WWW.JAMK.FI

Configuration management of FreeNEST cloud services

Ossi Rantapuska

Bachelor's thesis
May 2012

Degree Programme in Software Engineering
Technology, ICT

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Author(s)
RANTAPUSKA, Ossi

Type of publication
Bachelor´s Thesis

Date
23.5.2012

Pages
85

Language
English

Confidential

( ) Until

Permission for web publication
(X)

Title
CONFIGURATION MANAGEMENT OF FREENEST CLOUD SERVICES

Degree Programme
Software Engineering

Tutor(s)
KOTKANSALO, Jouko
VÄÄNÄNEN, Olli

Assigned by
JAMK University of Applied Sciences, SkyNEST project

Abstract

The main objective of the thesis was to find a suitable open source configuration management solution for the FreeNEST project platform developed by the SkyNEST project team at JAMK University of Applied Sciences. The aim was to simplify the systems configuration of multiple FreeNEST instances by making use of solutions specifically made for this purpose. The study also aimed to broaden the understanding of configuration management in general, exploring some of the possibilities that it offers for SkyNEST.

Multiple configuration management solutions were selected for basic evaluation. From this selection of tools, three candidates found most suitable for the needs of FreeNEST were picked for a more detailed comparison, along with the arguments as to why they were chosen. The three chosen solutions were evaluated in three FreeNEST use case scenarios, with all found strengths and weaknesses noted. Finally, one of the solutions was chosen to represent the most viable choice for FreeNEST configuration management tasks.

Related technologies such as cloud computing and virtualization were also investigated, along with theory about the history, current form and future of configuration management.

The appendices contain step-by-step tutorials for the installation, configuration and basic use of the three chosen management solutions.

Keywords
SkyNEST, FreeNEST, Configuration Management, Automated Deployment, Orchestration, Remote Configuration, Cloud Computing, Virtualization, Virtual machine, Fabric, Puppet, Chef, Capistrano, Landscape, Juju, Sprinkle, Poni, Push, Pull

Miscellaneous

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) RANTAPUSKA, Ossi	Julkaisun laji Opinnäytetyö	Päivämäärä 23.5.2012
	Sivumäärä 85	Julkaisun kieli Englanti
	Luottamuksellisuus ( ) saakka	Verkkojulkaisulupa myönnetty (X)

Työn nimi
FREENEST-PILVIPALVELUIDEN KONFIGURAATIONHALLINTA

Koulutusohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
KOTKANSALO, Jouko
VÄÄNÄNEN, Olli

Toimeksiantaja(t)
Jyväskylän Ammattikorkeakoulu, SkyNEST-projekti

Tiivistelmä

Opinnäytetyön tavoitteena oli löytää Jyväskylän Ammattikorkeakoulun SkyNEST-projektin tarpeita vastaava avoimen lähdekoodin konfiguraationhallintatyökalu. Työkalun tuli mahdollistaa SkyNESTin kehittämän FreeNEST-projektialustan instanssien joustava etäkonfiguraatio. Työssä perehdyttiin myös yleiseen teoriaan konfiguraationhallinnasta, selvittäen tällaisten ratkaisujen tarjoamia mahdollisuuksia.

Vertailuun sisällytettiin useita eri konfiguraatiotyökaluja. Näistä valittiin perusteluineen kolme FreeNEST-alustalle parhaiten soveltuvaa ratkaisua lähempään tarkasteluun. Näitä kolmea valittua työkalua vertailtiin kolmessa SkyNEST-työryhmältä saadussa käyttötarkoituksessa. Kaikki havaitut vahvuudet ja heikkoudet selostettiin. Lopuksi yksi näistä vertailluista tuotteista valittiin SkyNEST-projektille sopivimmaksi konfiguraationhallintaratkaisuksi.

Tutkimuksessa perehdyttiin myös hieman konfiguraationhallinnan historiaan, nykyiseen tilaan sekä tulevaisuudennäkymiin. Tutkimusta tehtiin myös läheisesti aiheeseen liittyviin teknologioihin, kuten pilvipalveluihin ja virtualisointiin.

Liitteet sisältävät yksityiskohtaiset ohjeet kolmen tarkemmin tutkitun konfiguraationhallintatyökalun asennuksesta, konfiguraatiosta sekä perustason käytöstä.

Avainsanat (asiasanat)
SkyNEST, FreeNEST, Konfiguraatio, Hallinta, Pilvipalvelut, Orkestraatio, Virtualisointi, Virtuaalikone, Etähallinta, Käyttöönotto, Konfigurointi, Projekti, Työkalu, Vertailu, Fabric, Puppet, Chef, Capistrano, Landscape, Juju, Sprinkle, Poni

Muut tiedot

CONTENTS

FIGURES

TABLES

TERMINOLOGY

ICT	Information and communications technology
Management Server	A server machine that manages the configuration of client nodes remotely.
Client node	A machine that is to be configured via a configuration management tool.
Multithreading	Efficiently executing multiple computational threads at the same time.
Virtualization	The creation of a virtual, rather than actual, version of something, such as a hardware platform. (Wikipedia 2012)
Virtual machine	A software implementation of a machine that executes programs like a physical machine. (Wikipedia 2012)
Bottleneck	A part of a computer system that significantly hinders the overall performance or capacity.
Latency	A measure of time delay experienced in a system.

IP Address	Internet protocol address. A numerical label assigned to each device that uses the Internet Protocol for communication. (Wikipedia 2012)
DSL	Domain-specific language. A programming language or specification language dedicated to a particular problem domain. (Wikipedia 2012)
GUI	Graphical user interface. A user interface that is represented and handled visually.
DevOps	An emerging set of principles, methods and practices for communication, collaboration and integration between software development and IT operations professionals. (Wikipedia 2012)
Amazon EC2	Amazon Elastic Compute Cloud. A web service that provides resizable compute capacity in the cloud. (Amazon Web Services LLC 2012)
Ubuntu Advantage	Paid support for Ubuntu Server and Desktop that provides services such as a knowledge base and Landscape.
Sudo	A program for Unix-like computer operating systems that allows users to run programs with the security privileges of another user (normally the superuser, or root). (Wikipedia 2012)

CM	Configuration management. A process for establishing and maintaining consistency of a product's performance, functional and physical attributes with its requirements, design and operational information throughout its life. (Wikipedia 2012)
SCM	Software configuration management. The task of tracking and controlling changes in the software.
Open source	Computer software that is available in source code form: the source code and certain other rights normally reserved for copyright holders are provided under an open-source license that permits users to study and modify the software. (Wikipedia 2012)
CPU	Central processing unit. The portion of a computer system that carries out the instructions of a computer program, to perform the basic arithmetical, logical, and input/output operations of the system. (Wikipedia 2012)
x86	A family of instruction set architectures based on the Intel 8086 CPU.
QEMU	QEMU stands for "Quick Emulator" and is a processor emulator. (Wikipedia 2012)
SSH	Secure Shell. A network protocol for remote administration of Unix computers.

1 BACKGROUND

1.1 History of Service Management

Before the advent of configuration management solutions, deploying and configuring servers used to be a time-consuming task. Most tasks had to be done manually, taking up an unnecessary amount of time and resources. Each server had to be installed and configured independently from each other. Not to mention the continuous maintenance to ensure that everything is running smoothly.

The history of software configuration management (SCM) in computing can be traced back as early as 1950s, when CM (Configuration Management), originally for hardware development and production control, was being applied to software development. Eventually, tools were written to manage software changes. Until the 1980s, SCM could only be understood as CM applied to software development. Some basic concepts such as *identification* and *baseline* (well-defined point in the evolution of a project) were already clear; what was at stake, however, was a set of techniques oriented towards the *control* of the activity, and using formal processes, documents, request forms, control boards etc. It is only after this date that the use of software *tools* applying directly to software artifacts representing the actual resources, has allowed SCM to grow to an autonomous entity from traditional CM. In early 2000s, distributed revision control systems like BitKeeper and GNU became viable. (Wikipedia 2012)

As cloud computing and virtual machines become increasingly prevalent, a demand has been formed for a more flexible approach to administrative systems tasks, such as configuration and deployment of servers. As a result, solutions have emerged to meet this demand. These tools are usually labeled “configuration management tools”, “automated deployment tools” or “orchestration tools”. Although there are differences between the solutions, their basic principle is to allow their users to dynamically manage a multitude of servers from a single remote machine. The

workload of managing multiple computers is shifted into a single instance, greatly simplifying the configuring and maintaining of an IT infrastructure.

1.2 What is IT Service Management?

A service is a means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks. A simple example of a customer outcome that could be facilitated by an IT service might be: "Sales people spending more time interacting with customers" facilitated by "a remote access service that enables reliable access to corporate sales systems from sales people's laptops". (itSMF Ltd. 2007)

Service Management is a set of specialized organizational capabilities for providing value to customers in the form of services. Service management is concerned with more than just delivering services. Each service, process or infrastructure component has a lifecycle, and service management considers the entire lifecycle from strategy through design and transition to operation and continual improvement. Effective service management is itself a strategic asset, providing the ability to carry out the core business of providing services by facilitating the outcomes customers want to achieve. (itSMF Ltd. 2007)

1.3 Service in a cloud

As cloud computing grows in momentum, services are continually being transformed to be cloud compatible. Moreover, many products are turned into services by moving them into the cloud. The end user no longer buys a concrete product; they just buy a slice of the cloud. Although the features of the products may remain the same, they are accessed remotely. Cloud services offer flexibility and mobility as the services are accessible from any location with an Internet connection, and the user does not need to worry about the server load or maintenance; this all comes as a part of the service.

1.4 FreeNEST as a cloud service

The FreeNEST project platform is a portable open source based project platform that provides a working environment for a software development team as well as provides new processes for using the environment efficiently inside a small organization. FreeNEST is developed by the SkyNEST project team at JAMK University of Applied Sciences. All major tool areas for software development are open source and pre-installed on one virtual server image. (SkyNEST 2011)

In short, FreeNEST aims to provide a project environment containing all the necessary components for managing the entire lifecycle of a project. Among other things, FreeNEST aims to create an agile environment for projects, change the focus from ICT to development, share the best practices to make them easier to use in new projects, and reduce communication problems. (SkyNEST 2011)

1.5 Objectives of this thesis

Being a cloud service, FreeNEST is installed on and accessed from server(s). Currently, each of these servers is configured manually. This leads us to the main objective of this thesis: to search and evaluate solutions that enable managing all the FreeNEST instances remotely from a single management server.

FreeNEST requires a scalable configuration management tool, i.e. one that works well with a large number of servers. Although a GUI is not completely out of the question, command-line based administration is preferred. The tool should operate well with OpenStack, which FreeNEST makes use of. The solution should be free, without any costs involved. Additionally, as FreeNEST aims to be as open source as possible, open source management tools should be preferred.

Within the timeframe, as many solutions as possible will be evaluated. Since SkyNEST uses Ubuntu, the tests will be conducted in that environment. After an overview of each solution, the list is narrowed down to the three most suitable candidates, along

with an explanation as to why they were chosen. The capabilities of these three solutions will be evaluated in three FreeNEST use case scenarios. Any advantages or disadvantages will be noted and presented in a comparison table. Finally, one of the solutions shall be chosen to represent the most viable choice for FreeNEST configuration management.

2 TECHNOLOGIES AND DEFINITIONS

2.1 Cloud Computing

2.1.1 Overview

In the past, companies had to periodically purchase new hardware and software to meet their growing resource requirements. Computers, servers and databases used to be locally operated. Balancing server capacity was left to the customer, requiring over-purchasing of hardware to support peak demand, which in turn resulted in overall underutilization. Services were purchased in-person, requiring manual business processes and fixed price contracts. Individuals would buy a concrete product to take home with them.

Cloud computing is the order of the day. There is no simple way to explain this somewhat vague term. Gartner defines it as “a style of computing in which massively scalable IT-related capabilities are provided “as a service” using Internet technologies to multiple external customers (Brodkin 2009). This means that the customer purchases services that are accessed over the Internet, instead of running them locally. Products are turned into services. This enables users to shift the workload of local computers into a network of computers that make up the so-called cloud, as depicted in figure 1. The local machine only needs an interface with which it can access the cloud, and the cloud’s network takes care of the rest. (Strickland 2008)

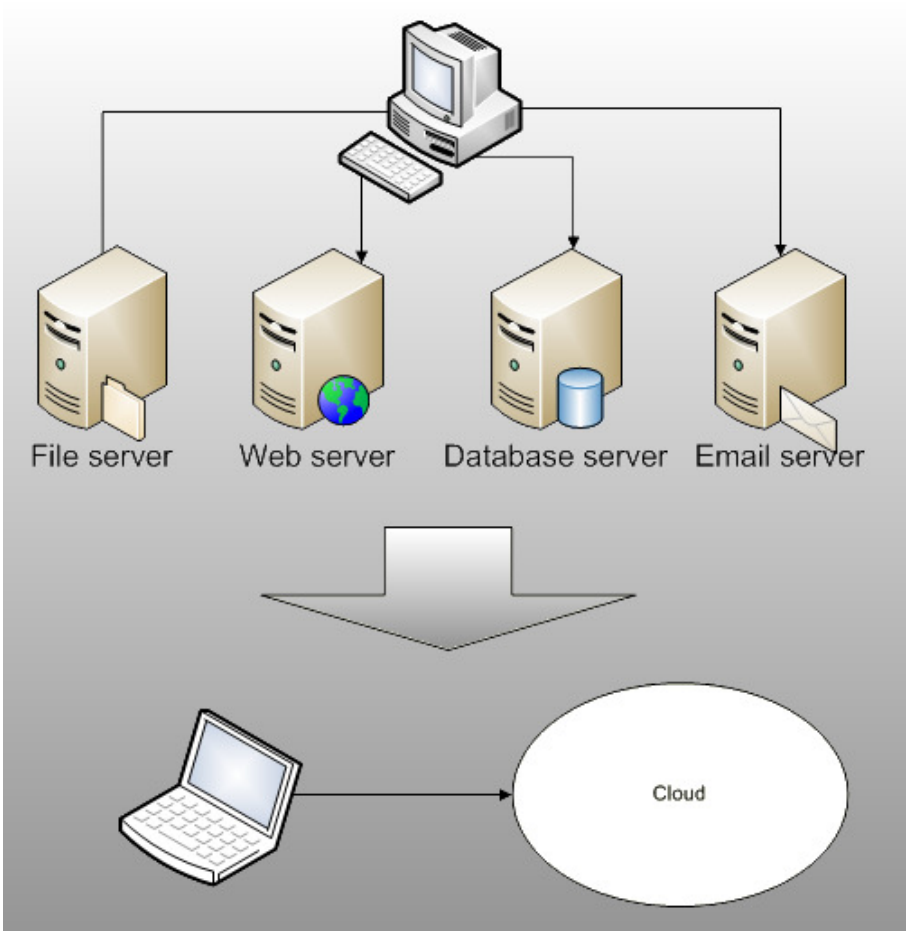

FIGURE 1. Cloud computing shifts the workload into the cloud

In its simplest form, cloud computing could mean a Web-based e-mail service. Instead of running an e-mail program on a local computer, one simply connects to a Web service (Strickland 2008). Both the program and the e-mail messages are stored remotely on the cloud, and are accessible from anywhere.

