A SECURE MODEL FOR EFFICIENT LIVE MIGRATION OF CONTAINERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ZEYNEP MAVUŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

A SECURE MODEL FOR EFFICIENT LIVE MIGRATION OF CONTAINERS

submitted by **ZEYNEP MAVUŞ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Pelin Angın
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Ali Hikmet Doğru
Computer Engineering, METU _____

Assist. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Assoc. Prof. Dr. Ahmet Burak Can
Computer Engineering, Hacettepe University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Zeynep Mavuş

Signature :

ABSTRACT

A SECURE MODEL FOR EFFICIENT LIVE MIGRATION OF CONTAINERS

Mavuş, Zeynep
M.S., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Pelin Angın

September 2019, 84 pages

Cloud services have become increasingly widespread in the past decade due to their ability to reduce the complexity and cost of managing computers and networks. Cloud applications are run in virtualized environments such as virtual machines and containers to be able to allocate resources in an inexpensive manner. Both of these approaches require effective resource utilization, for which an important enabling technology is live migration, which involves moving a service from one host to another with the minimum possible downtime. Live migration is also required for system maintenance, load balancing, and protecting services from attacks through moving target defense. While migrating a service, the system should not be vulnerable to attacks. In this work, we propose a secure model for efficient live migration of containers. Because the applications are isolated from each other while running in Docker containers, checkpointing was used to generate the required migration data. In our proposed model, we provide security of the migration data using secure authentication, and ensuring all connections between the nodes are protected to provide communication security, making the system protected against migration attacks. The

efficiency of the migration system designed based on the proposed model has been proven on stateless and stateful sample applications. Experiments with sample applications running on the Docker container platform demonstrate that the proposed approach achieves significantly better performance than its virtual machine live migration counterpart.

Keywords: Live Migration, Container, Virtual Machine, Docker, CRIU, Security

ÖZ

ETKİN KAPSAYICI CANLI TAŞIMASI İÇİN GÜVENLİ BİR MODEL

Mavuş, Zeynep
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Pelin Angın

Eylül 2019 , 84 sayfa

Bulut hizmetleri, bilgisayar ve bilgisayar ağlarının yönetme karmaşıklığını azaltma ve maliyetlerini düşürme yeteneklerinden dolayı son on yılda giderek yaygınlaşmıştır. Bulut uygulamaları, kaynakları masrafsız bir biçimde tahsis edilebilsin diye sanal makineler ve kapsayıcılar gibi sanallaştırılmış ortamlarda çalıştırılır. Bu yaklaşımların her ikisi de, mümkün olan en az aksama süresiyle bir hizmeti bir ana bilgisayardan diğerine taşımayı hedefleyen canlı taşıma teknolojisinin şartlarından, etkin kaynak kullanımını gerektirir. Canlı taşıma, aynı zamanda, sistem bakımı, yük dengelemesi ve sunucu hizmetlerinin hareketli hedef savunması yoluyla gelen saldırılara karşı korunması için de gereklidir. Bir servisi taşırken, sistem saldırılara açık olmamalıdır. Bu çalışmada, kapsayıcıların etkin bir şekilde canlı taşınması için güvenli bir model öneriyoruz. Docker kapsayıcılarında uygulamalar çalışırken birbirlerinden izole edildiğinden, gerekli geçiş verilerini oluşturmak için denetim noktası yöntemi kullanılmıştır. Önerilen modelimizde, güvenli kimlik doğrulaması kullanarak geçiş verilerinin güvenliğini sağlıyoruz ve düğümler arasındaki tüm bağlantıların iletişim güvenliğini sağlayarak sistemi taşıma saldırılarına karşı korumalı hale getiriyoruz. Önerilen modele dayanarak tasarlanan canlı taşıma sisteminin verimliliği, durumsuz

ve durumlu uygulama örnekleriyle kanıtlanmıştır. Docker kapsayıcı platformunda çalışan örnek uygulamalarla yapılan deneyler, önerilen yaklaşımın sanal makine canlı taşımasından önemli ölçüde daha iyi performans elde ettiğini göstermektedir.

Anahtar Kelimeler: Canlı Taşıma, Kapsayıcı, Sanal Makine, CRIU, Güvenlik

To my family...

ACKNOWLEDGEMENTS

First, I would like to thank to my supervisor Asst. Prof. Dr. Pelin Angın who guided me by motivating and encouraging with a great patience and understanding during my study. For their precious comments and questions, I also thank to my committee members.

I would also like to thank my managers and teammates from ASELSAN for their understanding and support during my academic studies. I would also like to express my special thanks to my friends for making them feel around me whenever I needed them, even if they were far away.

Last, I would like to thank my family: my parents Nurefşan and Türker Mavuş, for trusting me and supporting me for every choice I made throughout my life, my brother Ahmet Mavuş for giving me valuable advices and being a good role model to be followed for me.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGEMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xv

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xviii

CHAPTERS

1 INTRODUCTION 1

1.1 Motivation and Problem Definition 1

1.2 Live Migration Attacks 2

1.3 Live Migration Security Factors 3

1.3.1 Access Control 4

1.3.2 Authentication 4

1.3.3 Data Confidentiality 4

1.3.4 Communication Security 4

1.3.5 Data Integrity 4

1.3.6 Availability 5

1.4 Proposed Methods and Models 5
1.5 Contributions and Novelties 5
1.6 The Outline of the Thesis 6
2 BACKGROUND INFORMATION 7
2.1 What is a Container? 7
2.2 The Differences Between Containers and Virtual Machines 8
2.3 Docker Container Technology 9
2.3.1 Images 10
2.3.2 Containers 11
2.3.3 Services 11
2.4 Live Migration 11
2.5 Moving Target Defense 13
2.6 Stateless and Stateful Applications 14
2.6.1 Stateless Application 14
2.6.2 Stateful Application 15
2.7 SSH (Secure Shell Protocol) 15
2.7.1 Symmetric encryption 16
2.7.2 Asymmetric encryption 17
2.7.3 Hashing 17
2.8 Advanced Encryption Standard 18
2.9 Key Exchange Method 18
2.9.1 Diffie-Hellman Key Exchange Algorithm 19
2.9.2 Diffie-Hellman Usage in SSH 22

2.10 SFTP 23
2.11 Algorithms Used in SSH and SFTP 24
3 RELATED WORK 29
3.1 Secure Live Migration of Virtual Machines 29
3.1.1 Migration Network Isolation 29
3.1.2 Network Security Engine Hypervisor 30
3.1.3 Secure VM-vTPM Migration Protocol 30
3.1.4 Improved VM-vTPM Migration Protocol 31
3.1.5 Using IPSec Tunneling 31
3.1.6 Live Migration Defense Framework 32
3.1.7 Inter Cloud Virtual Machine Mobility 32
3.1.8 Trust Cloud Security Level 33
3.1.9 Secure VM Migration by Using RSA with SSL protocol 33
3.1.10 Trust Token based VM migration protocol 33
3.1.11 Runtime Monitors 34
3.1.12 Migration Using Memory Space Prediction 35
3.1.13 Encryption of Migration Data with AES Algorithm 36
3.2 Live Migration of Containers 37
3.2.1 ESCAPE Live Migration Model 38
3.3 Live Migration of Containers vs Virtual Machines 39
4 PROPOSED MODEL 41
4.1 Model Architecture 41
4.2 Checkpointing and Restoring 45

4.3 Model Execution Plan for Stateless Application 46
4.4 Model Execution Plan for Stateful Application 48
4.5 Security Evaluation 50
4.6 Attack Resilience 51
5 EXPERIMENTAL EVALUATION 53
5.0.1 Experiment Setup 53
5.0.1.1 Source and Destination Cloud Instance Setup 53
5.0.1.2 Docker and CRIU Configurations 54
5.0.1.3 Application Server Setup 55
5.0.1.4 Database Server Setup 56
5.0.1.5 Applications Used in Experiments 57
Clock Application 57
Installation 58
Face Recognition Application 58
Installation 58
5.0.1.6 SSH Connection and File Transfer Application 60
SSH Public Key Authentication: 60
5.0.2 Experiment Metrics 62
5.0.3 Experiment Results and Discussion 63
5.0.3.1 Multi User Test Scenario Results 68
5.0.3.2 Comparison with Layered Framework Experiment . . . 70
6 CONCLUSION AND FUTURE WORK 77
REFERENCES 79

LIST OF TABLES

TABLES

Table 2.1 Diffie-Hellman Mechanism [1] 22

Table 5.1 IP Addresses of Instances 54

Table 5.2 Stateless Clock Application Experiment Results 67

Table 5.3 Multi User Test Cases and Transfer Duration Results 69

Table 5.4 Layered Framework Experiment Setup Machines' Details 71

Table 5.5 Total Migration Duration and Transfer Data Size for LXC 72

Table 5.6 Total Migration Duration and Transfer Data Size for KVM 72

LIST OF FIGURES

FIGURES

Figure 2.1 Containers versus Virtual Machines 8

Figure 2.2 Docker Architecture 10

Figure 2.3 Overview of Live Migration 12

Figure 2.4 Pre-Copy Migration [2] 13

Figure 2.5 Post-Copy Migration [2] 13

Figure 2.6 Diffie-Hellman Algorithm Illustration 21

Figure 3.1 Migration Network Isolation [4] 30

Figure 3.2 Network Security Engine Hypervisor [4] 30

Figure 3.3 vTPM Migration Solution [4] 31

Figure 3.4 IPSec Tunneling [4] 32

Figure 3.5 Prediction Based Compression Architecture [5] 36

Figure 3.6 System Design [6] 37

Figure 3.7 Live VM Migration using AES encryption [6] 37

Figure 4.1 Model Architecture Overview 41

Figure 4.2 Model Architecture 43

Figure 4.3 Proposed Model Activity Diagram 44

Figure 4.4 CRIU Principle Diagram [3] 46
Figure 4.5 Clock Application Execution Plan 48
Figure 4.6 Face Recognition Application Execution Plan 50

Figure 5.1 Cloud Instance Details 53
Figure 5.2 Experimental Setting 54
Figure 5.3 Docker Configuration 55
Figure 5.4 Application Server Details 56
Figure 5.5 Database Server Details 56
Figure 5.6 SSL Connection Configuration 57
Figure 5.7 Containerizing Stages of an Application 59
Figure 5.8 Total Migration Time and Downtime 64
Figure 5.9 Upload Speed Effect on Downtime and File Transfer Duration . 65
Figure 5.10 Checkpoint and Resume Duration 66
Figure 5.11 Stateless Clock Application Experiment Results Graph 67
Figure 5.12 Multi-User Total Migration Time and Checkpoint File Sizes . . . 69
Figure 5.13 Multi-User Checkpoint and Resume Duration 70
Figure 5.14 LXC Results [7] 73
Figure 5.15 KVM Results [7] 74
Figure 5.16 Proposed Secure Model Results with Face Recognition Application 75

LIST OF ABBREVIATIONS

ABBREVIATIONS	
VM	Virtual Machine
AES	Advanced Encryption Standard
NIST	National Institute of Standards and Technology
SSH	Secure Shell
SFTP	SSH File Transfer Protocol
SSL	Secure Sockets Layer
CRIU	Checkpoint/Restore In User Space
RMM	Remote Monitoring and Management
RSA	Rivest Shamir Adelman
IDS	Intrusion Detection System
TCP	Transmission Control Protocol
LXC	Linux Container
KVM	Kernel-based Virtual Machine

CHAPTER 1

INTRODUCTION

Cloud Computing has recently become one of the most popular computing paradigms. This methodology is transforming IT services into remote computing services over the Internet in an on-demand manner. Additionally, these services can be configured according to the requirements of the customers. Because the cloud customers or users are not expected to pay for IT service infrastructure, buy the hardware and software resources (such as license), cloud computing is beneficial to the customers in many aspects like flexibility, portability and scalability.

In order to provide these benefits to the users, virtualization techniques are frequently utilized in cloud computing, as it needs to enable users to share the infrastructure, which makes the service availability less expensive. Without virtualization techniques, cloud providers would need to provide distinct resources to their customers, which is possible, but would result in high costs as a significant disadvantage.

In order to tackle the high service cost, virtualization technologies such as virtual machines and containers are used. In that case, the whole performance efficiency of the cloud has a strong relation with the performance efficiency of the VMs and containers. Other than the performance issue, security concern is also dependent to the security concern of the virtualized environment used in the cloud.

1.1 Motivation and Problem Definition

A basic cloud service can be defined as cloud provider's service present to users via internet access. The cloud applications are generally made resident in virtualized

environments such as virtual machines and containers. *Containers* are a new technology relative to virtual machines. There are key differences between two technologies: One of them is that while virtual machines imitate the operating system kernel, containers make use of the hardware and kernel of a single host operating system in a shared manner. Containers encapsulate applications with their required binaries in order to provide the application as a service. Therefore, containers have less virtualization cost and use less resources relative to VMs because of their lightweight nature. Both virtualization techniques could provide increase in efficiency in the utilization of the resources in the big data centers. This could be provided by the migration of the encapsulated service. Live migration has become an increasingly popular topic because of its contribution to the consolidation of services. There are many other reasons to migrate a service from a source host to a destination. These can be system maintenance (for a software or hardware update), load balancing, efficient resource utilization, protecting the service from an attack as a moving target defense method, etc. There are many studies in the literature on VM live migration as in [4], [5], [8]. In addition to these, migration management frameworks have also been developed for virtual machines like OpenStack to be able to apply load balancing between virtual machines. On the other side, migration management frameworks do not exist for containers because of the immaturity of the technology.

Although virtual machine migration has been extensively investigated and discussed in the cloud computing research area, container migration does not have the same set of investigations in the literature because it can be counted as a relatively new technology. Especially, existing solutions do not consider the security aspect of migration. In the cloud computing area, live migration is unprotected against many kinds of attacks such as network sniffing, man-in-the-middle, replay etc. An ideal live container migration system should take precautions for these.

1.2 Live Migration Attacks

Cloud services are publicly available to machines connected to the Internet, which makes cloud computing services an open target for online threats. Since live migration takes place between different machines in the cloud, also, the network that

connects these machines to each other should be resistant to outsiders' attacks. By using intrusion detection systems to catch suspicious behaviors on cloud machines, some solutions are proposed to tackle these problems. Live migration as a moving target defense strategy has been proposed as one of these solutions. Nevertheless, the live migration concept has still security issues which should be dealt with.

Live migration is susceptible to active and passive attacks. An active attack causes loss of data integrity. On the other hand, passive attacks cause loss of sensitive data confidentiality. Some of the most remarkable attacks can be listed as man-in-the-middle, denial of service, overflow and replay attacks [35].

- Man-in-the-Middle Attack: Attackers can eavesdrop on the data while migrating from source host to destination. This could cause loss of data integrity.
- Denial of Service (DoS) Attack: By using false resource advertisement, attackers can captivate more virtual machine towards themselves. They can migrate a virtual machine stealing the bandwidth resource by preventing required migrations. This can lead to serious problems in the cloud system, where migrations are started in an automatic manner.
- Overflow Attack: Stack overflow can be caused by attackers by creating congestion in the communication channel traffic, which can result in the memory corruption of the running process.
- Replay Attack: Attackers can re-transmit the previous replicates of memory pages to the destination host where the changed ones are required. This happens because of frequent dirty page occurrences. Attackers can also modify the order of memory pages sent to destination from source. This results in ordering problems in the destination host [35].

1.3 Live Migration Security Factors

A secure live migration technique should provide the security of the system from different perspectives. Each individual level of the architecture of the system should be taken into account separately. To be able to ensure the quality of proposed systems,

there are some security factors used as a checklist. The detailed factors making live migration secure are as follows:

1.3.1 Access Control

Unauthorized users are allowed to start, move and stop a machine if improper access control policies are defined. Therefore, if the appropriate control policies are defined, the VMs or containers and the host server becomes protected [36].

1.3.2 Authentication

Authentication is required between the source and the destination hosts. By this way whether the migration occurs between authorized source and target hosts can be checked [36].

1.3.3 Data Confidentiality

Data encryption is required while migrating data between source and destination hosts. If it is transferred as plain text, it makes the system open to man-in-the-middle attacks and sensitive data can get stolen in the migration period [36].

1.3.4 Communication Security

The data transmission channel should be protected and secure. In order to prevent the system from becoming open to active and passive attacks, a secure transmission channel should be defined on the migration path between source and destination hosts.

1.3.5 Data Integrity

Data integrity can also be destroyed due to some vulnerabilities in the migration module. In order to prevent that, methodologies for secure programming are required to

be used. Beside the migration data encryption, integrity can be protected by utilizing message authentication code (MAC), checksums and digital signatures.

1.3.6 Availability

If the system is not protected against denial-of-service attacks by allowing unauthorized users to initiate unnecessary outgoing migrations in an increasing manner, it makes the system resources unavailable to the authorized users. It is required to apply strong procedures for controlled access to protect availability.

1.4 Proposed Methods and Models

In our proposed model, we provide security of the migration data by using secure authentication and file transfer protocols. We propose a migration architecture which includes cloud host and target instances, an application server and a database server. All connections between the nodes are protected by secure protocols and encryption techniques specified in order to provide the communication security and make the system protected against migration attacks. The application server triggers the migration process between the nodes. The application server acts like a control manager over the migration system. The database server is a common pool in order to keep application results and serve them to the application server in order to return the result to the user by the application server. Performance efficiency is also measured and provided in order to show the overhead resulting from this encrypted channel approach.

1.5 Contributions and Novelties

Our contributions are as follows:

- A secure container live migration method is proposed.
- The efficiency of the implemented model has been proven on stateless and stateful sample applications.

- The impact of container encryption on the live migration system performance is provided.

1.6 The Outline of the Thesis

The rest of the thesis is organized as follows:

- Chapter 2 describes the tools and concepts vital to understanding the proposed live container migration model.
- Chapter 3 provides an overview of virtual machine and container live migration models in the literature.
- Chapter 4 describes the proposed model architecture.
- Chapter 5 describes the setup, analysis metrics, and results of the experiments with the proposed model.
- Chapter 6 gives a summary of the contributions and concludes with future work directions.

CHAPTER 2

BACKGROUND INFORMATION

This chapter provides the background information regarding the terminology, methodologies and tools used in this thesis. In addition, definitions and general overview of the live container migration concepts are explained in this chapter.

2.1 What is a Container?

A container is a software component to package an application with its code and all dependencies so it can be run as isolated from other processes. Containers of the shipping industry inspired the name of this technology. Instead of finding a unique packaging style for each product to be shipped, containers are used to make all the products fit in the ship by using standard sized containers, which are already fit in ship. If this idea is applied to the computer container technology, it means that containers provide a mechanism to package an application, which can be abstracted from its actual running environment. The containers include not only the software, but also its dependencies with their libraries, binaries and configuration files, and they get transported as a unit, by eliminating the differences between machines, operating systems and used hardware, which causes incompatibilities usually. This decoupling gives the ability to deploy container-based applications in an easy and consistent manner, regardless of the destination environment type [9].

2.2 The Differences Between Containers and Virtual Machines

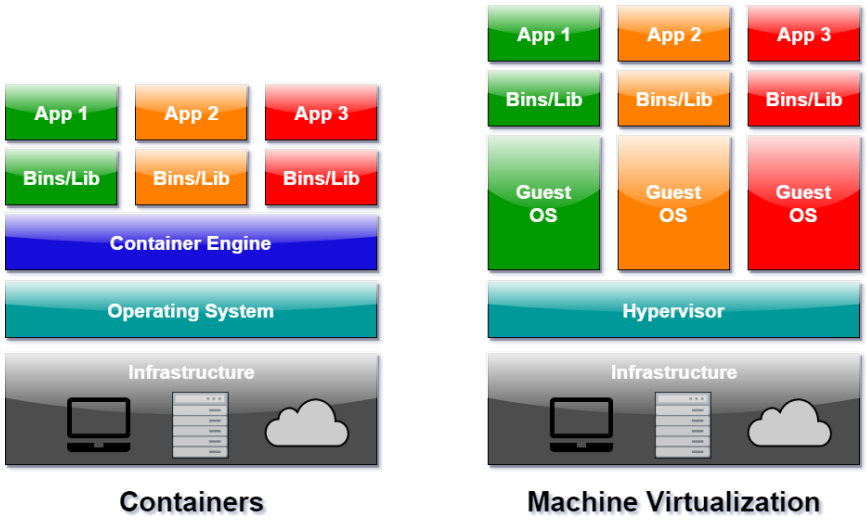

Figure 2.1: Containers versus Virtual Machines

Containers have become popular in an increasing manner and have become an alternative for classic VMs [10]. Virtual machines offer physical hardware abstraction. Therefore, they provide full isolation, meaning that they have their own applications running, users and networking, which are distinct from the host system and other guest systems and not visible to them or vice-versa. Multiple virtual machines are managed by hypervisors to work on a single machine. Each of the virtual machines contains a complete copy of an OS, the process, required binaries and libraries, which needs a memory space that is approximately tens of gigabytes. Therefore, booting can take a long time.

