# **Virtualization and Migration in Edge computing-Cloud computing models using OpenStack**

***A DISSERTATION***

*Submitted in partial fulfillment of the requirements for the award of*

*The degree*

**MASTER OF TECHNOLOGY**

*In*

**INFORMATION TECHNOLOGY**

***(Specialization:* Software Engineering*)***



By

Seva Sai Praveen

(ISE2017013)

*Under the Guidance of:*

Prof. O. P. Vyas

IIIT-Allahabad

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD**

(A University Established under sec.3 of UGC Act, 1956 Vide Notification No. F.9-4/99-U.3 Dated 04.08.2000

Of the Govt. of India)

**(A Centre of Excellence in Information Technology Established by Govt. of India)**

 **INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD**

**(Deemed University)**

(A Centre of excellence in IT, established by Ministry of HRD, Govt. of India)

**CANDIDATE’S DECLARATION**

****

I do thusly pronounce that the work displayed in this proposition entitled "**Virtualization and Migration in Edge computing-Cloud computing models using Openstack**", submitted in the fractional satisfaction of the level of Masters of Technology (M.Tech), in Information Technology at Indian Institute of Information Technology, Allahabad, is a bonafide record of my unique work completed under the direction of **Prof. O. P. Vyas** due affirmations have been made in the content of the proposition to all other material utilized. This theory work was done in full consistence with the prerequisites and limitations of the endorsed educational modules.

Place: Prayagraj Name of the student

Date: Seva Sai praveen

ISE2017013

**CERTIFICATE FROM SUPERVISOR**

Date:

I do thus suggest that the proposition work arranged under my/our supervision by **Seva Sai praveen** titled “**Virtualiszation and Migration in Edge computing-Cloud computing models using OpenStack**” be acknowledged in the halfway satisfaction of the necessities of the level of Master of Technology in Information Technology for Examination

Date:

Place: Prayagraj

Countersigned by Dean (A & R) Guide’s name & Signature

Prof. O. P. Vyas

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD**

 **(Deemed University)**

(A centre of excellence in IT, established by Ministry of HRD, Govt. of India)

**CERTIFICATE OF APPROVAL\***

The doing without postulation is thus affirmed as a solid report in the zone of Information Technology and its associated regions completed and exhibited in a way attractive to warrant its acknowledgement as an essential to the degree for which it has been submitted. It is comprehended that by this endorsement the undersigned don't really underwrite or favor any announcement made, supposition communicated or conclusion drawn in that yet affirm the proposal just for the reason for which it is submitted.

Signature & Name of the Committee members \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

On final examination and approval of the thesis \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **ACKNOWLEDGEMENTS**

I dedicate this paper for the sake of God, Who is the most thoughtful and the most kind and for my Parents and family who energized me for doing this work." Fruitful consummation of any errand gives us the best fulfilment and inside quality to put in consistent undertaking to achieve flawlessness in our work, however at any stage the individual alone never exists, she is constantly joined by a few people who give her help and proposals for the effective culmination of her work. Along these lines, it's a matter of incredible joy for me to thank each one of those individuals who has energized me and given me their kind-hearted bolster at each phase of my work. Most importantly I might want to pay appreciation to my incredible organization "Indian Institute Information Technology – Allahabad", where I got a stage to take in the new procedures and ideas to improve my aptitudes and to be a capable expert.

I am exceptionally appreciative to the noteworthy **Director, IIIT-Allahabad**, **Prof. (Dr.) P. Nagabhushan**, for his helping state of mind and support, to exceed expectations in examines.

I might likewise want to offer my profound hearted thanks to my Thesis Guide **Prof. O. P. vyas** without whom it would not have been conceivable to achieve this paper. Regardless of his furious timetable he generally reacted with a grin for talking about the issues that came amid this paperwork. He enabled me with the open hand to experiment with new arrangements.

These two years of my life were very memorable as I learned new skills, expertise and gained knowledge from different unique personalities of this institute.

Finally, I bow before my beloved Parents and my family who grew me up physically, mentally and spiritually with prayer and divine love.

Seva Sai Praveen

M.Tech (IT-SE)

# 

# **ABSTRACT**

The evolution of Internet of Things and cloud services have made a milestone, a new computing paradigm, Edge Computing, which has the idea to process the data at the edge of the network. The concerns of bandwidth, battery life and response time can be addressed with Edge computing. The essential features of edge computing are virtualization and migration across the edge nodes. I investigate the aptness of such technologies for a specific layer of the Edge hierarchy, namely the network edge. The main focus is on design of various techniques to facilitate OpenStack based virtualization and migration of containers across IoT devices, for Fog/Edge computing, under Stack4Things framework by Mobile and Distributed Systems Laboratory of University of Messina, Italy. As for the Cloud, essential features of Edge Computing are both virtualization and the capability to migrate virtual environments among nodes. In particular, we investigate the aptness of such technologies for a specific layer of the Edge hierarchy, namely the network edge. Indeed, this layer presents some characteristics that distinguish it from the Cloud so a virtualization or migration technique that represents a decent compromise for the Cloud may not be likewise suitable for the Edge. We focus on design of various techniques to facilitate OpenStack based virtualization and migration of containers across IoT devices, for Fog/Edge computing, under Stack4Things framework by Mobile and Distributed Systems Laboratory of University of Messina, Italy. Edge computing refers to the evolving technologies permitting computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services. Here we tend to outline “edge” as any computing and network resources on the trail between data sources and cloud knowledge centers. Previous work such as micro datacenter, cloudlet, and fog computing has been introduced to the community because cloud computing is not always efficient for data processing when the data is produced at the edge of the network.

