

# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



FASTER  
SMARTER  
SAFER

# About me...



- Bogdan Sass
- Trainer for Bittnet Systems Romania

# About you...



# Audience & Prerequisites

- **Training Audience**

- This course is intended for beginners who are interested in developing their skills with ***Ansible***

- **Prerequisites:**

- Familiarity with the Linux CLI and Linux command-line text editors
- Familiarity with YAML





# Ansible Basic Course Agenda

## Day 1

1. Introduction to Ansible

2. Running Ad Hoc Commands

3. Preparing hosts for Ansible

4. Basic Playbooks

## Day 2

5. Facts. Variables

6. Loops and Conditions

7. Working with Templates

8. Advanced topics

# Remember

- Daily schedule
  - Breaks
  - Lunch break
- Questions
- SSH over NAT - timeout
- ~~Mobile phones~~



Solutions for training the world.



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

The Lab Environment

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Pod breakdown</i>	2
<i>Accessing the pods</i>	2
<i>Tips and Tricks on Pod navigation</i>	4
<i>List of servers</i>	4



# Lab Overview and objectives

The purpose of this lab is to get yourself connected and familiarized with the workshop's virtual environment setup which you will use to practice the acquired knowledge.

## Pod breakdown

For each of the students participating in the class the instructor has assigned a „pod” which holds 3 machines as follows:

1. Ansible server
  - Hostname: **ansible-XX-01-hivemaster**
  - Running OS: Ubuntu 18.04. LTS
2. Ubuntu server
  - Hostname: **ansible-XX-02-ubuntu**
  - Running OS: Ubuntu 18.04. LTS
3. Centos server
  - Hostname: **ansible-XX-03-centos**
  - Running OS: CentOS 7

## Accessing the pods

The recommended way of accessing the servers is directly via SSH.

Depending on your current workstation OS one way of accessing the pod is as follows:

1. **If OS == Linux or OS == macOS:**

Best way to access the pods is to use the `ssh` command with a custom SSH port which will be provided by the course instructor.

The command to do this looks like:

```
# ssh -p 22 <username>@<host>
```

**Note:** On first login you will be receive a warning that the host authenticity cannot be established and you will be prompted to accept or deny the connection. We suggest you blindly accept!

Here is an output example of a successful connection:

```
~$ ssh -p 22 user@192.168.1.1
The authenticity of host '192.168.1.1 (192.168.1.1)' can't be
established.
ECDSA key fingerprint is
SHA256:4MqnLw0pJvsD91ZomYIeli+9CYCkKoG5JShJWFt8JaI.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '192.168.1.1' (ECDSA) to the list of known
hosts.
user@192.168.1.1's password:

Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-58-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of Sat Sep  40 21:17:27 UTC 2040

System load:  0.0                Processes:            101
Usage of /:   43.8% of 9.78GB    Users logged in:    1
Memory usage: 20%               IP address for enp0s3: 1.1.1.1
Swap usage:   0%                IP address for enp0s8: 192.168.1.1

30 packages can be updated.
0 updates are security updates.

Last login: Sat Sep  40 21:16:11 2040 from 8.8.8.8
user@master:~$
```