Containers offer application layer abstraction. Containerization means packaging source code and dependencies together. Containers offer virtualization technology, which makes them similar to virtual machines. However, they differ from each other in many aspects. Multiple containers can run on the same machine and the OS kernel is used commonly with other containers, each running as isolated processes in the user space. A container needs memory space that is approximately tens of megabytes, which means it requires less space than virtual machines. The goal of the containerization is to provide more computing power to the applications by sacrificing this

flexibility and therefore all containers use the same kernel [11].

Similar to virtual machines, containers run on a host machine, which can be virtual or bare metal. Containers can be configured to any specification required, providing isolated processes, users and networking, too.

The difference between virtual machines and containers is that containers contain only the application code and the required binaries and libraries that are required for the related container. Furthermore, this means that since a container does not need its own operating system, it uses only the resources required for the application upon container start [12] [13].

Containers also are stored as images. However, the main difference is that their images are much smaller and more portable than virtual machine images because they do not need a complete operating system installation.

Containerization technology, like OpenVZ [14], LXC [15] and Docker [16] become a different way of virtualization instead of classic VMs due to their lightweight structure [10].

2.3 Docker Container Technology

Docker [16] is a tool for software developers and system administrators to develop, transport and operate distributed applications using Docker Engine, which is a portable, lightweight tool to run and package the applications, and Docker Hub which is a cloud service for deployment and sharing applications and workflow automation [17].

Docker containers include software and all of its dependencies in a unit. This makes the developer sure that the application will run the same, independent of the host machine. Docker enables making applications independent of the infrastructure. Therefore, software can be delivered quickly [9].

Client-server architecture is the basis of the Docker structure. The Docker architecture consists mainly of three components: daemon, client registries and objects. Docker daemon is responsible for compiling, running, and distributing the contain-

ers. The client containers communicate with the daemon located on the host machine through a bridge [17].

Docker API requests are handled by the Docker daemon and it manages Docker objects, which are images, containers, networks and volumes.

Docker users first interact with Docker by means of docker client. The Docker client communicates with the Docker daemon. Docker images are stored in the Docker registry. Docker Hub is a registry that any user can access publicly.

When the *docker pull* or *docker run* commands are called, the necessary images are installed from the selected registry. When the *docker push* command is called, the built image is pushed to the selected registry.

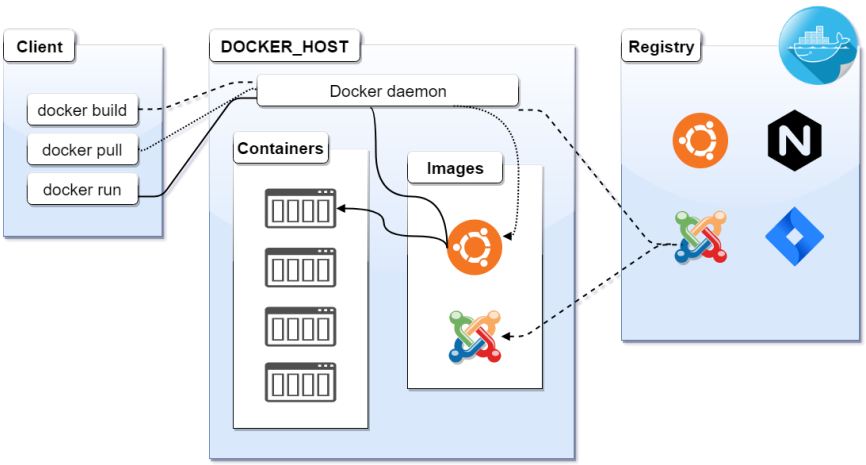

Figure 2.2: Docker Architecture

The main Docker objects can be listed as Images, Containers and Services.

2.3.1 Images

Docker containers can be created via an image, which is a read-only template. For instance, an image can include a CentOS operating system with an application server such as TomCat and some other web applications. Docker also provides means to create images and also to use previously created images from the registry.

In order to build an image, a Dockerfile is required. It contains the instructions re-

quired to create the image and execute it. Every instruction step in a Dockerfile forms a layer in the related image. When the Dockerfile is edited and the related image is recompiled, only the edited layers are recompiled. This makes images lightweight and efficient relative to other virtualization technologies [9] [17].

2.3.2 Containers

A container is an executable image instance. Docker API (or CLI) allows to create, start-stop or remove a container.

A container is isolated from others and its host machine. Isolation specifications of the container can also be manipulated.

2.3.3 Services

Services provide container scaling across Docker daemons. A service provides the opportunity to define the required status, i.e. the number of duplications of the service available at any given time [9].

2.4 Live Migration

The process of moving a running application instance or a virtual machine between different physical or cloud machines without making the client disconnected is defined as live migration. If a failure is encountered in the system, migration provides fault tolerance by moving the service to another machine. In addition to that, load balancing, reallocation of resources, hardware maintenance with acceptable downtime and high availability can be achieved using live migration. Memory, process and storage migration are the main parts of the VM and container live migration mechanisms.[8]

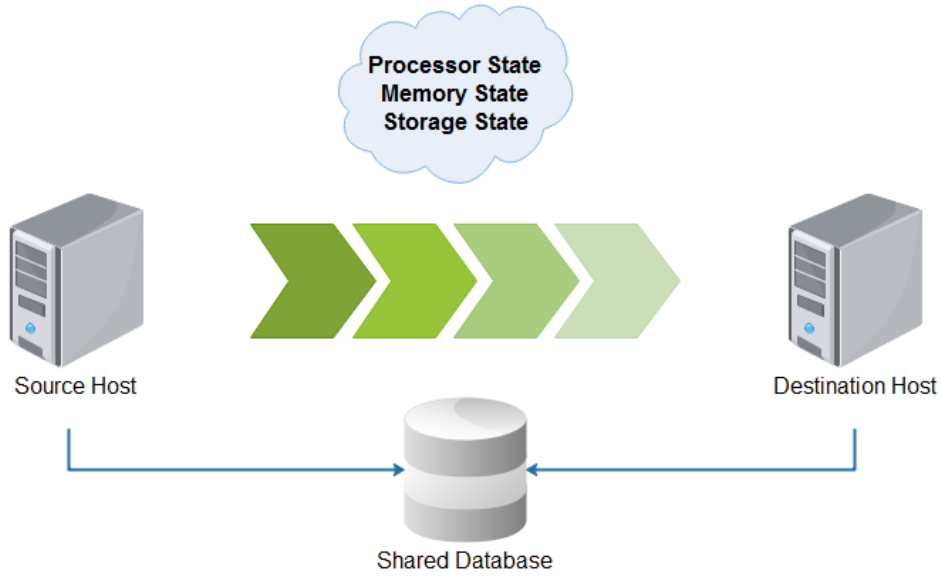

Figure 2.3: Overview of Live Migration

There are mainly two types of memory migration techniques. The first one is pre-copy memory migration. The second one is post-copy memory migration. The steps listed below is followed in pre-copy memory migration: [8]

- The source node continues to run while copying the memory pages from the source node to the destination node.
- Only the dirty pages that were changed during the previous transfer round are copied.
- The source node is stopped and the CPU state and the remaining dirty pages are transferred.
- The destination node is started.

The steps listed below are followed in post-copy memory migration: [8]

- The Source node is stopped. The CPU state is copied to the destination node.
- The destination node is resumed with no memory content.

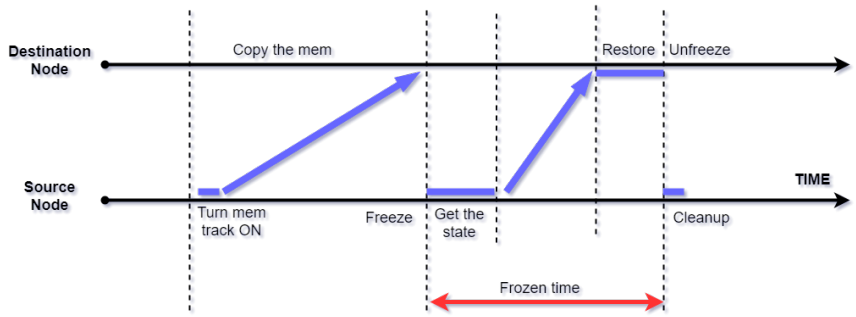

Figure 2.4: Pre-Copy Migration [2]

- If the destination node attempts to reach pages not transferred yet, the missing pages are requested over the network from the source node. The remaining states are copied from the source node to the destination node in parallel, meaning in the background.

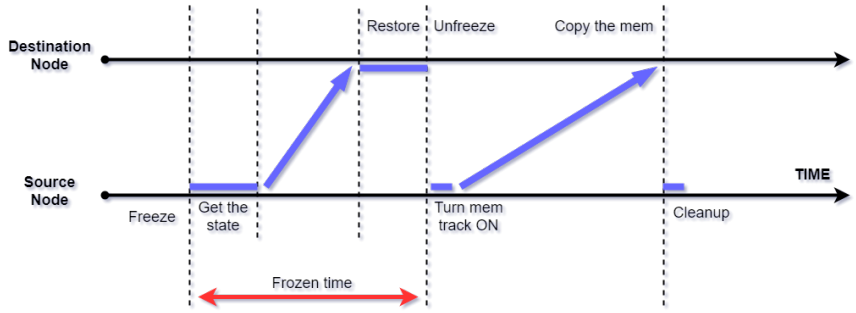

Figure 2.5: Post-Copy Migration [2]

The memory migration method consists of two methods (pre-copy and post-copy) as mentioned above. However, these methods have already been developed in Linux with a project named as CRIU. Instead of the familiar methods of transferring the entire memory, there are some efforts to improve performance by transferring the working set of the virtual machines and the status of CPU. [18]

2.5 Moving Target Defense

Moving Target Defense (MTD) is a game change strategy released in the National Cyber Leap Year Summit in 2009 [22]. The idea behind the MTD mechanism is

security through diversification. Its main concern is to alter the configurations of a target in a random and dynamic manner. These kinds of randomness in the system's configuration makes the complexity increase, which makes the attackers unable to exploit the system easily in this kind of a system, which has continuously altering configurations.

After Moving Target Defense was introduced at the National Cyber Leap Year Summit in 2009 [22] , it gained acceptance in many research areas. The shell game can be counted as a successful example of this concept. In the shell game, a person puts a marble sized object under one of the three shells, which are identical and looking downward. The person who placed the marble moves them in a consecutive and fast manner. When he stops, the player who finds the right shell containing the marble becomes the winner. Since the target (marble in our example) is randomly changing its place, it becomes hard to find for the player. Randomness and diversity are the main parts in this strategy. MTD became an alternative security solution in the last decades for researchers instead of investigating perfectly securing systems [22].

2.6 Stateless and Stateful Applications

The live migration concept can be categorized into stateless and stateful service migration. According to the category of the application, the migration procedure differs. In order to better understand the reason behind the difference of procedures applied when live migrating these services, requirements by definition of these application types should be classified accordingly. The migration system model should give reaction according to the classification of the related service.

2.6.1 Stateless Application

A stateless application can be defined as a computer program, which does not store any data computed in any of the sessions in order to be able to use in the following sessions. In other words, every session is conducted by assuming that it is the first execution of that program. The results of any of the executions of that program are not related to the results or data computed in the previous sessions.

Stateless application development is popular in the cloud computing environment, because it provides high scalability on the distributed system models in the cloud computing area. The main reason of that popularity comes from the ability of the easy redeployment in a failure situation occurence. This also makes live migration of the related stateless service easier, as any storage, program counter or state data related to that service is not needed to migrate between the instances. Only the redeployment of the service on the new instance is adequate to make the client keep on using the service.

2.6.2 Stateful Application

By contrast to stateless applications, stateful applications store data in order to continue execution. Application state has a crucial role in the migration process. Only making the related network traffic rerouted and making the application redeployed on the new machine as in the stateless applications are not enough. In addition to that, a checkpoint of the application, transfer of that checkpoint data and resuming the application on the new host are all required to make the client keep using the service without experiencing a service failure.

2.7 SSH (Secure Shell Protocol)

SSH (Secure Shell Protocol) enables establishing login connections and transferring files over the Internet or other insecure networks in a secure manner. In order to authenticate both sides of the connection, cryptography is used. All data transmitted is encrypted automatically and data integrity is provided at the same time [23]. This protocol is the most used way of administering remote servers securely. With the help of encryption techniques, it also provides a mechanism for sending commands between machines in addition to securely authenticated connection and file transfer. The three major components of SSH are as follows:

- The Transport Layer Security Protocol: This protocol provides server authentication, confidentiality, and integrity of SSH. It also has the capability of com-

pression if required.

- The User Authentication Protocol: This protocol provides the authentication of the client side user to the server side.

- The Connection Protocol: This protocol multiplexes a number of logical communication channels over a single underlying SSH connection [24].

SSH uses a number of authentication and encryption/decryption methods to provide secure connection between two remote devices. While conducting a transaction, the used methods can be categorized as symmetric/asymmetric encryption and hashing.

2.7.1 Symmetric encryption

Symmetric encryption is an encryption technique in which the same key can be used to both encrypt and decrypt the messages. In other words, anyone knowing the key can both encrypt and decrypt messages. This type of encryption is also called as shared secret encryption. There is generally only one key used for all functions.

SSH employs symmetric keys to encrypt the whole connection. Both of the client and server sides make contributions for the key decision and the decided secret key is not shared with parties other than the client and server. Key exchange algorithms are used for key creation. These algorithms enable both the server and client to have the same key by sharing specific public data and modifying them using specific secret data. The key built this way is based on sessions. When the session is started and connection is established, the remaining data must be encrypted by using this shared key. This is completed before client authentication.

The symmetric cipher systems used by SSH are the followings: 3DES, AES, Blowfish, CAST128 and Arcfour. Both of the client and server side can agree on a list of available ciphers they have.

2.7.2 Asymmetric encryption

Asymmetric encryption is an encryption technique in which two keys are required, namely the public key and private key. Public keys can be distributed publicly meaning that public keys can be shared by anyone without a restriction. On the other hand, private keys must only be known by the key owner. Public key should be related to its private key. However, the private key cannot be guessed by analyzing the public key.

The message encrypted with a receiver's public key is decrypted only by using the corresponding private key owned by the receiver. This mechanism provides the confidentiality. This encryption method is used in different stages of SSH. In the first step used in session encryption, the key exchange algorithm is utilized. By this algorithm, asymmetric encryption is used. Both of the server and client sides create their related key pairs and exchange their public keys to be able to create a key which is used in symmetric encryption, in this step. The asymmetric authentication used in SSH includes authentication based on key. In order to authenticate a client to a server, key pairs of SSH are used.

First, the client produces a key pair and then sends the public key to any remote server it tries to connect to. After that, this key is written in a file named as authorized keys in the /.ssh directory in the user's home folder on the remote server. After the symmetric encryption is setup to provide secure communication of the server and the client, the client must complete the authentication process. Then, the server uses this public key which is located in the file to encrypt a message to the client. If the message can be decrypted by the client, it is proven that the related private key belongs to the client. Then, the remote server can provide the environment for this client [25].

2.7.3 Hashing

Cryptographic hashing is also used in SSH. The main distinctive properties of cryptographic hash functions are that they are unique in practice and cannot be reversed and predicted. If we change even a small piece of data, the hash functions creates a completely different hash for that message. The given hash cannot be used to re-create the original message, but can say that the given message results in a specific hash with

the same hashing function. Data integrity is guaranteed with this hash mechanism. In SSH, hash based message authentication codes (HMAC) are used. They are used for assurance that the received message content is unchanged.

A message authentication code algorithm is chosen in symmetric encryption as well. Each message transferred after the encryption must include a message authentication code so that the receiver side can understand that the packet integrity is provided. By using the the message packet sequence number and the content of the actual message and symmetric shared secret, the message authentication code is computed [25].

2.8 Advanced Encryption Standard

Advanced Encryption Standard or AES is one of the most popular ways for encryption and decryption methodology for electronic and sensitive data approved for use by National Institute of Standards and Technology (NIST) [26].

Joan Daemen and Vincent Rijmen developed AES in 1998. AES is a symmetric-key block cipher. It is a bit block cipher and uses 128, 192 or 256-bit long keys [27]. Each data block has 128 bits to be encrypted or decrypted by the AES algorithm. Block cipher means that it divides data in blocks and makes each block combined with the key to obtain an encrypted version of the data. AES uses 10, 12, or 14 rounds to encrypt data. The key size, mentioned above as 128, 192, or 256 bits, is dependent on the number of rounds [28].

AES encryption is efficient and adaptable to most different platforms. In addition to that, AES has been verified for many security applications [27]. Although increasing the key length causes extra overhead in performance aspects, it provides a more secure way to encrypt/decrypt the data [29].

2.9 Key Exchange Method

Key exchange methods provide the encryption/decryption helper keys to be exchanged between hosts through use of a cryptographic algorithm.

If both sides need to exchange messages which are encrypted, each side must be capable of encrypting and decrypting to send and receive respectively. This capability mainly depends on the technique used for encryption. If two parties utilize a password or code, both will need to have a replica of that password or code. If they utilize a cipher, they will require the input keys for the cipher to gain the right output after the execution of the algorithm of the cipher. If the cipher algorithm is based on the symmetric key approach, both parties will need to have a replica of that key. If the cipher algorithm is based on an asymmetric key approach (with the public and private key properties), both parties will require the public key of the other party. The key exchange problem arises because of the need to exchange the keys to create a communication channel in a secure manner so that anybody else cannot have a replica or copy of those keys.

In order to provide data privacy while one party is sending the data to the other party, SFTP, HTTPS and FTPS file transfer protocols encrypt the data in the related file by using symmetric encryption. As mentioned above, the symmetric encryption has a rule that both parties who are in communication with each other must have a key as shared for encryption and decryption of the data in the files. But the main issue is caused by the difficulty of sharing a key between two parties. The parties in communication can be too far away from each other. Therefore, the key cannot be transmitted with the standard daily used methods by everyone. If it was so, it means that any user who accesses that key would have the capability to decrypt all the message traffic between the parties. The solution method must be user friendly, scalable and secure. In addition to that, it must be appropriate for the Internet environment channels which is mostly insecure and needs high speed. If not, it means that it cannot be usable for the areas which need sensitive, large sized transmissions between long distances conducted on a daily basis. The most popular method used to tackle the key exchange problem is the Diffie-Hellman key exchange algorithm.

2.9.1 Diffie-Hellman Key Exchange Algorithm

In 1976, Whitfield Diffie and Martin Hellman created an algorithm which is known with the name of the Diffie-Hellman (DH) Algorithm.

It is a widely available algorithm used in most of the secure connection protocols on the Internet. Diffie-Hellman is a method to exchange a shared key between two parties in a secure manner over a distrusted network. A shared key is crucial between the parties, who may not have ever established a connection previously, so that they can encrypt the messages sent/received in their communications. It is utilized by most of the protocols, such as Secure Shell (SSH), Secure Sockets Layer (SSL), and Internet Protocol Security (IPSec) [1].