Both public and private clouds exist. Public clouds are cloud services that are publicly available to everyone for a fee. Private clouds are similar cloud services, but are limited to operating within an enterprise and not offered for external customers. Although private clouds may not offer all the benefits of public clouds, largest enterprises may not justify transferring all of their IT resources to public clouds due to their possible lack of reliability.

Public cloud services can be divided into three categories: software, infrastructure and platform. Software-as-a-service (SaaS) means delivering software applications over the web. Infrastructure-as-a-service (IaaS) refers to remotely accessible server and storage capacity. Platform-as-a-service (PaaS) denotes a platform that lets developers build and deploy Web applications on a hosted infrastructure. (Brodkin 2009)

2.1.2 Software as a service (SaaS)

In this model, cloud providers install and operate application software in the cloud, while users access the software from cloud clients. This eliminates the need to install and run the application on the user's own computer, simplifying maintenance and support. (Wikipedia 2012)

An example of the SaaS model is Google's Gmail, a web-based e-mail service. It offers a simple piece of software – a tool to read and send email – as a remote service. The tool is never installed on the user's computer; it is operated via a web browser.

2.1.3 Platform as a service (PaaS)

In the PaaS model, cloud providers deliver a computing platform and/or solution stack typically including an operating system, a programming language execution environment, a database, and a web server. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers. (Wikipedia 2012)

Google App Engine is an example of a platform as a service. It allows running web applications on Google's infrastructure. With App Engine, there are no servers to maintain: the application is uploaded, and it is ready to serve the users. Google App Engine makes it easy to build an application that runs reliably even under heavy load, with features such as automatic scaling and load balancing. (Google, Inc. 2012)

FreeNEST offers a portable project platform for developers, so it is generally considered a PaaS type cloud service.

2.1.4 Infrastructure as a service (IaaS)

In this most basic cloud service model, cloud providers offer computers – as physical or more often as virtual machines, raw storage, firewalls, load balancers and networks. IaaS providers supply these resources on demand from their large pools installed in data centers. To deploy their applications, cloud users install operating systems and application software on the cloud. In this model, it is the cloud user who is responsible for patching and maintaining the software. (Wikipedia 2012)

OpenStack is an Infrastructure as a Service cloud computing project by Rackspace Cloud and NASA. OpenStack delivers a massively scalable cloud operating system for building private and public clouds. It provides an operating platform for orchestrating clouds. OpenStack is a community as well as a project, with the goal to help organizations run clouds for virtual computing or storage. OpenStack is a collection of open source projects: OpenStack Compute (provision and manage large networks of virtual machines), OpenStack Object Storage (redundant, scalable storage) and OpenStack Image Service (discovery, registration and delivery for virtual disk images). (OpenStack Wiki 2012)

2.1.5 Strengths and weaknesses

With the help of cloud computing, the end user no longer has to worry about the increasing hardware and software demands, as the cloud takes care of them. This removes a lot of responsibilities from the user, letting them focus on their key areas. The need for new hardware installations is greatly lowered, while software maintenance becomes much easier, as local application installations are no longer required. Among the biggest advantages of the cloud is its scalability: service delivery can be quickly scaled up or down according to the user's needs. Removing the need of maintaining an in-house IT infrastructure, cloud computing is a common way to

save on IT costs. Cloud services typically have a pay-per-usage model, which usually ends up being beneficial to the customer, compared to a fixed price model.

Although there are undisputed advantages to cloud computing, there are some negative aspects as well. Storing data in a cloud does not mean that it is guaranteed to be safe. Some online storage vendors have lost data, and were unable to recover it for customers (Brodkin 2009). As infrastructures move towards the public networks, it naturally brings into mind the question of security. How can the end users be sure that their information will not fall into the wrong hands? On the other hand, one could consider the security better than normal due to the centralization of data. Being dependent on a cloud also means being dependent on a network connection, as downtime could mean an almost complete halt in operations. In addition, using software over a network means that a certain amount of network latency will always occur. Thus, software applications that require minimal latency may still need to be installed locally.

According to Gartner Inc., cloud is a disruptive force that has the potential for broad long-term impact in most industries. While the market remains in its early stages in 2011 and 2012, it will see the full range of large enterprise providers fully engaged in delivering a range of offerings to build cloud environments and deliver cloud services. Oracle, IBM and SAP all have major initiatives to deliver a broader range or cloud services over the next two years. As Microsoft continues to expand its cloud offering, and these traditional enterprise players expand offerings, users will see competition heat up and enterprise-level cloud services increase. (Gartner, Inc. 2011)

2.2 Virtualization

2.2.1 Overview

Virtualization was first introduced in the 1960s by IBM to boost utilization of large, expensive mainframe systems by partitioning them into logical, separate virtual

machines that could run multiple applications and processes at the same time. (McCabe 2009)

While virtualization faded from the limelight for a while, it is now one of the hottest trends in the industry again, as organizations aim to increase the utilization, flexibility and cost-effectiveness in a distributed computer environment. (McCabe 2009)

Virtualization allows us to create virtual machine instances inside a single physical machine. These virtual machines behave like physical machines, supporting the installation of most operating systems and software as if they were installed on a physical computer. The virtual machine exists on software level only, but the software installed on it “thinks” that it is running on physical machine.

Virtualization offers many benefits. A server no longer has to serve just one purpose: it can run multiple virtual machines, distributing the load more efficiently (see figure 2). Virtual machines can easily be cloned. For example, if there is a need for multiple development platforms with identical software configurations, this can be easily accomplished with virtualization. Another benefit of virtualization is that it makes it easier to install multiple operating systems on a single computer. Since each virtual machine is an independent entity, there needs to be no concern of different operating systems causing compatibility problems. Virtualization is thus useful for testing software compatibility with various platforms and operating systems.

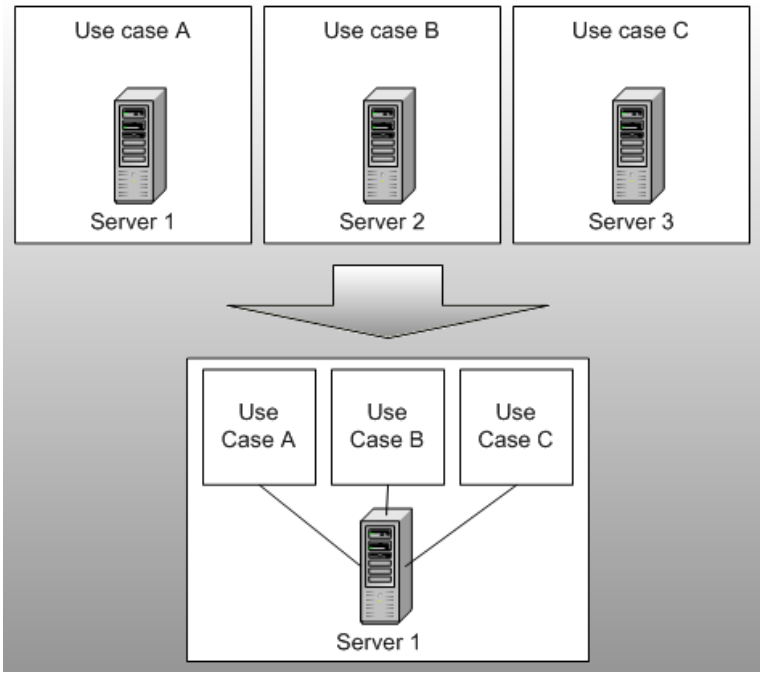

FIGURE 2. Virtualization can increase server efficiency

In addition to partitioning one machine into several virtual machines, it is also possible to use virtualization to combine multiple physical resources into a single virtual resource. A good example of this is storage virtualization, where multiple network storage devices are pooled into what appears as a single storage device. (McCabe 2009)

2.2.2 VMWare

With over 100,000 customers worldwide, VMWare, Inc. is the world's leading provider of virtualization solutions. As with most virtual machines, it aims to separate the operating system and hardware and encapsulate operating systems and applications into virtual machines, lowering hardware dependencies and improving flexibility. (Gilmartin, J. 2008)

VMWare software provides a completely virtualized set of hardware to the guest operating system. This makes VMWare virtual machines highly portable between computers, because every host looks nearly identical to the guest. If needed, a virtual machine can be paused on one host machine and resumed exactly at the point of suspension on another host. VMWare does not simulate each CPU instruction one-by-one, significantly increasing its performance while hampering compatibility between hosts using different instruction sets. (Wikipedia 2012)

VMWare Player is a free virtual machine for non-commercial use, while the commercial VMWare Workstation is a more feature-rich alternative. VMWare Workstation runs on Windows and Linux operating systems, while VMWare Fusion is targeted for Macs.

2.2.3 Oracle VM VirtualBox

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. VirtualBox is the only professional solution that is freely available as open source software. VirtualBox is a community effort backed by a dedicated company: everyone is encouraged to participate while Oracle ensures the professional quality of the product. (Oracle Corporation 2012)

VirtualBox is relatively simple to use and suitable for beginners. Although its performance or maturity may not yet be on par with the VMWare VMs, it has most of the expected features of a virtual machine. It also supports a wide range of operating systems – including Windows, Linux and Mac OS X.

2.2.4 KVM

KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, kvm.ko, that provides the core virtualization infrastructure

and a processor specific module, kvm-intel.ko or kvm-amd.ko. KVM also requires a modified QEMU although work is underway to get the required changes upstream. KVM is really a kernel infrastructure instead of a product. (KVM 2012)

Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware. KVM is included in the mainline Linux kernel and is stable and fast for most workloads. It is also available as a patch for recent Linux kernel versions. (KVM 2012)

KVM's biggest advantages are its simplicity and performance. As it takes full advantage of hardware-level support in new CPUs, KVM can achieve superior performance despite its relative simplicity.

2.3 The information technology infrastructure library

The information technology infrastructure library (ITIL) is a public framework that describes Best Practice in IT service management. It provides a framework for the governance of IT, the 'service wrap', and focuses on the continual measurement and improvement of the quality of IT services delivered, from both a business and a customer perspective. This focus is a major factor in ITIL's worldwide success and has contributed to its prolific usage and to the key benefits obtained by those organizations deploying the techniques. (itSMF Ltd. 2007)

Figure 3 visualizes the enhancements that ITIL brings to the service lifecycle. According to an Introductory Overview of ITIL v3 (itSMF Ltd. 2007), some of the benefits of ITIL include:

- Increased user and customer satisfaction with IT services
- Improved service availability
- Financial savings from reduced rework, lost time, improved resource management and usage
- Improved time to market new products and services
- Improved decision making and optimized risk

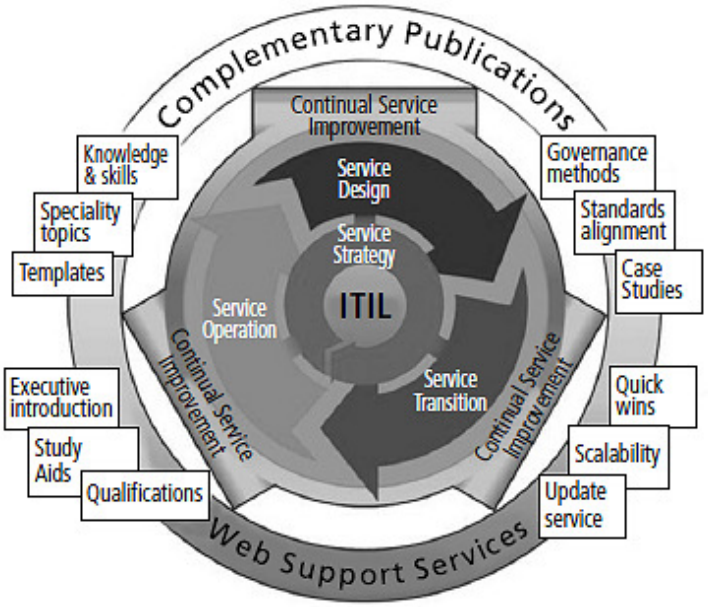

FIGURE 3. The service lifecycle enhanced with ITIL (Source: itSMF Ltd.)

