nomios

Paper Ansible

Basic concepts and best practices

Gerrit Van Mol

2020-2021

Contents

List of abbreviations

SRC	Source
SMC	System Management Controller
SSH	Secure Shell
SCP	Secure Copy Protocol
SFTP	Secure File Transfer Protocol
OS	Operating System
DVCS	Distribution Version Control System
CI	Continuous Integration
CD	Continuous Development
SIEM	Security Information and Event Management
PAM	Privileged Access Management
CLI	Command Line Interface

Glossary

Provisioning	The process of setting up IT infrastructure
Batch processing	Running of “jobs” that don’t require user interaction
Staging	A replica of the production environment to run tests

1 What is Ansible

Ansible is an open-source automation engine that can automate repetitive tasks, like configuring systems or provide continuous deployments and zero downtime rolling out updates. It gives more consistency, reliability and scalability to an IT environment. The YAML langue makes it easy to read scripts and configuration files like plain English.

Using only one machine Ansible can orchestrate multiple nodes from one or more control nodes. Nodes are controlled using OpenSSH by the control node. Next to SSH it is also possible to authenticate with Kerberos, LDAP or other authentication management systems. [1]

Ansible can be used for multiple applications like provisioning of virtual machines, configuration management, application deployment, security automation, etc.

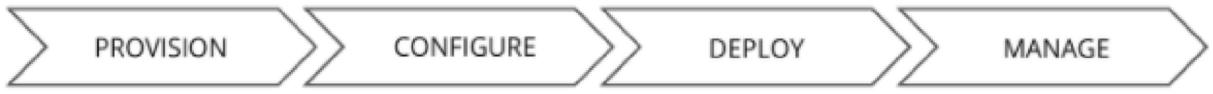


Figure 1: Ansible stage lifecycle [2]

1.2 Use cases in detail

1.2.1 Provisioning

Bare metal servers are occasionally in need of provisioning, Ansible can integrate datacenter management tools to invoke and start provisioning the servers.

When using different hypervisors, virtual storage/networks the diverse selection of modules can support a diverse environment and make it possible to scale more easily.

Next to bare metal servers and hypervisors it is also possible to provision network infrastructure and cloud solutions. The network automation capabilities allow for validation and continuous operation of physical network devices.

When Ansible plays a role for your (public or private) cloud environment it is possible to not only provision the cloud services but also the underlying infrastructure and applications inside the cloud [2].

1.2.2 Configuration management

Ansible is a state-driven resource model which means that it describes the state of the systems and services. The target machines current state does not matter as Ansible will transform the state of the machine as desired. This leads to less potential failures in comparison to script-based solutions as those often create irreversible actions. Ansible also has the possibility to validate the configuration for any potential syntax errors [3].

1.2.3 App deployment

As mentioned earlier the description of the desired state of a machine is defined in a "Playbook". These "Playbooks" are the executed and set the target machines in the desired state, it makes tasks repeatable and reliable.

Playbooks are delivered to a target host with the OS compatible modules over SSH, these modules are comparable to commands in the cli of the target machines OS. Once the machine received the module they are executed on the spot, once the job is done (state changed) the module is removed from the target host.

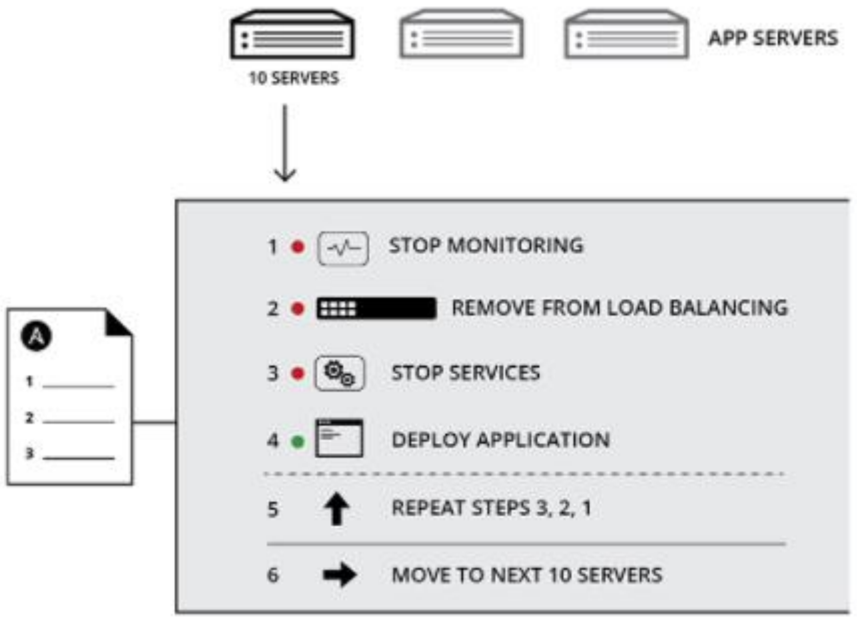


Figure 2: Zero downtime update roll out [4]

