Politecnico di Milano

Facoltà di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Towards Virtual Machine Consolidation in OpenStack

Advisor: Sam Jesus Alejandro Guinea Montalvo

Master thesis by:
Giacomo Bresciani matr. 804979
Lorenzo Affetti matr. 799284

Academic Year 2013-2014

dedica. . .

Ringraziamenti

Milano, 1 Aprile 2005

Affear

Table of Contents

List of Figures **ix**

List of Tables **1**

1 Introduction **3**

2 State of the art **5**

2.1 Introduction 5

2.2 Virtual Machine consolidation 6

2.3 Cloud test environments 6

2.3.1 Vagrant 6

2.3.2 Chef 7

2.3.3 Puppet 10

2.3.4 Docker 12

2.3.5 Dockenstack 13

3 aDock **15**

4 FakeStack **17**

5 Simulation tools **19**

6 Nova Consolidator **21**

7 Evaluations **23**

8 Conclusions and future works **25**

Appendices **27**

Bibliography **29**

List of Figures

2.1 Hypervisor and Docker Engine 13

List of Tables

Chapter 1

Introduction

Chapter 2

State of the art

2.1 Introduction

At the beginning of the development of the thesis we were mainly focused on the implementation of a module for OpenStack that would allow to implement different consolidation algorithms and test them to see their real impact on a cloud system in terms of resource allocation. During the first phases we faced with the problem of running, testing and benchmarking our code in an OpenStack environment: to deal with aspects like scheduling, VM placement, and consolidation we needed an highly configurable system that would allow us to run simulations and benchmark to evaluate the goodness of our solution. We wanted it to be fully customizable to match different requirements and let the user customize a lot of aspect such as the structure of the environments, the number of compute nodes (the nodes that host the Virtual Machines), their fake characteristics or the OpenStack services to run. Secondly we needed a way to automatically simulate, in a repeatable way, the workload generated from user applications that normally run on an OpenStack installation. At last we realized that it would be very useful to show the real time data of the simulations to analyze the behavior of the system in different configurations. Therefore we decided to develop aDock, a suite of tools for creating performant, sandboxed, and configurable cloud infrastructure experimentation environments that developer, sysadmins and researchers can exploit to access a fully functional cloud installation of OpenStack. For that reason this chapter is divided into two sections that describe the state of the arts of Virtual Machine

consolidation and of Cloud test environments.

2.2 Virtual Machine consolidation

In the cloud world is fundamental, as in any engineering field, to be able to test environment configurations and algorithms, both to analyze the behavior of tested code that integrates with a real environment and to benchmark and collect data for researches and experimentations. Unfortunately it can be expensive and complex to create and manage a cloud test environment in terms of time, resources and expertises, especially if the hardware resources like server machines or network infrastructures are limited. Fully understand and handle an OpenStack installation is not easy, especially for non sysadmins **#TODO** (Find a better way to say it) like developers or researchers, it has infact an high learning curve and often it is necessary a lot of time to achieve a good and desired result. To reduce the impact of these complications are available some tools that make the process of setting up a cloud infrastructure experimentation environment more easy and manageable. The three main ones are Chef, Puppet, and Dockenstack. In order to achieve the goal of running the desired sandboxed environment on limited hardware resources the three solution are used together with two core technologies for the cloud world, Vagrant and Containers (Docker), both outlined below.

2.3 Cloud test environments

2.3.1 Vagrant

Vagrant[1] is a virtualization framework for creating, configuring and managing development environments, written in Ruby, that allows to virtual development environments. It is a wrapper around virtualization software such as VirtualBox, KVM, VMware and could be used together with configuration management tools such as Chef and Puppet. Thanks to an online repository [2] it is possible to automatically download a Vagrant Box and run it with a single

[1] `www.vagrantup.com`
[2] `www.vagrantcloud.com`

command: `vagrant up vagrant-box-name`. It is also possible to create and configure custom Vagrant Box by simply writing a Vagrantfile:

```
box       = 'trusty64'
url       = 'http://files.vagrantup.com/precise32.box'
hostname = 'customtrustybox'
domain    = 'example.com'
ip        = '192.168.0.42'
ram       = '2048'

Vagrant::Config.run do |config|
  config.vm.box = box
  config.vm.box_url = url
  config.vm.host_name = hostname + '.' + domain
  config.vm.network :hostonly, ip

  config.vm.customize [
    'modifyvm', :id,
    '--name', hostname,
    '--memory', ram
  ]
end
```