Many organizations still see IT service management as being predominantly a technology issue. ITIL promotes a much more “joined up”, “end-to-end” approach to IT service management. This will only happen if practices and architectures that are focused on business needs and business processes are adopted. The ITIL framework gives a sound basis for achieving all of this once management tools and interfaces evolve to fully support them. (itSMF Ltd. 2007)

3 CONFIGURATION MANAGEMENT IN A CLOUD

3.1 Orchestration tools

The usage of orchestration is often discussed in the context of service oriented architecture, virtualization, provisioning, Converged Infrastructure and dynamic datacenter topics. Orchestration in this sense is about aligning the business request with the applications, data and infrastructure. It defines the policies and service

levels through automated workflows, provisioning and change management. This creates an application-aligned infrastructure that can be scaled up or down based on the needs of each application. (Wikipedia 2012)

With the popularity of cloud computing, it is now necessary to understand Service Orchestration in the context of this paradigm. At the most basic level an orchestrator is a human. The main difference between a workflow automation and orchestration is that workflows are processed within a single domain. Cloud Service orchestration therefore is the

- Composing of Architecture, Tools and Processes by humans to deliver a defined service
- Stitching of software and hardware components together to deliver a defined Service
- Connecting and automating of work flows when applicable to deliver a defined service

(Wikipedia 2012)

Orchestration tools focus on the deployment and managing of applications instead of generic system-level configuration management. It is important to note that there is significant overlap between service orchestration and configuration management. Many tools support both. This study focuses on configuration management instead of orchestration. An example of an orchestration tool is Ubuntu's juju. Figure 4 shows the layers of orchestration and configuration management tools in relation to a computer system.

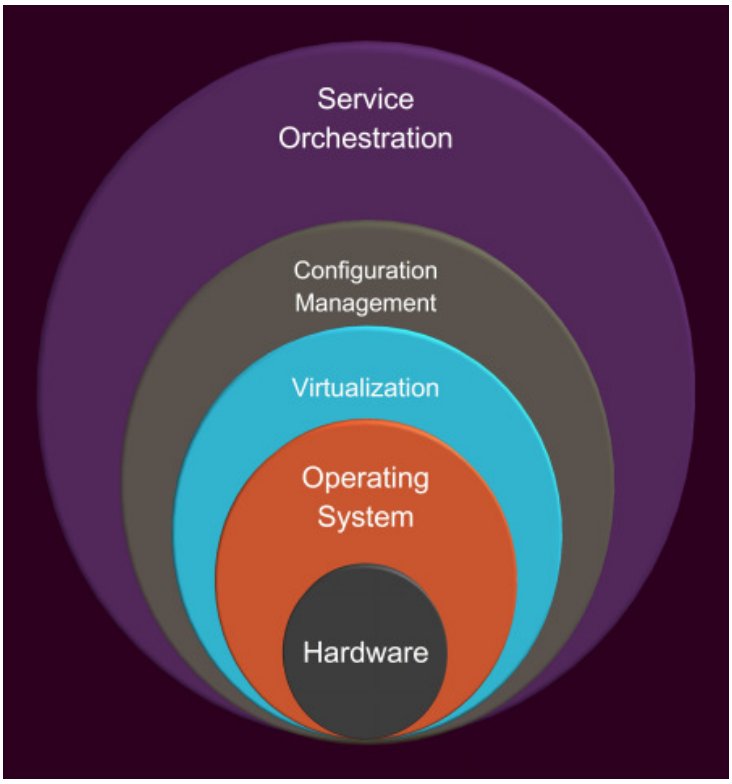

FIGURE 4. Layers of orchestration and configuration management (Source: Canonical ltd.)

3.2 Configuration management tools

Configuration management tools work on a more primitive level than orchestration tools, geared towards configuring software and servers instead of services and applications. If used properly, configuration management tools allow us to remotely make changes to multiple target machines (often called “nodes”) from a single management server.

Configuration management tools can be divided into two groups: some use the so-called “push” method, while others use the “pull” method. The goal of these is generally the same – to automate and simplify the configuration of multiple remote machines. However, the way these tools take to accomplish these changes is quite different.

3.2.1 “Push” tools

The simpler method is to *push* the changes from the management server into the target nodes. In this case, setting up the nodes is usually easy; almost everything is done on the management server. Examples of “push” tools are Fabric and Capistrano, about them more detailed later in this document.

The main advantages of a “push” system are simplicity and control. These configuration management systems are usually relatively easy to set up and get running. When changes are made to the nodes, they all take place at the same time – when the changes are pushed. This makes it easier to detect and correct if something went wrong. (Gheorghiu 2010)

Disadvantages of “push” systems are that they are usually not capable of full automation – they generally do not support a client/server protocol, and the lack of scalability for larger projects. Since all of the workload falls on the management server, a push system can start showing its limits unless it makes heavy use of multithreading. (Gheorghiu 2010)

3.2.2 “Pull” tools

With a “pull” system, the server acts as configuration storage, with the clients themselves *pulling* the needed configuration information from the server. Examples of “pull” tools are Puppet and Chef.

With a “pull” solution, it is possible to fully automate the configuration of a newly booted server (Gheorghiu 2010). Each client can be set to periodically check the server for changes and process them if necessary. As part of the workload is shifted to the clients, “pull” systems offer greater scalability for larger projects. However, the server will still become a bottleneck if too many clients try to connect at the same time.

"Pull" type systems are usually slightly more complex to get running, since both the client and server side will have to be set up properly. If used properly, "pull" systems offer increased automation for larger projects, operating independently and leaving little manual work for the user. They operate at a higher level than "push" systems: instead of saying "Install this package", one would say "ensure that this package is installed on these machine(s)". When the client runs its next check, it notices that and checks if the package is installed, and installs it if necessary.

An example of package management is shown in figure 5. Normally, every server would have to be maintained manually to make sure it has the right package version installed, and possibly install packages by hand. When using a configuration management tool, the configuration needs to be set only once, on the management server. The rest will be taken care of automatically, and the system will ensure that the correct package versions are installed on all nodes. Most tools support some kind of reporting so that the user has up-to-date information on the current state of each node.

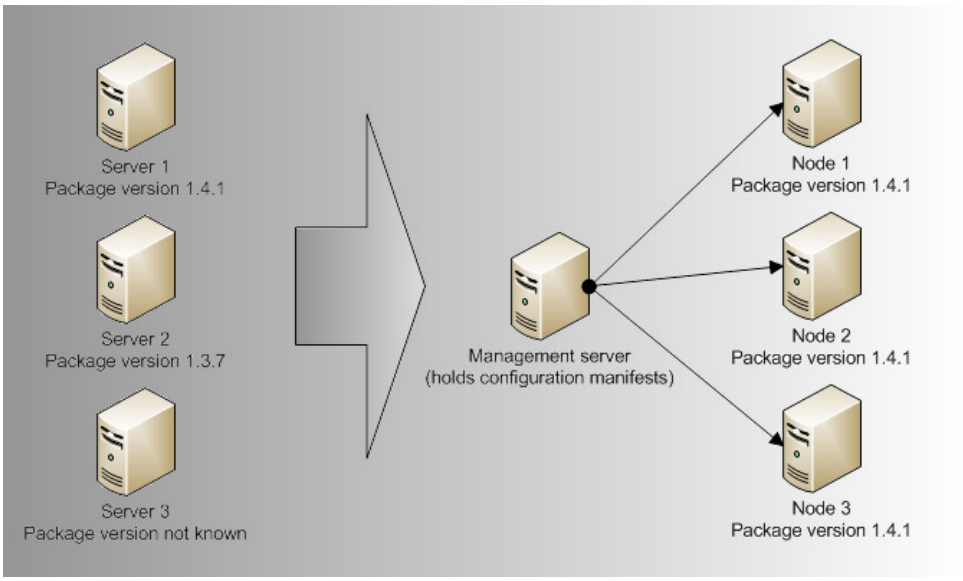

FIGURE 5. Using configuration management to manage packages

3.3 Solutions

3.3.1 Capistrano

Capistrano is a utility and framework for executing commands in parallel on multiple remote machines, via SSH. It uses a simple DSL that allows defining tasks, which may be applied to machines in certain roles. Capistrano was originally designed to simplify and automate deployment of web applications to distributed environments. (Capistrano overview 2012)

Although commonly used to deploy Rails applications, Capistrano also supports remote configuring of systems, using the Ruby language as its base. It can easily accomplish basic tasks, such as running commands remotely. Knowledge on Ruby will greatly help in solving more complex tasks.

Capistrano uses the “push” method to apply changes, thus it only needs to be installed on the management server. The client side requires SSH connectivity and a POSIX-compatible shell installed. Capistrano has no GUI; it is entirely command-line driven. It is used by creating scripts that use a custom DSL on top of Ruby. The scripts are run with the *cap* command. In addition to describing the tasks to be done, the scripts define the server addresses to connect to, and group tasks for specific servers.

Capistrano shares many similarities with Fabric (detailed in the next chapter). Although Capistrano is more deployment-focused, both tools work at the same level, using SSH to connect to target nodes and push forward the wanted changes. Capistrano has a slightly larger feature list, but Fabric wins in ease of use. Mostly it is a choice of language preference between the two: does one prefer Ruby (Capistrano) or Python (Fabric). In relation to the other solutions, Capistrano has mostly the same strengths and weaknesses as Fabric.

3.3.2 Fabric

Fabric is a Python (2.5 or higher) library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks. It provides a basic suite of operations for executing local or remote shell commands (normally or via sudo) and uploading/downloading files, as well as auxiliary functionality such as prompting the running user for input, or aborting execution. Typical use involves creating a Python module containing one or more functions, then executing them via the fab command-line tool. The Fabric development team is headed by Jeff Forcier. However, dozens of other developers pitch in by submitting patches and ideas via GitHub, IRC or the mailing list. (Hansen, C. & Forcier, J. 2012)

Fabric is a “push” type management tool similar to *Capistrano*, but uses the Python scripting language instead. Python scripts are infused with Fabric commands to make parts of them run remotely on the target machine(s). An important issue to note is that although Fabric makes heavy use of Python, it does not provide direct support for running scripts remotely. With Fabric, one is never “on the remote server”; the scripts are still run locally. Therefore, Fabric relies heavily on the execution of shell commands for solving tasks. Fabric and its configuration scripts are stored on the management server. Most of the workload with Fabric comes from writing the actual Python scripts.

As an example, if a file on a node were to be modified, a script could be created that makes use of Fabric’s file transfer commands: one could download the file to the management server, make the required changes, and finally upload it back to the node. The addresses of the nodes, as well as their required user names and passwords are specified in the scripts. Finally, the script is run with the *fab* command. Fabric then automatically connects to the specified node(s), performs the changes and outputs the results in the terminal (see figure 6).

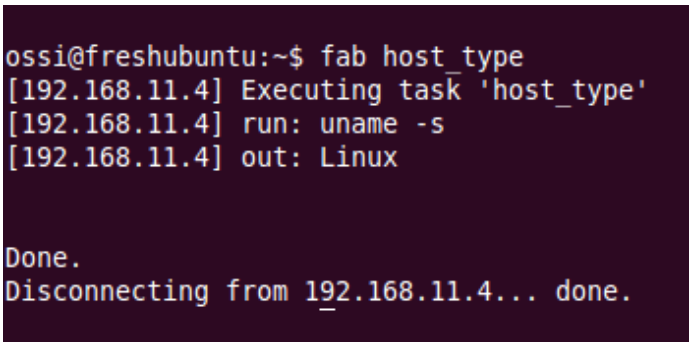

FIGURE 6. Output from Fabric running the shell command “uname –s”

Fabric is extremely easy to set up, relatively easy to use and makes use of the Python language, which is already familiar to the SkyNEST team. It is very programmer oriented, requiring little time to learn the tool itself. As the scripts mostly consist of ordinary Python code, a developer who has never even used Fabric will still have an idea of how they work.

3.3.3 Puppet

Puppet is an automated administrative engine for *nix systems developed by Puppet Labs. It performs administrative tasks (such as adding users, installing packages, and updating server configurations) based on a centralized specification. Puppet lets users focus more on how tasks should be accomplished and less on doing them. It aims to let computers do what they are good at; precisely perform patterns, so users can focus on creating solutions. (Puppet overview 2011)

Puppet has been developed to help the system administrator community move to building and sharing mature tools that avoid the duplication of everyone solving the same problem. It provides a powerful framework to simplify the majority of technical tasks that system administrators need to perform. Puppet can handle most of the details, and code can be downloaded from other system administrators to help get

the tasks done even faster. There are already hundreds of modules developed and shared by the community. (PuppetLabs 2012)

Puppet uses the “pull” type implementation of configuring nodes. Therefore, the Puppet software has to be installed and configured on both the management server and the nodes. Puppet is typically used in a client/server formation, with all the clients talking to one of more central servers. After the systems are configured correctly, the nodes periodically (or manually if so desired) connect to the management server to check for configuration changes, and synchronize with the latest updates. The nodes report back to the server to tell it what was changed. The management server acts as a storage for all the configuration manifests. All the configuration modifications will be done on or transferred to the management server. Figure 7 visualizes the data flow in a typical Puppet implementation.

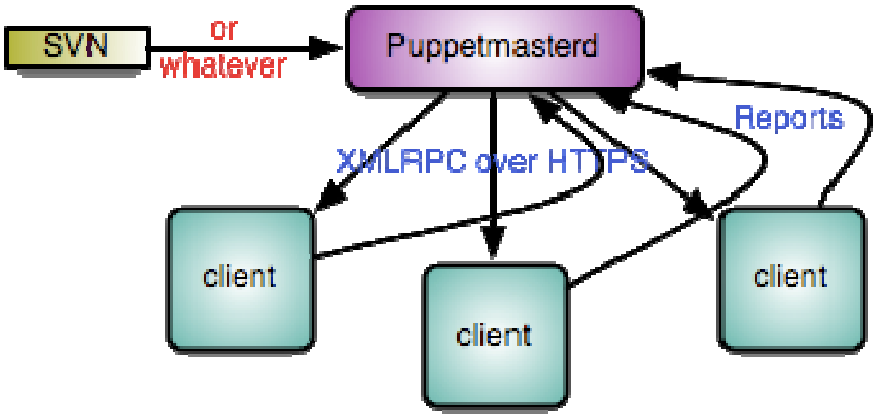

FIGURE 7. Data flow in a typical Puppet implementation (Source: PuppetLabs 2012)

In Puppet, the user specifies the states for the nodes, instead of directly making changes. One example would be setting the state of a package to “installed”. Therefore, Puppet can be seen as a higher level tool than for example Fabric. To describe these states, Puppet uses its own proprietary DSL that is derived from Ruby. It allows the user to specify the required state of each node and/or node group.