# **Table of Contents**

ACKNOWLEDGEMENTS 4

ABSTRACT 5

TABLE OF CONTENTS 6

LIST OF FIGURES 8

LIST OF TABLES 9

Chapter 1.Introduction 10

1.1 Overview 10

1.2 Forensic relevance 11

1.3 Digital Image Pipeline 12

1.4 Different types of Methods 13

1.4.1 Image Features 13

1.4.2 Forensic Techniques based on PRNU 14

1.5 Database Description 15

1.5.1 The Kaggle Image Database 15

1.5.2 The Dresden Image Database 16

1.5.3 The Level Design Reference Image Database 18

1.5.4 3D Max and Maya software Image Database 18

1.6 Motivation 18

1.7 Scope 19

1.8 Organization of Thesis 19

Chapter 2.Literature Survey 21

2.1Secure Digital Camera By Paul Blythe and Jessica Fridrich 21

2.2 Canon, the Canon Verification Kit 21

2.3Higher-order Wavelet Statistics and their Application to Digital Forensics 22

2.4 Methods for identification of images acquired with Digital cameras 22

2.5 Distinguishing Photographic Images and Computer Graphics 23

2.6On Discrimination between Photorealistic and Photographic Images 24

2.7Source camera identification based on CFA interpolation 24

2.8 Identifying Computer Generated and Digital Camera Images Using Fractional Lower Order Moments 25

Chapter 3 Correlation, SVM Approach and CNN Overview 27

3.1 Based on Correlation Approach 27

3.1.1 Methodology 27

3.1.2 Implementation 28

3.1.3 Results 29

3.2 Based on SVM Approach 31

3.2.1 Feature Extraction 31

3.2.2 Methodology 32

3.2.3 Implementation 33

3.2.4 Results 34

3.2.5 Limitations of Pattern Noise 34

3.3 CNN Overview 35

3.3.1 Popularity of Deep Learning 35

3.3.2 CNN Description 35

3.4 Layers of CNN 36

3.4.1 Convolutional Layer 36

3.4.2 Activation Function 37

3.4.3 Pooling Layer 37

3.4.4 Fully Connected Layer 38

3.4.5 Soft-Max 38

Chapter 4.Proposed Methodology 40

4.1 Approach 40

4.1.1 CNN Training 40

4.1.2 SVM Training 41

4.1.3 Deploying the System 41

4.1.4 Deep Learning Model Architecture 41

4.2 Deep learning model on Kaggle Dataset 42

4.3 Deep learning model on Dresden Dataset 45

Chapter 5.Implementation of Proposed Work 46

5.1 Software, Frameworks used for Deep Learning Model Implementation 46

5.1.1 Keras Deep Learning Framework 46

5.1.1.1 Guiding Principles of Keras Framework 46

Chapter 6. Result Analysis 49

6.1 Results 50

Chapter 7.Conclusion and Future Work 51

References 52

**LIST OF FIGURES**

Figure 1: The composite image of British Soldier 10

Figure 2: Altered Image of John Kerry and Jane Fonda 11

Figure 3: How Digital Image is Captured 13

Figure 4: Pattern Noise of Imaging Sensors 14

Figure 5: Dresden Image Database Images 17

Figure 6: Basic Image Specifications- Dresden Image Database 17

Figure 7: Level Design Reference Database Image Sample 18

Figure 8: Secure Digital Camera (Block Diagram) 21

Figure 9: Operations of CCD 23

Figure 10: The Geometry-Based Image Description Framework 24

Figure 11: CFA pattern using RGB values 25

Figure 12: Pattern Noise. 27

Figure 13: Matlab Implementation 28

Figure 14: Python Implementation 29

Figure 15: Correlation of test image with reference error from different cameras. 29

Figure 16: Correlation of test image residual with camera reference error pattern. 29

Figure 17: Correlation of test image residual with camera reference error pattern. 30

Figure 18: Correlation of test image residual with Maya reference 30

Figure 19: Correlation of test images with Maya reference pattern 30

Figure 20: Histogram correlation with Maya reference pattern 30

Figure 21: Image Source Classification 32

Figure 22: SVM Train 33

Figure 23: SVM Test 34

Figure 24: Popularity of Deep Learning for Image Related 35

Figure 25: General Structure of CNN 36

Figure 26: Convolutional Operations of the Kernels 37

Figure 27: The Max Pooling Operation 38

Figure 28:Operation of Fully-Connected Layer 38

Figure 29: Classification Algorithm 40

Figure 30: CNN Architecture 42

Figure 31: Pattern Noise for the Images 43

Figure 32: Code for Pattern Noise Calculation 45

Figure 33: Deep Learning Model in Keras 47

Figure 34: Training Loss Vs Validation Loss 48

Figure 35: Training Accuracy Vs Validation Accuracy 48

**LIST OF TABLES**

Table 1: Confusion Matrix for Different Source Camera 34

Table 2: Confusion Matrix of the CNN model using Kaggle Dataset 48

Table 3: Confusion Matrix of the CNN model using Dresden Dataset 48

Table 4: Literature Comparison 49

Chapter i1.Introduction

## Overview

Cloud computing with its evolution in 2005 has changed our lives tremendously providing the services like Infrastructure, platform and Software instances. These actually changed the way of running business, for instance GFS, MapReduce, Spark and Hadoop. The Internet of Things being introduced in 1999 for supply chain management has the concept of making a computer sense data without any intervention of humans and this is widely adapted to other fields like smart city and healthcare. With the availability of IoT in hand we dive into the post cloud era with new set of problems like big data. On this particular context the OpenStack open-source cloud platform can best serve the needs of cloud and IoT over the edge layer of the network.

* 1. **Smart City Context**

1. ***Edge computing***

Edge computing allows computation to be performed at the Network Edge, considering the downstream data referring to cloud services and upstream data referring to IoT services. Edge is computation of resources along the path between cloud data centers and data origin. The amounts of data being produced at the edge of the network is increasing tremendously and it would be more efficient to process the data at the origin itself. Previous work such as micro datacenter, cloudlet, and fog computing has been introduced to the community because cloud computing is not always efficient for data processing when the data is produced at the edge of the network.

1. ***Virtualization***

Virtualization is setting up a virtual environment on any current existing system, without any involvement with any of the other services provided by the server or host platform to other users. This could be of a single instance or a set of instances which resembles multiple operating systems

1. ***Migration***

Migration is the process of moving data, applications, system state or other business elements from an edge node to another. It involves shifting data or system states between environments too. The migration of an edge service/virtual environment from an edge node to another may be either stateless or stateful.

Lightweight virtualization technologies have sophisticated the era of cloud computing services by introducing flexibility and innovation to this domain. For example, the IoT and Mobile Edge Computing (MEC) have benefitted by the usage of container-virtualization. This directed to implement a growing trend on how efficiently we can redesign the critical components of IoT/Edge scenarios (e.g., gateways) to enable the concept of device virtualization. To morph the Italian city of Messina, into a smart city, an Open Data platform has been set up by the University of Messina through the employment of cost efficient microcontroller boards coupled with sensors, actuators and installed on buses, lamp posts, and buildings of local institutions, all over the urban area. The open source project by the University of Messina is named Stack4Things that assists in organizing fleets of IoT devices less concerning about the actual physical location , network configuration or their existing technology. The migration techniques involved are part of this system which enables the edge nodes (IoT devices) to work as a unit with cloud to process and analyze data. This provides the surety that all existing resources are used efficiently and helps in overall management of smart cities under SmartMe initiative.

* 1. **Motivation**

In this paper, I provide a comprehensive overview of both the state-of-the-art virtualization and migration approaches and their available implementations. More precisely, we survey their suitability for the network edge. The motivation behind designing of various mechanisms to facilitate container migration in SmartMe based applications can be summarized as follows:

1. The SmartMe initiative involves multiple use cases where the container runtime needs to be migrated - along with the running processes - and deployed on a new node on the network edge that is closest to the client that is leveraging that particular process in an attempt to minimize latency.
2. Zun does not provide any framework for migration of the runtime across nodes.
3. There needs to be thorough research on the container deployment and execution mechanisms in order to design a technique for migration.
4. There needs to be implementation of supporting architecture for handling the modified container image files.
5. OpenStack allows Fog based node container management by use of Zun. The container migration mechanism must be developed such that it integrates seamlessly with container lifecycle management of OpenStack Zun and is transferable across nodes in OpenStack cloud Infrastructure and IoT devices.

**Chapter i2. iLiterature iSurvey**

**2.1 Carlo Puliafito, Enzo Mingozzi, Carlo Vallati, Francesco Longo, Giovanni Merlino Virtualization and Migration at the Network Edge: an Overview, 4th IEEE International Conference on Smart Computing (SMARTCOMP) Taormina, Italy, July 2018.**

This paper provides us with insights about existing migration techniques at network edge and potential ways of implementation of virtualisation at network edge. In particular, I investigated the aptness of such technologies for a specific layer of the Fog hierarchy, namely the network edge. The major proposition was that all those virtualization and migration techniques that are suitable for the Cloud might not likewise suitable for the edge

**2.2 Roberto Morabito - Virtualization on Internet of Things Edge Devices with Container Technologies: a Performance Evaluation, IEEE Access May 2017**

This paper provides us with performance comparison and analysis of IoT devices(versions of Pi, Odroid) using container technologies for virtualization. To be precise, the Internet of Things (IoT) and mobile edge computing benefit from container virtualization by exploiting the possibility of using these technologies not only in data centers but also on these edge devices, which are equipped with fewer computational resources. This has created a trend to efficiently redesign the critical components of IoT/systems precisely the gateways to apply the concept of device virtualization.