Behind the scenes, the algorithm has basic mathematical concepts such as modular arithmetic operations and exponents algebra, which makes the algorithm relatively easy to understand. In order to analyze and explain the algorithm operations better, Alice and Bob, popular placeholder names in cryptology, will be used. The main aim of this algorithm is making Alice and Bob reach an agreement on a shared key so that anyone trying to overhear the communication between Alice and Bob cannot solve the shared secret. The shared key between Alice and Bob is utilized separately to compose keys to use in symmetric encryption algorithms, which will provide to encrypt data between the two parties. The term used as key does not appear in the network as different from shared secret and encryption key [1].

Diffie-Hellman process can be described easier by using a color blending example before diving into the details of the mathematical interpretation. In the beginning, we assume that Alice and Bob agree on an initial color by choosing a color inside a very crowded color collection. Both of the parties share this selected color with each other publicly. Secondly, Alice and Bob pick new different colors (secret colors). This picked color is related to the publicly selected color. Alice and Bob mix their selected secret colors with their public ones and after that they send the resulting colors to each other. Both parties picked different secret colors. Then it means that their color mixtures are different from each other. After receiving mixtures from each other, they mix this received blend with their secret colors. After this last mixing operation, they have the same color, which means the secret key that they will utilize to exchange data between the two parties.

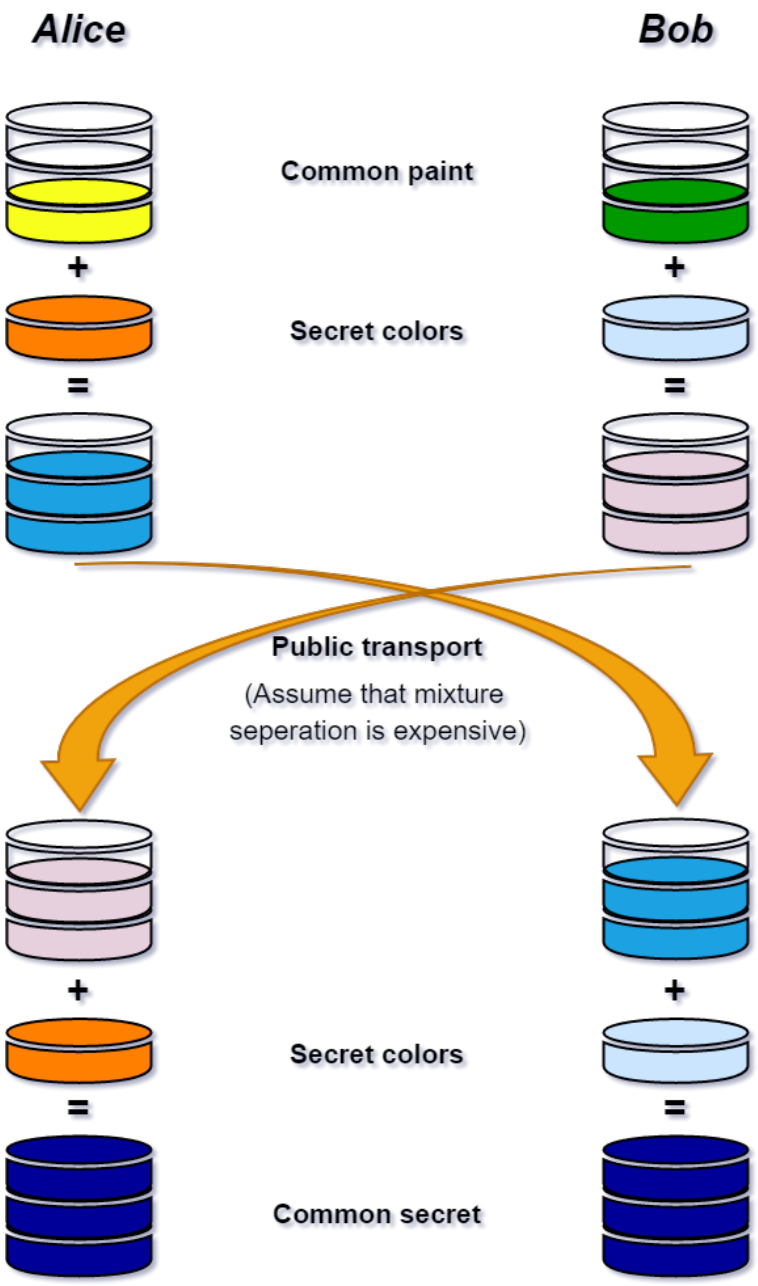

Figure 2.6: Diffie-Hellman Algorithm Illustration

In the following table, the mathematical operations used behind the Diffie-Hellman algorithm are described.

Alice and Bob agree on the values p and g	p is a large prime numeric value g is the base/generator
Alice selects a secret key a	Alice's secret key is a
Bob selects a secret key b	Bob's secret key is b
Alice finds her public key $x = g^a \mod p$	Alice's public key is x
Bob finds his public key $y = g^b \mod p$	Bob's public key is y
Alice and Bob exchange their public keys	Alice has p, g, a, x, y values Bob has p, g, b, x, y values
Alice calculates $k_a = y^a \mod p$	$k_a = (g^b \mod p)^a \mod p$ $k_a = (g^b)^a \mod p$ $k_a = g^{ba} \mod p$
Bob calculates $k_b = x^b \mod p$	$k_b = (g^a \mod p)^b \mod p$ $k_b = (g^a)^b \mod p$ $k_b = g^{ab} \mod p$
Alice's k_a is the same as Bob's k_b, or $k_a = k_b = k$	Alice and Bob both know the secret value k

Table 2.1: Diffie-Hellman Mechanism [1]

2.9.2 Diffie-Hellman Usage in SSH

Secure shell (SSH) is a popular network security protocol used for secure remote login on the Internet. It was considered as an alternative solution to the unsecured Telnet on the network and File Transfer Protocol (FTP) on the system. The main reason was neither Telnet nor FTP has data encryption capability, and instead just send data in plain text format. On the other hand, SSH has the capabilities for encryption, authentication and compression of messages transmitted. The kex (key exchange) protocol itself is a part of the SSH, especially stands for making both parties agree on the keys used in the SSH protocol. This constitutes the first step of the SSH mechanism. This occurs before the session keys are established. The protocol consists of three stages. The first stage is the greeting phase. In this step, both sides identify each other for

the first time. The supported algorithms list is included in here after the first greeting "Hello" message. This list includes the detailed provided Diffie-Hellman key groups. In the second stage, both of the two parties agree on a shared secret key with the help of the Diffie-Hellman key exchange algorithm. At the last stage, the application keys are formed by using the shared secret key, session digest and identifier. In order to provide host authentication, the key exchange is confirmed with the host key [30].

2.10 SFTP

SFTP (SSH File Transfer Protocol) means secure file transfer protocol. It executes on the SSH protocol. The whole security and authentication of SSH is provided by this protocol. It provides all the functionality offered by FTP and FTP/S, but in a more secure and reliable manner. It is also configurable easier than these protocols. Because of that, there is no reason to use the legacy protocols after that protocol.

Password sniffing and man-in-the-middle attacks can also be avoided with the help of the SFTP protocol. The data integrity is protected by using encryption/decryption techniques, cryptographic hash functions and the authentication of both the client and the server [31].

SSH cannot do file transfers. This means that there is nothing in the SSH protocol about file transfer: an SSH client cannot request anything from its peer to transmit or receive a file through that protocol. File transfer tools, scp1, scp2 and sftp, do not include the implementation of the SSH protocol themselves. Instead, they execute the SSH client as a subprocess, to make a connection to the remote machine and execute the remaining part of the file-transfer operation there. The SFTP protocol basically provides bidirectional file transfers by using a full-duplex byte stream connection [32].

SFTP advantages are given in the following items:

- It executes in a secure manner by using an SSH-protected channel for file transfer.
- Multiple commands for file copying and manipulation can be triggered with a

single sftp session. SCP opens a new session for each time it is triggered.

- It can be scripted by using the FTP command language.
- Software applications running an FTP client in the background can be replaced with SFTP, and by this way security of the file transfers of that application is provided [32].

2.11 Algorithms Used in SSH and SFTP

In this section, the basic workflow of the SSH connection and the algorithms used in SSH and SFTP are provided to better explain how it provides security. The secure shell protocol issues a client server model to make two hosts authenticated and provide the communication between them in an encrypted manner. The server side waits for connections by listening on a specified port. If the username and password are valid for connection, the server makes the connected client side authenticated and prepares the required environment to provide a secure channel between itself and the client. On the other hand, the host on the client side is in the charge of initiating the handshake with the host on the server side and supplies the username and password in order to authenticate. The establishment of a session with SSH is made in two main steps. The first step is to agree on the communication and make an establishment on encryption to provide security on the communication. The other step is to make the client authenticated and decide whether the privilege for the access from the client to the server should be given or not.

Once the connection request is received by the server, the supported protocol versions are returned as the response to the client. The connection process keeps on when any of the supported versions complies with the client. The server host also publishes its own public key. This key is utilized by the client in order to be sure about the host, regarding whether it is the expected one or not. Then, the server and the client agree upon a session key with the help of the variants of the Diffie-Hellman algorithm. This enables both sides to compose their private keys with the public key published by the other side to agree on the same session key. The whole session is encrypted with the help of this key. These mentioned keys are different from the keys which are used in

the client authentication to the intended server [25].

The flow of this process with the basic Diffie-Hellman is as given in the following:

- The server and client hosts choose a large prime number as the seed value for the algorithm.
- The server and client hosts choose an encryption algorithm. Generally AES encryption algorithm is used.
- Both the server and the client hosts select one more prime number and this is kept as a secret in themselves. This is the above mentioned private key.
- A public key is produced by using the selected encryption algorithm, the common prime number and the produced private key.
- This public key is sent to each other by the client and server hosts.
- The side who received the public key calculates the common secret key by using this public key sent by the other party, the commonly selected prime number and its own private key. By this way, the resulting secret key is the same although they are calculated separately on both sides.
- The communication is encrypted with this key after all.

The explained shared secret key process provides both sides to make contributions on the shared secret production stage. That is, the key is not manipulated by only one side of the communication. By this way, shared secret key information never travels on the communication channel. Because both sides have the same key in themselves in the end, it means that this key can be used for both encryption and decryption of the messages between them. After all these above mentioned process, the step for the user authentication starts.

This step contains the user authentication and making the decision on the user access to the server. For the user authentication, authentication by passwords are used. The server asks for the credentials of the user. This data is transmitted over the agreed connection in a secure manner. The other way used for the user authentication is applied with SSH key pairs. These key pairs are not symmetric. The data is encrypted

with the public key, but the data can only be deciphered by using the private key. It is allowed to share the public key because it is not possible to extract the private key by using the public key [25].

The authentication process using SSH key pairs proceeds as given in the following steps:

- An ID is transmitted by the client side to the server.
- The server searches the ID in the authorized_keys file.
- If it hits a public key with that ID in that file, the server side produces a number in a random manner and issues the public key in order to be able to encrypt the number.
- This generated encrypted message is transmitted to the client host.
- If the related private key is really known by the client, it means that the client has the capability to decrypt the message by using that. When it decrypts the message, it gets the number which is generated by the server randomly.
- The client host makes the composition of the number decrypted and the shared session key, which encrypts the communication. Then, it also computes an MD5 value of this composition.
- The computed MD5 value is transmitted to the server host by the client host.
- The server host also computes an MD5 value by using the number which is selected at the beginning randomly by itself and the shared session key. It then makes a comparison between its own computed value and the received MD5 value. If they are the same, it means that the private key is known by the client and it has the right to authenticate to the server [25].

AES, 3DES, Blowfish, Arcfour and CAST128 symmetric encryption algorithms are available and can be used in SSH by modifying configuration files of SSH. In addition to algorithm specification, we can also specify modes for some of the used algorithms. These modes include CBC (Cipher Block Chaining) and CTR (Counter) mode. The modes are named as block ciphers, which are methods to convert each

block of plain text to the cipher text or each block of cipher text to the plain text with the identical length. Blazhevski et al. [33] investigated block ciphers on AES algorithm. According to the results of the investigations, CTR mode is the most secure way of encryption among the other block ciphers. Another work on this topic shows that CBC mode has some vulnerabilities. However, they recommend the CTR mode instead of other modes to satisfy security of the used encryption method [34].

CHAPTER 3

RELATED WORK

This chapter includes a literature survey on live container and virtual machine migration models.

3.1 Secure Live Migration of Virtual Machines

VM platforms handle the live migration problem within the concept of high availability by applying checkpointing and record-and-replay strategies. While record-and-replay is the method that transfers every piece of information from primary to secondary, checkpointing is the strategy storing the state of the primary VM and transferring the state to the secondary node as a replication. In both methods, the synchronization of primary and secondary nodes is a challenging issue. While checkpointing handles this issue in a simple way, there are drawbacks such as the size of data and frequency of the process, affecting the performance of the migration [37]. Besides, the security of the data is an another issue that needs to be handled in the live migration concept. In order to prevent attacks, many solutions have been defined for virtual machines. In the following titles, the solutions in the literature are given.

3.1.1 Migration Network Isolation

In [38] the authors proposed a system with network isolation of the migration traffic. This approach is mainly based on setting the migration network apart from the others. With this method, no one else will be aware of the migration process. One of the examples of this method is Openstack [4].

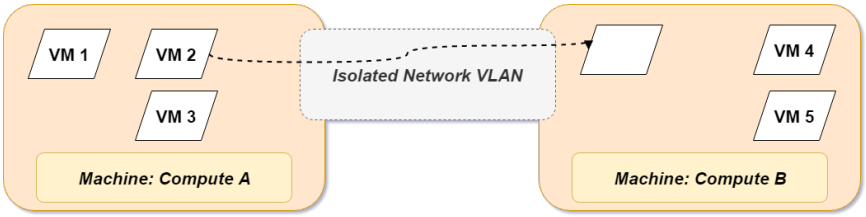

Figure 3.1: Migration Network Isolation [4]

3.1.2 Network Security Engine Hypervisor

In [39] the authors proposed a system by using Network Security Engine Hypervisor. By definition, hypervisor means something between middleware and OS. In other words, it can be named as a Meta OS in the virtual domain. The whole physical parts can be accessible by the hypervisor. Virtual machines can reach the resources over the hypervisor. It can function on operating system or on the hardware. Additional operations such as intrusion detection and prevention systems and firewall can be defined [4].

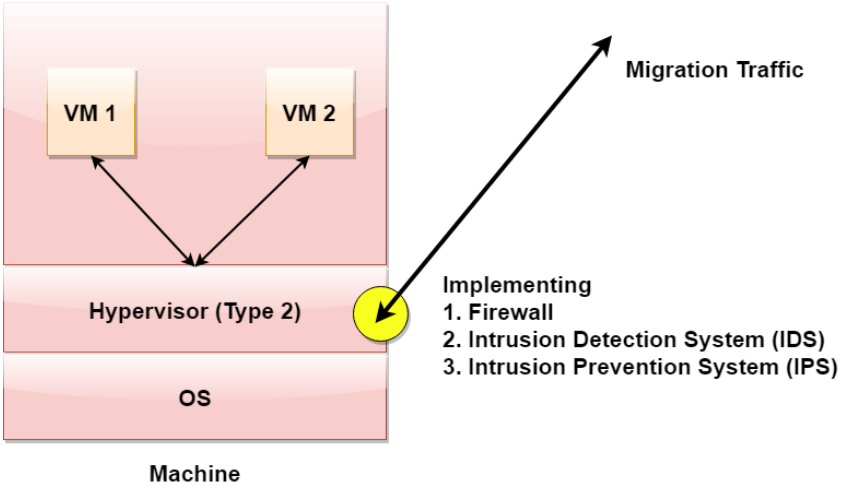

Figure 3.2: Network Security Engine Hypervisor [4]

3.1.3 Secure VM-vTPM Migration Protocol

In [40] the authors proposed a model by utilizing the secure VM-vTPM protocol. In case that Trusted Platform Module is included in the hardware, Virtual Trusted Plat-

form Module (vTPM) might be implemented. In order to use in the migration traffic, an authentication protocol providing security is presented by the vTPM module. By this authentication protocol, two machines, who are the "from" and "to" nodes of the migration process, authenticate, and confirm the integrity check of the connection between the source and destination before the migration process. After that preparation, the migration as encrypted is carried out in a secure manner [4].

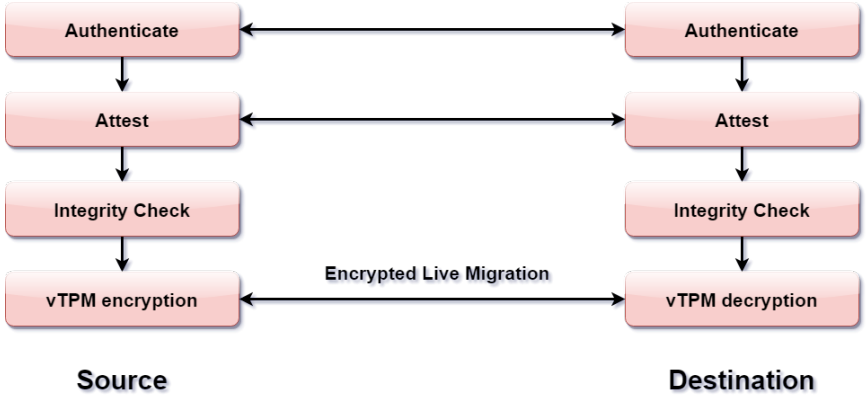

Figure 3.3: vTPM Migration Solution [4]

3.1.4 Improved VM-vTPM Migration Protocol

In [41] the authors proposed an improved version of the vTPM protocol for secure VM live migration. In this type of vTPM protocol, an additional layer is appended. This additional layer has the Diffie Hellman (DH) key exchange implementation.

3.1.5 Using IPSec Tunneling

In [42] the authors proposed a secure VM live migration model by utilizing the IPSec Tunneling method. IPSec Tunneling contains Internet Protocol Security (IPSec), which is a protocol in the network layer which provides IP traffic security. It offers that the traffic of migration should be encrypted as a standard internet protocol packet. Because of the long downtime duration, this solution makes the migration process decelerate [4].

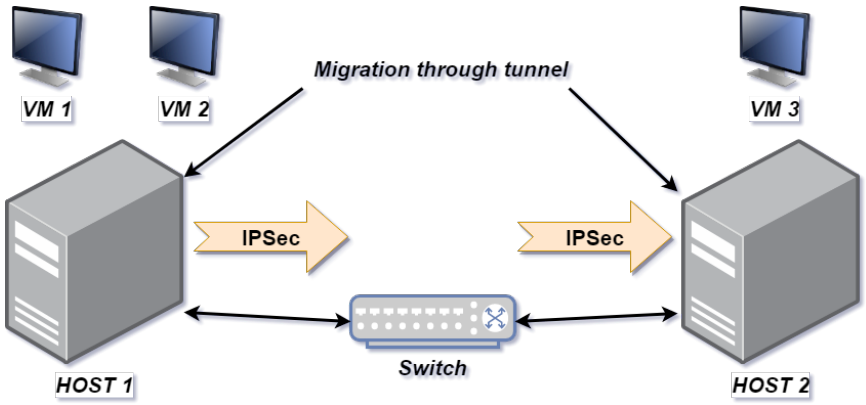

Figure 3.4: IPSec Tunneling [4]

3.1.6 Live Migration Defense Framework

In [43] the authors proposed a framework for VM live migration named as LMDF. Data centers of cloud providers are located in different places or even in different countries, which means that they can start the migration process from source users' VM to the destination users' VM without making the user feel the process. Live Migration Defense Framework was implemented, which will make the user notice the migration. This will enable the user to postpone the live migration to develop additional security assessments before the internal data migration. When the migration process is finished, the framework can check that the VM was migrated to the expected data center.

3.1.7 Inter Cloud Virtual Machine Mobility

In [45], the authors proposed a technology for VM live migration. The technology of Inter Cloud Virtual Machine Mobility contains inter cloud agents to make the channels secure channels between proxies. Inter Cloud proxies make the hosts which are included in this mobility keep their IP address private. In addition to that, unauthorized users will not be allowed to reach hosts which are utilized by this mobility. An SSH tunnel will be provided by the secure channel in between proxies. The aim of SSH tunneling is to camouflage the details of the "from" and "to" nodes of the migration. In total three channels are defined. They are secure inter cloud storage channel,

network and migration channel.

