DO407
Automation with Ansible

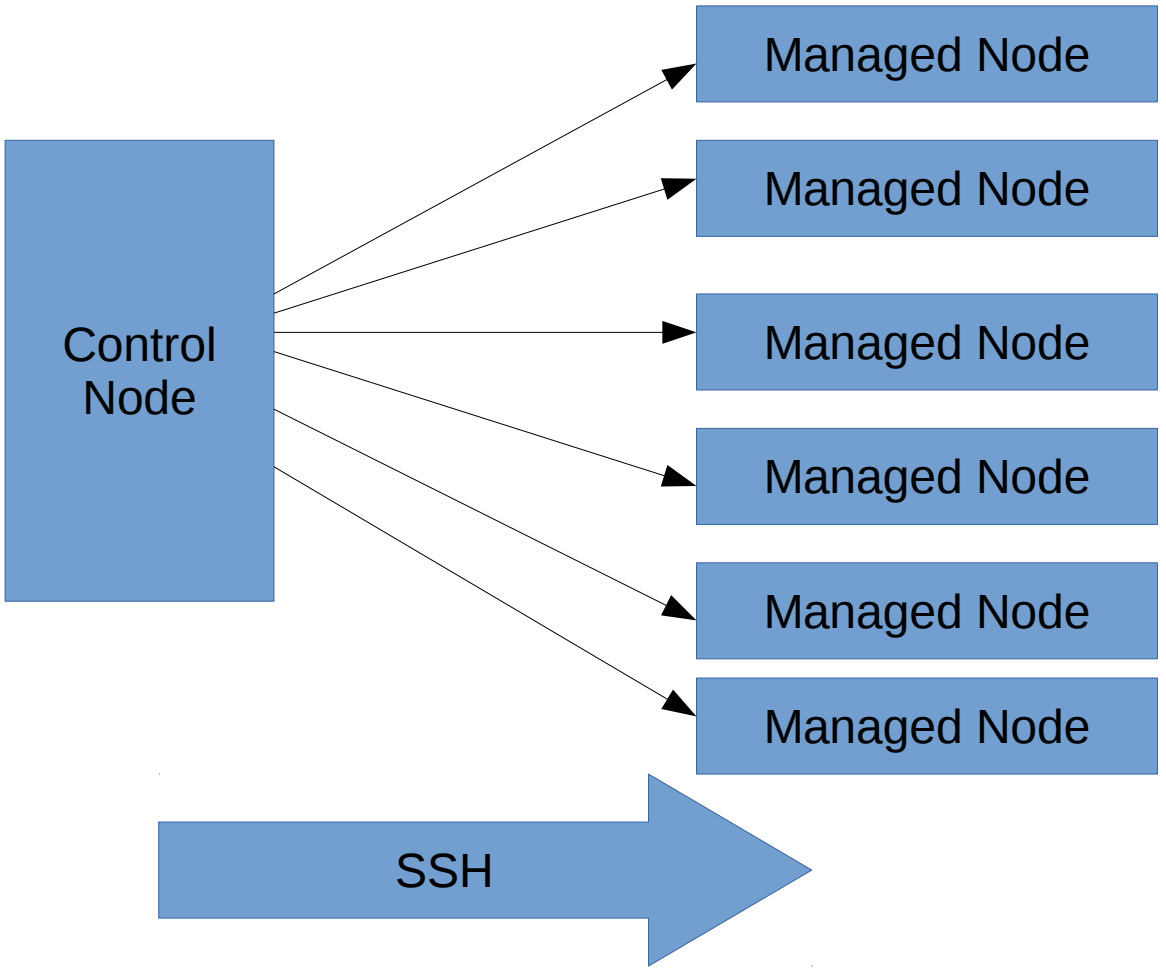
What is Ansible ?

