A Secure Model For Efficient Live Migration of Containers

Zeynep Mavuş and Pelin Angın*
Middle East Technical University, Ankara, Turkey
{e1670157, pangin}@ceng.metu.edu.tr

Received: August 3, 2019; Accepted: September 2, 2019; Published: September 30, 2019

Abstract

Cloud services have become increasingly widespread in the past decade due to their ability to reduce the complexity and the 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 method was used to generate 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 applications running on the Docker container platform demonstrate that the proposed approach achieves significantly better performance than its virtual machine live migration counterpart.

Keywords: Containers, live migration, security.

1 Introduction

Cloud computing, which is one of the most popular computing paradigms of this decade, transforms IT services to 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 cloud customers or users are not expected to pay for IT service infrastructure, buy the hardware and software resources (such as licenses), cloud computing provides significant benefits to the customers in many aspects like flexibility, portability and scalability.

While providing these benefits to the users, virtualization techniques are frequently utilized in the cloud computing field to enable users to share the infrastructure, which makes service provisioning less expensive. Without virtualization techniques, cloud providers would need to provide distinct resources to their customers, which is possible, but brings high costs as a significant disadvantage.

In order to tackle the high service cost, virtualization technologies such as virtual machines (VMs) and containers are used. In these cases, the whole performance of the cloud has a strong relation with the performance of the VMs and containers. Other than performance issues, security concerns for cloud

*Corresponding author: Pelin Angın, METU Department of Computer Engineering, Universiteler Mah. Dumlupinar Bulv. No:1 Ankara/Turkey, Email: pangin@ceng.metu.edu.tr, Tel: +90 3122105532 systems are also strongly correlated with the security concerns for the virtualized environments used in the cloud.

Containers are a new technology relative to VMs. Containerization technologies like OpenVZ [2], LXC [5] and Docker [4] provide a different way of virtualization compared to classic VMs due to their lightweight structure [36]. There are key differences between the two technologies: One difference is that while VMs imitate the operating system kernel, containers make use of 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 [22]. Therefore, containers have less virtualization cost and use less resources relative to VMs because of their lightweight nature [24]. Furthermore, since a container does not need its own operating system, it uses only the resources required for the application upon container start [21] [33]. Both of the virtualization techniques have capability to provide increased efficiency in the utilization of the resources in big data centers, which is achieved by the migration of the encapsulated service. Live migration has become an increasingly popular topic because of its contribution to the consolidation of services [8]. There are many other reasons to migrate a service from a source host to a destination. These include system maintenance (for a software or hardware update), load balancing, efficient resource utilization, service protection from attacks through moving target defense, etc. There are many studies in the literature on VM live migration as in [19], [16], [20]. In addition to those, migration management frameworks have also been developed for virtual machines, such as OpenStack, to be able to apply load balancing between virtual machines.

Although VM migration has been extensively investigated and discussed in the cloud computing arena, container migration does not have the same set of investigations in the literature, especially from a security point of view, because it can be considered as a relatively new technology. Live migration of containers can be vulnerable to many kinds of attacks such as eavesdropping, man-in-the-middle, denial of service (DoS) etc. The migration system should take precautions for these types of attacks.

In this paper, we propose a live migration model for containers, which provides security of migration data using secure authentication and data transfer protocols. The proposed model has been evaluated with stateful and stateless applications, and promising results have been achieved in terms of performance overhead of secure migration. The remainder of this paper is organized as follows: Section2 provides background information on live migration and a summary of existing techniques for live migration of VMs and containers. Section 3 explains the proposed secure live migration technique for containers. Section 4 provides the results of the experiments for secure live migration of sample application using the proposed model. Section 5 concludes the paper with future work directions.

2 Background and Related Work

2.1 Live Migration Attacks

Live migration of containers/VMs is susceptible to active and passive attacks. Active attacks cause loss of data integrity, whereas passive attacks cause the loss of sensitive data confidentiality. Some of the most remarkable attacks can be listed as man-in-the-middle, DoS, overflow and replay attacks[27].

- Man-in-the-Middle Attack: Attackers can eavesdrop on the data while migrating from source host to destination and modify data content, which could result in the loss of data integrity.

- Denial of Service (DoS) Attack: By using false resource advertisement, an attacker can attract more virtual machines towards a specific machine. This will result in migrating virtual machines, stealing the bandwidth resource by preventing the actual 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 processes.
- 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. [27]

2.2 Live Migration Security Factors

The factors that need to be achieved for making live migration secure are as follows:

- Access Control: Access control policies should be defined to ensure only users with granted privileges have control on the system [10].
- Authentication: Authentication is required between the source and the destination hosts for the migration process[10].
- Non-Repudiation: All the actions of both the source and destination hosts should be observed. While live migration is occurring, all activities should be logged[10].
- Data Confidentiality: Data encryption is required while migrating data between source and destination hosts[10].
- Communication Security: The data transmission channel should be protected on the migration path between source and destination hosts.
- Availability: The system should be protected against DoS attacks to make resources available for legitimate users.
- Privacy: The migration traffic is required to be isolated from the other networks in order to protect the system from man-in-the-middle and sniffing attacks[10].

2.3 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 [17]. Moreover, the security of the data is 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 as given below.

In [29] 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. [19]. In [35] the authors proposed a system using a Network Security Engine Hypervisor. In this architecture the physical resources are accessible only by the hypervisor [19]. In [15] the authors proposed a model utilizing the secure VM-vTPM protocol. In the case that a Trusted Platform Module is included in the hardware, a Virtual Trusted Platform Module (vTPM) might be implemented. In order to use in the migration traffic, an authentication protocol providing security is presented by the vTPM module [19]. In [32] 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.

In [30] the authors proposed a secure VM live migration model 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 [19]. In [25], the authors proposed a model for Inter Cloud Virtual Machine Mobility, which contains inter cloud agents to create 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 that are utilized by this mobility. An SSH tunnel is 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 [13] the authors separated the cloud domain into different sections with respect to security levels, where it is 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 into different areas. The fourth operation is migration waiting queue arrangement, which plans the VMs' migration petitions.

In [26] 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 perform authentication.

In [19] the authors proposed a system model for VM live migration 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 [19].

In [16] the authors proposed a system model for VM migration utilizing memory space prediction. In order to decrease the data loss caused by the attacks on memory space and to perform the migration of data in a secure manner, Virtual Machine Migration using Memory Space Prediction with compression method is defined and developed [16].

In [28] the authors proposed a system model using AES encryption during migration. Firstly, according to their requirements, resources are distributed to virtual machines. An under or overload is checked continuously 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, the AES encryption algorithm is used. As soon as the migration ends, the migrated data is decrypted and resource allocation is applied. This process continues in a cyclic manner [28].

2.4 Secure Live Migration of Containers

Due to the relative immaturity of the container technology, secure migration models for it are not as well-established as those of VMs. One model that stands out is the ESCAPE live migration model[12]. ESCAPE is inspired from the survival techniques in nature. This approach is based on the 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 achieve checkpointing of a running application, the model employs Docker containers 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 that 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 Linux container behaviors. Based on intrusion detection results, the migration process is initiated.

In the 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 utilizes the prey versus predator model to decide the next destination host to migrate the container to. 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, ESCAPE mounts the shared storage, which keeps the running container and apply 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, applying 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 conducts a risk assessment, which determines whether to migrate and where to migrate the running container to keep the victim away from the attacker[11].

2.5 Live Migration of Containers vs Virtual Machines

Machen et al. [18] proposed a layered framework for live migration of applications encapsulated either in containers or virtual machines. Experiments were conducted to make comparisons based on the results obtained when working with VMs and containers.

The framework aims at achieving good performance based on the need of frequent migrations 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 in order to take the advantage of the mobile edge cloud architecture. The proposed layered framework is compatible with the applications running in VMs and containers.

The authors separated the container state into different kinds of layers. These are the 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 the 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, the 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 improved performance, achieving decreased downtime. 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. An improved version of this technique uses three layers, which includes an application layer in addition. This separates the application from the instance layer into an additional application layer, which includes the idle version of the application and the data specific to the application. With the inserted application layer, the instance layer is only required to store the application's 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 layers.

