UNIVERSITY OF TARTU
FACULTY OF SCIENCE AND TECHNOLOGY

Institute of Computer Science
Computer Science Curriculum

Dmitri Tsumak

Large-Scale Provisioning and Configuration Management

Bachelor's Thesis (9 ECTS)

Supervisor: Artjom Lind, MSc

Tartu 2016

Large-scale provisioning and configuration management

Abstract:
Nowadays, many companies involve provisioning large number of servers. Those servers have different roles and respond for running multiple services, which should be maintained in a company. As a result, system administrators should come up with a solution to effectively manipulate those servers and services installed on them. One popular approach is to use configuration management tools.
In this research we will describe problems that system administrators face during large-scale provisioning. Furthermore, we will review and compare most popular configuration management tools to see how they solve these problems. In addition, we will describe the process of implementing and testing several Ansible configuration management scripts.

Keywords: large-scale provisioning, configuration management, system administration, Ansible in practice, configuration management scripts testing.

CERCS: T120, Systems engineering, computer technology

Suurte süsteemide seadistamine ja konfiguratsiooni haldamine

Lühikokkuvõte:
Tänapäeval on paljudel ettevõttetel suur arv servereid. Igal serveril on oma roll ning ülesanded, mida ta peab täitma. Nende ülesannete täitmise eest vastutavad rakendused, mis on nende serverite peale installeeritud. Seega suure serverite ja rakenduste arvu haldamiseks peab süsteemi administraatoritel olema kindel lähenemine. Nende ülesannete optimeerimisega tegeleb selline valdkond nagu "Configuration Management".
Selles teadustöös vaadeldakse probleeme, mida süsteemiadministraator peab lahendama suurte süsteemide korraldamisel. Lisaks analüüsitakse ja võrreldakse siin erinevaid kõige sagedamini kasutatavaid "Configuration Management"valdkonna tööriistu ning seda, mis lahendusi need tööriistaad pakkuvad püstitatud probleemide lahendamiseks. Lõpus kirjeldatakse kahe skripti implementeerimis- ja testimisprotsesse kasutades Ansible tööriista.

Võtmesõnad: suurte süsteemide seadistamine, konfiguratsiooni haldamine, süsteemide administreerumine, Ansible praktikas, konfiguratsiooni haldamise skriptide testimine.

CERCS: T120, Süsteemi konstrueerimine, infotehnoloogia

Contents

List of Figures

Listings

List of Terms

availability is an amount of time a system is in a functioning condition [22]. 6

bash is a scripting language for Unix command lines. 6

bootstrapping is a process of initial setup and configuration of the system. 14

cloud is a large-scale infrastructure with shared resources. 6

devops "is an enterprise software development phrase used to mean a type of agile relationship between Development and IT Operations." [10]. 6

git is one of the most popular source code management system. 14

immutable servers are machines, which run a particular service and can create, destroy or replace any of these machines without causing this service disruption. 23

kerberos is a network authentication protocol which uses "tickets" for nodes authentication[17]. 17

KVM is a kernel-based virtual machine [27]. 29

large-scale system is the system with large amount of hardware, number of running services and involved users. 6

monitoring is the process of collecting and parsing various data to be aware of systems and services states. 25

NFS is a network file system. Allows to store and access files on remote server. 30

scalability is an ability of the system to be expanded to better handle growing amount of work [11]. 6

serialization is the process of converting data from one format to another. For example, Java object can be converted into JSON data. Reverse process is called deserialization. For example, converting JSON data back to Java object. 8

workstation is a system, which is used for work. In configuration management it also refers to the system, from which commands are executed. 13

1 Introduction

Nowadays provisioning of large systems is a crucial task, which requires an intimate knowledge and a lot of experience in system administration area. Devops teams should be ready to overcome difficulties during planning, construction as well as maintenance of a large-scale systems such as clouds. After infrastructure is carefully planned and hardware is ready for use, system administrators start their work. Their task is to deploy and configure services and tools required for proper work of the system. They should think over such topics as system availability, scalability, security and redundancy. In addition, they should setup tools for monitoring, backup and logging system's work as well as services requested by customer.

It is a well known fact that configuration and maintenance of large-scale server infrastructures introduce a challenge for system administrators [12]. One of the objectives which they should be ready to overcome is continuous deployments of identical services across largely identical servers. For example, most of the cloud systems should have multiple database servers for better performance and load balancing. Therefore, manual configuration of numerous almost equal setups offer a chance for optimization and usually it is up to administrator's experience how to avoid the maintenance overhead. Common solutions rely on writing custom Bash scripts which obviously requires advanced Unix administration skills. The result scripts are hardly maintainable in team as there is no strictly defined coding conventions in many Unix scripting languages and structure of custom written scripts can vary according to the specialist's preferences. In consequence, system administrators should prepare instructions for other team members and users about aims of the scripts and how exactly they should be executed. Moreover, there is no version control system for configuration files in Unix by default. Therefore, administrators do not have enough information concerning the system's state. For example, it is impossible to check whether configuration files in server were updated without explicitly checking them. Furthermore, there is often a necessity in creating multiple environments, e.g. developing, staging and production during the development process of the large system. As a result, there should be an established workflow for tracking, updating and rolling back the configuration of every environment.

The systems engineering process which deals with described problems is called configuration management. There were multiple open-source solutions developed for managing configurations, which are widely in use. This research will focused on tools that are the most acknowledged by system administrators. Using these tools development teams spend less time on tracking, monitoring and backing up already deployed services and can easily deploy another service by using already defined installation instructions. In addition, it will be demonstrated how these

tools simplify the work of system administrator when dealing with large-scale cloud setups.
This thesis consists of five chapters. In second chapter configuration management field will be introduced and most popular open-source tools will be described and compared. In third chapter an overview of Ansible features in details will be made. In fourth chapter will be demonstrated Ansible usage in real life scenarios with some practical advises. In last chapter a conclusion and suggestions about further research will be given.

2 Related Work

In this chapter most important points of different configuration management tools will be introduced.

2.1 Introduction to configuration management

Configuration management (CM) is a system engineering process of tracking and consistent development of the product. It is also used in such fields as military, civil and industrial engineering [33]. In computer science two types of CM exist: software and system.
Software configuration management is used in software projects to handle and track application changes [33]. It allows developers to ensure that finally delivered software satisfies all requirements. System configuration management allows user to deploy, track and maintain services and applications on a server. There are even tools that allow to create snapshots of system's configuration states and roll back to them if needed. However, there are plenty of CM tools available nowadays and their features may vary. This research will be focused on system configuration management tools.

Good configuration management tool should have following features:

F-1 deploy, update, remove configuration files and services

- Basic operations that every configuration management tool should support.

F-2 version control capabilities

- It should be possible to track changes in configuration files. In case user wants to revert changes, it should be possible to roll back to the system's previous state.

F-3 execute single command remotely

- In case user wants to make a hot fix on the server, it should be possible to do this without executing the whole project. For example, it should be possible to update repositories cache on all managed nodes without creating separate project for that.

F-4 system scalability

- It should be possible to add additional services, configurations without a large effort. As a result, tool must be suitable not only for small, but also for large-scale setups.

F-5 OS independency

- Configuration management tool should allow to deploy services on most popular Unix-type systems without making huge changes in the project.

F-6 agent-less

- If it is not required to install any dependencies on the managed node, it would be an advantage of the tool.

F-7 code readability

- Due to the fact that large-scale systems are mostly set up by teams, great configuration management tool should have strictly structured and human-readable code and configuration files.

F-8 support

- It will be much easer for new users to get familiar with a tool, when it has large community and nice documentation.

F-9 free and open-source

- Tool should have at least limited free version with open-source code. For large-scale setups and better support it can be commercial.

In the next sections an overview of several configuration management tools will be made. For every tool it will be checked what features from the list described above it includes.

