CY TECH

Engineer Degree
« Cloud Computing Engineering »

Introduction to Ansible

Cloud Infrastructures course

Anais Dilvy - Robin Lecuppe - Dorian Mailhé - Noa Mourareau-Wong - Esteban Neuvy - Wassim Salablab

Contents

1 Introduction

In an increasingly connected world, time and financial resources used in IT solutions are constantly changing. Today, a company that wants to have a strong digital identity must be prepared for many more users than a decade ago. It is therefore understandable that it must ensure that it has several server instances or network devices, whether for technical or geographical reasons.

To meet this increased demand, some automation tools are emerging to allow IT teams to make up for this loss of time and money. This is the case with **Ansible**, which will be the subject of this report.

So, let's take a look at this tool in its entirety, from its specifications to its use, to its role in the company.

2 What is Ansible and why use it ?

2.1 A bit of context

Ansible has been around since 2012, when Michael DeHaan published the license. The software written in Python quickly became a popular cloud tool, and was acquired by Red Hat in 2015 [1]. It offers several tools such as :

- Inventories, to organise your development nodes and host lists.

- Playbooks, to perform operations from top to bottom, defined by a list of tasks in a YAML file.

2.2 Description

Ansible is an IT automation tool that helps with configuration management and automates the deployment/delivery of applications. More specifically, as their documentation says, "it can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates" [2]. To do this, it uses the SSH network encryption protocol (Secure Socket Shell) which allows for efficient multi-node management. On the other hand, the modules themselves communicate in JSON notation. This significantly optimises the consumption of server resources. This is one of its many advantages, along with the fact that: it is easy to use, requires no agents, and communicates reliably and securely via SSH.

2.3 Examples of use cases

From a simple and classic use case, to a slightly more complex one :
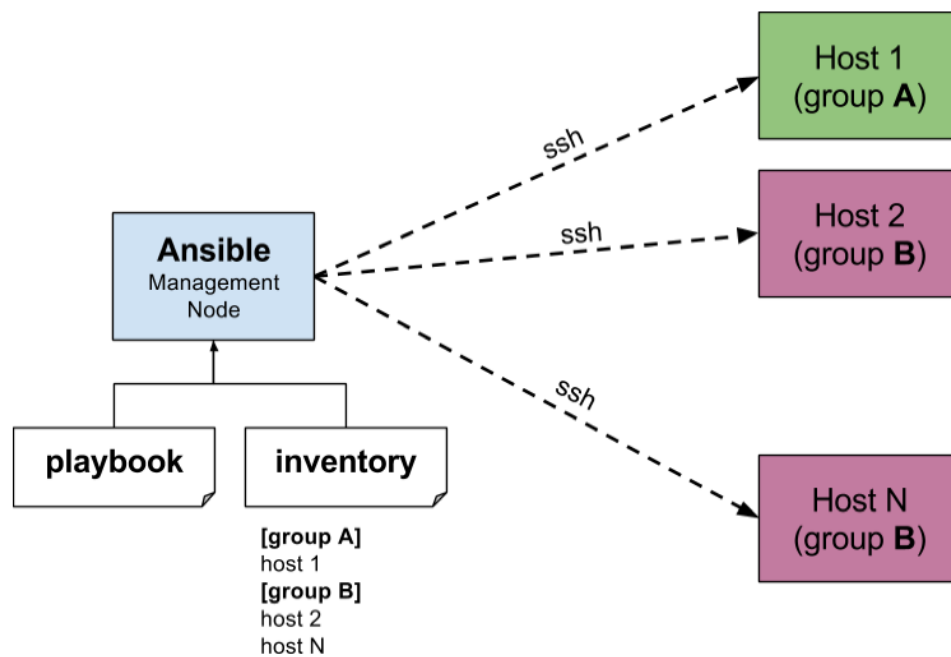
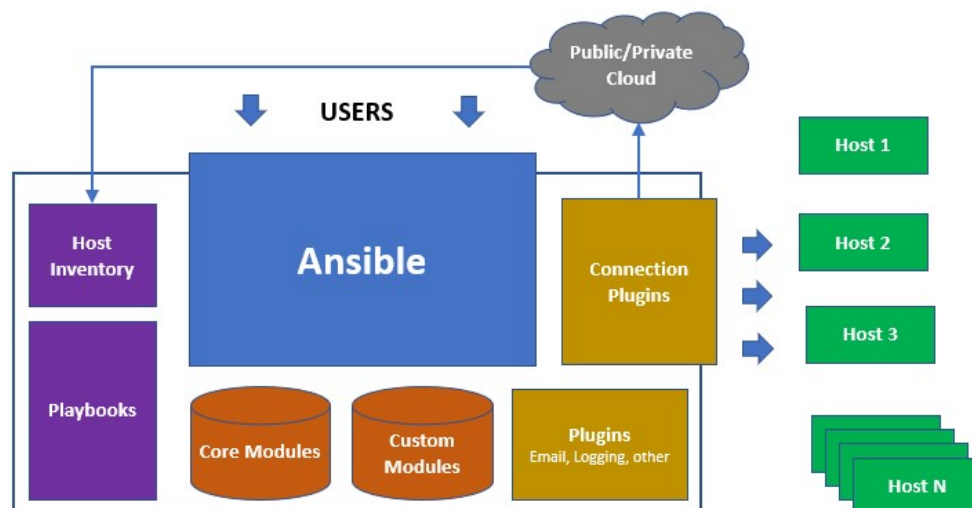Figure 1: A simple use case diagram of Ansible

