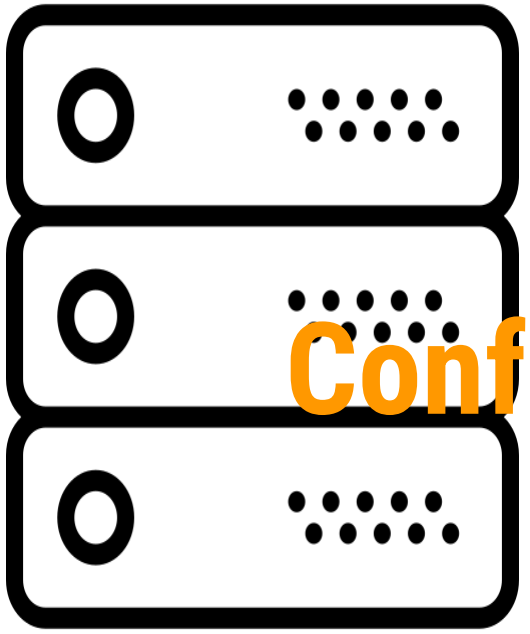
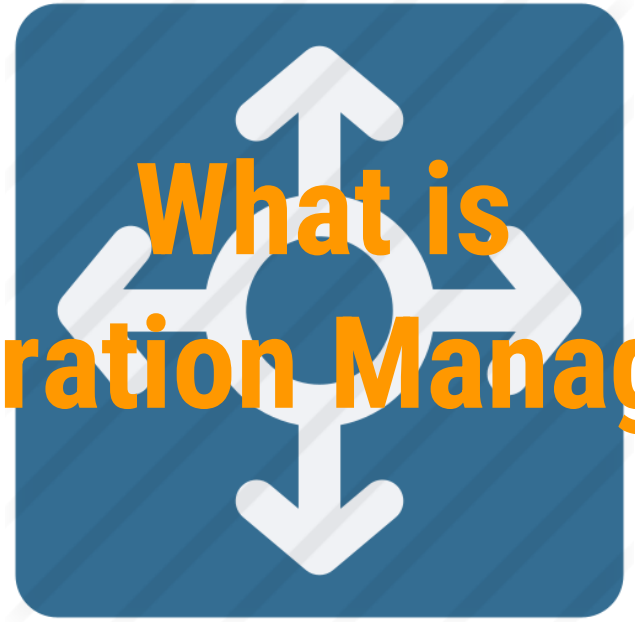
Ansible Advanced

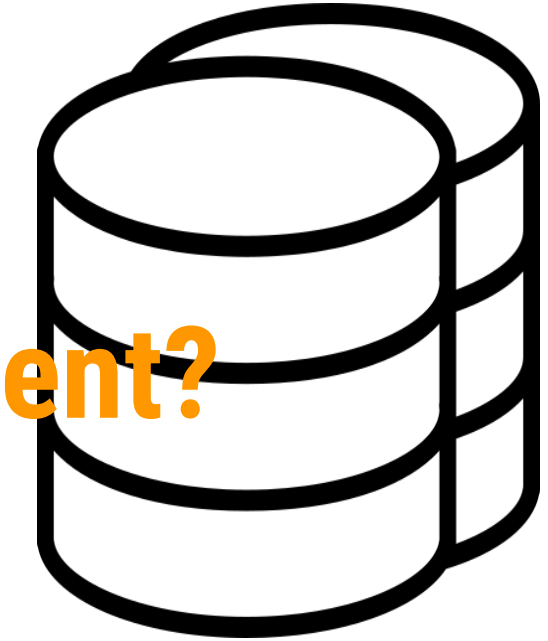
Traditional Datacenter



Servers



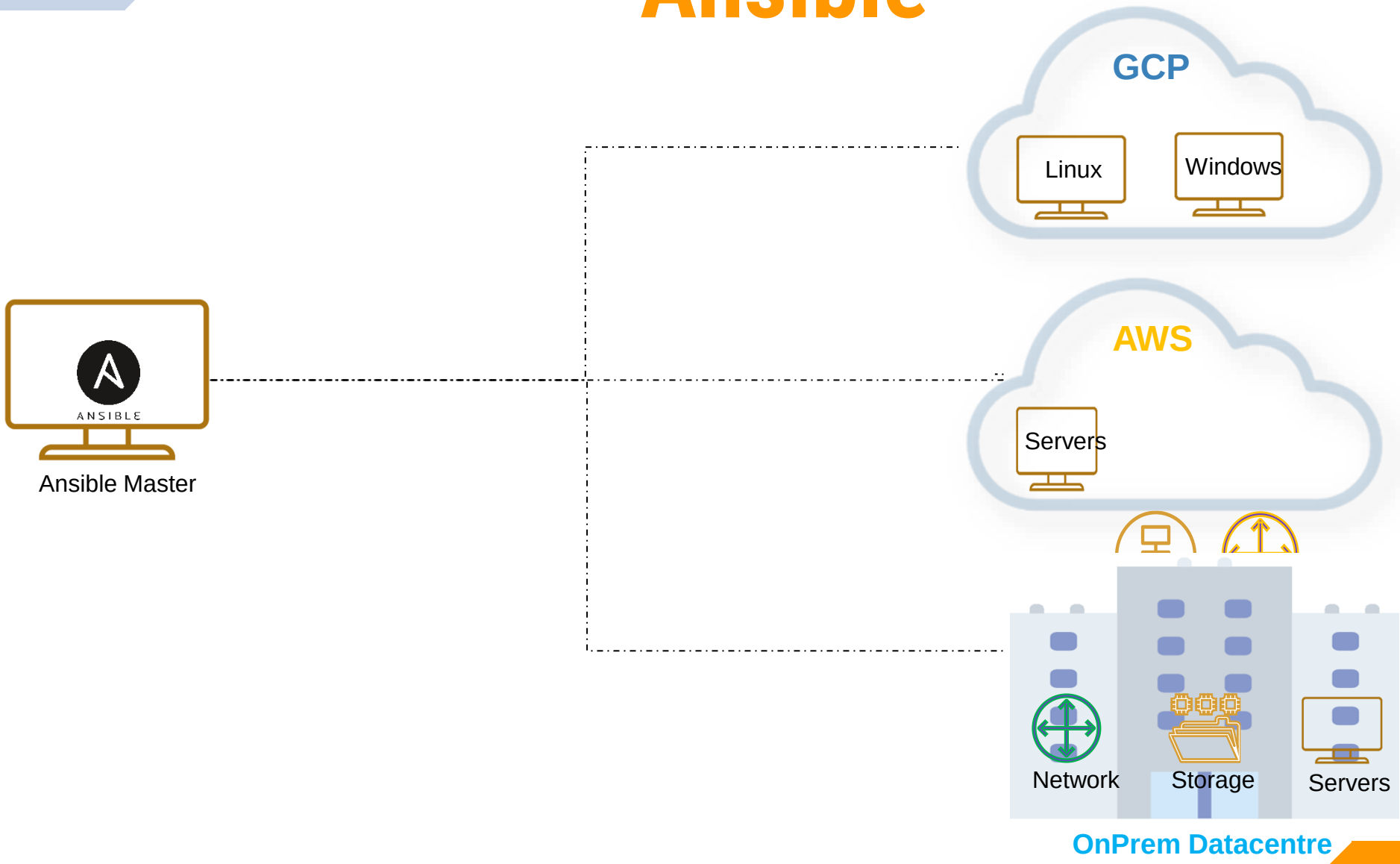
Network



Storage

What is Configuration Management?

Ansible

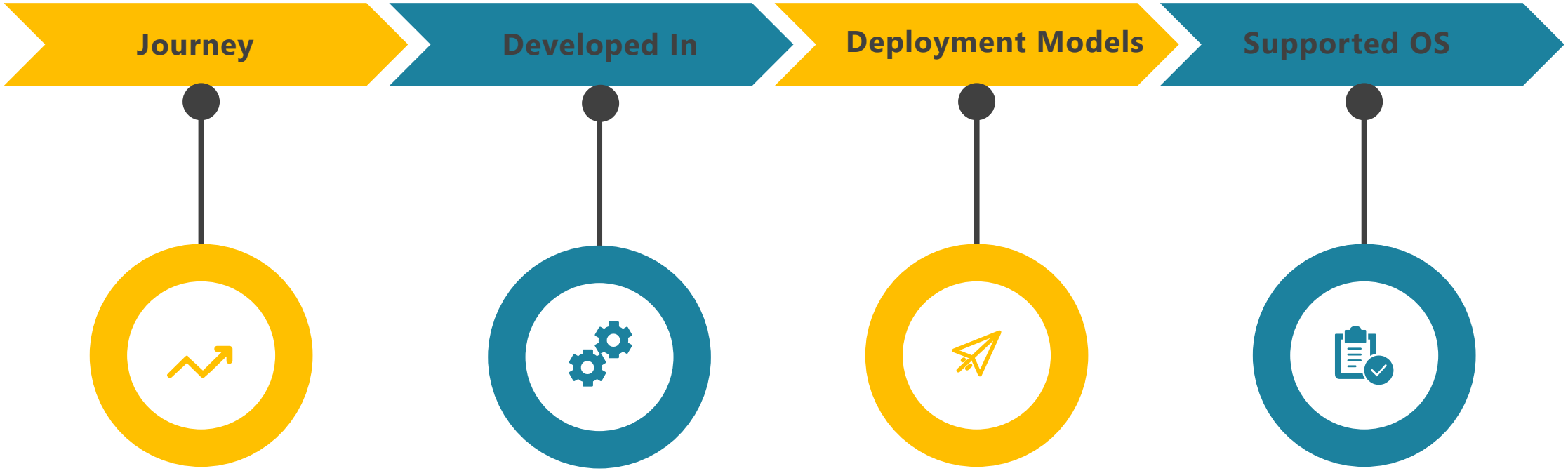


Ansible

Ansible is an easy-to-use IT Automation, Configuration Management & Orchestration Software for System Administrators & DevOps Engineers.