2.2 SaltStack

SaltStack features:

☒ deploy, update, remove configuration files and services (Refer to **F-1**)

☐ version control capabilities (Refer to **F-2**)

☒ execute single command remotely (Refer to **F-3**)

☒ system scalability (Refer to **F-4**)

☒ OS independency (Refer to **F-5**)

☒ agent-less (Refer to **F-6**)

☒ code readability (Refer to **F-7**)

☐ support (Refer to **F-8**)

☒ free and open-source (Refer to **F-9**)

SaltStack is an open-source configuration management tool and engine which can execute commands remotely. It is quite popular among cloud infrastructure developers due to its scalability and resiliency [20]. Project was started by Tomas Hatch in 2011 and was written in Python[37]. SaltStack is a modular service which allows user to connect or disconnect different modules. It allows not only to configure servers, but also to execute commands remotely and monitor system's work. Most of the SaltStack configuration files are written in **YAML**, which is very human-readable serialization language. In order to implement more specific configuration files, **Jinja2** language is used, which has more flexible syntax than YAML, but is less readable. SaltStack is based on the master-client model and consists of following components [36]:

- **Salt Master** - system that is responsible for sending commands and configurations to the salt minions. Similar to the master node in clusters terminology.
- **Salt Minions** - systems that receive commands and configurations from salt master. Similar to the slave nodes in clusters terminology.
- **Execution Modules** - commands that are executed from the salt master on the salt minions. Mostly used for the configuration hot fixes and real-time monitoring. For example, executing

```
$ salt-ping '*'
```

Listing 1: Ping salt minions

  will try to ping all the minions (* means all) connected to salt master from which command was executed (refer to Figure 1).

- **Formulas (States)** - shows the state of a system's configuration. For example, following command

```
$ salt '*' state.sls redis
```

Listing 2: Salt command to get the state of Redis service

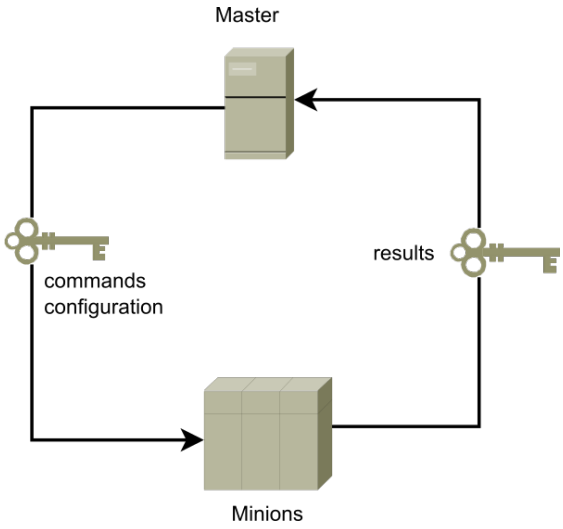

Figure 1: Communication between salt master and minions

will return the state of the Redis service on all salt minions:

```
redis-server:
    pkg.installed:
        - name: redis-server
service:
    - running
```

Listing 3: Response of salt state command for Redis service

- **Grains** - system variables, which hold information about operation system, memory and other properties of managed system.
- **Pillars** - user-defined variables, which store information about ports, file paths, configuration parameters and passwords. They are securely defined on the Salt master and are connected to one or more minions. For example, it is possible to define variables according to the minion's operation system using pillars:

```
1 {% if grains['os'] == 'RedHat' %}
2 apache: httpd
3 {% elif grains['os'] == 'Debian' %}
4 apache: apache2
5 {% endif %}
```

Listing 4: Salt apache variable depending on the OS family

If the command is executed on RedHat-type system (Fedora, CentOS, etc.), apache variable will be `httpd`. If it is executed on Debian-type system (Debian, Ubuntu, etc.), apache variable will be `apache2`.

- **Runners** - modules that run on the Salt master to perform different operations on the minions.
- **Returners** - modules that can be used on the Salt minions to send data to third party services such as databases.
- **Reactros** - triggers that execute functions when some event occurred.
- **Salt cloud** - makes possible to provision systems on different cloud providers, such as OpenStack and bring those under salt management.

As a result, SaltStack has a large number of components and each of them has its own function. It is also worth noticing that compared to many others CM tools Salt master communicates with minions using two-way authentication and encrypts all the traffic between them. In addition, it supports agent-less communication. It means that it is possible to execute commands from Salt master without installing minion agent on the system. The huge advantage of SaltStack is also platform independency, which allows to execute same commands on Windows and Unix operation systems. For those, who do not feel comfortable with shell commands, it is possible to use SaltStack Web GUI.

However, SaltStack is not recommended for newcomers, because its setup is challenging (there should be quite many components configured) and documentation is quite demanding to the reader in terms of system administration knowledge [20]. In spite of the fact that it is multi-platform tool, there are some problems introduced in operating with non-Linux operation systems. For example, minions installed on Windows systems send response to the master much slower [26]. In addition, SaltStack Web GUI is not fully optimized and not intuitive for many users.

2.3 Puppet

Puppet features:

- ☒ deploy, update, remove configuration files and services (Refer to **F-1**)
- ☒ version control capabilities (Refer to **F-2**)
- ☐ execute single command remotely (Refer to **F-3**)

☒ system scalability (Refer to **F-4**)

☒ OS independency (Refer to **F-5**)

☐ agent-less (Refer to **F-6**)

☒ code readability (Refer to **F-7**)

☒ support (Refer to **F-8**)

☒ free and open-source (Refer to **F-9**)

Puppet is another popular open-source configuration management tool. Compared to SaltStack it has been developing for 11 years and has great community behind it. Puppet is developed by Puppet Labs, which was founded by Luke Kanies in 2005 [34]. The project is written in Ruby and has commercial, as well as enterprise editions. Puppet is based on master-client model and is used for tracking, testing and deploying configuration files on managed servers. Furthermore, with Puppet it is possible to scale the infrastructure and deploy required services with one command.

Puppet installation process is very simple[23]:

```
$ apt-get install puppet       # On clients (nodes)
$ apt-get install puppetmaster # On server (master)
```

Listing 5: Puppet installation

Puppet uses declarative domain specific language (DSL) similar to JSON to define system's states in files called **manifests**. In those files users should define so called **resources**. They are used to describe what files, packages, services, etc. should be installed and how they should be configured on managed servers [23]. Example of resource declaration[29], which changes attributes of `/etc/passwd` file:

```
1 file { '/etc/passwd':
2   ensure => file,
3   owner  => 'root',
4   group  => 'root',
5   mode   => '0600',
6 }
```

Listing 6: Puppet resource

Resources are grouped into **classes** with parameters, which can change their behavior during runtime. For example, here class is defined for configuring attributes of `/etc/passwd` and `/etc/shadow` files. It can be noticed that this class consists

of two resources and can take optional `user` variable, which defines `owner` and `group` for these two files [28]:

```
class linux (String $user = 'root') {
  file { '/etc/passwd':
    owner => $user,
    group => $user,
    mode  => '0644',
  }
  file { '/etc/shadow':
    owner => $user,
    group => $user,
    mode  => '0440',
  }
}
```

Listing 7: Puppet class

There are also **modules**, which can help to split Puppet code into multiple manifest files. There are plenty of modules already available in Puppet community called **Puppet Forge**, which are free to download [35].

The process of applying configuration changes can be described with following figure:

It consists of following steps:

1. **Agent** sends gathered facts and variables about managed node to the master node.
2. **Master** verifies received facts and variables, determines what host is it and what should be installed on it.
3. **Master** prepares manifests for the execution, compiles them and sends them to the agent.
4. **Agent** receives compiled manifests as **catalogs**, applies them and sends report about results to the master node.