Provisioners in Vagrant allows to automatically install and configure software in a Vagrant Box as part of the "vagrant up" process, therefore you can start with a base Vagrant Box, adapt it to your needs and eventually share it with other developers who can reproduce the same virtual development environment. Vagrant in combination with configuration management software such as Chef and Puppet is used to create repeatable and easy to setup development and test environments that rely on Virtual Machines.

2.3.2 Chef

Description Chef[3] is a configuration management tool used to streamline the task of configuring and maintaining servers in a cloud environment and can be integrated with cloud-based platforms such as Rackspace, Amazon EC2, Google Cloud Platform, OpenStack and others. It is written in Ruby and

[3] `www.chef.io`

Erlang and uses a domain-specific language (DSL)[4] for writing configuration files called "recipes". "Recipes" are used to define in a declarative way the state of certain resources and define everything that is required to configure different parts of the system: they can contain the definition of software that should be installed and all the required dependencies, services that should be running or files that should be written. Given a "recipe" Chef ensures that all the software is installed in the right order and that each resource state is reached, eventually correcting those resources in a undesired state; "recipes" can be collected into "cookbooks" to be more maintainable and powerful. In addition Chef offers a centralized hub, called Chef Supermarket[5], that collects a large number of "cookbooks" from the community freely downloadable.
A base installation of Chef is composed by three main components, a `chef-server` that orchestrates all the Chef processes, multiple `chef-client` found on all the servers, and the user workstation that communicates with the Chef Server to launch commands.
To simplify the communication with the `chef-server` Chef provides a command-line tool called Knife that helps users to manage nodes, "cookbook" and "recipes", and the majority of possible operations.

Chef and OpenStack Chef and OpenStack can be combined and used together in different ways, many of which have a different goal compared to our thesis. Is it possible in fact to deploy and manage a production OpenStack installation running on multiple servers and supervised by a Chef Server using the subcommand `knife openstack` to control the OpenStack APIs through Chef and thus instantiate new physical servers with a chef-client installed or turn them off (`knife openstack server create / delete`). In this situation you can achieve a "1 + N" configuration that is one OpenStack Controller and N OpenStack Nodes, and the OpenStack services are predefined and you cannot configure an ad hoc configuration. Therefore an "All-in-One Compute" configuration can be chosen where all the OpenStack services are installed on a single node.
These configurations can be achieved with the help of Vagrant that will cover all the steps to install OpenStack on a virtual machine and configure all its

[4] A programming language specialized to a particular application domain.
[5] `supermarket.chef.io`

services (excluded Block Storage, Object Storage, Metering, and Orchestration). Within the OpenStack chef-repo[6] there is a Vagrantfile that configure a VirtualBox virtual machine that will host and All-in-One installation. Here is a part of it:

```
1 machine 'controller' do
2   add_machine_options vagrant_config: controller_config
3   role 'allinone-compute'
4   role 'os-image-upload'
5
6   chef_environment 'vagrant-aio-nova'
7   file('/etc/chef/openstack_data_bag_secret',
8        "#{File.dirname(__FILE__)}/.chef/encrypted_data_bag_secret")
9   converge true
```

Of course it is possible to setup a "1 + N" configuration using different Vagrantfile to create and configure one VM to host the Controller and N VMs to host the Compute nodes. However it is unlikely to succeed in running a lot of VMs on the same host especially if they will contain a fully functional OpenStack installation as a Virtual Machine typically require a significant amount of resources to operate.

#TODO ... Insert timing to install OpenStack with Chef...

