
Infrastructure as Code

Configuration Management

Terraform
by HashiCorp

Configuration management tools **install** and **manage** software on a machine that already exists.

**Terraform is not a configuration management tool**, and it allows existing tooling to focus on their strengths: **bootstrapping and initializing resources**

## Infrastructure as Code

- Terraform, CloudFormation
- Automate and manage all infrastructure components
- Responsible for providing network and servers

## Configuration Management

- Ansible, Puppet, Chef
- Configuring provisioned infrastructure resources
- Configures applications inside servers provisioned by IaC.

- Both follow the concept of Idempotency

## LIST

| Process |
| --- |
| Grinding |
| Pressing |
| Boiling |
| Filtering |
| Mixing |

## JSON – KEY – VALUE

**Raw Materials**

```
{
  "Process":[
    "Grinding",
    "Pressing",
    "Boiling",
    "Filtering",
    "Mixing"
  ]
}
```

[] - LIST

## YAML

**Raw Materials**

```
Process:
- Grinding
- Pressing
- Boiling
- Filtering
- Mixing
```

LIST - HYPHEN

## COMBINATION OF BOTH

| Make Coffee Pseudo Steps | | |
|---|---|---|
| Ingredients | Coffee<br>Flavour Liquid<br>Milk<br>Water<br>Sugar | |
| Process | Grinding<br>Pressing<br>Boiling<br>Filtering<br>Mixing | |
| Grinding | Coffee | Peets |
| | Texture | Coarse |
| Boiling | Temp | 92 degree C |
| | Flavour | Hazelnut |
| | Time | 5 Mins |
| Mixing | Milk | True |
| | Sugar | True |
| | No_of_Cups | 1 |

## JSON – KEY – VALUE

**Raw Materials**

```
{
 "Ingredients": [
  "Coffee",
  "Flavour Liquid",
  "Milk",
  "Water",
  "Sugar"
 ],
 "Process": [
  "Grinding",
  "Pressing",
  "Boiling",
  "Filtering",
  "Mixing"
 ],
 "Grinding": {
  "Coffee": "Peets",
  "Texture": "Coarse"
 },
 "Mixing": {
  "Milk": true,
  "Sugar": false,
  "No_of_Cups": 1
 }
}
```

LIST

LIST

DICT

DICT

## YAML – KEY – VALUE

**Raw Materials**

```
Ingredients:
- Coffee
- Flavour Liquid
- Milk
- Water
- Sugar

Process:
- Grinding
- Pressing
- Boiling
- Filtering
- Mixing

Grinding:
  Coffee: Peets
  Texture: Coarse

Mixing:
  Milk: 'True'
  Sugar: 'True'
  No_of_Cups: 1
```