Overall, Puppet is a good choice if stability and maturity are important aspects of deployments [20]. It is more suitable for larger-scale systems and has dozens of configuration modules available in Puppet Forge. In addition, Puppet's interface is recognized as one of the most advanced and user-friendly in this field. Similarly to SaltStack, it is a multi-platform tool. However, it has much easier setup than

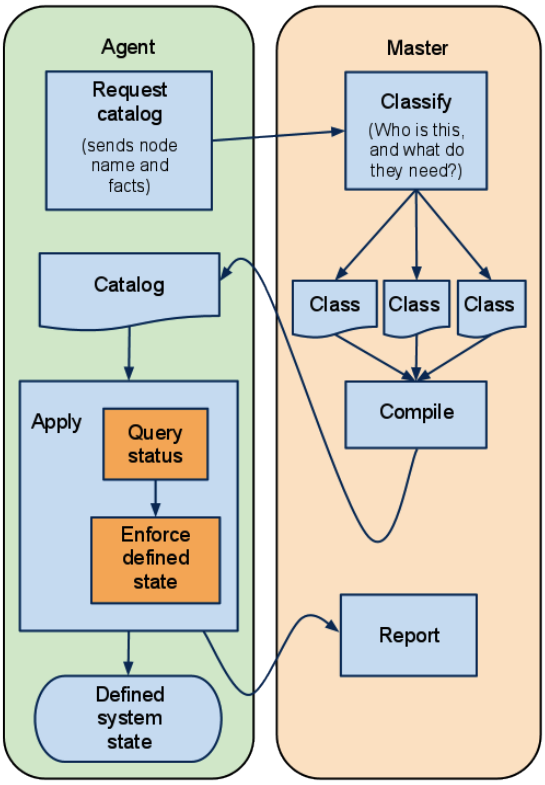

Figure 2: Communication between Puppet master and agent, (c) Lauren Malhoit [32]

SaltStack and is more suitable for those, who are new in configuration management area. It has a great documentation and community.

Nonetheless, there are some disadvantages that mostly occur during large-scale systems provisioning. Namely, completion of more complicated tasks requires Puppet command-line interface knowledge, which is written in Ruby. It means that in some situations it is required to understand Ruby syntax. In addition, large number of manifest files can decrease their maintainability and become a problem for new team members to understand the code.

2.4 Chef

Chef features:

- ☒ deploy, update, remove configuration files and services (Refer to **F-1**)
- ☒ version control capabilities (Refer to **F-2**)
- ☐ execute single command remotely (Refer to **F-3**)
- ☒ system scalability (Refer to **F-4**)
- ☒ OS independency (Refer to **F-5**)
- ☐ agent-less (Refer to **F-6**)
- ☐ code readability (Refer to **F-7**)
- ☒ support (Refer to **F-8**)
- ☒ free and open-source (Refer to **F-9**)

Chef is another open-source configuration management tool written in Ruby by Chef company. However, it is designed more for developers than system administrators, because the way configuration files, services and software will be deployed on servers should be described in Ruby programming language. As a result, knowledge of Ruby is one of the prerequisites for using this tool.
Chef uses master-client communication approach, but with one difference: nodes can be managed not only from the master node, instead it is possible to manage them from the workstation (Refer to Figure 3).

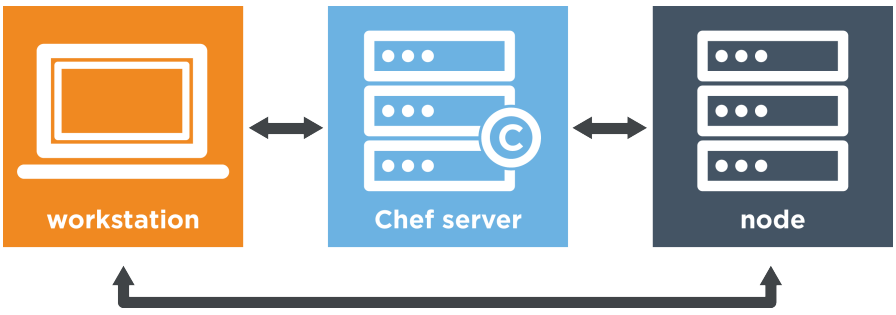

Figure 3: Communication between Chef elements, (c) Chef [13]

In fact, this approach is very similar to the development process of software project. It consists of following steps:

1. Writing the code on the workstation.

    - The code is organized into recipes, which are grouped into cookbooks (more on that later). In brief, cookbooks describe how the nodes are going to be configured. In order to communicate with master node, Chef development kit should be installed and `knife.rb` [16] configuration file should be created, which contains information about the master node and private keys locations.

2. Pushing the code to the master node.

    - It is analogous to the remote source code repository where all the cookbooks reside. Similarly to the Git, it is possible to configure own master node or use one provided by Chef. It is worth noticing, that second approach is chargeable. In most setups master node is the one which communicates with client nodes and administrator's workstation. In order to make communication between workstation and master node possible, workstation's public key should be added to the master node.

3. Bootstrapping or applying cookbooks on the client nodes from the workstation or the master node.

    - In order to bootstrap the node, it is required to have an SSH access to it from the workstation. Afterwards it is possible to apply the cookbook and establish connection between client and master nodes with the following command[14]:

```
$ knife bootstrap ADDRESS --ssh-user USER --sudo
    --identity-file PRIV_KEY --node-name node1 --
   run-list 'recipe[apache2]'
```

Listing 8: Establish connection between Chef client and master nodes

During the bootstrap process `chef-client` is installed on the target node, then it is associated with the master node and cookbooks specified after `--run-list` flag will be applied[14]. In order to verify whether the node was successfully bootstrapped, it is possible to use master node management console or execute one of the following commands from the workstation:

```
$ knife node list
$ knife node show
```

Listing 9: Chef managed node's bootstrap verification commands

- Next time the cookbook will be pushed to the Master node, it is not needed to bootstrap the client node again. Updated cookbook can be applied by running `chef-client` command on the client node. It can be done from the workstation with the following command [15]:

```
$ knife ssh ADDRESS 'sudo chef-client' --manual-
   list --ssh-user USER --identity-file PRIV_KEY
```

Listing 10: Apply cookbook on client node

Similarly to Puppet, Chef has its own structure of scripts, which describe how nodes should be configured. Chef project which is called **cookbook** can be created with a following command:

```
$ chef generate cookbook nginx
```

Listing 11: Generate Chef project skeleton

As a result, cookbook called `nginx` will be created with the following structure:

```
nginx
|── Berksfile
|── chefignore
|── metadata.rb
|── README.md
|── recipes
|   +── default.rb
|── spec
|   |── spec_helper.rb
|   +── unit
|       +── recipes
|           +── default_spec.rb
+── test
    +── integration
        |── default
        |   +── serverspec
        |       +── default_spec.rb
        +── helpers
            +── serverspec
                +── spec_helper.rb
```

Listing 12: Chef project skeleton

The important directory in cookbook is called **recipes**. Here are located files, which contain Chef **resources**. Those are snippets of Ruby code, which describe

how the system should be configured. For instance, if it is required to define recipe for Nginx server configuration, file called `nginx.rb` should be created in `nginx/recipes` directory with the following content:

```
# install nginx package
package 'nginx'

# ensure that it is enabled and started
service 'nginx' do
  supports :status => true
  action [:enable, :start]
end
```

Listing 13: Chef Nginx service recipe

After that created cookbook can be uploaded to the master node and applied on the desired client nodes (Refer to Listing 10).

On one hand, Chef is a good choice for DevOps teams, due to its code-driven approach. Especially if the team has experience in Ruby programming [20]. Chef has a strong version control capabilities [20]. As a result, it is a good choice for enterprise companies, whixh quite often make changes in their managed infrastructures. In addition, Chef community has a rich collection of cookbooks, which can be downloaded from their official website. Furthermore, Chef follows "Write once, use anywhere" approach. Namely, previously implemented cookbooks can be reused in different infrastructures. It also has a well structured documentation with tons of examples.

On the other hand, there are some aspects which can discourage from using this tool. One of them is requirement of Ruby knowledge for writing cookbooks [20]. In addition, using Chef in large teams can lead to the disorganization in cookbooks, when some of the team members will try to modify those cookbooks simultaneously. It is analogous to resolving conflicts in Git. In addition, this tool is not recommended for the beginners, due to its code-driven approach.

2.5 Conclusion

In this chapter such configuration management tools as SaltStack, Puppet and Chef were reviewed. Although, they have quite similar features, the way these features are implemented is different. According to the list of features **F-1** they have 6 - 7 out of 9 in their functionality. Due to the fact that Ansible tool has 8 out of 9 features in its functionality, it will be reviewed in a separate chapter.

