

Ansible Automation:

Ansible is a powerful automation tool that can replace many Bash scripts with more maintainable and scalable solutions. Ansible is an automation tool used for configuration management, application deployment, and task automation.

Key Ansible concepts:

1. Playbooks: YAML files containing a set of tasks to be executed on remote hosts.
2. Tasks: Individual units of work in a playbook.
3. Modules: Pieces of code Ansible executes to perform specific operations.
4. Inventory: A list of managed nodes that Ansible can work with.

Ansible Playbook Structure

```
- name: Playbook Name
  hosts: target_hosts
  become: yes/no
  vars:
    variable1: value1
  tasks:
    - name: Task 1 Name
      module_name:
        param1: value1
        param2: value2
```

- ``name``: A description of what the playbook or task does.
- ``hosts``: Specifies which hosts from the inventory this play applies to.
- ``become``: Whether to escalate privileges (like sudo).
- ``vars``: Define variables used in the playbook.
- ``tasks``: A list of tasks to be executed.

Ansible configuration file:

When we install ansible by default configuration files will get created in the following location:

/etc/ansible/ansible.cfg

This configuration file contains several sections, they are:

1. Defaults
2. Inventory
3. Privilege_escalation
4. SSH_connection
5. Paramiko_connection
6. Persist_connection
7. Colors

Here are some frequently used configuration options:

1. In the [defaults] section:
 - inventory: Specifies the default inventory file
 - remote_user: Default username for SSH connections
 - host_key_checking: Whether to check SSH host keys
 - roles_path: Where to look for roles
 - forks: Number of parallel processes to use
2. In the [privilege_escalation] section:
 - become: Whether to use privilege escalation by default
 - become_method: Default method for privilege escalation (e.g., sudo, su)
 - become_user: Default user to become when using privilege escalation
3. In the [ssh_connection] section:
 - ssh_args: Additional SSH arguments
 - control_path: Location of ControlPath sockets

Environment Variable:

``ANSIBLE_CONFIG` = `/opt/ansible_web.cfg``

Copy of Default Config File in current directory:

``/opt/web_playbooks/ansible.cfg``

Config file in home directory:

``ansible.cfg``

Default Config File:

``/etc/ansible/ansible.cfg``

If we have all types of configuration files then it follows the priority:

1. Environmental variable: 1st priority is always to the parameters configured in the file specified through an environmental variable

``ANSIBLE_CONFIG` = `/opt/ansible_web.cfg``

2. Current directory config file: 2nd priority **ansible.cfg** file in the current directory
3. Home directory config file: **.ansible.cfg** file 3rd priority in users home directory
4. Default config file

Example of ansible.cfg:

```
[defaults]
inventory = ./inventory
Log_path = /var/log/ansible.log
library= /usr/share/my_modules
roles_path=atc/ansible/roles
action_plugins=/usr/share/ansible/plugins/action
remote_user = ansible
host_key_checking = False
gathering= implicit
timeout=10
forks = 5

[privilege_escalation]
become = True
become_method = sudo
become_user = root

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

Ansible configuration variables:

There are different ways to pass the environmental variables in

1. For single playbook:

```
ANSIBLE_GATHERING= explicit ansible-playbook playbook.yml
```

2. If we want throughout the shell session, up to we exit from the shell:

```
export ANSIBLE_GATHERING= explicit  
Ansible-playbook playbook.yml
```

3. If we want to change on different shells, on different users on different systems is to create a local copy of configuration file in playbooks directory and update the parameter:

```
/opt/web-playbooks/ansible.cfg  
gathering =explicit
```

To find the different configuration options, what are the corresponding environmental variables are and what they mean

View configuration:

- To find the different configuration options, what are the corresponding environmental variables are and what they mean

ansible -config list ⇒ list all the configurations

- We have different config files in the system in default /etc/ansible.ansible.cfg , one in present directory , one in home directory, to see which config file is in active, we use

ansible-config view ⇒shows the current active config file details

- Shows as comprehensive list of current settings picked up, and where it is picked up

ansible-config dump ⇒ shows the current settings

Eg:

```
export ANSIBLE_GATHERING=explicit
ansible-config dump | grep GATHERING
DEFAULT_GATHERING(env:ANSIBLE_GATHERING)=explicit
```

- **Version Control:** Keep your `ansible.cfg` in version control along with your playbooks.
- **Project-Specific Configurations:** Use project-specific `ansible.cfg` files in your project directories for settings that should apply only to that project.
- **Comment Your Configurations:** Use comments (lines starting with `;` or `#`) to explain non-obvious settings.
- **Security:** Be cautious with settings like `host_key_checking = False`. While convenient for testing, it can be a security risk in production environments.
- **Use Environment Variables:** For sensitive information, use environment variables instead of hardcoding values in `ansible.cfg`.
- **Regular Review:** Periodically review and update your configuration to ensure it aligns with current best practices and your project needs.

If we want to change only one parameter in the config file , we dont need to copy the whole default config file , instead of copying the whole config file, we can override the single parameter using environment variables

What the environment variable should be?

Change the parameter in to uppercase and add the ansible word as prefix to it in uppercase

gathering =implicit ANSIBLE_GATHERING=explicit ⇒ this
environmental variables have highest precedence

YAML:

- Ansible playbooks or text file or config files are written in YAML
- YAML is used to represent config data
- Key value pair, separated by colon
- Space should be mandatory in between colon and value
- Number of spaces in front of each property should be same

Key value pair:

Fruit: Apple

Vegetable: Carrot

Liquid: Water

Array/list:

Fruits:

- Oranges
- Apple
- Banana

Vegetables:

- Carrot
- tomato

Dictionary/map:

Banana:

Calories: 104 #here the space up to calories and space up to fat should be same

Fat: 0.4g

Grapes:

Calories: 62

Fat: 0.3g

Dictionary vs list vs list of dictionaries:

Dictionary: if we want to display the all details of the single item/product we use dictionary

List: stores multiple items of same type of object

eg:

- red carer
- blue car
- Black car

List of dictionary:

Stores all info about each car:

- Color: blue
- Model:
 - Name: Corvette
 - Model: 1995
- transmission: manual
- Color: black
- Model:
 - Name: Corvette
 - Model: 1996
- transmission: manual
- Color: grey
- Model:
 - Name: Corvette
 - Model: 1997
- transmission: manual

Dictionary: unordered

List: ordered

Eg:

Dictionary:

1. Banana:
 - Calories: 105
 - Fat: 0.4g
2. Banana:
 - Fat: 0.4g
 - Calories: 105

Both 1 and 2 dictionaries are equal, but list:

1. Fruits:
 - Oranges
 - Grapes
 - Banana
 - Apple
2. Fruits:
 - Banana
 - Apples
 - Grapes
 - oranges

Both 1 and 2 list are not same because of their order

List of directories:

```
- name: apple
  color: red
  weight: 100g
- name: orange
  weight: 90g
  color: orange
- name: mango
  color: yellow
  weight: 150g
```

```
~
~
~
~
~
~
~
~
```


Ansible inventory:

- Ansible can connect to multiple servers by using ssh in linux, powershell in windows
- Agentless: to work with ansible we no need to install any other software on target machines.
- Information of target machines is stored in inventory file, if we don't create that file, the ansible uses the default inventory file to store the information about the target machines(.: etc/Ansible/hosts location)
- Inventory file is in ini format, simply displays n no. of servers one after the other.
- Way 1:
server1.company.com
server2.company.com
server3.company.com

- way2:

[Mail]

server1.company.com
server2.company.com

[db]

server3.company.com
server4.company.com

For giving alias name:

1. Alias name is given in beginning of line, and then address is assigned to `ansible_host` parameter
2. **Ansible_host** is an **inventory parameter** used to specify the ip address of a server
3. Another inventory parameters:
 - a. `ansible_connection-ssh/winrm/localhost` # defines how ansible connects to the server, like through windows or linux , etc
 - b. `ansible_port-22/5986` # default it is set to 22
 - c. `ansible_user-root/administrator` # defines user who is creating the connection like root or admin
 - d. `Ansible_ssh_pass- password` #display like text format which is not safe

Example:

```
web ansible_host=server1.company.com ansible_connection=ssh ansible_user=root
```

```
db ansible_host=server2.company.com ansible_connection=winrm ansible_user=admin
```

```
mail ansible_host=server3.company.com ansible_connection=ssh ansible_ssh_pass =  
p!2s#
```

```
localhost ansible_connection=localhost
```

Inventory formats:

1. INI
2. YAML

The image shows a comparison between INI and YAML inventory formats. On the left, the 'INI' tab is selected, displaying a hierarchical tree diagram of nodes and a text box stating: "The INI format is the simplest and most straightforward." On the right, the 'YAML' tab is shown, displaying two groups of servers: "[webservers]" with "web1.example.com" and "web2.example.com", and "[dbservers]" with "db1.example.com" and "db2.example.com".

INI:

Basic that follows in start up, they does only less number of tasks, like managing db, web.

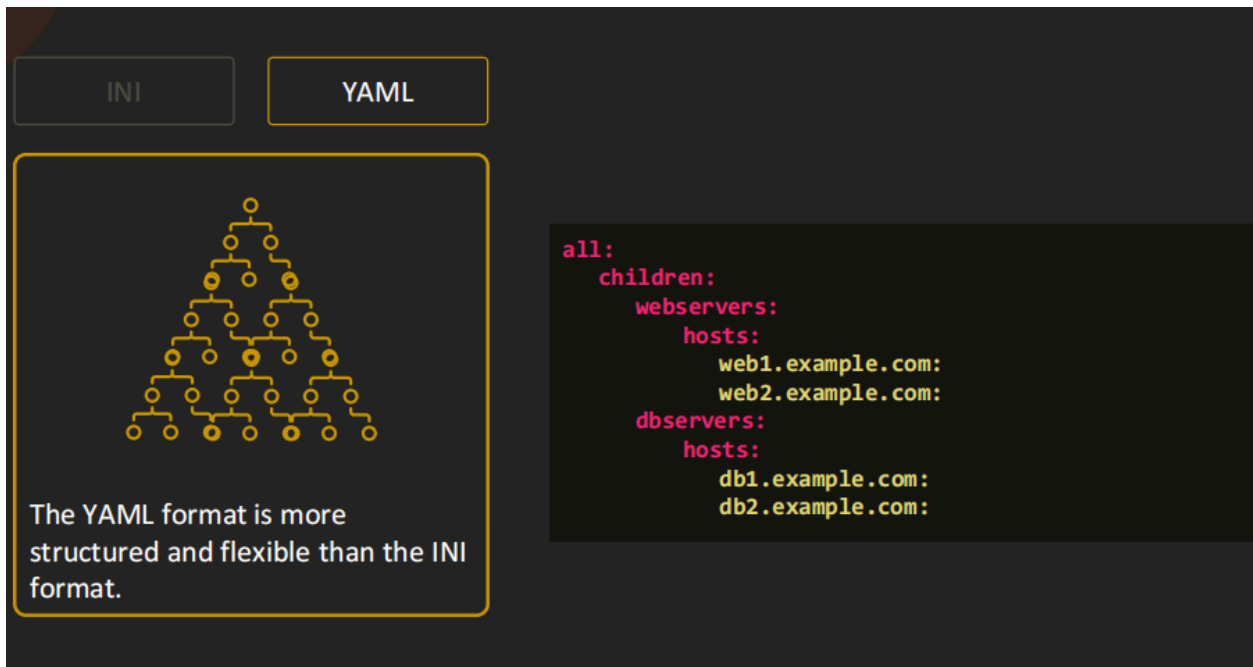
Example:

[Mail]
server1.company.com
server2.company.com

[db]
server3.company.com
server4.company.com

[web]
server5.company.com
server6.company.com

YAML:



```
all:
  children:
    webservers:
      hosts:
        web1.example.com:
        web2.example.com:
    dbservers:
      hosts:
        db1.example.com:
        db2.example.com:
```

The YAML format is more structured and flexible than the INI format.

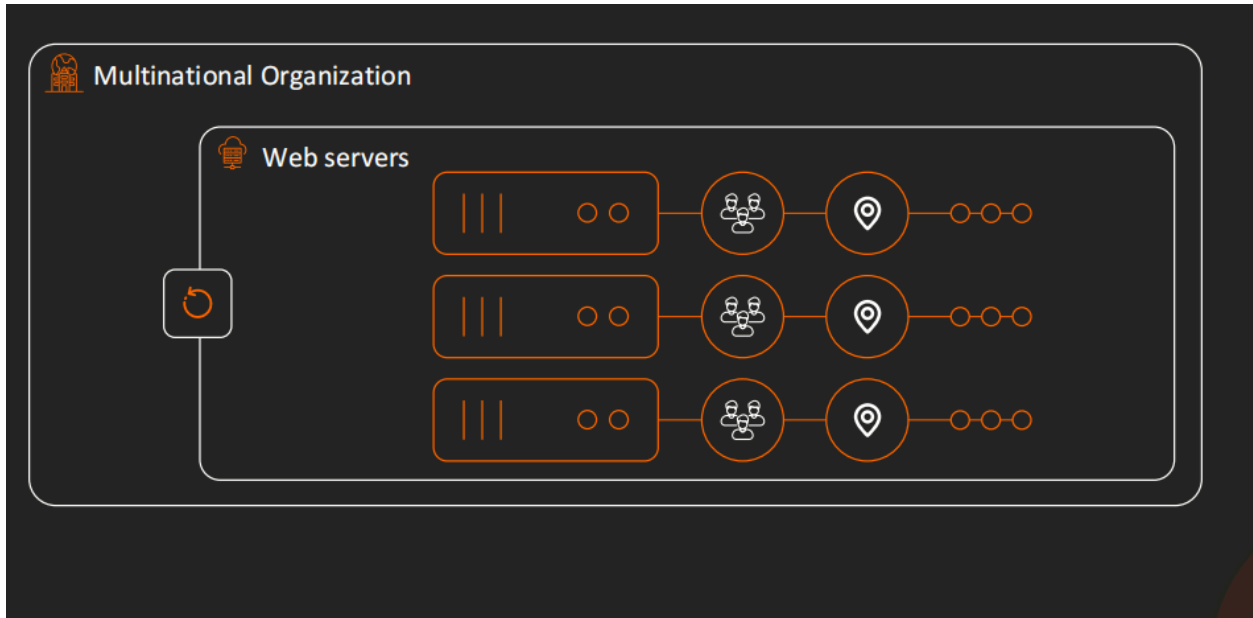
Uses in multinational companies like to maintain multiple tasks, supports multiple apps etc

Example:

All:

children:
 webserver:
 hosts:
 web1.example.com
 web2.example .com

Grouping:

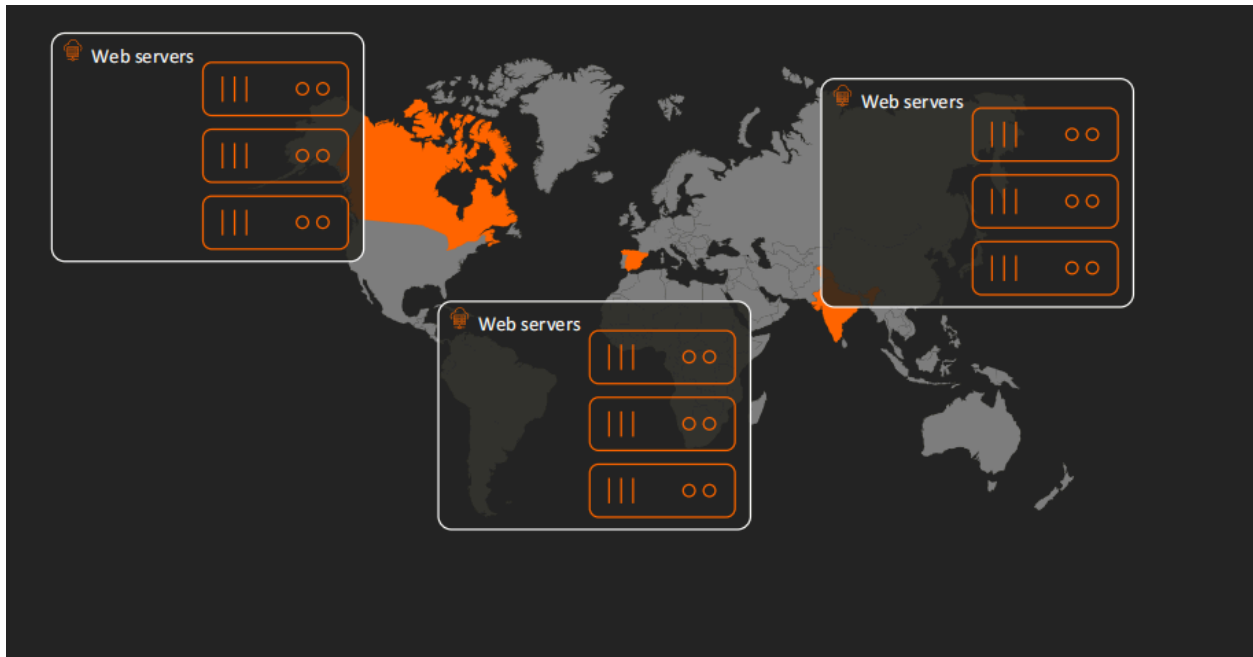


we categorize the servers based roles or locations or any other criteria is called grouping

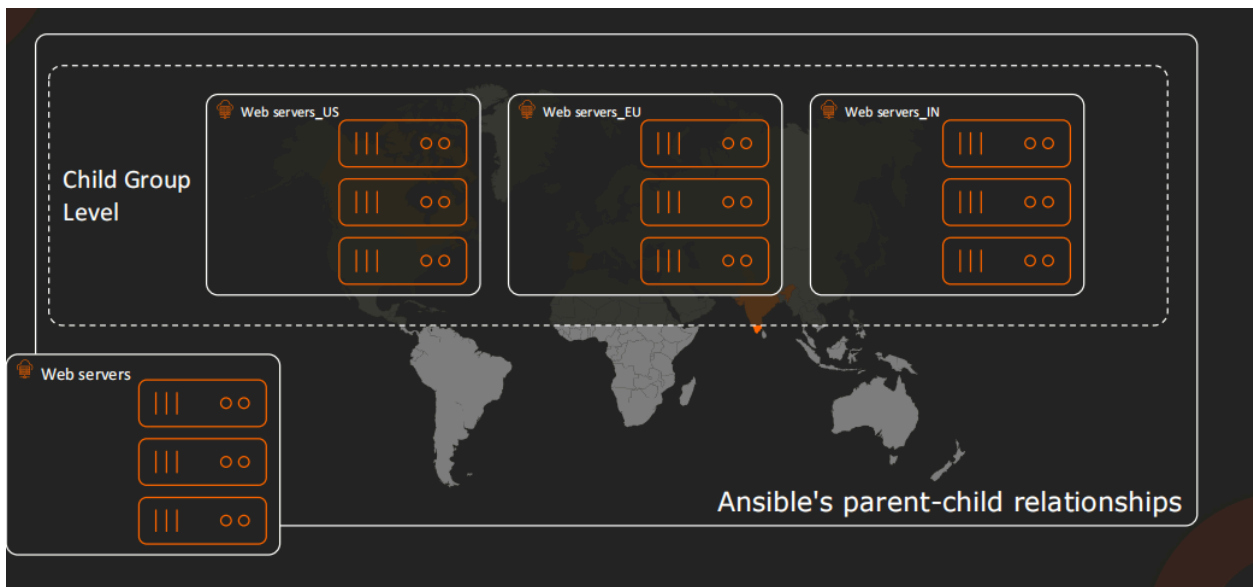
Collectively identify all the web servers under a common label named webservers
If we want to update the web servers , instead of mentioning each server we can mention/target the label name web servers then changes will apply to the all servers in that label this is called grouping this is done by ansible.