Methods	**Virtualization**	**Checkpoint / Restore**	**Synchronization File Transfer**	**Security**
Proposed Model	Container	CRIU	SCP	✓
Layered Framework for LXC	Container	Complementary Tools	RSYNC	X
Layered Framework for KVM	Virtual Machine	Complementary Tools	RSYNC	X

Table 1: Comparison of Live Migration Methods

In the work of Machen et al.[18] the difference between VM and container migration is analyzed. Experiment results show that containers (LXC was used for the experiments) provide an advantage over VMs (KVM was used for the experiments) for the total migration time, application downtime and the data transferred from the source node to the destination while migrating. The main reason is 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 application.

Table 1 summarizes the differences between existing works and the model proposed in this paper, which is described in detail in Section 3.

3 Proposed Model

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

3.1 Model Architecture

This section describes the proposed model architecture. In our proposed model, there are five main components. Two of these are the source and destination instances. The remaining main components are an application server, a database (DB) server and the client interface. As shown in Figure 1, all components have secure connections established between them.

The application server behaves as the controller 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, no migration process can be initiated if the application server does not take action.

The application server creates the related SSH channels between the instances and itself in this model. In order to achieve that, it creates an SSH channel between itself and the host instance (instance-1) first. Then, it commands instance-1 to run the application on Docker, and start migration if requested. That is, it commands instance-1 to send related checkpoint files to instance-2 (destination instance) 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 instance-1 to connect it to the other instance by using the SSH command parameters.

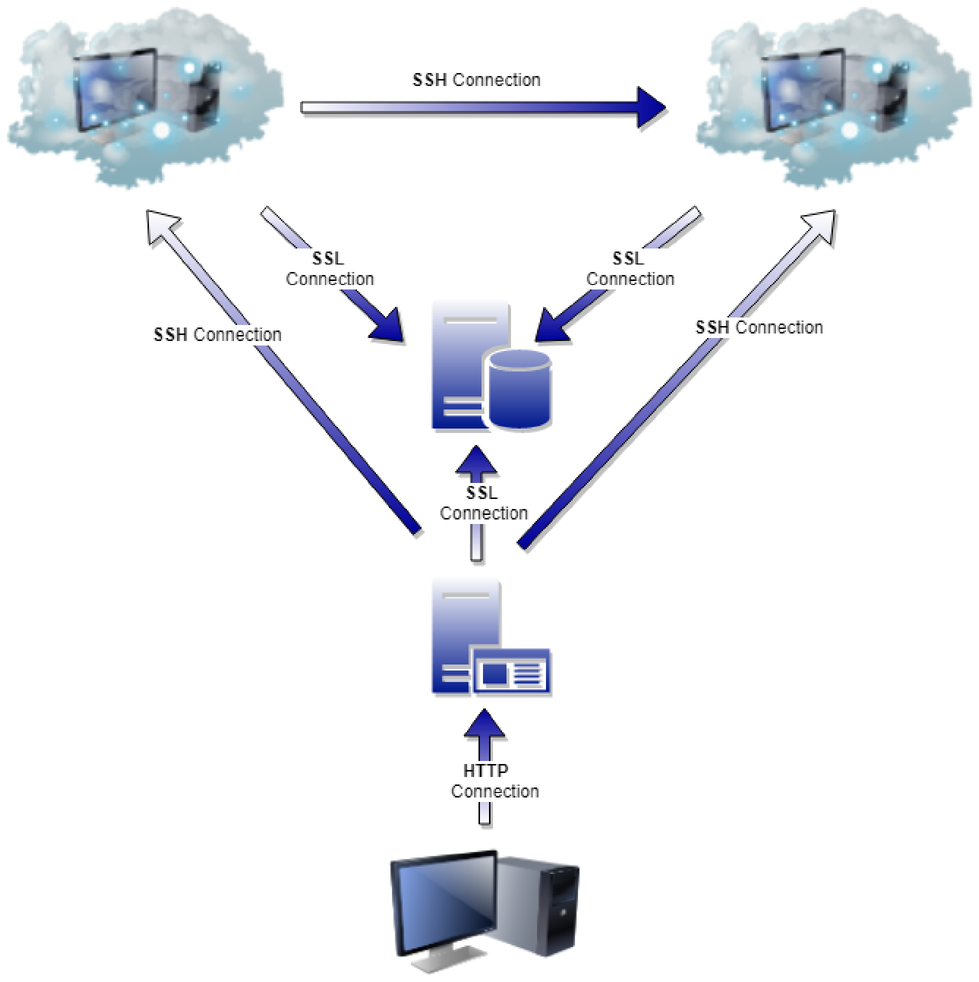


Figure 1: Model Architecture

An SSL connection is used for connecting to the DB server. The SSL connection can be set up to only allow connections from the specified IP addresses. By configuring SSL in this way, we make the DB server only allow the 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 the 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 2. Details of this process are explained in the following sections.

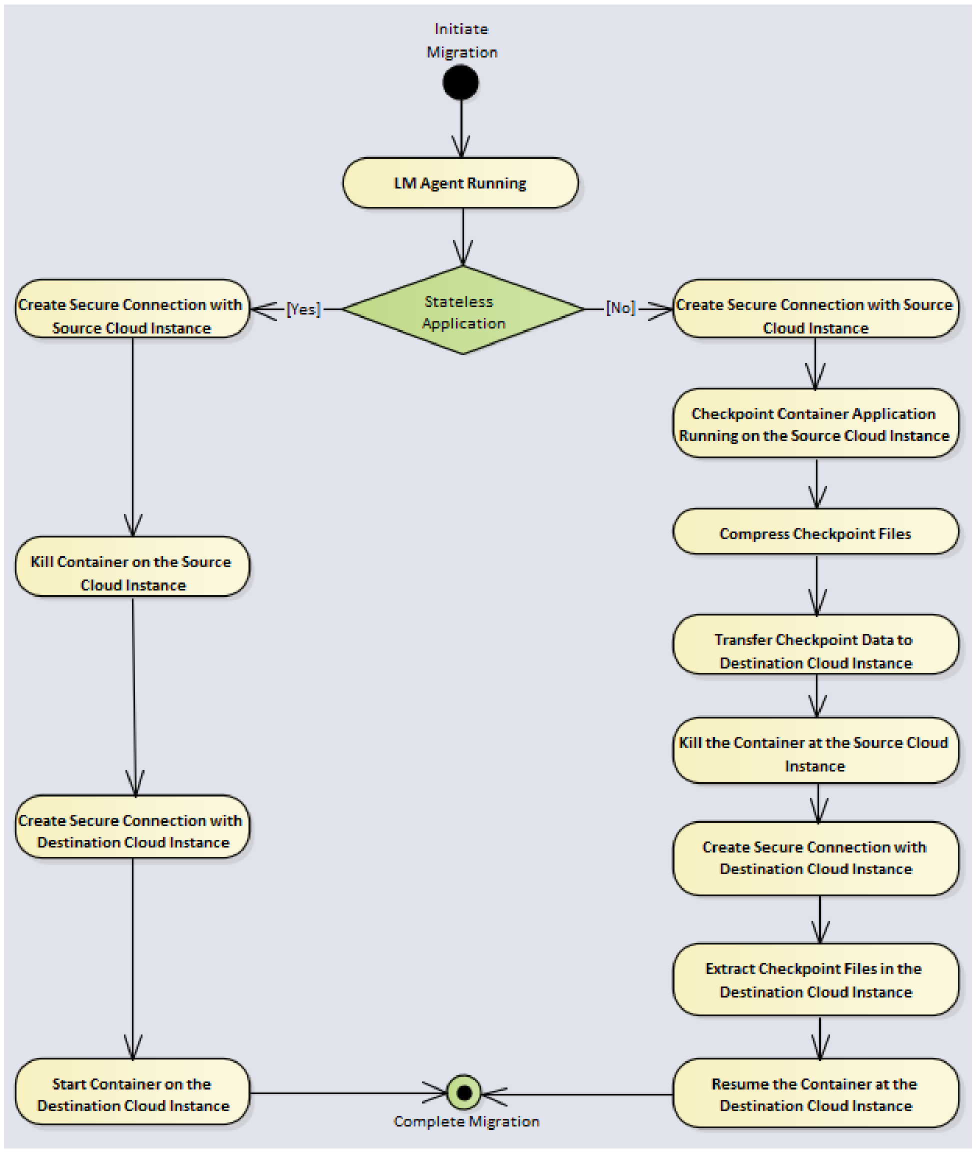

Initiate Migration
LM Agent Running
Stateless Application
[Yes]
[No]
Create Secure Connection with Source Cloud Instance
Kill Container on the Source Cloud Instance
Create Secure Connection with Destination Cloud Instance
Start Container on the Destination Cloud Instance
Create Secure Connection with Source Cloud Instance
Checkpoint Container Application Running on the Source Cloud Instance
Compress Checkpoint Files
Transfer Checkpoint Data to Destination Cloud Instance
Kill the Container at the Source Cloud Instance
Create Secure Connection with Destination Cloud Instance
Extract Checkpoint Files in the Destination Cloud Instance
Resume the Container at the Destination Cloud Instance
Complete Migration

