Ansible is used for: provisioning, configuration management, continuous delivery, application deployment, security compliance - repetitive tasks that require hundreds of executions on hundreds of servers

Ansible is simple, powerful and agentless

Default Ansible configuration file: /etc/ansible/ansible.cfg

To override default parameters in Ansible, you may copy the ansible.cfg file in the working directory and change the default values

You can also specify the location to the cfg file through an environmental variable:

$ANSIBLE\_CONFIG=/opt/ansible-web.cfg ansible-playbook playbook.yml

The order Ansible pick to take into consideration cfg files:

1) Paramater specified through an environmental variable

2) ansible.cfg in working/current directory

3) .ansible.cfg in the user home directory

4) /etc/ansible/ansible.cfg

The used cfg does not have to have all the values written – only the ones that you would like to override and it will take the default values for other parameters from the next ansible.cfg files in the priority chain.

Let’s say I want to override a single paramter from the CLI, without copying it to the current directory, this can be achieved. This example is for the gathering paramter which is „implicit”, changing it to explicit:

ANSIBLE\_GATHERING=explicit ansible-playbook playbook.yml

If you want to persist this change throught the shell session, you may set the environment variable shell wide and untill you exit the shell this will be persistent:

export ANSIBLE\_GATHERING=explicit

ansible-playbook playbook.yml

View configuration option, corresponding variables and what they mean: ansible-config list

Too see which .cfg file is active for a playbook: ansible-config view

To see a comprehensive list of current settings Ansible has picked up and where it picked that from:

ansible-config dump

Example:

export ANSIBLE\_GATHERING=explicit

ansible-config dump | GATHERING

**YAML files**

Ansible playbooks are text files or config files that are written in YAML. YAML file is used to represent data (configuration data)

Key: Value pair

Calories: 105

Fat: 0.4g

Carbs: 27g

Array/lists:

Fruits:

- Orange

- Apple

- Banana

A dictionary is a set of properties grouped together under an item

Banana:

Calories: 105

Fat: 0.4g

Carbs: 27g

Dictionary is an unordered collection (the properties can have different order, as long as properties values match)

List is is an ordered collection ( they have position, so the order matters)

# represents comment

Ansible inventory

Ansible can work with one or multiple systems in your infra in the same time (SSH – Linux, Powershell Remoting – Windows) – that’s what makes Ansible agentless – you don’t need to install additional software on the target machines in order to work with Ansible

Information about target systems is stored in an inventory file – if you don’t create an inventory file, by default Ansible uses the inventory file located in /etc/ansible/hosts – it is ini format file

Define a group of servers:

[servergroup]

server1.com

server2.com

You can use an inventory parameter that is used to specify the FQDN or IP address of a server (ansible\_host)

There are more inventory parameters like ansible\_host, like the following:

ansible\_connection – ssh(linux server)/winrm(windows server)/localhost(work with the localhot and not a remote host) – what defines how ansible connects to the target server, if it is linux, windows or localhost

ansible\_port – 22/5986 – by default it is set to 22 for ssh, but it can be changed with this parameter

ansible\_user – root/administrator – defines the user used to make remote connection (by default root)

ansible\_ssh\_pass – Password – defines the SSH password for linux – not ideal to store passwords in plain text format like this – the best practice is to set SSH key passwordless authentication between the servers

Note: For Linux based hosts, use ansible\_ssh\_pass parameter and for Windows based hosts, use ansible\_password parameter.

Example:

[servergroup]

web ansible\_host=server1.com ansible\_connection=ssh ansile\_user=root

db ansible\_host=server2.com ansible\_connection=winrm ansile\_user=admin

Ansible supports two main types of inventory formats, ini and yaml:

Ini format is the simplest and the most straightforward which have few servers and users. Example:

[webservers]

web1.example.com

web2.example.com

[dbservers]

db1.example.com

db2.example.com

Yaml format is more structured and flexible than the ini format. Example:

all:

children:

webservers:

hosts:

web1.example.com

web2.example.com

dbservers:

hosts:

db1.example.com

db2.example.com

student-node :- This host will act as an Ansible master node where you will create playbooks, inventory, roles etc and you will be running your playbooks from this host itself.  
  
  
node01 :- This host will act as an Ansible client/remote host where you will setup/install some stuff using Ansible playbooks. Below are the SSH credentials for this host:  
  
  
User: bob  
Password: caleston123  
  
  
node02 :- This host will also act as an Ansible client/remote host where you will setup/install some stuff using Ansible playbooks. Below are the SSH credentials for this host:  
  
  
User: bob  
Password: caleston123  
  
  
Note: Please type exit or logout on the terminal or press CTRL + d to log out from a specific node.