Pro and Cons Chef, as seen, is therefore a very powerful tool to create, manage and configure cloud environments and offers a lot of functionalities to structure the desired architecture. In combination with Vagrant is also useful to setup test environments for development or research purposes.
However, with regard to this last aspect, it has several limitations:

- *Heaviness*: due the greed of resources of a Virtual Machine is very difficult to achieve a "1 + N" configuration for development or research purpose on a single machine. On the other hand the "All-in-One Compute" solution that allows a full OpenStack installation on a single Virtual Machine is very simplistic and doesn't represent a real environment setting as it runs all the OpenStack services on a single node.

[6] https://github.com/stackforge/openstack-chef-repo

- *Lack of customization*: at the state of the art all of the described solutions install both the Controller node and the Compute node with a predefined set of installed service (in particular are installed all the OpenStack service excluded Object Storage, Metering, and Orchestration) so it is not possible to setup the environment with more or less services or new ones (as in our case).

- **#TODO** think others...

2.3.3 Puppet

Description Similarly to Chef (described in section 2.3.2) Puppet[7] is a configuration management system that allows you to define the state of a cloud infrastructure and then it automatically enforces the correct state.
Puppet uses a declarative model where are defined the resource states and (likewise Chef) it's manifest files are written in a Ruby-like DSL. In these manifest files are defined the configurations, the nodes and how the configurations apply to nodes. Again Puppet will take care of ensuring that the system reaches the expected state. All these files are enclosed in "modules", a self-contained bundles of code and data easy to share and reuse. There are a large amount of them on Puppet Forge[8] repository.
Puppet is structured in master-slave architecture: the master (that can be one or machines) serves the manifests and the files, and the clients polls the master at specific intervals of time to get their configurations so that the master never pushes nothing to them. This structure uses the "Puppet master" and "Puppet agent" applications.

Puppet and OpenStack As seen for Chef, Puppet can be very useful when dealing with OpenStack installation and maintenance. To configure and deploy an OpenStack infrastructure with the help of Puppet exists a set of "modules" freely downloadable from Puppet Forge that simplifies most of the operations such as OpenStack instances provisioning, configuration management and others. The module is `puppetlabs-openstack`[9] and allows to

[7] `www.puppetlabs.com`
[8] `forge.puppetlabs.com`
[9] `github.com/puppetlabs/puppetlabs-openstack`

deploy both a multi-node and an all-in-one installation. Compared to Chef, with regard to OpenStack, Puppet is a bit more flexible because it allows you to control in more details the OpenStack services installed on every node; for example you can, combining the following instructions, in the Puppet's manifest file of a node different results can be achieved:

Controller node:

```
1 node 'control.localdomain' {
2   include ::openstack::role::controller
3 }
```

Controller node:

```
1 node 'storage.localdomain' {
2   include ::openstack::role::storage
3 }
4
5 node 'network.localdomain' {
6   include ::openstack::role::network
7 }
8
9 node /compute[0-9]+.localdomain/ {
10  include ::openstack::role::compute
11 }
```

Obviously, in the same way for Chef, it is possible to configure multiple nodes to run in multiple Virtual Machines configured and launched with Vagrant and deploy the various OpenStack components with `puppetlabs-openstack`. This solution, as said earlier, is difficult to achieve on a machine with a limited amount of resources, and also on more powerful server is however slightly extensible.

Pro and Cons Puppet is an extremely powerful and mature tools for automated cloud infrastructure deploying: it streamlines the entire process automating every step of the software delivery process.
However from our point of view it is more relevant to how it behave when a single developer or a researcher needs to deploy a cloud infrastructure within a single machine with limited amount of resources (a development workstation for example) or he/she has a little sysadmin skills (**#TODO** find better way)

and want to run test for developing or researches purposes. With regard to this aspect Puppet used with Vagrant has some key limitations:

- *Heaviness*: A single Virtual Machine need generally a remarkable amount of resource, especially to host an OpenStack installation; for this reason it is very unlikely to be able to run the number of Virtual Machines needed to deploy a realistic multi-node installation of OpenStack on a single machine without compromising its usage. The all-in-one instead is frequently not sufficiently realistic, especially when testing algorithms or portion of code that involves multiple nodes.
- **#TODO** think others...