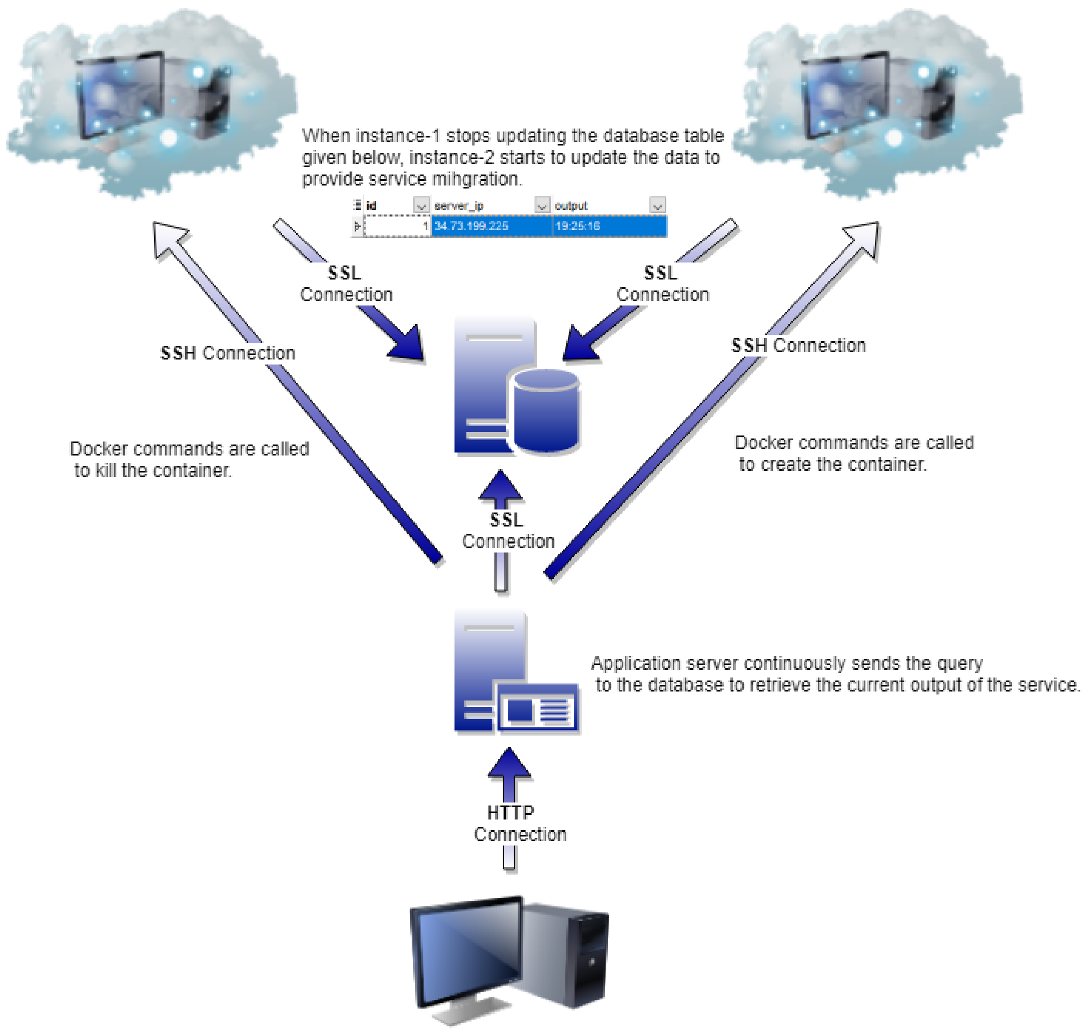
Figure 2: Proposed Model Activity Diagram

3.2 Model Execution Plan for Stateless Applications

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 logged 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 attack occurs on instance-1, the application server kills the container running the application in it by sending the relevant command to instance-1. Then, it runs the same application on instance-2, 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 that 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.



When instance-1 stops updating the database table given below, instance-2 starts to update the data to provide service mihgration.

id	server_ip	output
1	34.73.199.225	19:25:16

Docker commands are called to kill the container.

Docker commands are called to create the container.

Application server continuously sends the query to the database to retrieve the current output of the service.

Figure 3: Clock Application Execution Plan

3.3 Model Execution Plan for Stateful Applications

In our stateful application example, Face Recognition application, we give an image as input to the application. This application performs 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 change the model training set size by providing 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 and 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 a 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 located in the database server. This flag holds the information on whether the result set has been updated or not. The application server waits until the flag indicates that execution has 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.

3.4 Satisfied Security Factors

The described container migration model achieves the security factors discussed in Section 2 as follows:

Access control, which requires that unauthorized users are not allowed to start, move and stop a machine, is achieved with this model. In our case, the rights to perform these operations belong to the application server, which acts like the control manager.

Authentication is achieved, because all the machines in the system authenticate to each other in order to perform the required operations remotely. Instead of password-based authentication, we used public key authentication with the help of strong key exchange and symmetric encryption algorithms.

Non-repudiation is also satisfied. The application server observes all the activities and handles the exceptions. All the operations executing remotely are logged to the log files in the application server.

Data confidentiality is provided, too. While migrating, we use an SSH channel and the SFTP file transfer protocol to guarantee data encryption between the source and the destination. This also enables us to protect the migration process from man-in-the-middle attacks.

Communication security is provided, as the data transmission channel is protected via the SSH protocol. Data integrity is also provided, as the SSH channel utilizes the available MAC algorithms for integrity checking.

Availability is provided, as unauthorized users are not allowed to initiate a migration process, which enables the system to prevent DoS attacks initiating unnecessary outgoing migrations in an increasing manner, making the system resources unavailable to the authorized users.

The overflow attack is avoided, because the transmission channel is protected by the SSH protocol, where attackers that are unauthorized users (who cannot perform public key authentication) cannot cause congestion for the communication channel traffic.

The replay attack is achieved by re-transmitting the replicates of the dirty pages in a sequential manner. In our scenario, only the checkpoint folder in a compressed file format is transmitted from source to destination once for each migration. Besides, data integrity is satisfied by the SSH protocol, which results in avoiding replay attacks.