# What is an Ansible Module?

Deploy 50 Servers

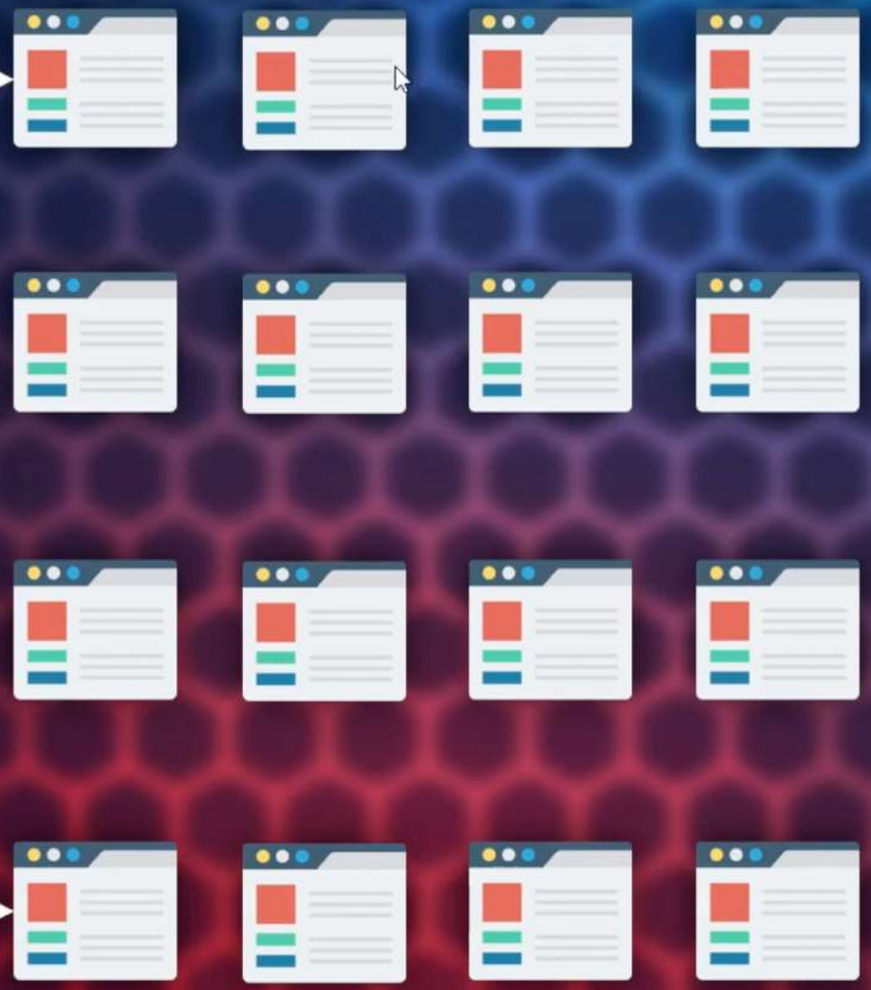
- HOST 1 : 10.23.12.11
- HOST 1 : 10.23.12.12
- HOST 1 : 10.23.12.13
- HOST 1 : 10.23.12.14
- HOST 1 : 10.23.12.15
- HOST 1 : 10.23.12.16

Install_playbook.yml

- Stop Monitoring
- Remove From Load Balancer
- Stop Services
- Deploy Application
- Check Service Status

Ansible Playbook

Same steps can be replicated over several servers with the same configurations



# What is an Ansible Module?

Modules (also referred to as "task plugins" or "library plugins") are discrete units of code that can be used from the command line or in a pl

Ansible:

- Executes each module on the remote target node.
- Collects return values.

```
ansible <pattern_goes_here> -m <module_name> -a <arguments>
```

A

```
ansible webservers -m service -a "name=httpd state=restarted"
```

A pattern usually refers to a set of groups (which are sets of hosts) – in the above case, machines in the "webservers" group.

B

```
ansible webservers -m ping
```

C

```
ansible webservers -m command -a "/sbin/reboot -t now"
```

ARGUMENTS:

- Most modules take key=value arguments, space delimited.
- Some modules take no arguments
- Command/shell modules simply take the string of the command you want to run.

## Commands modules

- **COMMAND**
  The command module takes the command name followed by a list of space-delimited arguments.
  The given command will be executed on all selected nodes.
- **EXPECT**
  The expect module executes a command and responds to prompts.
  The given command will be executed on all selected nodes.
- **PSEXEC**
  Runs a remote command from a Linux host to a Windows host without WinRM being set up.
  Can be run on the Ansible controller to bootstrap Windows hosts to get them ready for WinRM
- **RAW**
  Executes a low-down and dirty SSH command, not going through the module subsystem
- **SCRIPT**
  The local script at path will be transferred to the remote node and then executed.
  The given script will be processed through the shell environment on the remote node.
- **SHELL**
  It is almost exactly like the command module but runs the command through a shell (/bin/sh) on the remote node.
  For Windows targets, use the win_shell module instead
- **TELNET**
  Executes a low-down and dirty telnet command, not going through the module subsystem.
  This is mostly to be used for enabling ssh on devices that only have telnet enabled by default.