As show in the above figure it is possible to deploy applications and updates without the user noticing. This by updating each server or group of servers from the load balancer. When using a DVCS it is possible to create and download artifacts of the deployed application. However, it is also possible to make use of REST API's and update certain variables on another service. For example, push a notification or send a mail when the update was deployed and so on [4]. Using an API, it is also possible to interact with cloud services as these services allow you to insert an SSH key which then grants access to manage cloud instances or network software.

1.2.4 Continuous delivery

The stage and test method allows you to validate and test your Ansible “plays”. In the inventory file it is possible to divide environments into different groups of hosts. This allows to define testing and production machines. It is best however to keep production and testing machines defined in a separate inventory to prevent surprises. When a “play” is ready, it can be tested on the staging machines. If the test passes, you can choose to also automatically run/push the play onto production machines.

Plays will be covered in a later section of this paper. For now, just know that a play is a set of instructions written in YAML to set a machines configuration as desired. These machines could be monitoring systems, networks, load balancers, webservers, etc.

As Ansible a common way to use Ansible is to use it by calling it form a CI system when a play build was successful [5].

Tasks that would take place for testing:

- The CI will make Ansible run a playbook to deploy a staging environment with the new device configuration or application
- When stage test passes a notification can be sent to confirm deployment to production environment
- Once published the configuration or application version can be saved in an artifact on the CI server for later consultation or other purposes.

1.2.5 Security automation

Ansible provides a platform called the Ansible Automation Platform, this platform can combine multiple security solutions in to one. This simplifies and replaces the process of consulting each security solution that a company has running. These solutions exist of SIEM, PAM, IDS, IPS, Firewalls and so on. It is a platform that works with subscriptions, so it is not free. This is because of the support that comes with the platform subscription.

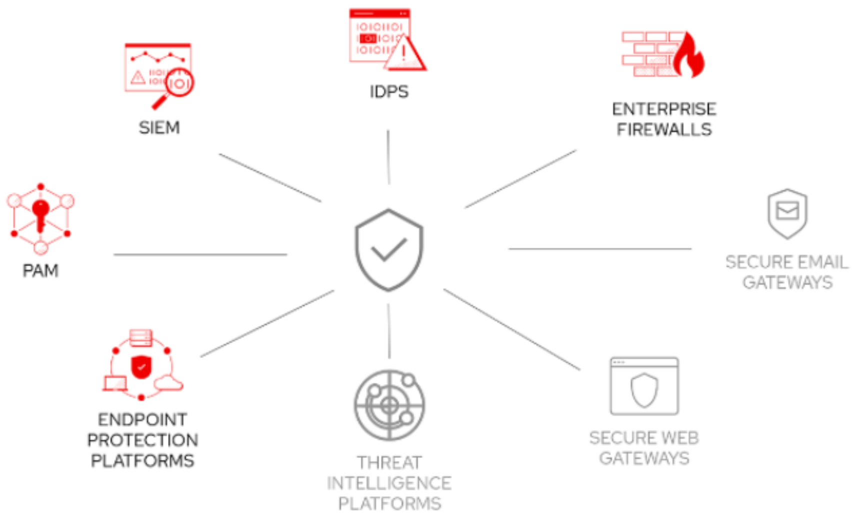


Figure 3: Solutions Ansible can integrate [12]

2 Ansible minimum requirements

Ansible can be run on any Linux or Mac-OS machine but not natively on Windows (only in WSL or docker). To install and run Ansible there are a few prerequisites that a machine must have. Control nodes and managed nodes both have basically the same minimum requirements for system resources:

- 512MB of RAM
- 1 CPU core
- 1GB of storage

Storage capacity depends on use case, it is recommended to go with 10 to 20GB of storage when working with databases or when a lot of output files are being saved. Next to these specifications all nodes that are supposed to run Ansible require the Python interpreter. At least Python 2 (version 2.7 and higher) or Python 3 (version 3.5 or higher) are expected to be installed for Ansible-core 2.11 and Ansible 4.0.0. When using Ansible-core 2.12 or Ansible 5.0.0, Python 3.8 or a higher version will be needed. These Python versions are the minimum for a control node as well as managed nodes [6].