3 Ansible

In this chapter Ansible configuration management tool will be introduced in details. Firstly, the introduction will be made. Secondly, most important components will be examined in details. Lastly, advantages and disadvantages of Ansible usage will be reviewed.
The reason why Ansible is analyzed in more details than other configuration management tools is that the author of the research paper has more practical experience in using Ansible, compared to other tools. Furthermore, Ansible has the most optimal combination of features. This is also the reason why practical part of the research was built using Ansible. More details are available in the fourth chapter.

3.1 Introduction

Ansible features:

- ☒ deploy, update, remove configuration files and services (Refer to **F-1**)
- ☐ version control capabilities (Refer to **F-2**)
- ☒ execute single command remotely (Refer to **F-3**)
- ☒ system scalability (Refer to **F-4**)
- ☒ OS independency (Refer to **F-5**)
- ☒ agent-less (Refer to **F-6**)
- ☒ code readability (Refer to **F-7**)
- ☒ support (Refer to **F-8**)
- ☒ free and open-source (Refer to **F-9**)

Ansible is an open-source IT automation tool. It allows to deploy applications, manage configurations and orchestrate complicated sequences of events not only for servers, but also for other components such as routers. The original author of the Ansible project is Michael DeHaan and it was initially released in 2012 [6]. However, in December 2015 Red Hat Inc. acquired Ansible Inc. In brief, Red Hat substantiated this decision with following sentence:

> We see in Ansible a perfect alignment with the core principles that shape Red Hat's management, both at the product level and at the portfolio level. [24]

It is quite easy to understand the way Ansible works (Refer to Figure 4):

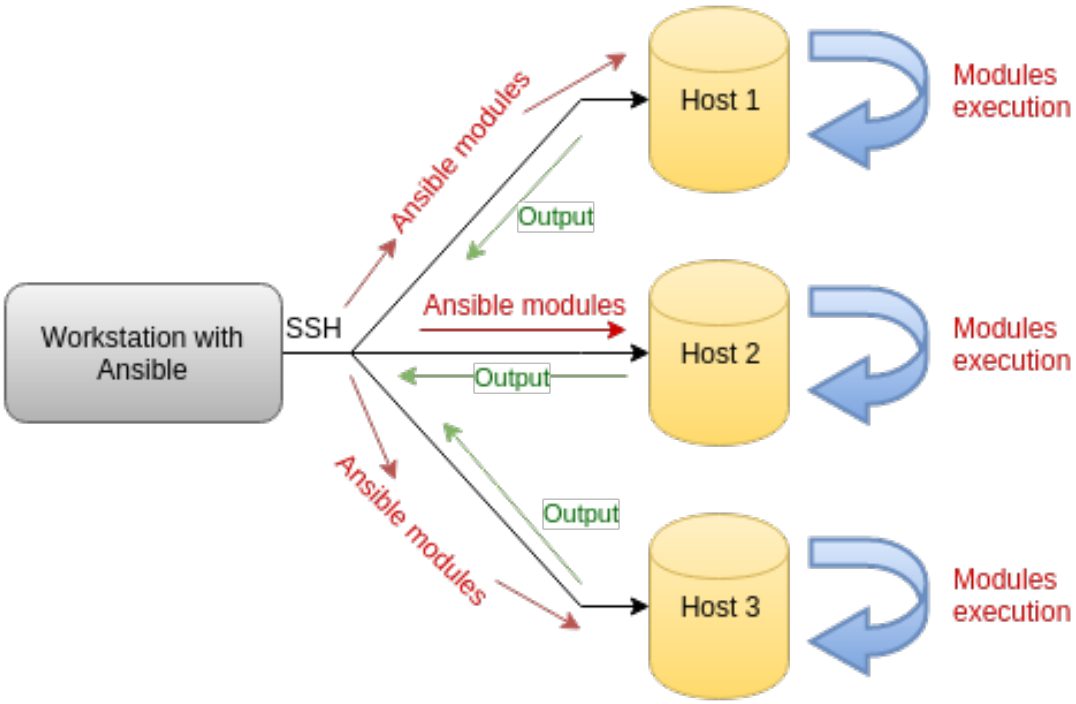

Figure 4: Communication between Ansible workstation and hosts

It is not needed to install any agent on the managed nodes i.e. it uses agent-less architecture. Although, OpenSSH server and Python should be presented on the machines, which are by default on Unix-type servers. Communication between workstation and managed nodes is over SSH [9]:

1. Ansible connects over SSH to the node. The user can be authenticated with password or key.

2. Ansible pushes programs called "Ansible modules" to the node. These programs define how the system should be configured.

3. Ansible executes those programs using Python and transmits their output to workstation's terminal.

4. Ansible removes those programs from managed node.

As a result, system administrators who used bash scripts to configure servers could easily replace them with Ansible scripts, because Ansible uses the same

principle for applying those scripts on the servers. It is also possible to establish communication between workstation and managed nodes using Kerberos.

3.2 Inventory

Inventory is an important file in Ansible project, which is responsible for holding information about managed nodes. It is an `INI`-type file, where hostnames, IP addresses, usernames, passwords, connection ports, etc. of managed nodes should be specified [3]. Moreover, inventory file allows to group hosts. For example, it is possible to group hosts by types of services that should be installed on them:

```
[mysql_servers]
host1
host2

[apache_servers]
host3
host4
```

Listing 14: Ansible hosts grouped by services

In addition, large-scale setups often have multiple environments. For instance, during cloud development there are usually three types of environments constructed. This approach is very similar to the one, which is used in software project development, where three main stages are developing, testing and production:

1. TST - testing environment. It is used for testing purposes. In this environment developers test new features of applications and system administrators monitor and configure services that soon will be in production environment.

2. STG - staging environment. This environment should be identical to the production environment in terms of configured services and applications. It is used after testing new features on TST environment to be sure that these features will not break anything on PRD.

3. PRD - production environment. This environment is used by end users and should be as stable as possible. Here testing must not be allowed.

Ansible inventory file can be used to group hosts according to the environment. It is also possible to specify IP addresses of the hosts [3]:

```
[tst_servers]
host1 ip_addr=10.10.7.7
host2 ip_addr=10.10.7.8

[stg_servers]
host3 ip_addr=10.10.8.7
host4 ip_addr=10.10.8.8

[prd_servers]
host5 ip_addr=10.10.9.7
host6 ip_addr=10.10.9.8
```

Listing 15: Ansible hosts grouped by environment

As a result, it will be much easier to provision large-scale systems with large number of nodes in different environments. After, these groups can be referred in Ansible playbooks and ad-hocs.

3.3 Ad-hocs and Playbooks

The are two ways of running commands on remote systems using Ansible. One of them is ad-hoc method, which is interactive [1]. This method allows to execute small Ansible commands remotely and see the result in a moment. For instance, if it is needed to ping all the managed hosts, Ansible ping module can be used for that:

```
$ ansible -m ping -u deployer tst_servers
host1 | success >> {
      "changed": false,
      "ping": "pong"
}

host2 | success >> {
      "changed": false,
      "ping": "pong"
}
```

Listing 16: Ansible ping all the hosts from command line

During execution of this command Ansible will use `ping` module to verify the ability to login over SSH as `deployer` user to every host specified in inventory file under `tst_servers` group. As a result, this method can be used for testing or for applying quick fixes on multiple servers at once. Currently, there are about 520

modules available for different purposes and their number grows with every release.