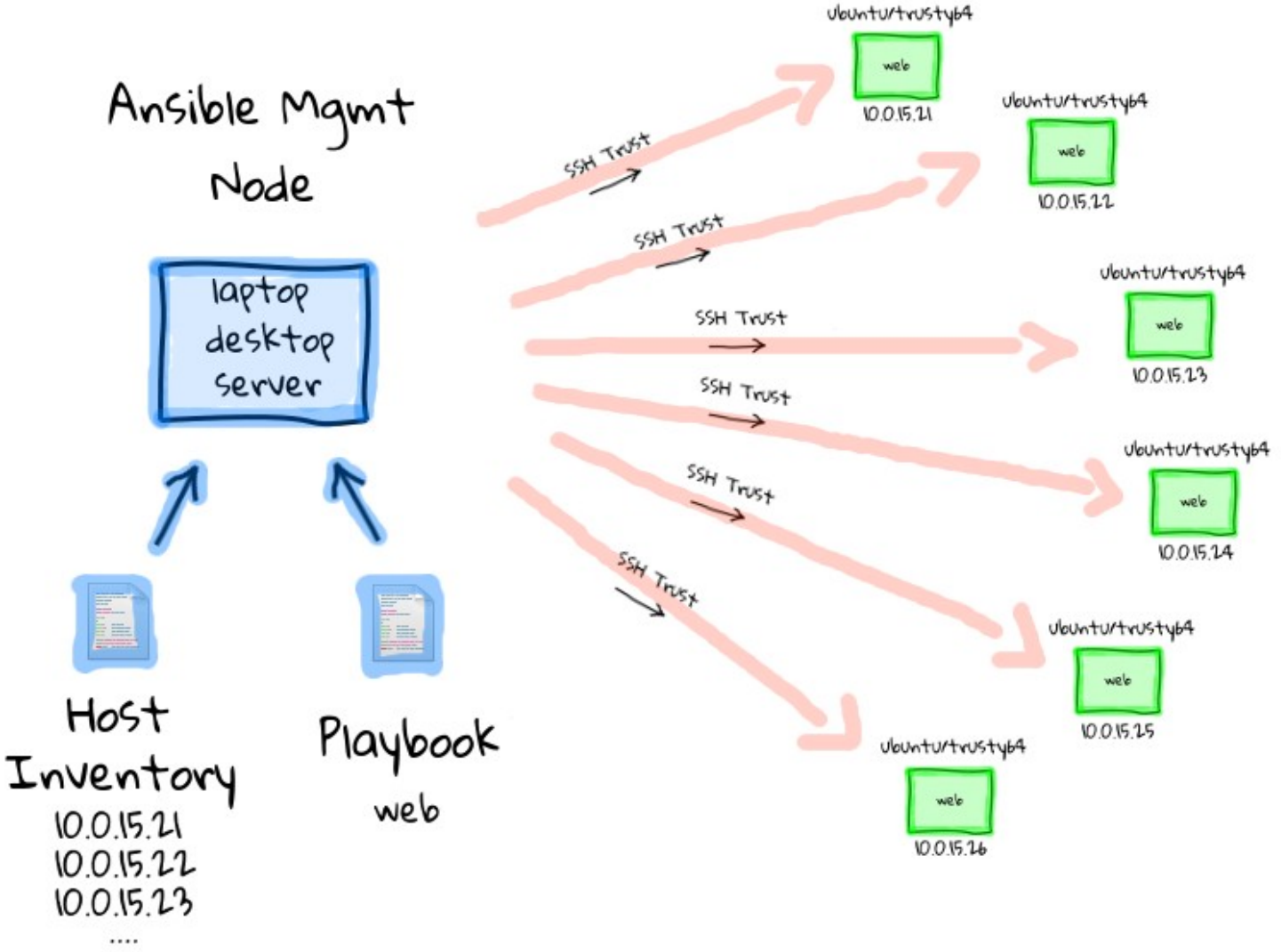
- Configuration Management
- Orchestration Utility
- Its Architecture is Agentless
- Originally written by Michael DeHaan, creator of cobbler provisioning application.
- Built, on Python
- Supported by well known DevOps tools like vagrant, Jenkins etc.

What is Ansible.....contd

- Ansible creates “plays” in “playbooks”.
- Work is pushed to remote hosts when Ansible executes.
- Modules are the programs that perform the actual work of the tasks of a play.
- Ansible is immediately useful because it comes with hundreds of core modules that perform useful system administrative work.

Ansible Architecture




Ansible Mgmt Node
laptop
desktop
server
Host Inventory
10.0.15.21
10.0.15.22
10.0.15.23
....
Playbook
web
SSH Trust
SSH Trust
SSH Trust
SSH Trust
SSH Trust
SSH Trust
ubuntu/trusty64
web
10.0.15.21
ubuntu/trusty64
web
10.0.15.22
ubuntu/trusty64
web
10.0.15.23
ubuntu/trusty64
web
10.0.15.24
ubuntu/trusty64
web
10.0.15.25
ubuntu/trusty64
web
10.0.15.26

Key Terminologies

- ***Configuration :*** Ansible have its own config which manages the behaviour of ansible
- ***Host Inventory:*** Its the list the of “Managed Hosts” and their connection configuration.
- ***Core Modules:*** There are around 400+ core modules that comes bundled with Ansible
- ***Custom Modules:*** Users can extend Ansible's functionality by writing custom modules, which are written in Python typically, but can also be written in Ruby, Python, shell etc.

Key Terminologies...cont'd

- ***Playbooks:*** Ansible playbooks are files written in YAML syntax that define the modules, with arguments, to apply to managed nodes. They declare the tasks that need to be performed.
- ***Connection Plugins:*** Plugins that enable communication with managed hosts or cloud providers. These include native SSH, paramiko SSH, and local.
- ***Plugins:*** Extensions that enhance Ansible's functionality. Examples include e-mail notifications and logging.