# Commands modules

## Execution Playbook

```
# Registering Variable from output
- name: return motd to registered var
 command: cat /etc/motd
 register: mymotd


# 'cmd' is module parameter
- name: Run command
 command:
   cmd: /usr/bin/make_database.sh
db_user db_name
   creates: /path/to/database
```

## Command Parameters

| Parameter | Choices/Defaults | Comments |
|---|---|---|
| argv list<br>added in 2.6 | | Passes the command as a list rather than a string.<br>Use argv to avoid quoting values that would otherwise be interpreted incorrectly (for example "user name").<br>Only the string or the list form can be provided, not both. One or the other must be provided. |
| chdir path | | Change into this directory before running the command. |
| cmd string | | The command to run. |
| creates path | | A filename or (since 2.0) glob pattern. If it already exists, this step **won't** be run. |
| free_form - | | The command module takes a free form command to run.<br>There is no actual parameter named 'free form'. |
| removes path | | A filename or (since 2.0) glob pattern. If it already exists, this step **will** be run. |
| stdin -<br>added in 2.4 | | Set the stdin of the command directly to the specified value. |
| stdin_add_newline boolean<br>added in 2.8 | •Choices:no<br>•yes ← | If set to yes, append a newline to stdin data. |
| strip_empty_ends boolean<br>added in 2.8 | •Choices:no<br>•yes ← | Strip empty lines from the end of stdout/stderr in result. |
| warn boolean | •Choices:no<br>•yes ← | Enable or disable task warnings. |

## Return Values

| Key | Returned | Description |
|---|---|---|
| **cmd** list | always | the cmd that was run on the remote machine<br>**Sample:**<br>['echo', 'hello'] |
| **delta** string | always | cmd end time - cmd start time<br>**Sample:**<br>0.001529 |
| **end** string | always | cmd end time<br>**Sample:**<br>2017-09-29 22:03:48.084657 |
| **start** string | always | cmd start time<br>**Sample:**<br>2017-09-29 22:03:48.083128 |

# EXPECT

The expect module executes a command and responds to prompts.

- python >= 2.6
- pexpect >= 3.3

The ansible-base code runs on both Python 2 and Python 3 because we want Ansible to be able to manage a wide variety of machines.

## Command Parameters

| Parameter | Choices/Defaults |
|---|---|
| **chdir** path | |
| **command** - / required | |
| **creates** path | |
| **echo** boolean | •**Choices:no** ← •yes |
| **removes** path | |
| **responses** dictionary / required | |
| **timeout** integer | **Default:** 30 |

## Execution Playbook

```
- name: Case insensitive password string match
  expect:
    command: passwd username
    responses:
      (?i)password: "MySekretPa$$word"
  # you don't want to show passwords in your logs
  no_log: true

- name: Generic question with multiple different responses
  expect:
    command: /path/to/custom/command
    responses:
      Question:
        - response1
        - response2
        - response3
```

Commands modules

# PSEXEC

Runs a remote command from a Linux host to a Windows host without WinRM being set up.

- pypsexec
- smbprotocol[kerberos] for optional Kerberos authentication

## Execution Playbook

```
- name: Run a PowerShell command
  psexec:
    hostname: server.domain.local
    connection_username: username@DOMAIN.LOCAL
    connection_password: password
    executable: powershell.exe
    arguments: Write-Host Hello World
    asynchronous: yes
```

## Command Parameters

| Parameter | Choices/Defaults | Comments |
|---|---|---|
| **arguments** string | | Any arguments as a single string to use when running the executable. |
| **asynchronous** boolean | •**Choices:no** ←<br>•yes | Will run the command as a detached process and the module returns immediately after starting the process while the process continues to run in the background.<br>The stdout and stderr return values will be null when this is set to yes.<br>The stdin option does not work with this type of process.<br>The rc return value is not set when this is yes |
| **connection_password** string | | The password for connection_user.<br>Required if the Kerberos requirements are not installed or the username is a local account to the Windows host.<br>Can be omitted to use a Kerberos principal ticket for the principal set by connection_user if the Kerberos library is installed and the ticket has already been retrieved with the kinit command before. |
| **connection_timeout** integer | **Default:**<br>60 | The timeout in seconds to wait when receiving the initial SMB negotiate response from the server. |
| **connection_username** string | | The username to use when connecting to the remote Windows host.<br>This user must be a member of the Administrators group of the Windows host.<br>Required if the Kerberos requirements are not installed or the username is a local account to the Windows host.<br>Can be omitted to use the default Kerberos principal ticket in the local credential cache if the Kerberos library is installed.<br>If process_username is not specified, then the remote process will run under a Network Logon under this account. |
| **encrypt** boolean | •**Choices:**no<br>•**yes** ← | Will use SMB encryption to encrypt the SMB messages sent to and from the host.<br>This requires the SMB 3 protocol which is only supported from Windows Server 2012 or Windows 8, older versions like Windows 7 or Windows Server 2008 (R2) must set this to no and use no encryption.<br>When setting to no, the packets are in plaintext and can be seen by anyone sniffing the network, any process options are included in this. |
| **executable** string / required | | The executable to run on the Windows host. |
| **hostname** string / required | | The remote Windows host to connect to, can be either an IP address or a hostname. |

Commands modules

# RAW

Executes a low-down and dirty SSH command, not going through the module subsystem.
Ex: Installing python on a system without python installed by default.