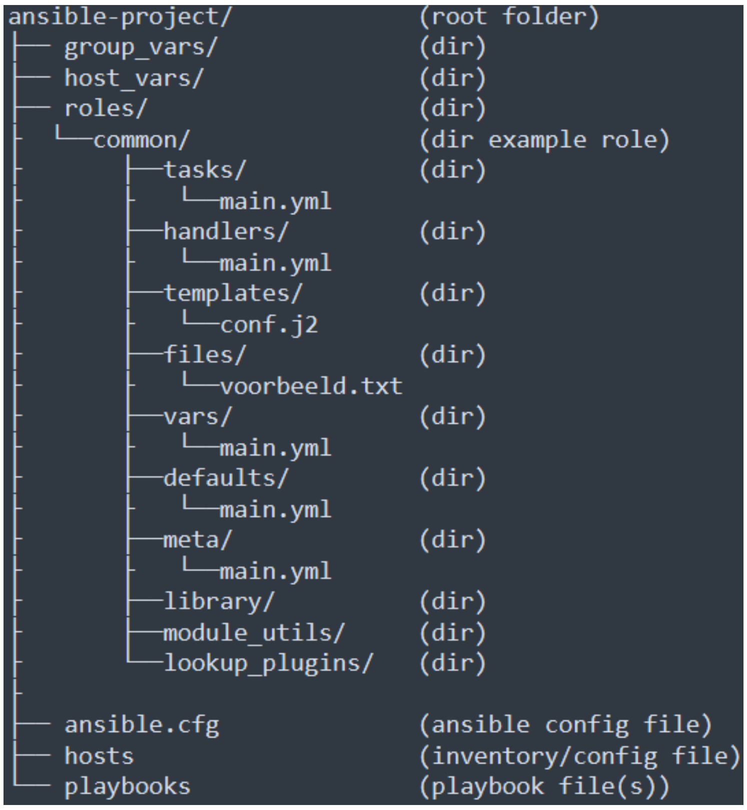
Because the control node needs to be able to talk to the managed nodes, these nodes will therefore need to be accessible over SSH, SFTP or SCP.

3 Ansible architecture

Ansible defines the machines it manages in the “inventory” file. This file is written in the INI or in YAML language. When writing in YAML it is advised to beware of the indentation and use spaces instead of using tabs, it is possible but will sometimes give spacing errors when running the play.

For YML files it is also important to start the file with “---” (three dashes). The indentation determines the hierarchy for the defined hosts or groups.

In the underlying figure is a small overview of the Ansible folder/file structure according to the best practices. That way you get a better understanding of what the following topics will cover and how it relates to each other.



```
ansible-project/                (root folder)
├── group_vars/                 (dir)
├── host_vars/                  (dir)
├── roles/                      (dir)
├   └──common/                  (dir example role)
├          ├──tasks/            (dir)
├          ├   └──main.yml
├          ├──handlers/         (dir)
├          ├   └──main.yml
├          ├──templates/        (dir)
├          ├   └──conf.j2
├          ├──files/            (dir)
├          ├   └──voorbeeld.txt
├          ├──vars/             (dir)
├          ├   └──main.yml
├          ├──defaults/         (dir)
├          ├   └──main.yml
├          ├──meta/             (dir)
├          ├   └──main.yml
├          ├──library/          (dir)
├          ├──module_utils/     (dir)
├          └──lookup_plugins/   (dir)
├
├── ansible.cfg                 (ansible config file)
├── hosts                       (inventory/config file)
└── playbooks                   (playbook file(s))
```

Figure 4: Folder structure example

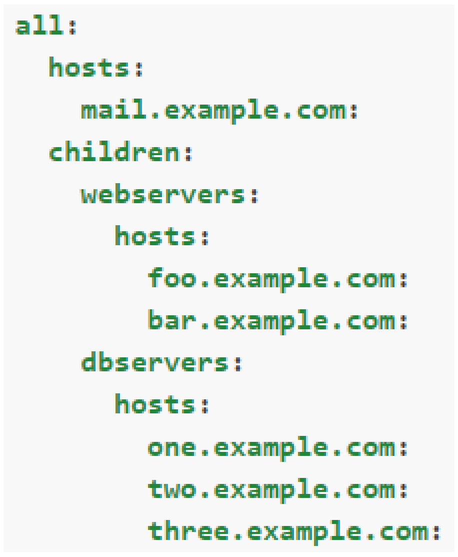
3.1 Inventory

Ansible runs against multiple nodes/hosts at the same time, using a list or group defined in the inventory file. Using patterns (commands) it is possible to target a specific host or group form the inventory file for Ansible to run against.