Another approach to define the way systems will be configured uses Ansible Playbooks [4]. Those are the scripts written in `YAML` [21]:

```
---
- name: Install and configure Nginx server
  hosts: tst_servers
  remote_user: dmitri
  sudo: yes

  tasks:

  - name: (os=Ubuntu) Install Nginx
    apt: name=nginx state=present update_cache=yes
    sudo: yes
    when: ansible_os_family == 'Debian' and
       ansible_distribution_release == 'trusty'
```

Listing 17: Example of Ansible playbook

During execution of the playbook (refer to Listing 17) Ansible will establish SSH connection with every host in `tst_servers` group as a `dmitri` user. Next it will gather facts about the system, which can later be used in playbooks, and runs tasks as sudo user [2]. In this example, it will try to install Nginx server on machines with Ubuntu 14.04 operating system. It will also check whether Nginx was already installed on these machines. If it is not, then it will update repositories and install Nginx. By changing `state` to `latest` Ansible will be forced to check whether the latest version of Nginx is installed on every playbook run.
In order to restart Nginx after its configuration is updated, `handlers` can be used [2]. They are responsible for checking states of services and move them from one state to another. Handler can be connected to the task with `notify` field. They are only triggered in the case when the task, which is connected to the handler has applied some changes. Handlers are applied only in the end of the playbook run. For example, in Listing 18 every time Nginx package will be installed or updated, `restart_nginx` trigger will be executed in the end of the playbook run. Handlers are executed in the end of the playbook run, because there is no necessity to move service from one state to another multiple times during execution of the playbook.

```
---
- name: Install and configure Nginx server
  hosts: tst_servers
  remote_user: dmitri
  sudo: yes

  tasks:

  - name: (os=Ubuntu) Install Nginx
    apt: name=nginx state=latest update_cache=yes
    sudo: yes
    when: ansible_os_family == 'Debian' and
       ansible_distribution_release == 'trusty'
    notify:
    - restart_nginx

  handlers:
  - name: restart_nginx
    service: name=nginx state=restarted
```

Listing 18: Example of Ansible playbook with handlers

Furthermore, it is also possible to spread all the tasks and handlers in different files and just import them by including into the playbook. In addition, it is possible to run Ansible playbook in testing mode and it will be clear what configuration files will be changed, services installed, etc. when the playbook will be executed in real mode.

In contrast, equal to the Ansible playbook (refer to Listing 18) Nginx installation script written in Bash would look like this:

```
#!/bin/bash

if [[ "$OSTYPE" == "linux-gnu" ]]; then
  command -v nginx >/dev/null 2>&1 && { echo "[+] Nginx
     already installed"; exit 0; }
  echo "Installing Nginx..."
  sudo apt-get update && sudo apt-get -y install nginx
fi
```

Listing 19: Example of installing Nginx using Bash

It can be seen that code written in YAML is more human readable than code written in Bash. In addition, due to the strict structure of YAML files, it is hard to distinguish which parts of the code were written by different system administrators. However, Bash scripts do not have strictly defined code writing conventions and code styles can vary according to the system administrator. It is also a challenge to get familiar with configuration scripts written in Bash for the new team members.

One of the greatest advantages of Ansible is that it is possible to group its playbooks by roles and use these roles in different projects [5]. For example, it is possible to define role for Nginx HTTP server installation and use it in Django application deployment playbook or anywhere else. Furthermore, it is possible to download roles from the Ansible community website [8] for free and use them in other playbooks.

3.4 Template

Template is an Ansible module, which allows to define dynamic configuration files [7]. These files are called templates and they are written in Jinja2 templating language [25].

```
1  # {{ ansible_managed }}
2
3  server {
4      listen 80;
5      server_name {{ server_name }};
6      location /static/ {
7          alias {{ static_path }};
8      }
9      location /media/ {
10         alias {{ media_path }};
11     }
12     location / {
13         include         uwsgi_params;
14         uwsgi_pass      unix:/home/{{ username }}/{{
               project_name }}/{{ project_name }}.sock;
15     }
16 }
```

Listing 20: Nginx site template example

For example, Nginx site template for hosting Django applications can be defined 21. As a result, this template can be used for multiple HTTP servers to run different Django applications. Variables, which are in curly brackets are injected into the template during playbook run [25]. They can be defined in explicit file or directly in the playbook. The first line of the template `# {{ ansible_managed }}` is used by Ansible to distinguish between regular files and templates. When the template is defined, it can be put on managed servers with the following task:

```
- name: Add site configs
  template: src=files{{ item }}/default dest={{ item
     }}/{{ project_name }} mode=0644
  sudo: yes
  with_items:
  - /etc/nginx/sites-available
  notify:
    - nginx_check_config
    - nginx_restart
```

Listing 21: Transfering Nginx site config to managed servers

This is very frequently used task, which will transfer template from the workstation located in `files/etc/nginx/sites-available/default` to the managed nodes to `/etc/nginx/sites-available/my_app`, where `my_app` is the value of `project_name` variable. It will also give 644 permissions to the file. If the template is already on the managed server and it is equal to the one defined on the workstation, then task will not be executed. However, if these files differ, the one on the managed server will be overwritten by the one on the workstation. In addition, `nginx_check_config` and `nginx_restart` handlers will be executed.

3.5 Conclusion

Ansible is a great configuration management tool for infrastructures, which consist of immutable servers, because it has a separate `inventory` file described above, where servers can be easily added, removed or replaced. In addition, Ansible is a multi-platform tool, because it is possible to use different modules to define playbooks for multiple operating systems. The operating system of the server can be determined by using facts, which are gathered by Ansible. Furthermore, playbooks are very readable and even users, who do not have much experience in Ansible can understand the purpose of the particular playbook. As a result, it is a good tool to get familiar with configuration management principles. Due to the fact that playbooks written before can be easily used in future, it is not needed to write playbooks for similar configurations. Moreover, it is possible to run playbooks in

testing mode, which is a great advantage when testing new playbooks or checking what configuration changes were applied on the servers manually.

However, Ansible does not have a version control system (VCS), which makes it hard to track changes made by different team members in the playbooks. In addition, playbooks in workstations can differ, which will result into conflict when applying those playbooks on the same managed servers. To overcome this disadvantage, in large companies it is a good practice to upload Ansible projects to the version control repository and add every configuration change through the pull requests. As a result, secret variables (usernames, passwords, keys, etc.) should be defined in separate file and not uploaded to the VCS. To use this approach, system administrators should know how to use version control systems. As a result, using Ansible with VCS makes the difference between programming and system administration much smaller, because the actions that should be performed to develop a software project and Ansible project are getting more similar.

4 Configuration management in practice

In this section will be described the implementation of Ansible playbooks for Zabbix and WordPress services.

4.1 Zabbix playbook

Due to the fact that large-scale systems and clouds have a lot of servers and each server has several services, monitoring becomes a crucial part. In most setups it is required to gather data about server's performance, services and send alerts to system administrators to warn about critical situations. Zabbix is an open-source monitoring platform mostly used by enterprise companies [30]. It allows to monitor different components inside a network. For example, it is possible to use Zabbix for monitoring Web applications, databases, network equipment, performance and availability of servers and other components of the infrastructure [31].

Zabbix monitoring tool consists of following components:

- **Server**, which holds and parses collected data regarding the infrastructure.
- **Agents**, which should be installed on the managed servers and configured to monitor several elements.
- **Web interface**, which allows to check graphs, read monitored information, etc.

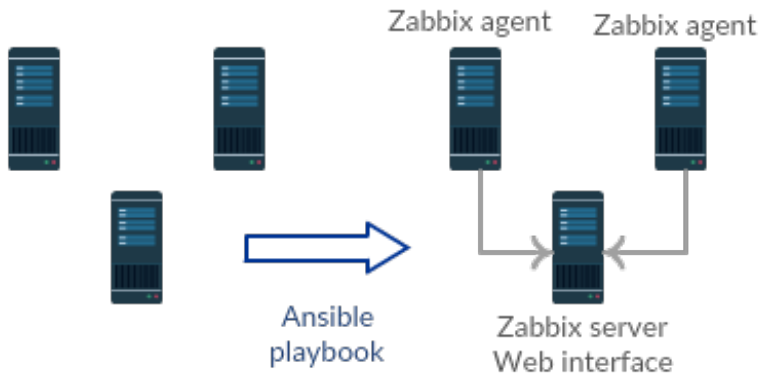

Figure 5: Configuring Zabbix with Ansible playbook

As a result, Zabbix is a good tool to show how quickly and easily it is possible to scale monitored servers. Namely, with running several Ansible playbooks it is

possible to deploy Zabbix server and multiple agents on the nodes. In addition, when new node is introduced Ansible playbook can be used to make it monitorable.

