Automation of the Infrastructure and Services for an OpenStack Deployment Using Chef Tool

Eduard Luchian, Cosmin Filip, Andrei Bogdan Rus,
Iustin-Alexandru Ivanciu, Virgil Dobrota
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
Corresponding author: Virgil.Dobrota@com.utcluj.ro

***Abstract*— The legacy automatic deployments of OpenStack are mostly focused on specific packets and configurations. This paper investigates the possibility to automate the deployment of both OpenStack and the underlying architecture at once (minimum one Controller Node and one Compute Node). The Chef tool proved to be a good candidate, together with Vagrant (for automatic creation and configuration of virtual environments) and VirtualBox (as hypervisor). Comparing to the manual installation, the time needed to have a fully functional solution decreased from more than 30 minutes/node (or hours for unexperienced or unlucky system admins) to 10 minutes/node (or even just 2 minutes/node if the Vagrant box has been executed before). As a major drawback, Chef does not know how to handle prompts from installed packages which imposed developing a workaround.**

Keywords—Chef; OpenStack; Vagrant; VirtualBox

I. INTRODUCTION

In the recent years the industry showed a rapidly growing interest regarding the cloud computing. This explains the need to rapidly scale infrastructures and automate deployments. While the cloud concept has many services to provide, the most sought is Infrastructure as a Service (IaaS). Although providers face many difficulties when building a cloud infrastructure (because of the managing both physical and virtual resources) the infrastructure deployment is the most intricate part. In cloud computing, the deployment and maintenance of those resources should be managed and automated so that the resources can be rapidly provisioned and released with minimal management effort coming from the service provider.

A forever present concern is managing infrastructure scalability which complexity varies greatly depending on the type of application. A cloud infrastructure is mainly considered to be a suite of hardware and software that enables the five essential characteristics of cloud computing [1]: a) on-demand self-service; b) broad network access; c) resource pooling; d) rapid elasticity; and e) measured service.

Both the Physical Layer and the Abstraction Layer are considered to be included in a cloud infrastructure. The first refers to the hardware resources that provide the necessary support for the cloud services. The second represents all of the software that is deployed across the Physical Layer. When referring to cloud computing, it is also helpful for the admin to design in advance whether a cloud is deployed within an organization or more broadly. Four main deployment models can be distinguished: a) private; b) public; c) community; and d) hybrid [2].

OpenStack is a large distributed IaaS-based cloud [3]. It is an open source project with a rapid release/upgrade circle. This cloud software suite is well known because of its compliance that means is critical for speed, reliability, etc. [4]. The goal of the OpenStack project is to let the companies build Amazon-like cloud services in their own data centers. In order to provide the latter stated goal a set of basic components configuration is needed.

While there are various tools and setups available for an automatic deployment of OpenStack cloud environment they are specialized in the specific packages and configurations regarding the cloud stack [12], [13]. All of them require an infrastructure to be preconfigured, the so called prerequisites. To the best of our knowledge there is no free suite that allows the automation of both the OpenStack environment and the infrastructure beneath. In order to realize this task a large set of automation scripts and configurations were needed.

This paper aims at assessing the levels of automation available for an OpenStack deployment, starting from the fact that a fully operational cloud testbed is not a trivial task. The starting project was actually another attempt to simplify called OpenStack-Liberty-Install-Script [5]. The whole work can be taken to a higher level of automation thanks to Chef and its capabilities. The process involves several steps and configurations that apart from being time consuming it also prone to error due to misconfiguration.

The rest of the paper is organized as follows: Section II presents a comparison between different automation tools. Section III approaches the prerequisites for the solution and an overview of the finished deployment. The experimental results are presented within the next section, followed by conclusions and further work.

II. AUTOMATION TOOLS

Over the time a lot of innovative tools have been developed to automate the installation of an OpenStack-based cloud. Some

of them are suitable for handling infrastructure automation while others tackle issues related to pure automation. Herein we investigated three of the most popular automation tools: a) Salt Stack; b) Puppet; and c) Chef.