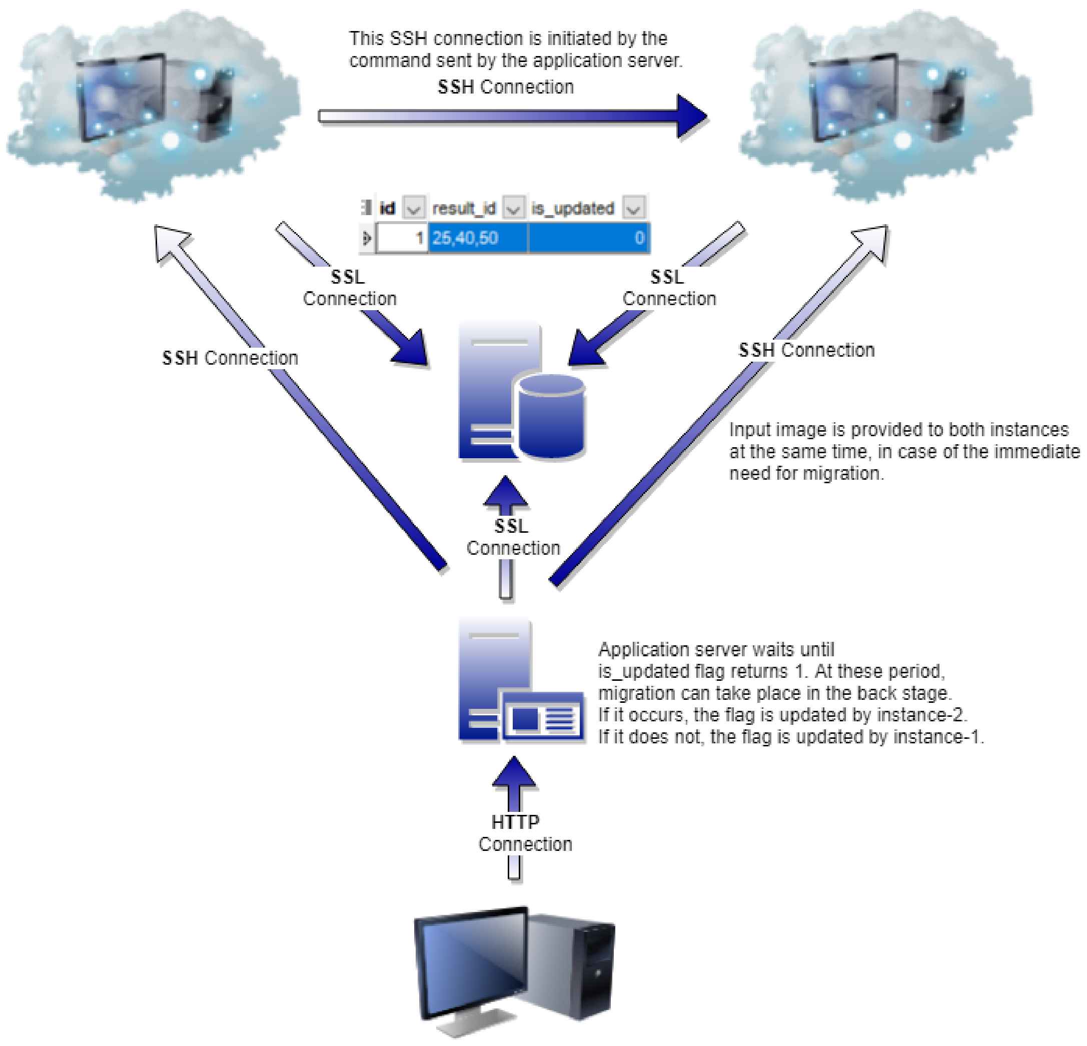

This SSH connection is initiated by the command sent by the application server.
SSH Connection
id
result_id
is_updated
1
25,40,50
0
SSL Connection
SSL Connection
SSH Connection
SSH Connection
Input image is provided to both instances at the same time, in case of the immediate need for migration.
SSL Connection
Application server waits until is_updated flag returns 1. At these period, migration can take place in the back stage. If it occurs, the flag is updated by instance-2. If it does not, the flag is updated by instance-1.
HTTP Connection

Figure 4: Face Recognition Application Execution Plan

4 Experiments

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

4.1 Experiment Setup

The experimental setup includes two identical Google Cloud [3] instances. One of them is the destination node, and the other one is the source node. The Checkpoint/Restore application (CRIU) runs on a Dockerized image. This image is based on an Ubuntu distribution. The details of cloud instance configurations are shown in Figure 5.

Throughout the document, these identical instances are named as instance1 and instance2, which are the source and destination machines on which the migration process is performed. The configuration details of the application server and database server are provided in Figure 6 and Figure 7.

PostGreSQL 10.8 is installed on the database server in order to execute the SQL commands sent by

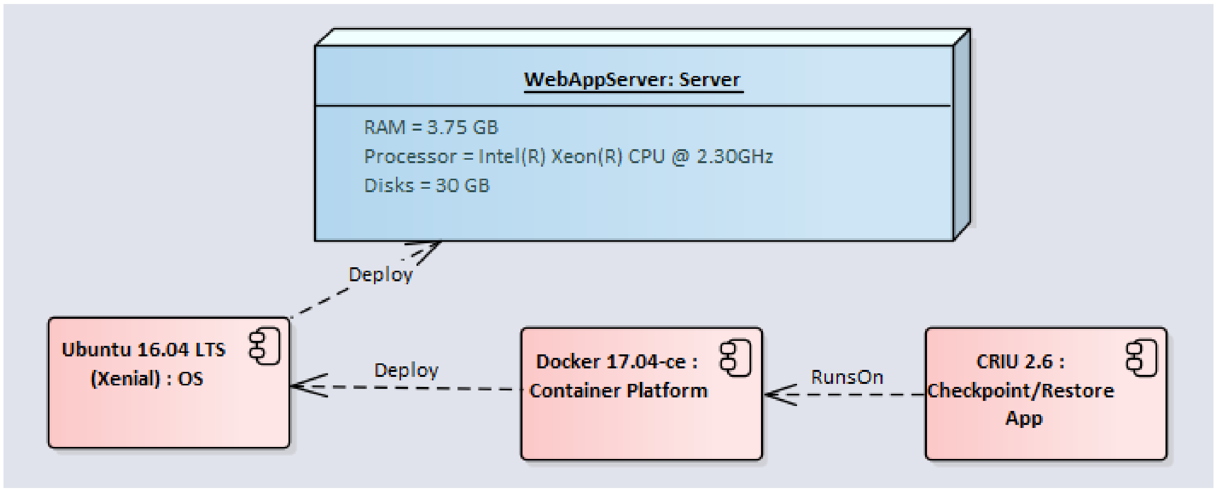

WebAppServer: Server
RAM = 3.75 GB
Processor = Intel(R) Xeon(R) CPU @ 2.30GHz
Disks = 30 GB
Deploy
Ubuntu 16.04 LTS (Xenial) : OS
Deploy
Docker 17.04-ce : Container Platform
RunsOn
CRIU 2.6 : Checkpoint/Restore App

Figure 5: Cloud Instance Details

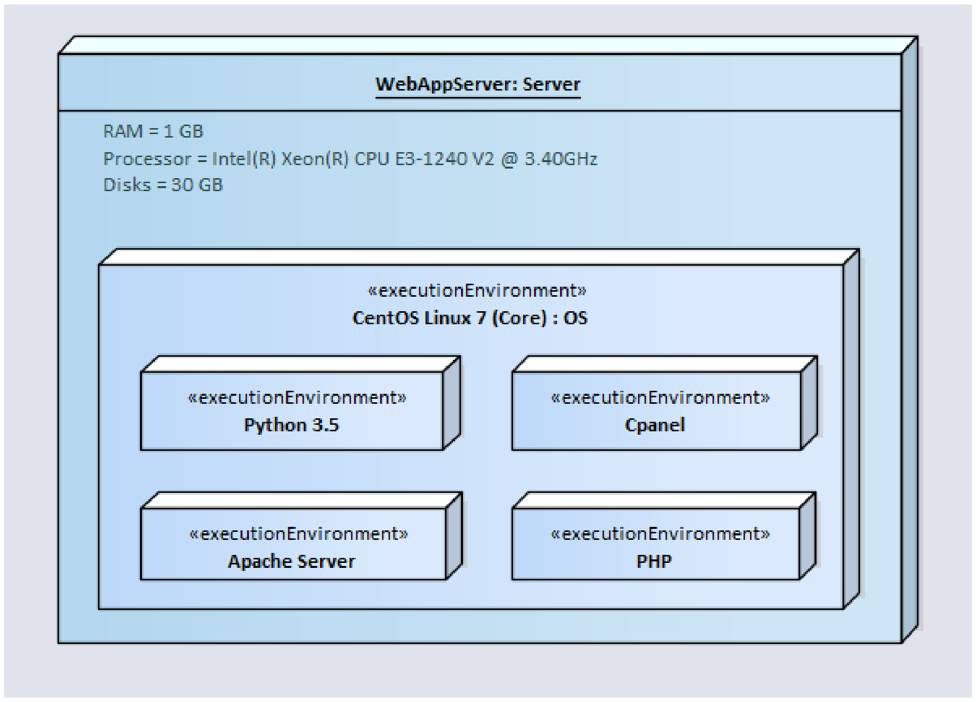


Figure 6: Application Server Details

instances. In order to connect to the DB, we use the SSL protocol. PostgreSQL provides the ability to give access to specific machines. Therefore, we specify the IPs of the instances and application server in the PostgreSQL configuration files. This means that no machine other than cloud instances and the application server can connect to the database to execute any query.