- Founded in Feb, 2012
- First commercial product release in 2012
- Multiple in-built functional modules
- Multiple Community Members
- 40,000+ Users
- 50,000+ Nodes managed in the largest deployments
- Support for Red Hat, CentOS, Ubuntu, Oracle Linux, MAC, OS, Solaris 10/11, Windows.
- Ansible Controller node Supported on Linux variants only

Ansible Introduction



Michael DeHaan – Feb 2012

Redhat Acquisition – Oct 2015

Python

Require Python as pre-requisite

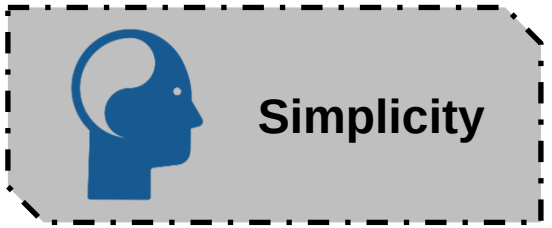
Push Based Architecture

Agentless

Master Node – Linux

Managed Node – Unix, Linux and Windows

Why Ansible?



Current IT Automation State

Manually Configure: Literally logging into every node to configure it.

Golden Images: Creating a single copy of a node's software and replicating that across nodes.

Custom One-off Scripts: Custom code written to address a specific, tactical problem.

Software Packages: Typically all or nothing approach.

Current IT Automation State

- **Manually Configure:**
  - Difficult to scale.
  - Impossible, for all intents and purposes, to maintain consistency from node-to-node.

Current IT Automation State

- **Golden Images:**
  - Need separate images for different deployment environments, e.g. development, QA, production, or different geo locations.
  - As number of images multiply it becomes very difficult to keep track and keep consistent.
  - Since they're monolithic copies, golden images are rigid and thus difficult to update as the business needs change.

Current IT Automation State

- **Custom One-off Scripts:**
  - No leverage – effort typically cannot be reused for different applications or deployments.
  - Brittle – as needs change, often the entire script must be re-written.
  - Difficult to maintain when the original author leaves the organization.

Current IT Automation State

- **Software Packages:**
  - These packages typically require that all resources be placed under management – cannot selectively adopt and scale automation.
  - As a result, longer deployments times.
  - Dated technology developed before virtualization and cloud computing – lacks responsiveness to changing requirements.

Why Configuration Management?

- To provide optimized level of automated way to configure Applications and Software's inside your system.
- Enable you to Discover, Provision, Configure and Manage the systems.
- Developers should be able to use a single command to build and test software in minutes or even in seconds.
- Maintaining configuration state of all systems simultaneously should be easy.
- Login into every client machine for CM tasks should not be mandate.
- It should be easy to maintain desired state as per policy

Why Orchestration with Ansible?

- A single tool for deployment and Configuration management
- Easy to manage and use
- Compatible with all major cloud service providers
- Can Orchestrate Infrastructure and Software both

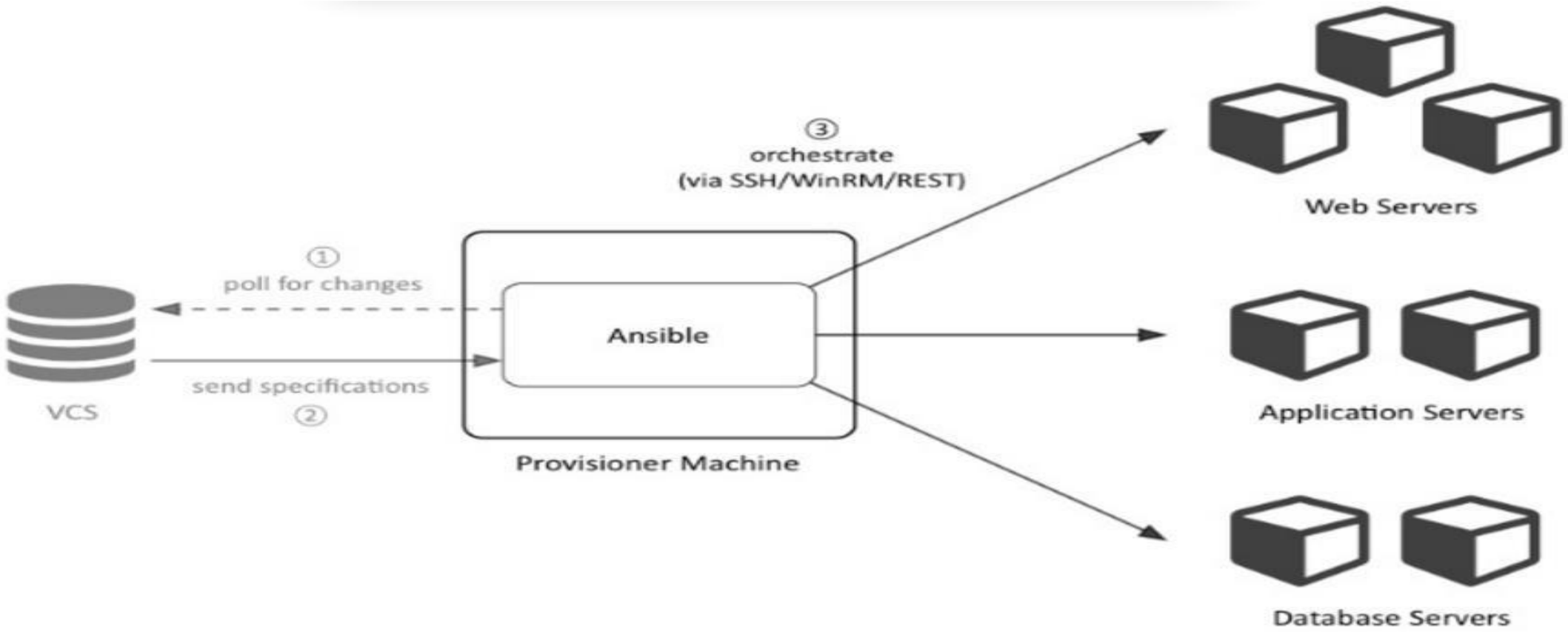
Ansible Components

- Ansible consists of **Agentless Model** and majorly have two parts:

