Inter-Cloud application migration and portability using Linux containers for better resource provisioning and interoperability

Ivin polo sony

Submitted as part of the requirements for the degree
of MSc in Cloud Computing
at the School of Computing,
National College of Ireland
Dublin, Ireland.

September 2015

Supervisor Dr Horacio gonzalez-velez

Abstract

In a cloud computing environment, availability, efficient resource utilization, fault tolerance and feasible resource provisioning is of primary concern. There are certain situations in which a virtual machine is running on top of a physical host and needs to be migrated for various reasons such as maintenance or lack of hardware resource availability. Live application migration is one of the widely used technique to address these concerns and this is achieved by migrating virtual machines over a network to another physical host.But migrating a virtual machine over a network uses up the bandwidth because along with the application an operating system is also transferred which adds as an overhead along with the application.A Linux container on the other hand is much more lighter since it uses the underlying operating system making it a prominent candidate for application migration. In this research paper application migration is done by migrating a Linux container over a network by considering the following factors: 1) Availability of the underlying network. 2) Consideration of total changes in the while migration 3) Availability of compute time in the destination host. The paper also sheds a light into the possibility of migrating a work load easily without thinking of the huge cost and time associated with migration of application among nodes.

Contents

List of Figures

Chapter 1

Introduction

Provisioning of available resources in a cloud environment has become extremely important. The cloud resource management strategies are mainly based on the monitoring of service requests and intelligent scheduling of processes [3]. Cloud computing is so popular because of the fact that the resources available to a particular machine are not isolated.Virtualization makes it possible to divide and share all kinds of resources in a distributed environment. The restriction that was there regarding the hardware availability of resources before deploying an application is removed because of the virtualized environment which removes the hardware dependencies of the application. Virtualization is done with the help of hypervisors or Virtual Machine monitor(VMM) such as Xen[12] which loads the kernel and other dependencies associated with a particular guest operating system [2]. There are certain situations in which a virtual machine is running on top of a physical host and needs to be migrated for various reasons such as maintenance or lack of hardware resource availability. There are different ways of achieving this like suspending the virtual machine and transferring it to a different host, but when the virtual machine is running a critical application, if it needs to be transferred without shutting it down, that is when you use Live virtual machine migration [2]. There are different scenarios in which virtual live migration really has advantages, but in certain cases when the size of the Virtual Machine is large, for example, if the Virtual Machine contains a MySQL database then this approach is expensive. The Cloud infrastructure came into existence mainly on the fact that available resources can be shared seamlessly between Virtual machines [3].

Cloud computing has allowed us to seamlessly share available resources among tenants. Despite the success of cloud computing , issues and limitations still exists. Lack of portability and interoperability among cloud platforms is an issue which is faced by the whole cloud computing industry. Some of the issues such as proprietary protocols and

formats are platform dependent, thus making it isolated to a particular platform. As discussed in the previous paragraph, the importance of resource sharing and migration of workloads in a cloud environment. If interoperability and portability can be achieved among cloud platforms, it not only would help remove vendor lock-in but also remove flaws of a particular cloud platform. A seamless migration of the state of a particular application form one cloud platform to another makes it possible to achieve portability and interoperability among cloud platforms. In the following chapters, various techniques of migration and isolation of applications are explored and migration of the state of various running applications using linux container are being implemented.

Chapter 2

Background

The research problem that is going to be discussed for the literature review is Inter-Cloud application migration and portability using Linux containers for better resource provisioning and interoperability" .

The literature review is organized into the following headings: Migration and resource management,Application Migration by process migration and Operating System Virtualization and Linux Containers. Migration and resource management section talks about the importance of resource provisioning in the cloud ecosystem and strategies that are being used to achieve it such as live VM migration. Furthermore, a comparison of live VM migration and application level migration is done and we have discussed different strategies that are used in live VM migration.

Application migration by process migration looks at application migration from the perspective of process migration.Process migration is a much more mature method than VM migration hosted in a hypervisor . This section explains the problems and complications that could occur due to process migration. Operating System Virtualization and Linux Containers section points out that OS Virtualization using Linux containers can be a feasible option for application migration.In this section discussion about various OS level Virtualization is done and consideration of the usage of Linux containers is assessed.

2.1 Migration and Resource Management

In cloud computing, Virtualization allows us to dynamically provision the available resources within a group of physical machines, however resource provisioning of a collection of VMs is a difficult task.Migration of a virtual machine from one physical machine to another is one of the strategies used for resource provisioning and balancing by most cloud providers.Mishra et al [18] describe live virtual machine migration as transferring the state of the virtual machine from one physical machine to another thus enabling uninterrupted maintenance and load balancing. In the cloud ecosystem Virtualization is one of the fundamental solutions for resource provisioning since Virtualization provides a virtualized view of the available resources. Virtual Machine Monitors (VMMs) or Hypervisors are used to isolate and allocate the available resources to each VM [2]. Since each VM has to be provisioned with a certain amount of resources before it is instantiated ,over provisioning of resources could happen. Mishra et al [18] point out that identifying the virtual machine types and parameters such as CPU cycles needed or the amount of RAM that could be required for memory usage could help in better resource provisioning rather than the conventional bin-packing approach. Unfortunately there is no fail safe method of identifying how much resource a VM would require especially for a web application, you could never predict the amount of traffic that could come in.

A load balancer handles most of the resources in the cloud ecosystem but prediction of network traffic is a complex task and it might not be accurate enough.Mishra et al [18] identify some of the techniques used for live migration such as suspend and copy, pre-copy and post-copy.Mishra et al [18] go on to explain that the process of suspending the VM ,copying all the pages to the target machine and resuming the VM is called suspend-and-copy live migration. Clark et al [4] describes the pre-copy approach an iterative page transfer from source to destination without suspension of the VM until all the pages that are required for proper working are transferred .In terms of Post-copy migration the transfer time is differed until the process that is being executed [9].These approaches do have conditions attached to the migration time i.e., if the size of the VM large,the time taken to transfer the pages would increase.