**2.3 Antonio Puliafito et al - Building a Smart City Service Platform in Messina with the #SmartME Project, 32nd International Conference on “Advanced Information Networking and Applications Workshops (WAINA)” 2018**

This paper gives insight on the SmartMe project. Projecting this definition in the technological context, a SmartME project can be thus considered as a service to an ecosystem of infrastructure and services aiming at implementing the aforementioned characteristics for a smart city. This holistic view calls for an all-encompassing approach able to embrace heterogeneous technologies and services, thus providing a wider (or even a global) solution to (Smart) city problems. In this light, there is the need for a scalable architecture aiming at reusing, multiplexing, and sharing technologies and services on the urban scale.

**2.4 Antonio Puliafito et al - Stack4Things: An OpenStack-Based Framework for IoT, - FICLOUD '15 Proceedings of the 3rd International Conference on Future Internet of Things and Cloud, 2015**

The open source project by the University of Messina is named Stack4Things that assists in organizing fleets of IoT devices less concerning about the actual physical location , network configuration or their existing technology. The migration techniques involved are part of this system which enables the edge nodes (IoT devices) to work as a unit with cloud to process and analyze data. This provides the surety that all existing resources are used efficiently and helps in overall management of smart cities under SmartMe initiative. Upon this concern OpenStack-based Internet of Things framework was developed by the Mobile and Distributed Systems Lab (MDSLab) at the University Of Messina, Italy. In this context, a competitive framework for IaaS such as OpenStack, with its flexibility in terms of coverage and expanded scope, provides a perfect solution.

**2.5 Giovanni Merlino, Dario Bruneo et al - Stack4Things: Integrating IoT with OpenStack in a Smart City context International Conference on “Smart Computing Workshops”, 2014**

This paper gives overview of architecture of IoT device and its integration with OpenStack services such as Nova, Celiometer, Horizon. It also elaborates on technical details and design decisions of the Stack4Things framework components in its standalone prototype - s4tProbe, s4tAgent, s4tCollector. In order to fill the gap between Smart City applications and the underlying infrastructure, in this paper a proposal to extend a well-known framework for the management of Cloud computing resources, OpenStack, to manage sensing and actuation ones, implementing in our *Stack4Things* solution an infrastructure-oriented approach, while coping with communication requirements and scalability concerns by leveraging Cloud-focused design choices and architectural patterns.

**2.6 Dario Bruneo - Head in a Cloud: An approach for Arduino YUN virtualization, IEEE conference 2017 i**

This paper proposes a Virtualization approach for Arduino YUN. In this paper focus is on smart board virtualization, implementing a flexible solution for Arduino boards based on Stack4Things, allowing to create, multiplex, migrate and deploy virtual boards in IoT-Cloud contexts. The results obtained by a preliminary implementation and experiments on the #SmartME testbed are shown in the paper to demonstrate the feasibility and the effectiveness of the proposed solution.

**2.7 Flávio Ramalho, Augusto Neto Virtualization at the network edge: A performance comparison IEEE 17th International Symposium on "A World of Wireless, Mobile and Multimedia Networks", 2016**

This paper elaborates on various virtualization techniques at network edge along with their performance comparison and analysis. A study was conducted to evaluate the performance of the container-based approach compared to a hypervisor-based virtualization when running on devices typically used at the network edge. The study was performed through the execution of several synthetic benchmarks providing an insight in the performance overhead introduced by Docker containers (lightweight-virtualization) and KVM VMs (hypervisor-virtualization) running at network edge devices.

**Literature survey summary**

Upon thorough observation and review of the above mentioned publications, openstack provides fundamental infrastructure building blocks that can be deployed anywhere, including the edge of the network. This flexible and modular nature of OpenStack means you can efficiently run the minimal services required at the edge, yet provide robust support for bare metal, container technologies and virtual machines. OpenStack is already the most highly distributed infrastructure software, running in thousands of datacenters around the world.

**Chapter i3. iMotivation**

The motivation behind designing of various mechanisms to facilitate container migration in SmartMe based applications can be summarized as follows:

1. The SmartMe initiative involves multiple use cases where the container runtime needs to be migrated - along with the running processes - and deployed on a new node on the network edge that is closest to the client that is leveraging that particular process in an attempt to minimize latency.
2. Zun does not provide any framework for migration of the runtime across nodes.
3. There needs to be thorough research on the container deployment and execution mechanisms in order to design a technique for migration.
4. There needs to be implementation of supporting architecture for handling the modified container image files.
5. OpenStack allows Fog based node container management by use of Zun. The container migration mechanism must be developed such that it integrates seamlessly with container lifecycle management of OpenStack Zun and is transferable across nodes in OpenStack cloud Infrastructure and IoT devices.

**Chapter 4i. iOpenStack, Docker, Zun Container service**

**4.1 Openstack**

OpenStack has its initial release on 21st October 2010 is a set of tools for developing, managing public and private cloud platforms. It has been supported by some of the biggest companies like redhat, Dell, CISCO, IBM and also some thousands of individual communities in development and cloud hosting. OpenStack looked like the face of Cloud Computing in future.

OpenStack is later on managed by a non-profit corporate entity called OpenStack Foundation established in September 2012 that overlooks both cloud based development and community-building. OpenStack provides services for the users to deploy virtual machines and instances that perform different tasks for managing a cloud environment. The horizontal scaing is made easy using OpenStack. Horizontal scaling means performing the tasks concurrently and serve more and more users easily on the go only just by launching up more instances.

For example, a mobile application that needs to communicate with a remote server might be able to divide the work of communicating with each user across many different instances, all communicating with one another but scaling quickly and easily as the application gains more users.

To be more precise, OpenStack is an open source platform, which provides anyone who chooses to, can get access to the source code of the software, make relevant changes or modifications they need, and freely share these changes back out to the community at large. It also means that OpenStack has the benefit of thousands of developers all over the world working in tandem to develop the strongest, most robust, and most secure product that they can.

The cloud is all about providing computing for end users in a remote environment, where the actual software runs as a service on reliable and scalable servers rather than on each end-user's computer. Cloud computing can refer to a lot of different things, but typically the industry talks about running different items "as a service" software, platforms, and infrastructure. OpenStack falls into the latter category and is considered Infrastructure as a Service (IaaS). Providing infrastructure means that OpenStack makes it easy for users to quickly add new instance, upon which other cloud components can run. Typically, the infrastructure then runs a "platform" upon which a developer can create software applications that are delivered to the end users.

**4.2 Openstack components**

OpenStack is made up of many different moving parts. Because of its open nature, anyone can add additional components to OpenStack to help it to meet their needs. But the OpenStack community has collaboratively identified nine key components that are a part of the "core" of OpenStack, which are distributed as a part of any OpenStack system and officially maintained by the OpenStack community.

**4.2.1 Nova**

Nova is the primary computing engine behind OpenStack. It is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks.

**4.2.2 Swift**

Swift is a storage system for objects and files. Rather than the traditional idea of a referring to files by their location on a disk drive, developers can instead refer to a unique identifier referring to the file or piece of information and let OpenStack decide where to store this information. This makes scaling easy, as developers don’t have the worry about the capacity on a single system behind the software. It also allows the system, rather than the developer, to worry about how best to make sure that data is backed up in case of the failure of a machine or network connection.