Requirements for Contol Node

- Ansible software is installed on the control node.
- A machine acting as a control node must have Python 2.6 or 2.7 installed.
- Red Hat Enterprise Linux 6 or 7 will run Ansible software.
- Windows is not supported for the control node at this time.

Requirements on Managed Node

- SSH must be installed and configured to allow incoming connections.
- Managed hosts must have Python 2.4 or later installed.
- The python-simplejson package must also be installed on Red Hat Enterprise Linux 5 managed hosts. It is not required on Red Hat Enterprise Linux 6 and 7 managed hosts, since Python 2.5 (and newer versions) provide its functionality by default.

Ansible Deployments

- It can detect various host specific information from “managed hosts”.
- For example, the version of RHEL running, installed applications, subscriptions entitled etc.
- Jboss version and its configuration and subscriptions as well.

Ansible Orchestration

- Can help in finishing provisioning of servers.
- Can do Rolling Upgrades with zero downtime
- Serialization is also possible.
- By default, it does parallel execution of “plays”.

Ansible Connection Plugins

- ***Control Persisit:*** A feature that improves Ansible performance by eliminating SSH connection overhead, when multiple SSH commands are executed in succession.
- RHEL7 uses default ssh which have “ControlPersist” feature.
- ***Paramiko:*** is a python implementation of ssh with ControlPersist feature, can be used on RHEL6.
- ***Local:*** this plugin is used to connect and manage “control node” itself.
- ***winrm:*** connection plugin used for managing windows machines.
- Docker: Ansible 2, we have docker plugin as well, which can connect from Docker host to containers.

Ansible Inventories

- ***Static Inventory:*** defined in an INI-like text file, in which each section defines one group of hosts (a host group)
- ***Dynamic Inventory:*** dynamically generated from various sources. public/private cloud providers, Cobbler system information, an LDAP database or CMDB.
- ***/etc/ansible/hosts*** is the default inventory

- Inventories can also define how to connect to managed hosts.

Example Inventory

```
[webservers]
localhost  ansible_connection=local
web1.example.com
web2.example.com:1234 ansible_connection=ssh ansible_user=ftaylor
192.168.3.7

[db-servers]
web1.example.com
db1.example.com
```

Wildcard Entries in Inventories

[olympia]
washington[1:2].example.com

[salem]
oregon[01:02].example.com

Defining Group of Group

```
[olympia]
washington1.example.com
washington2.example.com

[salem]
oregon01.example.com
oregon02.example.com

[nwcapitols:children]
olympia
salem
```

Defining Variables in Inventory

```
[webservers]
web1.example.com
web2.example.com:1234 http_port=8080

[webservers:vars]
http_port=80 maxRequestsPerChild=500

[db-servers]
web1.example.com
db1.example.com
```

Chapter – 2

Installing Ansible
&
Ansible CLI

Control Node

- Pre-requisites
  - Python 2.6
- Ansible is required to be installed only on control node, unlike Puppet of Chef.
- Python3 is not used by ansible as of now.
- Ansible is not a part of RHEL repo (as of now). Try GitHub or EPEL Repos.
- Officially we can get ansible from www.ansible.com

```
[root@controlnode ~]# yum list installed python
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
python.x86_64     2.7.5-34.el7     installed
```