- Arguments given to raw are run directly through the configured remote shell.
- This module does not require python on the remote system, much like the script module.
- This module is also supported for Windows targets.

## Command Parameters

| Parameter | Choices/Defaults |
|---|---|
| executable | |
| free_form - / required | |

## Execution Playbook

```
- name: Bootstrap a host without python2 installed
  raw: dnf install -y python2 python2-dnf libselinux-python

- name: Run a command that uses non-posix shell-isms
  raw: cat < /tmp/*txt
  args:
    executable: /bin/bash
```

Commands modules

# SCRIPT

**Runs a local script on a remote node after transferring it**

- The script module takes the script name followed by a list of space-delimited arguments.
- The local script at path will be transferred to the remote node and then executed.
- The given script will be processed through the shell environment on the remote node.
- This module is also supported for Windows targets

## Command Parameters

| Parameter | Choices/Defaults | Comments |
|---|---|---|
| chdir - added in 2.4 | | Change into this directory on the remote node before running the script. |
| cmd string | | Path to the local script to run followed by optional arguments. |
| creates - | | A filename on the remote node, when it already exists, this step will not be run. |
| decrypt boolean added in 2.4 | •Choices:no •yes ← | This option controls the autodecryption of source files using vault. |
| executable - added in 2.6 | | Name or path of a executable to invoke the script with. |
| free_form - | | Path to the local script file followed by optional arguments. |
| removes - | | A filename on the remote node, when it does not exist, this step will not be run. |

## Execution Playbook

```
- name: Run a script with arguments (free form)
  script: /some/local/script.sh --some-argument 1234

- name: Run a script with arguments (using 'cmd' parameter)
  script:
    cmd: /some/local/script.sh --some-argument 1234

- name: Run a script only if file.txt does not exist on the remote node
  script: /some/local/create_file.sh --some-argument 1234
  args:
    creates: /the/created/file.txt
```

Commands modules

# TELNET

Executes a low-down and dirty telnet command, not going through the module subsystem.

- This is mostly to be used for enabling ssh on devices that only have telnet enabled by default

## Execution Playbook

```
- name: send configuration commands to IOS
  telnet:
    user: cisco
    password: cisco
    login_prompt: "Username: "
    prompts:
      - "[>#]"
    command:
      - terminal length 0
      - configure terminal
      - hostname ios01
```

## Command Parameters

| Parameter | Choices/Defaults | Comments |
|---|---|---|
| command - / required | | List of commands to be executed in the telnet session. aliases: commands |
| host - | Default: "remote_addr" | The host/target on which to execute the command |
| login_prompt - | Default: "login: " | Login or username prompt to expect |
| password - | | The password for login |
| password_prompt - | Default: "Password: " | Login or username prompt to expect |
| pause - | Default: 1 | Seconds to pause between each command issued |
| port - | Default: 23 | Remote port to use |
| prompts - | Default: ["$"] | List of prompts expected before sending next command |
| send_newline boolean added in 2.7 | •Choices:no ← •yes | Sends a newline character upon successful connection to start the terminal session. |
| timeout - | Default: 120 | timeout for remote operations |
| user - | Default: "remote_user" | The user for login |

# ANSIBLE MODULES

- Cloud modules
- Crypto modules
- Identity modules
- Monitoring modules
- Notification modules
- Source Control modules
- Utilities modules
- Clustering modules
- Database modules
- Inventory modules
- Net Tools modules
- Packaging modules
- Storage modules
- Web Infrastructure modules
- Commands modules
- Files modules
- Messaging modules
- Network modules
- Remote Management modules
- System modules
- Windows modules

