Virtual Cluster Manager for Cloud Infrastructure

Pongsakorn U-chupala
Faculty of Computer Engineering
Kasetsart University, Bangkok, Thailand
Email: b5105274@ku.ac.th

***Abstract*—With cloud computing technology, specifically infrastructure as a service (IaaS), it is possible to conveniently deploy a virtual machine on the cloud. There are also a lot of cloud toolkit available to help administrator setup IaaS cloud. However, to deploy a set of virtual machine and configure them to work together as a virtual cluster is still a complicated and tedious task. Most cloud toolkit only individual virtual machine management none offer comprehensive virtual cluster management solution. To address this problem, We propose virtual cluster manager, a unified utility for virtual cluster management. This paper describe design and implementation of virtual cluster manager built on top of existing cloud toolkit as well as the case study of virtual cluster deployment in traditional way compare to using this utility.**

I. Introduction

With cloud computing[1] being a buzzword recently, there are wide range of cloud computing toolkit available nowadays. Each provides services in different abstraction. In this research, We focus on Infrastructure as a Service[2] (IaaS) paradigm. These tools offer the ability to manage resources ubiquitously with united cloud interface.

Even though virtual infrastructure (VI) provide virtually limitless computation capability to user. In reality, performance of each virtual machine (VM) running on the infrastructure is still limited by physical host machine. It turned out that we have already solved this problem with Cluster Computing. We could use this same methodology and create a cluster of VM instead of physical machine. This technology is called Virtual Cluster[3] (VC).

However, most VI toolkit (such as OpenNebula[4] and Eucalyptus[5]) only focus on individual VM management. Even though it is possible deploy VC with these existing tools by configure and deploy each VM manually, There is no convenient way to manage VC as a whole yet. To simplify this process, We propose a utility which manage VC with existing VI toolkit. This utility will provide to user another layer of abstraction which allow user to view the system as a repository for VC.

In this research, We developed CLI Utility called onevc for OpenNebula 3.0 (VI toolkit) which provide the ability to manage VC as a whole. This utility work alongside other built-in CLI utilities seamlessly and provide interface similar to existing built-in utilities for consistency.

Following is the organization of this paper. Section II describes OpenNebula in general and similar utility called "oneservice" and differences between "oneservice" and this research. Section III describe the architecture of VC manager.

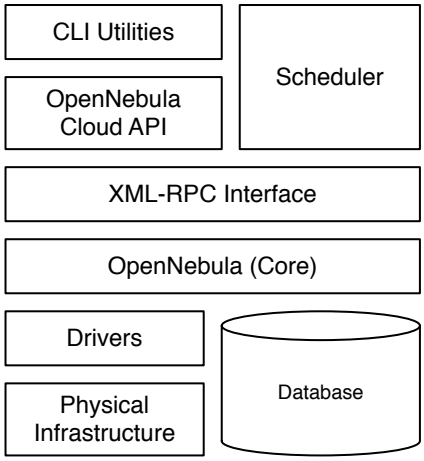

Fig. 1: OpenNebula software stack (simplified)

Section IV describe VC manager implementation based-on OpenNebula VI toolkit. Section V show a case study of VC deployment with existing tools compare to using VC manager. Section VI is the conclusion and future improvement ideas.

II. Related work

A. OpenNebula

In this paper and in our VC manager design, We refer to OpenNebula a lot so it is needed to explained briefly about OpenNebula and its architecture.

OpenNebula is VI manager. It orchestrate VM placement, networking and resources such as disk images and expose unified cloud interface to its user, thus create an Infrastructure as a Service (IaaS). It enable creation of private cloud with existing resources. OpenNebula also feature its modular architecture which allow it to work with wide range of hypervisor. This feature also enable OpenNebula to support creation of hybrid cloud as well.

To archive high level of scalability, OpenNebula architecture is designed to be modular. Using drivers, OpenNebula can be plugged into wide range of datacenter services. The software stack also divided into multiple layers. This approach allow us to modify and extend OpenNebula as needed.

OpenNebula is shipped with a number of CLI utilities such as onevm, onevnet, onehost and onetemplate. Each utility connect to OpenNebula via OpenNebula Cloud API (OCA) and provide abilities to manage many aspect of the cloud.