**4.2.3 Cinder**

Cinder is a block storage component, which is more analogous to the traditional notion of a computer being able to access specific locations on a disk drive. This more traditional way of accessing files might be important in scenarios in which data access speed is the most important consideration.

**4.2.4 Neutron**

Neutron provides the networking capability for OpenStack. It helps to ensure that each of the components of an OpenStack deployment can communicate with one another quickly and efficiently.

**4.2.5 Horizon**

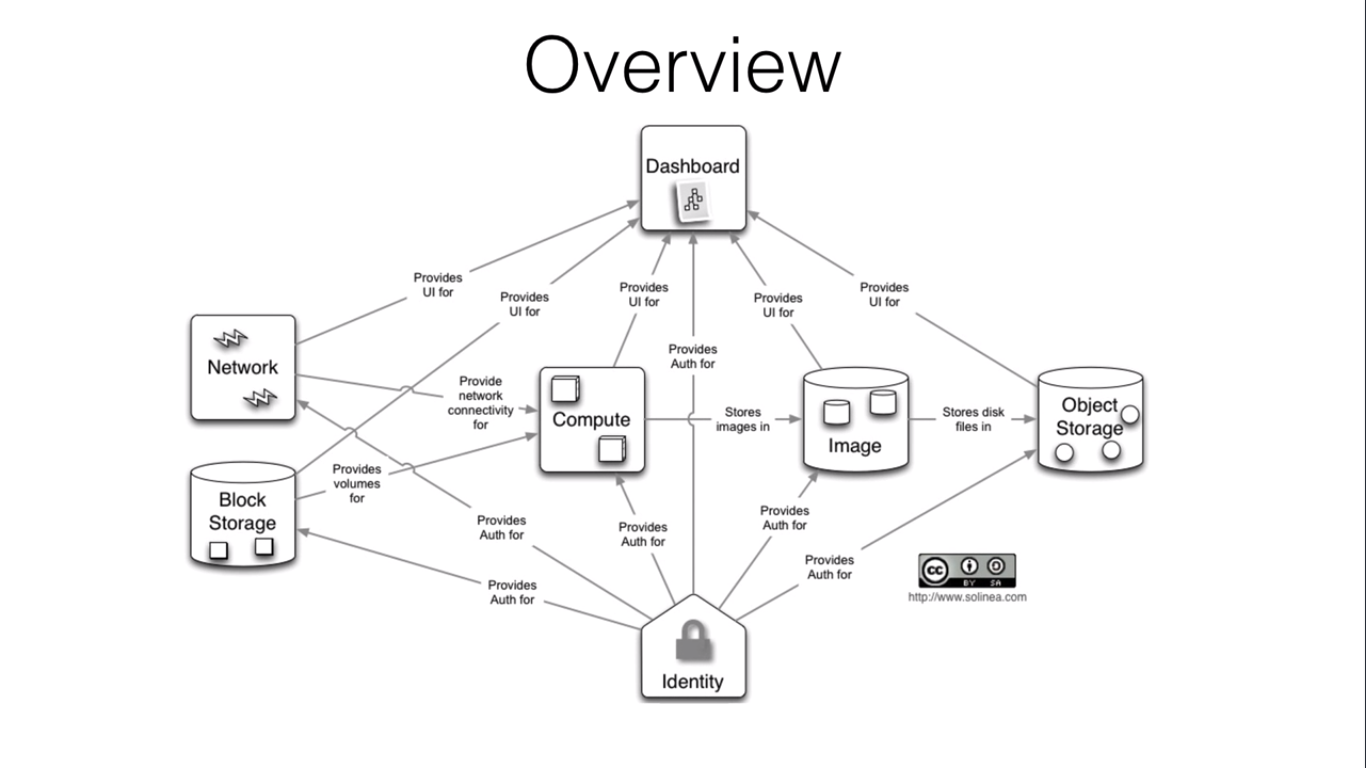
Horizon is the dashboard behind OpenStack. It is the only graphical interface to OpenStack, so for users wanting to give OpenStack a try, this may be the first component they actually “see.” Developers can access all of the components of OpenStack individually through an application programming interface (API), but the dashboard provides system administrators a look at what is going on in the cloud, and to manage it as needed.

**4.2.6 Keystone**

Keystone provides identity services for OpenStack. It is essentially a central list of all of the users of the OpenStack cloud, mapped against all of the services provided by the cloud, which they have permission to use. It provides multiple means of access, meaning developers can easily map their existing user access methods against Keystone.

**4.2.7 Glance**

Glance provides image services to OpenStack. In this case, "images" refers to images (or virtual copies) of hard disks. Glance allows these images to be used as templates when deploying new virtual machine instances.



**Cinder**

**Keystone**

**Swift**

**Glance**

**Nova**

**Neutron**

**Horizon**

**Fig1: OpenStack Components**

**4.3 OpenStack Environment setup**

**4.4 Zun Container service**

As the component providing container management service, Zun allows users to rapidly start and operate the management container without the participancy of management server or cluster. It is integrated with Neutron, Cinder, Keystone and other core OpenStack services to achieve the rapid popularization of container. By this way, all the original network, storage and identification verification tools of OpenStack are applied to the container system, so that containers can meet the security and compliance requirements. Zun plans to support multiple kinds of container technologies such as Docker, Rkt, and clear container. Now, the support for Docker technology has been completed. Currently, OpenStack has the following mainstream solutions supporting container technologies:

**4.4.1 Nova Docker driver**

This solution operates container as VMs. Nova Docker driver is added to carry out operations similar with that of regular VM to start, stop or create a Docker container. Due to the differences between Docker and VM, such operation mode will disable many functions of container, such as container correlation and port mapping.

Zun manages container as a kind of OpenStack resource, and integrates other services of OpenStack, to provide users with a unified and simplified API. Users can create and manage containers through the API, and do not need to consider the differences among different container technologies. Zun has been integrated with multiple OpenStack services. Keystone, Neutron and Kuryr-libnetwork are necessary services for running Zun. They provide Zun with authentication, network, connection between neutron and docker networks. For OpenStack users, it is easy to learn to use the Zun container.

The advantage of integrating OpenStack services is that users can extend the functions of container with the help of OpenStack’s existing functions. For example, by default, the Zun container can use the IP address assigned by Neutron and can use the authentication service provided by Keystone. Using Zun together with Neutron, users can create a container in the isolated network environment where the Nova instance is located. The VM’s Neutron function (security group, QoS) is also available for Zun containers. In the actual business, there are often scenes that need to save data for a long time. A common method is to use external services to provide a persistent volume for the container. Zun solves this problem by integrating with OpenStack Cinder. When creating a container, the user can choose to mount the Cinder volume to the container. A Cinder volume can be an existing or newly created volume in a tenant. Each volume will be bound to the container file system path, and the data stored under that path will be persisted.

**4.4.2 Docker**

Docker is a set of coupled software-as-a-service and platform-as-a-service products that use operating-system-level virtualization to develop and deliver software in packages called containers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc. The service has both free and premium tiers. Containers are isolated from each other and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines. Containers are created from images that specify their precise contents. Images are often created by combining and modifying standard images downloaded from public repositories.