Puppet is more of a professional solution for configuration management than a simple tool to make quick system changes. Since it has been around longer than most alternatives, it is well documented and supported. It also includes a large amount of built-in features, but the complexity comes at a cost: it is slightly laborious to get running and to learn its more advanced features. The user not only needs to learn a proprietary language, but also the usage of Puppet itself.

3.3.4 Juju

Juju is an open-source solution for service orchestration developed by Canonical Ltd. Juju was formerly known as Ensemble. Juju concentrates on the notion of service, abstracting the notion of machine or server, and defines relations between services that are linked with each other. These relations are automatically updated when two linked services observe a notable modification. This allows for easy scaling of services. (Wikipedia 2012)

Juju is designed to solve the needs of both developers and system administrators. It is a solution for orchestrating cloud services, i.e. deploying, connecting, and controlling service applications across all systems. Juju focuses on managing the service units needed to deliver a single solution, above simply configuring the machines or cloud instances needed to run them. (Ubuntu Wiki 2012)

Developers can benefit from juju's rapid deployment of dependencies and by reproducing deployments for test and staging purposes. System administrators can see what is deployed and track usage in the cloud, as well as monitor, scale, shrink and adjust deployment parameters in real time. Juju also allows these two groups to better collaborate on the exact deployment and upgrade processes. (Ubuntu Wiki 2012)

Juju uses so-called *charms* to deploy services. These charms are open and worked on by the community, and thus represent a best set of practices for deploying these services. Through the use of charms, juju provides shareable, re-usable and repeatable expressions of DevOps best practices. (Ubuntu Wiki 2012)

Juju can be described as a higher level solution than tools such as Puppet or Fabric, orchestrating services instead of managing configurations. In fact, some use juju in combination with a lower-level tool to accomplish system level tasks. Juju is generally cross-platform, but made with Ubuntu in mind. The public availability of charms is an advantage, as solutions for many tasks have already been done and can be re-used. Juju charms can be written in any language; all juju needs is a set of executable files that it can trigger appropriately (Ubuntu Wiki 2012).

As stated in the juju FAQ (Ubuntu Wiki 2012), juju is not yet ready to be used in production; the rough estimate is to have juju ready for Ubuntu 12.04. The FAQ also reveals that juju currently only deploys to the Amazon EC2 cloud, and that each service unit is currently deployed to a separate EC2 instance. Juju will support multiple services per machine in the future, but these issues together decrease its appeal. While tools such as Puppet are good at low level operations, juju has a high level scope and has good support for building relationships between services.

A juju environment needs its own dedicated bootstrap node, a utility node that is used to manage the environment. Next, requires charm(s). These can be written in any language or downloaded from charm libraries. A charm can be deployed using the *deploy* command. For scaling up a service, juju has the simple *add-unit* command. Finally, relations between services can be added with the *add-relation* command.

3.3.5 Landscape

Landscape is an easy-to-use systems management and monitoring service by Canonical Ltd. to manage multiple Ubuntu systems as easily as one and lower management and administration costs. With Landscape, essential information can be viewed and uniformity across installations delivered from a single console, problems can be identified with a clear view of live resource usage statistics from Landscape's dashboard, and clouds can be managed as easily as physical machines. (Canonical Ltd. 2012)

Landscape comes in two forms: *Hosted Edition* and *Landscape Dedicated Server*. The hosted edition is run on Canonical's servers. The dedicated server version allows customers to run the Landscape systems management server onsite.
Landscape provides an intuitive web interface for managing nodes. Most basic features are provided, such as making sure a specific package is installed, editing users and running scripts. All the activities done by Landscape can be monitored from an activity log (shown in figure 8).

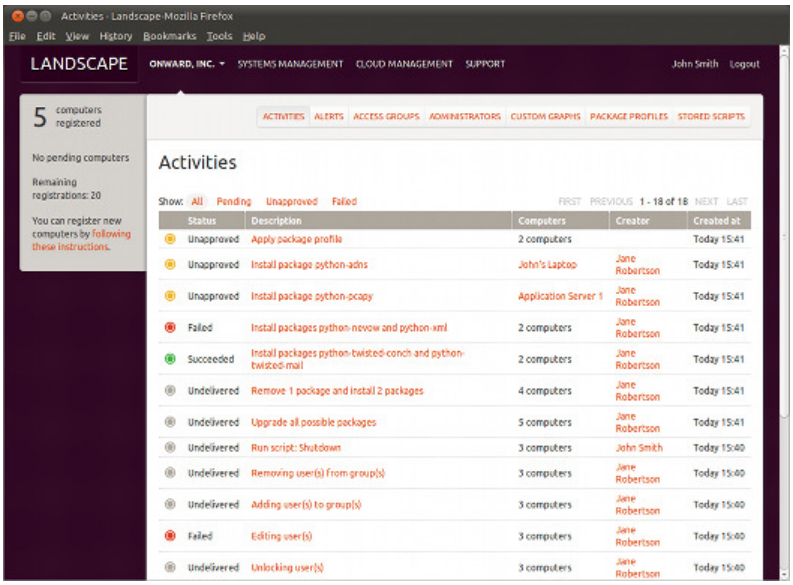

FIGURE 8. Activity log of Landscape (Source: Canonical Ltd.)

Landscape's main disadvantages are that it is not free, does not work on the command line and only works on Ubuntu systems, making it less flexible that the alternatives. It does also have some advantages. It includes comprehensive monitoring features that allow monitoring everything from CPU temperature to system load. Landscape can also be set to alert the user if there are important packages or security updates available. Landscape is very easy to use after getting used to it. All in all, it is a good solution for basic managing of multiple Ubuntu server installations, but is not as comprehensive or flexible as the alternatives.

3.3.6 Sprinkle

Sprinkle is a software provisioning tool developed by Marcus Crafter that can be used to build remote servers after the base operating system has been installed. Properties of packages such as their name, type, dependencies, etc., and what packages apply to what machines is described via a DSL that Sprinkle executes. One of the aims of Sprinkle is to define as concisely as possible a language for installing software. (Crafter, M. 2012)

Most configuration management solutions are either "push" or "pull" based. Sprinkle tries to merge these concepts together, combining the intelligent, state-based configuration of the "pull" method with the ease of installation and lack of need for specialized software of "push" solutions. Sprinkle uses Capistrano internally for communicating with remote systems, thus Capistrano is a pre-requisite for Sprinkle. However, this is pluggable; Sprinkle also supports custom delivery mechanisms.

To use Sprinkle, a deployment script and package definitions are created. In the deployment script, Sprinkle is told what packages are available, which packages are to be delivered on which servers, and what delivery mechanism to use. The package definitions contain the actual "meat" of the scripts. The title "package definition" can be misleading, since the script itself does not **have** to have anything to do with a

package. In addition to package-related commands such as installing a package, Sprinkle can run commands remotely and transfer files.

Sprinkle offers a nice blend of features and simplicity. The Ruby-based scripting DSL can take some time to learn, but with the exception of some ActiveSupport compatibility problems, Sprinkle was found to be quite to set up and get running. Its usage is somewhat similar to Fabric, but uses Ruby and is state-based instead of task-based. A heavy disadvantage of Sprinkle is that it has not been very actively updated. As of March 2012, the latest changes are dated 5 months ago. Moreover, documentation on the official site (https://github.com/crafterm/sprinkle) is pretty concise and other sources of documentation are hard to come by.

3.3.7 Chef

Chef is an open-source systems integration framework by Opscode inc., built specifically for automating the cloud. No matter how complex the realities of the business, Chef makes it easy to deploy servers and scale applications throughout the entire infrastructure. Because it combines the fundamental elements of configuration management and service oriented architectures with the full power of Ruby, Chef makes it easy to create an elegant, fully automated infrastructure. (Opscode 2012)

There is a multitude of Chef products available, thus it is important to distinguish the different options from each other. Firstly, there are three main versions of the product:

- Hosted Chef
- Private Chef
- Chef

Hosted Chef is a paid service that offers access to a fully supported automation environment hosted by Opscode. Private Chef is basically the same, but runs inside a pri-

vate network. This document focuses on the edition simply called “Chef”, the open source version of Chef. This open source edition is split into multiple versions:

- Chef Solo
- Chef Server
- Chef Client

Chef Solo is a standalone, locally running version of Chef that is not attached to a server. All the required information, including the configuration manifests, is stored locally on the node. They can be retrieved via a remote URL with shell commands.

Chef Server and Chef Client work together to make a client-server system similar to Puppet. The configuration files are stored on the server and the client nodes connect to the server to retrieve them. Compared to Chef Solo, this offers more dynamic configuration management, supporting features such as roles for nodes.

This document concentrates on the Server/Client version of Chef, as it seems best suited for the use cases of FreeNEST. Of all the compared solutions, Chef Server is probably the closest alternative to Puppet. It operates with configuration states, describing the state at which each resource should be. The premise of Chef is to make sure each resource is configured properly, and to make sure the servers are always running exactly as wanted. Due to the server-client method, Chef needs to be installed on both the server and client machines. Although the Chef Server stores the configuration information, Chef includes a handy tool called Knife with which changes to the configuration can be done via an external workstation, without the need to log onto the management server itself. Knife can also be used to communicate directly with the nodes using SSH. Figure 9 visualizes the architecture behind Chef Server and Chef Client.

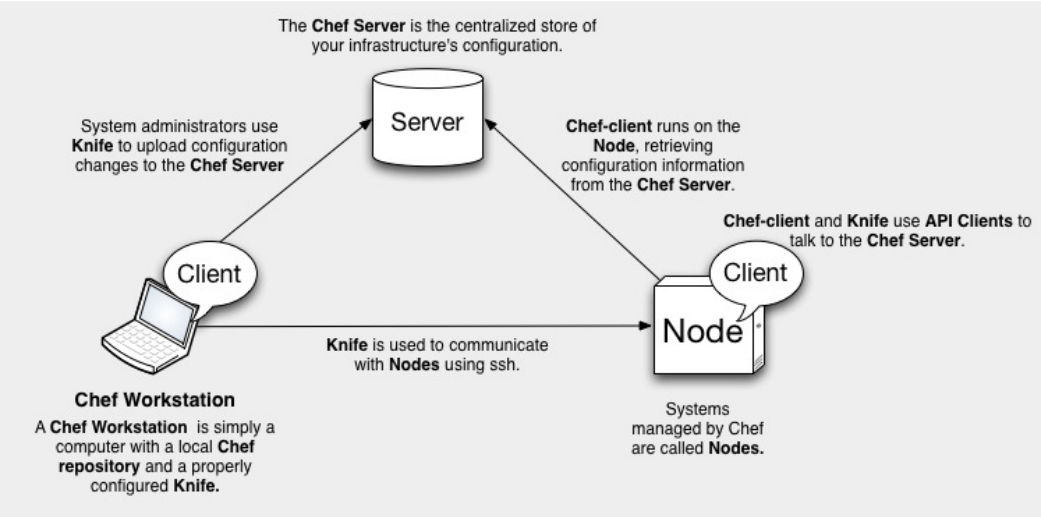

FIGURE 9. The architecture of Chef (Source: Opscode Wiki)

Chef arranges configuration files into cookbooks, recipes, nodes, roles and run-lists. This fairly logical arrangement makes it easy to modify existing configurations as well as add new ones. Cookbooks are the fundamental units of distribution in Chef. They store recipes, specifying what resources should be managed and how. Nodes are the systems managed by Chef, while roles group features of similar nodes together to provide specific sets of functionality. Both roles and nodes have run-lists that specify the recipes they use.

Chef is no doubt a well-rounded, feature-rich tool for configuration management. When it comes to FreeNEST, its perhaps biggest disadvantage is unnecessary complexity, as simpler tools like Fabric can be set up in a fraction of the time it takes to get Chef fully up and running.

3.3.8 Poni

Poni is a simple system configuration management tool implemented in Python for defining, deploying and verifying complex multi-node computer systems. Systems, nodes, installed software and settings are stored in a central Poni repository, so

there is a single location where the system is defined and documented. (Eloranta 2012)

Poni uses a command-line tool to control almost every aspect of the configuration, including editing the configuration manifests, adding relations and describing services. With Poni, the infrastructure is defined as a hierarchy of keys and values instead of just simple configuration files. As an example, there could be the following hierarchy:

- System webshop/backend
    - Node webshop/backend/postgres1
        - Config webshop/backend/postgres1/config1

By separating these three types of items hierarchically, changes to a specific part of the configuration can be made easily, as well as new nodes and configurations added to the system.

Configuration files and installation scripts are bundled into so-called *Poni configs*, each typically representing one software component, such as a HTTP server. Each node type is configured to include one of more of these configs. The actual processing of changes is done by creating template-based files or running custom functions remotely using a remote execution framework. Poni requires the use of SSH keys for authentication.

Although Poni does have potential, it has some serious drawbacks. Most importantly, it is early in its development – the initial version was released in November of 2010. Due to its young age and lack of popularity, Poni documentation is extremely hard to come by. Poni's official documentation at http://melor.github.com/poni/ does help getting started, but there is a lack of practical usage examples. Excluding the official documentation, Poni documentation is practically inexistent. This combined with the usage of a distinctive command-line tool can make it daunting to learn.