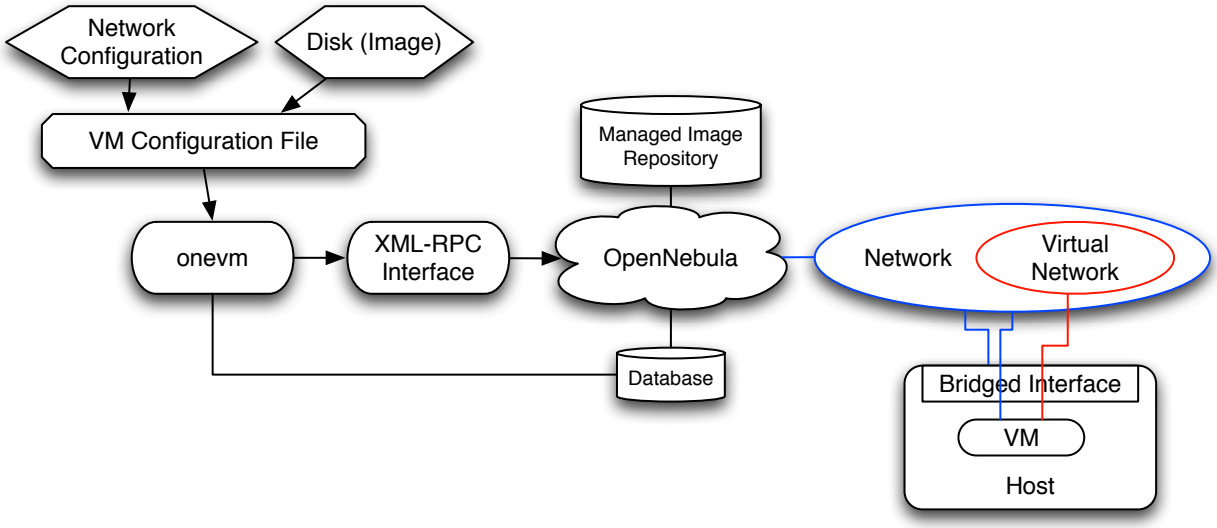

Fig. 2: Onevm operation flow

To deploy a VM with OpenNebula, user need to set up OpenNebula

B. oneservice

There is a concept of service which consist of multiple VM working together being developed for OpenNebula. For an example a web server service could consists of web servers, database servers and load-balancers. oneservice[6] utility allow user to describe service and its dependencies with Service Description Language (SDL). This idea is similar to the virtual cluster but the key difference is service management allow user to manage as group of virtual machine as a whole while virtual cluster management allow user to manage the whole cluster or just a subset of machines in the cluster.

III. Architecture

Similar to how onevm utility of OpenNebula works, to describe a VC to VC manager, We use hierarchical template system. This template system allow user to describe VC attributes in one single template file. This template extend existing VM template system by introducing template for VC. VC manager take VC template as an input and register the VC to database. After that, user can instruct the manager to do management operations on registered VC or each VM type individually. Supported management operations are create, deploy, suspend, stop, resume, delete, list and show.

A. Virtual Cluster Model

In this research, We define VC with a hierarchical model. A VC consists of one or multiple VM Type, served as a grouping method for VM. A number of VM could be instantiate in each VM type. Figure 3 shows hierarchy of VC in this model.

VM types in VC also forming it own hierarchy. We use this hierarchy along with other configurations to determine deployment order of each VM in a VC.

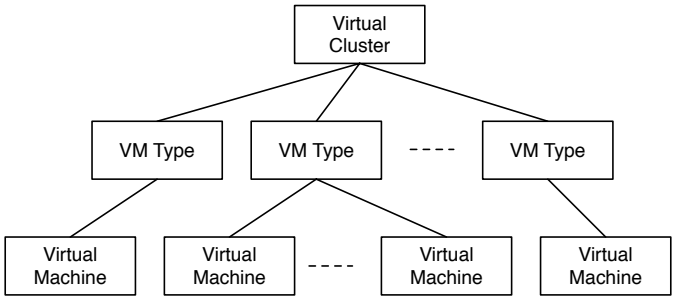

Fig. 3: Hierarchical model of virtual cluster

One example is a VC with one frontend machine and a number of compute node. To represent this VC with this model, We use 2 VM type to represent frontend VC and compute node VM. frontend VM type contains only one VM while compute nodes VM type contains n number of VM with compute nodes VM type being a child of frontend VM type.

B. Virtual Cluster Template

To describe VC with my model, We use template. For each VC, a VC template is used to describe its properties. This VC template syntax is the same with OpenNebula template syntax to preserve consistency with others built-in utilities, to be more familiar to user. VC properties may include information such as name, VC types and number of VM to deploy for each VM type.