Before containerization came into the picture, the leading way to isolate, organize applications and their dependencies was to place each and every application in its own virtual machine. These machines run multiple applications on the same physical hardware, and this process is nothing but Virtualization. But virtualization had few drawbacks such as the virtual machines were bulky in size, running multiple virtual machines lead to unstable performance, boot up process would usually take a long time and VM’s would not solve the problems like portability, software updates, or continuous integration and continuous delivery. These drawbacks led to the emergence of a new technique called Containerization.

**4.4.3 Containerization**

Containerization is a type of Virtualization which brings virtualization to the operating system level. While Virtualization brings abstraction to the hardware, Containerization brings abstraction to the operating system. Containers have no guest OS and use the host’s operating system. So, they share relevant libraries & resources as and when needed. Processing and execution of applications are very fast since applications specific binaries and libraries of containers run on the host kernel. Booting up a container takes only a fraction of a second, and also containers are lightweight and faster than Virtual Machines.

**4.4.4 Dockerfile**

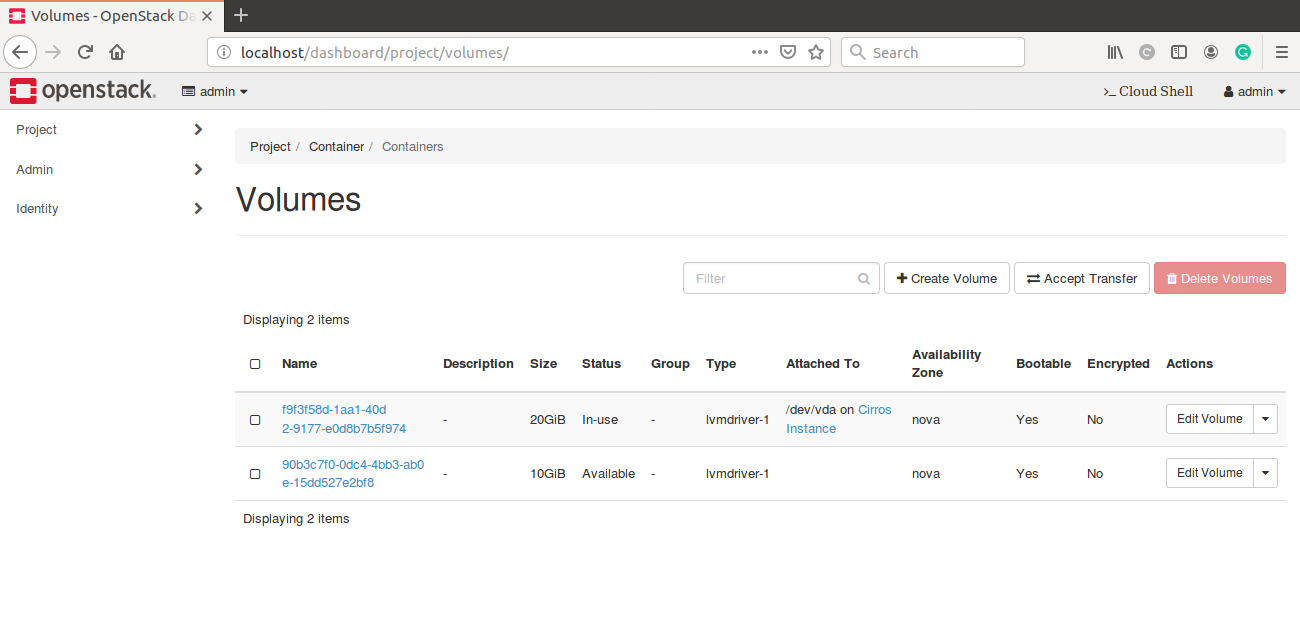
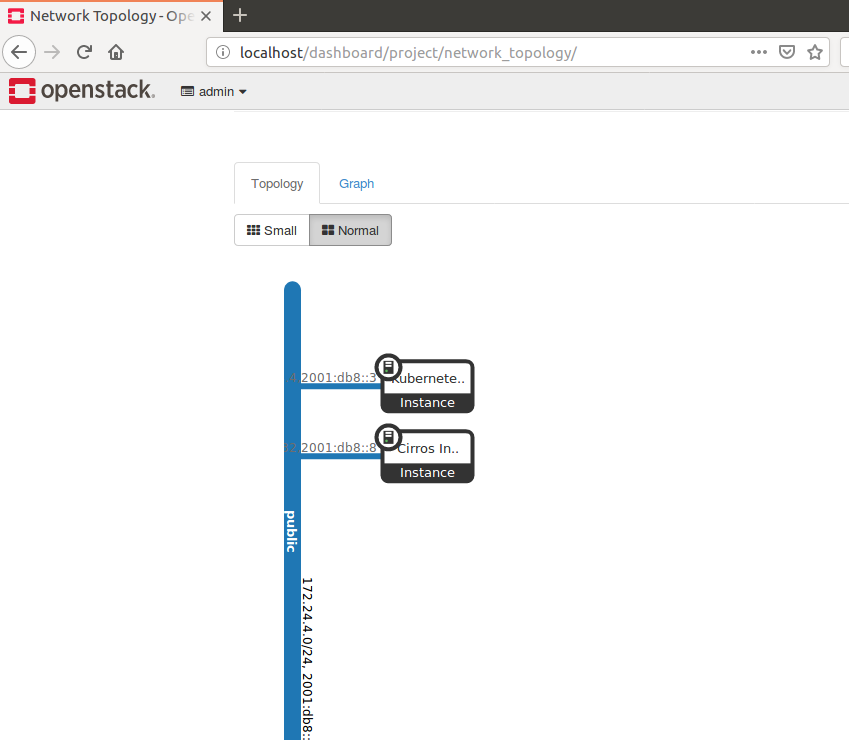
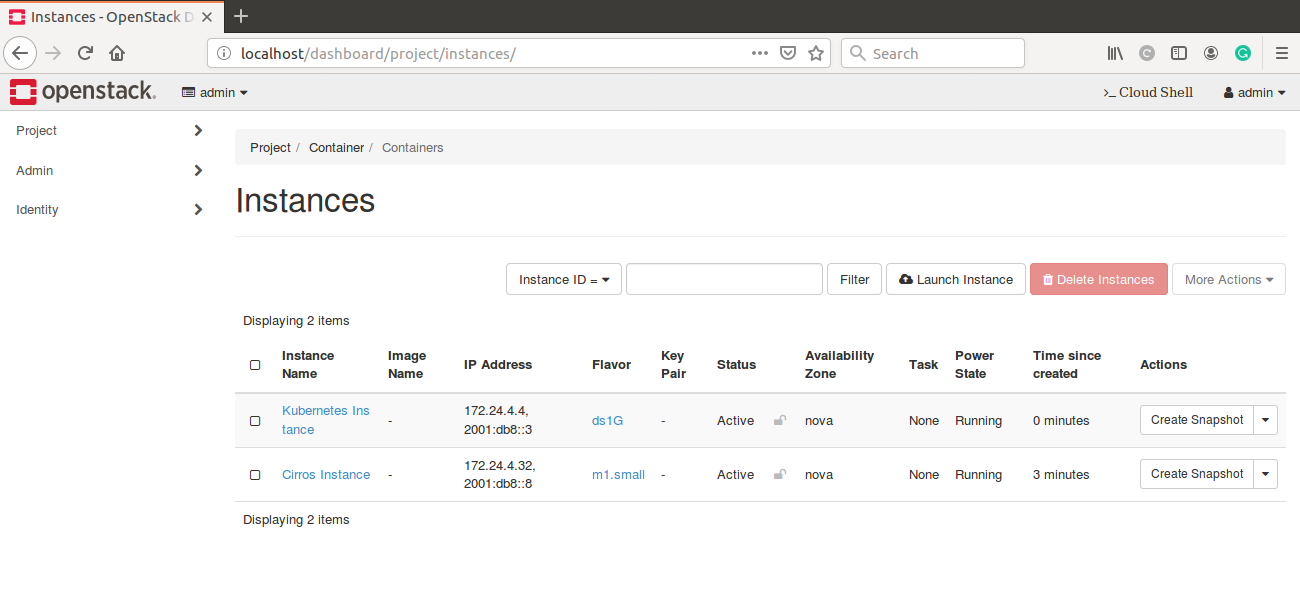
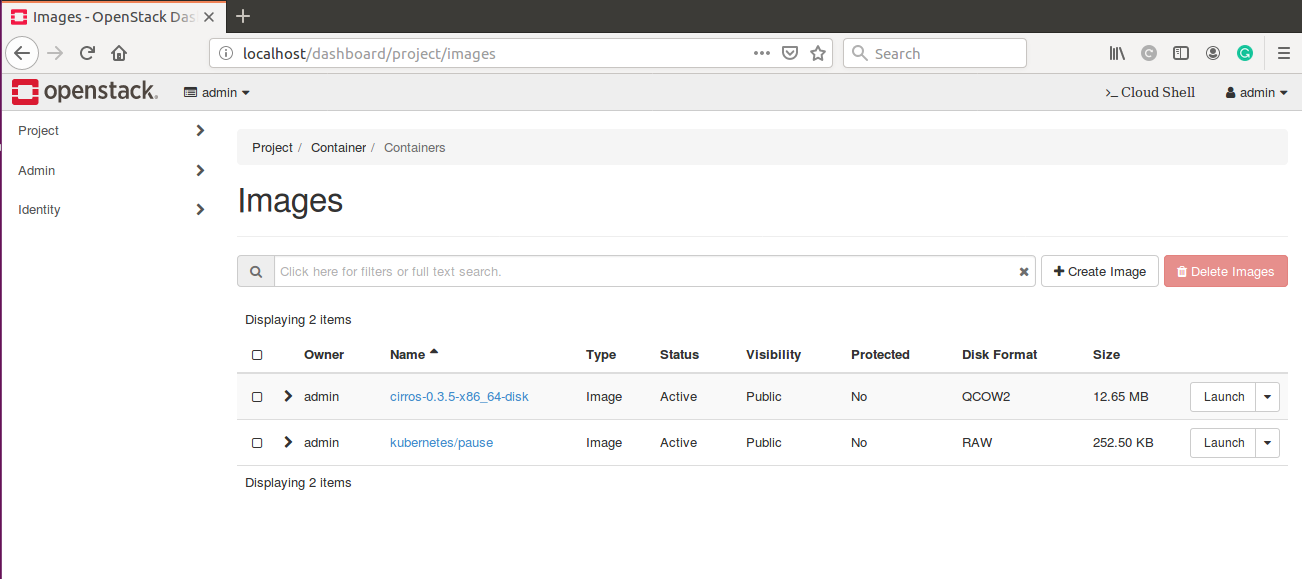
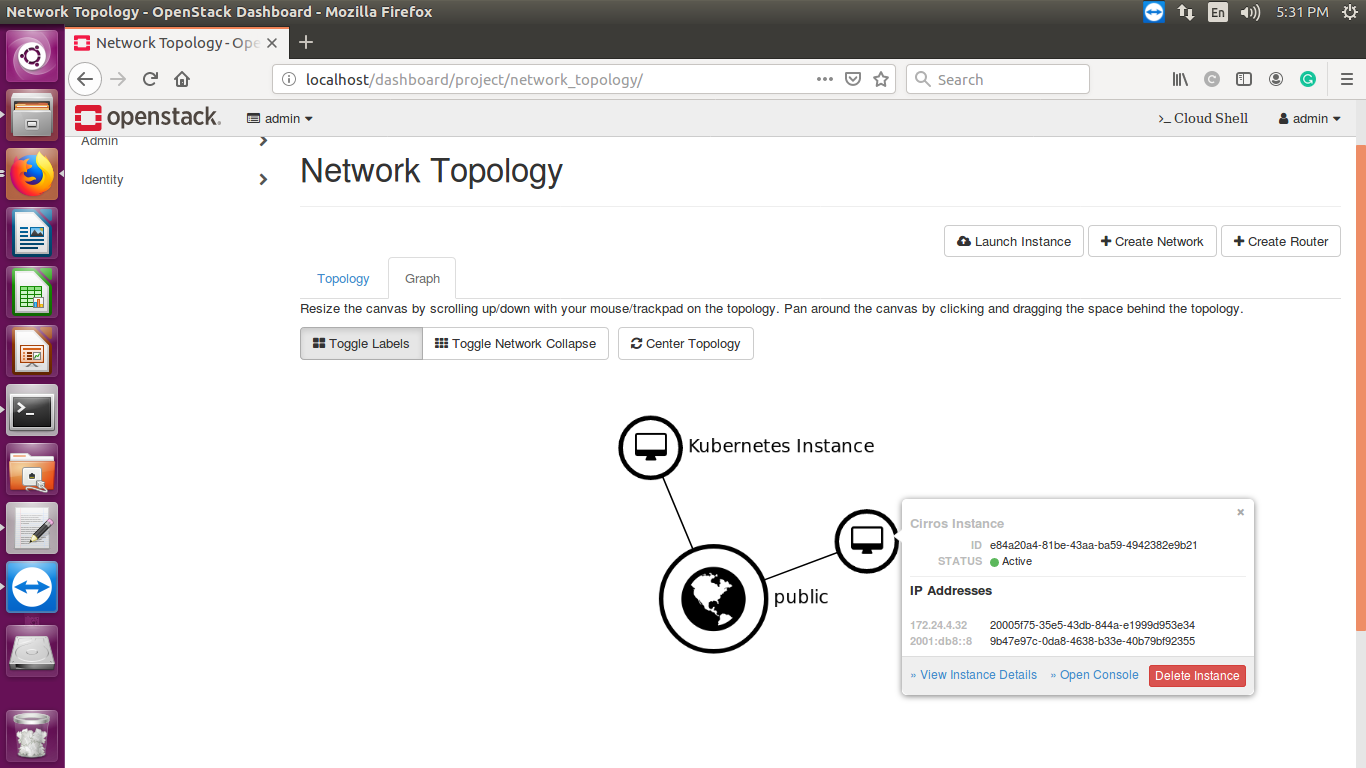
A Dockerfile is a text document which contains all the commands that a user can call on the command line to assemble an image. So, Docker can build images automatically by reading the instructions from a Dockerfile. You can use docker build to create an automated build to execute several command-line instructions in succession.

**4.4.5 Docker Image**

In layman terms, Docker Image can be compared to a template which is used to create Docker Containers. So, these read-only templates are the building blocks of a Docker Container. You can use docker run to run the image and create a container. Docker Images are stored in the Docker Registry. It can be either a user’s local repository or a public repository like a Docker Hub which allows multiple users to collaborate in building an application.

**4.4.6 Docker Container**

Docker Container is a running instance of a Docker Image as they hold the entire package needed to run the application. So, these are basically the ready applications created from Docker Images which is the ultimate utility of Docker.