4 EVALUATION SETUP AND CHOSEN SOLUTIONS

4.1 Use cases

This subchapter describes each of the given use cases of FreeNEST. These use cases represent possible ways in which the SkyNEST project could make use of configuration management solutions. Later in this document, each of these use cases is evaluated using the chosen solutions to see how well each solution manages.

4.1.1 Use case 1 – modifying a configuration file

In this use case, there is a specific file on the remote machine(s) that is to be modified. More specifically, the HOSTS file on the target machine(s) is modified, changing the domain name for a given IP address. A visualization of this use case is shown in figure 10.

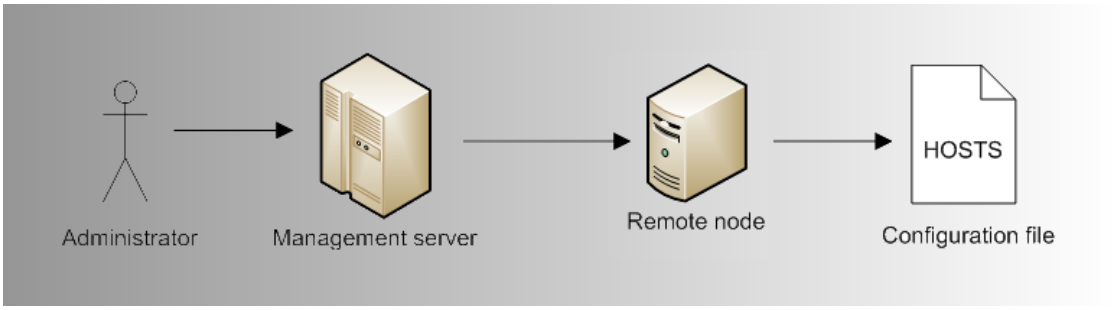

FIGURE 10. Visualization of use case 1.

4.1.2 Use case 2 – installing a specific version of a package

This use case consists of managing packages on a node running Ubuntu. The solution needs to remotely install a specific version of a given package. A visualization of this use case is shown in figure 11.

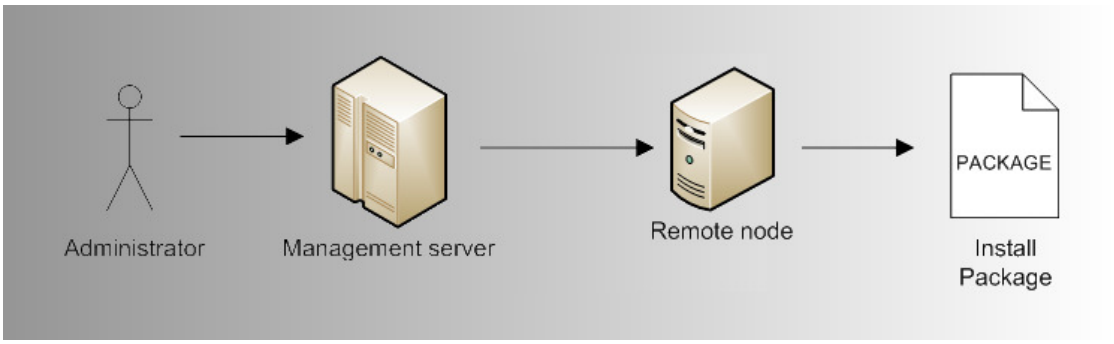

FIGURE 11. Visualization of use case 2.

4.1.3 Use case 3 – multiple target nodes

In this use case, multiple targets nodes need to be configured, each with their own specific configuration settings. A visualization of this use case is shown in figure 12.

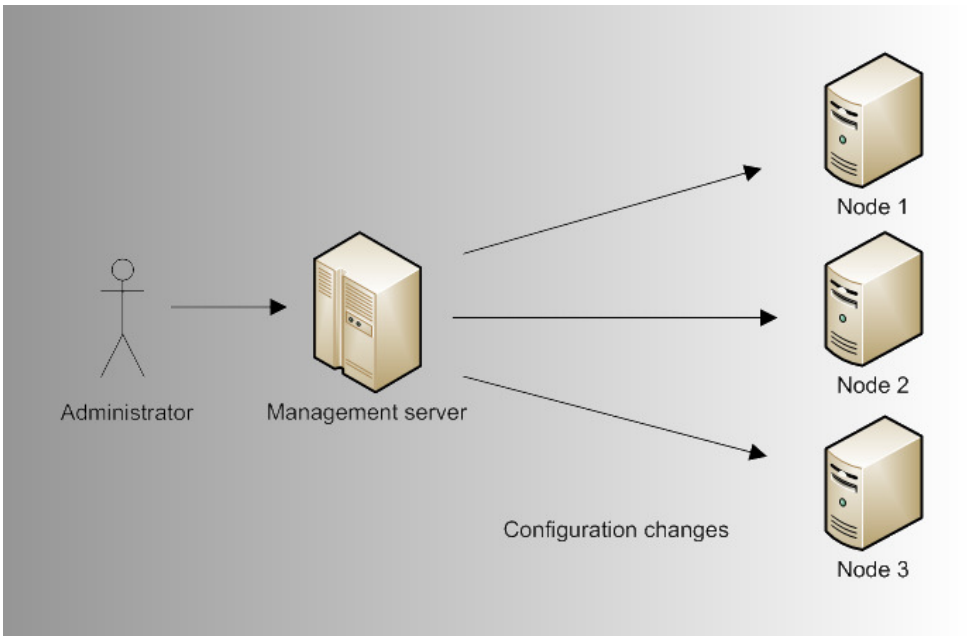

FIGURE 12. Visualization of use case 3.

4.2 Evaluation setup

Virtual machines were used in the evaluation of the given use cases. Here are the specifications of the evaluation setup:

- Virtual Machine: Oracle VM VirtualBox 4.1.8
- Host OS: Windows XP 32-bit SP3
- Client OS (both server and client): Ubuntu 10.04 LTS
- Fabric 1.4
- Puppet 2.7.10
- Chef 0.10.8

4.3 Chosen solutions

4.3.1 Fabric

4.3.1.1 Why Fabric?

Fabric was chosen for closer inspection due to its short learning curve, fast results and relatively good documentation. Fabric works well for quick and dirty changes to configurations. It is a suitable choice for developers due to its programmer-oriented base.

4.3.1.2 Features

Fabric uses SSH to connect the server to the node(s). Fabric's biggest advantage – its simplicity – also leads to its biggest drawback: the lack of comprehensive built-in features. Although it is easy to get started with Fabric, it can become complicated as the complexity of the environment increases. With Fabric, most of the tasks are solved by making heavy use of the *run* command, which executes shell commands.

One could say that instead of directly managing nodes, Fabric gives the user a basic framework on which to script a management system of their own. If multiple administrators run fabric commands simultaneously, unexpected results can occur due to the configuration changes being pushed into the node at the same time. Fabric is a “push” type tool, which makes it is easy to detect possible problems right after a push has been done.

4.3.1.3 Ease of use

Fabric itself is extremely simple to set up and use. The installation process of Fabric is described in Appendix 1. Other than SSH connectivity, the node machines themselves do not need to have anything particular installed. It is possible to get the first commands working in less than 10 minutes. More complex configuration management will require knowledge of Python coding. Due to the similarity with Python, Fabric scripting skills will not go to waste even if the tool itself is later abandoned. Python language is already being used by the SkyNEST team, so the learning curve should be nonexistent.

4.3.1.4 Performance

Fabric was not designed with parallel processing in mind, meaning that the management server has to push the changes to the target nodes one by one. As all of the workload falls on the management server, it can quickly become a bottleneck. With smaller projects, this is a non-issue, but if there are hundreds of nodes to configure, it can be a drawback. There is preliminary support for parallel processing, but it is not natively supported.

4.3.1.5 Maturity & documentation

Fabric has been around since 2009 (Hansen, C. & Forcier, J. 2012). It can be considered more mature than some alternatives, but not quite as mature as the veterans of this field, such as Puppet. Fabric is still somewhat young and developed by a relatively small group, so the future support for the project can be seen as uncertain. As of February 2012, Fabric is continually developed and actively updated. There have already been two major releases during the year. Fabric is documented quite well on the Fabric website at http://docs.fabfile.org/. Other sources of documentation can be hard to come by.

4.3.2 Puppet

4.3.2.1 Why Puppet?

Having its roots back in 2005, Puppet is mature compared to most other open source tools. Puppet is also a very popular and well-received solution, powering thousands of companies including Google, Oracle and Twitter (PuppetLabs 2012). It is relatively well documented over the Internet. Puppet also has a large community around it, as well as heaps of free, community-made modules to expand its functionality. In addition, Puppet Labs has announced support for OpenStack:

> *Coinciding with this week's OpenStack Summit in San Francisco, today we're pleased to highlight the great community efforts around the Puppet modules for OpenStack. Working together, Puppet Labs and OpenStack community members Cisco, Red Hat, Rackspace, Morphlabs, and eNovance have designed and developed a collection of modules that allow sysadmins to automatically provision, configure, and manage OpenStack clouds with Puppet open source or Puppet Enterprise. The first of these modules, OpenStack cloud provisioning, is available today for free download from Puppet Forge. (Puppet Labs 2012)*

According to the Puppet Blog (Puppet Labs 2012), Puppet was chosen as the 2011 Best Open Source Configuration Management Tool by Linux Journal readers.

4.3.2.2 Features

Puppet provides built-in commands for most basic tasks, such as:

- Installing packages
- Checking and copying file contents
- Ensuring a given service is running

Puppet can retrieve operating system facts such as IP addresses and SSH keys via Facter, an independent, cross-platform Ruby library. The proprietary DSL supports the use of variables and conditional statements, as well as parameterized classes. Templates can be used for the configuration manifests to make the system more flexible. There is also quite thorough support for customizing the sequence at which the configuration changes will be processed. Everything can be logged in a special reports directory so no change will go unnoticed.

4.3.2.3 Ease of use

Installing and configuring Puppet can be somewhat time-consuming. The installation process of Puppet is described in Appendix 2. Both the server and client side need the Puppet software installed. Since Puppet uses its own proprietary language, it can be considered to have a steeper learning curve than Fabric. Puppet can be a powerful tool, but only after it has been mastered.

4.3.2.4 Performance

Being a pull type management tool, Puppet shifts part of the load from the management server to the nodes themselves. Performance-wise, this is an advantage. As long as the system is not being bottlenecked by the server, the tasks can be processed simultaneously without individual nodes having to wait for their turn.

4.3.2.5 Maturity & documentation

The first version of Puppet was released in 2005 (Wikipedia 2012), so it can be considered one of the veterans in the field of configuration management. Due to its maturity, Puppet should in theory be less likely to cause problems or contain bugs. Puppet is documented quite extensively on the Puppet Labs website, including feature lists, tutorials and exercises among other facts. Due to its maturity and large user base, it is reasonably safe to assume that Puppet will continue to be supported and updated in the future.

4.3.3 Chef

4.3.3.1 Why Chef?

Chef was chosen mainly due to its impressive feature list, customizability and developer-oriented approach to tasks. It might seem slightly daunting at first, but can become a powerful tool in the right hands. The included Knife tool makes it possible to manually make quick changes to all the specified machines. For developers, the Ruby language that Chef uses may be more comfortable than the custom DSL that Puppet uses. There are Chef repositories available for deploying OpenStack, such as https://github.com/openstack/openstack-chef.

4.3.3.2 Features

Chef includes built-in features to modify many basic resources such as files, services and packages. Most of these features are comparable to Puppet's alternatives. In addition, there are free cookbooks available for download to increase the versatility of Chef. However, the real power of Chef comes from mastering the Ruby language to create custom cookbooks tailored for specific needs. As far as built-in features are concerned, Chef is comparable to Puppet.

4.3.3.3 Ease of use

Of all the studied solutions, Chef is probably the most laborious to install and get running – especially the client/server version. Appendix 3 provides step-by-step installation instructions. There are some tutorials available over the Internet, but it seems difficult to find simple, concise instructions for this task. The structure of the tool itself is quite logical, making it reasonably easy to use after a bit of learning. The inclusion of the Knife tool is a plus for usability, as it allows one to make configuration changes from outside the management server itself.

Compared to Puppet, one could say Chef is more programmer than administrator oriented. Using the Ruby language, it offers complex possibilities for those who know their way around scripting. Chef is overall more dynamic than Puppet; it gives more room for the user to decide how to accomplish tasks. On the other hand, there is also less help from the tool itself, leaving many tasks for the user to figure out. This favors developers, giving them an open playground for making their own solutions to tasks.

4.3.3.4 Performance

As with Puppet, Chef uses the "Pull" method to configure systems. This helps to prevent overloading the management server. Applying the configuration changes with Chef also seems to take less time than with Puppet. In a simple test consisting of installing a package, Chef took only about half the time that Puppet did.

4.3.3.5 Maturity & documentation

According to Wikipedia (Wikipedia 2012), the first release of Chef took place in January 2009. This makes its age about equal with Fabric. Although younger than Puppet, Chef also has a community around it. This makes finding documentation as well as third-party modifications easier. There is a multitude of configuration modules (known as *cookbooks*) freely available for download. Although Chef is essentially well documented in the Opscode Wiki (http://wiki.opscode.com/display/chef/Home), the documentation is somewhat unorganized and thus hard to follow. Chef has been received quite well, so it will likely keep evolving and continue to be supported.

5 Evaluation of chosen solutions

5.1 Overview

Here are presented a quick overview and observations of how the chosen solutions coped with the given use cases. Detailed, step-by-step instructions for performing the tasks are provided in Appendix 4 (Fabric), Appendix 5 (Puppet) and Appendix 6 (Chef).