The inventory file in production is not called "inventory" but "hosts". The default inventory file is located at "`/etc/ansible/hosts`". It is possible to use other or multiple inventory files, the inventory path just needs to be specified in the command line with "`-i <paht>`" option.

Inventory files can be written in YAML or INI, the format that is chosen is a personal preference.

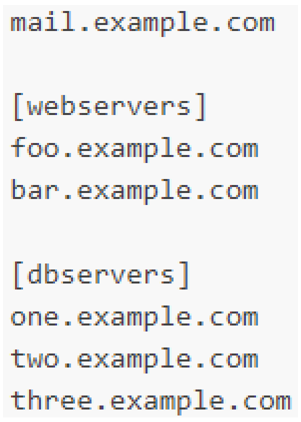
A basic YAML inventory file:



```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:
```

Figure 5: Basic YAML inventory [7]

A basic INI inventory file:

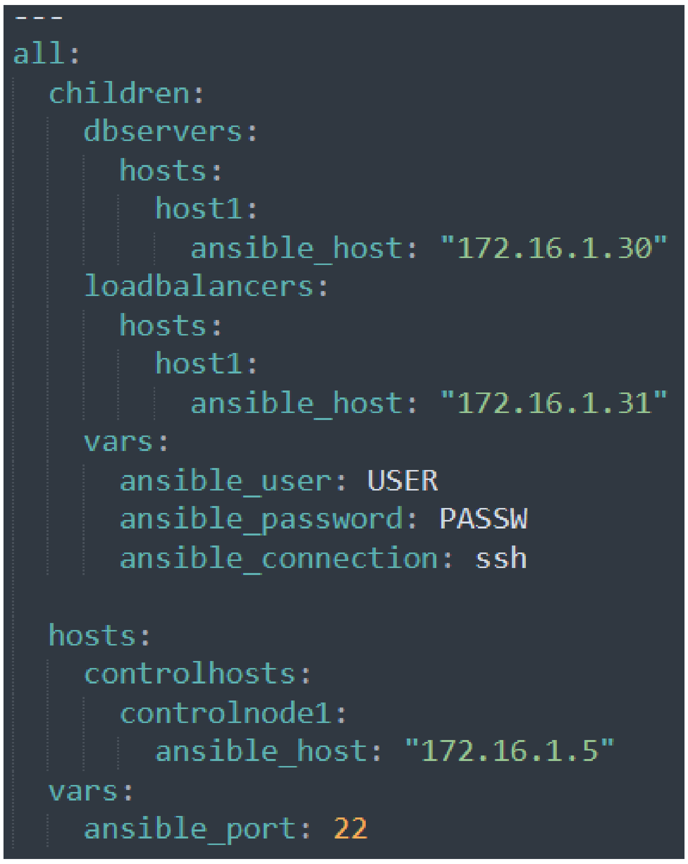


```ini
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

Figure 6: Basic INI inventory [13]

As can be seen in (figure 4), default group “all” contains every host. Every host will always belong to at least two groups (“all” and “ungrouped” or some other group). It is also recommended to group machines by platform, that way variables can be reused for other machines in the same group.

To prevent form giving along many variables to a playbook-command it is recommended to define these in the inventory file. For example, it is possible to define device IP addresses, OS and SSH users directly in the inventory. When devices are only accessible by IP, it is a must to define these, otherwise connection is not possible. If the devices us hostnames definition of the IP is not necessary [7].

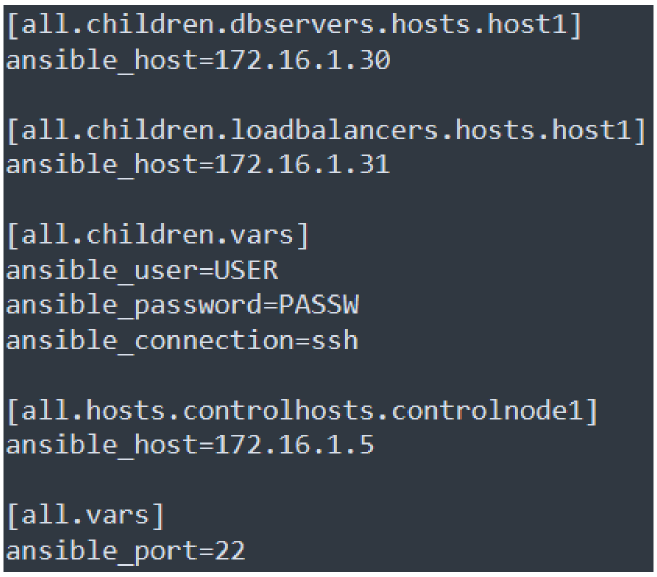