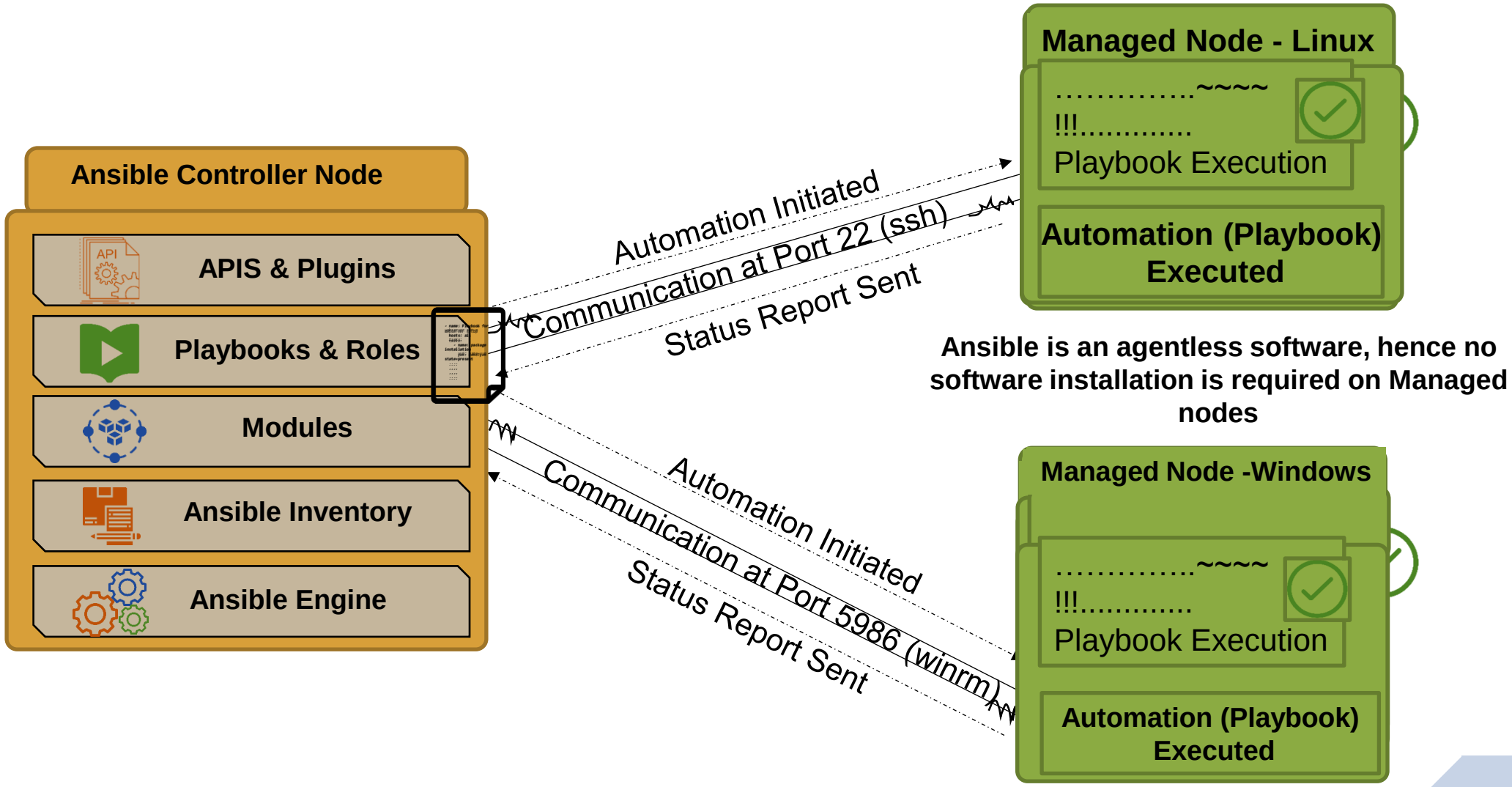
  - **Controller / Master**: The central configuration server where we will have our all configurations stored.
  - **Managed Nodes / Clients**: All clients getting configured from Ansible Master.

- **Note**:
- Ansible Master can be run from any Linux machine (Windows not supported) with Python 2 (version 2.7) or Python 3 (versions 3.5 and higher) installed.
- On the managed nodes, you need a way to communicate, which is normally SSH. By default this uses SFTP. If that's not available, you can switch to SCP in ansible.cfg. You also need Python 2 (version 2.6 or later) or Python 3 (version 3.5 or later).

Dataflow



Ansible Architecture



Ansible and its Peers

Many tools available in Market. Few things to consider, before selecting any tool:

- Configuration Management vs Orchestration
- Mutable Infrastructure vs Immutable Infrastructure
- Procedural vs Declarative
- Client/Server Architecture vs Client-Only Architecture

Ansible and its Peers

	Chef	Puppet	Ansible	SaltStack	CloudFormation	Terraform
Code	Open source	Open source	Open source	Open source	Closed source	Open source
Cloud	All	All	All	All	AWS only	All
Type	Config Mgmt	Config Mgmt	Config Mgmt	Config Mgmt	Orchestration	Orchestration
Infrastructure	Mutable	Mutable	Mutable	Mutable	Immutable	Immutable
Language	Procedural	Declarative	Declarative	Declarative	Declarative	Declarative
Architecture	Client/Server	Client/Server	Client-Only	Client/Server	Client-Only	Client-Only

Knowledge Checks

- What is Configuration Management?
- List a few available configuration Management tools.
- What are the Advantages of Ansible?
- Explain Data flow of Ansible.

Ansible Installation

Installation of Ansible

- The Ansible **master** is the machine that controls the infrastructure and dictates policies for the servers it manages.
- Currently Ansible can be run from any machine with Python 2.6 or 2.7 installed (Windows isn't supported for the control machine).
- This includes Red Hat, Ubuntu, Debian, CentOS, OS X, any of the BSDs, and so on.

Lab1: Installation of Ansible

- To install the Ansible Master, we need to install EPEL repository package:
  - Ansible Repository

http://fedoraproject.org/wiki/EPEL

Note: If internet connectivity is there just do:

- wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm

- Pre-installation
- Assign a hostname to your machine(Master) and make that name persist across reboot.

Lab1: Installation of Ansible

- yum install ansible
- rpm –qa | grep –i ansible
- ansible --version
- Default Configuration file is /etc/ansible/ansible.cfg
- Default Inventory file is /etc/ansible/hosts

Ansible Master Configuration

- Edit /etc/ansible/ansible.cfg for any Master Configurations
- Default options are fine
- All parameters can be overridden in ansible-playbook or with command line flags.

Special Shortcut: cat /etc/ansible/ansible.cfg | grep "^\["

Ansible Master Configuration

- Ansible comes with a default Ansible configuration file which can be customized by changing Ansible configuration parameters.
- **/etc/ansible/ansible.cfg** – by default base configuration file location.
- **~/.ansible.cfg** - user specific configuration file, this configuration file will be used by Ansible if Ansible is executed by logged in user.
- **./ansible.cfg** - the precedence will be given to this file, if Ansible run is executed from the directory path where ansible.cfg file is present.
- **ANSIBLE_CONFIG** - configuration file location defined by an environment variable.

Lab2: Working with Ansible.cfg

- Run Ansible --version command and check the outcome of “config file”.

- Copy /etc/ansible/ansible.cfg file into your home directory as below:
  - cp /etc/ansible/ansible.cfg ~/.ansible.cfg
- Run Ansible --version command and check the outcome of “config file”.

- Switch to /tmp and Copy /etc/ansible/ansible.cfg file into your Current directory as below:
  - cp /etc/ansible/ansible.cfg /tmp/ansible.cfg
- Run Ansible --version command and check the outcome of “config file”.

- Export ansible Config file into variable **ANSIBLE_CONFIG** as below:
  - cp /etc/ansible/ansible.cfg /ansible.cfg
  - export ANSIBLE_CONFIG=“/ansible.cfg”
- Run Ansible --version command and check the outcome of “config file”.

Ansible Authentication

- As Ansible is using SSH by default during communication, this communication connection supports both:

- **Password Based Authentication:** Password based authentication is acceptable if your environment is small and easily manageable. But it become very difficult to work with password-based authentications once you scale your environment. Password based authentication is only useful in Engineering Labs or Test Labs or Playbook creations tests.

- **Key Based Authentication:** Key based Authentication is adopted in Enterprise Environments. Here we create one generic user and amend the keys of the generic user in Managed Nodes for Key Based - Password less Authentication. This is a onetime task and can be used with any number of servers.

Ansible Inventory

- Ansible Inventory is a text-based list of individual servers and/or group of multiple servers.
- By default the Ansible Inventory location is **"/etc/ansible/hosts"**.
- You may have multiple Inventory files.
- Ansible Inventory can have Host Name or IP Address or Combination of both.
- Ansible provides the flexibility to pull inventory from Dynamic or Cloud sources with the help of scripts.
- You can specify a different inventory file using the -i <path> option on the command line.

Ansible Fundamentals

Case Study - 1

You need to manage a user, .

You care specifically about:

- his existence
- his primary group
- his home directory

Case Study - 1

- Tools built into most distro’s that can help:

- useradd
- usermod
- groupadd
- groupmod
- mkdir
- chmod
- chgrp
- chown

Case Study - 1

- Platform idiosyncrasies:
  - Does this box have ‘useradd’ or ‘adduser’?
  - What was that flag again?
  - What is difference between ‘–l’ and ‘–L’?
  - What does ‘–r’ means
  - Recursive
  - Remove read privileges
  - System user

- If I run this command again, what will it do?

Case Study - 1

- You could do something like this:

```
#!/bin/sh
USER=$1 ; GROUP=$2 ; HOME=$3
if [ 0 –ne $(getent passwd $USER > /dev/null)$? ]
then useradd $USER –home $HOME –gid $GROUP –n ; fi
OLDGID=`getent passwd $USER | awk –F: '{print $4}'`
OLDGROUP=`getent group $OLDGID | awk –F: '{print $1}'`
OLDHOME=`getent passwd $USER | awk –F: '{print $6}'`
if [ "$GROUP" != "$OLDGID" ] && ["$GROUP" != "$OLDGROUP" ]
then usermod –gid $GROUP $USER; fi
if [ "$HOME" != "$OLDHOME" ]
then usermod –home $HOME $USER; fi
```

Case Study - 1

What About?

- Robust error checking?
- Solaris and Windows support?
- Robust logging of changes?
- Readable code?
- What if need to create in 1000+ Servers?

Case Study – 1

Ansible Way of Configuration Management:

```
tasks:
 - name: Creating Gagandeep User
    user: name=gagandeep comment="Gagandeep Singh"  state=present

tasks:
 - name: Creating Singh Group
    group: name=singh state=present
```

Ansible Way: Maintaining State

- You(Even Ansible can do it on cloud) provision a node.
- Ansible configures it.
- Ansible maintains the desired state when needed.