A. *Salt Stack*

The first one proposes a new approach to infrastructure management building on a dynamic communications bus [6]. Salt can be used for data-driven orchestration, remote execution for any infrastructure and configuration management for any application stack. The objective is to enable high-speed communication with large numbers of systems, capable of maintaining remote nodes in defined states. Being written in Python, Salt is easy pluggable. The parallel execution of remote commands using AES encrypted protocol is one of its best features. The architecture is built around masters and minions, as depicted in Fig. 1.

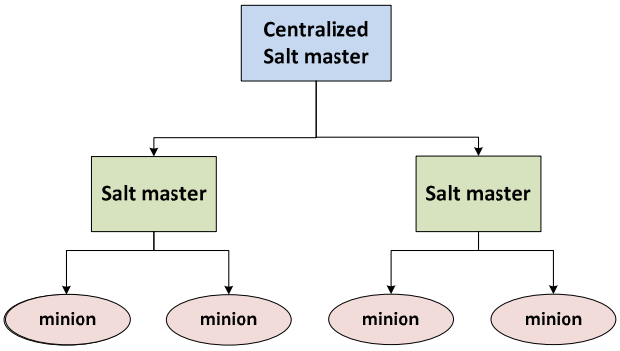

Fig. 1. Salt Architecture

The main components of a Salt configuration are the following:

- *Centralized salt master*: It runs tasks for the Master (authenticating minions, communicating with connected minions and salt CLI)
- *Salt Master*: It controls Minions.
- *Minion*: It receives commands from the Master, runs jobs and communicates results back to master

Salt has no default method for OpenStack deployment but its community is quickly growing so it presents a promising future but not enough for the moment.

B. *Puppet*

This is an open source client-server configuration management tool [7], in which clients periodically poll server for desired state and send back status reports to the server (master). It works in a highly distributed way for quickly upgrade and manage nodes, all throughout Puppet lifecycle. It is written in Ruby, using DSL for writing manifests and ERB for templates. It is very easy to add and remove nodes, each cluster may having multiple masters. Tasks are executed only if a node state does not match required configuration. The working architecture is presented in Fig. 2.

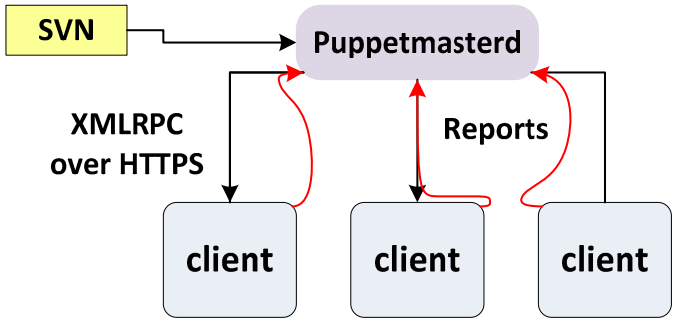

Fig. 2. Puppet Architecture

- *Puppet Master:* It receives queries and status reports from the Puppet Agents (Clients) and it sends commands for them.
- *Puppet Agent*: It sends queries to the Puppet Master, it runs Master's commands as needed, and it reports results back to the master.

C. *Chef*

This is a system and cloud infrastructure automation tool for installing applications and software to bare metal, virtual machine, and container-based clouds. The configuration is written in Ruby DSL, and it uses the concepts of organizations, environments, cookbooks, recipes and resources, all driven by supplied or derived attributes [8]. The tool has a set of control parts that work together to provide its functionality. Chef Workstation is used to control the deployment of configurations from the Chef Server to Chef managed nodes. Nodes are bootstrapped with agents and pull configurations from the server. The core is developed in Erlang and is designed to provide scale to tens of thousands of servers. It is developed around an infrastructure-as-a-code model with version control integral to the workstation. Directives run top to bottom, and the cookbooks, does not matter how many times are running, generate the same result. Note that the cookbooks are the main configuration blocks containing the specific information for the desired state of the node (i.e. recipes, metadata, attributes, resources, templates, libraries etc.). Throughout the study of the tools, a set of advantages and disadvantages were determined, as in Table I. We decided herein to follow Chef's approach.