Figure 2: A complex use case diagram of Ansible

3 How companies used to manage things before Ansible

According to Michael DeHaan, founder of Ansible himself, his software became so popular simply because it welcomed the advent of the cloud [3]. As a practical tool for this emerging technology, it became popular out of necessity. In this respect, it can be said that Ansible did not really have a predecessor. However, there are alternative solutions like Puppet, SaltStack or Chef. As you might expect, all of them were released between 2009 and 2011, with the exception of Puppet which was released in 2005.

It should be remembered that this need for scalability is fairly recent, at least for the general public. Before Ansible, it was managed on a case-by-case basis, by hand. It is to supplant this method that the software was created. Thus, this is where Ansible got its necessity from: it became so with the advent of the Cloud and VM/microservice management.

4 Case study

Here is a case study so you can see how Ansible was used in a company.

BP is a global energy company that explores for and produces oil and gas and manufacturers and markets fuels, lubricants, and petrochemicals—all with a focus on lowering carbon emissions to protect the planet. BP needed a reliable, modern technology infrastructure to speed application development and deployment.

The British company had difficulty with the products teams using multiple delivery models. This caused challenges for the application development and deployment. They wanted to change the infrastructure, so it could operate worldwide and be easily accessed.

To simplify processes and get better productivity, BP used Red Hat OpenShift Container Platform running on AWS (Amazon Web Services) to build an application. This platform provides process automation that empowers product delivery teams with self-service capabilities, a DevOps approach, and a continuous integration/continuous delivery (CI/CD) pipeline. "The combination of microservices, containers, and a fully automated CI/CD platform provides what developers have been asking for for years," said Costall. "They now have full self-service to deliver change from the initial idea, through the innovation, right through to production, as quickly as humanly possible."

With their new application, BP reduced the time to create a new environment from 3 weeks to 7 minutes, allowing developers to innovate quickly and better support business goals. This new platform is also more secure because it is running security scans on every build and container, rather than static scans based on project milestones. Also, the platform empowers DevOps teams to embrace self-service. "We used to think, 'There's the business. There's architecture. There's developers. There's support,'" said Bruno Rothgiesser, digital solutions architect at BP. "Now it's just one team, co-creating software and digital capabilities. That's a big change—a big, positive transformation in the way we work." [4].

5 How does it work ?

This section aims to give a brief summary of Ansible's way of work. But first, let's clarify a few of the key tenets of Ansible before delving deeply into the explanations.