3.1.8 Trust Cloud Security Level

In [46] the authors proposed a method for secure VM live migration. The cloud domain is separated into different sections with security levels. For each reliable section and cloud, a security level is assigned. It will be the responsibility of the Reliable Migration Module (RMM) to manage the VM migration. Four main operations are performed by this module. The initial operation is central security management of resources that will also manage scheduling for all security levels. In the second operation of RMM, integrity check and encryption are provided. The third operation is component security management, which will separate nodes in different areas. The fourth operation is migration waiting queue arrangement which plans the VMs' migration petitions.

3.1.9 Secure VM Migration by Using RSA with SSL protocol

In [47] the authors proposed a new way to secure VM live migration by utilizing RSA. RSA is a public key cryptographic system to provide data transmission security. It has been released as a method to make live migration secure by utilizing RSA encryption with the SSL protocol. They consider several stages to migrate a VM using RSA with SSL. In order to achieve this aim, it computes the physical host load. Then, by using pre or post copy they migrate memory from source to destination. In the end, RSA is used to encrypt VM content and authentication.

3.1.10 Trust Token based VM migration protocol

In [48], the authors proposed a VM migration model by using a token methodology. This methodology uses TPM and offers a platform trust credential (Trust Token), which each cloud implements to identify the Trust Assurance Level (TAL) of that cloud. Initially, the model performs policy setting, where the cloud users establish a migration policy for the VM migration management. The second step contains mi-

gration policy implementation. When the users require to migrate from one source machine to another reliable destination machine, the cloud service provider implements the policy. Third, migration is observed by cloud users. The cloud users can control whether their specified policy has been obeyed or not by the cloud server provider. If the Trust Assurance Level in the Trust Token of the destination host matches the Trust Assurance Level specified in the migration policy of user, then the migration will be applied.

3.1.11 Runtime Monitors

In [4] the authors proposed a system model for VM live migration by using runtime monitors. In this architecture, including monitors is suggested for the whole migration event in the live migration procedure. Therefore, for each hypervisor virtual machine, an independent monitor agent will be attached. The main task of this agent is to watch different processes and detect any abnormal event. The structure consists of three main elements: Control Manager, Monitor Agents and Database Module.

Control Manager is the main part of the architecture. It gathers all migration requests from the different hypervisors. Additionally, Control Manager assigns tasks to agents to watch the whole migration and owns the permission to pause or end the migration. Each migration begins with the permission of this manager.

Monitor Agents watch the migration process and inform any abnormal activity to the Control Manager. If any suspicious activity is detected, the Control Manager will decide what to do after analyzing the situation. For instance, it can decide to terminate the process accordingly. There are two kinds of monitor agents.

- Agent dedicated to VM: An agent including IDS that observes the whole migration process of a specified migrated virtual machine.
- Agent dedicated to Attack: A seperate agent that is allocated to monitor the network for particular kinds of threats on the live migration process. For example, DoS attack dedicated agent monitors for this type of threat.

A database module is involved in this architecture. All data about all hypervisors

and guest virtual machines are kept in the database. A newly established hypervisor has to have an entry containing all related data in this database such as MAC and IP addresses. This new record can only be added by the admin privileged users.

In the migration setup period and before establishing a TCP connection between the source and the destination, the hypervisor of the source machine requests migration from the control manager. The request contains the data about source and destination hosts in order to be utilized in the authentication phase. The authentication phase consists of these stages: authentication of both hosts, computing the load on the destination host and generating a new entry for the new virtual machine data to be entered in the database.

When the authentication is achieved, an encrypted TCP channel will be setup between the source and the destination hosts. An independent agent will be assigned to watch the migration process. The agent knows the whole needed data to watch the migration process together with a unique time stamp nonce.

Many agents will be watching the whole network for particular threats. An allocated agent is assigned to watch every migrated virtual machine and a dedicated agent will be assigned to observe each migrated VM action. [4]

3.1.12 Migration Using Memory Space Prediction

In [5] the authors proposed a system model for VM migration by utilizing memory space prediction. In order to decrease the data loss caused by the attacks on memory space and to make the migration of data in a secure manner, Virtual Machine Migration using Memory Space Prediction with compression method is defined and developed. The prediction based compression algorithm is utilized to send compressed data. Additionally, it reduces the migration time and down time. The hash algorithm included increases the migration security. [5]

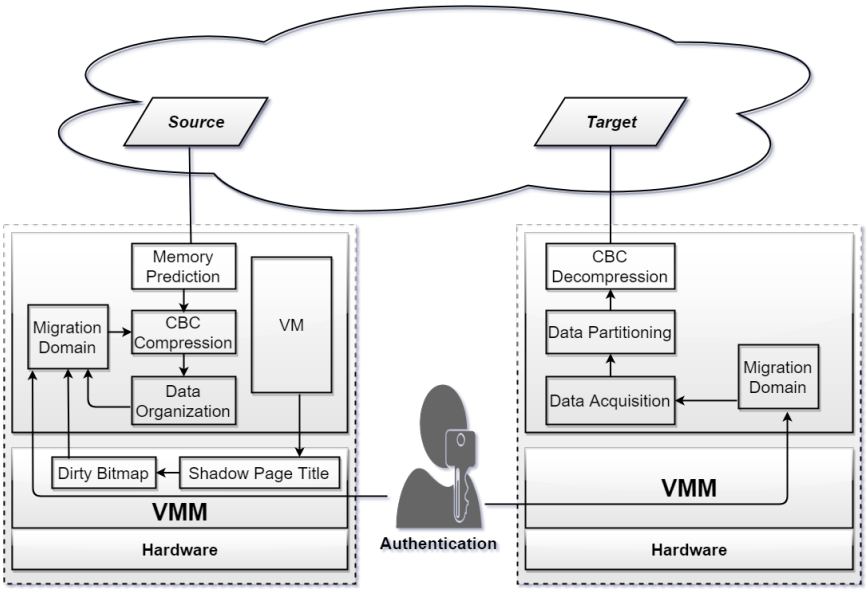

Figure 3.5: Prediction Based Compression Architecture [5]

3.1.13 Encryption of Migration Data with AES Algorithm

In [6] the authors proposed a system model by using AES encryption during migration. The execution steps of the system are as given in the following figure. Firstly, according to their requirements, resources are distributed to virtual machines. An under or overload is being checked continuously by using a cloud monitoring system. In case of an underload or overload, the K-means clustering algorithm is executed. When it finishes execution, VM migration is started. In order to provide security, AES encryption algorithm is used. As soon as the migration ends, the migrated data is decrypted and the resource allocation is applied. This process continues in a cyclic manner.

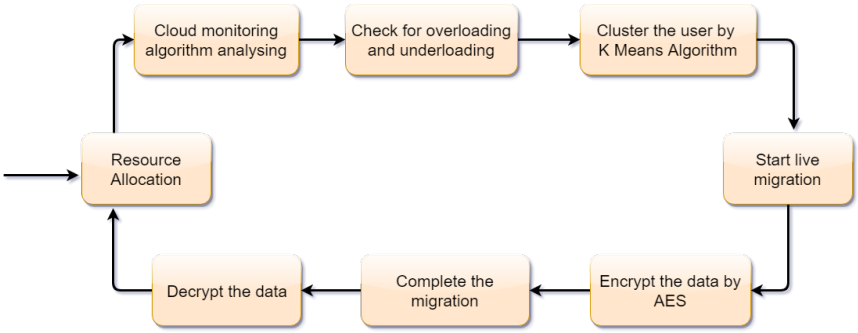

Figure 3.6: System Design [6]

The data which is loaded to the cloud might be compressed and changed. AES guarantees data confidentiality. To be able to solve these problems, which might occur in the live migration period, AES encryption is applied. The migration duration might be seriously affected from the encryption technique used. For the encryption, AES costs less than other techniques. Because of that, it is more convenient to use in migration. In addition to that, it is the least costly method and the application downtime is mostly decreased by that way. The overhead of encryption is also related to the respective speed of source and destination hosts [6].

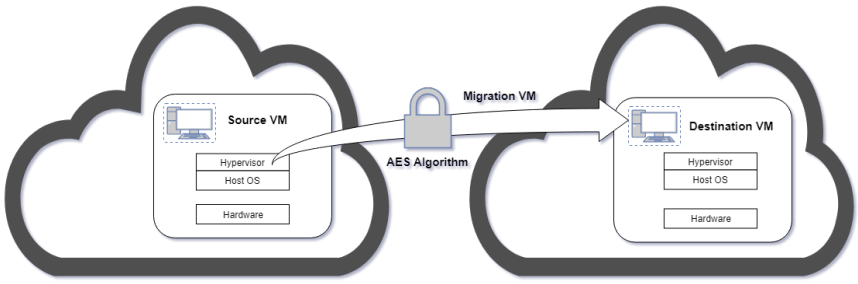

Figure 3.7: Live VM Migration using AES encryption [6]

3.2 Live Migration of Containers

Live migration process of containers heavily leans on checkpoint/resume strategies by benefiting the small sized nature of containers compared to VMs [49]. In order to overcome the attacks on containers, some solutions have been proposed. The solutions in the literature are given in the following.

3.2.1 ESCAPE Live Migration Model

In [50] the authors proposed the ESCAPE model for live migration of containers. The ESCAPE model is inspired from the survival technique in nature. This approach is based on a moving target defense mechanism by modeling the interaction between the attackers and their preferred victim hosts as a predator looking for a prey game. ESCAPE uses runtime container (prey) live migration to overcome attacks (predator). The whole process is driven by a behavior monitoring system, which is host-based and observes containers for detecting intrusions/attacks.

In order to enable checkpointing a running application, the model employs Docker (experimental version) that can run with the CRIU checkpoint tool to pause the running container and its attached applications taking a live memory content snapshot and any used documents/files. The images which are dumped are saved in persistent storage. The initial step is setting up the configuration of the container in order to host the application. To be able to monitor containers, the model employs an intrusion detection system. The system is especially designed to observe the Linux container behaviors. Based on intrusion detection the migration process is initiated.

In ESCAPE model, utilizing a virtual network interface with a specifically dedicated IP address for each container is preferred to achieve runtime migration. The direct access right to a dedicated network interface are given to applications in the container. ESCAPE deals with the runtime mapping by local or network wide mapping of interfaces and IPs.

ESCAPE initiates the migration process by choosing an appropriate destination host. The implementation currently performs the prey versus predator model to decide the next destination host to migrate the container. ESCAPE can also be capable of choosing the next destination with a distance logically far away to escape from attackers. By definition, the logical distance depends on a diversity scale. If the destination has more different settings on account of network, configurations and data center relation, it means that it is a good candidate to become a new destination for ESCAPE. By this way, the model intends to select a destination host to make the application of the same failure cause more complexity. After selecting a suitable destination, ES-

CAPE mounts the shared storage, which keeps the running container and applies the network settings to provide network relocation. It insulates the destination host from the active network by making its interfaces disabled and then begins the container process. When the container is started, ESCAPE changes the ARP table to redirect the network traffic from the source to the destination host and stops the source host. The migration process begins with checkpointing the container and terminating the process on the source host, applies an ARP update in order to alter the MAC/IP assignment of the previous server network interface to map the new one and resume the container and all related applications on the destination.

ESCAPE is led by an intrusion detection system monitoring container behavior for possible attacks. According to these attack indications, ESCAPE model conducts a risk assessment, which determines whether to migrate and where to migrate the running container to keep the attacker victim sufficiently far from the attacker [50].

3.3 Live Migration of Containers vs Virtual Machines

In [7] the authors proposed a layered framework for live migration of applications encapsulated either in containers or virtual machines. Experiments have been conducted to make a comparison based on the results obtained when working with VMs and containers.

The framework aims at achieving the good performance based on the need of frequent migration in the mobile edge cloud architecture. Mobile edge cloud is a network architecture, which provides cloud services at the cellular network edge.

When users travel around, it is required to migrate their applications as the cloud service in order to take advantage of the mobile edge cloud architecture. The proposed layered framework is compatible with the applications running in VMs and containers. The authors have separated the container state into different kinds of layers. These are base layer, instance layer and application layer. Base layer has the operating system and kernel with no application. Instance layer is for holding applications and their states while running. Actually, application layer is a division of instance layer. It only holds the application copy when the application is idle. The state when

the application is running is only saved in the instance layer. By this subdivision, application is copied when the service is running in the instance layer. The base layer is required to be available on each mobile edge cloud. Only the instance layer is migrated to the destination host. This layering technique provides the improvement from the performance aspect and provides less downtime. If we go into the details of the proposed mechanism, when we need to migrate a service application, we pause it first and migrate just the instance layer by making the assumption that the base layer is existent in all the mobile edge clouds. The migrated service can be restored by using the instance and the base layer together. Without the need to migrate the base layer in all the migration processes, it is intended to reduce the size of the transfer data. This is the two layer technique basically. The improved version of this technique is with three layers, which includes the application layer. This separates the application from the instance layer into an additional application layer which includes the idle version of the application and the data which is specific to the application. With the inserted application layer, the instance layer is only required to store the application in-memory state. When the migration process is started, the application layer is first migrated while the service is being executed. After that, the service is paused and the instance layer is sent to the target. By using all layers after the migration in the target node, the service can be resumed. An iterative file synchronization is used to send just the different parts between the application and instance layer.

In the research made in [7] the difference between VM and container migration is analyzed. Experiment results show that containers (LXC is used for the experiment) have more advantage over VMs (KVM is used for the experiment) for the total migration time, application downtime and the data transferred from the source node to the destination while migrating. The main reason for that is explained as that containers are lightweight relative to VMs and the container memory content is mostly related to the application running inside the container. However, for the VMs that is different, i.e. the VM memory content is related to a lot of other processes including background processes, which may be mostly unrelated to the migrated service.

CHAPTER 4

PROPOSED MODEL

This section describes the proposed model for secure live migration of containers.

4.1 Model Architecture

This section describes the proposed model architecture. The model aims to migrate a service from one node to another one without disconnecting clients from the application, with an acceptable downtime. In our proposed model, there are five main components. Two of them are the source and destination instances to play the roles of the migration source and destination sides. The live migration process could be operated between any nodes determined by any of the load-balancing modules, the attack detection module or another mechanism belonging to the cloud environment. The remaining main components are an application server, a database server and the client side. As shown in the 4.1, they have secure connections between them.

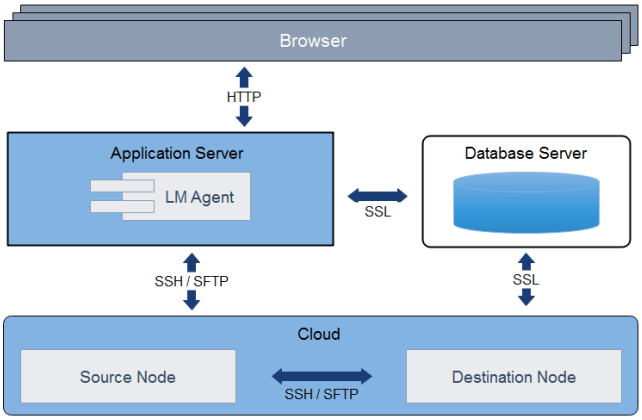

Figure 4.1: Model Architecture Overview

The application server behaves as the admin of our model that initiates the migration. It connects to the instances by SSH and issues commands over that SSH channel. Over that channel, the application server has the right to start, stop, kill, and remove the Docker containers. In addition to that, it has the right to completely shut down the instance machines. Because of storing authentication information in the application server, any migration process cannot be initiated if the application server does not take an action.

The application server creates the related SSH channels between the instances that would be assigned as source and destination nodes among all nodes on the cloud network. Also, the application server establishes secure channels between those nodes and itself in this model. In order to achieve that, it creates an SSH channel between itself and the source node first. Then, it commands the source node to run the application on Docker, and start migration if requested. That is, it commands the source node to send related checkpoint files to the destination node by using the SSH channel created by the parameters provided by the application server. Parameters are also provided by using the SSH channel, which means that an SSH command is sent to the source node to connect it to the other instance by using the SSH command parameters.

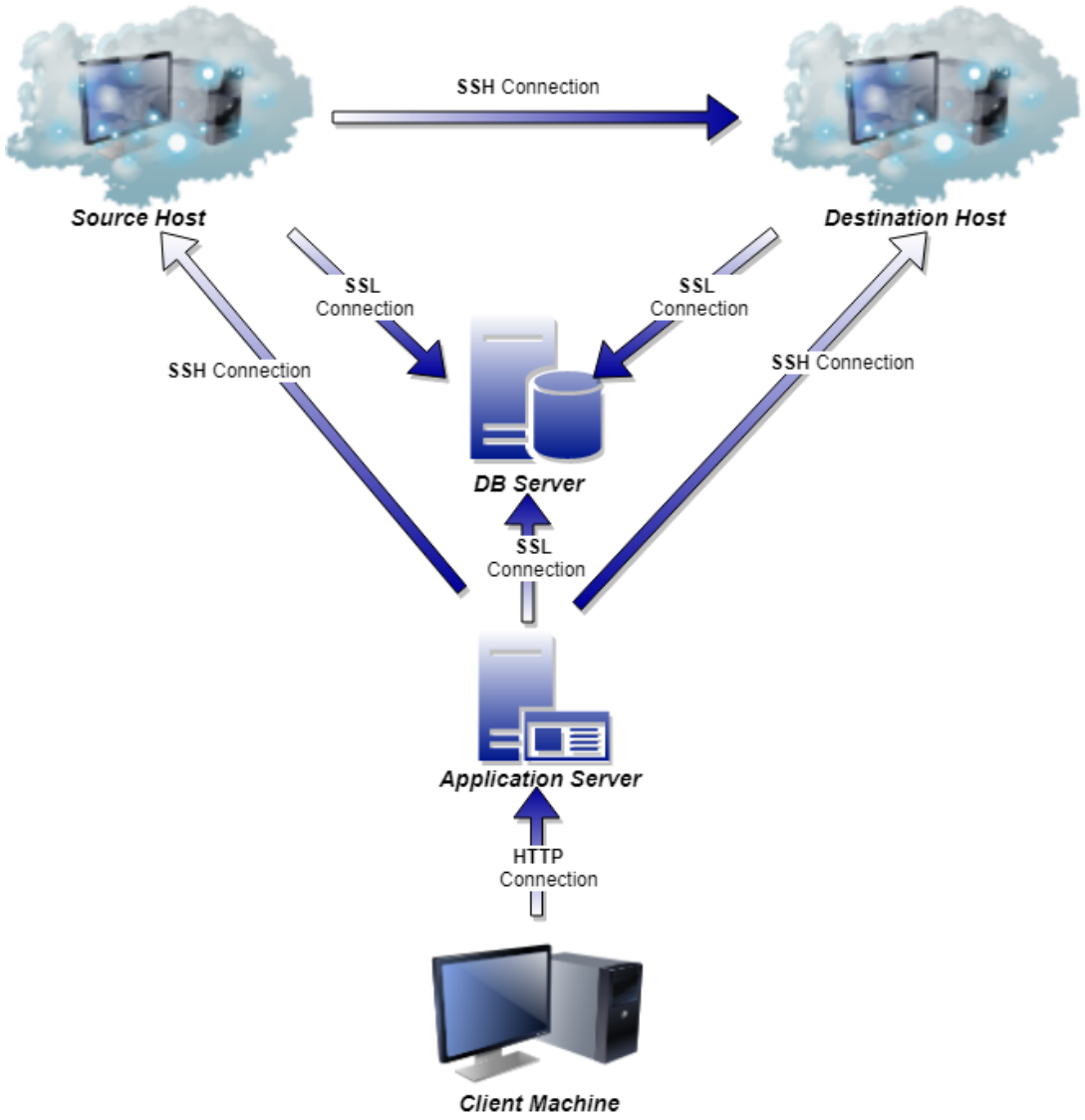

Figure 4.2: Model Architecture

An SSL connection is used for connecting to the DB server. The SSL connection can be configured to use MD5 encryption between the DB Server and the other machines. It also can be setup to allow the connections from the specified IP addresses. By configuring SSL in this way, we make the DB server only allow connections requested by the application server and cloud instances. No other machine with a different IP address is allowed by the database server.