To define groups that consist of other groups, you have to create it with :children – to organize the inventory in a hierarchical manner

[boston\_nodes]

sql\_db1

web\_node1

[dallas\_nodes]

sql\_db2

web\_node2

web\_node3

[us\_nodes:children]

boston\_nodes

dallas\_nodes

Where:

**[boston\_nodes]** and **[dallas\_nodes]** are regular groups that include specific hosts in Boston and Dallas, respectively.

**[us\_nodes:children]** is a special group that is composed of other groups, specifically **boston\_nodes** and **dallas\_nodes**. This means that any tasks or plays defined for the **[us\_nodes]** group will be applied to all hosts in both **boston\_nodes** and **dallas\_nodes**.

Ansible variables which stores information that varies with each host

To add a variable in an INI format: ansible\_connection=ssh. this is a variable

To add a variable in an yaml format, we would have to define vars in a key: value pair

-

name: Add DNS server to resolv.conf

hosts: localhost

vars:

dns\_server: 10.1.250.10

tasks:

- lineinfile:

path: /etc/resolv.conf

line: ’nameserver 10.1.250.10’

# to use the variable defined above, you can:

line: ’nameserver {{ dns\_server }}’

using variables through the double brackets it’s called Jinja2 templating ’{{ variable }}’

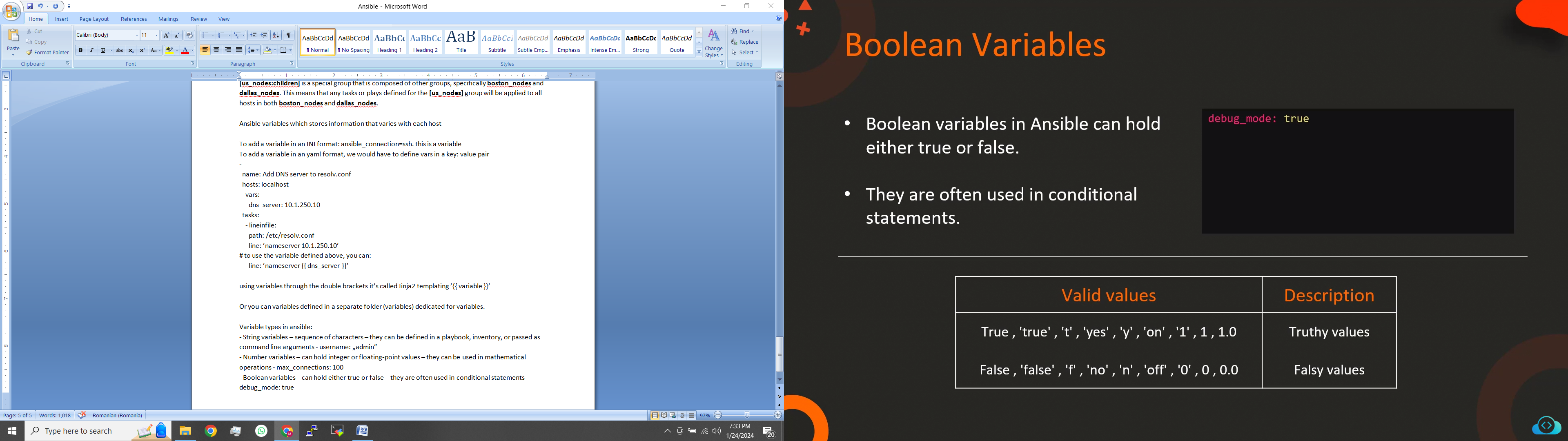
Or you can variables defined in a separate folder (variables) dedicated for variables.