5.2 Use case 1 – modifying a configuration file

This use case requires making line-specific edits to a file, which is usually not a built-in feature of configuration management tools. Thus, this use case requires one to either rely on third party modules or to code a custom solution.

5.2.1 Fabric

File modifying is easy to achieve with Fabric by making use of Python. One simply needs to make a script that edits the file to produce the expected results. However, since Fabric does not directly run scripts on the target node, the file needs to be either temporarily transferred to the management server for editing, or be directly edited using shell commands on the node. One way to achieve this task is to temporarily transfer the file to the management server, read through the file line by line, searching for a specific match and modifying the line whenever a match is found, and finally upload it back to the node. After the script is ready, the *fab* command is used to call the script. Target nodes and user information can be either included in the script(s) or as parameters for the *fab* command.

5.2.2 Puppet

While Puppet does not have a built-in feature for precise file content editing, there are user definitions available that can accomplish this task. For this use case, Simple Text Patterns can be used:

http://projects.puppetlabs.com/projects/1/wiki/Simple_Text_Patterns

By making use of these additional definitions, one can use Puppet to edit file contents with relative ease. The *ensure_key_value* definition is very well suited for the purposes of this use case; it finds a given string at the beginning of a line and modifies the rest of the file as specified. First, a new module for Puppet is created. The definition is added to the Puppet class, then called with the specified

parameters: *file* (the file to modify), *key* (the string to search for), and *value* (the contents to add to the line). Target nodes are specified in the Puppet file *nodes.pp*. The checkout itself is done from the nodes, connecting to the server and automatically finding out their required modules from the management server's *nodes.pp* file.

5.2.3 Chef

In Chef, Ruby can be used to add functionality to cookbooks. Unlike Fabric, Chef allows running Ruby script(s) directly on the target machine, so there is no need for file transferring - the file can be edited directly on the node. First, a new Chef cookbook is created. It includes a Ruby script that reads the lines of a given file one by one, modifying the lines that match a given substring.

After the cookbook is ready, it is uploaded to the Chef Server using the Knife tool, and added to the run list of the wanted node(s). The nodes periodically connect to the server, read their assigned run list and process the cookbooks and recipes specified there.

5.3 Use case 2 – installing a specific version of a package

This use case consists of making sure a specific version of a given package is installed. Installing packages is likely one of the most common uses of configuration management tools, thus most tools have this feature built-in. Specific versioning can be a bit more complex, but overall still relatively easy to achieve.

5.3.1 Fabric

Like most solutions, Fabric allows running shell commands on the target node. Installing packages requires administrator permissions, so the Fabric function *sudo* should be used for this purpose. It simply runs a shell command as sudo. As a

parameter, the function is given the shell command required for installing the wanted package.

Specific versioning with Fabric is slightly more complex. One option is to use a shell command such as *dpkg-query* to check which version of the package is currently installed. The **script** is run on the management server, while the **shell commands** are run on the target node. Therefore, one needs to parse the Fabric output to get the version information (the output of the *dpkg-query* command) into the script itself. After the version information is known by the script, simple Python functions and statements can be used to determine whether or not a new version should be installed.

5.3.2 Puppet

Once a basic directory structure has been created for Puppet, making a Puppet module for installing a package is a trivial task. There is a built-in command to ensure that a given package is installed. All that is needed is to tell Puppet the name of the package that needs to be installed. Puppet also supports ensuring a specific **version** of a package is installed. As long as the exact version number of the package is known, installing that specific version is just as easy as installing a package in general. After the module is created, it can be used by the node(s), and it will be processed on the next update.

5.3.3 Chef

Like Puppet, Chef has built-in support for package management. Knife makes it easy to create new cookbooks, as it automatically creates the default directory structure for them. Installing a specific version is no more laborious than installing a package, as Chef has built-in support for package versioning. Chef Server only needs to know the name and version of the package to be installed. After the cookbook is ready, it is added to the node's run list, informing Chef Server that the node needs to process these changes on its next update cycle.

5.4 Use case 3 – multiple target nodes

This use case consists of configuring multiple target nodes, with each node having its own, specified configuration.

5.4.1 Fabric

Fabric contains certain environment variables that can be modified. These include *env.hosts* and *env.roledefs.* The hosts variable is an array that stores the addresses of all the target nodes for the script. All that is needed is to add all the target node addresses into this variable in a script, and Fabric will run that script on all the given nodes, without the need for additional command-line parameters. The roledefs array can be used for grouping similar nodes into “roles”. After nodes have been grouped into a single role, only the role needs to be specified to Fabric, and the script will affect all the clients included in that role.

5.4.2 Puppet

Puppet has a specific file, *nodes.pp*, for storing the node information. The address and all the required modules for each node are specified there. Whenever a Puppet client connects to a Puppet server, it gets its tasks from this file. If more flexibility is required, one way is to use node inheritance. Using inheritance, one can create a base node with all the general configuring needed for every server, and then inherit individual nodes from the base node, adding more modules if needed. As the modules themselves are separate from the nodes, no changes are needed to them. Configuring multiple nodes works the same way as configuring a single node; the only differences are in the *nodes.pp* file.

5.4.3 Chef

Chef uses a logical structure with nodes, with cookbooks, recipes, nodes, roles and run lists. The cookbooks are the fundamental units of distribution in Chef, containing

the actual scripts for the configuration, known as recipes. They are separate from the nodes and roles, and can be added to the run lists of either a node or a role using Knife. When a Chef Client connects to the Chef Server, it looks at its assigned run list for tasks to achieve, and makes configuration changes accordingly. Therefore, to make changes to multiple nodes, each node or its role simply needs to have assigned recipes on its run list. More clients and roles can easily be added as necessary with Knife.

6 RESULTS AND CONCLUSION

6.1 Results

The conducted evaluation comprised only short-term usage tests. While results were gathered and a conclusion was drawn, one should keep in mind that long-term usage evaluation might produce drastically different results, quite possibly revealing more strengths and weaknesses of these products. The determined positive and negative aspects of each solution is described in table 1.

TABLE 1. Comparison of the top three solutions

Solution	Positive aspects	Negative aspects
Fabric	• Easy to set up and get started • Easy to expand with Python • Short learning curve due to Python language	• Relative lack of documentation • Uncertain future • Not made for parallel operations
Puppet	• Comprehensive feature list • Abundant documentation available • Maturity • Community + modules • Future proof	• Steeper learning curve due to proprietary DSL • Not as easy to tailor for specific needs as Fabric or Chef
Chef	• Comprehensive feature list • Developer-oriented nature, easy to expand with Ruby • Community + cookbooks • Knife tool makes performing changes easy	• Laborious installation • Incoherent documentation

6.2 Conclusion

Any of the compared solutions would probably suit the needs of SkyNEST. Fabric convinces with its simplicity; however, its lack of built-in features, concise documentation and uncertain future support eliminates it from being the top choice. Puppet and Chef are in many ways similar, both being “pull” tools that offer extensive features along with comprehensive documentation. Puppet has been around for a little longer than Chef, and does have a larger community around it. Chef combines Fabric’s customizability with a more extensive feature list, comprehensive documentation and future assurance. Comparing Puppet and Chef, both have their advantages and disadvantages. In the end one could say that it comes down to preference. When evaluating the suitability for FreeNEST, Chef gets a narrow win mainly due to its more developer-oriented approach. Although the task of installing and configuring Chef can be a bit daunting, the tool itself works logically and is pretty simple to use. The Ruby language can be used to make custom scripts and expand functionality, while Knife allows for easy changes to configuration from any machine that has it installed.

6.3 Future

Like cloud computing, configuration management and service orchestration are currently very active fields, constantly changing form. What will the future have in store for configuration management?

As development progresses, solutions will strive to lessen the human workload even further. Tools will be made easier to use, as well as more adaptable, stable and scalable. One probable change that will be seen in the near future is the fusion of lower-level and higher-level tools. In some ways, this has already begun taking place. Previously deployment-focused tools such as Capistrano are entering the systems administration territory, and vice versa. Why have one solution to handle system configuration, and another one for application deployment? A single solution to solve all orchestration tasks is much easier to manage and work with, not to mention

less expensive. There will be less focus on specific areas, and more focus on the overall picture. High level aspects will need to be combined with low level operations. Similar issues should be grouped together and handled by the same tools to allow for reliable bindings and predictable service agreements. Higher level functionality will increase, requiring less input from the user to accomplish tasks. Juju is an example of a higher level configuration tool, managing service units instead of systems or files. It is built from the ground-up for the cloud. Another higher level solution that might gain more ground in the future is Microsoft's System Center Opalis, an automation platform that can automate tasks across various systems without the need for scripting. This is achieved through workflow processes that orchestrate System Center and other management tools in an integrated manner. Some tools integrate other tools to provide a broader spectrum of management. As an example, Dell's Crowbar, an open source bare-metal provisioning tool is an extension of Chef Server. As cloud computing and virtualization become increasingly common, orchestration tools will follow these trends and be specifically built with those aspects in mind. Solutions will take more advantage of virtualization for increased scalability and reduced complexity, and will be built for the elasticity of the cloud. The current solutions will have to either adapt or be prepared to face some stiff competition.

REFERENCES

Strickland, Jonathan. 2008. How Cloud Computing Works. HowStuffWorks.com 8.4.2008. Cited 26.1.2012. http://computer.howstuffworks.com/cloud-computing/cloud-computing.htm

Brodkin, Jon. 2009. FAQ: Cloud computing, demystified. networkworld.com 18.5.2009. Cited 27.1.2012. http://www.networkworld.com/supp/2009/ndc3/051809-cloud-faq.html

Gartner, Inc. 2011. Top 10 Strategic Technologies for 2012. 18.10.2011. Cited 27.1.2012. http://www.gartner.com/it/page.jsp?id=1826214

McCabe, Larie. 2009. What is Virtualization, and Why Should You Care? 7.5.2009. Cited 28.1.2012. http://www.smallbusinesscomputing.com/testdrive/article.php/3819231/What-is-Virtualization-and-Why-Should-You-Care.htm

SkyNEST. 2011. Cited 31.1.2012. http://conceptnest.org/www/what-is-freenest

Gheorghiu, Grig. 2010. Automated deployment systems. push vs. pull. 3.3.2010. Cited 2.2.2012. http://agiletesting.blogspot.com/2010/03/automated-deployment-systems-push-vs.html

Hansen, C. & Forcier, J. 2012. Fabric 1.3.4 documentation. Cited 6.2.1012. http://docs.fabfile.org/en/1.3.4/index.html

Hansen, C. & Forcier, J. 2012. Fabric 1.4.0 changelog. Cited 20.2.1012. http://docs.fabfile.org/en/1.4.0/changelog.html

Puppet Labs. 2012. Puppet overview. Cited 7.2.2012. http://projects.puppetlabs.com/projects/puppet

Puppet Labs. 2012. Company overview. Cited 22.2.2012. http://puppetlabs.com/company/overview/

Puppet Labs. 2012. Introduction to Puppet. Cited 24.2.2012. http://docs.puppetlabs.com/guides/introduction.html

Virtualization. n.d. Wikipedia. Cited 8.2.2012. http://en.wikipedia.org/wiki/Virtualization

Virtual machine. n.d. Wikipedia. Cited 8.2.2012.
http://en.wikipedia.org/wiki/Virtual_machine

Latency (engineering). n.d. Wikipedia. Cited 8.2.2012.
http://en.wikipedia.org/wiki/Latency_(engineering)

IP address. n.d. Wikipedia. Cited 13.2.2012.
http://en.wikipedia.org/wiki/IP_address

Comparison of open source configuration management software. n.d. Wikipedia. Cited 20.2.2012.
http://en.wikipedia.org/wiki/Comparison_of_open_source_configuration_management_software

Capistrano overview. n. d. Capistrano Wiki. Cited 19.4.2012.
https://github.com/halorgium/capistrano

What is Juju? n. d. Ubuntu Wiki. Cited 12.3.2012.
https://juju.ubuntu.com/

Juju FAQ. n. d. Ubuntu Wiki. Cited 12.3.2012.
https://juju.ubuntu.com/FAQ

Juju (software). n.d. Wikipedia. Cited 12.3.2012.
http://en.wikipedia.org/wiki/Juju_(software)

DevOps. n.d. Wikipedia. Cited 12.3.2012.
http://en.wikipedia.org/wiki/DevOps

Amazon Web Services LLC. 2012. Amazon Elastic Compute Cloud (Amazon EC2). Cited 12.3.2012.
http://aws.amazon.com/ec2/

Canonical Ltd. 2012. Landscape. Cited 12.3.2012.
http://www.canonical.com/enterprise-services/ubuntu-advantage/landscape

Crafter, M. 2009. Sprinkle. Cited 14.2.2012.
https://github.com/crafterm/sprinkle

Opscode. 2012. Chef. Cited 14.2.2012.
http://www.opscode.com/chef/

Eloranta, Mika. 2012. Poni readme. Cited 20.3.2012.
https://github.com/melor/poni

Eloranta, Mika. 2012. Poni documentation. Cited 20.3.2012.
http://melor.github.com/poni/

Cloud computing. n.d. Wikipedia. Cited 2.4.2012.
http://en.wikipedia.org/wiki/Cloud_computing

Google, Inc. n.d. What is Google App Engine? Cited 2.4.2012.
https://developers.google.com/appengine/docs/whatisgoogleappengine

OpenStack Wiki. 2012. n.d. What is OpenStack? Cited 2.4.2012.
http://docs.openstack.org/bexar/openstack-compute/admin/content/ch01s01.html

Wikipedia. 2012. n.d. VMWare. Cited 2.4.2012.
http://en.wikipedia.org/wiki/VMware

Gilmartin, John. 2008. VMWare Overview. Slideshare.com 28.7.2008. Cited 2.4.2012.
http://www.slideshare.net/digitallibrary/vmware-overview