The migration process operates different flows for stateless and stateful applications. While there are common steps in both flows, such as security check and controlling

life-cycle of containers, transferring data between nodes is the separating point among these application types. Migration operations for stateless and stateful applications is visualized in Figure 4.3. Details of this process are explained in the following sections.

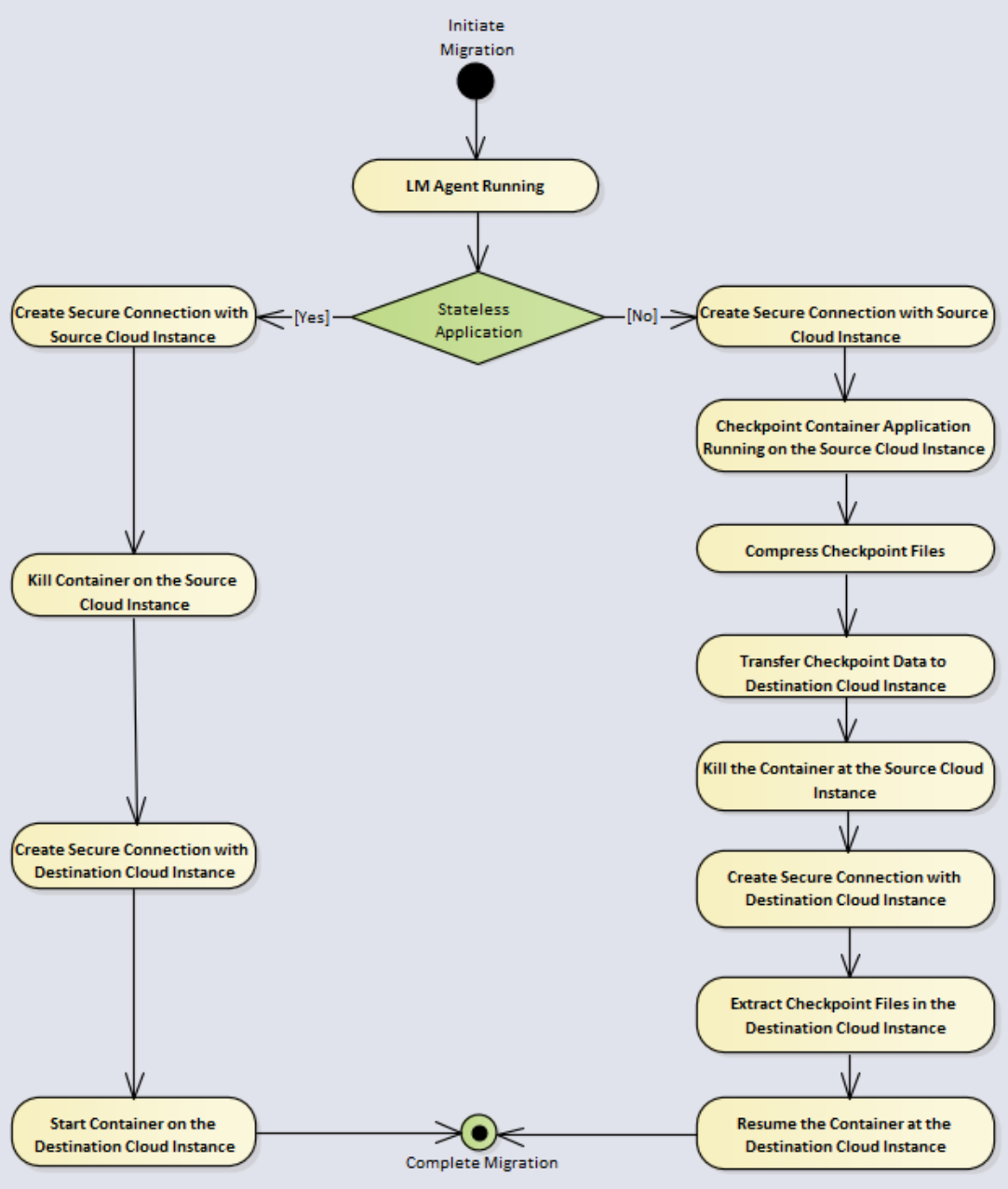

Figure 4.3: Proposed Model Activity Diagram

4.2 Checkpointing and Restoring

For the purpose of checkpointing/restoring, the CRIU tool is used. CRIU (Checkpoint/Restore In User Space) is a software tool for the Linux OS. Using this tool, a running application can be frozen and it can be check-pointed as a set of disk files. The files can be utilized to resume the application and run it starting from the state at the time of the freeze. Application live migration becomes possible with this feature. CRIU is supported as integrated into Docker, OpenVZ, LXC/LXD [19].

The main property of the CRIU tool is that it is basically developed in the user space, instead of the kernel space. This feature makes this tool provide live container migration by letting the users check-point and restore the currently running application instances. The process migration performed by the CRIU tool could be desribed mainly in three phases, which are checkpointing, page server activities and restoring.

CRIU provides the ability for check-pointing a running process as a set of files such as page maps, files descriptors and sockets opened. In other words, CRIU searches the process' tree to collect sufficient information about the related process for the resurrection [20]. In detail, at the beginning of checkpointing, the dumper process runs through process directories under /proc and establish a process tree structure by gathering the necessary information of related processes. Then, the parasite code is added to the relevant place of the task in order to run CRIU's subroutines inside the address space of the related process. The parasite code is connected to CRIU and takes commands from CRIU. After the dumping process, the parasite code is extracted from the task to returns to the original code. CRIU releases the process and gives the control to the operating system fully. In the end, CRIU evaluates the entire gathered data and records these information to dump files.

At restoring stage, CRIU reads the image files and resolves which resources are shared between processes. Then, by calling the operating system function fork(), CRIU creates processes on the destination node. After that, CRIU arranges neccessary settings for files, namespaces, maps, private memory areas, sockets and ownership. Finally, memory mappings to the exact location, timers, credentials and threads are restored to fulfill the resurrection of processes on the destination side [3].

CRIU is required only on containers having statefull applications, it is not preferred for use on stateless applications because the memory content and the states of the execution are not significant for restoration of the container. CRIU instantly stops the running container process and checkpoints it to a set of image files, which is needed to restore the container to the state in which it was stopped. In other words, CRIU is basically used to dump the container memory into a persistent collection of files, which makes the transfer and recover operations simpler [21]. The following figure is for showing the sequences for live migration of a container in CRIU.

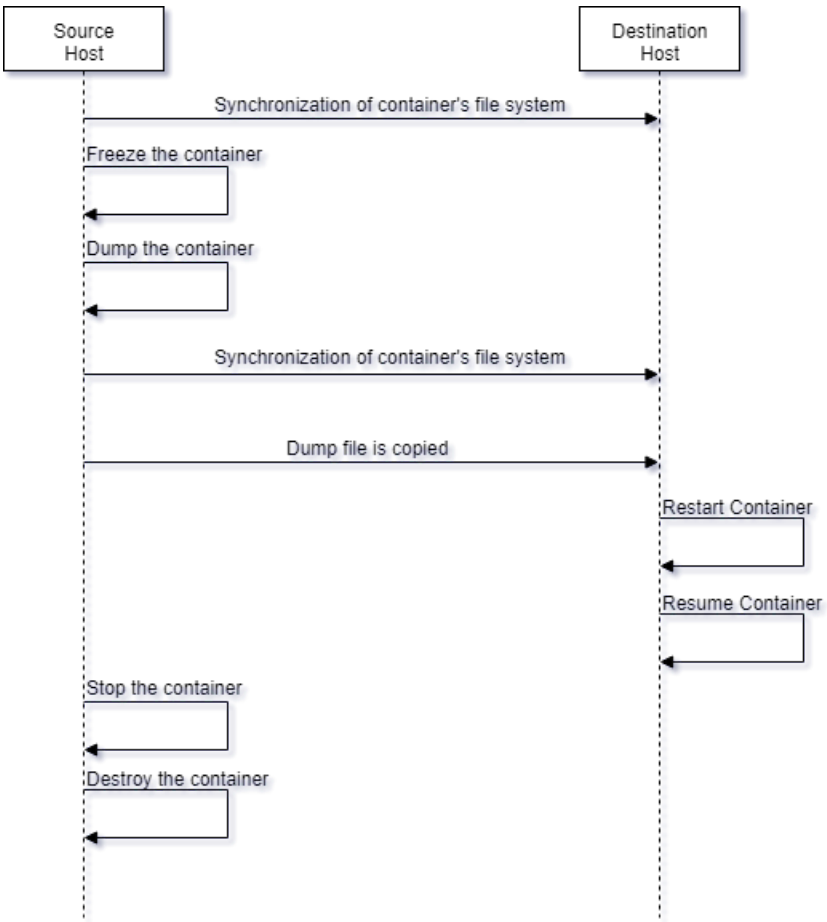

Figure 4.4: CRIU Principle Diagram [3]

4.3 Model Execution Plan for Stateless Application

In our stateless application example, Clock Application, we retrieve the system time from cloud instances. We created a table in the database to store the current instance

time. The users logging into the migration system, after navigating the clock tab should observe the clock timestamp with the provider cloud instance IP on the screen. Because our instances are located in the same geographic location, they have the same system time information. If the user navigates to the Clock tab by using its own browser application after connecting to the application server over the Internet, the user can see the clock data and this data is not affected by any user input and does not save any state at the execution time on the instance machine, which makes this application stateless, indeed.

If any migration decision is made for the source node, the application server kills the container running the application in it by sending a command to the source node. Then, it runs the same application on the destination node, in order to continue to provide the service to the end user.

Although the service provider address is changed after the migration process, the user does not have to change the address on the browser. This is because the application server is also a bridge between the user and service providers. The application server reads the output of the running application and serves the result to the end user. The instances which provide the service in the backstage write the application output to the database on the database server. The output is retrieved by the application server from the database and after parsing the output and rendering the new page, the application serves the output to the end user.

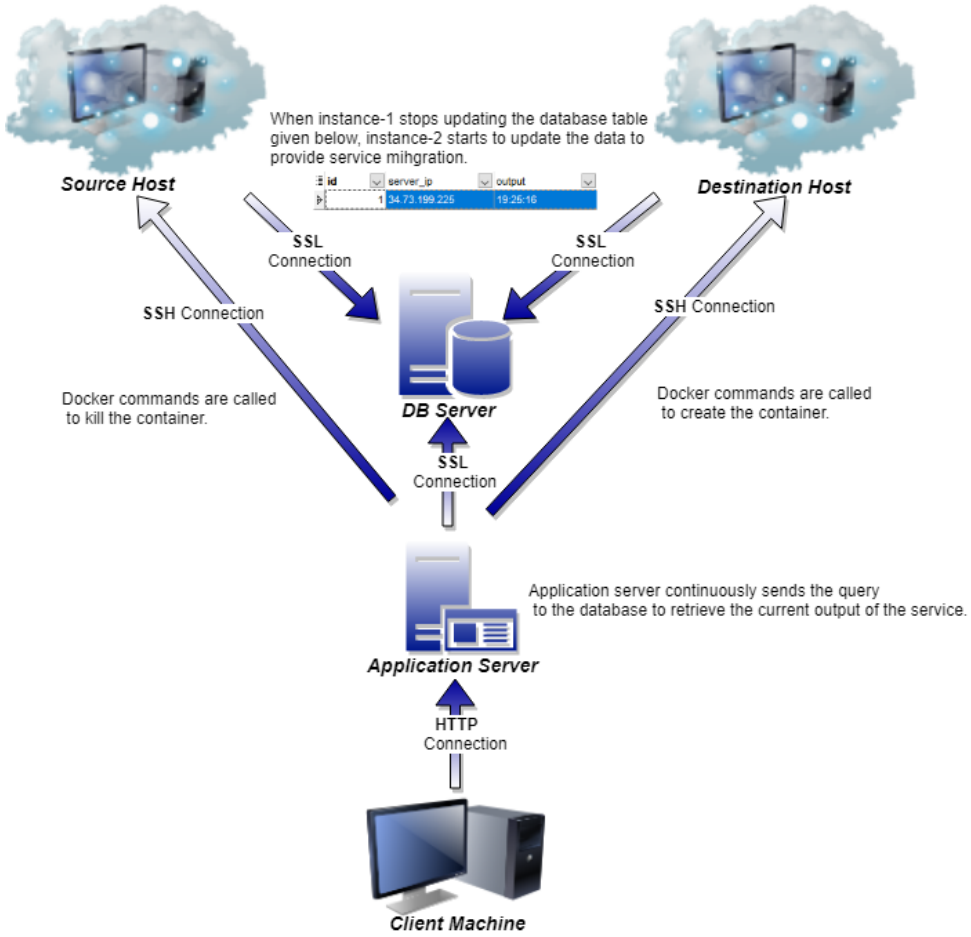

Figure 4.5: Clock Application Execution Plan

4.4 Model Execution Plan for Stateful Application

In our stateful application example, the Face Recognition application, we give an image as the input of the application. This application provides detection of the faces in the given image, saves them to the database, extracting the features of the given face and compares them with the images in the database. We integrated model training functionality to the application, in order to make analysis of the migration by making the application duration longer and making the checkpoint file size change dynamically. In other words, we will change the model training set size by giving parameters to the model training function in the source code. In order to be able to take metrics on the system performance, this modification is integrated to the application. It does

not affect the functionality of the application. It only affects the application runtime duration and the complexity of the checkpoint and resume operations.

The application server also acts as the bridge between the client and the cloud instances, which means that although the service provider address is changed after the migration process, the user does not have to navigate to the new address of the service provider. Because the application depends on the user input and is not required to execute infinitely as in the Clock application, we need to know the end of the program execution and make the end user wait until the execution is finished. In order to achieve that, we needed to save a status flag in the database. This flag holds the information on whether the result set is updated or not. The application server waits until the flag indicates that execution is ended. If the flag turns to 1, it means that the result is ready to display. The application server again retrieves the output from the database. It also parses the result and renders the page accordingly.

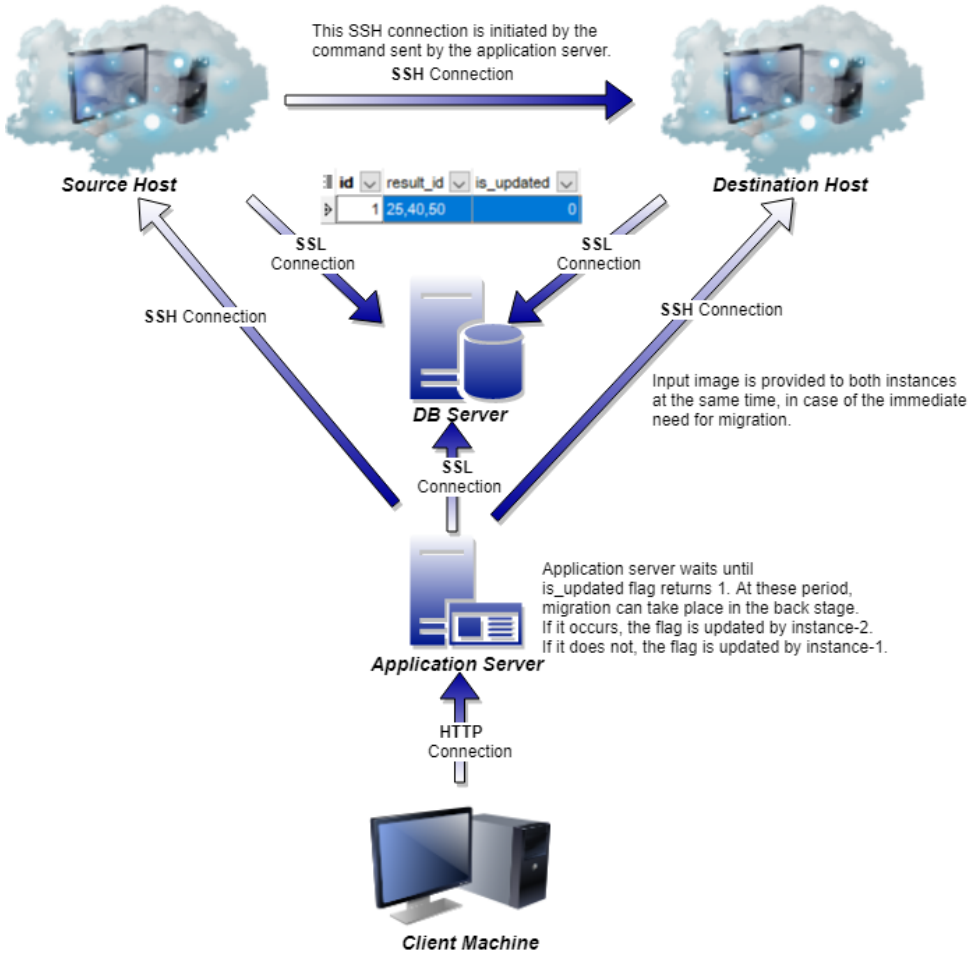

Figure 4.6: Face Recognition Application Execution Plan

4.5 Security Evaluation

Access Control is covered with this model. This factor requires that unauthorized users are not allowed to start, move and stop a machine. In our case, the rights to perform these operations belong to the application server. It acts like the control manager in our system. Only for test purposes, we added migration buttons to the user interface. Normally, the expected scenario includes that after the migration decision is made, the application server manages the remaining process.

Authentication is covered, because all the machines in the system authenticate to each other in order to perform the required operations remotely. Instead of password authentication, we used public key authentication with the help of strong key

exchange and symmetric encryption algorithms.

Data Confidentiality is covered, too. While migrating, we used an SSH channel and the SFTP file transfer method to guarantee the data encryption between the source and the destination. This enabled us to protect the migration process from man-in-the-middle attacks.

Communication Security is covered, as the data transmission channel is protected via SSH protocol. This security factor requires to define a secure migration path between the source and the destination hosts, which is achieved via the SSH protocol.

Data integrity is also provided by this system, as the SSH channel utilizes available MAC algorithms to provide data integrity.

Availability security factor is similar to the access control and authentication security factors. Because unauthorized users are not allowed to initiate a migration process, the system avoids DoS attacks initiating unnecessary outgoing migrations in an increasing manner, which makes the system resources unavailable to the authorized users.

4.6 Attack Resilience

By satisfying the mentioned security factors in the previous section, we can avoid the attacks given in the following:

- The first attack avoided is the man-in-the-middle-attack. This is because nobody else can actively eavesdrop on the data while migrating with the help of the secure mechanism of the SSH protocol.
- DoS is also avoided with this model. The resources are not allowed to be used by unauthorized users. In addition to that, SSH has a configuration parameter named as MaxSessions which specifies the maximum open sessions' count allowed per network connection. The default is 10. If we try to open more sessions than this specified value, the SSH connection is automatically closed, which prevents DoS.

- Overflow attack is also avoided, because the transmission channel is protected by the SSH protocol, attackers which are unauthorized users (who cannot perform public key authentication) can not cause a congestion in the communication channel traffic.

- Replay attack is achieved by re-transmitting the replicates of the dirty pages in a sequential manner. The frequent dirty page occurrence results in a replay attack. In our scenario, only the checkpoint folder in a compressed file format is transmitted from source to destination after the application server triggers the migration. The destination host is specified by the application server and the file transmission is performed once for each migration.

CHAPTER 5

EXPERIMENTAL EVALUATION

This section describes the experimentation setup used to analyze the overhead caused by proposed secure live migration model.

5.0.1 Experiment Setup

5.0.1.1 Source and Destination Cloud Instance Setup

The experimental setup includes two identical Google Cloud [9] instances. One of them is the destination node, the other one is the source node. Checkpoint/Restore application (CRIU) runs on a dockerized image. This image is based on an Ubuntu distribution. The details of cloud instances' configuration is visualized in Figure 5.1.