Cloud resource provisioning systems suspend/migrate running jobs or Virtual Machines in order to utilize and balance the available resources but the cost of moving the whole Virtual Machine is too high due to its size. Migration of the application is much more feasible and efficient [19]. He,Guo & Guo [8] point out that VM migration is a complicated process which requires freezing and transferring of the state of the VM and memory to another physical machine over the network.

Similarly,Wood et al [34] state that in a Wide area network scenario also the VM migration is a task which is difficult to achieve especially when the size of the VM can be as large as 50 gigabytes.

Wood et al [34] also states that one of the most important parameter that has to be considered is the amount of I/O during VM migration which could create a bottle neck in the network which leads to increase in migration time and of-course higher downtime.

Furthermore,He,Guo & Guo [8] mentions that resource provisioning and management is cumbersome especially because guest the Operating System(OS) always occupies a considerable amount of VM as well as cloud resources. Looking at the above stated scenario ,a better approach towards resource provisioning and management is needed, especially in a hybrid cloud environment. To conclude, the main points to take away from this section are the following. Live VM migration is one of the core enabling factors of resource provisioning in cloud computing, but in certain scenarios when the size of the VM is too large, it could lead to congestion in the network and increase the downtime of the VM. Furthermore, various migration techniques that are carried out in the cloud ecosystem such as pre-copy and post-copy migration is being discussed.

In the next section application migration using process migration is going to be reviewed to analyze if process migration is the right way to do application migration.

2.2 Application Migration by process migration

Milojíčić et al [17] define process migration as the transfer of a current running process to another machine without losing the state of the process.An application is a combination of many processes running together as a whole. So in terms application level migration , process migration is an obvious step forward.Application migration , by migrating a group of processes , is a more fine grained approach towards the management of available resources in the cloud ; for example, migration of a memory intensive application from a memory limited host machine to a memory abundant machine in real time.Nguyen and Thoai [19] suggest that Check-pointing and restarting the process which was saved during a state when the process was error free and independent of any failure is a good method for application migration.Even though the technique suggested by Nguyen and Thoai [19] is highly fault tolerant and doesn't create any difference in the application state, it is also pointed out that there is a downtime during the check-pointing and freezing stage, this would hamper the user experience in a big way.

Since an application contains one or more processes, all the issues that are there for

process migration are also there for application migration, so Clark et al [4] propose VM copying and migration over process migration.

Nguyen and Thoai [19] also point out that Virtual machine migration, helps in maintaining the consistency of in-memory data compared to process migration.

Milojíčić et al [17] point out the residual problem of process migration making it not a good solution for application migration. Apart from the problems that were suggested by Milojíčić et al [17] there are other dependencies attached to the process and maintaining these dependencies could create an extra overhead which could gradually lead to more time for migration of the process. Process migration involves continuous monitoring of memory pages and copying them according to need in the pre-copy approach of migration.

Ibrahim et al [10] point out that constant monitoring of page modification could cause an increase in transitional look-aside buffer(TLB) flushes and soft page faults on write operation which could in turn hamper the performance of the application. Furthermore, continuous copying of pages to the Input output buffer which is sent to the destination machine may cause memory contention.

It is a fact that live migration would involve continuous monitoring and copying of the application memory from target to source [10, 4], in the case of process migration also the this gradual copying of memory is done consistently while the application is still running which could in turn cause memory contention.

To conclude, process migration is a swift way of transferring application from one machine to the other and of-course it is a much more fine grained approach.As mentioned above there are major problems that could occur in certain scenarios and process migration is not the answer for application migration where the fault tolerance is really low.

The various migration techniques that are used all depend on the copying of memory from one machine to other, this technique is fine until the migration happens live and there not much frequent changes in the memory ,but if the application changes the memory more frequently the more complicated the migration process becomes.

The above two sections provide a contradictory view about application migration in general. The first section talks about the heaviness of the VM-hypervisor architecture and how troublesome it would be to migrate a large VM. In the second section there is discussion about process migration, but some major flaws and troubles associated with process migration are identified. The reality is that there is no literature available which explains an architecture which flawlessly migrates an application without any

implications. In the next section operating system Virtualization is explored because by evaluating the problems with hypervisor based live virtual machine migration and process migration, it is found that it can be a light weight approach towards application migration.

2.3 Operating System Virtualization and Linux Containers

One of the major advantages of doing OS level Virtualization is the transparent migration of applications [15]. Similarly Osman et al [20] point out that for a transparent application migration, hardware and operating system Virtualization are the main approaches.Osman et al [20] goes on to explain that Virtualization enables individual applications to be executed in a virtualized environment.

Laadan & Nieh [15] emphasize the fact that with OS Virtualization , the process is isolated within the virtual environment making it possible to run an application within the context without effecting other instances of the same application running in other virtualized environments.Therefore , application independence could be achieved, which implies that different versions of the same application could run at the same physical machine and also this concept could well be implemented to cloud environments where physical machines could be replaced by virtual machines. Although there are two types of OS Virtualization namely host independent and host dependent OS Virtualization , only host-independent OS virtualization has the capability to do application migration. Laadan & Nieh [15] explains that host independent OS Virtualization has a private virtual namespace which could isolate and identify the resources which where referenced by the application.Operating System Virtualization is also known as container-based virtualization.

Xavier et al [35] call container-based vitualization the light weight alternative for hypervisors and Xavier et al [35] also state that the instances of applications running on container instances perform at a near-native level due to its closeness to the base operating system.There have been various implementations of Linux containers in the past; both hypervisors and Linux containers are mature technologies.It is recently that Linux containers have gained renewed attention because of the need for density in the instances of applications

For example, hosting companies have a homogeneous environment where many instances of the same application are deployed on their servers but needs to be isolated from each other, for this kind of homogeneous environment containers are the best

suite because, no matter how many instances of containers are created in an OS it won't effect the system performance unlike VMs .