Oracle Corporation. 2012. Oracle VM VirtualBox. Cited 2.4.2012.
https://www.virtualbox.org/

KVM. 2012. Kernel Based Virtual Machine. Cited 2.4.2012.
http://www.linux-kvm.org/page/Main_Page

itSMF Ltd. 2007. An Introductory Overview of ITIL ® V3. Cited 3.4.2012.
http://www.itsmfi.org/files/itSMF_ITILV3_Intro_Overview.pdf

Wikipedia. 2012. n.d. History of software configuration management. Cited 3.4.2012.
http://en.wikipedia.org/wiki/History_of_software_configuration_management

Wikipedia. 2012. n.d. Orchestration (computing). Cited 16.4.2012.
http://en.wikipedia.org/wiki/Orchestration_(computing)

Puppet Labs. 2012. Puppet Labs Announces Support for OpenStack. puppetlabs.com 17.4.2012. Cited 20.4.2012.
http://puppetlabs.com/blog/cloud/puppet-labs-announces-support-for-openstack/

Wikipedia. 2012. n.d. Configuration Management. Cited 24.4.2012.
http://en.wikipedia.org/wiki/Configuration_management

Wikipedia. 2012. n.d. Sudo. Cited 24.4.2012.
http://en.wikipedia.org/wiki/Sudo

Wikipedia. 2012. n.d. Open-source software. Cited 24.4.2012.
http://en.wikipedia.org/wiki/Open-source_software

Wikipedia. 2012. n.d. Central processing unit. Cited 24.4.2012.
http://en.wikipedia.org/wiki/Central_processing_unit

Wikipedia. 2012. n.d. QEMU. Cited 24.4.2012.
http://en.wikipedia.org/wiki/Qemu

Puppet Labs. 2012. Puppet Wins Best Open Source Configuration Management Tool. puppetlabs.com 1.12.2011. Cited 24.4.2012.
http://puppetlabs.com/blog/puppet-wins-best-open-source-configuration-management-tool/

APPENDIX 1. Tutorial on installing and configuring Fabric

Overview

This is a tutorial on how to install Fabric on the client and server side running fresh installs of Ubuntu 10.04. Server in this case means the management server, while the client(s) is/are the nodes to be configured via Fabric. Unless otherwise stated, the following steps are required for both the server and the client.

Server side installation

First, we install all the packages that Fabric requires:

```
sudo apt-get install python-dev python-setuptools ssh
```

Next, we install Fabric itself. Note that this should be done via *easy_install* as the *apt* version tends to be outdated:

```
sudo easy_install fabric
```

Client side installation

As Fabric does most of the work on the server side, only SSH connectivity is needed for the client:

```
sudo apt-get install ssh
```

Testing the connection

For testing the connection between the management server and node(s), create a file called *fabfile.py* with the following contents:

```
from fabric.api import run

def host_type():
           run('uname -s')
```

The above Fabric script simply runs the command 'uname –s' on the node. NOTE: Since it is Python code, the indentation must be correct for it to work.

Now we can test the newly created script with the following command (replace the ip address with the ip address of the target node):

```
fab -H '192.168.11.12' host_type
```

If everything is working, a line similar to the following should be outputted:

```
[192.168.11.12] out: Linux
```

APPENDIX 2. Tutorial on installing and configuring Puppet

Overview

This is a tutorial on how to install Puppet on the client and server running fresh installs of Ubuntu 10.04. Server in this case means the management server running Puppetmaster, while the client is the node to be configured via Puppet. Unless otherwise stated, the following steps are required for both the server and the client.

Installing the prerequisites

Install the base packages via apt:

```
sudo apt-get install ruby rubygems libopenssl-ruby
```

Installing through apt-get won't get us the newest version of RubyGems, so we have to update it to 1.8.15 manually:

```
cd /tmp
sudo wget http://production.cf.rubygems.org/rubygems/rubygems-1.8.15.tgz
sudo tar –xzf rubygems-1.8.15.tgz
cd rubygems-1.8.15
sudo ruby setup.rb
```

RubyGems should now be version 1.8.15. You can check to make sure it is by typing in the following command:

```
gem –v
```

Installing Puppet

Now we are ready to install Puppet itself:

```
sudo gem install puppet --no-ri  --no-rdoc
```

Configuring the hostnames

Now we should configure the hostnames so that our server and client can easily find each other. First we should find out the IP address of each machine with the following command:

```
ifconfig
```

You can find the IP address in the resulting lines. It should look similar to this:

```
inet addr:192.168.11.10
```

Next, add the following lines to /etc/hosts:

```
xxx.xxx.xxx.xxx puppetmaster.example.com puppetmaster puppet
xxx.xxx.xxx.xxx puppetclient.example.com puppetclient
```

Notice that you need to replace xxx.xxx.xxx.xxx with the IP address of each machine, and example.com with whatever domain name you are using.

[SERVER] Setting up Puppetmaster

First we should create a basic directory structure for Puppet. We can use this handy template from Bitfield Consulting:

```
cd /etc
sudo wget http://bitfieldconsulting.com/files/powering-up-with-puppet.tar.gz
sudo tar -xzf powering-up-with-puppet.tar.gz
```

The directory structure should now exist at /etc/puppet. We are ready to start the Puppetmaster service. Since this is the first time we run it, we should also add the 'mkusers' parameter.

```
sudo puppet master --mkusers --verbose
```

From now on, whenever we need to start the puppetmaster service (say after a system restart), we only need to type in the following:

```
sudo puppet master
```

[CLIENT] Requesting a certificate

Now that we have Puppet installed on the server and the client, and the Puppetmaster service started, we can test that the connection works. To do this, we run the following command (replace puppetmaster.example.com with the actual hostname of the management server):

```
sudo puppet agent –test –server='puppetmaster.example.com'
```

Since puppet master hasn't signed a certificate for the client yet, there will be some errors. However, it should also create a new SSL certificate request.

[SERVER] Signing the certificate

On the host running Puppetmaster, type the following:

```
sudo puppetca --list
```

This will print out a list of all the cerfiticate requests the host has received. You should see the client (in this case "puppetclient.example.com") on the list. The final thing we need to do is to sign the certificate:

```
sudo puppetca --sign 'puppetclient.example.com'
```

This should produce a message stating that the certificate was signed.

[CLIENT] Testing the connection

If everything has worked successfully up to this point, we should now have a working connection between the Puppet master and Puppet client. We can test it with the following command:

```
sudo puppet agent --test --server='puppetmaster.example.com'
```

Since there are no tasks specified yet, the checkout will not actually do anything except output a few lines. If the connection works, you should see something similar to the following:

```
info: Caching catalog for puppetmaster
info: Applying configuration version '1298651839'
notice: Finished catalog run in 0.25 seconds
```

APPENDIX 3. Tutorial on installing and configuring Chef Server and Chef Client

Overview

This is a tutorial on how to install Chef Server and Chef Client on two separate machines running fresh installs of Ubuntu 10.04. Unless otherwise stated, the following steps are required for both instances. There are multiple ways to install Chef. In this tutorial we use Chef-solo to bootstrap Chef Server and Chef Client on the target machines.

Installing the prerequisites

Install the following packages via apt:

```
sudo apt-get install ruby rubygems ruby-dev libopenssl-ruby rdoc ri irb
build-essential wget ssl-cert ssh
```

Install RubyGems 1.3.7 from source:

```
wget http://production.cf.rubygems.org/rubygems/rubygems-1.3.7.tgz
tar xvfz rubygems-1.3.7.tgz
cd rubygems-1.3.7
sudo ruby setup.rb
```

Installing Chef Solo

Use the following command to install Chef Solo:

```
sudo gem install chef
```

Modifying the HOSTS file

On both machines, modify the file */etc/hosts* so that *chef.example.com* points to the Chef Server machine and *client.example.com* points to the Chef Client machine.

[SERVER] Bootstrapping Chef Server

Create the file */etc/chef/solo.rb* with the following contents:

```
file_cache_path "/tmp/chef-solo"
cookbook_path "/tmp/chef-solo/cookbooks"
recipe_url "http://s3.amazonaws.com/chef-solo/bootstrap-latest.tar.gz"
```

Create the file */etc/chef/chef.json* with the following contents:

```
{
"bootstrap": {
"chef": {
"url_type": "http",
"init_style": "runit",
"path": "/srv/chef",
"serve_path": "/srv/chef",
"server_fqdn": "chef.example.com",
"webui_enabled": true
}
},
"run_list": [ "recipe[chef-server::rubygems-install]" ]
}
```

Next, use Chef Solo and the newly created files to bootstrap the Chef Server install:

```
sudo chef-solo -c /etc/chef/solo.rb -j /etc/chef/chef.json
```

[SERVER] Configuring the Chef Server

First, create the directory *~/.chef*, copy the validation files from */etc/chef* there and give the current user the permissions to that directory:

```
sudo mkdir -p ~/.chef
sudo cp /etc/chef/validation.pem /etc/chef/webui.pem ~/.chef
sudo chown -R $USER ~/.chef
```

Next, we should create an initial configuration by running the following commands:

```
cd ~
sudo knife configure -i
```

The process is interactive, asking the user for a series of settings. You can accept the default settings for most of the questions by pressing enter. The only ones you need to change are the following:

- **chef server URL**: http://chef.example.com:4000
- **admin client's private key:** .chef/webui.pem
- **validation key:** .chef/validation.pem

[CLIENT] Bootstrapping the Chef Client

Create the file */etc/chef/solo.rb* with the following contents:

```
file_cache_path "/tmp/chef-solo"
cookbook_path "/tmp/chef-solo/cookbooks"
recipe_url "http://s3.amazonaws.com/chef-solo/bootstrap-latest.tar.gz"
```

Create the file */etc/chef/chef.json* with the following contents:

```
{
"bootstrap": {
"chef": {
"url_type": "http",
"init_style": "runit",
"path": "/srv/chef",
"serve_path": "/srv/chef",
"server_fqdn": "chef.example.com",
"webui_enabled": true
}
},
"run_list": [
"recipe[chef-client::service]",
"recipe[chef-client::config]"
]
}
```

Next, use Chef Solo and the newly created files to bootstrap the Chef Client install:

```
sudo chef-solo -c /etc/chef/solo.rb -j /etc/chef/chef.json
```

[CLIENT] Configuring the Chef Client

On the client, we should now have a file called */etc/chef/client.rb*. Edit the file, modifying/adding the following lines:

```
chef_server_url http://chef.example.com:4000
validation_key "/etc/chef/validation.pem"
client_key "/etc/chef/client.pem"
```

Now we need to copy the validation key */etc/host/validation.pem* from the host to the client. After you have done this, run the following command on the client to generate the file */etc/chef/client.pem:*

```
sudo chef-client
```

After the *client.pem* file is created, you can remove the *validation.pem* from the client as it is no longer needed.

[CLIENT] Testing the connection

At this stage, the connection between the Chef Server and Chef Client should be working. You can test it by running chef-client again. It should produce a line similar to this:

```
WARN: Node <nodename> has an empty run list.
```

This means that the connection works. As there are no run lists assigned to the node yet, it will not actually do anything.

APPENDIX 4. Fabric usage examples

USE CASE 1

In this use case, we modify the contents of the HOSTS file on the target node. More specifically, we use Fabric to modify the hostname and domain name for a given IP address.

Creating a fabfile

First, we move to a folder of our choosing and create the file *fabfile.py.* Using this filename allows Fabric to detect the file automatically. Add the following Python code into the file:

```
from fabric.api import *
import fileinput
from config import *

env.user = user
env.password = password
env.hosts = target_hosts

def modify_host():

           file = get(filename, "tmp")

           for line in fileinput.FileInput(file, inplace = 1):
                      if ip in line:
                                 line = ip + " " + name
                                 print line
                      else:
                                 print line,

           put("tmp", filename, use_sudo=True)
```

In the first three lines, we import some libraries for specific functions, as well as our own custom configuration file, *config.py.* In the next three lines, we set some Fabric environment variables according to the values found in the imported config file.

Finally, there is the file modifying function, *modify_host()*. This is mostly just ordinary Python code. The function opens a file, goes through every line, and if it finds the given IP address, it adds the given name into that line. The only Fabric-related functions are on the first and last lines, *get* and *put.* The *get* function downloads the file from the node to the management server for editing. Next, we process the changes, and finally, send the file back to the node with the *put* function.

Creating a config file

It is convenient to hold the configuration variables in their own file. We will call this file config.py. Add the following contents:

```
target_hosts  = ['192.168.11.12']
user = "username"
password = "password"
filename = "/etc/hosts"
ip = "192.168.11.6"
name = "hostname.example.com"
```

- **target_hosts** is a list of nodes to be configured. In this case, we only have one. If you need to specify multiple nodes, use comma as a delimiter between each IP address.
- **user** is the username with which the changes will be done. Replace “username” with a real user name.
- **password** contains the user’s password. Replace “password” with the actual password.
- **Filename** specifies the file that is to be modified on the target node.
- **ip** in this case is the IP address to search for in the HOSTS file.
- **name** is the host/domain name to be added for the given IP address.

Testing the fabfile

Now that we have our files ready, we can test how they work. Assuming that Fabric is installed, this can be done by running the *fab* command with the name of our function as a parameter:

```
sudo fab modify_host
```

Since we already have all the needed values (IP address, username, password) in the config file, this should do the trick. The output should be similar to the following:

```
[192.168.11.12] Executing task 'modify_host'
[192.168.11.12] download: /home/user/fabric/tmp <- /etc/hosts
[192.168.11.12] put: tmp -> /etc/hosts
Done.
Disconnecting from 192.168.11.12… done.
```

If it worked, the HOSTS file on the target node should now be changed accordingly.