2.3.4 Docker

Docker[10] is an open platform for developers and sysadmins to build, ship, and run distributed applications. Its core is the Docker Engine: it exploits Linux containers to virtualize a guest Operating System on an host one avoiding the considerable amount of resources necessary to run Virtual Machines.
The main difference between the Docker solution and the Virtual Machine one lies in the way in which the hypervisor and the Docker Engine manage the guest Operating System. A Virtual Machine, as shown in figure 2.3.4 on page 13, hosts a complete Operating System including application, dependency libraries, and, more important, the kernel; otherwise the Docker Engine runs as an isolated process in userspace on the host operating system and allows all the guest containers to share the kernel. Thus, it enjoys the resource isolation and allocation benefits of Virtual Machines but is much more portable and efficient; for our goals this aspect represents the possibility to run at the same time a larger number of containers compared to what we are able to achieve with Virtual Machines and also to ship pre-built images of our modules **#TODO** Better way... .
To configure and then build a container image you have to write a Dockerfile that is a text document containing all the commands which you would have normally executed manually in order to take the container to the desired state and then call `$ sudo docker build .` from the directory containing the file.

[10] `www.docker.com`

The command `$ sudo docker run` will finally launch the container that will be almost instantly running.
Moreover Docker offers an online platform called Docker Hub[11] where you can upload both Dockerfile and pre-built container images to streamline the sharing process.

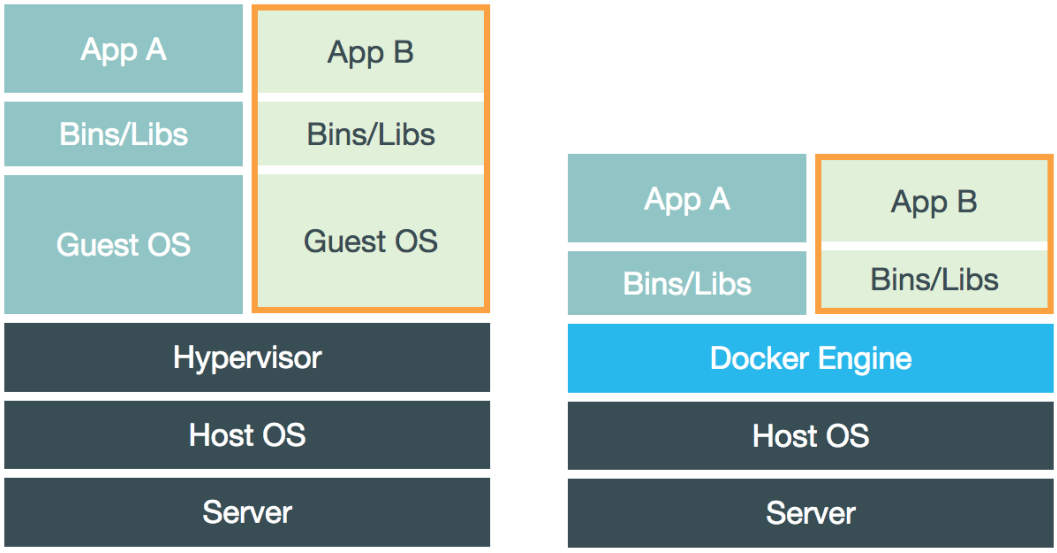

Figure 2.1: Hypervisor and Docker Engine

2.3.5 Dockenstack

One of the first an attempts to create a cloud test environments based on OpenStack and Docker is a Dockenstack[12]. It is an independent and not actively supported project, but is a good starting point to show the potential derived from using Docker.
The project is basically composed by a Dockerfile and a bunch of scripts that will setup and configure an OpenStack installation using DevStack (**#TODO** shall we talk about it in the intro?) in a Docker container.
A pre-built image is available on Docker Hub, so with the command `docker run -privileged -t -i ewindisch/dockenstack` Docker will automatically download and run the container.

[11] `hub.docker.com`
[12] `github.com/ewindisch/dockenstack`

It is however a very basic solution as it only allows all-in-one OpenStack installation

Chapter 3

aDock

Chapter 4

FakeStack

Chapter 5

Simulation tools

Chapter 6

Nova Consolidator

Chapter 7

Evaluations

Chapter 8

Conclusions and future works

Appendices

Bibliography