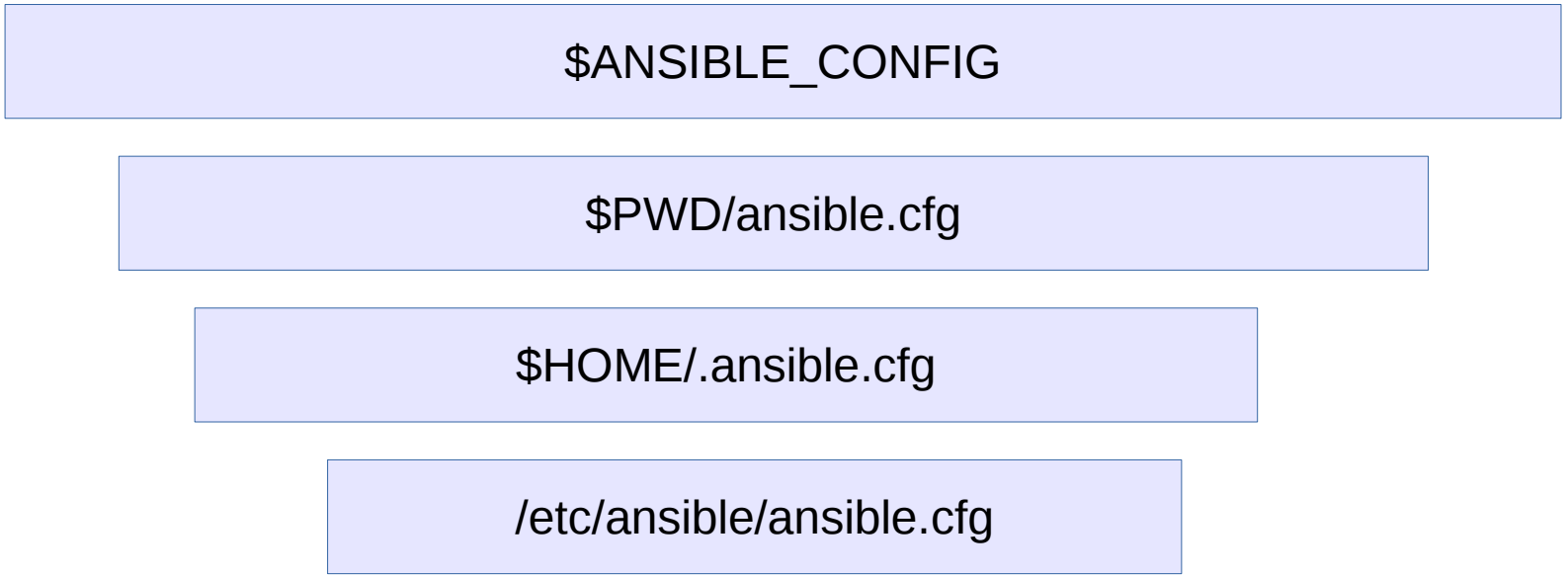
Installation

- Once you have the repo, you can simply use “yum” to install ansible.

  [root@controlnode ~] # yum install ansible

- Testing for installation can be done as simple as running a simple help command for ansible.

  [student@controlnode ~] $ ansible -h


$ANSIBLE_CONFIG
$PWD/ansible.cfg
$HOME/.ansible.cfg
/etc/ansible/ansible.cfg

Before – Doing Actual Things !!

- Before start doing actual things some setup has to be prepared.
- Remote User should be planned.
- SSH-Key based authentication (for password less auth)
- Either Privileges has to be given to the user or not.
- If yes, privilege escalation will be passwordless ?

Referencing Inventory Hosts

- Firstly, we should validate the configuration that we have done.
- We can do that by referencing hosts in our inventory.
- $ ansible --version
- $ ansible <host-pattern> -i /path/to/inventory/file [options]

Example: Inventory

```
[student@controlnode ~]$ cat myinventory
web.example.com
10.1.1.254

[lab]
labhost1.example.com
labhost2.example.com

[test]
test1.example.com
test2.example.com

[datacenter1]
labhost1.example.com
test1.example.com

[datacenter2]
labhost2.example.com
test2.example.com

[datacenter:children]
datacenter1
datacenter2

[new]
192.168.2.1
192.168.2.2

[student@controlnode ~]$ ansible web.example.com -i myinventory --list-hosts
  hosts (1):
    web.example.com
```

```
[student@controlnode ~]$ ansible 192.168.2.1 -i myinventory --list-hosts
  hosts (1):
    192.168.2.1
```

```
[student@controlnode ~]$ ansible labhost1.example.com,test,192.168.2.2 -i myinventory --
list-hosts
  hosts (4):
    labhost1.example.com
    test1.example.com
    test2.example.com
    192.168.2.2
```

```
[student@controlnode ~]$ ansible lab -i myinventory --list-hosts
  hosts (2):
    labhost1.example.com
    labhost2.example.com
```

```
[student@controlnode ~]$ ansible all -i myinventory --list-hosts
  hosts (6):
    labhost1.example.com
    test1.example.com
    labhost2.example.com
    test2.example.com
    192.168.2.1
    192.168.2.2
```

Running Ad-hoc Command

Basic Syntax

$ ansible host-pattern -m module [-a 'module arguments'] [-i inventory]

Example:

$ ansible host-pattern -m command -a “ls /etc” -i inventory

Things behind Ad-hoc commands

- Ansible config file is consulted.
- Default Module to use ?
- Inventory / Hosts to manage ?
- Connection Parameters ?
- Privilege Escalation

Default Module

- Default module to use is configured in ansible configuration in ***“defaults”*** section using ***“module_name”*** parameter. ***“command”*** is the default module used.

```
[defaults]
module_name = command
```

Inventory & Connection Parameters

- You can use default inventory as per the ansible configuration.
- Else, you can choose a custom inventory to be used for selecting hosts using “-i” parameter with ansible CLI.
- Connection parameters can also be defined in inventory as well !!

  [webserver]

  web1.example.com ansible_connection=ssh

Connection Parameters...