Variable types in ansible:

- String variables – sequence of characters – they can be defined in a playbook, inventory, or passed as command line arguments - username: „admin”

- Number variables – can hold integer or floating-point values – they can be used in mathematical operations - max\_connections: 100

- Boolean variables – can hold either true or false – they are often used in conditional statements – debug\_mode: true



- List variables – cam hold an ordered collection of values and can be of any type:

packages:

- nginx

- postgresql

- git

- Dictonary variables – can hold a collection of key-value pairs – the keys and values can be of any type

If you define a variable at the host level, it will take precedence over the variable defined at the group level, example:

/etc/ansible/hosts

web1 ansible\_host=171.20.1.100 dns\_server=10.5.5.4

web2 ansible\_host=171.20.1.101

[web\_servers}

web1

web2

[web\_servers:vars]

dns\_server=10.5.5.3

The outcome would be that for web1 dns\_server=10.5.5.4 because it is defined at the host level. for web 2 it will be dns\_server=10.5.5.3

Furthermore, a variable defined in the playbook takes precedence over the variables definet at the group or host level

You could pass an extra variable in a command that takes precedence over all:

ansible-playbook playbook.yml –extra-vars „dns\_server=10.5.5.6”

Highest priority:

1. Extra Vars

2. Play Vars (given through playbook)

3. Host Vars

4. Group Vars

Register variables, register output scope:

To store the output of one task to use it later:

let’s say you want to get the hosts from the first task in the task module you introduce

---

- name: Check /etc/hosts file

hosts: all

tasks:

- shell: cat /etc/hosts

register: result #the register directive with the first command and specify a variable name (result)

- debug:

var: result / result.rc or result.stdout, depending if you want to see the return code or the standard output

Another way to do this, without the debug module:

---

- name: Check /etc/hosts file

hosts: all

tasks:

- shell: cat /etc/hosts

ansible-playbook –i inventory playbook.yml –v

Variable scoping – a scope defines the accessability or the visibility of a variable.

A variable is accessible depending on how and where it is defined

Variables scopes – host – the variable is accessible within a play that is run for that host, play – when you define a variable within a play, like in the above example, it is accessible only on that play, global variables – accesible everywhere throught the playbook execution (extra variables)

Magic variables is called hostvars and can be used to get variables defined on a another host:

hostvars[’web2’].dns\_server

Examples:

---

- name: Print dns server

hosts: all

tasks:

- debug:

msg: ’{{ hostvars[’web2’].dns\_server }}’ # get the DNS server defined on web2 node

or: msg: ’{{ hostvars[’web2’].ansible\_facts.host }}’ # get the host name, or IP of web2 node

or: msg: ’{{ hostvars[’web2’].ansible\_facts.architecture }}’

or: msg: ’{{ hostvars[’web2’].ansible\_facts.devices }}’

or: msg: ’{{ hostvars[’web2’].ansible\_facts.mounts }}’

or: msg: ’{{ hostvars[’web2’].ansible\_facts.processor }}’

this could be written in another way:

or: msg: ’{{ hostvars[’web2’][’ansible\_facts’][’mounts’] }}’

Another Magic Variable is: groups – return all hosts under a given group

msg: ’{{ groups[’<group-name’] }}’

Another Magic Variable is: groups\_names – return all the groups the current host is part of

msg: ’{{ group\_names }}’

Another Magic Variable is: inventory\_hostname – gives you the name configured for the host in the inventory file

msg: ’{{ inventory\_hostname }}’

Ansible facts

When you run a playbook and ansible connects to a target machine it first collect information about the machine (basic system information like the system architecture, version of the OS, processor and memory details, serial number etc), it collects information about the host network connectivity (the different interfaces, IP addresses, FQDN, MAC address etc) as well as device info (different disks, volume, mounts, and the amount of space available on them), date time on those system and other settings.