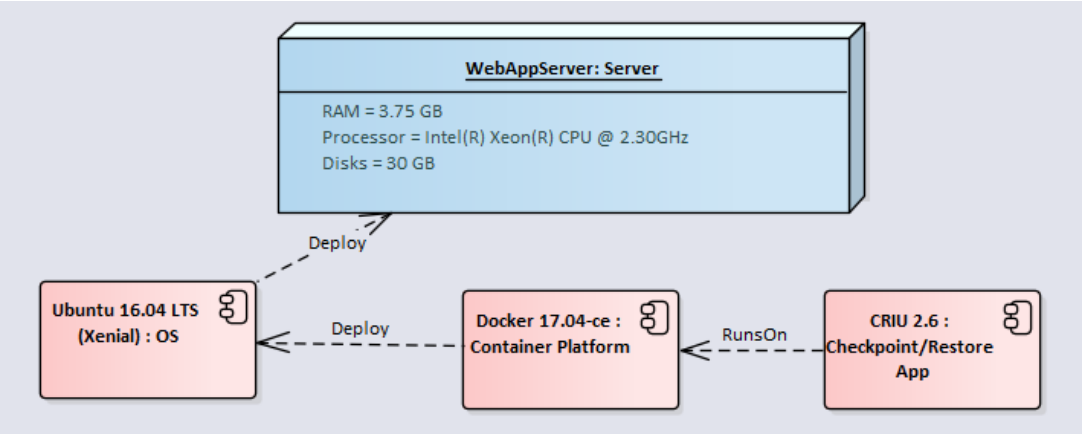

Figure 5.1: Cloud Instance Details

Throughout the document, these identical instances are named as instance1 and in-

stance2, which are source and destination machines on which the migration process is performed. Access point for those machines are given as IP addresses in the following table.

Name	**IP Address**
instance1	35.232.36.90
instance2	35.232.243.128

Table 5.1: IP Addresses of Instances

5.0.1.2 Docker and CRIU Configurations

Docker is required to be installed on both instances. The checkpoint/restore functionality is supported in the Docker 17.04.0-ce version. However, one additional configuration is needed to achieve use of CRIU tool together with Docker to checkpoint the applications. Because CRIU is an experimental feature of Docker, we need to enable the experimental option of the installed Docker. In order to do that, we should create a new file by using the following command on the terminal window:

sudo nano /etc/docker/daemon.json

and we need to type the below lines into that file:

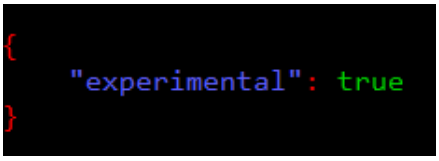

Figure 5.2: Experimental Setting

After saving that file, we need to type the following in the terminal window:

sudo service docker restart

In order to be sure about that experimental functions are enabled, we need to call this command in the terminal:

docker version

The output of that command should provide the result as given in the figure below.

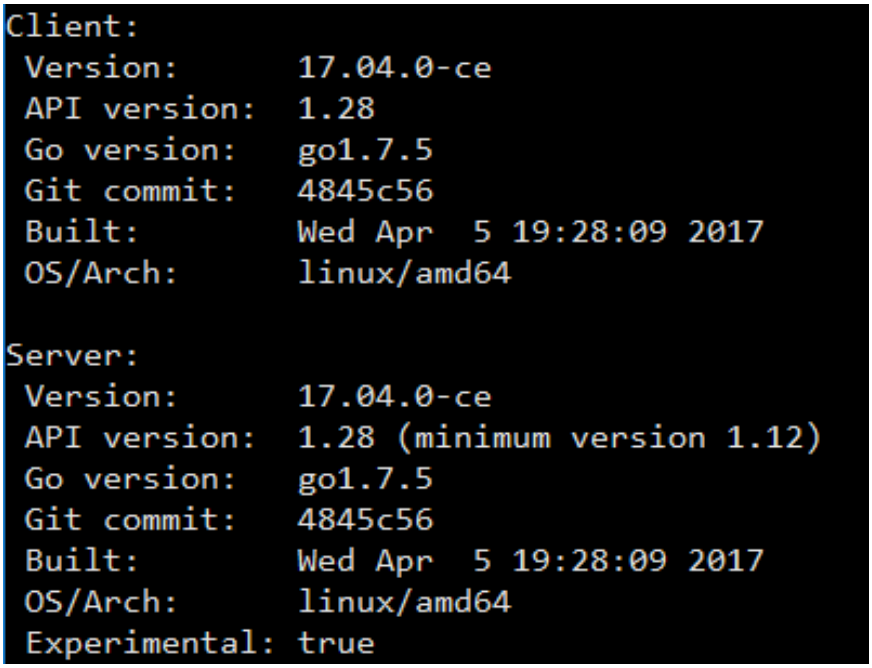

Figure 5.3: Docker Configuration

After Docker installation, CRIU installation is required by using the following command:

sudo apt-get install criu

The experiment is conducted with the version of 2.6 for CRIU.

5.0.1.3 Application Server Setup

Stateful/Stateless applications run on the execution environment given in 5.4:

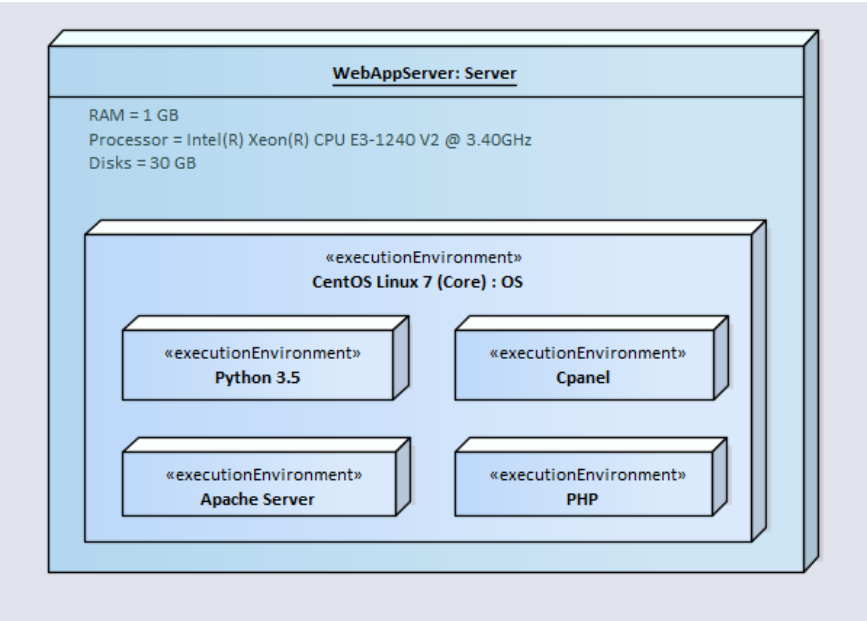

Figure 5.4: Application Server Details

Python 3.5, Cpanel, Apache Server and PhP are installed in order to make the application server meet the requirements of the system architecture.

5.0.1.4 Database Server Setup

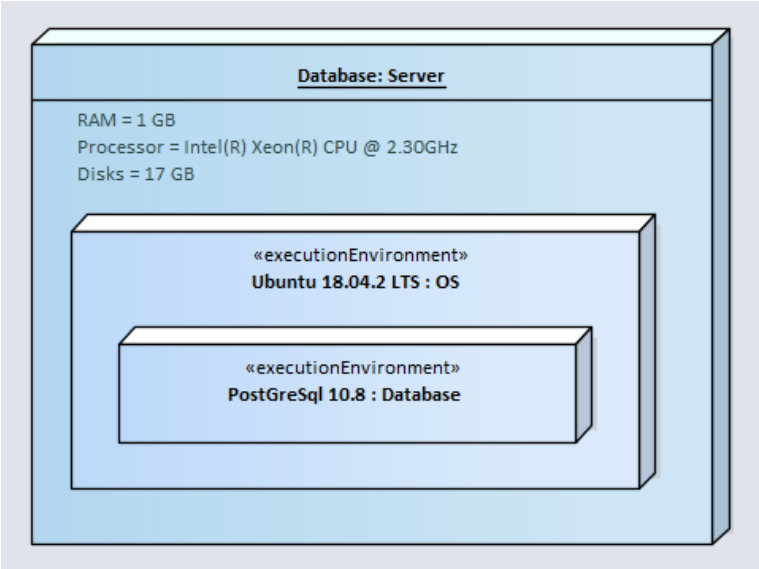

Figure 5.5: Database Server Details

The configuration of the database server setup is given in Figure 5.4.

PostGreSql 10.8 should be installed on the database server in order to execute SQL commands sent by instances. In order to connect to DB, we use the SSL protocol. Postgresql provides the ability to give access to specific machines. Therefore, we specify IP of the instances and application server to the Postgresql configuration files. This means that no other machine other than cloud instances and the application server can connect to the database to execute any query. In the database cloud instance, under /etc/postgresql/10/main folder, there are configuration files provided to specify allowed IP addresses. The configuration file named as pg_hba.conf is edited in the system to allow just cloud source and destination instances and the application server as given in the following:

```
# Allow postgres user from the hosts with specified IP addresses to connect
# to database named as "face".
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
  host  face            postgres        94.102.5.113/24         md5
  host  face            postgres        35.232.36.90/24         md5
  host  face            postgres        108.59.86.12/24         md5
```

Figure 5.6: SSL Connection Configuration

5.0.1.5 Applications Used in Experiments

The container live migration system needs to be analyzed with two kinds of applications: stateless and stateful, because the system reaction can change based on the application type. For the stateless applications, if an attack is detected, the container is migrated to a new host without a need to save application state and resume from that state. Therefore, the experiment with stateless applications does not include CRIU tool utilization. For stateful ones, checkpointing is required to achieve the goal of preventing the loss of service provided to the cloud client.

Clock Application For the stateless application example, a clock application is used. This application mainly gives the system time as the output. The resulting system time with the instance IP address is displayed on a browser on client ma-

chines.

Installation

- Python 3.3+ is required to be installed on the cloud instances having containers. In this experiment python:3.6-slim-stretch is used.
- ClockApp folder including source code, Dockerfile and requirements.txt files, should be copied to the instance.
- In order to create a docker image, after navigating to the installed ClockApp folder, it is required to be built by using Dockerfile by typing the below command:

  docker build –tag=clockimage .

Face Recognition Application For the stateful application example, a face recognition application is used. This application provides to recognize and manipulate faces from Python console or the command line by using a popular face recognition library. It also enables to apply face recognition on a folder containing images. In order to recognize faces, it uses the dlib library, which is developed with a deep learning technique. One of the features of this application is face detection, which means that it detects all the people's faces seen on a photo and gives the coordinates of the faces on the photo as output. The other feature is to identify the people's faces appearing on a photo. In other words, it matches the given face with the other face photos of the same people which is saved in the DB.

Installation

- Python 3.3+ is required to be installed on the cloud instances having containers. In this experiment Python python:3.6-slim-stretch is used.
- dlib installation is required. In order to do that:
  - First, the source code should be cloned from github:

    git clone https://github.com/davisking/dlib.git

- Build the main dlib library:

  cd dlib

  mkdir build

  cd build

  cmake ..

  cmake –build .

- Compile&Install the required Python extensions after navigating to the dlib folder:

  python3 setup.py install

- Then, installation of the module from pypi using pip3 is required:

  pip3 install face_recognition

- In order to create a Docker image, after navigating to the installed face_recognition folder, it is required to build with using Dockerfile by typing the below command:

  docker build –tag=facerecog .

In order to containerize an application, it is required to build an image as described in the Dockerfile. Then, on that image, containers can be run by using the docker run command. In the following figure, the schema of this process is given.

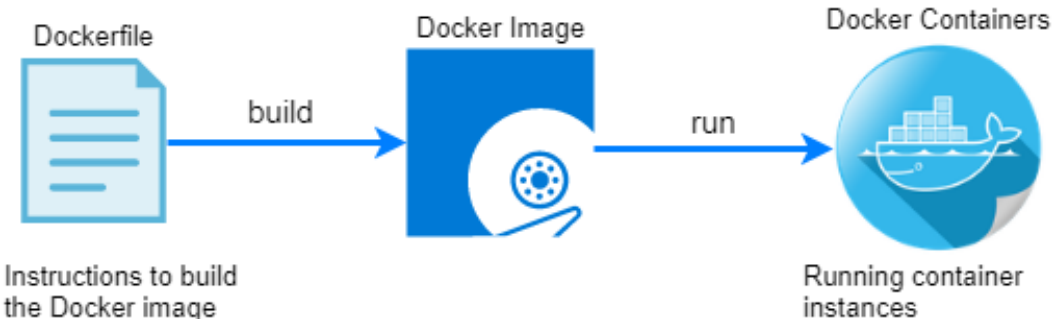

Figure 5.7: Containerizing Stages of an Application

5.0.1.6 SSH Connection and File Transfer Application

An application is developed to establish an SSH connection between cloud instances and between an instance and an application server. This application also provides to transfer checkpoint files with SFTP. In order to do that, ssh should be configured to provide remote login in the related machine. Configuration steps should be as given in the following:

- SSH configuration is applied by editing the sshd_config file. In order to achieve that we need to type the following on the command line:
  sudo nano /etc/ssh/sshd_config

  In that file, these features should be set as given in the following:

  - PermitRootLogin yes
  - RSAAuthentication yes
  - PubAuthentication yes
  - AuthorizedKeysFile /root/.ssh/authorized_keys

- Application uses public key authentication.

- Application server sends a request to the source instance to execute this Python application. Therefore, this Python code should be placed in the correct path of the source instance specified in the application server. In our experiment, it is placed under the "/home/zmavus/dockerScript/" folder.

- The application is developed with Python. In the previous, applications used in the experiment require the Python installation steps. Therefore, before applying these steps, we need to verify the Python installation. Python 3.3+ is required for the experiments.

SSH Public Key Authentication: As an alternative to login to an SSH server with a password, public key authentication is supported by secure shell protocol. Public key authentication when compared to password-based authentication is not the main subject for brute force hacking attacks. In addition to that, if the password is known

by a hacker, it means that the hacker can login to the system without requiring any extra information. However, if a public key is known by a hacker, the hacker cannot authenticate directly to the system because the private key pair of that public key is known only by the ssh server. Therefore, it is not enough to authenticate to the system, which makes the connection more secure when compared to password authentication. Moreover, public key authentication has many other benefits if multiple clients try to login to the same system. The single server password is not required to be shared between clients. Instead of this, public key authentication provides multiple clients to login to the system without having that single password. It makes the login process easier for a client to many servers without a need to manage all required passwords.

Public key authentication has a requirement to have a key pair, namely public and private keys. The private key is never shared with other parties. Only the public key is shared and when it is matched with a private key when authenticating, it means that the sender of the public key can login to the system. All key pairs are unique and these two keys work together in that mechanism. [51] Public key authentication is achieved in our system by applying the following steps:

- A key pair is generated by using the following command on the client side:
  ssh-keygen -t rsa
  This command generates two different files. The one is named as id_rsa which contains the private key. The other one is named as id_rsa.pub which contains the public key. The private key file is saved in /<username>/.ssh/id_rsa. The public key file is saved in /<username>/.ssh/id_rsa.pub.

- The public key is uploaded to the server by using the following command in the client side:
  ssh-copy-id -i /.ssh/id_rsa.pub <username>@<server_ip>
  This command edits the /.ssh/authorized_keys in order to add public key in it. In order to do that, it asks for password once.

- The permission of the files on the .ssh directory is corrected on the server side by using the following commands:
  chmod 700 /root/.ssh;
  chmod 640 /root/.ssh/authorized_keys

In our experiments the above process is applied among the application server, source and destination nodes. Because the user inputs entered are forwarded by the application server to the cloud instances over the SSH channel. The checkpoint files are forwarded from the source of the live migration process to the destination.

The communication encryption and key exchange algorithms are handled by the SSH server and client programs. There are also configurations provided to the users in order to make a choice between the algorithms in order to manage the security level. The checkpoint folder contains the application snapshot and it should be migrated in a secure manner. In order to achieve that, by using the algorithm configuration interface of the paramiko, we set the algorithms by injecting these lines into the SSH connection and File Transfer application source code:

paramiko.Transport._preferred_ciphers = ('aes256-ctr',)
paramiko.Transport._preferred_kex = ('diffie-hellman-group-exchange-sha256',)

In other words, for SSH communication encryption, AES algorithm in counter mode with 256-bits key size is used. For the key exchange process, Diffie Hellman Key Exchange algorithm is used with the SHA-256 algorithm as hashing. SHA-256 is the Secure Hashing Algorithm, which generates a hash value of 256-bits.

5.0.2 Experiment Metrics

In this section, performance of the proposed model is measured based on the metrics listed below.

- **Total migration time** refers to the time frame starting from the migration operation initialization, and ending at the successful resume operation of the containerized application at the destination host.
- **Application downtime** refers to the total amount of time passed between stopping the containerized application at the source and resuming it at the destination from where it stopped. In this time period, service becomes unavailable to the user in the background. Because of that, this is an important metric to be

able to decide performance of the migration technique. Downtime is the summation of the time needed to transfer files between source and the destination, stop the container at the source and start it at the destination.

- **File transfer duration** refers to the time frame required to transfer all checkpoint files from source host to the destination. This metric has a great impact on the downtime metric.
- **Transfer file size** refers to the size of the folder generated by the CRIU. It changes according to application state and memory used at that time.
- **Upload speed** has a great impact on file transfer duration, which means it has also impact on the downtime metric.
- **Checkpoint duration** refers to the timeframe required to take the application snapshot at a specific time of the execution.
- **Resume duration** refers to the timeframe required to resume an application at the destination node from where it stopped on the source node.

5.0.3 Experiment Results and Discussion

The first step in the experiment is to run one container on the source node and migrate it to the destination node. By using that step, the gathered data of the specified metrics during the whole migration process is analyzed.

For the stateless application, there is no need to checkpoint the application. Because it is stateless, the snapshot of the application is not required to be taken. Therefore, checkpoint related metrics are not applicable to stateless applications. For the stateful applications, all of the metrics given in the previous section are applicable.

The results obtained in the first step of the experiment show that the above metrics have a strong relation in between. Total migration time basically depends of the file size to be transferred during migration. In addition to that, the migration process consumes the system resources like network bandwidth, CPU time etc. Unless there are enough free system resources on the source machine, the migration time is influenced negatively from these parameters. Although the checkpoint file is not counted as large

to transfer between nodes, the migration time can get expanded because of low bandwidth. The first step is repeated seven times by using the Python logging library, log files are generated in order to monitor the whole process and get the duration of each function used in the live migration process.

The migration performance of the model can be evaluated by analyzing the total migration time and downtime, which means that these are the main metrics to be analyzed. Downtime is a part of the total migration time. The Figure 5.8 shows both of the total migration time and downtime analysis of containers and indicates that the total migration duration has a dependency on the checkpoint file size which is transferred to the destination node. Test results given in Figure 5.8 belongs to Face Recognition Application.

