature review

Virtual machines provide a certain level of user space isolation that abstracts the instances, with all active processes and tasks, from the underlying operating system, by implementing a separate kernel for each instance and virtualizing the hardware. In addition a virtual machine distances its file system from the underlying system by requiring an own partition or volume for deployment, which provides even more isolation. Application containers do not provide the same amount of user space isolation, but is fairly close by using kernel interfaces in order to disallow access to the underlying system from within a container. However the separate operating system in virtual machines are still superior in user space isolation and therefore these kind of instances are to prefer in cloud application platforms that require a higher security and overall application privacy.

Application containers, on the contrary, are more lightweight in comparison to virtual machines, requiring less resources for deployment. In virtualized environments the abstraction leads to that except for the assigned tasks, a complete operating system has to be driven, causing higher resource consumption. A cloud application platform expecting to host a high quantity of instances independent of the performance of respective instance is urged to use application containers for hosting application.

In performance, regarding the CPU and main memory usage, application containers are shown to be superior to virtual machines, by executing CPU operations faster and requiring less time for accessing the RAM. The container-based environments do also perform better in receiving and responding TCP data through network, just as handling HTTP requests through an Apache web server. Application container is clearly the recommended type of instance for cloud application platforms having a high priority in performance.

Both types of instances seem to perform well in responding to messages at increasing size of messages. The response speed is consistent, however application container respond faster than virtual machines, leading to the conclusion that containers logically should be able to handle more requests per second at higher load. For cloud application platforms aiming for a high level of scalability, application containers are more likely to be able to handle a larger workload, while keeping the request speed at a satisfying rate, than virtual machines.
5 Results

5.1 User space isolation and resource allocation

Figure 6 displays all active processes, at the time the picture was taken, inside the virtual machine.

![Active processes in virtual machine](image)

Figure 6: Active processes in virtual machine

At startup the instance allocates virtual memory according to the assigned amount, meaning that the virtual machine will always try to use the physical memory, even if the host machine is out of free memory.

In figure 7 the disk usage of the virtual machine is exposed. At creation the disk space to be used is allocated so that it is free to use when the instance is running. This implies that virtual machines in Openstack always occupy the allocated space, independent of if it is used or not.
The virtual machine uses a separate partition for storage, labeled `vda1` inside the instance.

Figure 8 shows the active processes within the container.

The application container abstract its processes from the host operating system, however it completely shares the resources with its host. The resource usage of CPU and available
memory are exposed directly from the underlying system. The same goes for the disk space as shown in figure 9.

```
root@6d8f6f6298eb:~# df -h
Filesystem  Size  Used  Avail  Use%  Mounted on
none        92G   10G   77G    12%  /
tmpfs       5.9G   0  5.9G    0%  /dev
home        5.9G   0  5.9G    0%  /sys/fs/group
/dev/sda1   92G   10G   77G    12%  /etc/hosts
home        64M   0  64M    0%  /dev/shm
tmpfs       5.9G   0  5.9G    0%  /sys/firmware
```

Figure 9: Disk usage of Container

The container shares the hard drive volume of the host operating system, labeled as sda1, however the filesystem itself is isolated from the underlying system. Changes made to the container file system do not appear on the host.