- While connecting to a remote host, a user has to be used to connect as.
- This user can be either specified on ansible CLI by using **“-u”** parameter.
- Else, can be configured in ansible configuration in ***“defaults”*** section using ***“remote_user”*** parameter.

  [defaults]

  remote_user = devops

Privilege Escalation

- While connecting to remote hosts using “remote_user” you can do all tasks that “remote_user” can do.
- What if you want to do a task which “remote_user” is not having privileges to.
- In such cases, we can use “privilege escalation” to escalate the privileges of “remote_user”.
- Setting for privilege escalation can be defined on cli using ***“--become”, “--become-user”, “--become-method”, “--ask-become-pass”***

Privilege Escalation...

- These settings can be defined persistently in ansible.cfg as well under “privilege_escalation” section.

  [privilege_escalation]

  become = True

  become_method = sudo

  become_user = root

  become_ask_pass= False

Chapter-3

- Ansible playbooks are written using YAML language.
- YAML Ain't Markup Language
- YAML is designed primarily for representation of data structures such as list and associative arrays in an easy to write human-readable format.
- YAML files optionally start with three hyphens and end with three dots.
- Indent is very important
- Tab and space is different. Space provided by tabs is different for different editors

- Dictionaries

  Key/value data pairs used in YAML are also referred to as dictionaries, hashes,or associative arrays.

  Key: value

  - ---

    name: Automation using Ansible

    code: DO407

    - Other representations

      { name: Automation using Ansible, code:DO407 }

- LISTS

  Like arrays in other programming languages

  – Single dash followed by a space

  - red
  - green
  - blue

  - Other representation

    fruits:

      [red, green, blue]

- Comments -#
- Verifying syntax using python

  python -c 'import yaml, sys; print yaml.load(sys.stdin)' < yaml file

IMPLEMENTING ANSIBLE PLAYBOOK

- Adhoc commands for simple task, one at a time,not for complex configuration and orchestration methods.
- Playbooks are files which describe the desired configurations or procedural steps to implement on managed hosts.
- Playbooks offer a powerful and flexible solution for configuration management and deployment.

Ansible Playbook

- Playbooks are text files written in YAML format.
- Ansible playbook contain plays.
- A play defines set of operations to performed on managed hosts.This operations are called tasks and managed hosts are reffered to as hosts.
- Tasks are performed by Ansible modules.

IDEMPOTENT

- If a playbook is run once to put the hosts in the correct state, the playbook should be written so it is safe to run it a second time and it should make no further changes to the configured systems.
- A playbook with this property is idempotent.
- Most Ansible modules are idempotent, so it is relatively easy to ensure this is true.

Writing a Playbook

- Within a playbook file, plays are expressed in a list context where each play is defined using a YAML dictionary data structure.
- Therefore, the beginning of each play begins with a single dash followed by a space.
- Within a play, administrators can define various attributes. An attribute definition consists of the attribute name followed by a colon and a space (:) and then the attribute value.

```
-
    attribute: value
```

Attributes

- Name attribute

  The name attribute can be used to give a descriptive label to a play. This is an optional attribute, but it is a recommended practice to name all tasks with a label. This is especially useful when a playbook contains multiple plays.

- name: my first play

Hosts attribute

- The set of managed hosts to perform tasks on is defined using the hosts attribute. Host pattern is also possible.
- host: managedhost.example.com
- User attributes,Privilege escalation attributes.
- remote_user: remoteuser
- become: True/False
- become_method: sudo
- become_user: privilege_user

Task attribute

- The main component of a play, the operations to be executed on the managed hosts, are defined using the tasks attribute.
- This attribute is defined as a list of dictionaries.
- Each task in the tasks list is composed of a set of key/value pairs.

```
tasks:
    - name: first task
      service: name=httpd enabled=true
```

  - The dash in the first entry marks the beginning of the list of attributes which pertain to the task.
  - The second entry is space-indented to the same indentation level as the first entry to convey that the two attributes refer to the same parent task.

Playbook formatting

- Single line/Dictionary Formatting
- Multiline formatting

- Blocks
  - Blocks can be used to group related tasks together. This not only improves readability but also allows task parameters to be performed on a block level when writing more complex playbooks.