5.1 Some brief definitions

- Modules: Ansible operates by connecting to your nodes and sending them programs known as **modules**. The majority of modules take parameters that specify the desired system state.
  The modules are then executed by Ansible (by default through SSH) and thereafter removed.

- Inventory: The hosts and groups of hosts that the commands, modules, and tasks in a playbook depend on are specified in the **inventory** file. Depending on your Ansible setup and plugins, the file may be in one of several forms.
  .INI and .YAML are examples of common formats.

- Playbooks: Work is done by modules. A play may be created by combining one or more jobs. A **playbook** is made up of two or more plays together. playbooks are collections of commands that run automatically against hosts.
  With extremely fine-grained control over how many computers to handle at once, playbooks can precisely orchestrate different slices of the infrastructure topology.

- Plugins: The hosts and groups of hosts that the commands, modules, and tasks in a playbook depend on are specified in the **inventory** file. Depending on your Ansible setup and plugins, the file may be in one of several forms. INI and YAML are examples of common formats.

6 How to install it?

Ansible is an agentless automation tool, it means that Ansible must be installed on a single host called control node and the devices that are managed by the control node are called the managed nodes. First, we will focus on the control node installation.

In fact, the Ansible control node is very easy to set-up, as it only requires Python 3.8 (or a newer version) to run, according to the official documentation. Moreover the Ansible control node must be installed on UNIX systems only (Linux or MacOS) but it is possible to emulate UNIX systems with Cygwin. Then there are several ways to install it, but the simplest way is to do it with python.
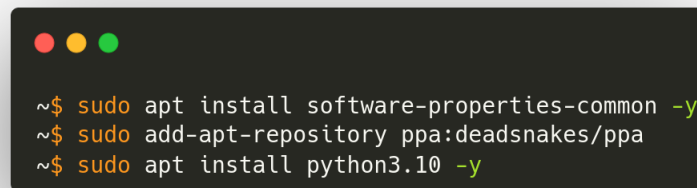
6.1 Python Installation

This section describes the different steps to install Ansible with python "pip" module. The first step to install Ansible with Python is to check that you have Python 3.8 or higher.

Figure 3: Get your Python version

If you have the right version, skip to the next step. Otherwise, your version is probably older than 3.8 and you will have to install a newer. **It is strongly recommended to use a virtual environment to avoid compatibility issues.**
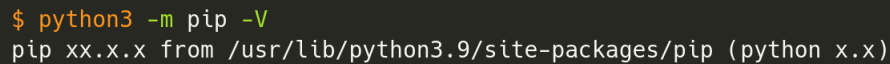
Figure 4: Update Python version

```
~$ sudo apt install software-properties-common -y
~$ sudo add-apt-repository ppa:deadsnakes/ppa
~$ sudo apt install python3.10 -y
```

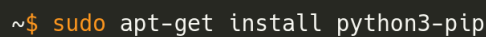Check if pip module is installed with the following command in a terminal :

Figure 5: Get python pip module version

```
$ python3 -m pip -V
pip xx.x.x from /usr/lib/python3.9/site-packages/pip (python x.x)
```

If pip module is not installed in the control node, install it with the following command:

Figure 6: Get python pip module

```
~$ sudo apt-get install python3-pip
```

6.2 Ansible Installation

Once the python pip module has been installed, it is necessary to install Python's Ansible module:

The ansible control node is now installed.

Figure 7: Install Ansible

```
~$ python3 -m pip install --user ansible
```

# 7   How to configure Ansible?

The control node is now installed. Since Ansible is an agentless automation tool that works with SSH to establish the connection between the control nodes and the managed node, we need to install and configure an SSH server.

7.1   SSH installation and configuration

SSH is a network protocol that provides a secure way to access a remote computer and transfer data by encrypting it using a public and private key. The following section shows how to set up an SSH server on an Ubuntu UNIX distribution.