4.2 Applications Used in Experiments

Container live migration systems need to be analyzed with two kinds of applications: stateless and stateful, as the system reaction can change based on the application type. For stateless applications, if

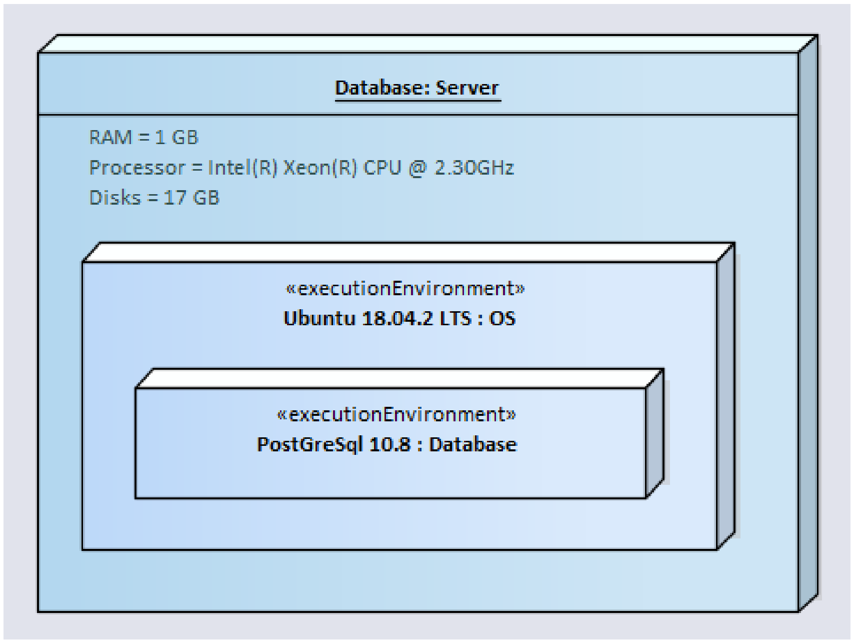
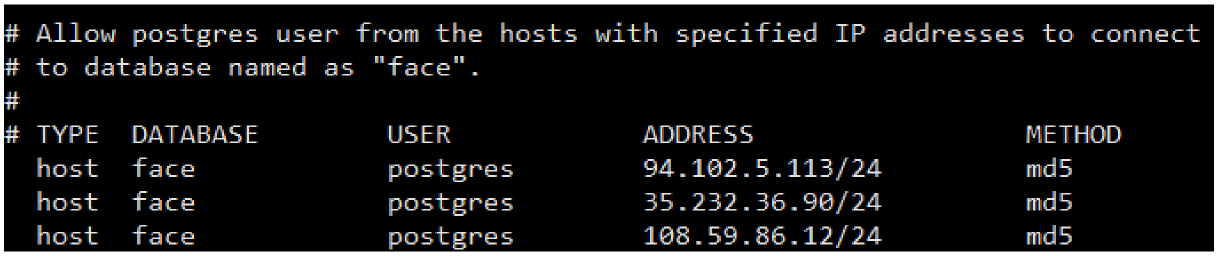