By looking at the active processes of the host operating system while running an empty program in respective instance simultaneously it is visible that the application container runs its processes directly inside the host operating system, while the virtual machine has its own processes, completely abstracted from the host machine, creating a complete isolation of running tasks.

```
root@6d8f6f6298eb:~# top
USER      PR  NI  VIRT  RES  SHR S %CPU %MEM     TIME+ COMMAND
simon  20    0  0    0    0  0 S   0.0    0:00.01  0.00...```

Figure 10: Active processes of host operating system
In this test both instances executed the same program for reaching a high CPU consumption, where the first process in figure 10 is the running virtual machine and the second process is the executed program inside the application container.

The disk usage of the host server exposes more about how the instances are deployed, as visible in figure 11.

Figure 11: Disk usage of host server

For the virtual machine a new volume is created and listed by the server as sdc. However the application container is deployed directly into the main partition of the host server, listed as sda1 in figure 11.

Figure 12 and 13 show the results of a comparison of the allocated resources after creation and initialization of the two types of instances, application container and virtual machine.
Figure 12: Graph showing RAM usage of instances in Idle state

Figure 13: Graph showing disk usage of instance after creation
The resource usage in figure 12 is the approximately needed amount of memory in order to keep the container and virtual machine alive, while figure 13 shows the amount of disk space needed to run respective instance. The actual disk allocation for the virtual machine on the host system depends on the size of the volume attached to the instance. Because of the container sharing the kernel of the host operating system and only needing resources to run the assigned command it occupies significantly less resources than the virtual machine.

5.2 Performance experiment

Figure 14 describes the results of comparing virtual machine and container in CPU performance.

![Graph showing number of executed operations within a given time](image)

Figure 14: Graph showing number of executed operations within a given time

The application container and virtual machine are performing fairly the same when it comes to the CPU performance comparison. The containerized environment is executing the instructions just a tiny bit faster at a larger amount of executed operations.

The comparison of bandwidth in RAM operations is shown in figure 15.
The virtual machine turns out to perform a little bit faster in RAM operations than the application container. Also the application container experiences overhead at the start of the test.

The next figure contains results for the comparison of requests handled per seconds between virtual machine and application container.
According to the measurements the virtual machine shows a faster progress in handling HTTP requests. The application container is approximately 3000 requests per second slower in processing the workload. During this test the resource usage was recorded and it is clearly visible that the virtual machine requires more of the physical computer resources in order to process the HTTP requests.

Figure 16: Graph showing the requests handled per second by different instances
The virtual machine does have 4 virtual CPUs and the application container has 4 threads available for processing data, meaning that both types of instances have a maximum limit of 400% for the CPU usage. Figure 17 describes that the virtual machine uses a higher amount of the available CPU resources for processing the same amount of data as the application container. Figure 18 exposes the RAM usage of respective instance for the same test.
The application container appears to use less than a ninth of the amount of RAM than the virtual machine does. Otherwise both types of instances seem to have a consistent usage of main memory for handling HTTP requests.

The following test contains the results of a file transfer to respective instance, where the transfer speed was measured in MB per second.
The application container clearly performs better in transferring files through network, by transferring with almost 150% the speed of the virtual machine.

5.3 Scalability experiment

In the scalability experiment the two instances, virtual machine and application container, were analyzed in terms of behaviour in heavy loads. Figure 20 shows the CPU usage of each instance at a sequential increasing number of connections, while figure 21 exposes the RAM usage for the same test.
Figure 20: CPU usage in % of instances while handling HTTP requests at increasing number of connections

RAM usage while handling HTTP requests at increasing number of connections
Figure 21: RAM usage in MB of instances while handling HTTP requests at increasing number of connections

The comparison shows that the container occupies less resources, for serving different amounts of clients, than the virtual machine. While the virtual machine at most uses up to 280 % of the CPU and 1035 MB of the main memory throughout the test, the container only has its peak at 235% CPU and 492 MB memory usage.

In the same test the average number of handled requests per second was measured at each interval. Figure 22 describes the behavior of the request handling speed of respective instance.

![Requests handled per second at increasing number of connections](image)

Figure 22: Requests handled per second at increasing number of connections

The virtual machine turns out to outperform the application container slightly in request handling speed, however at a quantity of clients above 9,000 the two instances performed almost equally. The application container is more stabilized in maintaining the request handling speed, while the virtual machine is decreasing at a faster rate.
6 Analysis

By analysing the produced outcome from section 5 Results and comparing it to the found results from section 4 Literature review, conclusions can be drawn in order to answer the research questions as formulated in section 2 Research questions.

6.1 User space isolation in comparison to resource allocation

6.1.1 Analysis of the results