TABLE I. COMPARISON OF AUTOMATION TOOLS

Tool	*Pros*	*Cons*
Salt Stack	• Easy to start, install, deploy and manage • Highly scalable architecture • Python base language	• Documentation is hard to understand • OpenStack support is not yet mature • Not a great support for non-Linux operating systems
Puppet	• Automation of compliance across environment, high value to enterprise • Web UI & Reporting tools	• Hard to learn for new users • Difficult to scale

Tool	*Pros*	*Cons*
Chef	• Large community of cookbooks and development tools • Ability to handle physical, virtual, and containers deployments. • Provides hosted services like Chef • Excellent at managing operating systems	• Complex to set up the whole stuff because it requires good understanding of Ruby • Huge amount of documentation. • It requires an agent to be installed and pulled configuration

III. PREREQUSITES AND INFRASTUCTURE OVERVIEW

The main idea is to automate all the possible tasks that were needed in order to use the existing scripts to deploy OpenStack and to provide a customizable toolset. We started with the virtualization part and ended with the automation of scripts. In order for this to be possible a set of software solutions were needed to run on a Windows machine:

- *Chocolatey* is a tool that permits the installation of the most known pieces of software by just using `cmd` [9]. There is a dedicated repository in which all the installation kits of this software are being kept. It is somehow the same thing as "apt-get" is for Debian based systems. The tools that were installed using Chocolatey are Vagrant and VirtualBox. The installation is a silent one, which means no additional specifications or prompts are required during this process.
- *VirtualBox* is a hypervisor for x86 and x64 architectures and it is a cross-platform virtualization application (i.e. can be used on server, desktop or embedded) [10]. This tool extends the capabilities of the existing computer so that it can run multiple operating systems. Its installation was performed herein through Chocolatey.
- *Vagrant* is a tool that automatically creates and configures virtual environments [11]. It is somehow complementary to VirtualBox. It can be used with other virtualization tools such as VMware, Hyper-V or KVM. Vagrant is also considered a configuration management tool, like Chef, Puppet or Salt.

We selected VirtualBox because almost all the features are free and can be implemented by any user. The alternative would be VMware in which case only few of the features provided are free of charge, and even worst they are not cheap at all.

In this work we propose the following solution:

- *Step 1:* Installation of Chocolatey
- *Step 2:* Installation of VirtualBox and Vagrant using Chocolatey
- *Step 3:* Writing the configuration files for Vagrant
- *Step 4:* Writing the configuration files for Chef
- *Step 5:* Installation of virtual machines using Vagrant and Chef
- *Step 6:* Automatic deployment of OpenStack using Vagrant and Chef

Let us discuss now the simple architecture, available on all platforms (Windows, Linux, iOS etc.), as depicted in Fig. 3. The cloud infrastructure can be easily maintained with Chef, because this tool transposes the infrastructure into code. The processes could be automated, tested and reproduced in a very easy manner.

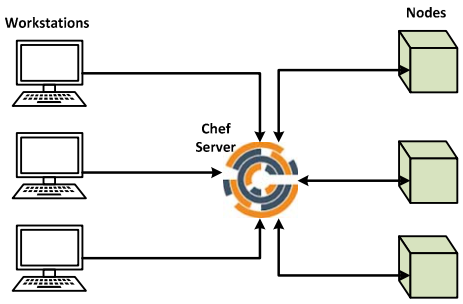

Fig. 3. Chef mini architecture

There are three main components:

- *Chef Server* is the most important part because it stores the whole configuration data for all nodes. It has also the role to administrate the access rights.
- *Chef Workstation* is the place where the cookbooks, recipes and all of the configuration parts are created (to be deployed to Chef Nodes through Chef Server). The code can be tested before deploying with a tool called Kitchen. The cookbooks created can be used privately or, as in Vagrant boxes, they can be uploaded.
- *Chef Nodes* are the places were the cookbooks, recipes and all of the configuration parts are stored. Once all the pieces of software are in place, the configuration for the actual deployment needs to be set. Chocolatey and Virtual box do not require any additional configuration.

For Vagrant the configuration regarding the virtual machines attributes and the network infrastructure in VirtualBox is required. The file that will contain all the specifications can be created by introducing the command `vagrant init`. The file will be created in the home directory for vagrant but this file can be moved anywhere. The specific configuration for the current deployment is presented in Fig. 4.

Note that for the current deployment VM name, host name, network interfaces addresses and hardware configurations are included in the file for automated deployment. The Chef internal configurations are somewhat more complex and numerous, but can be seen as under a single "umbrella" called Cookbooks.