To describe each VM type in VC template, we use the same OpenNebula VM template structure. We also allow pointing to external VM template path or template id of template registered in onetemplate built-in utility. If VC template is defined with either of this 2 methods, we also allow VM type template overriding from VC template as well. This allows VM type sharing between many VC template as we could modify VC specific configuration such as networking

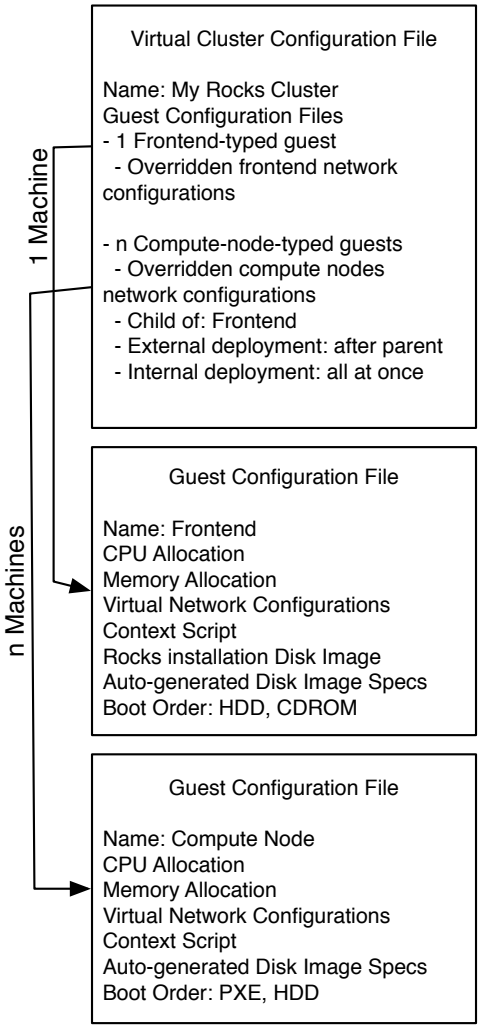

Fig. 4: Example VC template (detail omitted)

as needed.

Each VM type is also configurable with external deployment policy such as “with parent”, “after parent” or “manual” and internal deployment policy such as “all together” or “one by one”. These policies will be use to determine deployment order of each VM for automatic cluster deployment. See figure 4 for an example of Rocks Cluster VC template

To determine deployment order, firstly, we use external deployment policy and VM type hierarchy to determine deployment order of VM type. VM type that don’t have parent will be deploy first. If there are any child VM type of these VM which external deployment policy set to “with parent”, those VM type will be deploy at the same time. After all of these VM type successfully deployed (reached “running” state), their childs will be deploy next, and so on until all VM type is deployed. For deployment order of each machine in the same VM type, we use internal deployment policy.

C. Virtual Cluster State

We assign state to each VM type in a VC and the VC itself in order to keep tracking. During initialization, all VM

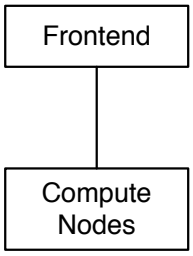

Fig. 5: VM type hierarchy of example VC template

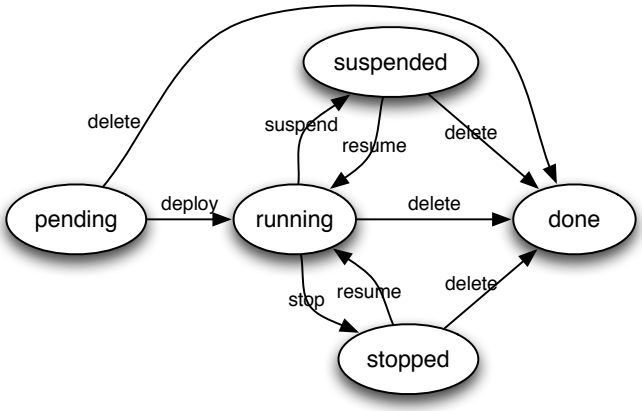

Fig. 6: VC and VM Type state diagram

type will be assigned to “pending” state. State of VM type is shifted along with VC management operations (deploy, suspend, resume, etc.). State shifting is described in figure 6. These states was designed to support OpenNebula’s VM state[7] and bear a similarity with it. In a sense, they are like a simplify version or OpenNebula’s VM state.

When all VM type in a VC is in the same state, VC itself is assigned with that state as well. Otherwise, the VC is assigned “shift” intermediate state to indicate that the state is being shifted and currently inconsistent.