There are various implementations of Linux containers, to name a few openVZ, docker containers etc [14, 11]. The main advantage that a container has on a virtual machine running on a hypervisor is its ability to instantiate quickly as compared to VMs since there is no requirement of a guest OS for a container. In terms of the migration of containers there are a few approaches that have been implemented. Romero & Hacker [21] explain the live migration process in OpenVZ as a three step process . Firstly, the container is suspended and the memory is dumped onto a file ,secondly the dumped memory file is copied onto the target machine and the migrated container is resumed and the source container is stopped and memory is cleaned. Although the live migration reduces downtime, killing and copying the container onto a different machine is faster. The different approaches towards migration, effects the resources available in the cloud in a big way . Romero & Hacker [21] have compared live migration approaches in terms of parallel applications and the results suggest that frequent checkpointing operations in the application actually reduces the performance of the application. There are various literature available which has implemented check-pointing for efficient application portability, for example, Condor project[24] uses a check-pointing library but these implementation doesn't allow inter process communication which is needed by certain applications [25]. The CRIU [26] implementation support process tree unlike other implementations of checkpoint and restart mechanism which could be an efficient candidate for live migration of Linux containers. CRIU [26] implementation's Linux kernel API would help in management of the user space. In a Linux container based environment unlike hypervisors all the containers share the same operating system kernel so process aggregation within the linux kernel is something which is imperative for application migration. The recent improvement in the Linux kernel (Linux kernel 2.6.24) enables process aggregation and freeze of a group of processes [13] . Cgroups or control Groups [13] helps in resource tracking, grouping and partitioning of processes.Docker [11] is a Linux container Virtualization platform, this utilizes the advantages of cgroups and namespaces within the Linux kernel.

Parameter	LXC	Warden	Docker	OpenVZ
Process Isolation	Uses pid namespace	Uses pid namespace	Uses pid namespace	Uses pid namespace
Network Isolation	Uses cgroups	Uses cgroups	Uses cgroups	Uses cgroups
Filesystem Isolation	Using chroot	Overlay File system using overlayfs	Using chroot	Using chroot
Container Lifecycle	Tools lxc-create,lxc-stop,lxc-startto create,start,stop a ontainer	Containers are managed by running commands on a warden client which talks to warden server	Uses Docker daemon and a client to manage the containers	uses vzctl to manage container lifecycle

Table 2.1: Comaprison of different linux containers [5]

From the above table it is clear that docker container [11] is a prominent choice for application portability and maintenance. Even though docker container doesn't support live migration [11] but a combination of CRIU [26] with docker containers make it possible for efficiently isolating a running application by pausing and resuming a running docker container [27].

Chapter 3

Design and Specification

In this chapter, the overall specification of the combination of tools which would be used in the application migration is explained. The proposed combination of tools would automate the migration of workloads/applications between two physical nodes.

3.1 Networking Requirements

During live server migration across network, the ability to have the IP addresses of the new virtual instance reachable just after migration is a challenge. To satisfy this requirement an independent daemon is setup to maintain the same IP address as the source virtual instance. Unlike the conventional virtual machine migration techniques such as VMotion by VMware, this experiment doesn't require the destination server to be in the same VLAN.

3.2 System/Software Requirements

For this experiment Docker containers are used as linux containers.Docker requires a 64-bit installation. To run properly, docker needs the following software to be installed at runtime:

1. Criu Library (www.criu.org)
2. Runc Open Container
3. Flocker plugin [23]

4. kernel must be 3.10 at minimum
5. iptables version 1.4 or later
6. procps (or similar provider of a "ps" executable) XZ Utils 4.9 or later
7. a properly mounted cgroupfs hierarchy (having a single, all-encompassing "cgroup" mount point is not sufficient)
8. Configure a DNS server for use by Docker.
9. Linux, FreeBSD, or OS X
10. Vagrant (1.6.2 or newer)
11. bash

3.3 Checkpointing and migrating RunC Opencontainer running on native network

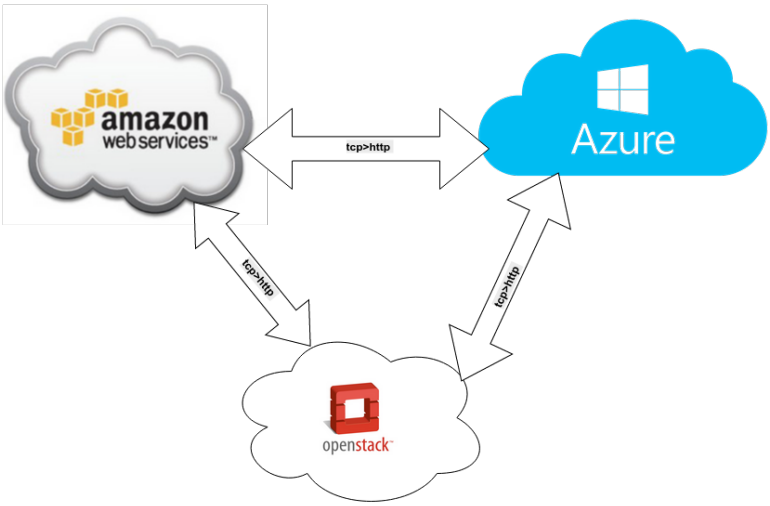

Figure 3.1: Overall architecture of the cross platform application migration

The experiment is for migrating workloads/applications between different cloud platforms with the state of the application still maintained. In the figure 3.1 it shows an overall architecture of the process of migrating a Linux container from and to OpenStack , Amazon and Azure cloud platforms. For this experiment a the file system of a Docker container is used for spawning a runC Linux container.