```yaml
---
all:
  children:
    dbservers:
      hosts:
        host1:
          ansible_host: "172.16.1.30"
    loadbalancers:
      hosts:
        host1:
          ansible_host: "172.16.1.31"
    vars:
      ansible_user: USER
      ansible_password: PASSW
      ansible_connection: ssh

  hosts:
    controlhosts:
      controlnode1:
        ansible_host: "172.16.1.5"
  vars:
    ansible_port: 22
```

Figure 7: Inventory with variables – YML format

As displayed in (figure 7) variables are defined for all the hosts in the “loadbalancers” and “dbservers” group. It is also possible to define these variables per host, only difference is that the “vars” keyword would not be necessary. Since the connection is made over SSH it is important to specify a variable with the port number for all groups (port 22).

The following example shows the same inventory file from (figure 7) except that it is now in INI format.



```
[all.children.dbservers.hosts.host1]
ansible_host=172.16.1.30

[all.children.loadbalancers.hosts.host1]
ansible_host=172.16.1.31

[all.children.vars]
ansible_user=USER
ansible_password=PASSW
ansible_connection=ssh

[all.hosts.controlhosts.controlnode1]
ansible_host=172.16.1.5

[all.vars]
ansible_port=22
```

Figure 8: Inventory with variables - INI format

3.2 What is a Playbook

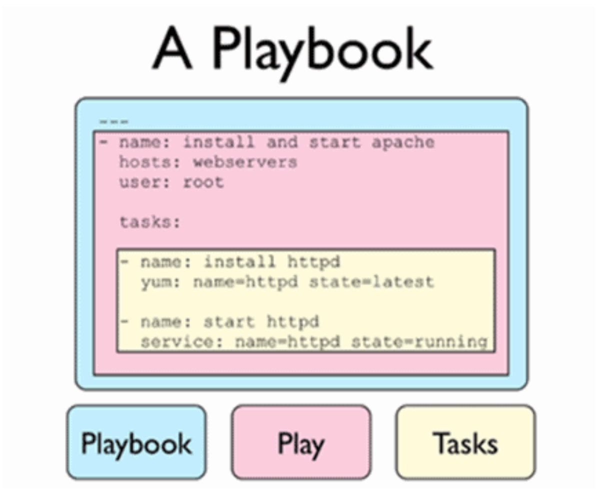
Playbooks are a form of scripts that exist out of a list of commands. These commands are then being sent to remote machines to build an automated infrastructure. Repetitive tasks that needed to be done one at a time can now all be done simultaneously with one command. The contents of a playbook contains a set of configuration values, and is not complicated to understand as some programming languages.

Ansible playbooks consist of one or more tasks these tasks can be combined to form a "play". A playbook exists out of one or more "plays" in an ordered list. Playbooks are basically a list of tasks that automatically execute against hosts.

How playbooks, plays and tasks work together is per definition:

- Playbook: at the highest level and is just a list of plays
- Play: binds tasks to a host/group or list of hosts
- Task: a definition or call to a module/command

In (figure 9) the elements in a playbook are highlighted to give a clear view of what exactly a play, task and playbook is.



```
---
- name: install and start apache
  hosts: webservers
  user: root

  tasks:
  - name: install httpd
    yum: name=httpd state=latest

  - name: start httpd
    service: name=httpd state=running
```

Figure 9: Playbook overview [14]

The only way to execute a playbook by using its name in the command. Running playbooks can be done with the command "`anisble-playbook playbook-name.yml`". Using "`--verbose`" will show a detailed output during execution and is more helpful during troubleshooting [8].

With the syntax checker, it is possible to check for playbook syntax errors as it is a relative common thing. Using the following command:

```
ansible-playbook playbook-name.yml --syntax-check
```

Notice the extra parameter/module "`--syntax-check`" the playbook will be validated. Next to "`--syntax-check`" there are multiple checks one can perform to validate a playbook. A short list of possible checks:

- --list-tasks
- --list-hosts
- --check
- --diff

Ansible Lint is also a command-line tool to help check your playbooks behaviour and if possible, improve it.

3.3 What is a Play

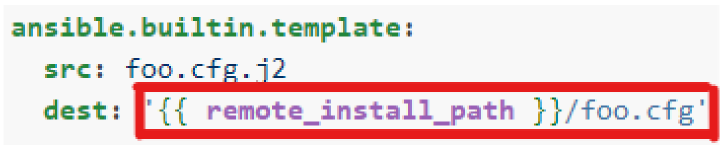
Plays are essentially elements that connect tasks to a host/node where they need to be executed. This way Ansible knows which nodes will be affected and how. A play consists of one or more tasks. In (figure 7) you can see that the host keyword is used to address the targeted group name. The group or host mentioned is the target that will be affected with the following tasks.

3.3.1 Variables

Variables are used for groups that share the same variable values for OS or SSH. This way there are no redundant definitions and maintenance is easier. Variables are comparable to those in the programming world for those who are familiar with a programming language.

Variable declaration can be static (declared in a task itself) or be defined in a subfolder (vars) when using roles. If the variable is declared in a subfolder the variable is given the same name as it is called with. That way Ansible finds it automatically. Every variable declared somewhere else than in the task itself must be encapsulated between double brackets. For example; "{{ variable }}" quotes around those brackets are required otherwise a syntax error is thrown.

In the following (figure 10) a template is copied over from to a specified path (full path is defined in the vars folder and given to the variable).



```
ansible.builtin.template:
  src: foo.cfg.j2
  dest: '{{ remote_install_path }}/foo.cfg'
```

Figure 10: destination file path in variable [15]

If more variables are needed in the same task, it is possible to use a list of variables. These would be declared as a summation list or in a subfolder when using roles, and in a list format known as an array.

Summation of variables in task itself:

```
direction:
   - north
   - east
   - south
   - west
```

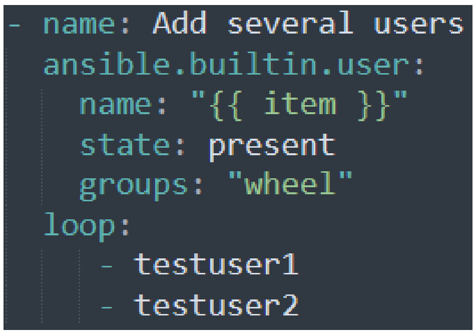
Array of variables declaration in subfolder (vars):

```
direction: [‘north’, ‘east’, ‘south’, ‘west’]
```

Next to giving values along in a task it is also possible to receive value output of a task. The “`register`” keyword saves the output results temporarily. It is afterwards possible to print this output to the screen using “`debug`” or save it in a file by writing it to a specific destination.

3.3.2 Conditions and loops

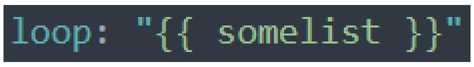
Loops are used when tasks need to be executed multiple times. For example, this is often used to create multiple users or change file ownership on a system. In the example below the creation of multiple users is show with a standard loop.



```
- name: Add several users
  ansible.builtin.user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
     - testuser1
     - testuser2
```

Figure 11: Starndard loop example [16]

The loop variables can be defined in the vars section or subfolder when using roles. An array definition in the vars section would simplify the loop section as follows.

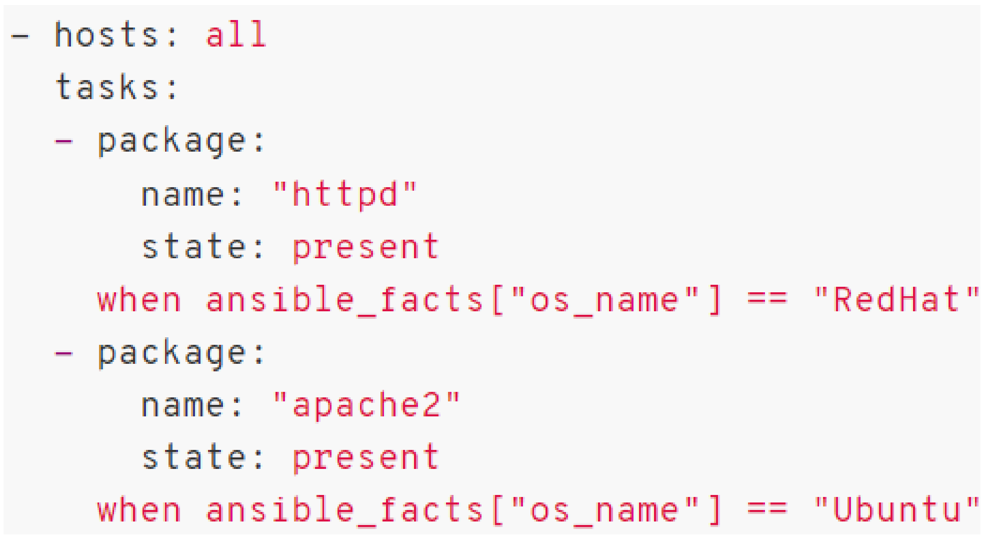


```
loop: "{{ somelist }}"
```

Figure 12: Array with variables to use in loop [16]

In a playbook there is the option to check if a value matches defined criteria. These values can be from a previous task, variable or data from a remote host. Making it possible to only perform a task on a certain host or use variables depending on the given value, etc. Using registered values, loops, roles and or others there are a lot of options to create a condition [9].

For example, it is possible that an administrator needs to install Apache on multiple servers with different operating systems. But the Apache package does noet have the same name on every distribution. When running a task with package “apache2” for an Ubuntu machine this will result in errors if executed on a different distribution. Therefore, conditionals are the perfect solution so specifications can be defined for multiple machines and supplied when a condition is met. Below you can see a visualization of the described situation.