The first step is to install SSH server on the control and managed nodes:

Figure 8: Install SSH

```
~$ sudo apt install openssh-server
```
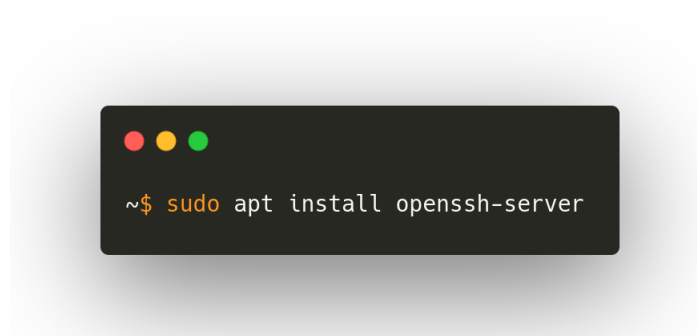
The second step is to setup SSH server on each managed node. To do this, you must modify the SSH configuration file.
Open the configuration file and uncomment the following lines:

- PermitRootLogin yes

- PubkeyAuthentication yes

Figure 9: Open the SSH configuration file

```
~$ sudo nano /etc/ssh/sshd_config
```

Close the file and restart the server.

Figure 10: Restart SSH server

```
~$ sudo service ssh restart
```

To simplify the use of Ansible, it would be interesting to create a root profile on the managed nodes.

Figure 11: Create Root Login

```
~$ sudo passwd root
```

Then generate an SSH key pair and follow the instructions:

Figure 12: Generate SSH key pair

The SSH keys are created. The final step is to share the public key to the remote computer:

Figure 13: Share SSH key

SSH server is set up. It is possible to use Ansible without SSH, you just have to specify the remote connection method. However, SSH is the easiest way to use it.

7.2 Ansible setup

As seen previously, Ansible works thanks to two types of objects: the playbooks which gather the instructions that must be performed by the managed nodes and the inventory which is a list of all the managed nodes. Those files are written in YAML or INI formats. The following section explains how to configure a basic inventory.

7.2.1 Inventory

An inventory is a list of all managed nodes. In this list, they are grouped by usage. For example, the human ressource service does not need the development tools. But all services need the latest version of their operating systems. The inventory file makes it possible to differentiate usage and to target only certain machines in a computer pool.
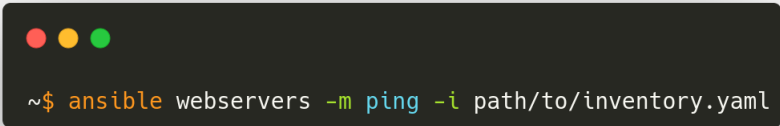
Figure 14: Ansible inventory YAML

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

In this example, there are three groups: "all", "webservers", and "dbservers". Each of them contains hosts. Host may be an IP Address or a complete hostname.
Thanks to these groups, Ansible is able to attribute tasks to a specific group of machines:

Figure 15: Ansible Command

```
~$ ansible webservers -m ping -i path/to/inventory.yaml
```

This command **pings** all the hosts in "webservers" group.

7.2.2 Playbooks

A playbook is a file that contains instruction to perform operations. You must specify lot of fields like:

- The name of the playbook

- Group of host

- Task with their subtasks, their names and the hosts

Figure 16: Ansible Playbook

```
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```
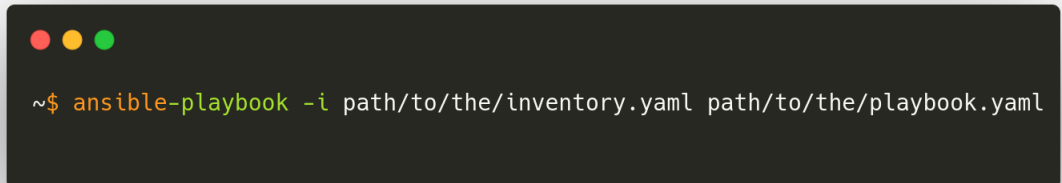
In this example there are two Plays, one targeting the web servers and the other targeting the database server. The first Play updates the Apache web server and copies the Apache configuration file from the control node to the managed nodes. The second Play updates PostgreSQL server and starts it. All these Plays need administrator privileges, which is why there is the Ansible variable **remote_user**.

To execute this Playbook, enter the following command in a terminal:

Figure 17: Run a Playbook