4.1.1 Playbook implementation

Ansible implementation for Zabbix monitoring tool was divided into two playbooks. One of them is responsible for deploying and configuring Zabbix server and Web interface, another for deploying and configuring Zabbix agent [40]. This allows to configure monitoring on managed nodes separately from monitoring server.

Zabbix server deployment was divided into four steps:

1. Zabbix repository configuration [40]. In this step it should be checked whether repository is already configured. If it is not, then Zabbix release package should be downloaded and installed, then repositories should be updated and to clean up the system release package can be removed. When the repository is configured, it should be possible to download Zabbix packages.

   In order to do this step manually following shell script should be executed [40]:

```
1 #!/bin/bash
2
3 if [ ! -f /etc/apt/sources.list.d/zabbix.list ];
     then
4     wget http://repo.zabbix.com/zabbix/3.0/ubuntu/
        pool/main/z/zabbix-release/zabbix-release_3
        .0-1+trusty_all.deb
5     dpkg -i zabbix-release_3.0-1+trusty_all.deb
6     apt-get -y update
7     rm zabbix-release_3.0-1+trusty_all.deb
8 fi
```

   Listing 22: Zabbix repository manual installation

   This script will be barely readable for users who are not familiar with Bash scripting. However, Ansible tasks that produce the same result are more human-readable (refer to Appendix A.1).

2. MySQL database configuration. First it should be checked whether MySQL database is installed. After MySQL service should be started and enabled. In order to do this step manually following shell script could be executed [40]:

```
#!/bin/bash

if [[ "$OSTYPE" == "linux-gnu" ]]; then
  command -v mysql >/dev/null 2>&1 && { echo "[+]
     MySQL database already installed"; exit 0; }
  echo "Installing MySQL database..."
  apt-get update && apt-get -y install mysql-server
  service mysql start
fi
```

Listing 23: MySQL database installation

Again users, who are not familiar with Bash scripting will not understand this script completely. However, Ansible tasks, which produce the same result are more human-readable (refer to Appendix A.2).

3. Zabbix server installation [40]. In this step Zabbix server packages should be downloaded and installed from Zabbix repository. Next its database should be initialized and database user created. Lastly, configuration file should be updated with database parameters and Zabbix server should be restarted. After completing this part of installation, Zabbix will be able to collect and parse monitored data.
   Unfortunately, automation of this step with Bash script is hardly achievable due to the fact that there must be configuration file modified. In addition, it is also a challenge to configure MySQL server parameters such as database name, username, password. In order to do this step manually following commands should be executed from command prompt[40] [39]:

```
# apt-get install zabbix-server-mysql
# mysql -uroot
mysql> create database zabbix character set utf8;
mysql> create user '<username>'@'<hostname>'
   identified by '<password>';
mysql> grant all privileges on zabbix.* to <username
   >@<hostname> identified by '<password>';
mysql> quit;
# cd /usr/share/doc/zabbix-server-mysql
# zcat create.sql.gz | mysql -uroot zabbix
# nano /etc/zabbix/zabbix_server.conf
# service zabbix-server restart
```

Listing 24: Zabbix server installation

However, Ansible tasks allow user to automate this process (refer to Appendix A.5). Such variables as database user username, password and database name, hostname will be taken from separate variables file located in `group_vars/zabbix_servers` on the workstation. After that Zabbix server configuration file will be transfered from `files/etc/zabbix/zabbix_server.conf` (refer to Appendix A.4) to `/etc/zabbix/zabbix_server.conf` on the managed host and Zabbix server will be restarted. Note, that variables specified in `group_vars/zabbix_servers` will be injected into configuration file.

4. Zabbix Web interface installation [40]. Similarly to the previous step Zabbix Web interface package should be downloaded and installed from Zabbix repository, if it is not presented on managed server. Next Apache configuration file for Zabbix Web interface should be modified and Apache server should be restarted. Web interface final configuration is done in interface itself.
   Due to the fact that configuration file for Zabbix Web interface should be modified, it will be difficult to automate this step by using Bash scripts. However, following commands can be executed from the command prompt:

```
# apt-get -y install zabbix-frontend-php
# nano /etc/zabbix/apache.conf
# service apache2 restart
```

Listing 25: Zabbix Web interface installation

   Same results can be achieved by using Ansible tasks (refer to Appendix A.6). In addition, it is possible to automate `/etc/zabbix/apache.conf` file configuration by transferring it from `files/etc/zabbix/apache.conf` on the workstation. Furthermore, it is also possible to skip the final step of installation in interface by transferring additional file from `files/etc/zabbix/web/zabbix.conf.php` on the workstation to `etc/zabbix/web/zabbix.conf.php` on the managed node.

Zabbix agent deployment playbook was separated into two steps:

1. Zabbix repository configuration [40]. This step is identical to the one described for the Zabbix server installation.

2. Zabbix agent installation [40]. Here Zabbix Agent should be installed from Zabbix repository, if it is not installed yet. Next Zabbix agent configuration file should be updated. Lastly, Zabbix agent service should be restarted.

   Due to the fact that configuration file modification is needed, it is difficult to automate this step using Bash scripting.

```
# apt-get -y install zabbix-agent
# nano /etc/zabbix/zabbix_agentd.conf
# service zabbix-agent restart
```

Listing 26: Zabbix agent installation

However, Ansible tasks will be able to do that (refer to Appendix A.7). Zabbix agent configuration template will be transferred from `files/etc/zabbix/zabbix_agentd.conf` on the workstation to `/etc/zabbix/zabbix_agentd.conf` on the managed node. Template variables will be taken from `group_vars/zabbix_agents` file. After Zabbix agent will be started and enabled. Due to the fact that host, where Zabbix server is installed can also be an agent, following if-clause was added to the agent's configuration template:

```
1 {% if inventory_hostname in groups['zabbix_servers']
      %}
2 ServerActive=127.0.0.1
3 {% else %}
4 ServerActive={{ zabbix_server }}
5 {% endif %}
```

Listing 27: Zabbix agent setup for Zabbix server

As a result, during variables injection Ansible will check whether managed node is Zabbix server or agent. If it is a server, then `ServerActive` variable value will be server's localhost IP. If it is an agent, then `ServerActive` value will be `zabbix_server` variable value, which should be specified in `group_vars/zabbix_agents` file.

4.2 WordPress playbook

WordPress is a powerful tool, which can be used to quickly create a website, a blog or a small Web application [38]. It is a perfect solution for regular users, who do not have experience in programming and creating Web applications using different frameworks. In addition, it is a free and open-source software [38]. However, it could be a challenge for regular users to deploy WordPress to their own server, because it requires command line knowledge and system administration experience. As a result, it could be nice to automate WordPress installation process using Ansible playbooks. In consequence, it is possible to develop GUI, which will execute Ansible playbook on background to deploy and configure WordPress on user's server.

4.2.1 Playbook implementation

WordPress deployment and configuration was divided into three steps:

1. MySQL database configuration. This step is identical to the one described in Zabbix server playbook description. It is worth noticeable, that properly written Ansible tasks can be reused in other playbooks. MySQL database is used by WordPress to store website data.

2. Apache server configuration. Due to the fact that WordPress does not have OS specific packages such as deb or rpm, which automate Apache installation, it should be installed manually. First Apache package should be installed, if it is not installed yet, then it should be started and enabled. Apache serves WordPress Web application.

   Minimal Apache installation is simple and it is possible to do with Bash script:

```
1 #!/bin/bash
2
3 if [[ "$OSTYPE" == "linux-gnu" ]]; then
4   command -v apache2 >/dev/null 2>&1 && { echo "[+]
      Apache server already installed"; exit 0; }
5   echo "Installing Apache server..."
6   apt-get update && apt-get -y install apache2
7   service apache2 restart
8 fi
```

   Listing 28: Apache server installation

   However, it will be not as readable as Ansible tasks (refer to Appendix A.8) and its code will vary according to the system administrator.