IV. Implementation

VC Manager is implemented as OpenNebula CLI utility and named onevc for consistency with other OpenNebula CLI utilities. onevc is written in Ruby just like other built-in CLI utilities. It works along with other built-in CLI utilities on the same layer and relied on the same OCA interface. However, we need to create some more tables in the database to hold VC information.

A. Database

We uses existing mechanism to manage VM so it use existing `vm_pool` table as well. For VM type, we use existing template system and by storing VM type configuration as template in existing `template_pool`. We define 3 additional tables in the database, `vc_pool`, `vc_node` and `vm_type`. `vc_pool` stores VCs registered to onevc, their configuration from their template and their state. `vm_type` associates each VM type specify in VC template with the actual VM type configuration

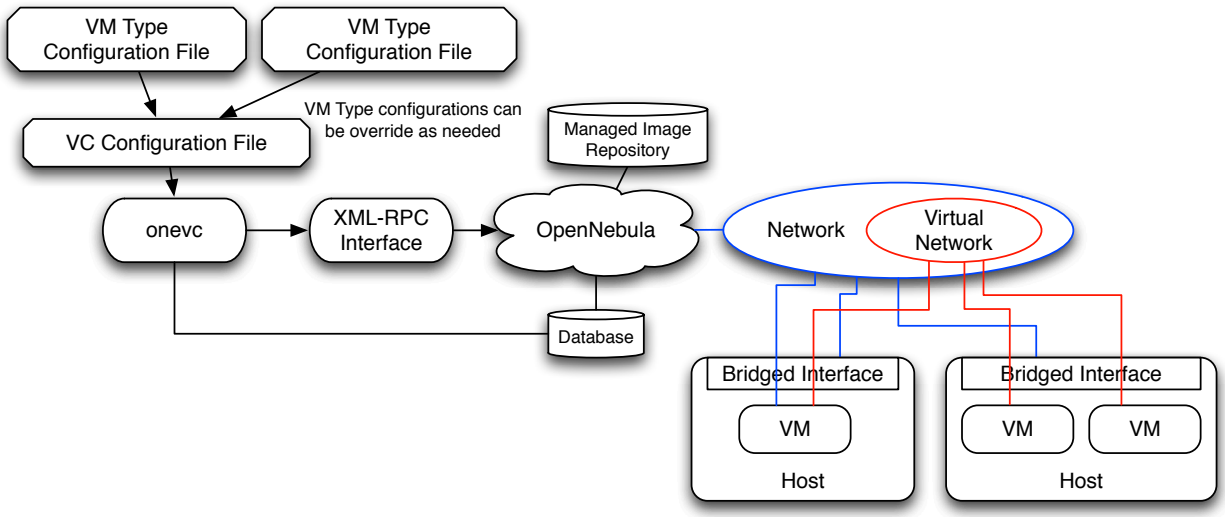

Fig. 7: Onevc operation flow

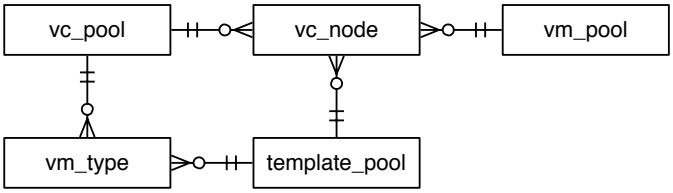

Fig. 8: ER diagram of onevc database

stored in `template_pool` `vc_node` associates each VM with its VC in `vc_pool` and VM type in `template_pool` as well as recording its state. Figure 8 show relationship of between each table.

B. Operation

User supply VC template to onevc in the same fashion as with onevm utility. onevc will then validate the template, put it into onevc database and assign all VM type and the VC itself with "pending" state. After that, onevc will create all VC with OpenNebula and put all of them in OpenNebula's "pending" state, waiting for deployment. User could then deploy registered VC and do management tasks with onevc utility.

onevc utility support these operations.

- `create` register VC template to onevc utility and initialize VC
- `deploy` deploy specify VC or specify VM type according to VM type hierarchy and deployment policy in VC template
- `suspend` suspend VC or specify VM type in reverse order of deployment
- `stop` stop VC or specify VM type in reverse order of deployment
- `resume` resume VC or specify VM type in the same order as deployment
- `delete` delete VC or specify VM type in reverse order of deployment
- `list` list registered VCs and their statuses
- `show` show specify VC information

V. Case Study

To show advantages of our manager design, let's take a look at the following cases.