Virtual machines are running complete operating systems with large amounts of active processes executing simultaneously, however the host operating system is only seeing one single process for the complete virtual machine. This creates almost complete abstraction from the host server in terms of process management, since the virtualized instance is unaware of other processes than it owns and the host server is not aware of the processes executed by the virtual machine. The file system of the virtualized environment in this study is isolated by creating a new partition for the guest operating system, which is visible in the host server but not mounted. In that way the file system of the guest instance is independent of its host, including abstraction from the underlying file system.

But the advanced level of user space isolation has its price. The separate operating system, including an own kernel [12], forces the virtual machine to require more physical resources, such as main memory and storage, from the host server in order to run as intended. The virtualized instance in this study uses approximately 577 MB of the main memory and 965 MB of the disk storage after it is created, however for the creation of a virtual machine in Openstack a volume of fixed size is required. The instance will always occupy storage memory according to the size of its volume, which is 20 GB in this study, independent of the amount of storage the virtual machine actually is using. Main memory is assigned to the instance by reserving a fixed amount of virtual memory in the underlying system. In that way the virtual machine will always think that the RAM is available, independent of the actual amount of available main memory in the host system. This also prevents the instance from using more RAM as it was assigned.

For application containers Docker provides a user space isolation almost equal to the virtual machines in Openstack, however there are still differences, since the containerized environment shares the kernel with the underlying operating system [13]. The active processes are isolated in a way so that the container only is aware of the ones it owns, just as the for the virtual machine. The host operating system, on the contrary, is aware of all processes that are executed inside containers, as they are clearly visible when listing all processes of the host system. A similar solution is found for the file system of the application container. The container is isolated from the underlying system so that it exclusively has access to the files and directories.
it owns or is sharing with the underlying system, however the containerized environment is deployed directly inside the main partition of the host operating system. This implies that it is possible to easily access the files of the container from the host server.

The advantage of sharing the kernel with the host operating system is the low resource consumption for deploying an application container. After creation the container in this study occupies approximately 5 MB of the main memory and 0.05 MB of disk storage, which is significantly less than the virtual machine. The physical resources of the host server are completely shared with the application container, where RAM and storage memory is allocated only according to the usage of the application container. This type of sharing leads to that a container is able to use as much of the resources as available on the host server, however a limitation can be set by the Docker engine in order to limit the amount of physical resources for an application container to be able to use.

Table 1 shows a summary of the discovered similarities and differences between virtual machines in Openstack and application containers in Docker, in terms of user space isolation and resource allocation.

| File system of host server accessible from within instance | Virtual machine in Openstack | No | Application container in Docker | No |
| File system of instance accessible from host server | No | Yes |
| Running processes of host server visible from within instance | No | No |
| Running processes of instance visible from host server | No | Yes |
| Type of resource allocation | Static | On-demand |
| Resource occupation at deployment | Higher | Lower |

Table 1: Table summarizing similarities and differences between virtual machines in Openstack and application containers in Docker

6.1.2 Comparison to the literature review

The outcome from section 5.1 User space isolation and resource allocation matches the results from section 4.1 User space isolation against resource allocation.
Virtual machines in Openstack behave equal to other virtualized environments, in terms of isolating the instance from the host server by implementing a separate kernel. The investigation of active processes in this study confirms the fact that the host operating system only is aware of the single process running the virtual machine. The found literature explains that a scheduler assures the virtual machine to obtain the required resources and share the access time equally with its host. By examining the file system of the virtualized environment, the theory about the separated volume used by virtual machines is confirmed.

As the literature review describes, the containerized environment is isolated in a way that disallows the container from accessing information it does not own, in terms of process management and file access. However in the literature it is delineated that the host operating system is able to share files with the application container. The found material does not describe the abstraction of the container as seen from its host server, but in the experiments it was found that the Docker environment only abstracts the host operating system from the containers, but not conversely.

The resource requirements found in the empirical study match the indications of the examined literature. Virtual machines require a large amount of physical resources for deployment because they run a separate operating system, while application containers share the operating system of their host and therefore occupy less main and storage memory for deployment. In the experiments it is also visualized that the virtualized environment reserves specific amounts of resources from the host server, while containerized environments implement on-demand occupation of physical resources.

6.1.3 Answering the first research question