```
~$ ansible-playbook -i path/to/the/inventory.yaml path/to/the/playbook.yaml
```

8 Alternatives

As seen previously Ansible is a server automation and configuration management tool that helps organizations maintain and manage large amounts of virtual and private servers. By allowing to automate repetitive tasks, deploy packages and applications simultaneously, and set up new servers. But it isn't the only tool for that, let's explore now the alternatives to Ansible and what they do.

8.1 CHEF

Chef is an open source tool developed by Opscode in 2009. Its system configuration is built using a Domain Specific Language (DSL) written in Ruby and Erlang. This tool enables fast and consistent scalability as the requirements of a business must evolve. It is able to manage data centers and cloud environments, and maintain high server uptime. Moreover it has two configurations modes, a server/client mode and a standalone mode.

8.1.1 How it works

The way that Chef works is that you write recipes that are compiled into cookbooks and those cookbooks are the definition of how you would set up a node. And a node is a selection of servers that you have configured in a specific way. For example node A could be your Apache Linux servers, node B your mySQL servers and node C your Python servers. Chef is able to communicate back and forth between the nodes to understand what nodes are being impacted and need to have instructions sent out to them to correct that impact. You can also send instructions from the server to the nodes to make a significant update or a minor update. So there is a great communication coming back and forth through its own tool for communication called Knife.
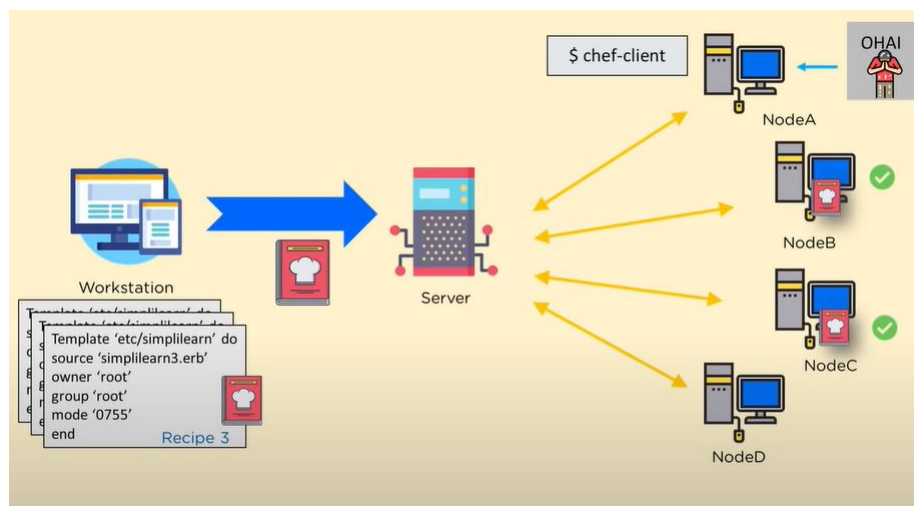
8.1.2   Differences between CHEF and ANSIBLE

Despite their apparent similarity in interoperability and scalability of configuration management, there are still a few differences to note between them. First, Chef uses a master/client architecture with the server running on the master machine while the client-side runs an agent on each client machine. Whereas Ansible uses an agentless architecture by pushing changes over SSH from a single source to many remote servers. Also, Chef Client pulls configuration directly from the server using Ruby DSL while on the other hand Ansible server pushes the configuration to the agent nodes using YAML, and also uses YAML to manage configurations.

8.2   SALTSTACK

Saltstack is an open source tool developed by Thomas S Hatch in 2011. Saltstack is written in Python but is language-agnostic, meaning that it supports any language for configuration files, file types, and templating engines. Saltstack is very flexible, by allowing for agent-only, server-only and agent/server environments. Moreover the solution uses an automated system for detecting and reacting to events within any system, thus offering the most effective monitoring and management system for large, complex environments.

8.2.1   How it works

The way that Salstack works is that you have a master environment and agents that are referred as minions. Master configuration parameter allows Minions to connect to multiple masters simultaneously by specifying a list of the Master addresses. Thus the minion identifies a master by its name or IP adress. A master identifies a minion by its host name. The communication in SSH begins after the acceptance of the minion by the master and after accepting an exchange of encryption keys. Minions can then be commanded using criteria like operating system, regular expression on hostname, architecture type, etc.