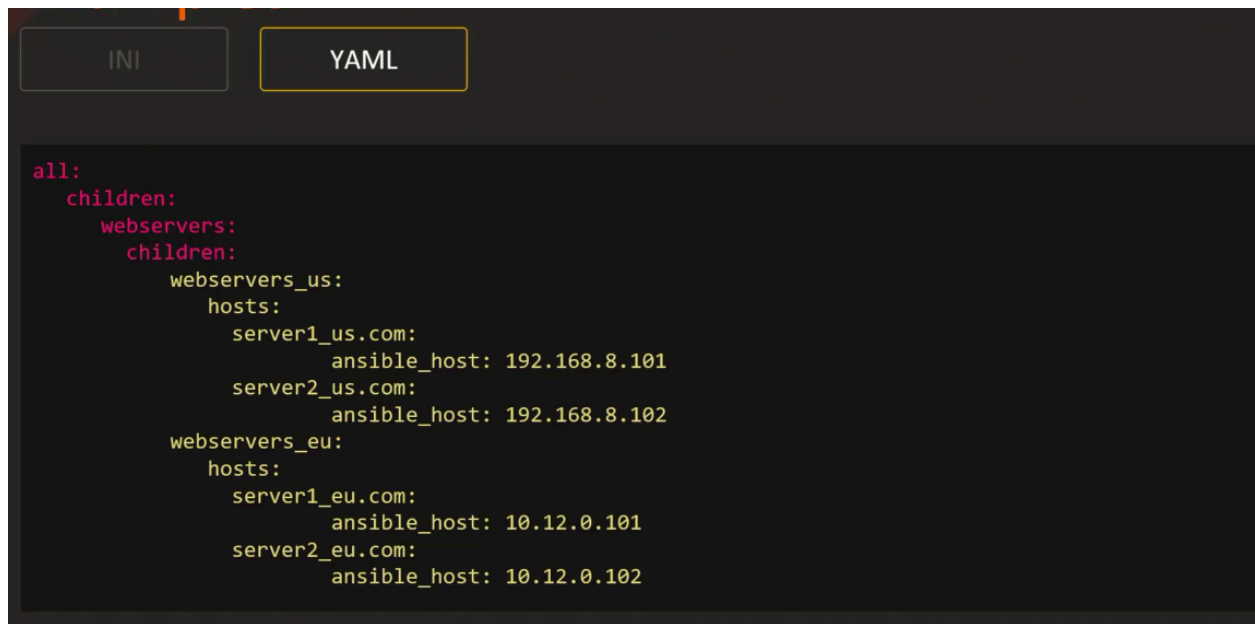
Parent -child relationship:

If we have multiple web servers in different location:



If we are having the web servers in different locations, then we create a web servers label as a parent label and listing web servers according to the location as a child of that parent label





The screenshot shows a code editor with two tabs: 'INI' and 'YAML'. The 'YAML' tab is active, displaying a YAML configuration. The structure is as follows:

```
all:
  children:
    webservers:
      children:
        webservers_us:
          hosts:
            server1_us.com:
              ansible_host: 192.168.8.101
            server2_us.com:
              ansible_host: 192.168.8.102
        webservers_eu:
          hosts:
            server1_eu.com:
              ansible_host: 10.12.0.101
            server2_eu.com:
              ansible_host: 10.12.0.102
```

Ansible variables:

Variable stores, hostnames, username, password info



The screenshot shows a code editor with two tabs: 'Playbook.yml' and 'variables'. The 'Playbook.yml' tab is active, displaying a playbook task. The 'variables' tab is also visible, showing two variables.

```
Playbook.yml
-
  name: Add DNS server to resolv.conf
  hosts: localhost
  tasks:
    dns_server: 10.1.250.10
      path: /etc/resolv.conf
      line: 'nameserver 10.1.250.10'
```

```
variables
variable1: value1
variable2: value2
```

We can add vars in playbook like:

Name: Add DNS server to resolv.config

hosts:.....

Vars :

dns_server=10.1.250.10

tasks:

.....

Or we can add another variables file separately and add variables into it:

```
variables
variable1: value1
variable2: value2
```

```
- name: Set Firewall Configurations
  hosts: web
  tasks:
  - firewallld:
      service: https
      permanent: true
      state: enabled

  - firewallld:
      port: '{{ http_port }}/tcp'
      permanent: true
      state: disabled

  - firewallld:
      port: '{{ snmp_port }}/udp'
      permanent: true
      state: disabled

  - firewallld:
      source: '{{ inter_ip_range }}/24'
      Zone: internal
      state: enabled
```