```
- hosts: all
  tasks:
  - package:
      name: "httpd"
      state: present
    when ansible_facts["os_name"] == "RedHat"
  - package:
      name: "apache2"
      state: present
    when ansible_facts["os_name"] == "Ubuntu"
```

Figure 13: Conditionlas example [17]

3.3.3 Sensitive variables

When using variables that are sensitive like passwords or keys are better kept safe in a vault. Ansible vault encrypts variables and files so when they are accidentally pushed to a version control system the data is not show in plain text. Using the “ansible-vault” command it is possible to encrypt or decrypt variables and files. When running playbooks with Ansible vault, the password will need to be specified to temporary decrypt the file. Declaration of the vault password(s) can be done in the “ansible.cfg” file. It is also possible to specify another location or always prompt for the vault password when executing the playbook.

When vault passwords are saved in the “ansible.cfg” file it is important to create a valid “.gitignore” file when using a version control system. Otherwise, it is useless to encrypt your data if the vault password is shared.

More on vaults and encryption can be read here [10].

3.4 What is a Task

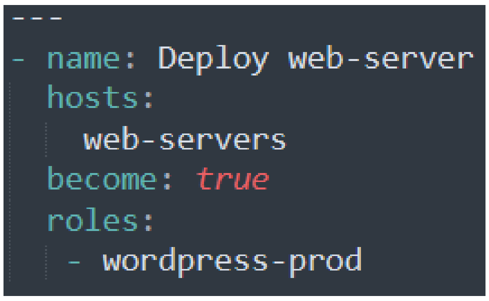
Tasks are used to perform a certain job that needs to be done on a host. Tasks can be named, and it is therefore important to do so. Always describe what the task does and why it does it.

Once a task is finished on the target machine(s) Ansible moves on to the next. Each task is executed as defined from top to bottom. If a task fails on a target host, it will no langer be included to perform further tasks defined in the playbook. When a playbook is run, the output is shown of whether the executed task has made a change (yellow), succeeded (green) or failed (red).

Ansible helps you reach the desired state of a machine, but it is possible that this machine is already in the desired state and the task is not performed. This is because Ansible checks for that state before executing a task, and therefore the final state will not change. Executing a playbook multiple times will conclude to the same outcome. However, beware that this is not the case for all ansible modules.

4 Roles and handlers

A role is a separate part in Ansible and is only once defined in a playbook. Roles have their own sub-folders under the roles folder. Because in most cases not all hosts need the same configuration/tasks, it is often better to split them up. This is where roles come into the picture. They ensure that everything remains organized and that each part, for example variables (vars) are defined in a separate folder. If a task is executed with a mentioned variable, Ansible will automatically search for this variable (usually in the "main.yml" file of the "vars" folder). Like a play, a role defines the tasks and handlers. Roles do not determine which hosts the role will run on. References to a role will therefore always take place in a play.



```
---
- name: Deploy web-server
  hosts:
    web-servers
  become: true
  roles:
   - wordpress-prod
```

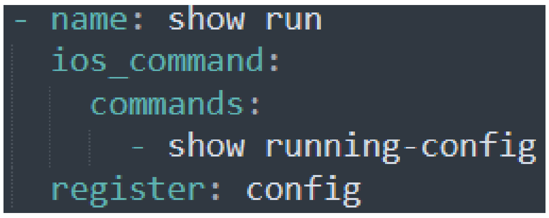
Figure 14: Role reference in a play

Using Ansible Galaxy (a public repository of Ansible roles) you can pull a template folder structure. With the command "`ansible-galaxy init nameOfRole.yml`" all folders are created using a default template. It is also possible to use custom templates or install specific collections. More information about roles and Ansible galaxy can be found here. When pulling a template, the role folder structure is made in your current location of the CLI.

Next to roles there are handlers, these are very similar to tasks. The only difference is that handlers only execute on condition that the "notify" keyword is called. Notify will also only execute when a change has taken place in terms of machine configuration or state. An example use case for this is when the firewall rules have been updated. For the changes to take affect the service needs to be restarted, "notify" will be responsible for this restart.

5 Plugins and modules

Modules are the keywords that are defined within a task and make it possible to interact with specific operating systems, devices and services. In the following (figure 11) you can see that a module for commands is called upon (ios_command). Afterwards a Cisco command is given as variable and will be executed on the remote machine which will return the running config of a Cisco router. The running config is then saved in the "config" variable using the "register" keyword, which makes it possible to view the output or save it as a file in a specific location for later down the line.