Note: You make to sure the state is configured as per environment requirements.

Ansible : Infrastructure as Code

- Descriptive
- Straightforward
- Transparent

```
[root@gagan]# cat ntp.yml
---
# This is my Host section
- hosts: localhost
# This is my Task section
  tasks:
  - name: NTP Installation
    yum: name=ntp state=present
  - name: NTP Service
    service: name=ntpd state=started enabled=yes
```

Ansible : Idempotency

- Ansible enforces in an idempotent way.
- The property of certain operations in mathematics or computer science is that they can be applied multiple times without further changing the result beyond the initial application.
- Able to be applied multiple times with the same outcome.

Ansible Terminology

- **Controller/Master:** The Ansible master is the machine that controls the infrastructure and dictates policies for the servers it manages. It operates both as a repository for configuration data and as the control center that initiates remote commands and ensures the state of your other machines.

- **Managed/Agent Nodes :** The servers that Ansible configures are called Clients/Nodes.

- **Ansible Inventory:** Ansible Inventory represents which machines it should manage using a very simple INI file that puts all of your managed machines in groups of your own choosing.

- **Ansible Adhoc-tasks:** Ansible uses adhoc requests to confirm simple and small tasks on any server right a way without login into the client. The best example is to check the Alive Status for whole managed inventory.

Ansible Terminology

Playbooks: A structured way to put all of the defines tasks for your application or your whole setup.

Modules: In built functions which executes at the backend to perform underlined tasks in Ansible.

***yml files:** describe a set of desired states that a system needs to be in, for example “apache needs to be installed and running”.

Ansible Tower: Ansible Tower by Red Hat helps is a web-based solution that makes Ansible even more easy to use for IT teams of all kinds. It’s designed to be the hub for all of your automation tasks.

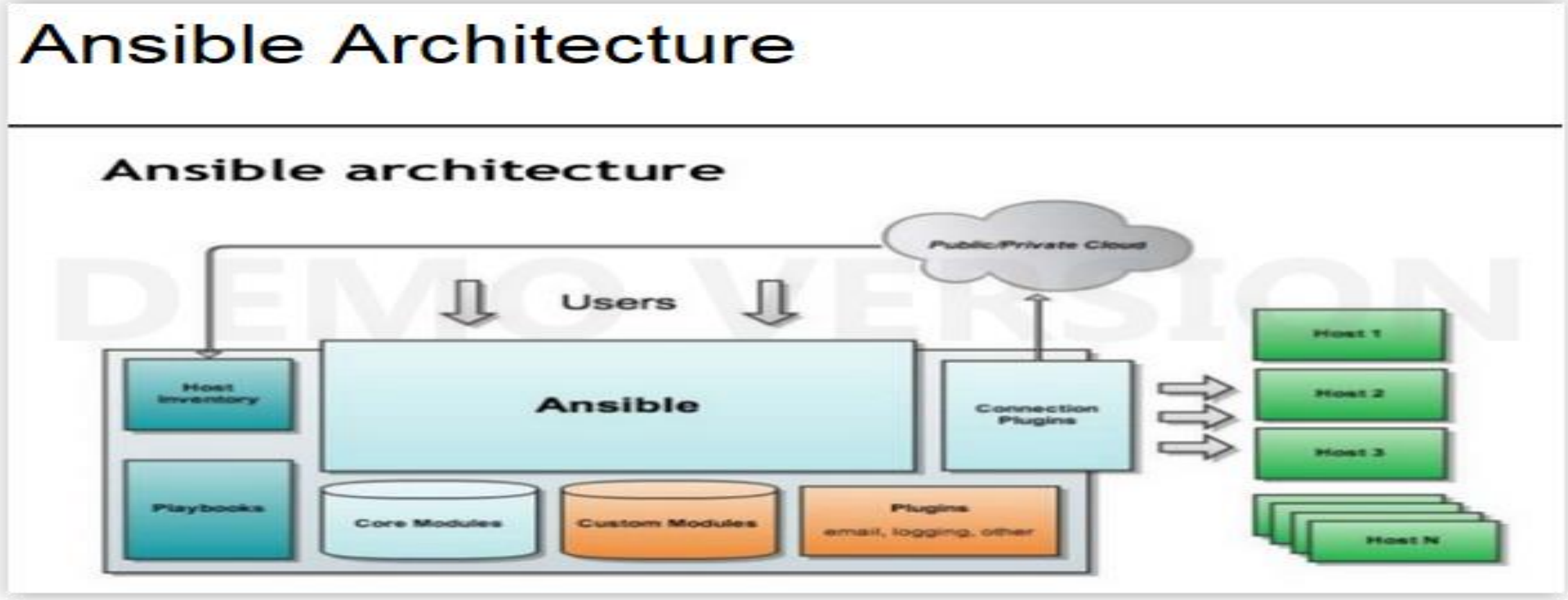
Ansible Terminology

- **Ansible Galaxy:** Ansible Galaxy is a free site for finding, downloading, and sharing community developed roles. Downloading roles from Galaxy is a great way to jumpstart your automation projects..

- **Ansible for Unix/Linux:** The Ansible master communicates and manage Unix/Linux Clients using SSH by default.

- **Ansible for Windows:** Starting in version 1.7, Ansible also contains support for managing Windows machines. This uses native PowerShell remoting, rather than SSH. and uses the “winrm” Python module to talk to remote hosts.