A. Ease of use

We use number of command used to deploy a VC to demonstrate ease of use of the manager. The less command needed, the more convenient it is. Using the example VC template in Figure 4, We deploy the VC with existing utility and onevc utility.

1) With existing built-in utility: Firstly, We created 2 templates for frontend and compute nodes and register it to onetemplate built-in utility with this command.

```
$ onetemplate create
<path_to_templatefile>
```

Then, We instantiate 1 frontend VM follow by a number of compute node. To instantiate VM with onetemplate, We used this command.

```
$ onetemplate instantiate <template_id>
```

Where `<template_id>` is one template's template id given to registered template. It took 2 commands to register template, 1 command to instantiate fronted and 1 command to instantiate each compute node so We used 3+n commands in total. We also need to maintain each template individually and make sure that configurations in both template worked together.

2) With onevc utility: Let the name of VC configuration file is `myrockscluster.one`, Firstly, user need to register the template with this command

```
$ onevc create myrockscluster.one
```

After registration, user can use this command.

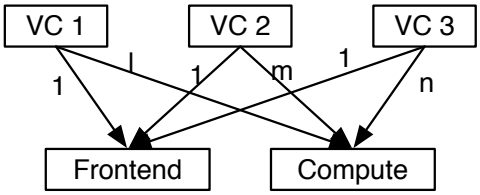

Fig. 9: VM type sharing between multiple VC

```
$ onevc deploy myrockscluster
```

According to deployment policies in the template, onevm will deploy frontend VM first. After it reach "running" state, it will deploy n number of compute nodes at once. User can choose to deploy each VM type manually by using these following commands.

```
$ onevc deploy myrockscluster frontend
$ onevc deploy myrockscluster computenodes
```

Note that if fronted VM type isn't at "running" state when user manually deploy compute nodes then onevm will try to move frontend to "running" state by deploying or resuming as necessary.

In total, We used 2 commands for automatic deployment and 3 commands for manual VM type deployment regardless of how many compute nodes are there in the cluster.

B. Template Reusability

With VM type hierarchical structure. It is possible to reuse VM type across multiple VC. To demonstrate this point, we use number of template needed as a measurement. In this case, we will deploy 3 virtual Rocks Clusters[8] with different number of compute node for each VC (l, m and n compute nodes).

1) With existing built-in utility: For each VC, we need 2 VM templates file for frontend node and compute node. Even if some part of frontend node and compute node share the same configuration, there are differences between each template (such as network configuration to use) so configuration file need to be duplicated and maintained individually. In total, 6 VM templates are needed. Each node also need to be deployed manually so OpenNebula can not keep track of number of compute node for each VC at all. Figure 9 show VC Hierarchy of this configurations.

2) With onevc utility: Because frontend node and compute node for each VC bare similarity, we define them with single VM type template for each one. Each VC can just point to these 2 VM type template, specify number of compute node needed and override unique configuration for each VC. We need 3 VC template and 2 VM type template so there are 6 templates needed in total.

VI. Conclusion and Future Work

In this paper, we describe a convenient utility for VC management with existing VI toolkit. Instead of management each VM in a VC individually, this utility offers a method to manage VC as a whole. In order to flexibly describe VC, We use hierarchical VC model along with the template system. This template system also offer policy configurations which to be used alongside VM type hierarchy to determine deployment order. With this architecture, We implemented onevc utility for OpenNebula and show how to deploy typical VC with this tool compare with existing built-in utility. It is clear from the result that this utility made VC management simpler.

However, We only look into VC management and leave VC template management totally to user. For this issue, We could create another utility to manage VC template. This utility could work in the same manner as onetemplate built-in utility of OpenNebula manage VM template but with VC template instead.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[2] A. Ranabahu and E. Maximilien, "A best practice model for cloud middleware systems," *Towards bes*, p. 41, 2009.

[3] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual clusters for grid communities," in *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, vol. 1, may 2006, pp. 513 – 520.

[4] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[5] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009, pp. 124–131.

[6] W. Iqbal. (2010, 11) Opennebula service manager. [Online]. Available: http://opennebula.org/software:ecosystem:oneservice

[7] Managing virtual machines 3.0. [Online]. Available: http://opennebula.org/documentation:rel3.0:vm_guide_2

[8] P. M. Papadopoulos, M. J. Katz, and G. Bruno, "Npaci rocks: tools and techniques for easily deploying manageable linux clusters," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 7-8, pp. 707–725, 2003. [Online]. Available: http://dx.doi.org/10.1002/cpe.722