Figure 7: Database Server Details



```
# 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 8: SSL Connection Configuration

an attack is detected, the container is migrated to a new host without the 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 machines.

Face Recognition Application For the stateful application example, a face recognition application is used. In order to recognize faces, it uses the dlib library, which utilizes deep learning. One of the features of this application is face detection, which means that it detects all the people's faces seen on a photo. The other feature is to identify the people's faces that appear on a photo.

SSH Connection and File Transfer Application An application was developed in Python 3.3+ to establish an SSH [6] connection between cloud instances and between an instance and the application server. User inputs are forwarded from the source to the destination machine by using this secure channel.

Also, this application provides transfer of checkpoint files with SFTP [1]. In order to do that, SSH is configured to provide remote login in the related machine. This application uses SSH Public Key Authentication [7] to achieve connectivity between source and destination during the live migration process in a secure way. The communication encryption and key exchange algorithms are handled by the SSH server and client programs [34]. 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. For SSH communication encryption, AES algorithm in counter mode with a 256-bit key size is used [31]. For the key exchange process, the Diffie Hellman Key Exchange algorithm is used with SHA-256 for hashing [14] [9].

4.3 Experiment Metrics

In this section, the performance of the proposed model is evaluated 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, the service becomes unavailable to the user in the background. Because of that, this is an important metric to decide the 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 CRIU. It varies according to the application state and memory used at that time.
- **Upload speed** has a great impact on file transfer duration, which means it also has an impact on the downtime metric.
- **Checkpoint duration** refers to the time frame required to take the application snapshot at a specific time of the execution.
- **Resume duration** refers to the time frame required to resume an application at the destination node from where it stopped on the source node.

4.4 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 this 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, therefore, checkpoint related metrics are not applicable. For the stateful applications, all of the metrics given above are applicable.

The results obtained in the first step of the experiment show that the above metrics have a strong relation. Total migration time basically depends of the file size to be transferred during migration. In addition to that, the migration process consumes system resources like network bandwidth, CPU time etc. Unless there are sufficient resources on the source machine, the migration time is influenced negatively from these parameters. Although the checkpoint file is not considered as too large to transfer between nodes, the migration time can increase because of low bandwidth. The first step was repeated 10 times by using the Python logging library, log files were 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. Experiment results for the Face Recognition Application are shown in Figure 9, which shows both the total migration time and downtime of containers.

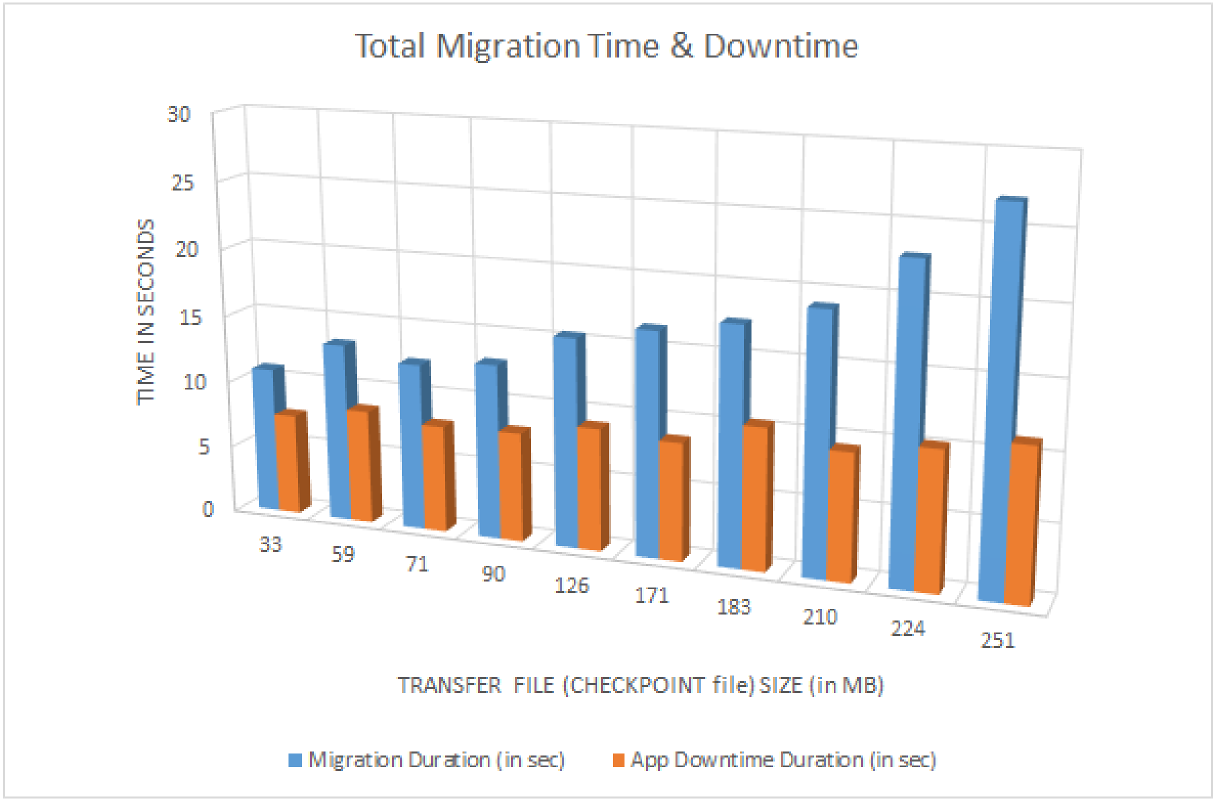


Figure 9: Total Migration Time and Downtime

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 the file upload speed in the migration channel between the source and the destination nodes. Figure 9 shows that both of the total migration time and downtime increase when the transfer file size increases. However, it does not increase linearly because of the non-linear network bandwidth between the source and the destination nodes.

Checkpoint and resume operations are also part of the total migration time. Relative to file transfer's effect on total migration time, checkpoint and resume durations have a smaller effect when migrating one container from source to destination host as given in Figure 10.

For the stateless application, the duration for killing the container on the source host, starting the container on the destination host, downtime and total migration time metrics are the only applicable ones. Downtime refers to the time frame between killing the container on the source and starting the container on the destination. Total migration duration refers to all the migration preparation operations such as establishing a 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

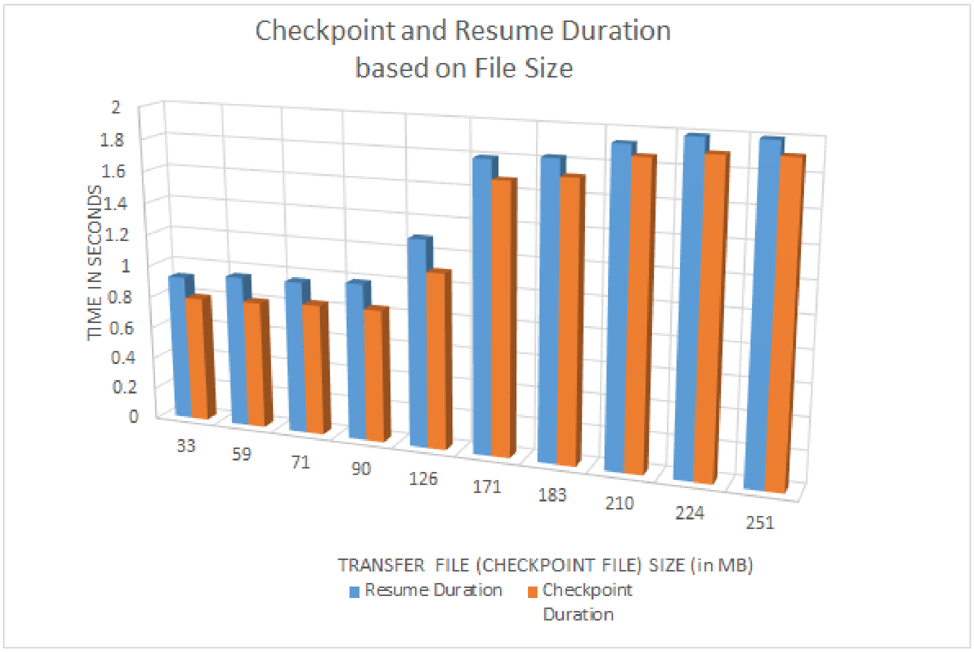


Figure 10: Checkpoint and Resume Duration

experiment was performed ten times and the results are provided in Table 2

Kill Duration (sec)	Start Duration (sec)	Downtime (sec)	Total Migration Duration (sec)
0.711551905	0.563214064	2.454	3.289
0.57733798	0.55944109	1.87	2.577
0.650537014	0.559459925	2.426	3.097
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
0.665262938	0.55846405	2.423	3.23

Table 2: Stateless Clock Application Experiment Results

The stages of the application are given in Figure 11. Kill duration, start duration and downtime are the stages of the application migration. According to the graph the biggest part of the migration time frame belongs to application downtime, which is the time between killing the container on the source host and starting the new container on the destination host.

4.4.1 Multi User Test Scenario Results

For the stateless application example (Clock Application) in our experiment, 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

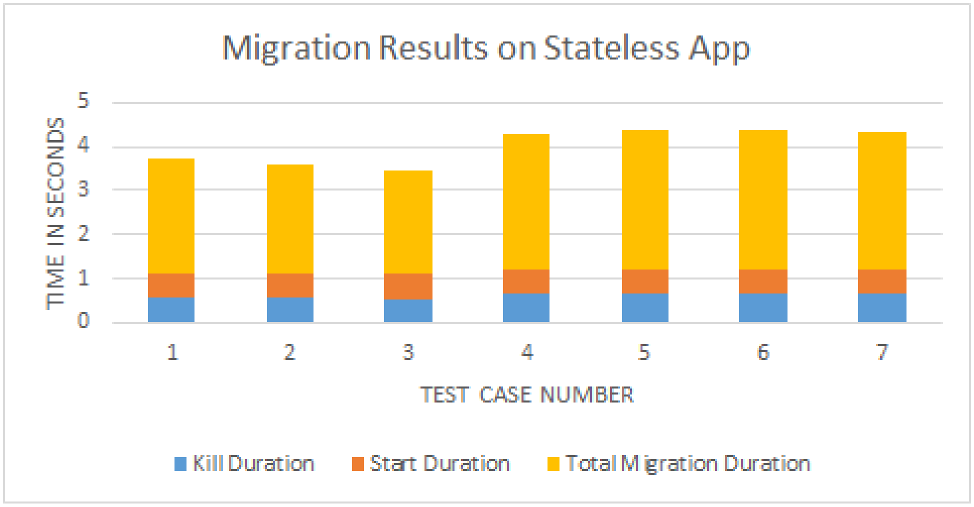


Figure 11: Stateless Clock Application Experiment Results

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 for 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. Experiments are conducted with 1 CPU & 3.75 GB memory configuration. It is observed that the average upload speed for the multi-user test case is 21.15 MB/sec. Experiment results are shown in Figure 12.

For the Face Recognition application, total migration time change according to file size and container count are given in Figure 12. The file size reaches gigabytes, and the file transfer process, which includes encryption and decryption at the source and destination, relatively increases the total migration time.

In Figure 13, we observed that the resume operation takes much longer time compared to the checkpoint operation, when we increase the number of containers. Because 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 because of that reason, which results in a long resume duration relative to checkpointing.