## Cloud modules

```yaml
# Basic provisioning example
- ec2:
  key_name: mykey
  instance_type: t2.micro
  image: ami-123456
  wait: yes
  group: webserver
  count: 3
  vpc_subnet_id: subnet-29e63245
  assign_public_ip: yes

# Advanced example with tagging and CloudWatch
- ec2:
  key_name: mykey
  group: databases
  instance_type: t2.micro
  image: ami-123456
  wait: yes
  wait_timeout: 500
  count: 5
  instance_tags:
    db: postgres
  monitoring: yes
  vpc_subnet_id: subnet-29e63245
  assign_public_ip: yes
```

## Clustering modules

```yaml
- name: Create a k8s namespace
  k8s:
    name: testing
    api_version: v1
    kind: Namespace
    state: present

- name: Create a Service object from an inline definition
  k8s:
    state: present
    definition:
      apiVersion: v1
      kind: Service
      metadata:
        name: web
        namespace: testing
        labels:
          app: galaxy
          service: web
      spec:
        selector:
          app: galaxy
          service: web
        ports:
        - protocol: TCP
          targetPort: 8000
          name: port-8000-tcp
          port: 8000
```

Hosts
SFTP Premium
Port Forwarding
Ansible-Master
Ansible-Host-1
History
Ansible-Master

```yaml
# Install Nginx using Ansible
# Start and Enable Nginx
# Check the status of Nginx
- name: Setup Nginx server on myserver list
  hosts: myservers
  become: True
  vars:
    nginx_version: 1.12
    base_path: /home/ec2-user
  tasks:
    - name: Install the latest version of nginx
      command: amazon-linux-extras install nginx{{ nginx_version }}=latest -y
      args:
        creates: /sbin/nginx

    - name: Start nginx
      service:
        name: nginx
        state: started

    - name: Enable nginx
      service:
        name: nginx
        enabled: yes

    - name: Ensure nginx is at the latest version
      command: nginx -v
      register: nginx_version

    - name: Print the version of nginx
      debug:
        msg: "The current version of nginx is {{ nginx_version.stderr_lines[0] }}"

    - name: Get status of nginx installed
      command: systemctl status nginx

    - name: Ansible copy file to remote server
      copy:
        src: "{{ base_path }}sample.txt"
        dest: "{{ base_path }}"
```

INVENTORY FILE PATH : /home/ec2-user/inventory.txt

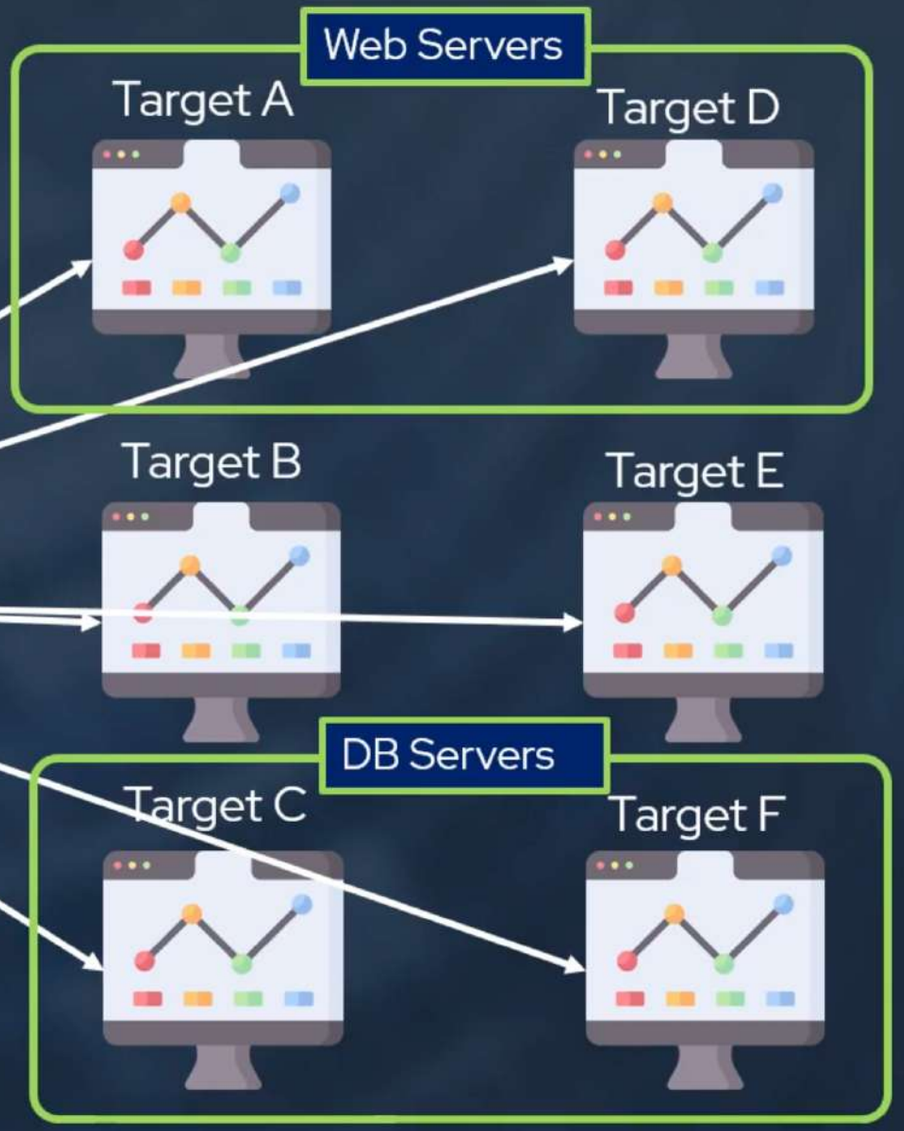
```
[webservers]
targeta.hello.com
targetd.hello.com

[dbservers]
targetc.hello.com
targetf.hello.com
```

```
[common]
targeta.hello.com
targetd.hello.com
targetc.hello.com
targetf.hello.com
```