Multiple Plays

- A playbook can contain one or more plays.
- Example – refer
- Executing a play book.

  ansible-playbook file.yml
- Syntax verification.

  ansible-playbook –syntax-check file.yml
- Executing a dry run - [This causes Ansible to report what changes would have occurred if the playbook were executed, but does not make any actual changes to managed hosts.]

  ansible-playbook -C file.yml
- Step-by-step execution --step

Chapter-4
Managing Variables and Inclusions

Global Scope {Command-line,ansible-config}

Play Scope {Variables set in a play}

Host Scope { Variables set for a particular Host }

Managing Facts

- Ansible facts are variables that are automatically discovered by Ansible from a managed host.
- Facts are pulled by the setup module and contain useful information stored into variables that administrators can reuse.
- Ansible facts can be part of playbooks, in conditionals, loops, or any other dynamic statement that depends on a value for a managed host; For example:
- A server can be restarted depending on the current kernel version.
- The MySQL configuration file can be customized depending on the available memory.
- Users can be created depending on the host name.
- Demo

Custom Facts

- Administrators can create their own facts and push them to a managed node. Upon creation, the custom facts will be integrated and read by the setup module.
- Custom facts can be used for:

  Defining a particular value for a system based on a custom script.

  Defining a value based on a program execution.
- Custom facts are found by Ansible if the facts file is saved in the /etc/ansible/facts.d directory.
- Files must have .fact as an extension.
- A facts file is a plain-text file in INI or JSON format.

Managing Inclusions

- When working with complex or long playbooks, administrators can use separate files to divide tasks and lists of variables into smaller pieces for easier management.
- There are multiple ways to include task files and variables in a playbook.
- include, include_vars,var_files, group_vars, host_vars

Ansible Handlers

- Handlers are tasks that respond to a notification triggered by other tasks.
- Each handler has a globally-unique name, and is triggered at the end of a block of tasks in a playbook.
- If no task notifies the handler by name, it will not run.
- If one or more tasks notify the handler, it will run exactly once after all other tasks in the play have completed.
- Handlers can be seen as inactive tasks that only get triggered when explicitly invoked using a notify statement.
- Handlers are always run in the order in which the handlers section is written in the play, not in the order in which they are listed by the notify statement on a particular task.

- Handlers defined inside an include can not be notified.
- If a task that includes a notify does not execute

  the handler will not be notified. The handler will be skipped unless another task notifies it.
- Ansible notifies handlers only if the task acquires the CHANGED status.

Handling Errors

- Ignore the failed task: By default, if a task fails, the play is aborted; however, this behavior can be overridden by skipping failed tasks.
- Force execution of handlers: By default, if a task that notifies a handler fails, the handler will be skipped as well. Override this behaviour by using force_handlers
- Override the failed state: A task itself can succeed, yet administrators may want to mark the task as failed based on specific criteria.
- Override the changed state: When a task updates a managed host, it acquires the changed state; however, if a task does not perform any change on the managed node, handlers are skipped. The changed_when keyword can be used to override the default behavior of what triggers the changed state.

Ansible Blocks and Error Handling

- Block: Define the main tasks to run
- Rescue: Defines the tasks that will run if the tasks defined in the block clause fails.
- Always: Defines the tasks that will always run independently of the success or failure of tasks defined in the block and rescue clauses

Implementing JINJA2 Templates

- Ansible uses the Jinja2 templating system to modify files before they are distributed to managed hosts.
- Generally speaking, it is preferable to avoid modifying configuration files through logic in templates.
- Ansible allows Jinja2 loops and conditionals to be used in templates, but they are not allowed in playbooks.

Using Jinja2 Templates in Playbooks

- Jinja2 templates are a powerful tool to customize configuration files to be deployed on the managed hosts.
- When the Jinja2 template for a configuration file has been created, it can be deployed to the managed hosts using the template module, which supports the transfer of a local file in the control node to the managed hosts.
- To use the template module, use the following syntax. The value associated with the src key specifies the source Jinja2 template, and the value associated to the dest key specifies the file to be created on the destination hosts.

Implementing Roles in ANSIBLE

The Real Problem

- Data centers have a variety of different types of hosts.
- Web servers, Database servers, and others can have software development tools installed and configured on them.
- An Ansible playbook, with tasks and handlers to handle all of these cases, would become large and complex over time.

What is Role in Ansible

- Ansible roles allow administrators to organize their playbooks into separate, smaller playbooks and files.
- Roles provide Ansible with a way to load tasks, handlers, and variables from external files.

Benefits of Roles

- Roles group content, allowing easy sharing of code with others.
- Roles make larger projects more manageable .
- Roles can be written that define the essential elements of a system type: web server, database, git repository or other purpose.
- Roles can be developed in parallel by different admins.