3. WordPress installation [19]. Firstly, PHP dependencies should be installed to be able to execute WordPress code. Due to the fact that running WordPress as a root user can cause security risks, there should be separate user created, who will own all WordPress files. Next WordPress installer should be downloaded and extracted to the `/var/www/html/` directory with correct owner attribute. Lastly, MySQL user and database should be created, WordPress configuration should be modified and Apache server should be restarted.

   In this installation several variables must be set by user such as usernames and passwords. In addition, WordPress configuration file should be modified.

As a result, it will be a challenge to write Bash script, which will do these things. However, it is still possible to execute all the commands manually as a root:

```
apt-get -y install php5-gd libssh2-php php5-mysql
   php5 libapache2-mod-php5 php5-mcrypt
adduser <username1>
wget http://wordpress.org/latest.tar.gz
tar xvzf latest.tar.gz
rsync -avz wordpress/* /var/www/html/ && rm -rf
   wordpress
chown -R <username1>:<username1> /var/www/html/*
mysql -uroot
mysql> create database <database_name> character set
    utf8;
mysql> create user '<username2>'@'<hostname>'
   identified by '<password2>';
mysql> grant all privileges on <database_name>.* to
   <username2>@<hostname> identified by '<password2
   >';
mysql> quit;
cp /var/www/html/wp-config-sample.php /var/www/html/
   wp-config.php
nano /var/www/html/wp-config.php
```

Listing 29: WordPress server manual installation

All these commands can be transferred to the Ansible tasks to automate the process of WordPress server installation (refer to Appendix A.9). User parameters will be taken from separate variables file `group_vars/wordpress_servers`, where all WordPress server variables are stored. Next, latest WordPress configuration file will be moved from `files/var/www/html/wp-config.php` on workstation to the `/var/www/html/wp-config.php` on the managed server. Apache server will be restarted using Ansible handlers.

4.3 Testing

In order to test created playbooks it was decided to set up a small infrastructure.

The University of Tartu distributed systems research group has a cluster with six nodes inside a private network, which is separated from the Tartu University Eduroam network with the `MikroTik` router. One of this nodes was decided to use for testing the playbooks. The idea was to boot three virtual machines on

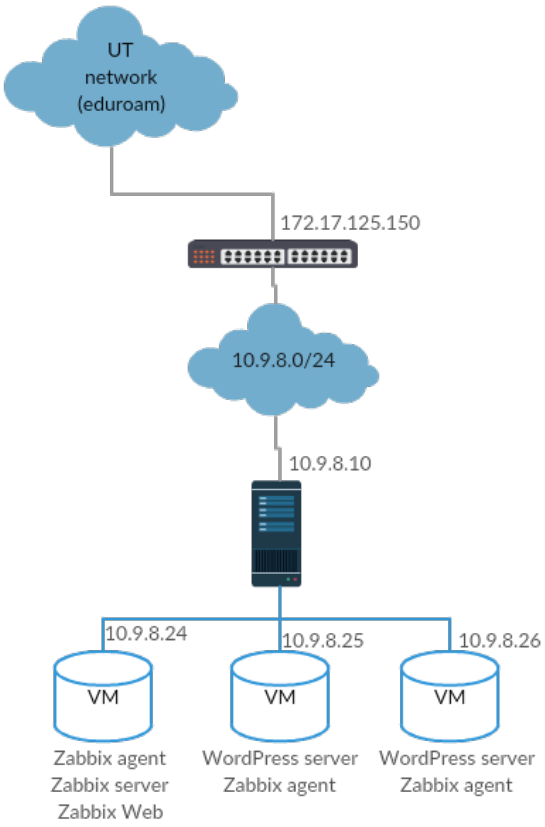

Figure 6: Infrastructure for testing.

this node using KVM hypervisor and use node's bridged network [18] interface as virtual machines' interfaces. As a result there will be a cluster with four nodes in the same network (10.9.8.0/24). In order to make the workstation to be in the same network with virtual machines an additional `xenbr0` virtual network interface was created. This interface was configured to use `eth0` interface of the workstation.

```
auto eth0
iface eth0 inet manual
  if ifconfig eth0 up
  down ifconfig eth0 down

auto xenbr0
iface xenbr0 inet dhcp
  bridge_ports eth0
  bridge_stp off
  bridge_fd 0
```

Listing 30: Bridged network setup

As a result, all the virtual machines, which will be configured to use `xenbr0` interface will be in the same bridged network as the workstation. In educational purposes it was decided to use 8GB USB memory stick as a node storage with installed Debian 8 on it. However, during node configuration revealed that each virtual machine with Ubuntu 14.04 operating system reserves around 4 GB of storage. As a result, there was not enough storage room on the node to boot 3 virtual machines.

It was decided to do following steps:

1. Transfer system to the memory stick with 16GB storage room.
2. Move virtual machines storage files to the NFS server.

When node storage and KVM was successfully configured, three virtual machines were set up:

```
# virsh list --all
 Id    Name                           State
----------------------------------------------------
 26    zabbix-server                  running
 28    wordpress1                     running
 29    wordpress2                     running
```

Listing 31: Testing machines

The process of testing the playbooks consisted of following stages:

1. Adding public keys to the virtual machines for SSH connections.
2. Modifying `/etc/hosts` file of the workstation by adding several entries for virtual machines:

```
10.9.8.24 zabbix-server
10.9.8.25 wordpress1
10.9.8.26 wordpress2
```

Listing 32: Adding entries to /etc/hosts file

3. Checking `hosts` file:

```
[zabbix_servers]
zabbix-server

[zabbix_agents]
zabbix-server
wordpress1
wordpress2

[wordpress_servers]
wordpress1
wordpress2
```

Listing 33: Checking hosts file

4. Changing variables of the playbooks in `group_vars` directory.

5. Running the playbooks:

```
# ansible-playbook -i hosts zabbix_server.yml
# ansible-playbook -i hosts zabbix_agent.yml
# ansible-playbook -i hosts wordpress.yml
```

Listing 34: Running the playbooks

An additional `-i` flag will tell Ansible which inventory file to use. It is useful to run playbooks with `--diff` flag to check what was changed in the files. In addition, it is possible to make a test run by adding `--check` flag. As result, nothing will be changed on the server, but system administrator will be able to check what changes will be applied.

6. Checking in the browser whether all the services are up and running. The result should be as show in Figure 6.

In conclusion, it is a good practice to use virtual machines for testing configuration management scripts, because it is possible to create snapshots, which can be used to revert system to the previous state. It is a beneficial approach, because during scripts development there is a necessity to run them against testing server a lot of times.

4.4 Results

The goal of the research was to study the applicability of Ansible for fast deployments and management of the infrastructure. Zabbix and WordPress were selected for the case studies. During the experimentations Ansible playbooks for these services were developed. The results showed clear advantages of using Ansible compared to manual setups or Bash scripts. In addition, developed playbooks and their components, such as Apache HTTP server and MySQL database, can be used in future projects. For example, it is possible to easily make all the infrastructures monitorable or quickly deploy WordPress to make a Web application. Furthermore, those playbooks can be uploaded to the GitHub or Ansible Galaxy. In consequence, other researchers will be able to use those scripts in their projects. Furthermore, established processes of implementing playbooks can be used in Distributed Systems Group to automate configuration of multiple clusters.
However, there are some restrictions to the operating system where those playbooks could be run. Namely, playbooks are written for Ubuntu 14.04 operating system and tested only on it. In order to make them executable on other Unix type operating systems, several tasks can be added, which will be executed differently depending on the server's operating system.

5 Conclusion

In this research it was demonstrated how different configuration management tools can simplify the work of system administrator. Furthermore, it was overviewed what solutions to the described problems these tools provide when dealing with large-scale systems. In addition, advantages and disadvantages of popular configuration management tools were described such as SaltStack, Puppet, Chef and Ansible. There was also overviewed the process of creating Ansible playbooks for such services as Zabbix and WordPress. Lastly, there was an example and practical recommendations provided on how the configuration management scripts can be tested.
In future, Distributed Systems Group could use this knowledge to automate deployment of several clusters. Moreover, there could be multiple Ansible playbooks defined to test students homework in "System Administration" course lab by logging into their virtual machines and comparing their configuration files with correct ones. Lastly, this research can be included to the "System Administration" course additional materials, therefore students would be able to get familiar with those tools.