Developer

Ansible Master




RUN WEB SERVER : ansible webservers –m ping -i inventory.txt

RUN DB SERVER : ansible dbservers –m ping -i inventory.txt

RUN COMMON : ansible common –m ping -i inventory.txt

INVENTORY FILE PATH : /etc/ansible/hosts/inventory.yml

| SLNO | HOSTNAME | IP ADDRESS | USERNAME | CONNECTION TYPE | PASSWORD/SSHKEY | SERVICE |
|---|---|---|---|---|---|---|
| 1 | Target1 | 12.23.22.1 | Ec2-user | SSH | Ec2-key.pem | WEB SERVICE |
| 2 | Target2 | 12.23.22.2 | Ec2-user | SSH | Ec2-key.pem | WEB SERVICE |
| 3 | Target3 | 12.23.22.3 | Ec2-user | PASSWORD | ******** | DB SERVICE |
| 4 | Target4 | 12.23.22.4 | Ec2-user | PASSWORD | ******** | DB SERVICE |
| 5 | Target5 | 12.23.22.5 | Ec2-user | SSH | Ec2-key.pem | WEB SERVICE |

# GROUP AND HOST VARIABLES

Assigning a variable to one machine: host variables

**VARIABLES**

```
[mywebservers]
ap1.pythoholic.com
ap2.Pythoholic.com
eu1.Pythoholic.com
eu2.Pythoholic.com

[mywebservers:vars]
port_manager=3366
proxy=proxy.pythoholic.com
base_url=/etc/ansible/hosts
```

**HOST VARIABLES**

```
[mywebservers]
ap1.pythoholic.com   port_manager=3366  proxy=proxy.pythoholic.com  base_url=/etc/ansible/hosts
ap1.pythoholic.com   port_manager=3226  proxy=proxy.pythoholic.com  base_url=/etc/ansible/hosts
eu1.pythoholic.com   port_manager=3566  proxy=proxy.pythoholic.com  base_url=/etc/ansible/hosts
eu2.pythoholic.com   port_manager=3366  proxy=proxy.pythoholic.com  base_url=/etc/ansible/hosts
```

## HOW TO WORK WITH VARIABLES IN ANSIBLE

Ansible uses variables to manage differences between systems. With Ansible, you can execute tasks and playbooks on multiple different systems with a single command.
You can define these variables in your playbooks, in your inventory, in re-usable files or roles, or at the command line.

| Valid variable names | Not valid |
|---|---|
| foo | *foo, Python keywords such as async and lambda |
| foo_env | playbook keywords such as environment |
| foo_port | foo-port, foo port, foo.port |
| foo5, _foo | 5foo, 12 |

**How to create VALID variable names**

- A variable name can only include letters, numbers, and underscores.
- Python keywords or playbook keywords are not valid variable names.
- A variable name cannot begin with a number.
- Variable names can begin with an underscore.