Ansible Communications



Ansible Communication test

- Communication checks with password authentications:

```
[root@gagan-controller ~]#ansible demo -m ping --ask-pass
SSH password:
centos-managed | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Lab 3: Ansible configuration

- Edit Client IP and Name entry in /etc/hosts file for Name to IP Mapping:

  172.31.19.138 gagan-client // /etc/hosts
- Put client name entry in in /etc/ansible/hosts file (under unmanaged section/top) , so that it can be managed with ansible:

  gagan-client // /etc/ansible/hosts
- Run ansible commands to check if client host is manageable with ansible or not:

  ansible gagan-client –m ping --ask-pass // ask-pass as we have not provided password in file
- Provide username and password in /etc/ansible/hosts file and run command without password:

  gagan-client ansible_user=USER ansible_password=PASSWORD // /etc/ansible/hosts
- Generate SSH keypair and copy keypair in remote machine for passwordless connection:

  ssh-keygen –t rsa // press enter thrice after this – no passphrase

  ssh-copy-id gagan-client // hit enter and provide password for next machine

  ansible gagan-client –m ping

  ansible-doc -l

Ansible Modules

- Modules are the Basic Building Block of Ansible.
- These are the readymade tools to perform various tasks and operations on “Managed Nodes”.
- Modules can be used with Ansible Ad-Hoc Remote Executions and/or Playbooks as a core building blocks.
- Ansible Ships with multiple in-built Modules (approx. 2000+ in Ansible 2.7).
- Can be used for Standalone servers, Virtual Machines and for any Public/Private Cloud Instances.

Ansible Modules

- Two types of Ansible Modules: Core Modules & Custom Modules.
- Robust Module Documentation on website.
- Command line utility on Module information and usage.
- CLI utility ansible-doc on “Controller Node”.
- Execute ansible-doc -l to list all available Modules.
- Execute ansible-doc <module-name> to find all details about Modules.
- For GUI refer **“http://docs.ansible.com/ansible/modules_by_category.html”**

Ansible Adhoc Execution

- Easy to learn ad-hoc command line utility - “ansible”.
- Quick On-demand tasks on “Managed Nodes”.
- 1 to 1 approach, single ad-hoc command is used to perform single operation.
- Multiple ad-hoc operations require multiple “ansible” ad-hoc run.
- Ad-hoc execution syntax.
- Ansible Ping Communication Test with “Managed Nodes”.
- Various real time examples with ad-hoc execution.

Ansible Adhoc Execution

- Let’s try executing a remote command, before that make sure you have out an entry of the host in “/etc/ansible/hosts”

- Connect to the master and type:

ansible <host-name/IP> -m ping --ask-pass

ansible "*" -m ping --ask-pass

- First argument = target client
- Second argument = function to execute
- Other arguments = params for the function

Ansible Adhoc Execution

Adding Username and Connection method in “/etc/ansible/hosts”:

You can specify each host for specific connection type/port and connection username:

- ansible_host : The name of the host to connect to, if different from the alias you wish to give
- ansible_port: The ssh port number, if not 22
- ansible_user: The default ssh user name to use.
- ansible_ssh_pass: The ssh password to use (never store this variable in plain text; always use a vault. See Variables and Vaults)
- ansible_ssh_private_key_file: Private key file used by ssh. Useful if using multiple keys and you don’t want to use SSH agent.

user20-client ansible_connection=ssh ansible_user=centos

Ansible Adhoc Execution

- There are a bunch of predefined :

«**Ad-Hocmodules**»

«**execution modules**»

«**Ad-Hocmodules**»: ansible all/"Client or Group“ -a “<adhoc-command>" --ask-pass

«**execution modules**» ansible all/"Client or Group“ -m <module_name> -a <arguments>

- Note: To list all Ansible Modules run below command:

ansible-doc -l

Ansible Adhoc Execution

- For example, executing a shell commands:

ansible 192.168.74.51 -a "ls -l /tmp" --ask-pass

ansible 192.168.74.51 -a "uname -a" --ask-pass

ansible 192.168.74.51 -a "cat /etc/redhat-release" --ask-pass

ansible 192.168.74.51 -a 'service ntpd status' --ask-pass

ansible “*” --list-hosts // Its’ll show all the hosts that’ll effect with the command

Ansible Adhoc Execution

- For example, executing Ansible Modules:

ansible 192.168.74.51 -m ping --ask-pass

ansible 192.168.74.51 -m user -a "name=rahejagagan state=present" --ask-pass

ansible 192.168.74.51 -m file -a "path=/var/tmp/gagandeep mode=777 group=rahejagagan state=touch" --ask-pass

ansible 192.168.74.51 -m service -a "name=ntpd state=stopped" --ask-pass

ansible 192.168.74.51 -m file -a "path=/var/yog/rah/test mode=777 state=directory" --ask-pass

Lab 4: Ansible Adhoc Commands

- Check the uptime using Ansible Ad-hoc execution with password-less authentication
- Check OS release using Ansible Ad-hoc execution with password-less authentication
- Install a package named “telnet” on managed host
- Create a user named “yourname” with bash shell having user id of 9999 on managed host
- Create a file named “/tmp/myfile” with permission 777 and user + owner as root on managed host
- Copy a local file to remote machine
- Run multiple commands parallelly with shell module.

Lab 4: Ansible Adhoc Commands

- Check the uptime using Ansible Ad-hoc execution with password-less authentication

  ansible client -m uptime
- Check OS release using Ansible Ad-hoc execution with password-less authentication

  ansible client -a "uname -a"
- Install a package named “telnet” on managed host

  ansible client -m package -a "name=telnet state=present"
- Create a user named “yourname” with bash shell having user id of 9999 on managed host

  ansible client -m user -a "name=gagandeep uid=9999 state=present"
- Create a file named “/tmp/myfile” with permission 777 and user + owner as root on managed host

  ansible client -m file -a "path=/tmp/myfile mode=777 group=root owner=root state=touch"
- Copy a local file to remote machine

  ansible client -m copy -a "src=/tmp/myfile dest=/root mode=777"
- Run multiple commands parallelly with shell module

  ansible client -m shell -a "ls;uname -a"

Facts

Ansible uses “facts” to gather information about the host system (any host).

“ansible <client-name> -m setup”

“ansible <client-name> -m setup –ask-pass”

Command returns a list of key value pairs (specific to Ansible).

The returned key value pairs are “facts”. Example:

- [root@user20-master ~]# ansible user20-client -m setup | grep -i ansible_user_id
- "ansible_user_id": "centos",
- [root@user20-master ~]# ansible user20-client -m setup -b | grep -i ansible_user_id
- "ansible_user_id": "root",
- [root@user20-master ~]#

Ansible Playbooks

Ansible Playbooks

- YAML Structure
- Ansible Playbooks
- Playbooks Structure
- Playbooks Syntax
- Playbooks Pre-Execution
- Playbooks Smoke Test
- Playbooks Real Time Run

YAML Structure

Though YAML syntax may seem daunting and terse at first, there are only three very simple rules to remember when writing YAML for Playbooks.

- Rule One: **Indentation**
  - YAML uses a fixed indentation scheme to represent relationships between data layers. Ansible requires that the indentation for each level consists of exactly two spaces. **Do not use tabs.**

YAML Structure

Rule Two: **Colons**

- Dictionary keys are represented in YAML as strings terminated by a trailing colon. Values are represented by either a string following the colon, separated by a space:

```
my_key: my_value
```

- In Python, the above maps to:

```
{'my_key': 'my_value'}
```

- Alternatively, a value can be associated with a key through indentation.

```
my_key:
   my_value
```

YAML Structure

Rule Three: **Dashes**

- To represent lists of items, a single dash followed by a space is used. Multiple items are a part of the same list as a function of their having the same level of indentation.

```
- list_value_one
- list_value_two
- list_value_three
```

- Lists can be the value of a key-value pair. This is quite common in Ansible:

```
my_dictionary:
  - list_value_one
  - list_value_two
  - list_value_three
```

Ansible Playbooks

- **Ansible Playbooks are the file in YAML format with sequential instructions to perform operations on Managed Nodes.**
- Written in YAML (**YAML** Ain't Markup Language).
- In simple layman language Playbooks are simple YAML files containing implementation steps.
- Opposite to Ad-Hoc requests, no restrictions on running multiple operations on managed nodes in a single run.
- Very easy to write and understand than other configuration Management tools.
- Ansible Playbooks file extension is .yml or yaml.

Ansible Playbooks

- The Playbook are typically divided into three parts, with YAML format:
  - Start of a Play (Hosts, variables, connections and Users)
  - Tasks list
  - Handlers

Ansible Playbook

first_playbook.yml

```
---
- hosts: all
tasks:
  - name: NTP OS Package Installation
    package: name=ntp state=present
  - name: NTP File Configurations
    copy: src=/etc/ntp.conf dest=/etc/ntp.conf
    notify:
    - restart ntp
  - name: To start NTP services
    service: name=ntpd state=started enabled=yes
handlers:
  - name: restart ntp
    service: name=ntpd state=restarted
```

Start of the Playbook

Generic Section

Playbook Sections

Tasks Section

Handlers

Playbooks Structure

- Playbooks in Ansible are "collection of Plays".
- **Plays** are nothing but a “collection of attributes“ and a “sequence of operations” to be performed on Managed Nodes.
- “Collection of Attributes” defines set of Managed Nodes, Communication Connection, Privilege escalations and different variable associated with playbooks.
- The "sequence of operations" is called "**Tasks**" in Ansible.
- Every Play is associated with one or more Ansible "Tasks".

Playbooks Structure

- The Tasks are created with the help of Ansible Modules to perform actions on Managed Nodes.
- Plays are associated with Managed Nodes and are called "hosts".
- Plays are associated with "Managed Nodes" to perform action on them.
- Playbooks are executed on Managed Nodes in sequence of contents written inside, so order of contents inside Playbook matters.
- Playbooks may have multiple plays associated with set of different "Managed Nodes".

Playbooks Structure

- **Start of a Play - Basic (Hosts and Users):**
- First part of Ansible Playbooks
- Provides the hosts/group of hosts to target
- Provides which user you want to perform the tasks
- For each play in a playbook, you get to choose which machines in your infrastructure to target and what remote user to complete the steps (called tasks)

Examples:

```
---
- hosts: webservers
  remote_user: root
```

Playbooks Structure

Examples:

You can also login as you, and then become a user different than root:

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
  become_user: postgres
```

You can also use other privilege escalation methods, like su:

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
  become_method: su
```

Playbooks Structure

- **Tasks list:**

- Each play contains a list of tasks. Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task. When running the playbook, which runs top to bottom, hosts with failed tasks are taken out of the rotation for the entire playbook. If things fail, simply correct the playbook file and rerun.

- The goal of each task is to execute a module, with very specific arguments

Playbooks Structure

Here is what a basic task looks like. As with most modules, the service module takes key=value arguments:

```
tasks:
  - name: make sure apache is running
    service: name=httpd state=started
```

Variables can be used in action lines. Suppose you defined a variable called vhost in the vars section, you could do this:

```
tasks:
  - name: create a virtual host file for {{ vhost }}
    template: src=somefile.j2 dest=/etc/httpd/conf.d/{{ vhost }}
```

Playbooks Structure

- **Handlers:**

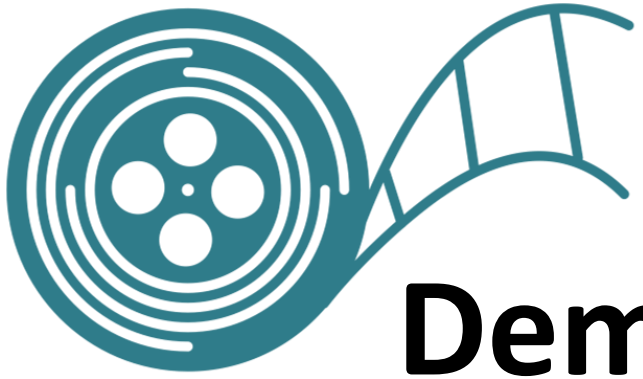
- In simple layman language Handlers are **Running Operations On Changes.**

- Handlers are lists of tasks, not really any different from regular tasks, that are referenced by a globally unique name, and are notified by notifiers.

- If nothing notifies a handler, it will not run.

- Regardless of how many tasks notify a handler, it will run only once, after all of the tasks complete in a particular play.

- The things listed in the notify section of a task are called handlers.



Demo

Playbooks Run

- Perform syntax check.
- Perform Dry Run Test.
- Perform Real time run.
- Check Results.

Playbooks Pre-execution

- Before running the Playbooks, lets explore some useful tips:
- Finding Modules and Attributes:
- ansible-doc –l
- ansible-doc -v service

- To see what hosts would be affected by a playbook before you run it:
- ansible-playbook playbook.yml --list-hosts

- Command to see all FACTS:
- ansible all -m setup --ask-pass

Playbooks Syntax Checks

Syntax checks for Playbooks:

#ansible-playbook --**syntax-check** play1.yml

[root@ansible]#ansible-playbook --syntax-check play1.yml
ERROR! Syntax Error while loading YAML.

The error appears to have been in '/etc/ansible/play1.yml': line 3, column 8, but may be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

0
- hosts: all
    ^ here

Playbooks Dry Run

Dry Runcheck for playbooks:

```
# ansible-playbook  <playbook-name>  --check

# [root@Ansible]#ansible-playbook play1.yml --check
SSH password:
PLAY [all] ********************************************************************
TASK [setup] ******************************************************************
ok: [192.168.74.51]
TASK [File Creation] **********************************************************
changed: [192.168.74.51]
TASK [Directory Creation] *****************************************************
changed: [192.168.74.51]
PLAY RECAP ********************************************************************
192.168.74.51              : ok=3    changed=2    unreachable=0    failed=0
```

Playbooks Real Time Run

Real time run for playbooks:

```
# ansible-playbook play1.yml
SSH password:
PLAY [all] *********************************************************************
TASK [setup] *******************************************************************
ok: [192.168.74.51]
TASK [File Creation] ***********************************************************
changed: [192.168.74.51]
TASK [Directory Creation] ******************************************************
changed: [192.168.74.51]
PLAY RECAP *********************************************************************
192.168.74.51              : ok=3    changed=2    unreachable=0    failed=0
```

Step Run

Moving through task by task execution:

Special Case run Ansible playbooks with –step

Example:

```
[root@user20-master plays]# ansible-playbook ntp.yaml --step
PLAY [all] ********************************************************************************************************************
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: n
Perform task: TASK: Second task (N)o/(y)es/(c)ontinue:
**********************************************************************
Perform task: TASK: Third task to start NTP Services (N)o/(y)es/(c)ontinue: y
TASK [Third task to start NTP Services] ***************************************************************************************
ok: [user20-client]
PLAY RECAP
********************************************************************************************************************
user20-client          : ok=1    changed=0    unreachable=0    failed=0
[root@user20-master plays]#
```

Lab5: Working with Playbook

- Let's do some Ansible Playbooks creations:

- Create two users – “your name” and “your name1” on next machine using playbook
- Create a group – “your name” on next machine

- Perform Syntax check
- Do Dry run of playbook
- Run the playbook

Lab6: Working with Playbook

- Create a Playbook for User and Group Creation with user name "usertest", shell bash, userid 6666 and pass the comments as "my first user". Group details will be name "grouptest" and group id 7777.

- Create a Playbook for files and directories:

  create a directory with root ownership; inside this directory, create one file with “test” with ownership of usertest (the user we have created in 1st example). Copy some content into the newly created file.

Handlers

- In simple layman language Handlers are **“Running Operations on Changes”.**
- Handlers are list of tasks with unique names.
- “notify” is the keyword to trigger operations mentioned in Handlers.
- Regardless of how many tasks notify a handler, it will run only once, after all of the tasks complete in a particular play.
- If nothing changed in the tasks, Handlers will not run.
- Multiple “notify” with unique names are permitted.
- Handlers are always mentioned at last.

Handlers

Let's understand the importance of Handlers with below use cases:

- Playbook without Handlers
- Playbook with Handlers (with improper usage of handler)
- Playbook with Proper use of Handler

Lab - NTP

- Let’s do a practical for NTP module including:

- Package
- Files
- Service

Lab - NTP

NTP without proper config

NTP with proper config

Lab 7: Working with handlers

- Create a Playbook for NTP configuration to run the operations on change of the configuration on /root/ntp.conf file change.
  - Task 1- to install ntp on next machine
  - Task 2 – to copy /root/ntp.conf file from master to client machine at /etc/ntp.conf
    - Add Notify section pointing to restart-ntp task of handler
  - Task 3: to start the service
  - Add handler restart-ntp to restart the ntp service

- Create one file /root/ntp.conf on master machine and run the playbook.

Ansible Variables

Ansible Variable overview

- Like any other Automation language Ansible also supports variables.
- In layman language variable is just an assigned value (string or characters).
- Variables can be used with any playbook, tasks, roles, templates and even with inventory files.
- There is a well-defined variable naming scheme in Ansible, it should always start with alphabet.
- Variables are first defined and then declared to be used in the Ansible.
- In simplest way variables are defined inside playbooks under “vars” section in the play.

Ansible Variable overview

- Variable Definition
- # Defining variable section in playbook using vars parameter
- vars:
- user1:raman

- Variable Declaration
- tasks:
- - name: user creation
- user: name={{ user1 }} state=present

Ansible Variable overview

You can define empty variables in playbook and later on provide variable values during runtime

Lab 8.1 : Working with Variables

- Create a playbook for to install httpd(apache webserver)

Lab 8.2 : Working with Variables

- Install an httpd package in playbook. Package name should go as variable
- Configure hostname and IP address in webpage (using facts) and owner (through variable)
- Handler to be used for restarting httpd service

Ansible Facts

- Facts are the special variables automatically discovered by Ansible.
- Dedicated “setup” module.
- Facts output is in JSON format.
- Facts provide information about hostname, kernel, network, OS and much more.
- The information from facts can be filtered using -a filter=<value> option to get specific information.

Ansible Facts

- [root@gagan-controller ~]# ansible [host] -m setup
- [root@gagan-controller ~]# ansible [host] -m setup | grep -i "ansible_"
- [root@gagan-controller ~]# ansible [host] -m setup -a filter="ansible_all_ipv4_addresses"
- [root@gagan-controller ~]# ansible [host] -m setup -a filter="ansible_date_time"

Ansible Facts

- Facts values are used as variables to create useful generic and dynamic playbooks .
- Facts outputs are JSON in format.
- Declared format in playbook is {{ <ansible_facts> }}
- Ansible automatically replace the Facts variable with values while Ansible run.

Ansible Facts

- First step is to find facts to be used in Playbook.
- [root@gagan-controller ~]# ansible centos-managed -m setup -a filter="ansible_hostname“
- [root@gagan-controller ~]# ansible "*" -m setup -a filter="ansible_eth0“

Note: "ansible_eth0"will be having detailed output in JSON format which you can even filter out for specific value. In our example we will filter out “interface address” as {{ ansible_eth0.ipv4.address }} which means check ansible_eth0 then go to ipv4 section and then return the address attribute value.

Ansible Facts

```
[root@]# cat factstest.yml
---
- name: This is my First Debug Play
  hosts: all
  tasks:
    - name: Testing Ansible Facts {{ ansible_hostname }}
# My task with outputs fetched from Facts
      debug: msg="Host {{ ansible_hostname }} is having IP address {{ ansible_eth0.ipv4.address }}"
...
```

Ansible Verbose Run

Run Ansible in Verbose mode to show detailed execution logs

```
[root@user20-master plays]# ansible-playbook debug-register.yml -vv
ansible-playbook 2.4.2.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /bin/ansible-playbook
  python version = 2.7.5 (default, Oct 30 2018, 23:45:53) [GCC 4.8.5 20150623 (Red Hat 4.8.5-36)]
Using /etc/ansible/ansible.cfg as config file

[root@user20-master plays]# ansible-playbook debug-register.yml  | wc -l
21
[root@user20-master plays]# ansible-playbook debug-register.yml -vvvvv | wc -l
233
```

Ansible Debug

Debug Module is used for debugging the outputs.

It prints statements during execution and can be useful for debugging variables or expressions without necessarily halting the playbook.

Parameter	Choices/Defaults	Comments
msg	**Default: Hello world!**	The customized message that is printed. If omitted, prints a generic message.
var		A variable name to debug. Mutually exclusive with the 'msg' option.
verbosity	**Default: 0**	A number that controls when the debug is run, if you set to 3 it will only run debug when -vvv or above

Ansible Register

Ansible comes with a special variable name “Register” to capture command output run by Ansible run and save them into variable value.

It is mostly used with Debug Module.

Register - Printing specific value

Specific value from Ansible Register output can also be printed:

Lab 9 : Working with Registers

- Create a playbook with Debug and Register variable for below:
  - Install NTP package on next machine (you can use existing playbooks)
  - Capture the output of installation task into Register
  - Print the output with Debug Module
  - Run the playbook and check the output
  - Edit the debug level to 1 (-v) in playbook
  - Run the playbook and check the difference
  - Run the Playbook in verbose mode with –v and then check the difference.

Ansible Ignore Errors

By default if a task fails, Ansible stops running further playbook tasks and exits the program.

When you want to ignore failure for a task, which don’t have any impact on pending tasks, you can ignore errors for that task.

Working with Conditions

Conditional tasks can be performed using “When” Statement.

Ansible allows special statements with “AND” and “OR”.