With all the configurations prepared, let us now deploy in a fully automated manner the testbed presented in Fig. 5, orchestrated by OpenStack.

```
Vagrant.configure(2) do |config|

config.vm.define :controller do |controller|
  controller.vm.hostname = 'controller'
  config.vm.box = "cfilip/Ubuntu-OS-Controller"
  controller.vm.network :private_network, ip: '10.0.0.11'
  controller.vm.network :private_network, ip: '10.0.1.11'

  controller.vm.provider "virtualbox" do |vb|
      vb.customize ["modifyvm", :id, "--cpus", 1]
      vb.customize ["modifyvm", :id, "--memory", 4096]
      vb.customize ['modifyvm', :id, '--natnet1', '10.3.3.0/24']
    end
 end

  config.vm.define :compute1 do |compute1|
  compute1.vm.hostname = 'compute1'
  config.vm.box = "cfilip/Ubuntu-OS-Compute"
  compute1.vm.network :private_network, ip: '10.0.0.21'
  compute1.vm.network :private_network, ip: "10.0.1.21"

  compute1.vm.provider "virtualbox" do |vb|
      vb.customize ["modifyvm", :id, "--cpus", 1]
      vb.customize ["modifyvm", :id, "--memory", 4096]
      vb.customize ['modifyvm', :id, '--natnet1', '10.3.3.0/24']
    end
 end

end
```

Fig. 4. Vagrant file

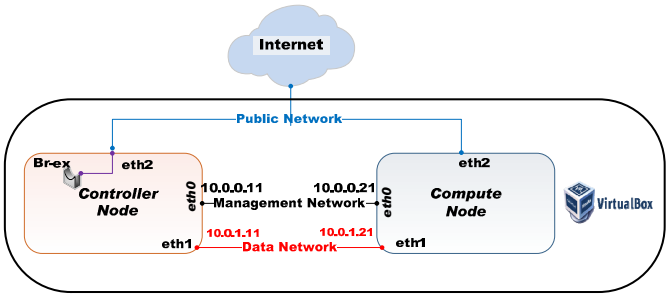

Fig. 5. OpenStack cloud architecture

The cloud has two main components, the Controller Node and the Compute Node, each with three network interfaces: a) Public – allowing Internet access for the machines and instances within OpenStack (Controller); b) Management – for managing messages from the Controller to the Compute; Data – for the traffic of the instances reaching services on the Controller or the Internet through it.

- *Controller* provides the central management system of OpenStack. It manages databases with user and instances information, authentication and authorization for identity management and image storage. The cloud controller allows users to control OpenStack services through a user dashboard (web-based interface).
- *Compute Node* represents the resource core of OpenStack, providing processing, memory, network and storage. It needs a powerful CPU to support virtualization and a so-called hypervisor to create, manage and monitor virtual machines (VMs).

IV. Experimental Results

The tests started with the creation of the two virtual machines (Controller, Compute Node) using Vagrant, according to the file presented in Fig. 4. Note that if `vagrant up` is executed for the first time on a box, the total amount of time that is spent until the virtual machine is ready to be used depends on the Internet connection. The box is first downloaded from Vagrant Cloud and then installed in VirtualBox. The total time spent in this case was about 10 minutes. If the box was already downloaded, if we rerun `vagrant up` command the time decreased a lot (we completely finished the installation in about 2 minutes). The results for the Controller machine are presented in Fig. 6.

```
> vagrant up
Bringing machine 'controller' up with 'virtualbox' provider...
=>...
=>...
=>...
=>controller: Machine booted and ready!
=>controller: Checking for guest addition in VM...
=>controller: Setting hostname...
=>controller: Configuring and enabling network interfaces...
=>controller: Mounting sharedfolders...
=>controller: done...
```

Fig. 6. Succesful vagrant up for Controller

For both virtual machines the total amount of time required for a complete installation was about twenty minutes. If the boxes were downloaded in advance the time decreased to 4 minutes. Note that in both cased the amount of time is significantly reduced compared to a standard manual installation (which exceeds 30 minutes).