# DEFINING VARIABLES AT RUNTIME

1. Passing variables at the command line using the --extra-vars (or -e) argument.
2. Request user input with a vars_prompt

## KEY=VALUE Format

```
ansible-playbook install_nginx.yml --extra-vars "version=1.12.1 build=latest"
```

## JSON String Format

```
ansible-playbook install_nginx.yml --extra-vars '{"version":"1.12.1","build":"latest"}'
```

## JSON or YAML file as variable format

```
ansible-playbook install_nginx.yml --extra-vars "@my_input_file.json"'
```

# GROUP AND HOST VARIABLES

Assigning a variable to one machine: host variables

Assigning a variable to many machines: group variables

If all hosts in a group share a variable value, you can apply that variable to an entire group at once.

## GROUP VARIABLES

```
[mywebservers]
ap1.pythoholic.com
ap2.Pythoholic.com
eu1.Pythoholic.com
eu2.Pythoholic.com

[mywebservers:vars]
port_manager=3366
proxy=proxy.pythoholic.com
base_url=/etc/ansible/hosts
```

## INHERITING GROUP VARIABLES

```
[ asia ]
ap1.pythoholic.com
ap2.Pythoholic.com

[europe]
eu1.Pythoholic.com
eu2.Pythoholic.com

[emea_region:children]
asia
europe

[emea_region:vars]
port_manager=3366
proxy=proxy.pythoholic.com
base_url=/etc/ansible/hosts
```

```
[region:children]
emea_region
pacific_region
aus_region
```

## CREATING YOUR VARIABLE FILE

You can define variables in reusable variables files and/or in reusable roles.

**A** You can create your own variable file.

```
// my_variable.yml which is inside /vars folder

ap_south_1: John
us_west_2: Jesse
us_east_1:  David
food:
 - breakfast
 - lunch
 - dinner
```

```
// This is your playbook

- hosts: web_servers
  vars:
    myvar: hello_world
  vars_file:
    - /vars/my_variable.yml
  tasks:
  - name: Run a shell command and fetch the nginx version
    shell: nginx -v
    register: nginx_version
```

## REGISTERING ANSIBLE VARIABLES

**A** You can create variables from the output of an Ansible task with the task keyword register

```
- hosts: web_servers
  tasks:
        - name: Run a shell command and fetch the nginx version
        shell: nginx -v
        register: nginx_version

        - name: Print the version of Nginx
        debug:
            msg: "The current version of ansible is {{ nginx_version }}
```

**B** {{ aws_regions["asia"]["ap-south-1"]["mumbai"] }}

```
{{ aws_region.asia.ap-south-1.mumbai }}
```

# DEFINING THE ANSIBLE VARIABLE

Define variables with multiple values using YAML lists.

## A Using a list of values in the variable

```
aws_region:
 - ap-south-1
 - us-west-2
 - us-east-1
```

```
region: "{{ aws_region[0] }}"

Value : ap-south-1
```

## B Using a key value pair as variables

```
aws_region:
   ap-south-1 : John
   us-west-2 : Jesse
   us-east-1 : David
```

```
aws_region['ap-south-1']

aws_region.us-west-2
```

***By using the Square brackets***

***By using the dot operator***

# DEFINING THE ANSIBLE VARIABLE

Simple variables combine a variable name with a single value

## A install_path: /home/ec2-user/hosts

```
- name: Copy file to a location on the Server (Target)
  copy:
    src: /home/ec2-user/sample.txt
    dest: '{{ install_path }}/inventory.cfg'
```

***{{ install_path }} Jinja2 Template Expression***

## B install_path: /home/ec2-user/hosts

```
- hosts: app_servers
  vars:
    app_path: {{ install_path }}/hosts
```

***ERROR! Syntax Error while loading YAML.***

```
- hosts: app_servers
  vars:
    app_path: "{{install_path }}/22"
```

***Using Double Quote – Works Just fine !!!***