4.4.2 Comparison with Layered Framework Experiments

The experiments conducted in [18] to analyze the live migration performance of the layered framework proposed by the authors were performed with the following applications:

- Game Server Application: This is a server for an online game, which is used to transmit/receive messages according to user position.

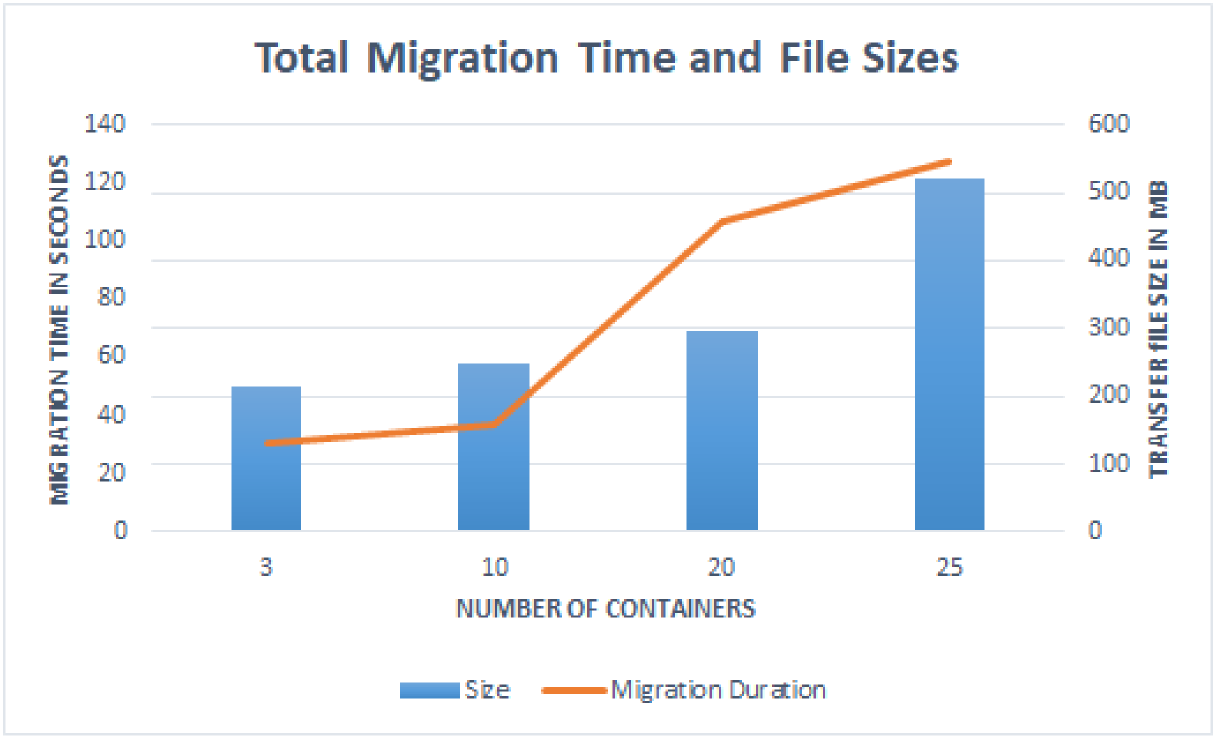


Figure 12: Multi-User Total Migration Time and Checkpoint File Sizes

- RAM Simulation Application: In order to simulate memory consumer applications, a script application is used, which is 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 50 MB in the mobile edge cloud together with the application.
- Face Detection Application: This is an application used to detect faces in a video.
- No Application: This represents the operating system with no additional applications 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 Table 3:

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

Table 3: 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 machines were used as users. Each virtual machine standing

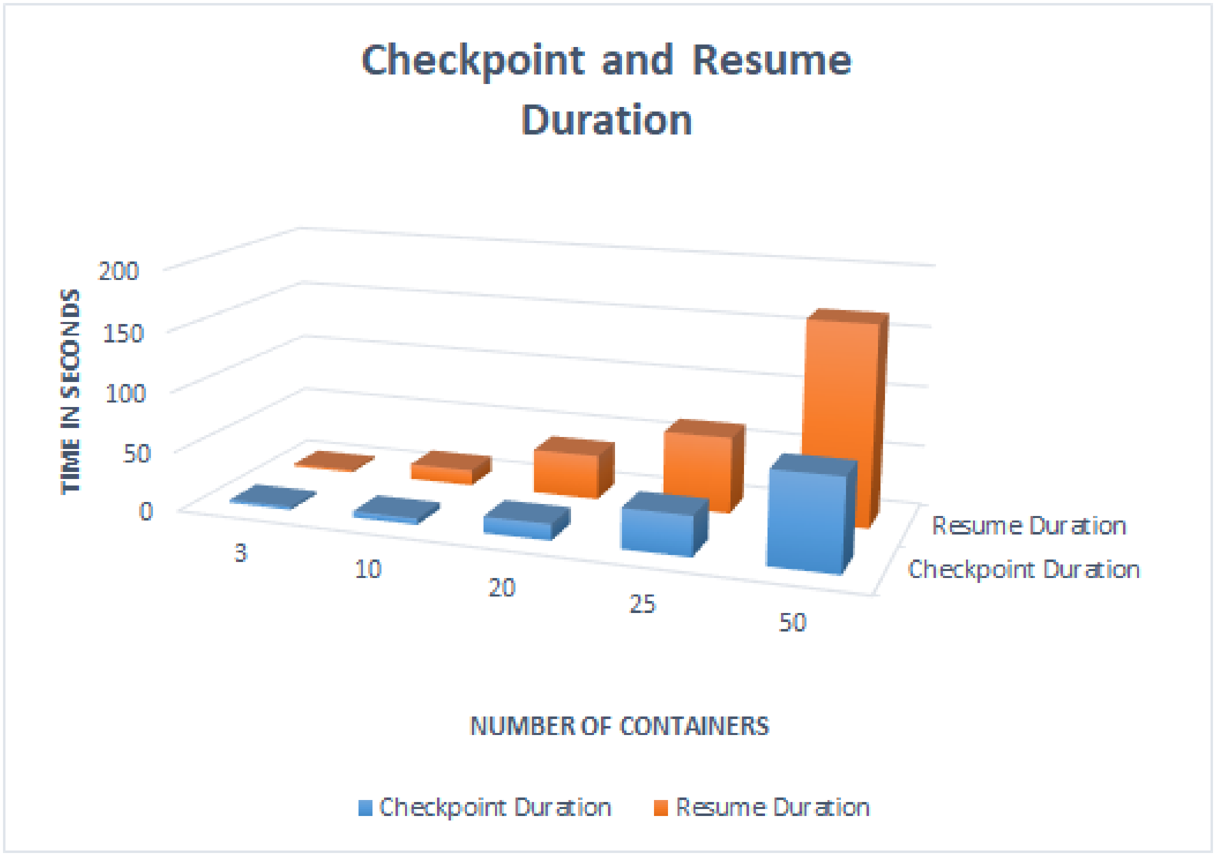


Figure 13: Multi-User Checkpoint and Resume Duration

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 those experiments, the rsync command was used for iterative file system synchronization. Rsync is a command for copying and synchronizing files/folders between remote machines 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 with the rsync command on its own unless a secure channel is used between the machines.

	LXC Total Data Tranfer 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 4: Total Migration Duration and Transfer Data Size for LXC

The experiment results as shown in Table 4 and Table 5 are obtained by computing the mean value of the ten distinct executions.

Based on the experiment results, the authors state that containers' performance is better than VMs'.

	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: Total Migration Duration and Transfer Data Size for KVM

The reason is that containers are lightweight compared to virtual machines, which means that the file system and the memory context is related to the running application. On the other side, VMs can have much more unrelated information to the application [18].

When we compare the given results with the results of our performance analysis experiment results, we can support the idea provided by the authors of [18], which states container performance is better than VM performance when live migrating the application. Although we have run our experiments with instances having 1 vCPU and changing link bandwidth, we obtained better results especially compared to KVM migration performance. Our architecture 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 column of the results given in Table 4 and Table 5. 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 transfer data size close to 99 MB is migrated within approximately 13 seconds on average with 36 MBps bandwidth, which is much less than the KVM migration result. In order to make bandwidth values of the experiments same, we can multiply the bandwidth of 12.5 MBps with 3, which means that it will migrate 297 MB transfer data in that time frame, 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 a 107.5 sec downtime is given in Table 5. Even for transfer data of size 250 MB, our model downtime is measured as 11.127 seconds.

When we compare the results with the results given in Table 4, we see that the migration data size is small with respect to our model. Although the size of the transfer data is smaller than the transfer data obtained in our model, the total migration time takes longer with LXC. For instance, in 15.5-seconds time frame, transfer data with size 10.0 MB can be migrated with 100 Mbps (12.5 MBps) bandwidth. In our architecture, in that time frame, we could migrate approximately 126 MB of transfer data with 35 MBps bandwidth. In order to make bandwidth values of the experiments same, we can multiply the bandwidth with 12.5 MBps with 3, it means that it will migrate 30.0 MB transfer data in that time frame, which is still less than the our 126 MB result obtained in our test case.

In the layered framework, security issues are not mentioned, whereas our model's main aim is to achieve secure live migration. By using 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.

5 Conclusion

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

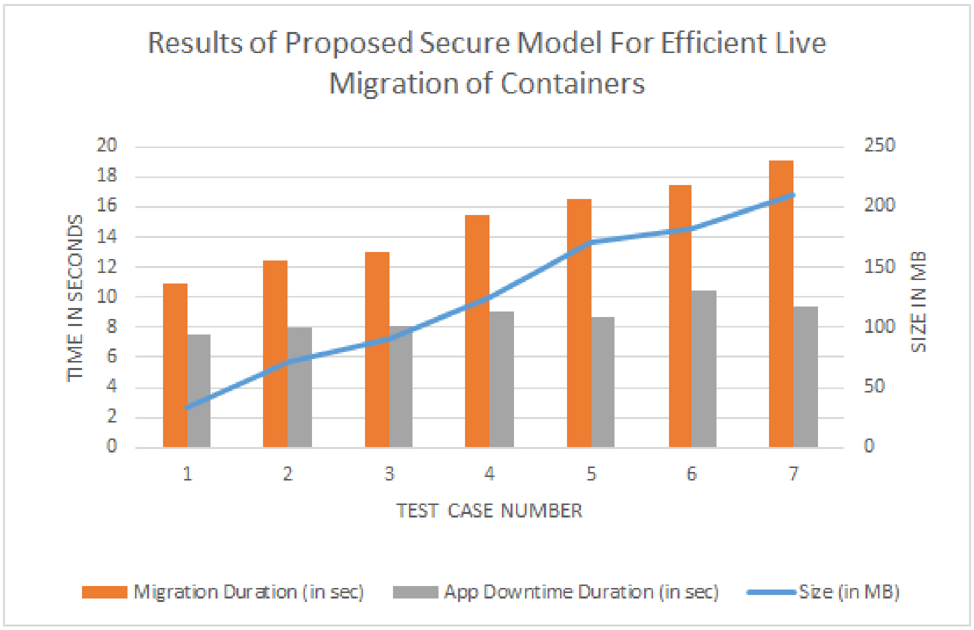


Figure 14: Proposed Model Results with Face Recognition Application

of the users. Because the cloud services diminish the cost and complexity of resource management, it has become preferred by users and the organizations. Cloud services as applications running in the cloud are encapsulated in virtualized environments in order to lower the resource costs. Virtual machines and containers are utilized to create a virtual environment for cloud services. In order to achieve effective resource utilization, live migration should 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 contributions to efficient resource utilization, live migration also plays an important role for system maintenance, load balancing and applying moving target defense techniques to make services avoid attacks. Based on the small occupied space in memory and time frame required to start up a service fast, the container technology has become a popular alternative for VMs [23]. While there are many studies in the literature on VM migration issues, secure container migration is a relatively new topic that need further exploration.

In this work, we propose a model to apply live migration on containers in a secure manner. Docker containers were used to implement the model and conduct the experiments in this study. By using Docker containers, faster and more lightweight migration is possible. In the proposed model, the checkpointing approach is used to take the snapshot of the running application and resuming functionality on Docker container services. Via this method, we achieved decreased transfer file size, which affects model efficiency and the time required to transfer data in a secure way, over an encrypted channel. 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.

As future work, optimizations on the multi-user scenario should be applied. 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 a control manager for the migration process. In order to avoid single point of failure, the application server can be distributed into multiple application servers.

References

[1] "SSH File Transfer Protocol," https://www.ssh.com/ssh/sftp/, [Online; accessed on September 2, 2019].