After the completion on these two nodes, in order to make of this architecture a fully functional OpenStack-based cloud, we had to run the Chef client. As during the execution time the tool performed automatically a lot of Linux commands, it was rather difficult to follow them step by step. This was solved by introducing some markers into the scripts for the key steps. The result of Chef Client setup can be seen at the end of its running when all the recipes are applied. It will be marked by a green line, followed by a message that tells the script was successfully applied to the node. The result of the setup performed by the Chef Client in case of for the Controller node are presented in Fig. 7.

```
+----+------------------+------------+----------+-------+----------------------------+-----------------+
| Id | Binary           | Host       | Zone     | Status | State | Updated_at                 | Disabled Reason |
+----+------------------+------------+----------+--------+-------+----------------------------+-----------------+
| 1  | nova-cert        | controller | internal | enabled | up    | 2016-06-29T22:42:05.000000 | -               |
| 2  | nova-consoleauth | controller | internal | enabled | up    | 2016-06-29T22:42:03.000000 | -               |
| 3  | nova-scheduler   | controller | internal | enabled | up    | 2016-06-29T22:42:00.000000 | -               |
| 4  | nova-conductor   | controller | internal | enabled | up    | 2016-06-29T22:41:58.000000 | -               |
+----+------------------+------------+----------+--------+-------+----------------------------+-----------------+

+--------------------------------------+--------------------+------------+-------+-----------------+---------------------------+
| id                                   | agent_type         | host       | alive | admin_state_up | binary                    |
+--------------------------------------+--------------------+------------+-------+----------------+---------------------------+
| 05b7d46d-427a-406a-ab29-4bf012f26e14 | Metadata agent     | network    | :-)   | True           | neutron-metadata-agent    |
| 32d616c4-6963-4424-b33e-e61bcf2cb408 | Open vSwitch agent | network    | :-)   | True           | neutron-openvswitch-agent |
| 782e298a-d322-47cb-8828-89d16510d6df | L3 agent           | network    | :-)   | True           | neutron-l3-agent          |
| ca6d8620-ccb4-4a88-b81a-a797136340ec | DHCP agent         | network    | :-)   | True           | neutron-dhcp-agent        |
+--------------------------------------+--------------------+------------+-------+----------------+---------------------------+
[2016-06-30T01:05:50+30:00] INFO: bash[controller2] ran successfully
    -execute "bash" "/tmp/chef-script20160630-1678-1spmsq7"
[2016-06-30T01:05:51+30:00] INFO: Chef Run complete in 503.437402685 seconds
```

Fig. 7. OpenStack cloud architecture

The Controller node is the one that takes the longer time to complete its installation, because a lot of updates are made and there are quite a few feature installations. For a complete running of the scripts on it we obtained a total time of 8 minutes and 25 seconds. All the components that were required by the Controller were successfully installed and they are all *up* and

enabled. For a complete installation of OpenStack the Chef Client was required also on the Compute node. The same approach was used again and the final part of the script (see Fig. 8).

```
        openvswitch-switch stop/waiting
        openvswitch-switch start/running
        nova-compute start/running, process 7903
        neutron-plugin-openvswitch-agent stop/waiting
        neutron-plugin-openvswitch-agent start/running, process 7916
[2016-06-29t23:32:15+03:00] INFO: bash [compute2] ran successfully
```

Fig. 8. Chef Client Compute node

After all steps were accomplished the expectations were to have an OpenStack environment that is accessible from browser using the Controller's IP address 10.0.0.11. By accessing it from the browser we concluded that the results are very good. The GUI was fully functional and so were the all other services. The overall time of the process was significantly reduced, as it is presented in Fig. 9.