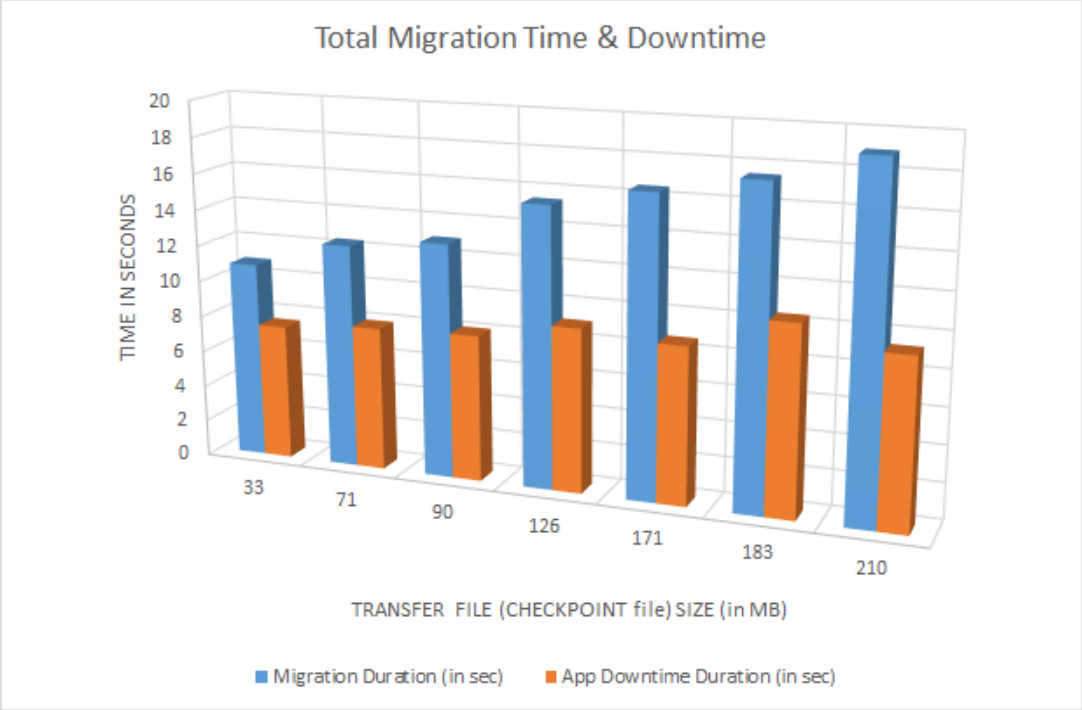

Figure 5.8: Total Migration Time and Downtime

Downtime has a great impact on total migration time. However, there is still another factor which affects the total migration time other than the downtime. This factor is file upload speed in the migration channel between source and destination nodes. Because Figure 5.8 shows that both of the total migration time and downtime increases when the transfer file size increases. However, it does not increase linearly because

of non-linear network bandwidth between the source and the destination nodes.

Upload speed during migration is the parameter which has a big impact on migration performance. Migration Link bandwidth has inverse ratio with both the downtime and total migration time. When the upload speed gets higher, it means that faster file transfer will occur and it will result in less time to complete the migration process. Figure 5.9 shows the relation between them according to the data logged in seven test cases.

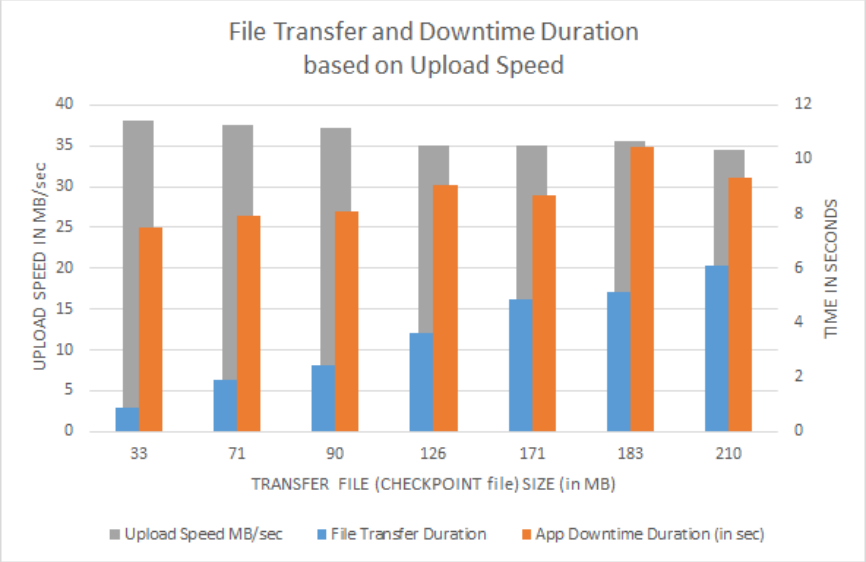

Figure 5.9: Upload Speed Effect on Downtime and File Transfer Duration

Checkpoint and resume operations are also part of the total migration time. Relative to file transfer effect on total migration, checkpoint and resume duration has a smaller effect on the total migration time when migrating one container from source to destination host.

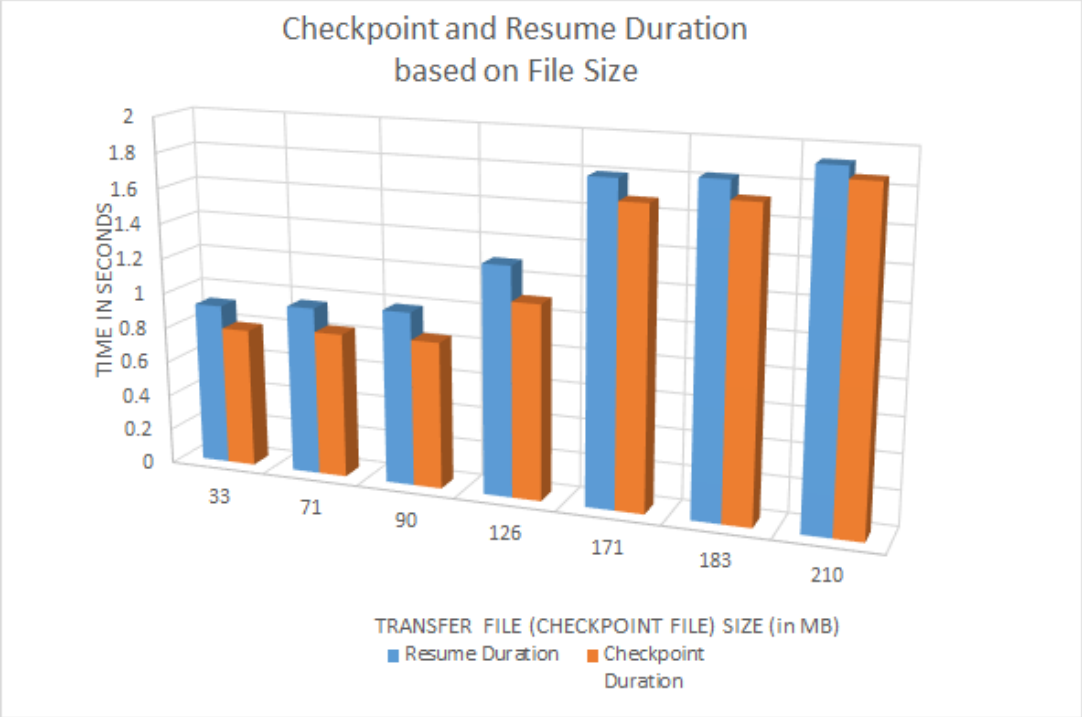

Figure 5.10: Checkpoint and Resume Duration

For the stateless application, killing the container duration on source host, starting the container on the destination host, downtime and total migration time are the only metrics applicable. Downtime refers to the timeframe between killing the container on source and starting the container on destination. Total migration duration refers to all the migration preparation operations such as establishing secure SSH connection using public key authentication and removing the containers with the same name on the hosts, before starting on the destination and after killing on the source. The experiment is performed seven times and the results are provided in the Table 5.2

Kill Duration	Start Duration	Downtime	Total Migration Duration
0.57733798	0.55944109	1.87	2.577
0.546288013	0.568042994	1.888	2.481
0.538717031	0.559737206	1.764	2.346
0.64366889	0.557837963	2.408	3.075
0.65865612	0.556710005	2.392	3.171
0.657931805	0.5573771	2.386	3.162
0.637104988	0.55532217	2.396	3.15

Table 5.2: Stateless Clock Application Experiment Results

The stages of the application are given in the following Figure 5.11. Kill duration, start duration and downtime are counted as the stages of the application migration. According to the graph the biggest part of the migration timeframe belongs to the application downtime, which is the time difference between killing the container on the source host and starting the new container on the destination host.

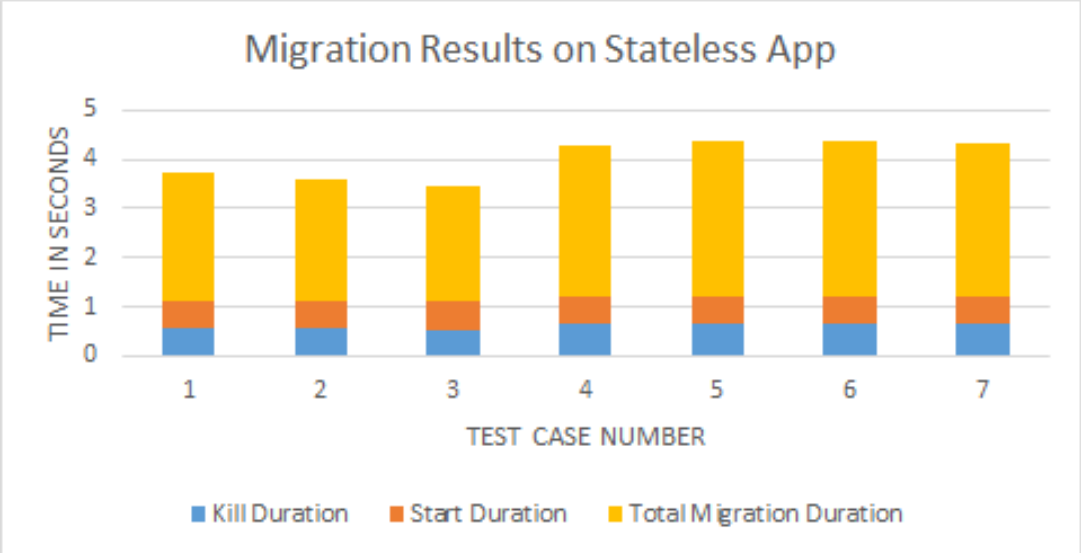

Figure 5.11: Stateless Clock Application Experiment Results Graph

5.0.3.1 Multi User Test Scenario Results

For the stateless application example (Clock Application) in our experiments, there is no need to create a new container for each user in the cloud instances. Because the service does not save any state, the output of the service is saved in the database server periodically. Therefore one container is enough to provide service to all users connected to the system.

For the stateful application example (Face Recognition Application), the service is not running infinitely. For each request taken from the user, a new container is created and started to run. When the execution of the application is finished, the container is killed by using the command provided for the Docker daemon. In order to provide isolation between the processes initiated based on the user requests, we preferred creating a new container for each request taken.

When the number of containers increases, the checkpoint file size increases as we expected. This is because for the each container a new checkpoint folder is generated by the CRIU tool. After checkpointing all the containers, we compress the folders into one and transfer this compressed file to the destination host. Table 5.3 shows the number of containers used in the experiments and both of the CPU and RAM resources allocated to achieve the migration process for that many containers are given. Upload speed and file transfer duration measured fors these experiments are also given in the table. When we increase the container count to 50, it is required to increase the resources to handle the migration process.

Number of Containers	Source		Destination		Upload Speed (MB/sec)	File Transfer Duration (sec)
	CPU Count	RAM (GB)	CPU Count	RAM (GB)		
3	1	3.75	1	3.75	20.9	10.240219
10	1	3.75	1	3.75	21.9	11.338588
20	1	3.75	1	3.75	19.4	15.227305
25	1	3.75	1	3.75	22.4	23.300659

Table 5.3: Multi User Test Cases and Transfer Duration Results

For the Face Recognition application, total migration time changes according to file size and container count given in the Figure 5.12. The file size reaches a value in gigabytes, the file transfer process which includes encryption and decryption at the source and destination relatively increases the total migration time. However, increase in upload speed results in decrease in downtime and total migration time.

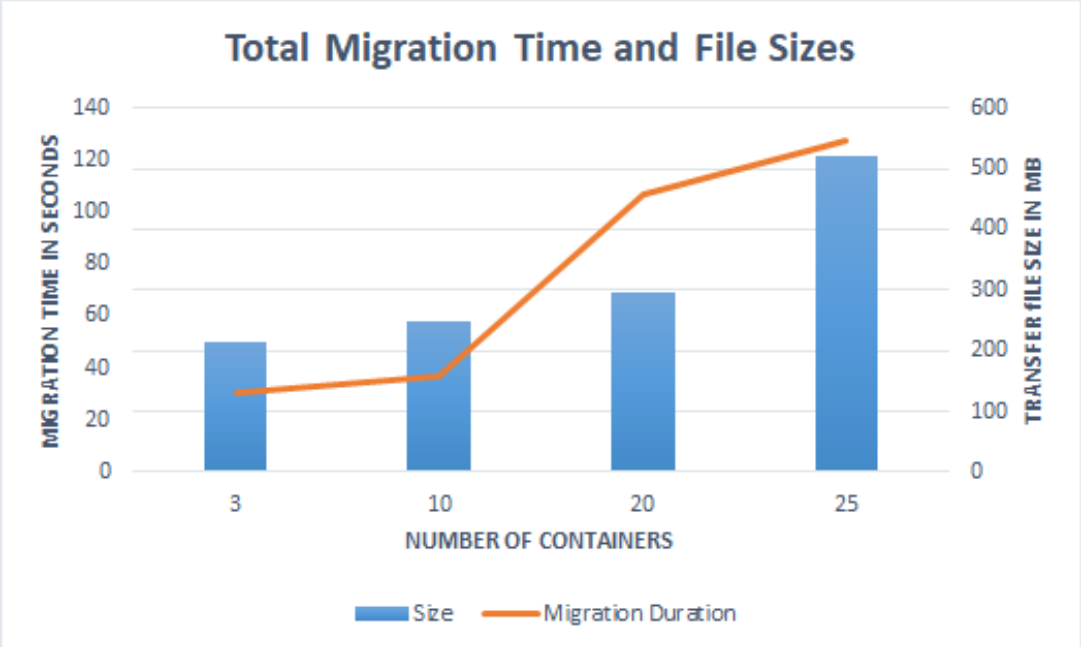

Figure 5.12: Multi-User Total Migration Time and Checkpoint File Sizes

In the Figure 5.13, we observed that the resume operation takes much longer compared to the checkpoint operation, when we increase the number of containers. When we migrate multiple containers at the same time, in order to speed up the checkpoint operation, each container is stopped after checkpointing. However, in the destination host, while reading the checkpoint files from the disk, each newly resumed application also starts to consume the resources. Each checkpointing operation results in freezing one of the containers but each resuming operation results in starting one of the containers. Therefore, resource consumption increases rapidly, which results in a long resume duration relative to checkpointing.

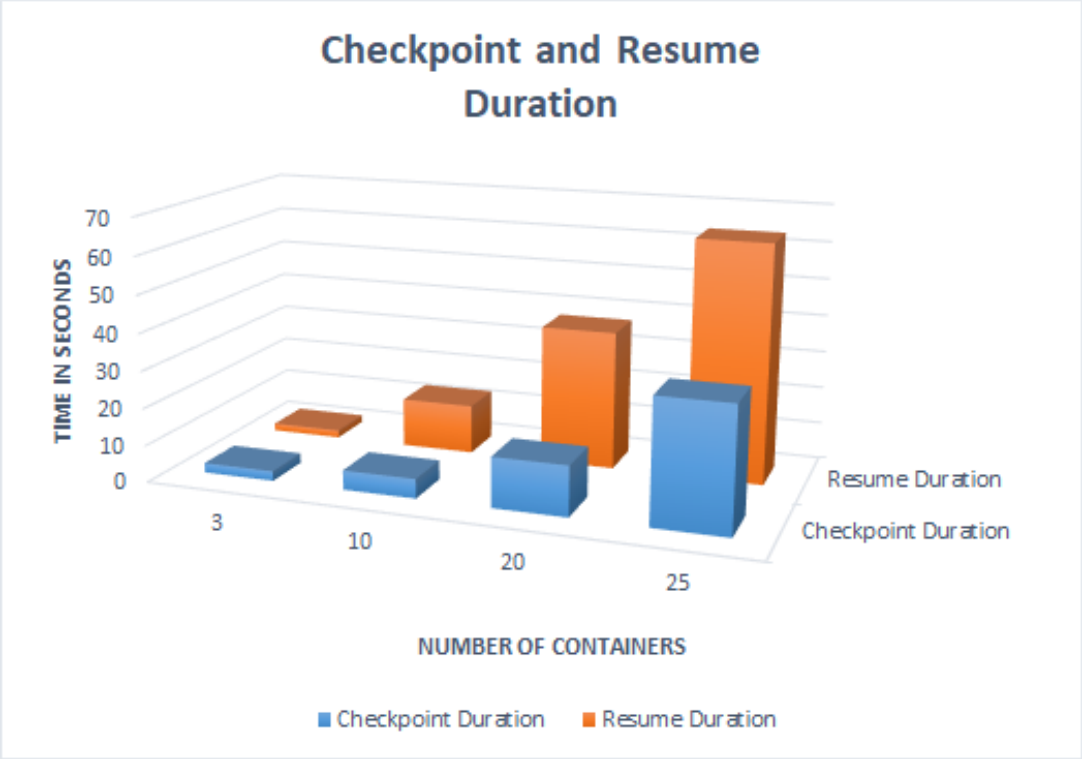

Figure 5.13: Multi-User Checkpoint and Resume Duration

5.0.3.2 Comparison with Layered Framework Experiment

The experiments conducted in [7] to analyze the live migration performance of the proposed layered framework were performed with the following applications:

- Game Server Application: That is a server for an online game which is used to

transmit/receive messages according to user position.

- RAM Simulation Application: In order to simulate memory consumer applications, a script application is used, named as RAM Simulation. The memory utilization of the application is specified by the user as a parameter to that script.
- Video Streaming Application: This application requires to store a video of size 50 MB in the mobile edge cloud together with the application.
- Face Detection Application: That is an application used to detect faces in a video.
- No Application: This represents only the operating system with no additional application running. This is used for analyzing the additional cost of the migration of the applications.

Three virtual machines were used to perform the experiments. The details of the VMs used in the experiment setup are as given in the following table:

Operating System	Ubuntu 15.10
Ram	2 GB
Processor	2 vCPU Intel Core i7 @ 2.6GHz
Link Bandwidth	100 Mbps

Table 5.4: Layered Framework Experiment Setup Machines' Details

Two of the virtual machines used in that setup were used as mobile edge clouds on which the migration process takes place. The other virtual machine was used as user. Each virtual machine standing for a mobile edge cloud includes a KVM and LXC in the specified operating system in a nested manner to be able to provide the service to the user. In that experiment, the rsync command is used for iterative file system synchronization. Rsync is a command to copy and synchronize files/folders between remote machines that is used in Unix/Linux systems. Via using the rsync command, a Linux system can be reflected to another system by performing synchronization of

files, networks, data backups across the source and destination machines. No encryption is performed on the rsync command on its own unless a secure channel is used between the machines.

	LXC Total Data Transfer Size (in MB)	**LXC Total Migration Time (in sec)**	**Downtime (in sec)**
No Application	1.4	6.3	2.0
Game Server	1.6	6.4	2.0
RAM Simulation	97.1	19.8	15.3
Video Streaming	7.4	8.5	3.3
Face Detection	10.0	15.5	3.7

Table 5.5: Total Migration Duration and Transfer Data Size for LXC

	KVM Total Data Transfer Size (in MB)	**KVM Total Migration Time (in sec)**	**Downtime (in sec)**
No Application	65.2	79.7	56.1
Game Server	69.7	80.9	58.7
RAM Simulation	170.4	93.0	72.0
Video Streaming	87.3	86.4	61.3
Face Detection	99.0	152.3	107.5

Table 5.6: Total Migration Duration and Transfer Data Size for KVM

The experiment results as shown in the Table 5.5 and Table 5.6 are obtained by computing the mean value of seven distinct executions.

According to the test results, the authors claim that containers' performance is better than VMs'. The reason is that containers are lightweight compared to virtual machines, which means that the file system and the memory context are related to the running application. On the other hand, VM can have much more unrelated information to the application. Rsync method is required to take the difference of much more

data when compared to containers. Therefore, migration duration and downtime is less than VMs [7].

The following Figure 5.14 and Figure 5.15 are drawn with the data given in the Table 5.5 and Table 5.6 in the order of the applications given (No Application, Game Server, RAM Simulation, Video Streaming, Face Detection given from left to right numbered as from 1 to 5).