```
- name: show run
  ios_command:
    commands:
      - show running-config
  register: config
```

Figure 15: ios_command module

There are a lot of plugins one can make use of. The action plugin is most popular and works together with modules to execute actions required by playbook tasks. They often automatically do prerequisite work in the background before modules execute.

You can enable a custom action plugin by either dropping it into the "action_plugins" directory next to your play, inside a role, or by putting it in one of the action plugin directory sources configured in "ansible.cfg".

An example of a plugin can be seen in (figure 13) called "become" this tells the machine that the connecting client wants to be root user. In the vars or roles subfolder vars the login credentials are supplied.

6 Installation

If Ansible needs to be installed on remote hosts make sure you have an SSH connection and you are in the desired directory location of the system.

Installation steps (Ubuntu):

1. Update/upgrade the machine:
   ```
   sudo apt-get update && upgrade -y
   ```
2. Pull Ansible repository:
   ```
   sudo apt-add-repository ppa:ansible/Ansible
   ```
3. Install Ansible:
   ```
   sudo apt-get install ansible -y
   ```
4. Install python:
   ```
   sudo apt-get install python3 -y
   ```

After installation the directory structure can be made if not already done so.

6.1 Password less authentication

To prevent from having to supply a password when connecting to a machine we can copy an SSH-key to the controlled nodes. This is called password-less authentication and makes the initial connection seamless.

Here’s how to setup password-less authentication:

1. Create an SSH-key: “`ssh-keygen -t -rsa`”
2. Copy the public key to remote machine:
   “`ssh-copy-id -i ~/.ssh/id_rsa.pub user@nodeIP`”
3. Connect to machine (no login password should be required).

References

[1] "About Ansible - Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/index.html. [Accessed 03 11 2021].

[2] R.H. Ansible, "Red Hat Ansible use case - Provisioning," [Online]. Available: https://www.ansible.com/use-cases/provisioning. [Accessed 03 11 2021].

[3] R.H. Ansible, "Ansible for Configuration Management," [Online]. Available: https://www.ansible.com/use-cases/configuration-management. [Accessed 03 11 2021].

[4] R.H. Ansible, "Ansible for Application Deployment," [Online]. Available: https://www.ansible.com/use-cases/application-deployment. [Accessed 03 11 2021].

[5] R.H. Ansible, "Ansible for Continuous Delivery," [Online]. Available: https://www.ansible.com/use-cases/continuous-delivery. [Accessed 04 11 2021].

[6] Ansible, "Installing Ansible — Ansible Docs," [Online]. Available: https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#prerequisites. [Accessed 04 11 2021].

[7] "Build Your Inventory — Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/network/getting_started/first_inventory.html. [Accessed 16 11 2021].

[8] R.H, "Intro to playbooks — Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html. [Accessed 16 11 2021].

[9] R.H, "Conditionals — Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html. [Accessed 22 11 2021].

[10] R.H, "Encrypting content with Ansible Vault — Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/vault.html. [Accessed 17 11 2021].

[11] N.B.N.V., "Security- en netwerkoplossingen voor ondernemingen en de publieke sector," 02 11 2021. [Online]. Available: https://www.nomios.be/. [Accessed 03 11 2021].

[12] R.H. Ansible, "Automating Endpoint Protection with Ansible," [Online]. Available: https://www.ansible.com/blog/automating-endpoint-protection-with-ansible. [Accessed 04 11 2021].

[13] R.H, "How to build your inventory — Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html. [Accessed 16 11 2021].

[14] S. Singh, "Ansible playbook overview," 24 06 2021. [Online]. Available: https://k21academy.com/devops-foundation/ansible-playbook-galaxy-tower/. [Accessed 16 11 2021].

[15] R.H, "Using Variables — Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html. [Accessed 17 11 2021].

[16] R.H, "Loops - Ansible Documentation," [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#standard-loops. [Accessed 22 11 2021].

[17] "An introduction to Ansible facts," [Online]. Available: https://www.redhat.com/sysadmin/playing-ansible-facts. [Accessed 22 11 2021].