[2] "OpenVZ Containers," https://openvz.org/Main, [Online; accessed on September 2, 2019], 2005.

[3] "Containers at Google," https://cloud.google.com/containers/, [Online; accessed on September 2, 2019], 2008.

[4] "Docker Container Technology," https://www.docker.com/, [Online; accessed on September 2, 2019], 2013.

[5] "Linux Containers," https://linuxcontainers.org/, [Online; accessed on September 2, 2019], 2013.

[6] "Understanding the SSH Encryption and Connection Process," https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process, [Online; accessed on September 2, 2019], 2014.

[7] "SSH Public Key Authentication," https://www.tecmint.com/ssh-passwordless-login-using-ssh-keygen-in-5-easy-steps/, [Online; accessed on September 2, 2019], 2015.

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

[9] 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*, vol. 1, no. 2, pp. 69–73, 2012.

[10] M. Aiash, G. Mapp, and O. Gemikonakli, "Secure live virtual machines migration: Issues and solutions," in *Proc. of the 28th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA'14), Victoria, British Columbia, Canada*, no. March. IEEE, 2014, pp. 160–165.

[11] M. Azab, B. Mokhtar, A. S. Abed, and M. Eltoweissy, "Toward Smart Moving Target Defense for Linux Container Resiliency," in *Proc. of the 41st IEEE Conference on Local Computer Networks (LCN'16), Dubai, United Arab Emirates*. IEEE, November 2016, pp. 619–622.

[12] M. Azab, B. M. Mokhtar, A. S. Abed, and M. Eltoweissy, "Smart Moving Target Defense for Linux Container Resiliency," in *Proc. of the 2nd IEEE International Conference on Collaboration and Internet Computing (CIC'16), Pittsburgh, Pennsylvania, USA*. IEEE, November 2016, pp. 122–130.

[13] 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," in *Proc. of the 26th IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'12), Shanghai, China*. IEEE, May 2012, pp. 2230–2236.

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

[15] 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, December 2015.

[16] G. J. Jeincy, R. S. Shaji, and J. P. Jayan, "A secure virtual machine migration using memory space prediction for cloud computing," in *Proc. of the 2016 IEEE International Conference on Circuit, Power and Computing Technologies (ICCPCT'16), Nagercoil, India*. IEEE, March 2016, pp. 1–5.

[17] W. Li and A. Kanso, "Comparing containers versus virtual machines for achieving high availability," in *Proc. of the 2015 IEEE International Conference on Cloud Engineering (IC2E'15), Tempe, Arizona, USA*. IEEE, March 2015, pp. 353–358.

[18] 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, February 2018.

[19] 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.

[20] 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, January 2014.

[21] A. A. Mohallel, J. M. Bass, and A. Dehghantaha, "Experimenting with docker: Linux container and baseos attack surfaces," in *Proc. of the 2016 International Conference on Information Society (i-Society'16), Dublin, Ireland*. Infonomics Society, October 2017, pp. 17–21.

[22] P. E. N, F. J. P. Mulerickal, B. Paul, and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," in *Proc. of the 2015 International Conference on Control Communication and Computing India (ICCC'15), Trivandrum, India*. IEEE, November 2015, pp. 697–700.

[23] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete Container State Migration," in *Proc. of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS'17), Atlanta, Georgia, USA*. IEEE, June 2017, pp. 2137–2142.

[24] S. Nadgowda, S. Suneja, and A. Kanso, "Comparing Scaling Methods for Linux Containers," in *Proc. of the 2017 IEEE International Conference on Cloud Engineering (IC2E'17), Vancouver, British Columbia, Canada*. IEEE, April 2017, pp. 266–272.

[25] K. Nagin, D. Hadas, Z. Dubitzky, A. Glikson, I. Loy, B. Rochwerger, and L. Schour, "Inter-cloud mobility of virtual machines," in *Proc. of the 4th Annual International Conference on Systems and Storage (SYSTOR'11), Haifa, Israel*. ACM, May 2011, pp. 1–12.

[26] 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. 11–19, 2012.

[27] 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.

[28] S. Sengole Merlin, N. M. Arunkumar, and M. A. Angela, "Automated Intelligent Systems for Secure Live Migration," in *Proc. of the 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT'18), Coimbatore, India*. IEEE, April 2018, pp. 1360–1371.

[29] 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.

[30] A. Tamrakar, "Security in Live Migration of Virtual Machine with Automated Load Balancing," *International Journal of Engineering Research & Techonology*, vol. 3, no. 12, pp. 806–811, 2014.

[31] B. Thiyagarajan and R. Kamalakannan, "Data integrity and security in cloud environment using AES algorithm," in *Proc. of the 2014 International Conference on Information Communication and Embedded Systems (ICICES'14), Chennai, India*. IEEE, February 2014, pp. 1–5.

[32] X. Wan, X. Zhang, L. Chen, and J. Zhu, "An improved vTPM migration protocol based trusted channel," in *Proc. of the 2012 International Conference on Systems and Informatics (ICSAI'12), Yantai, China*. IEEE, May 2012, pp. 870–875.

[33] 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," in *Proc. of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'13), Belfast, UK*. IEEE, February 2013, pp. 233–240.

[34] T. Ylonen, "SSH - Secure Login Connections over the Internet," in *Proc. of the 6th USENIX Security Symposium (USENIX Security'19), San Jose, California, USA*, July 1996, pp. 37–42.

[35] 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, December 2013.

[36] 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," in *Proc. of the 11th IEEE International Conference on Cloud Computing (CLOUD'18), San Francisco, California, USA*, vol. 2018-July. IEEE, 2018, pp. 178–185.

Author Biography

Zeynep Mavuş received the bachelor's degree in computer engineering in 2013 and the M.S. degree in computer engineering in 2019, both from Middle East Technical University, Turkey. She is currently a software engineer at Aselsan in Ankara, Turkey. Her research interests include cloud computing, virtualization technologies and machine learning.

Pelin Angın received the B.S. degree in computer engineering from Bilkent University, Turkey, and the Ph.D. degree in computer science from Purdue University, West Lafayette, IN, USA. She was a Visiting Assistant Professor (2014–2015) at Purdue University, and then a Postdoctoral Research Associate, until 2016. She is currently pursuing her academic career with Middle East Technical University, Ankara, Turkey, as an Assistant Professor of computer engineering. She is affiliated with the S2RL and WINS research laboratories. Her research interests lie in the fields of distributed systems, cloud computing, and IoT security.