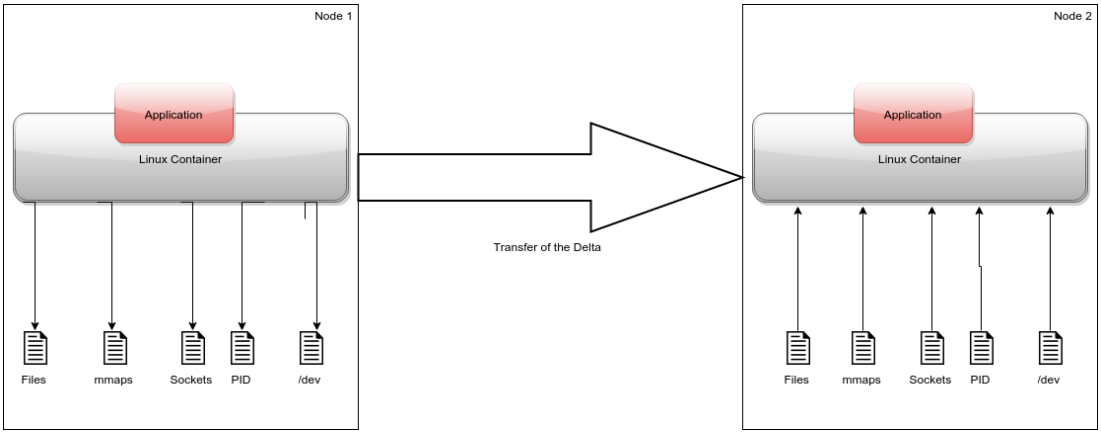

Figure 3.2: Checkpointing , dumping , transfer and restoring container state between nodes

The figure 3.3 shows checkpointing ,dumping, transferring of the state of the runC container with application state between two physical nodes. To maintain the state of the application all the system variables are dumbed to the disk. The system variables include memory maps, sockets , PID structure , devices etc. The file system then has to be compared to the template file system of the application that is then available on the destination node. This whole process is then automated by the automation script. This experiment is isolated only to the context of machines running linux as its base operating system although the proposed concept can also be applied on windows based machines also. The base operating system that is being considered for this experiment is Ubuntu 14.04 with kernel version 3.16.A direct comparison of live migration of application hosted on a virtual machine and a linux container will be made during the proposed experiment.The main consideration for efficiency of the live migration is of time taken for the migration and effect of migration on the performance of the migrated application/s.

3.4 Volume attached docker migration with a shared disk backend

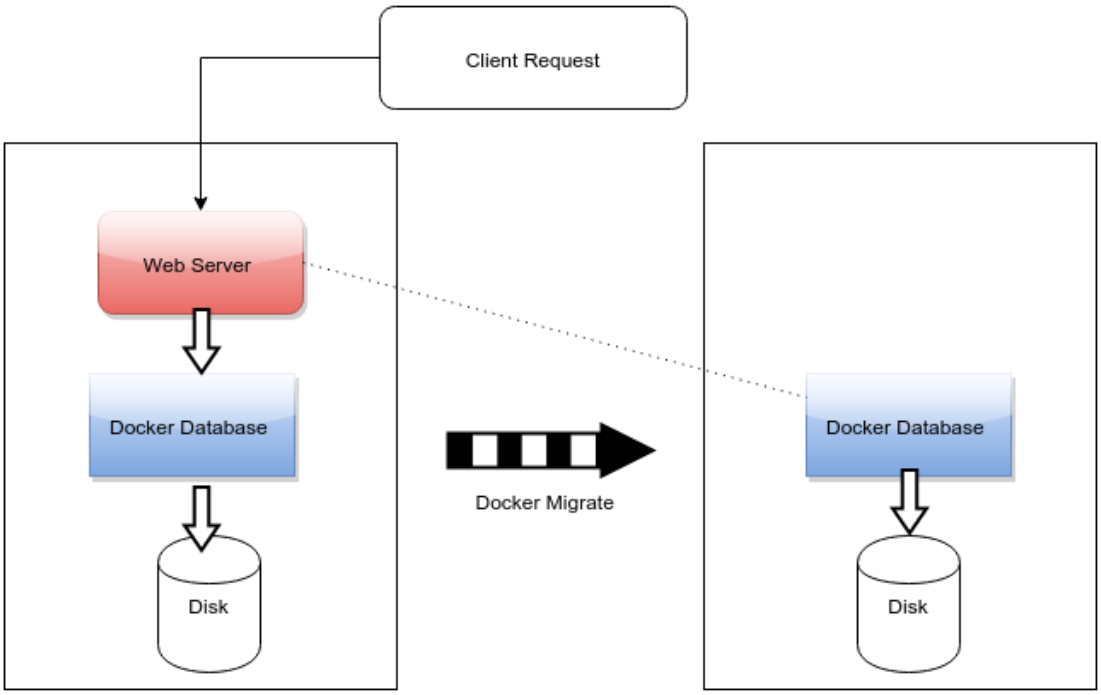

Figure 3.3: Architecture of migration of database docker containers from one node to another

In the experiment the docker containers are database servers connected to a webserver. The database server is then moved to another physical node and there is no change in the overall configuration of the network.The state of the database container should persists even after the transfer. For the configuration of the setup. Some docker plugins are being used for networking and persistent volume transfer. Docker container does not allow transfer of volumes attached to a docker container. For this reason flocker plugin is used to migrate the attached volume with the docker container. The IP address of the container changes when its on the new host and it is very difficult discover the network change while using a container. For this reason weave [33] is used for network discovery and identification.

Chapter 4

Implementation

This chapter describes the implementation and configuration of the migration applications between different Cloud platforms. For the experiment,a popular data structure server, Redis[1] is considered since it a memory persistent key value pair database unlike other database servers which stores on the disk.For this experiment a software which utilizes memory persistence was of importance to prove the state changes of the application. The implementation consists of two experiments and a performance analysis of containers on source and destination platforms of the migrations.

4.1 Checkpointing and migrating RunC Opencontainer running on native network

The experiment is conducted for migrating a Linux container from an Amazon EC2[22] to a Microsoft Azure compute instance[29]. Both the cloud platforms have the exact same configuration. Both the cloud instance have the same kernel configuration and version number.Public IP addresses are also assigned to both the instances. The following implementation is also carried out the same way on both the instances.

Docker natively does not support live migration, so for this experiment Runc Opencontainer[31] is used with a Redis binary running within the container.

4.1.1 RunC Opencontainer

RunC is a command line interface tool for spawning and running Linux containers according to the Open Container Initiative specifications[30].