Requirements of ROLE

- Static files and templates can also be associated and referenced by a role.
- The files that define a role have specific names and are organized in a rigid directory structure

The following **tree** command displays the directory structure of the **user.example** role.

```
[user@host roles]$ tree user.example
user.example/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
```



```
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Ansible role subdirectories

Subdirectory	Function
`defaults`	The **`main.yml`** file in this directory contains the default values of role variables that can be overwritten when the role is used.
`files`	This directory contains static files that are referenced by role tasks.
`handlers`	The **`main.yml`** file in this directory contains the role's handler definitions.
`meta`	The **`main.yml`** file in this directory contains information about the role, including author, license, platforms, and optional role dependencies.
`tasks`	The **`main.yml`** file in this directory contains the role's task definitions.
`templates`	This directory contains Jinja2 templates that are referenced by role tasks.
`tests`	This directory can contain an inventory and **`test.yml`** playbook that can be used to test the role.
`vars`	The **`main.yml`** file in this directory defines the role's variable values.

Deploying Roles with Ansible Galaxy

- Ansible Galaxy [https://galaxy.ansible.com] is a public library of Ansible roles written by a variety of Ansible administrators and users.
- It is an archive that contains thousands of Ansible roles and it has a searchable database that helps Ansible users identify roles that might help them accomplish an administrative task.
- Ansible Galaxy includes links to documentation and videos for new Ansible users and role developers.

Implementing Ansible Vault

- Ansible may need access to sensitive data such as passwords or API keys in order to configure remote servers.
- Normally, this information might be stored as plain text in inventory variables or other Ansible files.
- But in that case, any user with access to the Ansible files or a version control system which stores the Ansible files would have access to this sensitive data.
- This poses an obvious security risk.

Secure Methods

- Theere are two primary ways to store this data more securely:
- Use Ansible Vault, which is included with Ansible and can encrypt and decrypt any structured data file used by Ansible.
- Use a third-party key management service to store the data in the cloud, such as Vault by HashiCorp, Amazon's AWS Key Management Service, or Microsoft Azure Key Vault.

- Command line utility ansible-vault

ANSIBLE TOWER

- Ansible Tower is a web-based UI that provides an enterprise solution for IT automation.
- It has a user-friendly dashboard to manage deployments and monitor resources.
- Ansible Tower complements Ansible, adding automation, visual management, and monitoring capabilities.
- Tower provides user access control to administrators.

- Graphically managed of inventory
- Synchronized with a wide variety of cloud sources, including Red Hat OpenStack Platform.
- Tower's dashboard is the user's entry point.
- Automation capabilities allow the implementation of continuous delivery and configuration management, while integrating their management in one tool.

- Tower's System Tracking
- Tower's Remote Command Execution
- Tower's role-based access control engine

ANSIBLE TOWER ARCHITECTURE