This information are known as FACTS in Ansible. Ansible gathers these facts using the Setup module which is run automatically by Ansible when you run an Ansible Playbook (first task run by ansible)

All facts gathered by Ansible are stored in a variable named ansible\_facts ( in the debug module specify var: ansible\_facts)

You could choose to not run the facts gathering task by entering between hosts and tasks:

gather\_facts: no #you can turn it on with true

Gathering facts is by default enabled in the /etc/ansible/ansible.cfg file

gathering = implicit # you can turn it off with explicit

Playbook setting will take precedence over the config file

**Lab – Ansible Variables and Facts - Example of ansible playbooks:**

**inventory:**

localhost ansible\_connection=local nameserver\_ip=8.8.8.8 snmp\_port=160-161

node01 ansible\_host=node01 ansible\_ssh\_pass=caleston123

node02 ansible\_host=node02 ansible\_ssh\_pass=caleston123

[web\_nodes]

node01

node02

[all:vars]

app\_list=['vim', 'sqlite', 'jq']

user\_details={'username': 'admin', 'password': 'secret\_pass', 'email': 'admin@example.com'}

**ansible.cfg:**

[defaults]

host\_key\_checking = False

**app\_install.yaml:**

---

- hosts: all

become: yes

tasks:

- name: Install applications

yum:

name: "{{ item }}"

state: present

with\_items: "{{ app\_list }}"

**playbook.yaml:**

---

- hosts: localhost

vars:

car\_model: 'BMW M3'

country\_name: 'USA'

title: 'Systems Engineer'

tasks:

- command: 'echo "My car is {{ car\_model }}"'

- command: 'echo "I live in the {{ country\_name }}"'

- command: 'echo "I work as a {{ title }}"'

**user\_setup.yaml:**

---

- hosts: all

become: yes

tasks:

- name: Set up user

user:

name: "{{ user\_details.username }}"

password: "{{ user\_details.password }}"

comment: "{{ user\_details.email }}"

state: present

**update\_service.yml:**

- hosts: all

tasks:

- name: Install a new package

apt:

name: new\_package

state: present

- name: Update the service

service:

name: my\_service

state: restarted

- name: Check service status

service:

name: my\_service

state: started

When update\_service.yml playbook is already ran on a server and we try to run it again in check mode against same server, Ansible will indicate which tasks would result in changed status.

The task Install a new package would not be marked as changed because it's just ensuring the package is present and that was already installed on the server from earlier execution(s).

The task Update the service would be marked as changed because restarting a service is a change in state.

The task Check service status would not be marked as changed since it's only verifying the service's state, and previous task has already started it.

**configure\_database.yml that modifies a configuration file on your database servers:**

- hosts: all

tasks:

- name: Set max connections

lineinfile:

path: /etc/postgresql/12/main/postgresql.conf

line: 'max\_connections = 500'

- name: Set listen addresses

lineinfile:

path: /etc/postgresql/12/main/postgresql.conf

line: 'listen\_addresses = "\*"'

Playbook – a single YAML file which contains:

- Play – defines a set of activities (tasks) to be run on hosts

- Task - an action to be performed on the host, such as:

- Execute a command (ansible module – ex – command: date)

- Run a script (ansible module – ex – script: test\_script.sh)

- Install a package (ansible module – ex – yum:

name: httpd

state: present

- Shutdown/Restart (ansible module – ex – service:

name: httpd

state: started

ansible –doc –l

ansible-playbook --help

There are several modes for verifying playbooks:

- Check mode – dry run mode where ansible executes the playbook without making any actual changes on the hosts – allows you to see what changes would be made on the hosts without applying them

To run a playbook in check mode use the --check

Example: ansible-playbook install\_nginx.yml --check

- Diff mode – provides a before(current state)-after comparison of playbook changes when used with check mode which can help us understand and verify the impact of the playbook changes before applying them

To run a playbook in diff mode: --diff. Example:

ansible-playbook configure\_nginx.yml --check --dif

Syntax check: --syntax-check. Example:

ansible-playbook configure\_nginx.yml --syntax-check

Ansible-lint is a command line tool that performs linting on Ansible playbooks, roles and collections.

- checks you code for potential errors, bugs, stylistic erros and suspicious constructs.

- it’s akin to having a seasoned ansible mentor guiding you, proiding valuable insights and catching issues that might have slipped past your notice.

ansible-lint database\_setup.yml

ansible\_os\_family is a variable used by Ansible to define the various os flavours of the Linux system

Ansible conditionals, examples:

when: ansible\_os\_family == ”Debian”

when: ansible\_os\_family == ”RedHat” **or**

ansible\_os\_family == ”SUSE”

when: ansible\_os\_family == ”Debian” **and**

ansible\_distribution\_version == ”16.04”

Conditionals in loops