```
---
- name: Conditional checks
  hosts: all
  tasks:
    - name: Installing web packages on all centos version 7 servers
      yum: name=httpd state=installed
      when: ansible_distribution == “CentOS” and ansible_distribution_major_version == “7”

 - name: "shut down CentOS 6 and Debian 7 systems"
    command: /sbin/shutdown -t now
    when: (ansible_facts['distribution'] == "CentOS" and ansible_facts['distribution_major_version'] == "6") or
          (ansible_facts['distribution'] == "Debian" and ansible_facts['distribution_major_version'] == "7")
```

Working with Conditions

Working based on previous task output

```
- hosts: all
  tasks:
  - name: Register a variable
    package: name=ntp state=installed
    register: ntp_out
    ignore_errors: true
  - name: debug
    debug:
      var: ntp_out
  - name: Use the variable in conditional statement
    shell: echo "motd contains the word ansible"
    when: ntp_out.rc == 0
    register: echo_output

  - name: Display echoed message
    debug:
      var: echo_output.stdout

  - name: Register a variable
    shell: cat /etc/motd
    register: motd_contents

  - name: Use the variable in conditional statement
    shell: echo "motd contains the word hi"
    when: motd_contents.stdout.find('hi') != -1
    register: echo_output

  - name: Display echoed message
    debug:
      var: echo_output.stdout
```

Lab 10

- Create a playbook with Debug and Register variable for below:
  - Install NTP package on next machine (you can use existing playbooks)
  - Create task 1 for running command “/dev/null”
  - Rest tasks of NTP and service run will stay as it is.
  - Run the playbook and observe playbook getting failed at task 1 itself
  - Put a condition for task one to ignore error if fails
  - Run the playbook and spot the difference
  - Add another task at last, to run only if tasks 1 fails or based on condition of previous tasks run (pass/fail)
  - Run the playbook and check the difference

Working with loops

Often, you'll want to do many things in one task, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached.

Handling multiple similar tasks simultaneously can be done in Ansible to keep your code small and simple.

php gcc talk vim httpd

use **with_items or loop**

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  with_items:
     - testuser1
     - testuser2
```

Working with loops

e.g

```
- file:
    src: '/tmp/{{ item.src }}'
    dest: '{{ item.dest }}'
    state: link
  with_items:
    - { src: 'x', dest: 'y' }
    - { src: 'z', dest: 'k' }
```

Use Loop instead of with_items

With_item is being replaced with loop now.

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
     - testuser1
     - testuser2
```

Working with loop and condition

Re-iteration with condition:

```
- name: Run with items greater than 5
      command: echo {{ item }}
      loop: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

Reading variables/item from files:

```
 - name: task to create multiple users
      user: name={{ item }} state=absent
      with_lines: cat /root/gagan/users.txt
```

Lab 11 : Working with Loops

Install below packages on CentOS version 7 machine only (condition of Centos 7):

php

gcc

talk

vim

httpd

Used conditions and Loops

Reboot Machines during a play

```
- name: Run with items greater than 5
  command: echo {{ item }}
  loop: [ 0, 2, 4, 6, 8, 10 ]
  when: item > 5

- name: Reboot a slow machine that might have lots of updates to apply
  reboot:
    reboot_timeout: 300

- name: second task for package
  package:
    name: telnet
    state: absent
```

Retries

We can retry failed task on a specific task:

```
- name: second task for package
  package:
    name: telnet
    state: absent
  retries: 2
```

Limiting playbook to limited servers can be done via:

ansible-playbook first.yml --**limit** client

Ansible Blocks

Blocks create logical groups of tasks. Blocks also offer ways to handle task errors, similar to exception handling in many programming languages.

```
tasks:
  - name: Install, configure, and start Apache
    block:
      - name: Install httpd and memcached
        yum:
          name:
          - httpd
          - memcached
          state: present

      - name: Apply the foo config template
          template:
          src: templates/src.j2
          dest: /etc/foo.conf

      - name: Start service bar and enable it
        service:
          name: bar
          state: started
          enabled: True
    when: ansible_facts['distribution'] == 'CentOS'
    become: true
    become_user: root
    ignore_errors: yes
```

Best Practices

Logs

By Default, logging is disabled. Enable same for audit purpose and for storing the output for a later reuse.

Uncomment below logging configuration in ansible.cfg file:
log_path = /var/log/ansible.log

To disable tasks logs at master/client level, below parameters can be touched:
prevents logging of task data, off by default
#no_log = True
prevents logging of tasks, but only on the targets, data is still logged on the master/controller
#no_target_syslog = False

#By default, logs goes as per syslog configuration on client machine.

Working with tags

Tags helps us to **run/avoid** specific **tasks/blocks/roles** from a playbook.

```
 - name: first task for creating a user
     user:
       name: gagandeep2
       state: present
       tags:
         - prod

    - name: Run with items greater than 5
      command: echo {{ item }}
      loop: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
      tags:
        - dev
```

```
# ansible-playbook first.yml --tags dev
# ansible-playbook first.yml --tags prod,db
# ansible-playbook first.yml --skip-tags prod
```

You can also use **--start-at-task** for running playbook from a specific task.

Disable Gather_facts if not required

You can disable the facts in playbook to avoid network congestion due to Facts collection feature of Ansible before every node run:

```
---
- name: This is my First Debug Play
  hosts: all
  gather_facts: no

  tasks:
    - name: Testing Ansible Facts {{ ansible_hostname }}
      debug: msg="Host {{ ansible_hostname }} is having IP address {{ ansible_eth0.ipv4.address }}"
...
```

Note: This playbook will fail as we are using environment variables without fetching the facts

Import tasks from other playbooks

```
---
- hosts: gagan-client
  vars:
    user: gagandeepsingh
    uid: 10000
  tasks:
    - name: Creating {{user}}
      user: name={{user}} uid={{uid}} state=present
      register: outputs
    - name: Checking outputs
      debug: var=outputs
    - name: Import tasks from
      import_tasks: ./ntp.yml
```

... OR

```
- name: Reuse Task 2 from Other Playbook
  hosts: your_target_hosts
  tasks:
    - import_playbook: path/to/other_playbook.yml
      tags: task2
```

…OR

```
- name: Reuse Task 2 from Other Playbook
  hosts: your_target_hosts
  tasks:
    - name: Import Task 2 from Other Playbook
      include_tasks: path/to/other_playbook.yml
      tags: task2
```

Note: import_tasks playbook should have only tasks included.

Forks

By default, Ansible runs each task on all hosts affected by a play before starting the next task on any host, using 5 forks. If you want to change this default behavior, you can use a different strategy plugin, change the number of forks, or apply one of several play-level keywords like serial.

Ansible talks to remote nodes in parallel and the level of parallelism can be set either by passing

--forks with commands

or editing the default in the configuration file.

Ansible.cfg:

Forks=10

The default is a very conservative five (5) forks, though if you have a lot of RAM + Good NW bandwidth, you can easily set this to a higher value for increased parallelism.

You can also set fork during execution with –f option.

Strategy

By default, Ansible runs each task on all hosts affected by a play before starting the next task on any host, using 5 forks.

#ansible-doc -t strategy -l
debug Executes tasks in interactive debug session
host_pinned Executes tasks on each host without interruption
linear Executes tasks in a linear fashion
free Executes tasks without waiting for all hosts
#
Linear: The default behavior described above is the **linear** strategy.
Free: Ansible will not wait for other hosts to finish the current task before queuing more tasks for other hosts. All hosts are still attempted for the current task, but it prevents blocking new tasks for hosts that have already finished.

You can change behavior in ansible.cfg file:
#strategy = free
Or in playbook too:
- hosts: all
  strategy: free

To do the practical, better have 8-10 tasks with 4-5 managed hosts and use yum lock on next machine.

Rolling Update Batch Size

By default, Ansible will try to manage all of the machines referenced in a play in parallel. For a rolling update use case, you can define how many hosts Ansible should manage at a single time by using the serial keyword:

```
---
- name: test play
  hosts: webservers
  serial: 2
  gather_facts: False

  tasks:
    - name: task one
      command: hostname
    - name: task two
      command: hostname
```

Rolling Update Batch Size

In the previous page example, if we had 4 hosts in the group ‘webservers’, 2 would complete the play completely before moving on to the next 2 hosts:

```
PLAY [webservers] ****************************************

TASK [task one] ******************************************
changed: [web2]
changed: [web1]

TASK [task two] ******************************************
changed: [web1]
changed: [web2]

PLAY [webservers] ****************************************

TASK [task one] ******************************************
changed: [web3]
changed: [web4]

TASK [task two] ******************************************
changed: [web3]
changed: [web4]

PLAY RECAP ***********************************************
web1     : ok=2   changed=2   unreachable=0   failed=0
web2     : ok=2   changed=2   unreachable=0   failed=0
web3     : ok=2   changed=2   unreachable=0   failed=0
web4     : ok=2   changed=2   unreachable=0   failed=0
```

Rolling Update Batch Size

The serial keyword can also be specified as a **percentage**, which will be applied to the total number of hosts in a play, in order to determine the number of hosts per pass:

```
---
- name: test play
  hosts: webservers
  serial: "30%“
```

Or can be defined as a **list**:

```
---
- name: test play
  hosts: webservers
  serial:
    - 1
    - 5
    - 10
```

Max Failure Percentage

By default, Ansible will continue executing actions as long as there are hosts in the batch that have not yet failed. The batch size for a play is determined by the serial parameter. If serial is not set, then batch size is all the hosts specified in the hosts: field.