References

[1] Ansible. Ansible Adhoc. http://docs.ansible.com/ansible/intro_adhoc.html. Last Accessed: 09.05.2016.

[2] Ansible. Ansible Introduction Video. https://fast.wistia.net/embed/iframe/qrqfj371b6?popover=true. Last Accessed: 09.05.2016.

[3] Ansible. Ansible Inventory. http://docs.ansible.com/ansible/intro_inventory.html. Last Accessed: 09.05.2016.

[4] Ansible. Ansible Playbook. http://docs.ansible.com/ansible/playbooks_intro.html. Last Accessed: 09.05.2016.

[5] Ansible. Ansible role. http://docs.ansible.com/ansible/playbooks_roles.html. Last Accessed: 09.05.2016.

[6] Ansible. Ansible software. http://cdn2.hubspot.net/hub/330046/file-479069823-pdf/pdf_content/Achieving_Rolling_Updates_and_Continuous_Deployment_with_Zero_Downtime.pdf. Last Accessed: 09.05.2016.

[7] Ansible. Ansible template. http://docs.ansible.com/ansible/template_module.html. Last Accessed: 09.05.2016.

[8] Ansible. Galaxy. https://galaxy.ansible.com/. Last Accessed: 09.05.2016.

[9] Ansible. Work Principle. https://www.ansible.com/how-ansible-works. Last Accessed: 09.05.2016.

[10] Vangie Beal. Devops definition. http://www.webopedia.com/TERM/D/devops_development_operations.html. Last Accessed: 09.05.2016.

[11] André B. Bondi. *(Characteristics of scalability and their impact on performance. Proceedings of the second international workshop on Software and performance*. 2000.

[12] Carlos Casanova. Configuration Management, Why bother? http://allthingsitsm.com/configuration-management-why-bother/. Last Accessed: 10.05.2016.

[13] Chef. Chef communication figure. https://learn.chef.io/manage-a-node/ubuntu/. Last Accessed: 09.05.2016.

[14] Chef. Chef node bootstrap. https://learn.chef.io/manage-a-node/ubuntu/bootstrap-your-node/. Last Accessed: 09.05.2016.

[15] Chef. Chef node update. https://learn.chef.io/manage-a-node/ubuntu/update-your-nodes-configuration/. Last Accessed: 09.05.2016.

[16] Chef. Chef server setup. https://learn.chef.io/manage-a-node/ubuntu/set-up-your-chef-server/. Last Accessed: 09.05.2016.

[17] Microsoft Corporation. Kerberos protocol. https://tools.ietf.org/html/rfc4556. Last Accessed: 09.05.2016.

[18] Decker, Langille, Rijsinghani, McCloghrie. Bridged Network. https://tools.ietf.org/html/rfc1286. Last Accessed: 11.05.2016.

[19] DigitalOcean. WordPress installation. https://www.digitalocean.com/community/tutorials/how-to-install-wordpress-on-ubuntu-14-04. Last Accessed: 09.05.2016.

[20] Josh Dreyfuss. CM tools comparison. http://blog.takipi.com/deployment-management-tools-chef-vs-puppet-vs-ansible-vs-saltstack-vs-fabric/. Last Accessed: 09.05.2016.

[21] Clark C. Evans. YAML. http://yaml.org/. Last Accessed: 11.05.2016.

[22] EventHelix. Availability term. http://www.eventhelix.com/RealtimeMantra/FaultHandling/reliability_availability_basics.htm#.VzHJnkF948o. Last Accessed: 10.05.2016.

[23] Alessandro Franceschi. Puppet. http://www.example42.com/tutorials/PuppetTutorial. Last Accessed: 09.05.2016.

[24] Red Hat. Red Hat acquired Ansible. https://www.redhat.com/en/about/blog/why-red-hat-acquired-ansible. Last Accessed: 09.05.2016.

[25] Jinja. Jinja template. http://jinja.pocoo.org/docs/dev/templates/. Last Accessed: 09.05.2016.

[26] johnccfm. Windows minions response. https://github.com/saltstack/salt/issues/27866. Last Accessed: 09.05.2016.

[27] Linux KVM. KVM. http://www.linux-kvm.org/page/Main_Page. Last Accessed: 09.05.2016.

[28] Puppet Labs. Puppet classes. https://docs.puppetlabs.com/puppet/latest/reference/lang_classes.html. Last Accessed: 09.05.2016.

[29] Puppet Labs. Puppet resources. https://docs.puppetlabs.com/puppet/latest/reference/lang_resources.html. Last Accessed: 09.05.2016.

[30] Zabbix LLC. Zabbix. http://www.zabbix.com/. Last Accessed: 09.05.2016.

[31] Zabbix LLC. Zabbix features. http://www.zabbix.com/features.php. Last Accessed: 09.05.2016.

[32] Lauren Malhoit. Puppet communication figure. http://tr1.cbsistatic.com/hub/i/2015/05/07/13ecf8b9-f499-11e4-940f-14feb5cc3d2a/puppet-agentmaster.png. Last Accessed: 09.05.2016.

[33] Department of Defense USA. Configuration Management. http://www.pica.army.mil/PROD_TECHDATA/Files/MIL-HDBK-61A.pdf. Last Accessed: 09.05.2016.

[34] Puppet. Puppet. https://docs.puppet.com/guides/faq.html. Last Accessed: 09.05.2016.

[35] Puppet. Puppet Forge. https://forge.puppet.com/. Last Accessed: 11.05.2016.

[36] SaltStack. Documentation. https://docs.saltstack.com/en/getstarted/overview.html. Last Accessed: 09.05.2016.

[37] Paul Venezia. Salt software. http://www.infoworld.com/article/2609482/data-center/data-center-review-puppet-vs-chef-vs-ansible-vs-salt.html. Last Accessed: 09.05.2016.

[38] WordPress. WordPress. https://wordpress.org/. Last Accessed: 09.05.2016.

[39] Zabbix. MySQL server initial setup. https://www.zabbix.com/documentation/3.0/manual/installation/install_from_packages. Last Accessed: 11.05.2016.

[40] Zabbix. Zabbix installation. https://www.zabbix.com/documentation/3.0/manual/installation/install#from_distribution_packages. Last Accessed: 09.05.2016.

Appendices

A Zabbix and WordPress playbooks

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible
Last Accessed: 12.05.2016

A.1 Ansible Zabbix repository tasks

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/tasks/zabbix_repo.yml
Last Accessed: 12.05.2016

A.2 Ansible MySQL server tasks

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/tasks/mysql_server.yml
Last Accessed: 12.05.2016

A.3 Ansible MySQL variables file

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/group_vars/zabbix_servers
Last Accessed: 12.05.2016

A.4 Ansible Zabbix server configuration file

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/files/etc/zabbix/zabbix_server.conf
Last Accessed: 12.05.2016

A.5 Ansible Zabbix server configuration tasks

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/tasks/zabbix_server.yml
Last Accessed: 12.05.2016

A.6 Ansible Zabbix Web interface configuration tasks

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/tasks/zabbix_web.yml
Last Accessed: 12.05.2016

A.7 Ansible Zabbix Agent configuration tasks

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/tasks/zabbix_agent.yml
Last Accessed: 12.05.2016

A.8 Ansible Apache configuration tasks

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/tasks/apache_server.yml
Last Accessed: 12.05.2016

A.9 Ansible Wordpress configuration tasks

Dmitri Tsumak. https://github.com/tsudmi/thesis-ansible/blob/master/tasks/wordpress_server.yml
Last Accessed: 12.05.2016

Non-exclusive licence to reproduce thesis and make thesis public

I, Dmitri Tsumak (date of birth: 18th of March 1994),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Large-Scale Provisioning and Configuration Management

supervised by Artjom Lind

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 09.05.2016