The differences between virtual machines in Openstack and application containers in Docker, in aspects of user space isolation is, firstly, that the containerized environments are parts of the file system belonging to the host operating system, while the file systems of virtual machines are abstracted by assigning a separate partition. Secondly application containers only consist of a one way isolation, in terms of file system and executed processes, where the host server is fully aware of the actions performed by a container. Virtual machines, on the other hand, are isolated from both sides, where the host operating system is completely unaware of the executed tasks and contained files of the virtualized environment.

The resource allocation differs, firstly, in a way where virtual machines require a higher amount of physical resources than application containers in order to be deployed. Secondly, by creating the separated partition, virtual machines reserve a certain amount of disk storage at deployment, and at start up virtualized environments allocate a determined amount of virtual
main memory as well. Application containers, on the contrary, allocate resources on demand while running.

6.2 Performance with respect to request handling

6.2.1 Analysis of the results

In the comparison of CPU performance the virtual machine and application container almost performed equally, where the containerized environment only executed approximately 1,000 operations more than the virtual machine within 60 seconds. On an average the application container executes 10 matrix multiplications per second more than the virtualized environment.

The RAM bandwidth is almost equal between the two types of instances as well. However, this time the virtual machine is operating a tiny bit faster than the application container, by executing RAM access operations with an average of approximately 100 MB per second higher than the container.

Although the physical resources are virtualized in the virtual machine the performance is almost equally between the two types of instances. Surprisingly the main memory access time is even shorter for the virtualized environment. This may be due to the virtual memory allocation done by the virtual machine at startup, as found in section 5.1 User space isolation and resource allocation. The application container is allocating main memory on demand, which may be the cause for the low access speed at the start of the test.

The virtual machine was faster in responding to HTTP requests by maintaining an average request handling speed of 17.881 requests per second, while the application container had an average of 15.164 handled requests per second. Seemingly the increased performance in handling requests includes heightened resource usage. The virtualized environment used an average of 60 % CPU utilization above the application container, and about ten times as much main memory as the container.

Transferring files through network works better in application container than virtual machines. The container was sending the file with an average speed of 147 MB per second while the virtual machine sent approximately 112 MB per second.

According to the results of request handling and file transferring speed, with respect to CPU and RAM performance, the assumption can be highlighted that the virtual machine in combination with the Openstack network is faster in receiving and sending small data bundles, while the application container using the Docker network is better in handling larger data collections.
6.2.2 Comparison to the literature review

The produced results from section 5.2 Performance experiment partly match the conclusion from section 4.2 Performance evaluation.

In the found literature application containers are found to perform better than virtual machines in CPU operations. However the virtualized environments are expected to execute operations, using the CPU, almost equally fast as containers, similar to the outcome from the performance experiment in this study. Concluding that the comparison between virtual machines in Openstack and application container in Docker, in terms of CPU performance, is equal to the comparison found in the literature.

According to the literature review, containerized environments are faster in accessing memory compared to virtual machines. This was not the case in the empirical study, where the virtualized environment maintained a faster average RAM bandwidth, and the container even had some additional overhead at the first allocation of main memory.

In the literature study, application containers were faster in handling HTTP requests, just as generally receiving and responding TCP data through network. By performing the performance experiments the virtual machine was evaluated to actually handle HTTP faster than the application container. However the container was still faster in handling large amounts of data sent through the network, matching the found results from the literature.

6.2.3 Answering the second research question

The comparison between virtual machine in Openstack and application container in Docker, in terms of performance, can be described as a draw. The virtual machine was faster in accessing the main memory and handling HTTP requests through an Apache web server, while the application container was faster in executing CPU operations and transferring files through network.

6.3 Scalability in terms of request handling speed

6.3.1 Analysis of the results

The results of the scalability comparison between the virtual machine and application container show that the virtualized environment requires a higher quantity of physical resources for handling HTTP requests of increasing number of connections than the container, similar to the results of requests handled per second in the performance comparison. However in the scalability test the resource usage of the containerized environment was not as low as in the performance test, meaning that the amount of connections is more decisive for resource usage of the application container than the amount of requests. The virtual machine, on the contrary, used
approximately the same amount of resources in the scalability experiment as in the performance experiment.

As in the performance test of request handling, the virtual machine performed better in handling an increasing number of connections than the application container. On an average basis the virtualized environment handled approximately 2,000 requests per second more than the containerized environment while the number of connections was increasing. As the number of connections increased the request handling speed decreased in both instances. The request handling speed of the virtual machine decreases faster than of the container, but the speed itself remains above the speed of the application container throughout the test. At a number of connections higher than 8,000, the request handling speed is almost equally between the two types of instances.