USE CASE 2

This example demonstrates how to use Fabric to install a specific version of a given package on the target node.

Creating a fabfile

Our goal here is to install a given package on the target node IF the currently installed version is older than the given version number. For this, we can use something similar to the following script:

```
from fabric.api import *

env.hosts = ['x.x.x.x']
env.user = "username"
env.password = "password"

package name = "curl"
package_version = "7.19.7"

def isPackageInstalled(pkg):

    output = run("dpkg-query -W -f='${Status} ${Version}'" + " " + pkg)

    if (output.find("install ok") != -1):
        return True

    return False

def getPackageVersion(pkg):

    output = run("apt-cache showpkg " + pkg)
    version = output.splitlines(3)[2].split("-")[0]
    return version

def install_package():

    with hide('running', 'stdout', 'stderr'):

        install = 1
        print "\n"

        if (isPackageInstalled(package_name) == True):
            version = getPackageVersion(package_name)
            Print "Version: " + version
            if (version >= package version):
                install = 0
                print "Already up to date, nothing to do"
            else:
                print "Older version detected, updating..."
        else:
            print "Package not found, installing..."

        if (install == 1):
            sudo('apt-get install ' + package name)
```

Replace “x.x.x.x” with the target node's ip address, "username" with the user name and "password" with the user's password. Package_name indicates the package to be installed, while package_version is the given version number. Here is a short description of each function:

- **isPackageInstalled(pkg)**: runs a command on the node and parses the results to check whether or not the given package is installed. Returns TRUE or FALSE.
- **getPackageVersion(pkg)**: runs a command on the node and parses the results to find out the version of the given package. Returns the version as a string.
- **install_package()**: The main block, checks the state and version of the package. If needed, installs the package using apt-get. To make the output cleaner, the 'with hide()' statement is used here to hide most of the output from the user.

After the script is ready, all we need to do to install the package is run the following command:

```
fab install_package
```

After this, you can go to the client machine to check if the package was installed properly.

USE CASE 3

In this use case, we demonstrate how to use Fabric to configure multiple target nodes, each with their own specific configuration settings.

As the earlier examples of Fabric demonstrate, Fabric includes an environment variable called *env.hosts*. This variable is an array of host addresses that can be modified in a Fabric script. This means that configuring multiple servers is quite simple: just specify all the addresses where the script is to be applied (example below).

```
env.hosts = ['192.168.11.4', '192.168.11.5']
```

We can also group host addresses together using *roles*. Below is an example of a role definition:

```
env.roledefs['webservers'] = ['www1', 'www2', 'www3']
```

The role can then be used with the following commandline parameter:

```
-R 'webservers'
```

Using the above parameter, Fabric connects to each of the specified webservers (www1, www2 and www3).

APPENDIX 5. Puppet usage examples

USE CASE 1

Unfortunately, Puppet does not have a built-in feature for text editing of files. This can be remedied by making use of the user added definitions found behind the following link:

http://projects.puppetlabs.com/projects/1/wiki/Simple_Text_Patterns.

We will be using the definition “ensure_key_value”. It suits our needs perfectly, as it finds a given string at the beginning of a line and changes the rest of that line. Note that this definition requires the GNU sed editor (comes with Ubuntu default installation) to work.

Creating a Puppet module [SERVER]

For this example, we will create a module for modifying the target machine’s HOSTS file. On the management server, navigate to the Puppet directory structure, go to the *manifests* folder and create the following file by the name “nodes.pp”:

```
node “puppetclient.example.com” {
            include modify_hosts
}
```

The above lines instruct Puppet that the class “modify_hosts” should be included when configuring the node “puppetclient.example.com”. For this to be of any use, however, we need to create the actual “modify_hosts” class. We will do this next.

Navigate to the modules directory, create a new directory called “modify_hosts” with a subdirectory “manifests”. In the subdirectory, create the following file with the name “init.pp”:

```
class modify hosts {

define ensure_key_value($file, $key, $value, $delimiter = " ") {
    # append line if "$key" not in "$file"
    exec { "append $key$delimiter$value $file":
        command => "echo '$key$delimiter$value' >> $file",
        unless => "grep -qe '^[[:space:]]*$key[[:space:]]*$delimiter' -- $file",
        path => "/bin:/usr/bin:/usr/local/bin",
        before => Exec["update $key$delimiter$value $file"],
    }

    # update it if it already exists...
    exec { "update $key$delimiter$value $file":
        command => "sed --in-place='' --
expression='s/^[[:space:]]*$key[[:space:]]*$delimiter.*$/$key$delimiter$value/g'
$file",
        unless => "grep -xqe '$key$delimiter$value' -- $file",
        path => "/bin:/usr/bin:/usr/local/bin"
    }
}

ensure_key_value { “hosts”:
    file => “/etc/hosts”,
    key => “192.168.11.8”,
    value => “newname”
}

}
```

Although the class looks a bit bulky, most of it is just part of the definition that we grabbed from the web. In the final lines, we run the definition and give it the required parameters:

- **File:** The name of the file to modify
- **Key:** The IP address to search for
- **Value:** The text to enter after the IP address

Note that sometimes the HOSTS file uses tabs as delimiters. In that case, we need to add the parameter “delimiter” and make its value a tab space:

```
delimiter => “          “
```

Testing the Puppet module [CLIENT]

Now we can test our newly created module to check that it works as wanted. Run the following command:

```
sudo puppet agent –test –server='puppetmaster.example.com'
```

If everything works okay, Puppet should output something similar to the following:

```
info: Caching catalog for puppetclient.example.com
info: Applying configuration version '1332534849'
notice:
/Stage[main]/modify_hosts/modify_hosts::Ensure_key_value[hosts]/Exec[update
192.168.11.8 newname /etc/hosts]/returns: executed successfully
notice: Finished catalog run in 0.29 seconds
```

The HOSTS file on the target node should now be modified accordingly.

USE CASE 2

In this use case, we ensure that a specific version of a given package is installed on the target node(s). This example assumes that you already have Puppet installed and set up on both the server and the client. Server in this case means the management server, while the clients are the target machines to be configured.

Creating a Puppet module [SERVER]

For this example, we will create a simple module for installing the package "curl". On the management server, navigate to the Puppet directory structure, go to the *manifests* folder and create the following file by the name "nodes.pp":

```
node "puppetclient.example.com" {
            include curl
}
```

The above lines instruct Puppet that the class "curl" should be included when configuring the node "puppetclient.example.com". For this to be of any use, however, we need to create the actual "curl" class. We will do this next.

Navigate to the modules directory, create a new directory called “curl” with a subdirectory “manifests”. In the subdirectory, create the following file by the name “init.pp”:

```
class curl {
           package {‘curl’:
                      ensure => ‘7.19.7-1ubuntu1.1’,
           }
}
```

The above lines instruct Puppet that in the class “curl”, we want to ensure that version “7.19.7-1ubuntu1.1” of the package named “curl” is installed on the target machine. Note that the class name and package name do not need to be the same, although they are in this example.

Testing the Puppet module [CLIENT]

Now we can test our newly created module to check that it works as wanted. Make sure that curl is not yet installed on the client, and run the following command:

```
sudo puppet agent –test –server=’puppetmaster.example.com’
```

If everything works okay, Puppet should output something similar to the following:

```
info: Caching catalog for puppetclient.example.com
info: Applying configuration version ‘1332534849’
notice: /Stage[main]/curl/Package[curl]/ensure: ensure changed ‘purged’ to
‘present’
notice: Finished catalog run in 4.69 seconds
```

After that, you can try running the following command to make sure curl is actually installed:

```
curl --version
```

USE CASE 3

In this use case, we demonstrate how to use Puppet to configure multiple target nodes, each with their own specific configuration settings.

In Puppet, we can use the file “nodes.pp” to describe the required configuration modules for each node. Here is an example of a simple nodes.pp file:

```
node 'puppet1.example.com' {
            include web
}
node 'puppet2.example.com' {
            include curl
}
```

This tells Puppet that the web class is required by the node ‘puppet1.example.com’, while the class curl is required by ‘puppet2.example.com’. This means that in the next update run, puppet1.example.com will include the *web* class, while puppet2.example.com will include the class *curl*. To make things more flexible, we can create a base node type and then inherit the classes of that node in another node. Here is an example of using inheritance:

```
node webnode {
            include web
            include db
            include apache
}
node 'puppet1.example.com' inherits webnode {
}
node 'puppet2.example.com' inherits webnode {
}
```

Using inheritance, we can attach similar types of nodes to a similar configuration without having to modify the includes of each node individually.

APPENDIX 6. Chef usage examples

TIP: If at any time you lose connection between the Chef Server and Chef Client, you can try restarting the Chef Server service with the following command:

```
sudo /etc/init.d/chef-server restart
```

USE CASE 1

In this use case, we modify the contents of the HOSTS file on the target node. More specifically, we use Chef to modify the hostname and domain name for a given IP address.

Creating a cookbook

We will call our cookbook “hosts”. Use the following command to create the directories and files for the cookbook:

```
sudo knife cookbook create hosts
```

Knife should now have created the directory structure for the cookbook, located in the cookbook directory – by default */var/chef/cookbooks*. Navigate to *cookbooks/hosts/recipes*. Edit or create the file *default.rb*, adding the following lines:

```
require “fileutils”

FileUtils.cp( “/etc/hosts”, “/etc/hosts tmp” )

read file = File.open( “/etc/hosts tmp”, “r” )
write_file = File.open( “/etc/hosts”, “w” )

ip = “xxx.xxx.xxx.xxx”
name = “newname.example.com”

read file.each { |line|
if line =~ /#{ip}\s(.*)/
    write_file.puts(ip + “ “ + name)
else
    write file.puts(line)
end
}

read_file.close
write file.close
FileUtils.rm(“/etc/hosts_tmp”)
```

The string “ip” contains the IP address to look for, while the string “name” contains the new domain name. As you may notice, the above script is just ordinary Ruby code. Chef allows us to run Ruby scripts on the target machine, so we can use the above code to modify the HOSTS file. In short, the script makes a copy of the current HOSTS file, copies that file line by line into the actual HOSTS file, modifying the domain name if the line includes the given IP address. After the processing is done, the files are closed and the temporary file is removed.

Adding the cookbook to a runlist

We will assume that the target node’s hostname is client1. This means that Chef will automatically create a node called “client1” when that client connects. To add the newly created cookbook to the node’s runlist, use the following command:

```
sudo knife node run_list add client1 hosts::default
```

With the above command, we add the recipe “default” from the cookbook “hosts” to the runlist of client1.

Testing the cookbook

To test the newly created cookbook, simply run the following command on the client machine:

```
sudo chef-client
```

If working, the domain name of the specified IP address should now be changed in the HOSTS file.

USE CASE 2

This use case demonstrates how to install a specific version of a given package on a node using Chef.

Creating a cookbook

We will be installing the package “curl”, so we will also use that as the name of our new cookbook. Use the following command to create the directories and files for the cookbook:

```
sudo knife cookbook create curl
```

Knife should now have created the directory structure for the cookbook, located in the cookbook directory – by default */var/chef/cookbooks*. Navigate to *cookbooks/curl/recipes*. Edit or create the file *default.rb*, adding the following lines:

```
package “curl” do
            version “7.19.7-1ubuntu1.1”
            action :install
end
```

This tells Chef Server that we want to *install* version *7.19.7* of the package *curl*. Before we can use the cookbook, we have to upload it to the Chef Server’s list of cookbooks:

```
sudo knife cookbook upload curl
```

Adding the cookbook to a runlist

We will assume that the target node’s hostname is client1. This means that Chef will automatically create a node called “client1” when that client connects. To add the newly created cookbook to the node’s runlist, use the following command:

```
sudo knife node run_list add client1 curl::default
```

With the above command, we add the recipe “default” from the cookbook “curl” to the runlist of client1.

Testing the cookbook

To test the newly created cookbook, simply run the following command on the client machine:

```
sudo chef-client
```

If working, it should produce output similar to the following:

```
INFO: Processing package[curl] action install
INFO: package[curl] installed version 7.19.7.1ubuntu1.1
```

USE CASE 3

In this use case, we use Chef to configure multiple nodes, each with their own specific configuration.

Creating the cookbooks

First, we create two cookbooks; one for each node. Use the following commands to create the directories and file for the cookbooks. In this example, we will call the cookbooks “a” and “b”.

```
sudo knife cookbook create a
sudo knife cookbook create b
```

Knife should now have created the directory structure for these cookbooks, located in the cookbook directory – by default */var/chef/cookbooks*. Navigate to *cookbooks/a/recipes*. Edit or create the file *default.rb*, adding the following line:

```
package "curl"
```

Note that only this one simple line is needed for the basic installation of a package. Do the same for cookbook b, but add the following line instead:

```
package "wget"
```

Before we can use these cookbooks, we need to upload them to Chef Server with the following commands:

```
sudo knife cookbook upload a
sudo knife cookbook upload b
```

We now have two very basic cookbooks ready for use.

Adding the cookbooks to runlists

Now we need to instruct Chef which cookbooks are required by which nodes. We can do this by editing the run-lists of the nodes. The node entries themselves should be automatically created with the client's hostname when the Chef Clients connect to the Chef Server. For this example, we'll assume the two Chef Client hostnames are client1 and client2. Since we already have our cookbooks ready, all we need to do now is run the following commands:

```
sudo knife node run_list add client1 a::default
```

With the above command, we add the recipe "default" from the cookbook "a" to the runlist of client1. Let's proceed similarly with client2:

```
sudo knife node run_list add client2 b::default
```

Instead of using the cookbook a, we use the second cookbook called “b” for the node client2.

Testing the settings

We can now test our newly made Chef configuration. Run the following command on each of the clients:

```
sudo chef-client
```

If everything works as planned, client1 should now have the package “curl” installed, while client2 will install “wget” instead.