In some situations, such as with the rolling updates described above, it may be desirable to abort the play when a certain threshold of failures have been reached. To achieve this, you can set a maximum failure percentage on a play as follows:

```
---
- hosts: webservers
  max_fail_percentage: 30
  serial: 10
```

Lab 13 : Max-fail-percentage

- Install an httpd package in playbook
- Configure hostname and IP address in webpage (using facts) and owner (through variable)
- Handler to be used for restarting httpd service
- Use Register and Debug to print the output of installation of httpd package
- Use serial to run it on one host at a time
- Use Max-fail-percentage to 30%
- Encrypt inventory file using vault
- Run playbook and verify the outcome in your browser
- Disable gather facts and then check the runtime output

Lab 14 : Software Orchestration

Create a playbook for Tomcat end to end installation and extraction:

- Pre-requisities: java-1.8.0-openjdk package should be present
  wget should be present
  /opt/tomcat directory should exists

- Now download tomcat tar file from http://mirror.23media.de/apache/tomcat/tomcat-8/v8.5.29/bin/apache-tomcat-8.5.29.tar.gz and copy it under /opt/tomcat directory

- Extract the downloaded tomcat file
- Change the permissions of startup.sh shutdown.sh catalina.sh under /opt/tomcat/apache-tomcat-8.5.29/bin/ directory

- Now download the sample application from https://tomcat.apache.org/tomcat-8.0-doc/appdev/sample/sample.war and place it under /opt/tomcat/apache-tomcat-8.5.29/webapps/

- Wait for 60 secs

- Start the tomcat services /opt/tomcat/apache-tomcat-8.5.29/bin/catalina.sh start

- Verify your ans by checking the browser at client on port 8080

Lab 14 : Software Orchestration

```
[root@master centos]# cat test.yml
- name: Tomcat web server deployment
  hosts: all

  tasks:

  - name: Pre-req's for Tomcat to ensure no sample war is present
    file: path=/opt/tomcat/ apache-tomcat-9.0.75 /webapps/sample.war state=absent

  - name: Install prerequisties
    yum: name=java-1.8.0-openjdk state=present

  - name: Installing wget
    yum: name=wget state=present

  - name: Creating Directory structure
    file: name=/opt/tomcat state=directory

  - name: Downloading Tomcat Binaries
    get_url: url=https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.75/bin/apache-tomcat-9.0.75.tar.gz dest=/opt/tomcat

  - name: Extracting Tomcat
    shell: cd /opt/tomcat; tar -xvzf apache-tomcat-9.0.75.tar.gz; cd apache-tomcat-9.0.75/bin/; chmod 777 startup.sh shutdown.sh catalina.sh

  - name: War deployment
    shell: cd /opt/tomcat/apache-tomcat-9.0.75/webapps/; wget https://tomcat.apache.org/tomcat-9.0-doc/appdev/sample/sample.war; cd
/opt/tomcat/apache-tomcat-9.0.75/bin; sleep 60

  - name: Starting Tomcat
    command: /opt/tomcat/apache-tomcat-9.0.75/bin/catalina.sh start
```

Ansible Vault

Ansible Vault

Ansible Vault is used to store confidential data. It can encrypt or decrypt the playbooks/hosts files to protect sensitive information.

Dedicated CLI utility “ansible-vault”.

Pre-requisite is to have cryptography package.

Ansible-vault positional arguments:

create	Create new vault encrypted file
decrypt	Decrypt vault encrypted file
edit	Edit vault encrypted file
view	View vault encrypted file
encrypt	Encrypt YAML file
encrypt_string	Encrypt a string
rekey	Re-key a vault encrypted file

Ansible Vault

```
[root@ansible-controller playbooks]# ansible-vault create vault.yml
New Vault password:
Confirm New Vault password:
[root@ansible-controller playbooks]# ansible-vault encrypt existingplaybook.yaml
New Vault password:
[root@ansible-controller playbooks]# ansible-vault decrypt existingplaybook.yaml
Enter Vault password:
[root@ansible-controller playbooks]# ansible-vault edit vault.yml
Enter Vault password:
```

Ansible Vault

```
[root@ansible-controller playbooks]# ansible-vault view vault.yml
Vault password:
---
- name: Installing telnet with Vault protected file
  hosts: all
  tasks:
    - name: Telnet Package
      yum: name=telnet state=present
```

Ansible Vault

```
[root@ansible-controller playbooks]# ansible-playbook --check vault.yml
ERROR! Decryption failed on /var/tmp/playbooks/ vault.yml

[root@ansible-controller playbooks]# ansible-playbook --check vault.yml--ask-vault-pass
Vault password:
PLAY [Installing telnet with Vault protected file]
**********************************************************************************************
TASK [Gathering Facts]
**********************************************************************************************ok:
[ansible-managed]
TASK [Telnet Package]
**********************************************************************************************changed:
[ansible-managed]
PLAY RECAP
**********************************************************************************************ansible-
managed          : ok=2    changed=1    unreachable=0    failed=0
```

Multiple Password Management

ansible-vault encrypt --vault-id **password1**@prompt /etc/ansible/hosts
ansible-vault encrypt --vault-id **password2**@prompt loop.yml

[root@gagan-ansible gagan]# ansible-playbook loop.yml --vault-id password1@prompt --vault-id password2@prompt
Vault password (**password1**):
Vault password (**password2**):

PLAY [first play]
****
*

Lab 15 : Working with Vaults

Working with vaults:

Encrypt your playbook with ansible-vault command
Run playbook and spot the output
Provide password with --ask-vault-pass and then spot the outcome
Encrypt your inventory file as well with different password and vault-id
Now run playbook with providing vault-id and password for same.

Ansible Roles

Ansible Roles

Ansible Roles are the way to keep Ansible Playbooks well organized.

Role Directory structure allow to load tasks, handlers, variables, static files, templates.

Can be shared with external world with ease.

Best way to orchestrate application deployments.

Default configuration directory /etc/ansible/roles.

Dedicated Role creation utility: ansible-galaxy.

Ansible Roles

Lets create our first role with ansible-galaxy utility, execute this command in dedicated directory only.

ansible-galaxy init gaganrole.

Go inside the gaganrole directory and walk through the directory structure

Ansible Roles Structure

Default – This section is use to hold default variables by a role.

Handlers – This section is used to hold function associated with notify to trigger service-related operations.

Meta – This section is used to hold information like author of a role, description, company name, license and more importantly the **dependencies**.

Tasks – This is the section which will contain the actual code.

Tests – Test cases will be kept here.

Vars – This section is also for variables but with higher precedence than defaults section.

files - contains files which can be deployed via this role.

Ansible Roles- Demo

Lets create a simple role to understand how actual role works.

Create a Role Structure.

Define the role in the playbook directory, test and execute the playbook.

Deployment of Ansible Roles

```
[root@ansible-controller role]# cat defaults/main.yml
---
# defaults variable file for file for gaganrole
user:
  gagandeep
uid:
  7654
```

Note: make sure, not to write "vars" on top.

Deployment of Ansible Roles

```
[root@ansible-controller role]# cat tasks/main.yml
---
# tasks file for gaganrole
- name: Creating {{user}}
  user: name={{user}} uid={{uid}} state=present
  register: outputs
- name: Checking outputs
  debug: var=outputs
```

Note: make sure, not to write "tasks" on top.

Deployment of Ansible Roles

```
[root@ansible-controller playbooks]# cat gaganroletest.yml
---
- name: Testing role
  hosts: all

# Define roles here
  roles:
    - gaganrole
```

Deployment of Ansible Roles

Perform Syntax checks on playbook

Dry run test

Real time run on the playbook containing role

Lab 16 : Working with Roles

Create a role for NTP Service with below

Create tasks for below:

- User creation [yourname] with UID 1020
- Package Installation
- File Copying
- Service Start

Use Variables for Username and UID

Handlers should be there to restart the NTP Service

Ansible Roles Import tasks

You can Import other tasks/playbooks in the main.yml file to ease your code and enhance the re-usability of code:

```
# roles/example/tasks/main.yml
- name: added in 2.4, previously you used 'include'
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'
- import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'
# roles/example/tasks/redhat.yml
- yum:
    name: "httpd"
    state: present
# roles/example/tasks/debian.yml
- apt:
    name: "apache2"
    state: present
```