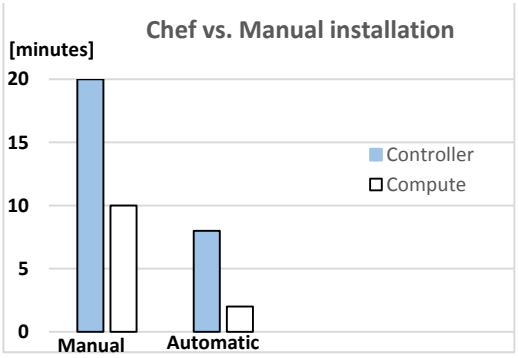

Fig. 9. Performance evaluation of the Chef vs. manual installation

We proved that Chef is one of the best tools for deploying OpenStack-based clouds. Behind the scene there are a lot of scripting in multiple programming languages, but Chef knows how to handle all of them. Although it has a lot of good aspects, there is a major drawback: it does not know how to handle prompts from installed packages. A prompt is a text or symbol used to represent the system's readiness to perform the next command. It was designed to run a complete installation from the beginning up to the end, without being interrupted by the user. For the current implementation the prompts were automated within some bash scripts. For longer term all the prompts within a script must be replaced and their parameters have to be hardcoded or replaced by variables (stored in data bags, similar to registers in Windows).

When we evaluated the performance, we took into consideration all the six steps proposed in Section III. We noticed that a complete manual installation (both tools and scripts) would take at least two hours if everything goes well. In our approach the total time spent including the installation of additional tools (Vagrant and Chef) was maximum one hour (for a good Internet connection).

V. Conclusions and Future Work

The fully automated deployment of OpenStack is not a trivial task, as a great number of tools for every specific operation is needed. The proposed solution brings together three very powerful software packages in order to provide a true cloud deployment, i.e. Chef, Vagrant and VirtualBox. This type of automatic setup has the possibility to easy keep up with the new OpenStack releases. Thus it needs just few line changes to upgrade the installation to a newer version. The solution can be easily scaled by cloning the cookbook for the Compute node and by increment assigning of IP addresses.

Future work may include Jenkins for process automation and the development of a Vagrant configuration for using Docker containers.

Acknowledgment

This work was partially supported by the CHIST-ERA "DIONASYS" project. However the views expressed in this paper are solely those of the authors and do not necessarily represent the views of the entire project.

References

[1] NIST (US National Institute of Standards and Technology, (2011) The NIST Definition of Cloud Computing, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[2] "Cloud Computing: The Concept, Impacts and the Role of Government Policy", OECD Digital Economy Papers, No. 240, OECD Publishing, 2014, http://dx.doi.org/10.1787/5jxzf4lcc7f5-en

[3] "Home » OpenStack Open Source Cloud Computing Software", *Openstack.org*, 2016. [Online]. Available: http://www.openstack.org/ [Accessed: 22- Jul- 2016].

[4] "What is OpenStack?", *Opensource.com*, 2016. [Online]. Available: https://opensource.com/resources/what-is-openstack [Accessed: 22- Jul- 2016].

[5] "Biohazardhpk/OpenStack-Liberty-Install-Script", *GitHub*, 2016. [Online]. Available: https://github.com/Biohazardhpk/OpenStack-Liberty-Install-Script/tree/GRE-tunneling-with-OVS [Accessed: 22- Jul- 2016].

[6] "SaltStack Documentation", *Docs.saltstack.com*, 2016. [Online]. Available: https://docs.saltstack.com/en/latest/ [Accessed: 22- Jul- 2016].

[7] "How it works", *Puppet*, 2016. [Online]. Available: https://puppet.com/product/how-puppet-works[Accessed: 22- Jul- 2016].

[8] "Site Map — Chef Docs", *Docs.chef.io*, 2016. [Online]. Available: https://docs.chef.io/ [Accessed: 22- Jul- 2016].

[9] "Chocolatey - The package manager for Windows", *Chocolatey.org*, 2016. [Online]. Available: https://chocolatey.org/ [Accessed: 22- Jul- 2016].

[10] "Oracle VM VirtualBox", *Virtualbox.org*, 2016. [Online]. Available: https://www.virtualbox.org/ [Accessed: 22- Jul- 2016].

[11] "Documentation - Vagrant by HashiCorp", *Vagrantup.com*, 2016. [Online]. Available: https://www.vagrantup.com/docs [Accessed: 22- Jul- 2016].

[12] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems", *Sixth IEEE International Conference on Cloud Computing 2013*, June 28 - July 3 2013, pp. 887-894.

[13] G. Katsaros, M. Menzel, A. Lenk, J.R. Revelant, R. Skipp, and J. Eberhardt, "Cloud Application Portability with TOSCA, Chef and Openstack", *IEEE International Conference on Cloud Engineering IC2E 2014*, 11-14 March 2014, pp. 295-302.