Building configuring and installing RunC

Configuring Golang and Golang version manager

```
1  bash < <(curl -s -S -L https://raw.githubusercontent.com/moovweb/gvm/master/ ↩
       binscripts/gvm-installer)
2  echo " [[ -s "$HOME/.gvm/scripts/gvm" ]] && source "$HOME/.gvm/scripts/gvm"" >> ↩
       ~/.bashrc
3  gvm install go1.4
4  gvm use go1.4
```

Runc Install

```
1  mkdir -pv $GOPATH/src/github.com/opencontainers/
2  cd github.com/opencontainers
3  git clone https://github.com/opencontainers/runc
4  cd runc
5  make
6  sudo make install
```

Building configuring and installing CRIU

Configuring Kernel

```
1  General setup options
2
3  CONFIG_CHECKPOINT_RESTORE=y (Checkpoint/restore support)
4  CONFIG_NAMESPACES=y (Namespaces support)
5  CONFIG_UTS_NS=y (Namespaces support -> UTS namespace)
6  CONFIG_IPC_NS=y (Namespaces support -> IPC namespace)
7  CONFIG_PID_NS=y (Namespaces support -> PID namespaces)
8  CONFIG_NET_NS=y (Namespaces support -> Network namespace)
9  CONFIG_FHANDLE=y (Open by fhandle syscalls)
10 CONFIG_EVENTFD=y (Enable eventfd() system call)
11 CONFIG_EPOLL=y (Enable eventpoll support)
12
13 Networking support -> Networking options options for sock-diag subsystem
14
15 CONFIG_UNIX_DIAG=y (Unix domain sockets -> UNIX: socket monitoring interface)
16 CONFIG_INET_DIAG=y (TCP/IP networking -> INET: socket monitoring interface)
17 CONFIG_INET_UDP_DIAG=y (TCP/IP networking -> INET: socket monitoring interface ->  ↩
       UDP: socket monitoring interface)
18 CONFIG_PACKET_DIAG=y (Packet socket -> Packet: sockets monitoring interface)
19 CONFIG_NETLINK_DIAG=y (Netlink socket -> Netlink: sockets monitoring interface)
20
21 Other options
```

```
22
23 CONFIG_INOTIFY_USER=y (File systems -> Inotify support for userspace)
24 CONFIG_IA32_EMULATION=y (x86 only) (Executable file formats -> Emulations -> IA32 ↩
       Emulation)
```

Installation :-

```
1 git clone https://github.com/xemul/criu
2 sudo apt-get install libprotobuf-dev libprotobuf-c0-dev \
3   protobuf-c-compiler protobuf-compiler python-protobuf \
4   python-ipaddr libaio-dev asciidoc xmlto
```

For this experiment, the file system of a running container is exported and it then used by runC to spawn and run a Linux container. The runC container is then redirected to the OCF(Open Container Format) file which contains the specifications of the said container.

```
1 sudo runc start config.json
```

Once the runC container starts, the config.json file which container argument for starting the redis server is executed. The runC container is configured to use the native network. Once the redis server starts, it joins the native operating system's process tree. The redis server can then be accessed by the port number that is defined in the config file.

Checkpointing:- The Checkpointing process is very complex in terms of Linux containers because , to maintain all the state of a particular application it is imperative to dump and maintain all the variables that are assigned and are dependent. If there is even a slight change in the one of the files. The restore process fails. For the checkpointing process the runC container is configured with CRIU [26] binary.

```
1 sudo runc checkpoint
```

The runC CLI checks creates a folder called "criu-work" in the /var/run/oci/container-id. The folder also contains the specification needed for restoration of the state of the container.

Migrating: A copy of the file system of the initial container is kept of the destination node. A comparison between both the source file system and a template file system is made and only the difference is transfered to the destination.

4.1.2 Automation Script

The automation script is being remotely executed using an ssh shell.The open source library paramiko[6] is used to connect to both the cloud platforms and to execute commands remotely. The role of the automation script is the following: a) Check the connectivity of the destination cloud platform. b) Copy the initial file system to the destination. c) Checkpoint the active runC container on the source cloud platform. d) Execute a "diff" command on the file system base. e) Copy the delta of the diff to the destination cloud platform f) Reinitialize the runC container on the destination cloud platform.

```
1  k = paramiko.RSAKey.from_private_key_file(PRIVATEKEY)
2  c = paramiko.SSHClient()
3  c.set_missing_host_key_policy(paramiko.AutoAddPolicy())
4  print "connecting"
5  c.connect( hostname = server, username = "ubuntu", pkey = k )
6  print "connected"
7  commands = [ "cd /home/ubuntu/Runc_Experiments/Redis/ && bash rsyc.sh " ]
8  for command in commands:
9    print "Executing {}".format( command )
```

4.2 Volume attached docker migration with a shared disk backend

A docker container with attached volume is being migrated in this implementation.This setup is important to prove the portability of data among different nodes. For achieving this setup Flocker plugin [23] for docker [11] and Weave [33] for networking between docker containers.

4.2.1 Flocker

Flocker is an open source container data volume manager for dockerized applications.For this experiment, Flocker is used since a native docker container does not allow us to migrate a data volume attached to a native docker container.

Installing Flocker Client

```
1  sudo apt-get -y install apt-transport-https software-properties-common
```

```
2  sudo add-apt-repository -y "deb https://clusterhq-archive.s3.amazonaws.com/ubuntu/ ↩
       $(lsb_release --release --short)/\$(ARCH) /"
3  sudo apt-get update
4  sudo apt-get -y --force-yes install clusterhq-flocker-cli
```

Installing Flocker node service

```
1  sudo apt-get -y install apt-transport-https software-properties-common
2  sudo add-apt-repository -y "deb https://clusterhq-archive.s3.amazonaws.com/ubuntu/ ↩
       $(lsb_release --release --short)/\$(ARCH) /"
3  sudo apt-get update
4  sudo apt-get -y --force-yes install clusterhq-flocker-node
```

4.2.2 Weave

Weave is also a plugin that is used with docker containers. Weave helps in creating a virtual network for docker containers which are deployed among multiple hosts and it also helps in automatic discovery.Weave is being used in this experiment to make docker containers accessible to the outside world.

```
1  sudo curl -L git.io/weave -o /usr/local/bin/weave
2  sudo chmod a+x /usr/local/bin/weave
```

4.2.3 Node setup

For this setup two instances are created on virtualbox[32] and Weave and Flocker plugins are installed on to it. Vagrant[7] is used to power-up these machines.