**Chapter i5. Problem Definition**

**5.1 iObjectives**

1. Research container image format support for OpenStack Zun
2. Research and implement alternatives to DockerHub for pulling container-images
3. Investigate the passing of parameters to RunC container runtime from Zun CLI
4. Design checkpoint/restore functionality in addition to container commit

### 5.1.1 Research container image format support for OpenStack Zun

Zun currently uses runC to spawn containers from images pulled from DockerHub. There appears to be conflict in image format and compatibility of dockerHub container images with runC and Zun. In order to ensure compatibility of dockerHub images and checkpoint-commited images which are to be migrated across edge devices, the underlying format of dockerHub images and runC container runtime needs to be thoroughly investigated.

### 5.1.2 Research and implement alternatives to DockerHub for pulling container-images

The system in its current implementation only has support for images pulled via DockerHub. However, in order to make the process of migration simple and seamless it is intended to allow the system to pull container images stored at any end URL, and successfully spawn them as containers on the zun managed nodes.

### 5.1.3 Investigate the passing of parameters to RunC container runtime from Zun CLI

Since Zun uses runC as container runtime and any commands and parameters passed to Zun are to be finally run on the container, an exhaustive inspection of how exactly the commands and parameters that are being passed to Zun are resulting in the actions of container needs to done.

### 5.1.4 Design checkpoint/restore functionality in addition to container commit

Checkpoint/Restore is the most critical functionality that must be targeted before migration can be achieved. Checkpoint and restore, in terms of a single process system implies that the runtime and execution state of the process is saved on the persistent memory as a snapshot along with allowing us to restore the process from its previous snapshot whenever desired. In addition to allowing checkpoint/restore inside the container runtime, the container image may also be modified with the process snapshot so that future containers may be spawned from the modified image itself.

**Chapter i6. Methodology**

**6.1 Research container image format support for OpenStack Zun**

The container management service of OpenStack cloud infrastructure, *Zun*, uses *runC*, a lightweight universal container runtime, to spawn containers from their container images. *runC*, developed by OpenContainer Initiative, can spawn containers which abide by their Image Specification (Open Container Image - OCI format). OpenStack Zun, by default, uses dockerHub as disk/container-image repository to pull images. These images on dockerHub are in Docker V2 Image format, which itself is based on OCI image format[9][10]. Consequently, the images which are pulled from dockerHub are compatible with runC, and thereby Zun, and can be spawned to form Containers. The images when pulled from dockerHub are downloaded to persistent storage of local machine and can be accessed independent of OpenStack via docker command line interface.

Any container service

Openstack  
Zun container service

Loading the container image

Default

Image pulling

Local machine storage

Image pulling

**Docker Hub**

(Open Container Image format)

**Docker Hub**

(Docker V2 image format)

**Fig 2: Image compatibility issues**

## 6.2 Research and implement alternatives to DockerHub for pulling container-images

### 6.2.1 Glance

The OpenStack Glance is an image service which serves the purpose of registering, retrieving the disks and server images. The OpenStack Glance is more like a warehouse for virtual images. Glance has RESTful API that allows querying of VM image metadata as well as retrieval of the actual images. Glance supports wide range of image formats [11]. Container images can be pulled from and committed to Glance image repository of the cloud infrastructure via Glance CLI [12].

### 6.2.2 Zun API expansion to use images stored at any endpoint

The Zun API parser is modified to create a new *pullimage*command that can take any URL endpoint along with an image name, and download the OCI specification compatible image from that URL. In addition, the single command is able to add that image to the OpenStack image repository - Glance. Once this image is added to the OpenStack image management repository, the command will integrate with Zun to spawn a container from the image and run it.

The existence of this command is imperative to avoid the entanglement of the container migration system with OpenStack image management, and therefore allows users to pull images from any endpoint as required by them. *pullimage* functionality uses python library functions to fetch and download file(container image) over HTTP/HTTPS from any URL end point. The downloaded container image is now added the local OpenStack Glance repository using Glance command line API[12] and is now available to use by Zun from the Glance image repository. With reference to the information mentioned in Section 6.1, these images must be OCI image format compliant and only then can they be successfully spawned into working containers using the Zun API.

## 6.3 Investigate the passing of parameters to RunC container runtime from Zun CLI

Zun is almost completely wrapped around a Docker sublayer for management of containers. Thus, Zun CLI is limited to invoking the relevant docker-py library functions only. Zun does not have any direct interaction with RunC although the containers that are being spawned by Zun via the use of Docker-py library are essentially using a RunC runtime.

The architecture of Zun involves basically 3 layers:

* Layer 1: Zun API for container management
* Layer 2: Docker-py: Python library for Docker container management
* Layer 3: RunC container runtime

In order to passing parameters to the RunC container, we must leverage the Docker layer as per the architecture of Zun in order to avoid future conflicts in design and maintaining consistency. This has been implemented and tested by adding additional commands, parameters and flags to the Zun implementation that are supported by Docker but were not previously a part of Zun.

## 6.4 Design checkpoint/restore functionality in addition to container commit

Zun currently supports the container level image modification which implies that it allows containers that are spawned along with any files on board to be persisted as a new container-image on the host system. However, it does not retain the state of any running process inside the container. This functionality is part of *zun commit*feature provided as part of the standard API.

The use cases enlisted as part of the SmartMe project, and Stack4Things initiative involve the checkpointing of processes and their migration and subsequent deployment on a different node along the network edge. This could not be achieved using current Zun API and thus we implemented a new technique to achieve this. The flowchart in Fig .3 depicts the use of CRIU (Checkpoint/Restore In User-space) tools to achieve the aforementioned functionality.