## 2. If OS == Windows:

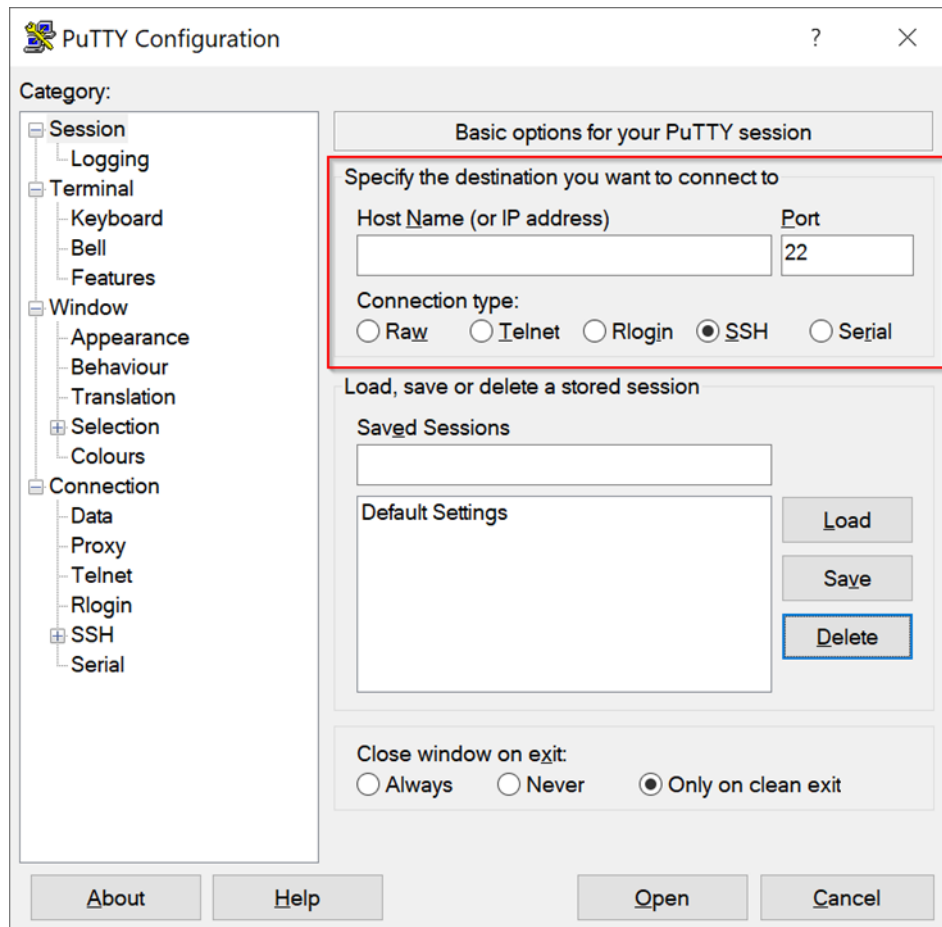
Windows does not have an SSH client out of the box.

You can use any SSH client you prefer - there are many options available. Several examples include: PuTTY, SecureCRT, MobaXterm, MremoteNG, etc  
For this guide, we will use **PuTTY** as an example.

We will first need to download putty and the quickest way to do so is from this link:

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

Once you download the application, you can run it directly (no installation needed). You will see the following window.



The Host Name section will hold the hostname of the machine (it can be an IP address or a domain name / hostname) while the Port section will hold the custom port of the SSH session as described by your instructor.

Once you have the details filled in click on „Open” and you will be greeted by a user/password prompt. You can find the credentials below.

If for any task you need root privileges, you can use `sudo -i` to become root.

## Tips and Tricks on Pod navigation

Each server has a private IP address, connected to the lab network. All the servers can talk to one another directly using the private IP addresses.

Also, key-based SSH authentication has been configured between servers in the same pod. When logged in on one server, you should be able to jump to the other servers by simply doing `ssh servername`. For the server name, you can use the name (as shown in the table below), or simply "srv01", "srv02" (... etc)

## List of servers

The names of the servers are of the form:

**ansible-XX-YY-details**

where:

- XX = user ID
- YY = server number

The servers can be accessed via SSH using the following IP and ports:

**labs.sass.ro**

For example, if your user ID is 01 and you want to access server **01** (the hivemaster), you would connect to:

**labs.sass.ro:20403**

Login credentials:

Username: **student**

Password: **Ansible123\$**

Remember - once you are connected to one of the servers in your pod, you can jump to the others by simply doing an `ssh srv02` or `ssh srv03`!

User ID	Srv01 Port	Srv03 Port	Srv03 Port
01	20403	20404	20405
02	20406	20407	20408
03	20409	20410	20411
04	20412	20413	20414
05	20415	20416	20417
06	20418	20419	20420
07	20421	20422	20423
08	20424	20425	20426
09	20427	20428	20429
10	20430	20431	20432

### Alternative access

If you are facing some issues with accessing the infrastructure using direct SSH connection, you can also use the SSH Proxy provided at <http://ssh.labs.sass.ro>.

So, open your favorite browser and go to <http://ssh.labs.sass.ro> then login using the following credentials:

Username: **ansible-basic-user-XX**

Password: **Ansible123\$**

! Make sure to replace XX with your own “User ID” 😊

Now, you should have access to all your 3 servers.



# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



FASTER  
SMARTER  
SAFER

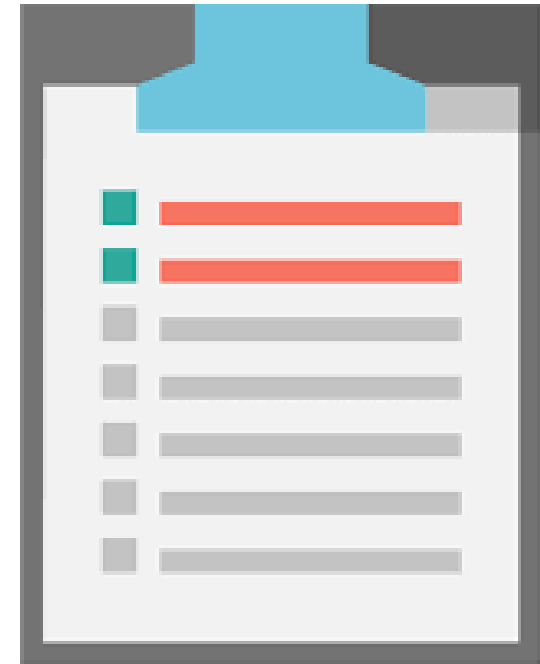


# 1. Introduction to Ansible



# Topics covered:

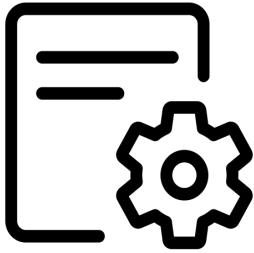
- What is Ansible
- Ansible history
- Installing Ansible



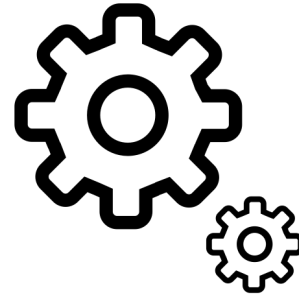
# What is Ansible?

- Ansible is an automation tool, used for tasks such as configuration management, application deployment and provisioning.
- Ansible allows you to create machines, describe how these machines should be configured and/or what actions should be taken on them.

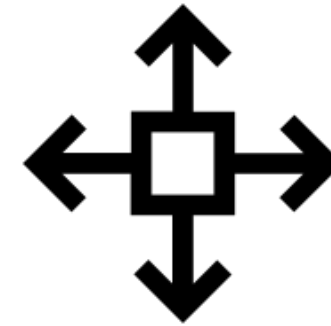
# What is Ansible?



Provisioning



Configuration  
Management



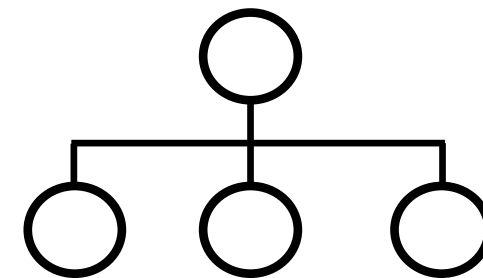
Application  
Deployment



Continuous  
Delivery



Security &  
Compliance



Orchestration

# Why Ansible?



Ansible uses an extraordinarily **simple** syntax written in YAML.



Ansible has **powerful** features that can enable you to model even the most complex IT workflows (infrastructure, networks, OS and services)