4.2.4 Proxy server

A proxy server to load balance the HTTP traffic coming into the network.The proxy server is an open-source python implementation.For this configuration reactor module of Twisted Web[16] library is used.

```
1  from twisted import reactor as reactor
2  reactor.listenTCP(8080, server.Site(proxy.ReverseProxyResource();
3  reactor.run()
```

Docker-compose[28] is used for configuration and orchestrations of docker containers for this experiment.

Chapter 5

Evaluation

The main aim of this experiment is to prove that Linux containers can be used to overcome vendor-lockin and interoperability problems in a very efficient way by migrating the state of a running application.One of the main problems of migration is mistakes that happens during manual configurations, but in this experiment all system variables and file system required for the application is maintained .

5.1 Migration Process

The migration process consists of the following phases:

1. **Phase I** : During the first phase available resources are checked. The memory utilized and the required memory of the destination node is checked. The space required for creating a new container is checked. Once the requirements are fixed the memory and space for the container is reserved on the destination node.

2. **Phase II** : During the second phase the same base image of the to be migrated application is pulled down from the global repository. The network proxy is initiated and diverted to the destination node. Before the traffic is diverted the diff function is called to check if there are any changes in the new container and the changes are applied.

3. **Phase III** : During the third phase all the traffic is diverted to the new container and the initial container is removed or paused.

5.2 Checkpointing and migrating RunC Opencontainer running on native network

The efficiency of a good process of migration can be identified by the time taken for the migration over the network. The following graph compares the migration time of VM and Linux containers on a network.

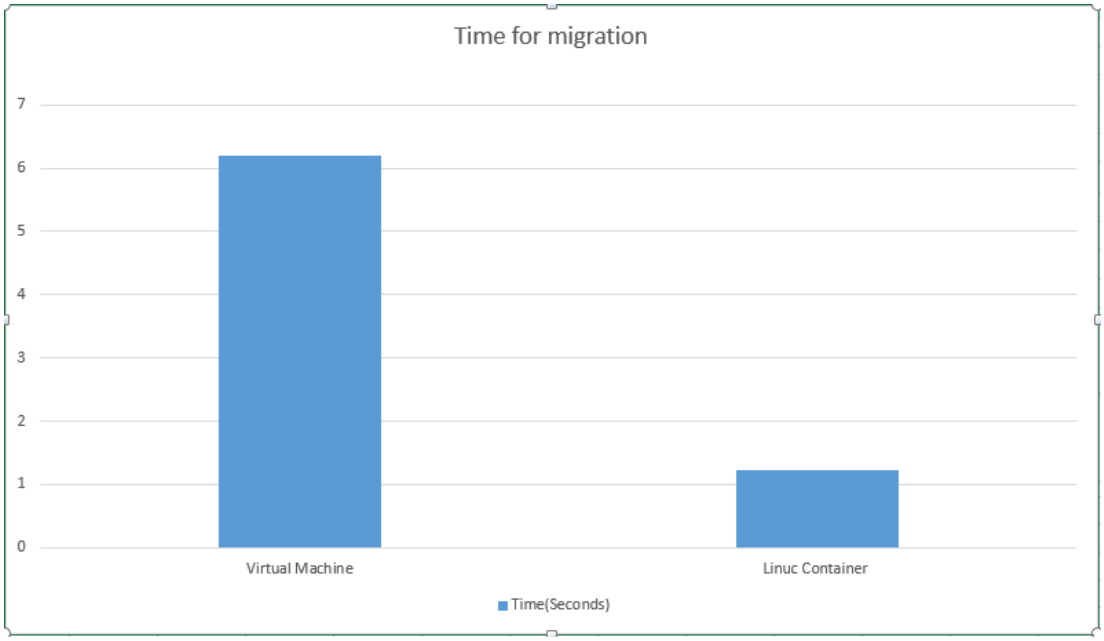

Figure 5.1: The average time required for migration VM vs Linux Container

In the 5.1 a VM with a configuration of 1 GB memory and Ubuntu as operating system is migrated along with the same configuration for a Linux container.Both the VM and Linux container is then installed with a Redis Server. Evaluation is done by examining the migration of an Redis server[1]. The evaluation is done by identifying the effect of migration on the Redis server. The other evaluation criteria that is going to be fundamental is the network congestion during the migration process. A real-time network monitoring is developed for this experiment. The tool is going to continuously monitor the network traffic. In terms of expected result the migration speed and the effect of Linux containers on the network should be less than that of a virtual machine.