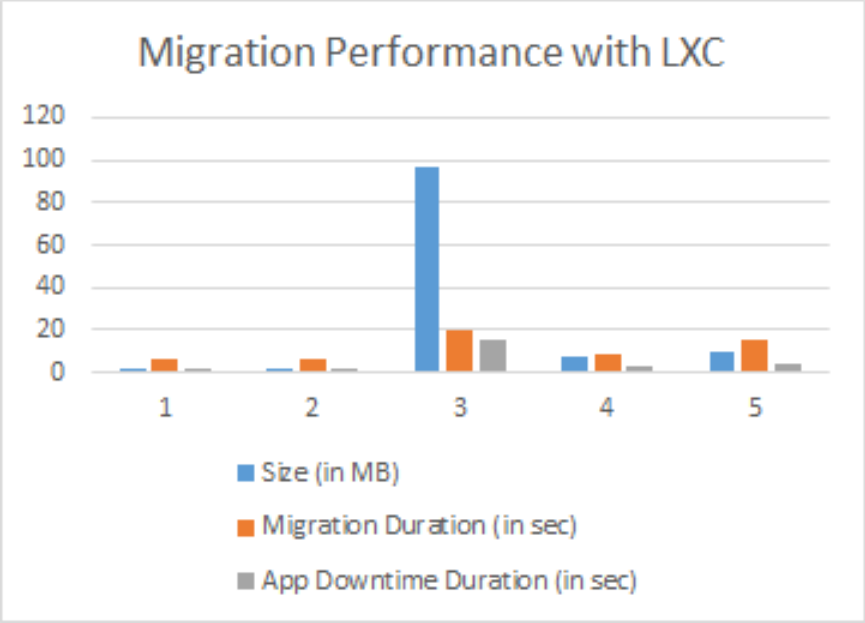

Figure 5.14: LXC Results [7]

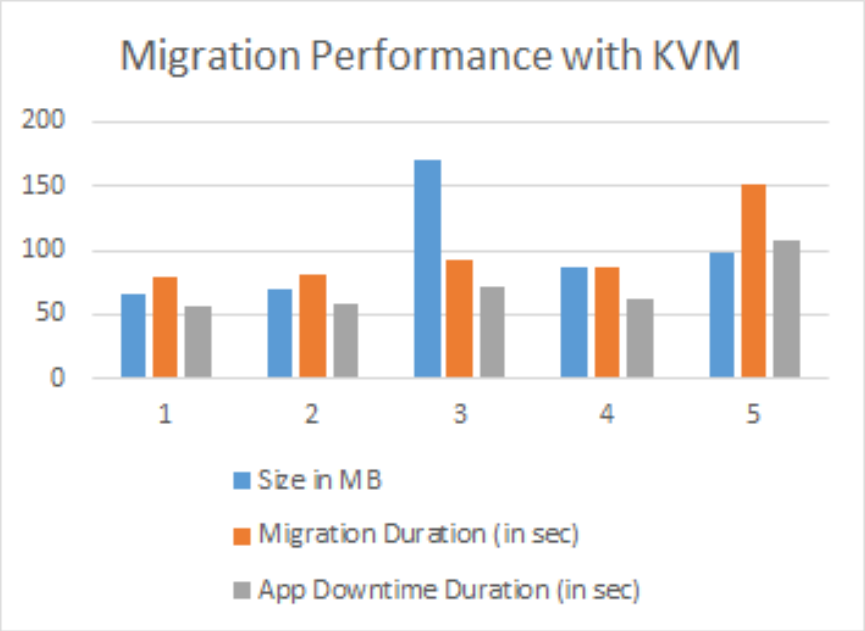

Figure 5.15: KVM Results [7]

When we compare the above results with the results of our architecture performance analysis experiment results, we can support the idea provided by the authors of the paper given in [7], which is that container performance is better than VMs for live migration of applications. Although we have run our experiments with the instances having 1 vCPU, 3.75 GB memory and changing link bandwidth used in that experiment, we obtained better results especially than the KVM migration performance. Our architecture flow assumes that all the applications that need to be migrated have already been installed to the potential migration machines. Therefore, in order to compare our results with the layered framework, we need to use the data given in the Application Found results of the layered framework as given in the Table 5.5 and Table 5.6. If we compare the boundary values given in the test results, with the 99 MB transfer data size, the migration process takes 152.3 seconds on average with 100 Mbps (12.5 MBps) bandwidth. However, the transferred data size is close to 99 MB, migrated within approximately 13 seconds on average with 36 MBps bandwidth, which is much less than the KVM migration result clearly. In order to make bandwidth values of the experiments the same, we can multiply the bandwidth of 12.5 MBps with 3, which means that it will migrate 297 MB of data in that timeframe, which can be migrated approximately in 30 seconds in our model, which is still much

less than the result obtained in the KVM migration model. In addition to that, in this test scenario 107.5 sec downtime is given in the Table 5.6. For even a transferred data with size 250 MB, our model downtime is measured as 11.127 seconds.

When we compare the results with the results given in Table 5.5, we see that the migration data size is small with respect to our model. Although the size of the transferred data is smaller than the transferred data obtained in our model, the total migration time takes longer with LXC. For instance, in a 15.5-second timeframe, transferred data with size 10.0 MB can be migrated with 100 Mbps (12.5 MBps) bandwidth. In our architecture, in that timeframe, we could migrate approximately 126 MB of transfer data with 35 MBps bandwidth. In order to make bandwidth values of the experiments the same, we can multiply the bandwidth of 12.5 MBps with 3, which means that it will migrate 30.0 MB transfer data in that timeframe, which is still less than the 126 MB result obtained in our test case.

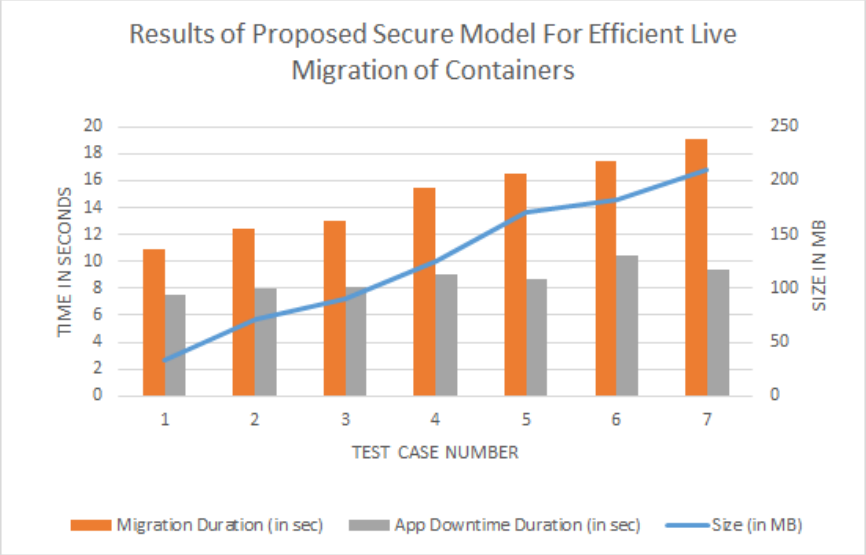

Figure 5.16: Proposed Secure Model Results with Face Recognition Application

While Machen et al [7] uses the rsync tool to transfer data incrementally belonging to the specified process, our proposed model compresses application's data and transfers this compressed data, which saves bandwidth drastically. While incremental synchronization of data by benefiting from differences of instances seems to be a pretty good idea, it could suffer from the duration finding deltas and building the current version of the application from the old version and deltas on the destination side. If the appli-

cation's data changes frequently, then the majority of the data should be transferred. Incremental synchronization could lose its advantage easily in these scenarios. Our proposed model handles the situation in a standard way by transferring compressed application's data that provides predictable effort.

In the layered framework, security issues are not mentioned. However, security is a very important aspect and gets ahead of the performance in mission-critical domains. In our framework, additionally we applied encryption on the transfer data. By using the public key authentication, we also protect the passwords of the instances. Instead of rsync, we used the SFTP protocol over an SSH channel with the enabled channel encryption mechanism.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Cloud computing provides an efficient common pool of hardware and software resources which can be scaled up and down in a simple manner, based on the various and continuously changing requirements of the users. Because cloud computing diminishes the cost and complexity of resource management, it has become preferred by users and the organizations. Cloud services as the applications running in the cloud are encapsulated in virtualized environments in order to lower the resource cost. Virtual machines and containers are utilized to create a virtual environment for cloud services. In order to achieve effective resource utilization, live migration feature should also be supported by these virtualization techniques.

Live migration, by definition, is the process of moving a running service between different physical or cloud machines. In addition to its contribution to efficient resource utilization, live migration also plays an important role for system maintenance, load balancing and applying moving target defense technique to make the service avoid attacks. Based on the small occupied space in memory and time required to start up a service fast as provided by container virtualization, the container technology is getting adopted in a fast manner. [52] There are many studies in the literature on VM migration issues already [4], [5], [35]. Container is still a relatively new technology when compared to virtual machines. There are still issues that need to be addressed in the existing literature. One of the most important issue is container live migration security.

When a migration decision is made and the process to migrate the service is started, the system should not be vulnerable to attacks. In this work, we propose a model to apply live migration on containers in a secure manner. Docker containers are used

to implement the model and conduct the experiments in this study. By using Docker containers, faster and more lightweight migration is possible. Because virtual machine migration encounters serious problems from the performance aspect and the key features of the containers such as having lightweight structure and supporting application isolation, container live migration has more advantages relative to virtual machine live migration [53].

In the proposed model, checkpointing is used to take the snapshot of the running application. In order to achieve that, we utilized the CRIU [19] tool providing checkpoint and resume functionality on Docker container services. Via this method, we achieved to decrease transfer file size during migration, which affects model performance efficiency and the time required to transfer data in a secure way, in an encrypted channel in our proposed model. In our proposed model, in order to provide communication security and make the system protected against migration attacks, we provide security of the migration data using secure authentication, and ensuring all connections between the nodes are protected. The efficiency of the migration system designed based on the proposed model has been proven on stateless and stateful sample applications. Experiments with sample applications running on the Docker container platform demonstrate that the proposed approach achieves significantly better performance than its virtual machine live migration counterpart.

Optimization on the multi user scenario should be applied as a future work. In addition to that, a machine learning algorithm can be developed in order to decide the migration process source and destination nodes. In our model, the application server acts like control manager for the migration process. In order to avoid single point of failure, application server can be divided into multiple application servers working in a distributed manner.

REFERENCES

[1] David A. Carts, "A Review of the Diffie-Hellman Algorithm and its Use in Secure Internet Protocols," tech. rep., SANS Institute, 2001.

[2] "Container Live Migration." https://www.infoq.com/articles/container-live-migration, 2016. [Online] Last accessed on 03-March-2019.

[3] S. Nadgowda, S. Suneja, and A. Kanso, "Comparing Scaling Methods for Linux Containers," in *IEEE International Conference on Cloud Engineering (IC2E)*, pp. 266–272, 2017.

[4] A. M. Mahfouz, M. L. Rahman, and S. G. Shiva, "Secure live virtual machine migration through runtime monitors," *2017 10th International Conference on Contemporary Computing, IC3 2017*, vol. 2018-Janua, no. August, pp. 1–5, 2018.

[5] G. J. Jeincy, R. S. Shaji, and J. P. Jayan, "A secure virtual machine migration using memory space prediction for cloud computing," *Proceedings of IEEE International Conference on Circuit, Power and Computing Technologies, ICCPCT 2016*, pp. 1–5, 2016.

[6] S. Sengole Merlin, N. M. Arunkumar, and M. A. Angela, "Automated Intelligent Systems for Secure Live Migration," *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2018*, no. Icicct, pp. 1360–1371, 2018.

[7] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live Service Migration in Mobile Edge Clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, 2018.

[8] V. Medina and J. M. Garcia, "A Survey of Migration Mechanisms of Virtual Machines," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–33, 2014.

[9] "Containers at Google." https://cloud.google.com/containers/, 2008. [Online] Last accessed on 24-February-2019.

[10] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A Comparative Study of Containers and Virtual Machines in Big Data Environment," *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2018-July, pp. 178–185, 2018.

[11] Á. Kovács, "Comparison of different linux containers," *2017 40th International Conference on Telecommunications and Signal Processing, TSP 2017*, vol. 2017-Janua, pp. 47–51, 2017.

[12] A. A. Mohallel, J. M. Bass, and A. Dehghantaha, "Experimenting with docker: Linux container and baseos attack surfaces," *International Conference on Information Society, i-Society 2016*, pp. 17–21, 2017.

[13] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013*, pp. 233–240, 2013.

[14] "OpenVZ Containers." https://openvz.org/Main, 2005. [Online] Last accessed on 01-April-2019.

[15] "Linux Containers." https://linuxcontainers.org/, 2013. [Online] Last accessed on 01-April-2019.

[16] "Docker Container Technology." https://www.docker.com/, 2013. [Online] Last accessed on 21-December-2018.

[17] P. E. N, F. J. P. Mulerickal, B. Paul, and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," *International Conference on Control Communication and Computing India*, 2015.

[18] U. Deshpande, D. Chan, T. Y. Guh, J. Edouard, K. Gopalan, and N. Bila, "Agile Live Migration of Virtual Machines," *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, pp. 1061–1070, 2016.

[19] "Checkpoint Restore in User Space." https://criu.org/Main_Page, 2012. [Online] Last accessed on 03-March-2019.

[20] Y. Chen, "Checkpoint and Restore of Micro-service in Docker Containers," no. Icmii, pp. 915–918, 2015.

[21] M. Azab, B. M. Mokhtar, A. S. Abed, and M. Eltoweissy, "Smart Moving Target Defense for Linux Container Resiliency," in *IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pp. 122–130, 2016.

[22] J. Zheng and A. S. Namin, "A Survey on the Moving Target Defense Strategies: An Architectural Perspective," *Journal of Computer Science and Technology*, pp. 207–233, 2019.

[23] T. Ylonen, "SSH - Secure Login Connections over the Internet," in *Proceedings of the 6th USENIX Security Symposium*, pp. 37–42, 1996.

[24] W. Stallings, "Protocol Basics: Secure Shell Protocol," in *The Internet Protocol Journal*, pp. 18–30, 2009.

[25] "Understanding the SSH Encryption and Connection Process." https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process, 2014. [Online] Last accessed on 10-March-2019.

[26] "National Institute of Standards Technology." https://www.nist.gov, 2001. [Online] Last accessed on 24-March-2019.

[27] M. Panda, "Performance analysis of encryption algorithms for security," *International Conference on Signal Processing, Communication, Power and Embedded System, SCOPES 2016 - Proceedings*, pp. 278–284, 2017.

[28] B. Thiyagarajan and R. Kamalakannan, "Data integrity and security in cloud environment using AES algorithm," *2014 International Conference on Information Communication and Embedded Systems, ICICES 2014*, no. 978, pp. 1–5, 2015.

[29] K. B.Adedeji and J. O. Famoriji, "Investigating the Effects of varying the Key Size on the Performance of AES Algorithm for Encryption of Data over a Com-

munication Channel," *International Journal of Applied Information Systems*, vol. 7, no. 8, pp. 6–10, 2014.

[30] M. Ahmed, B. Sanjabi, D. Aldiaz, A. Rezaei, and H. Omotunde, "Diffie-Hellman and Its Application in Security Protocols," *International Journal of Engineering Science and Innovative Technology (IJESIT)*, vol. 1, no. 2, pp. 69–73, 2012.

[31] "SSH File Transfer Protocol." https://www.ssh.com/ssh/sftp/. [Online] Last accessed on 30-April-2019.

[32] D. J. Barrett and R. Silverman, "The Secure Shell: The Definitive Guide," 2001.

[33] D. Blazhevski, "Modes of operation of the aes algorithm," *The 10th Conference for Informatics and Information Technology (CIIT 2013)*, no. Ciit, pp. 212–216, 2013.

[34] K. G. Paterson and G. J. Watson, "A Formal Security Treatment of SSH-CTR," *Advances in Cryptology — EUROCRYPT 2010*, vol. 216676, pp. 345–361, 2010.

[35] J. B. Princess, G. Jeba, L. Paulraj, and I. J. Jebadurai, "Methods to Mitigate Attacks during Live Migration of Virtual Machines – A Survey," *International Journal of Pure and Applied Mathematics*, vol. 118, no. 20, pp. 3663–3670, 2018.

[36] M. Aiash, G. Mapp, and O. Gemikonakli, "Secure live virtual machines migration: Issues and solutions," *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014*, no. March, pp. 160–165, 2014.

[37] W. Li and A. Kanso, "Comparing containers versus virtual machines for achieving high availability," *Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, pp. 353–358, 2015.

[38] J. Shetty, A. M R, and S. G, "A Survey on Techniques of Secure Live Migration of Virtual Machine," *International Journal of Computer Applications*, vol. 39, no. 12, pp. 34–39, 2012.

[39] F. Zhang and H. Chen, "Security-preserving live migration of virtual machines in the cloud," *Journal of Network and Systems Management*, vol. 21, no. 4, pp. 562–587, 2013.

[40] P. Fan, B. Zhao, Y. Shi, Z. Chen, and M. Ni, "An improved vTPM-VM live migration protocol," *Wuhan University Journal of Natural Sciences*, vol. 20, no. 6, pp. 512–520, 2015.

[41] X. Wan, X. Zhang, L. Chen, and J. Zhu, "An improved vTPM migration protocol based trusted channel," *2012 International Conference on Systems and Informatics, ICSAI 2012*, no. Icsai, pp. 870–875, 2012.

[42] A. Tamrakar, "Security in Live Migration of Virtual Machine with Automated Load Balancing," vol. 3, no. 12, pp. 806–811, 2014.

[43] S. Biedermann, M. Zittel, and S. Katzenbeisser, "Improving security of virtual machines during live migrations," pp. 352–357, 07 2013.

[44] H. Alshahrani, A. Alshehri, R. Alharthi, A. Alzahrani, D. Debnath, and H. Fu, "Live Migration of Virtual Machine in Cloud : Survey of Issues and Solutions," in *Int'l Conf. Security and Management | SAM'16 |*, pp. 280–285, 2016.

[45] K. Nagin, D. Hadas, Z. Dubitzky, A. Glikson, I. Loy, B. Rochwerger, and L. Schour, "Inter-cloud mobility of virtual machines," *ACM International Conference Proceeding Series*, 2011.

[46] Y. Chen, Q. Shen, P. Sun, Y. Li, Z. Chen, and S. Qing, "Reliable migration module in trusted cloud based on security level - Design and implementation," *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2012*, pp. 2230–2236, 2012.

[47] V. P. Patil and G. a. Patil, "Migrating Process and Virtual Machine in the Cloud: Load Balancing and Security Perspectives," *International Journal of Advanced Computer Science and Information Technology*, vol. 1, no. 1, pp. pp. 11–19, 2012.

[48] M. Aslam, C. Gehrmann, and M. Björkman, "Security and trust preserving VM migrations in public clouds," *Proc. of the 11th IEEE Int. Conference on Trust,*

Security and Privacy in Computing and Communications, TrustCom-2012 - 11th IEEE Int. Conference on Ubiquitous Computing and Communications, IUCC-2012, pp. 869–876, 2012.

[49] W. Li, A. Kanso, and A. Gherbi, "Leveraging Linux containers to achieve High Availability for cloud services," *Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, pp. 76–83, 2015.

[50] M. Azab, B. Mokhtar, A. S. Abed, and M. Eltoweissy, "Toward Smart Moving Target Defense for Linux Container Resiliency," *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pp. 619–622, 2016.

[51] "SSH Public Key Authentication." https://www.tecmint.com/ssh-passwordless-login-using-ssh-keygen-in-5-easy-steps/, 2015. [Online] Last accessed on 01-July-2019.

[52] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete Container State Migration," *Proceedings - International Conference on Distributed Computing Systems*, no. Section III, pp. 2137–2142, 2017.

[53] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient Live Migration of Edge Services Leveraging Container Layered Storage," *IEEE Transactions on Mobile Computing*, vol. PP, no. c, p. 1, 2018.