6.3.2 Comparison to the literature review

The results from section 5.3 Scalability experiment for the most part do not match the found information from section 4.3 Scalability evaluation.

In the found literature application containers are explained to be easier to scaled up in case of heavy load to handle trailing requests. The results of the resource usage in section 5.1 User space isolation and resource allocation also visualize that containers in Docker require less resources than virtual machines in Openstack, meaning that a higher quantity of containers can be launched and used for handling requests. The results of section 5.3 Scalability experiment shows that virtual machines also require more resources for handling HTTP requests at increasing number of connections.

The literature describes that application containers are faster in responding to requests at increasing size of messages sent. The results of this study, on the contrary, shows that virtual machines, deployed in Openstack, are faster in responding to HTTP requests with increasing number of connections compared to application containers deployed through Docker.

6.3.3 Answering the third research question

Comparing the request handling speed, with respect to CPU and RAM usage, between a virtual machine in Openstack and application container in Docker shows that virtual machines are able to handle more HTTP requests per second than application containers at an increasing number of connections. However application containers, on the other hand, require less CPU and RAM utilization for handling HTTP requests.
Conclusion

In this study two types of environments, used as cloud application platforms, were compared. Openstack, running a virtual machine, and Docker, running an application container. The virtualized and containerized environments were compared in different aspects, such as user space isolation and resource allocation, several performance attributes and scalability, in terms of request handling speed at increasing load. The goal was to discover which type of instance is more suitable for different utilizations of cloud application platforms.

Virtual machines, deployed in Openstack, provide an isolated environment where the instance is abstracted from the host operating system from both sides. The virtual machine is unaware of operations done by its host and vice versa, while application containers, deployed through Docker, only are isolated from within the container, where the host, on the contrary, is aware of actions performed by the container.

Docker manages the physical resources of its application containers through on-demand allocation, allowing them to only occupy the amount of resources needed for executing the assigned tasks. Virtual machines in Openstack, on the other hand, reserve the physical resources, as indicated at the time of creation. Unfortunately these virtual machines require a higher amount of physical resources for deployment than application containers.

The performance, in terms of CPU and RAM efficiency, is almost equally, where application containers in Docker are a tiny bit faster in executing CPU operations and virtual machines in Openstack are slightly faster in accessing the main memory. HTTP requests are handled faster by the virtualized environment, as long as request sending rate is consistent, while the containerized environment is faster in transferring larger files through network.

In scalability the virtual machine is handling HTTP requests faster at an increasing number of connections, as well. However, the virtualization also requires a higher amount of physical resources for serving the requests.

The conclusion is that virtual machines, run by Openstack, are the recommended type of instance in cloud application platforms that either require a high level of security or higher web server performance, due to the total isolation and faster HTTP request handling speed. The virtualization is also more suitable in situations requiring to handle heavy loads by small amounts of instances. Application containers, deployed through Docker, however, are recommended in scenarios of many file transfers, but also in situations where a high amount of instances need to be run simultaneously, due to the low requirement of resources.
8 Future work

The results produced in this study could be improved by future work, testing each type of instance running an advanced web application, including request handling, database data selection and file transferring, all together. In that way the results would be more similar to nowadays scenarios.

In Openstack it is possible to configure the compute node to use the Docker engine for container deployment [23]. An interesting approach would be to compare Docker application containers deployed through Openstack against application container deployed directly through Docker. Such investigation would provide information about the Openstack platform compared to the Docker hypervisor, which could be interesting for companies providing a cloud application platform using Docker containers.

A scalability solution in Openstack [24], for running higher quantities of virtual machines in different data centers separated geographically, is to run multiple Openstack platforms on different data centers, while accessing all platforms through one parent Openstack API. That allows one Horizon dashboard to manage virtual machines located on different servers in separated server networks. This would be an interesting area to investigate for companies providing large scale platforms using multiple data centers.

This study investigates the performance and scalability of one single instance of each type. Cloud application platforms, however, are meant to run multiple applications in separate virtual machines or application containers. Therefore it would be a useful investigation to test the performance and scalability of two instances of each type running simultaneously. This would provide information about how the resources are distributed between instances of the same type, and how instances affect each other.
References