```
#Sample Inventory File

Web http_port=      snmp_port=      inter_ip_range=

#Sample variable File - web.yml

http_port: 8081
snmp_port: 161-162
inter_ip_range: 192.0.2.0
```

Jinja2 Templating

- ❌ source: {{ inter_ip_range }}
- ✅ source: '{{ inter_ip_range }}'
- ✅ source: Something{{ inter_ip_range }}Something

1. We can add the variables in the inventory file and can fetch it to our playbook
2. We can also create a variable file -web.yml and add all the variables and values to that variables into that file as shown in above picture

This format of using variables in play books is called jinja2 templating.

In jinja2 technique we use :

'{{variable_name}}' ⇒ correct

{{variable_nme}} ⇒ wrong

If we mention the variable in between the sentence:

{{variable_name}} ⇒ correct

Variable types:

1. String: sequence of chars
2. Number variables: integer, float
3. Boolean: true or false
4. list
5. Dictionary

Variable presidency:

What if variable defined in two different places like a group variable in inventory file and as host variable

Example: this is the **inventory file**

Web1 ansible_host=172.20.1.100

Web1 ansible_host=172.20.1.102 `dns_server=11.2.3.2 # declaring the dns server at the host`

Web1 ansible_host=172.20.1.103

[web_servers]

web1

web2

web3

`[web_servers:vars] #group variable`
`dns_server=10.5.5.3`

Defining Inside the Playbook:

—

- name: configuring dns
host: all
var:
 dns_server:10.5.5
tasks:
- nsupdate:
 Server: '{{dns_server}}'

Precedency:

1. Group vars
2. Host vars
3. Play level
4. Extra vars option

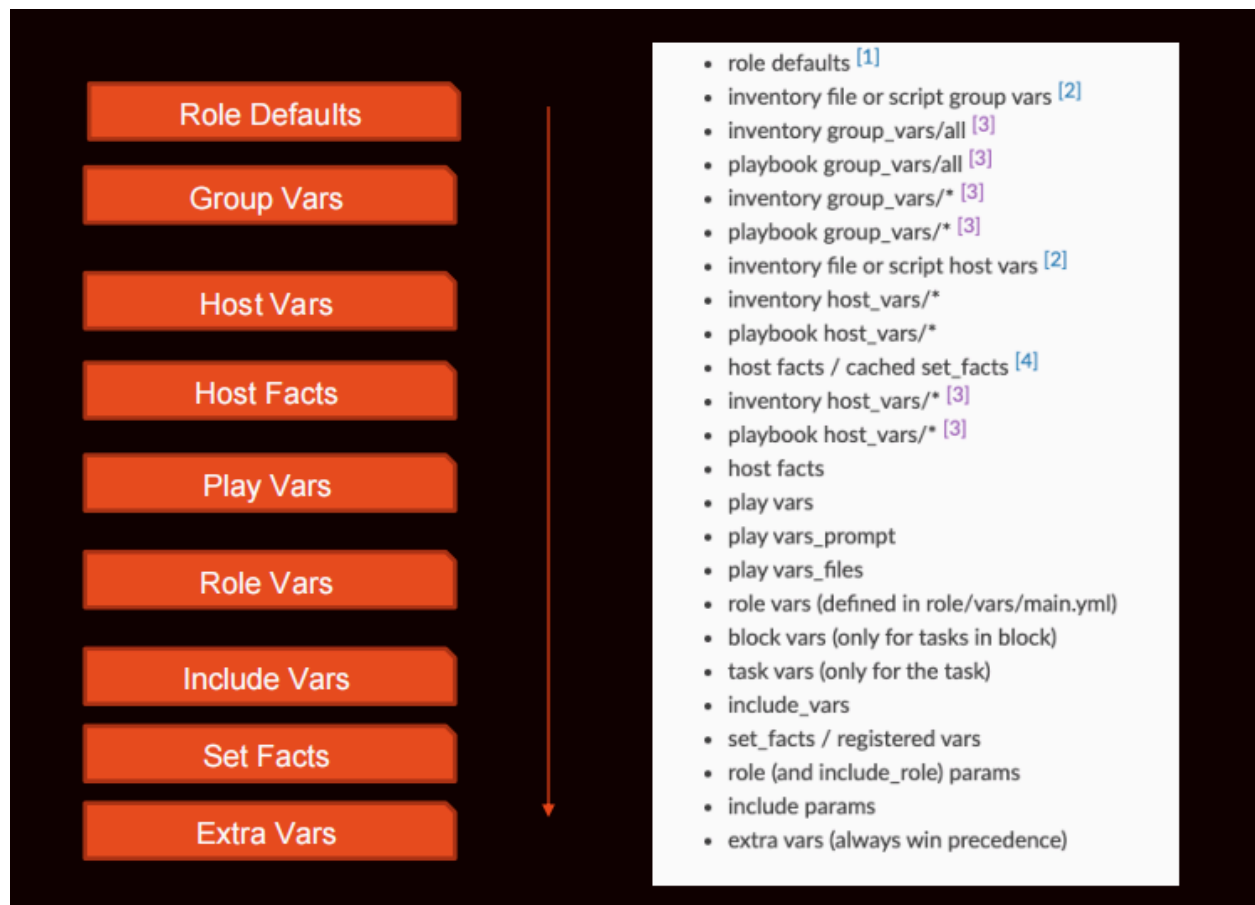
Group var has lowest precedence

Extra var has highest precedence

Priority increases from top to bottom, 1st it checks group var and take the value of group var if there is host var then that values is replaced with host var values and so on

Extra vars:

```
ansible-playbook playbook.yml --extra-vars "dns_server=10.5.5.6"
```



Register variable:

```
playbook
---
- name: Check /etc/hosts file
  hosts: all
  tasks:
    - shell: cat /etc/hosts

    register: result

  - debug:
    var:
      result

ok: [web2] => {
  "output": {
    "ansible_facts": {
      "discovered_interpreter_python": "/usr/bin/python3",
    },
    "changed": true,
    "cmd": "cat /etc/hosts",
    "failed": false,
    "rc": 0,
    "start": "2019-09-12 05:25:34.158877",
    "end": "2019-09-12 05:25:34.161974",
    "delta": "0:00:00.003097",
    "stderr": "",
    "stderr_lines": [],
    "stdout": "127.0.0.1:localhost\n::1:localhost\nfe00::0:subnet2\nff00::0:subnet3\nff02::1:subnet2\nff02::2:subnet3\n172.20.1.1:web2\n",
  },
}

playbook
---
- name: Check /etc/hosts file
  hosts: all
  tasks:
    - shell: cat /etc/hosts

    register: result

  - debug:
    var:
      result
```

By using a register variable we can store the output and as a result and we can use that result later.