- Ansible Tower is based on a Django web application that requires that Ansible be installed.
- It also relies on a database back-end that uses PostgreSQL.

Ansible Tower-Installtion scenarios

- Depending on the scale of deployment and number of hosts to be managed
- Single machine installation with an internal database
- Single machine installation with an external database
- Active/Passive redundancy using multiple machines

- Tower does not support all the functionality provided by the Ansible CLI, including playbook development support, where
- Ansible's CLI is the best option.
- A RESTful API which supports all the functionality provided through the Tower's dashboard.
- A CLI supporting this RESTful API is also provided by Tower.


Introduction to Vagrant

What is Vagrant?

- A tool to build development environments based on virtual machines
- Focused to create environments that are similar as possible or identical with production servers
- Created by Mitchell Hashimoto
- Written in Ruby
- Initially builted on top of VirtualBox API, today offers VMWare Fusion support (as $79 per licence)

VAGRANT COMPONENTS- VAGRANT, BOX, PROVIDER

- VAGRANT
  - Software automates the build and configuration of virtual machines.
  - A command line interface is provided to administrators for the management of Vagrant projects.

VAGRANT COMPONENT-2

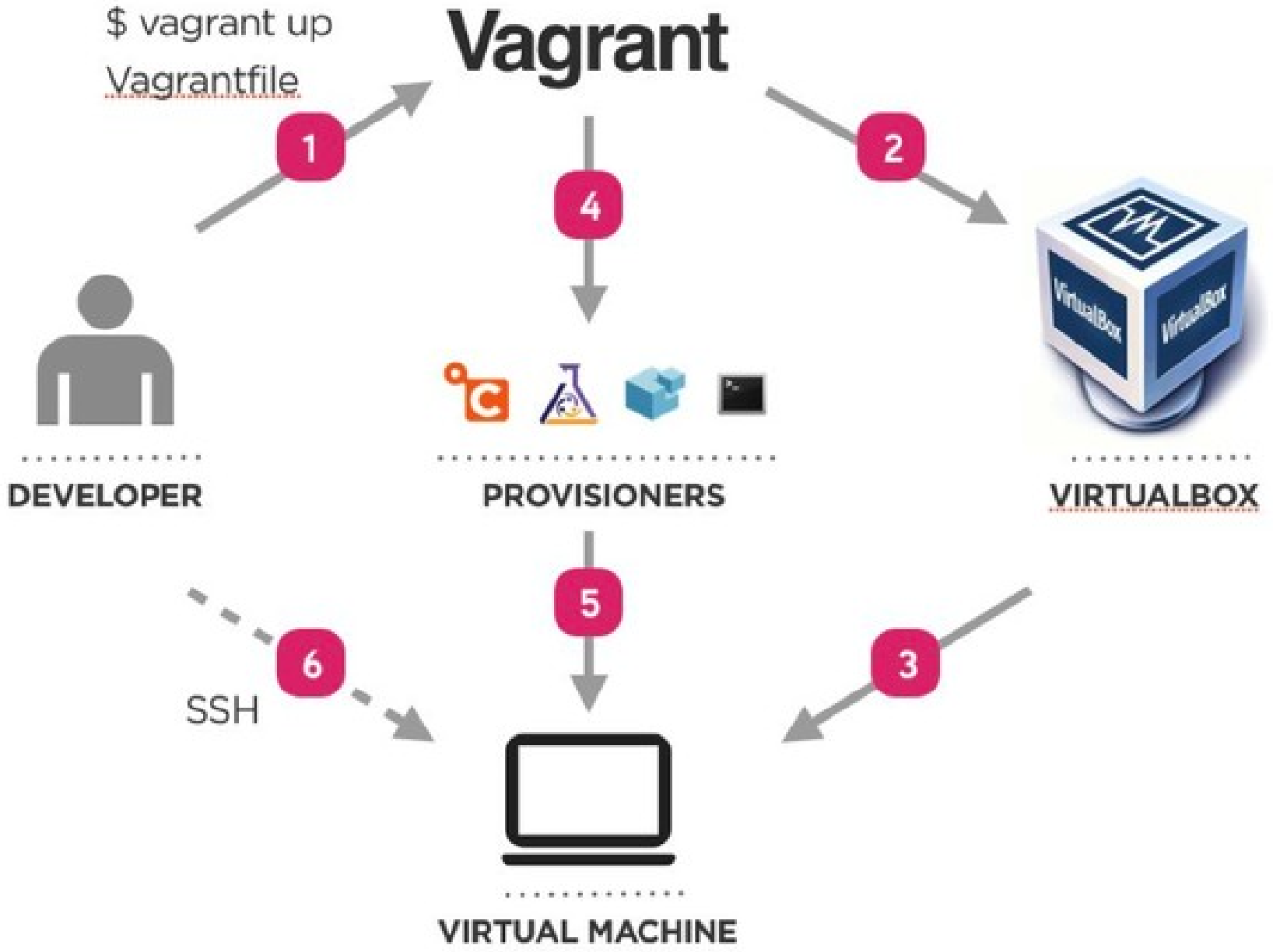
- BOX
  - A box is a tar file that contains a virtual machine image.
  - A box file serves as the foundation of a Vagrant virtualized environment and is used to create virtual machine instances.
  - For greater flexibility, the image should contain just a base operating system installation to be used as a starting point for creating different virtual machines regardless of the specific requirements of their applications.
  - Application-specific requirements can be fulfilled through automated configuration after the virtual machine is created.

VAGRANT COMPONENT-3

- PROVIDER
  - A provider allows Vagrant to interface with the underlying platform that a Vagrant box image is deployed on.
  - Vagrant comes packaged with a provider for Oracle's VirtualBox.
  - Alternative providers are currently available for other virtualization platforms such as Vmware, Hyper-V, and KVM.

VAGRANT AND HYPERVISORS

Vagrant	Hypervisors
Is a CLI interface	can or cannot employ a CLI interface
Needs a level 2 Hypervisor to create VM s	It can create VMs and has all the tools in it
Needs a host OS to run	Can be the host OS


$ vagrant up
Vagrantfile
Vagrant
1
2
3
4
5
6
SSH
DEVELOPER
PROVISIONERS
VIRTUALBOX
VIRTUAL MACHINE

How I create a environment?

- Inside your project, create a Vagrantfile:

```
$ vagrant init <your box name>
```

How I add a box?

- Great box repository: www.vagrantbox.es
- Run this command:

```
$ vagrant box add <name> <url> <provider> # virtualbox
```

How I access it?

- You need to set forwarding ports between guest and host to work (bind on 0.0.0.0!)
- Just add the following code in your Vagrantfile, restart server and access in browser:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  # ...

  config.vm.network :forwarded_port, guest: 3000, host: 3000

  # ...
end
```

How I customize it?

- You can change memory, CPU cores and other things in Vagrantfile
- Just see VBoxManage options
- Example:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  # ...

  config.vm.provider :virtualbox do |vb|
    vb.customize [ 'modifyvm', :id, '--memory', '1024' ]
    vb.customize [ 'modifyvm', :id, '--cpus',   '4'    ]
  end

  # ...
end
```