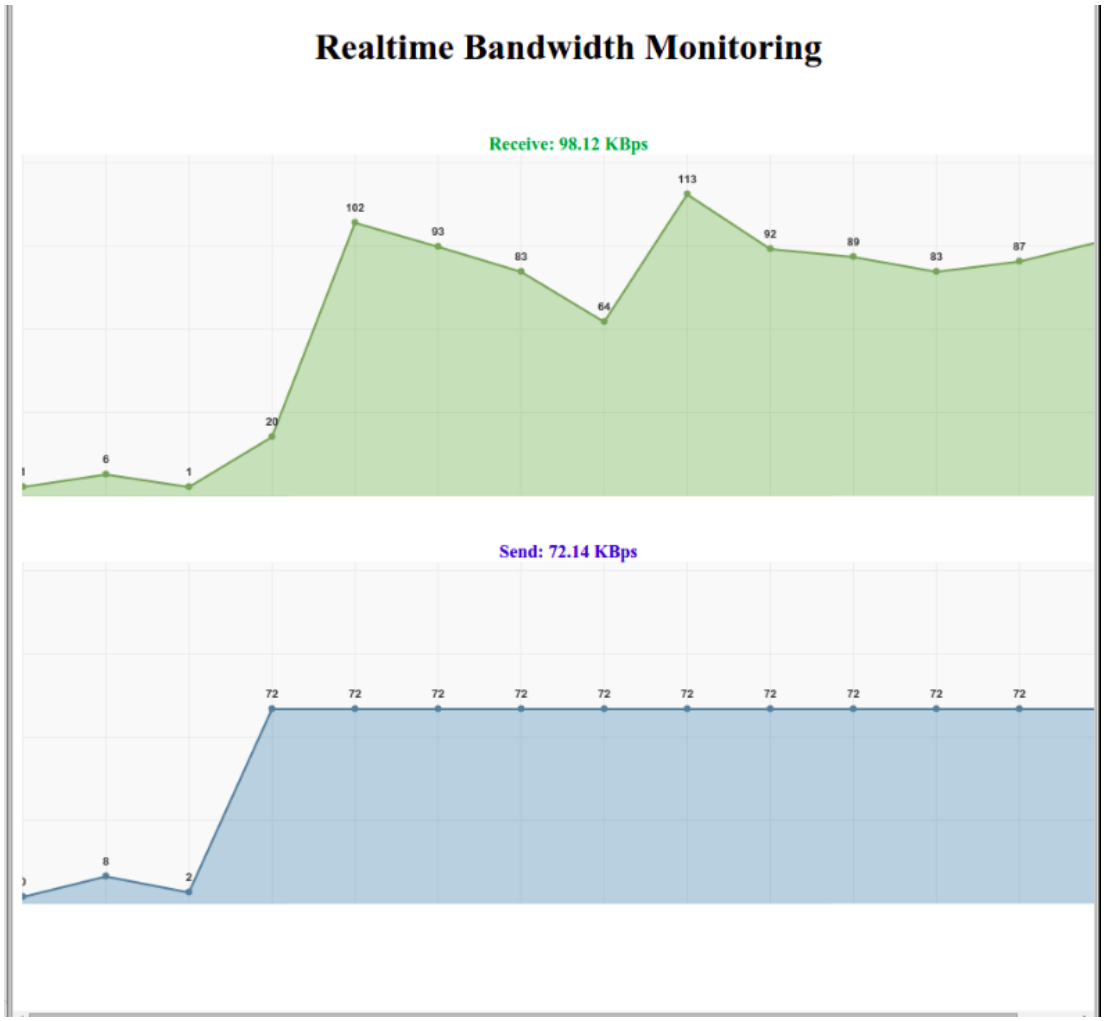

Figure 5.2: Network Congestion during migration

As the above graph shows a real time state of the network during the migration process of the delta file system to the destination cloud platform i.e Azure.

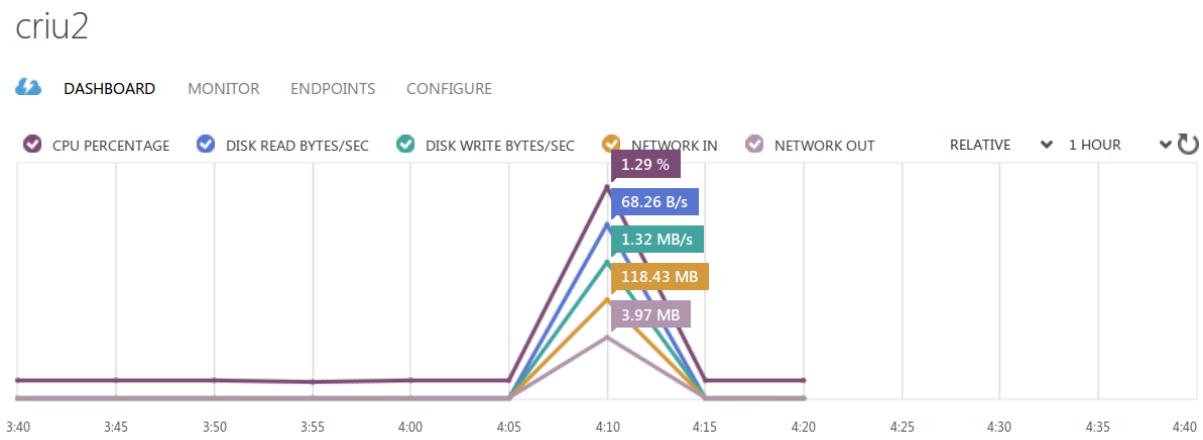

Figure 5.3: Relative network congestion on Azure for 10 iterations

The above graph (5.3) shows the relative network usage during the iterative migration of Linux containers to and from the azure cloud instance.

From the figures 5.3 and 5.2 it can be seen that the migration process has negligible effect on the overall network. The process of migration is also better compared to a VM migration both in terms of speed and network congestion.

5.3 Volume attached docker migration with a shared disk backend

The objective of this experiment was to identify the statefulness of the application migration. In this experiment data moves along with the state of the docker container. The network identity is preserved even after migration.

```
vagrant@master:~$ curl -L -sS http://172.16.255.251:8080
disk: 1
vagrant@master:~$ curl -L -sS http://172.16.255.251:8080
disk: 2
vagrant@master:~$ curl -L -sS http://172.16.255.251:8080
disk: 3
```

Figure 5.4: Curl request before migration

5.4 shows a counter at the web server which is hosted. The counter increases on every curl request.

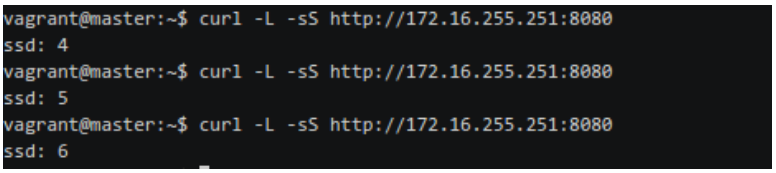

Figure 5.5: Curl request after migration

As the figure 5.5 shows that count of the curl request is still maintained even after migration. From figures 5.4 and 5.5 it is clear that the migration was stateful and the network identity is preserved. This means that there is no need for reconfiguration of the database's new location after the migration after the migration.

Chapter 6

Conclusions

The thesis portrayed a proof of concept that, in the said configurations the state of an application can be transfered across cloud platforms without any kind of changes to the application. In the evaluation chapter two experiments were conducted. The first part was to prove that the state of the application is maintained across different cloud platforms. The experiment also proved that all the variables that are needed for a particular application can be transfered to the destination cloud platform with out any hassle.The second experiment proved the statefulnes of the data which are associated with the docker container. The migration did not require any kind configuration change. The web server was not aware the migration making this scenario apt for web applications. The experiment also proves that there is no difference in performance of the application on different platforms.