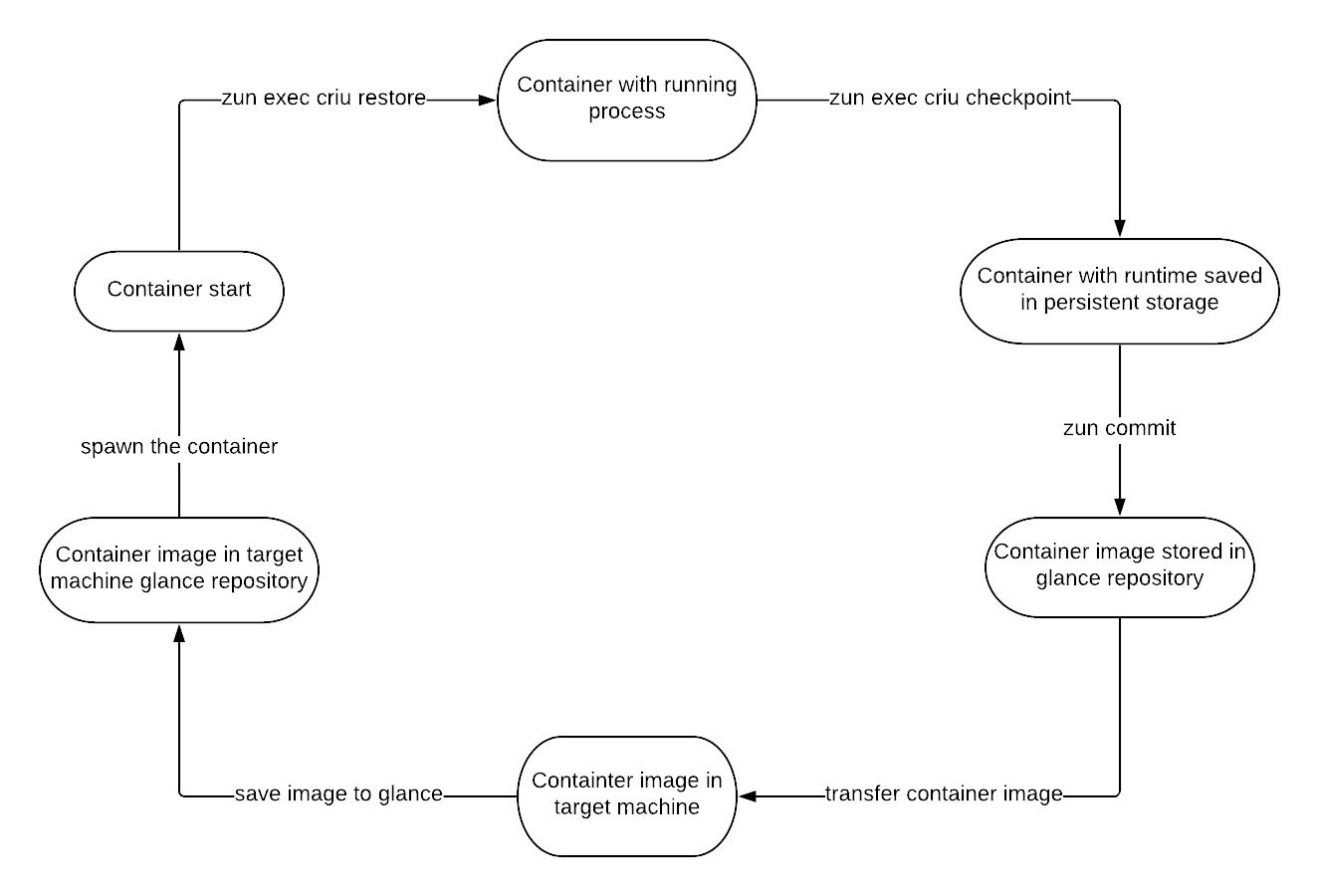
Since the Zun container management uses Docker-Py implementation under the hood, future work may include modifying the Zun API to achieve checkpointing by leveraging docker-py’s capabilities.The following commands checkpoint/restore the specified container:

**Checkpoint:**

sudo docker checkpoint create --checkpoint-dir=<dir-checkpoint-files> <container-ID> <checkpoint-name>

**Restore:**

sudo docker create --name <container-name> <container-image> sudo docker start --checkpoint=<checkpoint-name> --checkpoint-dir=<dir-checkpoint-files> <container-name>



**Fig 3: Migration Workflow**

### 6.4.1 CLI for the Migration workflow

**Transfer container image**

scp -r <path-to-image-dir> <dst>:/<path-to-image>

**Save image to local glance of target machine**

openstack image create --public --container-format <docker/bare/any other-OCI-compatible> --disk-format raw --file <filename> <container-image-name>

**Spawn the container**

Zun run [-i] --image-driver glance <container-image-name> <command>

In order to run a simple bash terminal session on the container by default, replace command with /bin/sh.

-i: Flag is optional and makes the container session interactive.

**Chapter i7.Results and Conclusion**

This section is aimed at explaining how the various problem objectives come together in order to achieve the final goal of setting up the foundation for container migration via OpenStack Zun.

* The research on container-image formats was fundamental to starting our research on container migration technologies since the spawning of containers as well as their migration is directly linked with the container image format through the choice of tools and protocols that would eventually be used for migration. The default container runtime being runC for Zun, the migration techniques researched in the future must be compliant with the image format that is being used in Zun. As the Section 6.1 shows, the findings of the research show that OCI (Open Container Initiative) is the format specification that Zun is currently compliant with and any and all tools and technologies being used must support this image format specification in order to integrate with Zun’s container management.
* In addition to compatibility with Zun, the image-format specification is also directly related to which image repository is used for hosting and subsequent pulling of images on the network edge devices. Since Zun is currently using a RunC based runtime, the images being hosted on our image repository of choice must be compatible with OCI specification. DockerHub proves to be one such choice for the following reasons:
  + Zun itself makes use of a docker-py sublayer (Docker’s python library) to manage containers and is therefore easily integrated with Docker’s commands to pull images from DockerHub.
  + DockerHub hosts images that are in Docker V2 image, which itself is built upon OCI image format[9][10], that allows the images to be used to spawn and run containers without the need of any format based errors or interconversion.
* The use of only DockerHub is a constraint on the scope of the project since it limits the system in terms of file hosting systems. Thus, a new API specification was created in Zun that brings together functionality from Zun and Glance (OpenStack image hosting service) to allow the user to download and spawn containers from OCI compliant images that are hosted at any URL endpoint eg GDrive, AWS S3 etc.
* The use of *pullimage* functionality and subsequent spawning makes it easy to intermediately host images on private servers before being pulled by other edge nodes and spawned as containers.
* The use of generic file hosting services allows us to host and store container images that are not Docker V2 compliant but are OCI compliant and thus can be used successfully with Zun container management service.
* The mechanism for Checkpoint/Restore is the core of the migration process. We were able to successfully create a working flow of commands and calls that are depicted in Section 6.4, 6.5 and Fig 1. By following the mentioned steps we would be able to :
  + Stop a running process on the container and save its runtime and state as a set of files on the container.
  + Commit the files to a new container image that can be migrated and deployed to a new node or allow the current node to be stopped and killed with no loss to the progress of the system.
  + Spawn the committed container image on new/same node and restore the saved process to resume from the same state where it was paused.
  + Critical to the process of migration of running processes
* Since everything after the completion of the above step exists as files in the controlling system, it can be easily moved to any other node through remote sync protocols or existing container migration techniques in order to continue execution of the runtime in the new node.
* The work involved as part of this project is open-source and is available for everyone to view and leverage.

**References**

[1] Carlo Puliafito, Enzo Mingozzi, Carlo Vallati, Francesco Longo, Giovanni Merlino - Virtualization and Migration at the Network Edge: an Overview

[2] Roberto Morabito - Virtualization on Internet of Things Edge Devices with Container Technologies: a Performance Evaluation - May 2017 - IEEE Access PP(99)

[3] Antonio Puliafito et al - Building a Smart City Service Platform in Messina with the #SmartME Project - 2018 32nd International Conference on Advanced Information Networking and Applications Workshops

[4] Antonio Puliafito et al - Stack4Things: An OpenStack-Based Framework for IoT - 2015 3rd International Conference on Future Internet of Things and Cloud

[5] Giovanni Merlino, Dario Bruneo et al - Stack4Things: integrating IoT with OpenStack in a Smart City context

[6] Dario Bruneo - Head in a Cloud: An approach for Arduino YUN virtualization - 2017 Global Internet of Things Summit

[7] Flávio Ramalho, Augusto Neto - Virtualization at the network edge: A performance comparison - 2017 IEEE WoWMoM

[8] Stack4Things - [https://github.com/MDSLab/stack4thing](https://github.com/MDSLab/stack4things)

[9] Docker Registry API to be standardised in OCI - https://blog.docker.com/2018/04/docker-registry-api-standardized-oci/

[10] Docker leads OCI release of V1.0 runtime and image format specifications - <https://blog.docker.com/2017/07/oci-release-of-v1-0-runtime-and-image-format-specifications/>

[11] Image formats support of Glance - <https://docs.openstack.org/python-glanceclient/latest/cli/details.html>

[12] Glance command line API - <https://docs.openstack.org/python-glanceclient/latest/cli/details.html>