The "states" files allow to describe a representation of the state in which a minion (or agent) must be. The pillars are informations about minions stored and generated on the master, whereas "grains" are informations about a minion stored and generated on a minion.

8.2.2   Differences between Saltstack and Ansible

Saltstack is considered to be a more flexible and scalable option than Ansible. But the setup phase is slightly more complicated than with Ansible. Moreover in terms of graphical interface to create and manage their environment, their relatively new web UI is much less developed as compared to Ansible.
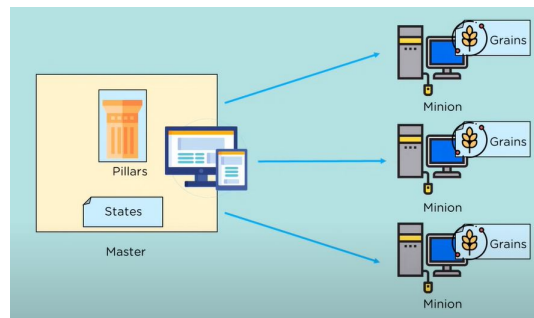
8.3 PUPPET

Puppet is an open source tool developed by Puppet Labs in 2005 and written in Ruby. It uses PuppetDSL with YAML datastore for writting manifests. And it is used by over 30 000 organizations, including Google, Red Hat, Siemens, Stanford, and Harvard universities.

8.3.1 How it works

It is based on a master/slave infrastructure. The master server has the manifests that are put together in a catalog, and those catalogs are then pushed out to the clients over an SSL connection, through the local agent on the client.

8.3.2 Differences between Puppet and Ansible

Puppet uses a server/client architecture, requiring a longer installation process than Ansible, which as an agentless system only needs installation on the master node. In addition, Ansible uses YAML for configuration management while Puppet uses PuppetDSL with YAML datastore. Finally the configuration management language style in Ansible is procedural while Puppet's one is declarative.

9   Conclusion

To conclude, Ansible is a very efficient and easy to use automation tool. From the simplest use cases to the most complex architectures, the software is suitable for both one-time users and large enterprises.

Having tried it, we were all able to master the basics quickly and easily, while understanding the use cases we could have. As the software allows for several installation methods, we were given the choice of our preferred way of doing this. As far as the use itself is concerned, the hardest part was often the management of the VMs and the IP/network management of the configurations: using ansible was much less difficult. Here is the work we have done, the one we presented during the oral presentation : `https://github.com/dilvy-anais/cloud-infra`.

Of course, it can be used with other technologies such as Terraform to be even more efficient and to extend its scope.

References

[1] La Rédaction JDN. *Ansible : comprendre l'outil d'automatisation IT open source (gratuit)*. URL: `https : / / www . journaldunet . fr / web - tech / guide - de - l - entreprise-digitale/1443876-ansible-comprendre-l-outil-d-automatisation- it-open-source-gratuit/`. (last modification: 31.01.2022).

[2] Ansible project contributors. *Ansible Documentation*. URL: `https://docs.ansible. com/ansible/latest/index.html`. (last modification: 09.11.2022).

[3] Ben Rometsch. *How Ansible got started and grew*. URL: `https://opensource.com/ article/21/2/ansible-origin-story`. (last modification: 16.02.2021).

[4] Red hat. *BP case study*. URL: `https://www.redhat.com/en/success-stories/bp`. (last modification: 09.11.2022).