The experiment not only shows the efficiency of application migration but also other economic aspects as well. Consider a case where a user is a subscriber of services from two cloud providers, one is considerably cheap in a certain point of time. Now in the case of VMs migrating a whole data-center for certain time of the day would be one big operation. But in the case of the scenario explained in the experiment it is just a matter of seconds depending on the data that is available and there is no need of an extra configuration management.Although this solution is not a perfect answer for vendor lock-in and interoperability but this experiment opens up a new possibility of migrating applications without thinking about the cost and configuration changes involved.

6.1 Limitation of the implementation

The experiment was executed using a Redis server because of the limitation of non availability of resources in terms of compute time and memory.With a higher configuration and compute time the results can be achieved with other applications such as a web server mail server or a video streamer. The configuration was made only considering TCP connections between servers since it is difficult to freeze the other type of sockets in user space.The checkpointing library[26] used in this experiments have certain limitation associated with checkpointing. Tasks with debuger associated with it cannot be checkpointed since the library uses the same debuger (ptrace) to checkpoint the tasks , so the system does not allow to debugers to run. This checkpoint mechanism only allow sockets TCP, UDP, UNIX, packet and netlink, anything other than this is not checkpointed. The Linux container only works in a 64 bit machine.

Bibliography

[1] Salvatore Sanfilippo aka antirez. Redis(Software). https://redis.io, 2015. [Online; accessed 19:52, 9 February 2015].

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003.

[3] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[4] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[5] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia. Virtualization vs containerization to support paas. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 610–614. IEEE, 2014.

[6] Jeff Forcier. Paramiko . https://github.com/paramiko/paramiko/, 2015. [Online; accessed 19:52, 9 July 2015].

[7] Mitchell Hashimoto. Vagrant(Software). https://vagrantup.com, 2015. [Online; accessed 19:52, 9 February 2015].

[8] Sijin He, Li Guo, and Yike Guo. Real time elastic cloud management for limited resources. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 622–629. IEEE, 2011.

[9] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *SIGOPS Oper. Syst. Rev.*, 43(3):14–26, July 2009.

[10] K.Z. Ibrahim, S. Hofmeyr, C. Iancu, and E. Roman. Optimized pre-copy live migration for memory intensive applications. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–11, Nov 2011.

[11] Docker Inc. wdockercompo sehatisdocker. https://www.docker.com/whatisdocker/, 2015. [Online; accessed 19:52, 9 February 2015].

[12] Ian Pratt University of Cambridge Computer Laboratory Keir Fraser, Steven Hand. Xen. http://www.xenproject.org/, 2015. [Online; accessed 19:52, 9 February 2015].

[13] kernel.org team. cgroup. https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt, 2015. [Online; accessed 19:52, 9 February 2015].

[14] Kir Kolyshkin. Main Page. http://openvz.org/Main_Page, 2008. [Online; accessed 19:52, 9 February 2015].

[15] Oren Laadan and Jason Nieh. Operating system virtualization: Practice and experience. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, SYSTOR '10, pages 17:1–17:12, New York, NY, USA, 2010. ACM.

[16] Twisted matrix laboratories. What is Twisted? . https://twistedmatrix.com/trac/, 2015. [Online; accessed 19:52, 9 July 2015].

[17] Dejan S. Milojíčić, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Comput. Surv.*, 32(3):241–299, September 2000.

[18] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo. Dynamic resource management using virtual machine migrations. *Communications Magazine, IEEE*, 50(9):34–40, September 2012.

[19] Duy Nguyen and Nam Thoai. Ebc: Application-level migration on multi-site cloud. In *Systems and Informatics (ICSAI), 2012 International Conference on*, pages 876–880. IEEE, 2012.

[20] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The design and implementation of zap: A system for migrating computing environments. *SIGOPS Oper. Syst. Rev.*, 36(SI):361–376, December 2002.

[21] F. Romero and T.J. Hacker. Live migration of parallel applications with openvz. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 526–531, March 2011.

[22] Amazon team. Amazon Elastic Compute Cloud (Amazon EC2) . https://aws.amazon.com/ec2/, 2015. [Online; accessed 19:52, 9 July 2015].

[23] ClusterHQ team. Flocker(Software). https://clusterhq.com/, 2015. [Online; accessed 19:52, 9 February 2015].

[24] Condor Team. Condor. http://research.cs.wisc.edu/htcondor/, 2015. [Online; accessed 19:52, 9 February 2015].

[25] Condor Team. Condor. http://research.cs.wisc.edu/htcondor/manual/v7.9/2_4Road_map_Running.html, 2015. [Online; accessed 19:52, 9 February 2015].

[26] CRIU Team. CRIU. http://criu.org/, 2015. [Online; accessed 19:52, 9 February 2015].

[27] CRIU Team. CRIU. http://criu.org/Docker, 2015. [Online; accessed 19:52, 9 February 2015].

[28] Docker team. Docker compose. http://weave.works/, 2015. [Online; accessed 19:52, 9 February 2015].

[29] Microsoft Azure team. Microsoft Azure . https://azure.microsoft.com/en-us/documentation/articles/fundamentals-introduction-to-azure/, 2015. [Online; accessed 19:52, 9 July 2015].

[30] Opencontainer team. Opencontainer Initiative. https://www.opencontainers.org/, 2015. [Online; accessed 19:52, 9 July 2015].

[31] Runc team. RunC. https://runc.io/, 2015. [Online; accessed 19:52, 9 July 2015].

[32] Virtualbox team. Virtualbox. https://www.virtualbox.org/, 2015. [Online; accessed 19:52, 9 February 2015].

[33] Weaveworks team. Weave(Software). http://weave.works/, 2015. [Online; accessed 19:52, 9 February 2015].

[34] Timothy Wood, K Ramakrishnan, J Van Der Merwe, and P Shenoy. Cloudnet: A platform for optimized wan migration of virtual machines. *University of Massachusetts Technical Report TR-2010-002*, 2010.

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