# Why Ansible?

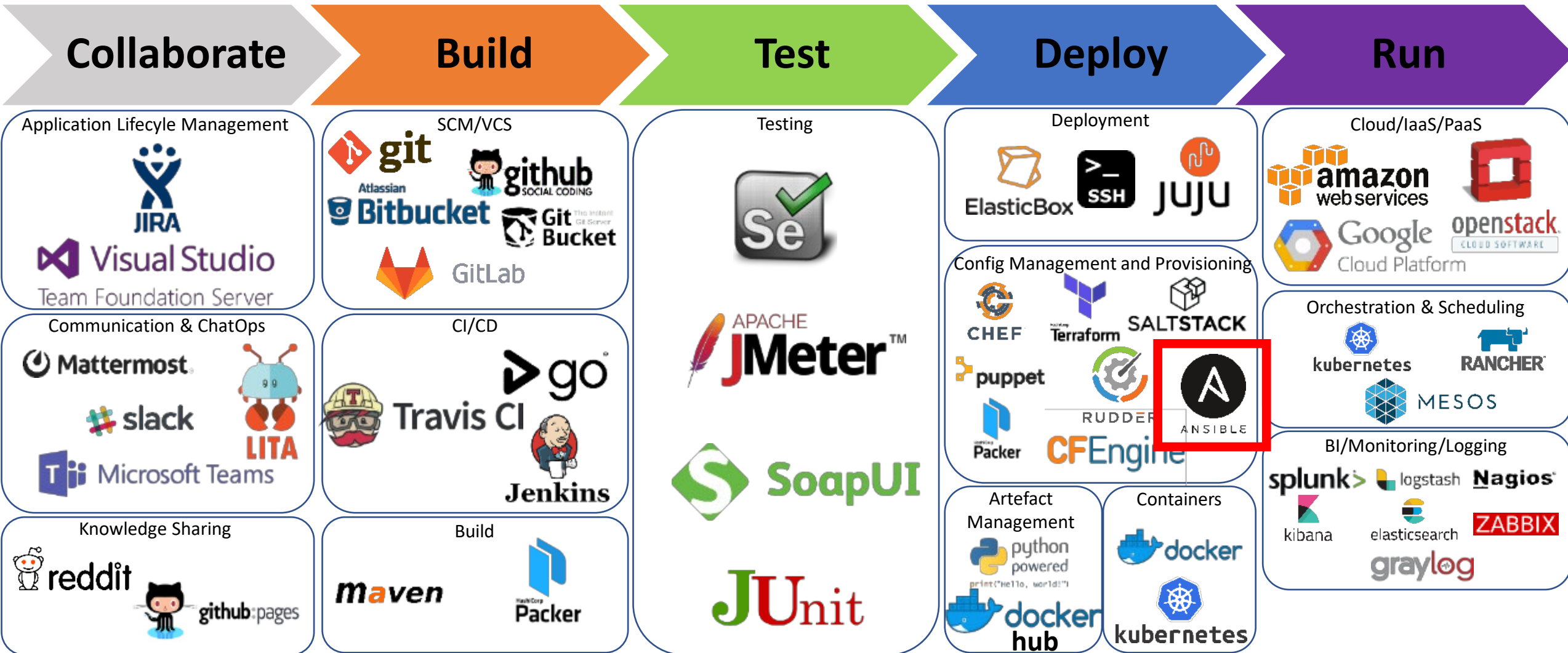


It is completely **agentless**, there are **no agents/software** or additional firewall ports that you need to install on the client systems



**No extra software** on your servers means more resources for your applications.

# Ansible in DevOps



# What Ansible Automates

## CLOUD

AWS  
Azure  
Digital  
Ocean  
Google  
OpenStack  
Rackspace  
**+more**

## OPERATING SYSTEMS

RHEL and  
Linux  
UNIX  
Windows  
**+more**

## VIRT & CONTAINER

Docker  
VMware  
RHV  
OpenStack  
OpenShift  
**+more**

## STORAGE

NetApp  
Red Hat  
Storage  
Infinidat  
**+more**

## WINDOWS

ACLs  
Files  
Packages  
IIS  
Regedit  
Shares  
Services  
Configs  
Users  
Domains  
**+more**

## NETWORK

Arista  
A10  
Cumulus  
Bigswitch  
Cisco  
Cumulus  
Dell  
F5  
Juniper  
Palo Alto  
OpenSwitch  
**+more**

## DEVOPS

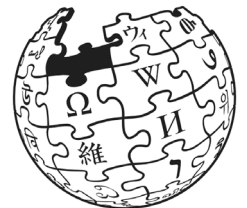
Jira  
GitHub  
Vagrant  
Jenkins  
Bamboo  
Atlassian  
Subversion  
Slack  
Hipchat  
**+more**

## MONITORING

Dynatrace  
Airbrake  
BigPanda  
Datadog  
LogicMonitor  
Nagios  
New Relic  
PagerDuty  
Sensu  
StackDriver  
Zabbix  
**+more**

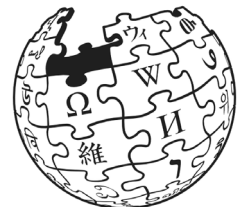
# Ansible History

- The term "**ansible**" was coined by **Ursula K. Le Guin** in her 1966 novel *Rocannon's World*, and refers to fictional instantaneous communication system.
- The Ansible tool **was developed** by **Michael DeHaan**, the author of the provisioning server application Cobbler.



# Ansible History

- Ansible, Inc. (originally AnsibleWorks, Inc.) was the company set up to commercially support and sponsor Ansible.
- Red Hat acquired Ansible in October 2015.
- Ansible is included as part of the **Fedora** distribution of Linux, owned by Red Hat.
- Is also **available** for **Red Hat Enterprise Linux, CentOS, Debian, Scientific Linux, Oracle Linux** via Extra Packages for Enterprise Linux (**EPEL**) and Ubuntu as well as for other operating systems.





# Installing Ansible

## 1. Installing prerequisites

```
student:~$ sudo apt update
student:~$ sudo apt -y install software-properties-common
```

## 2. Add Ansible official repository

```
student:~$ sudo apt-add-repository ppa:ansible/ansible
[...]

http://ansible.com/
More info: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Press [ENTER] to continue or Ctrl-c to cancel adding it.
[...]
Fetched 17.0 kB in 1s (17.6 kB/s)
Reading package lists... Done
```

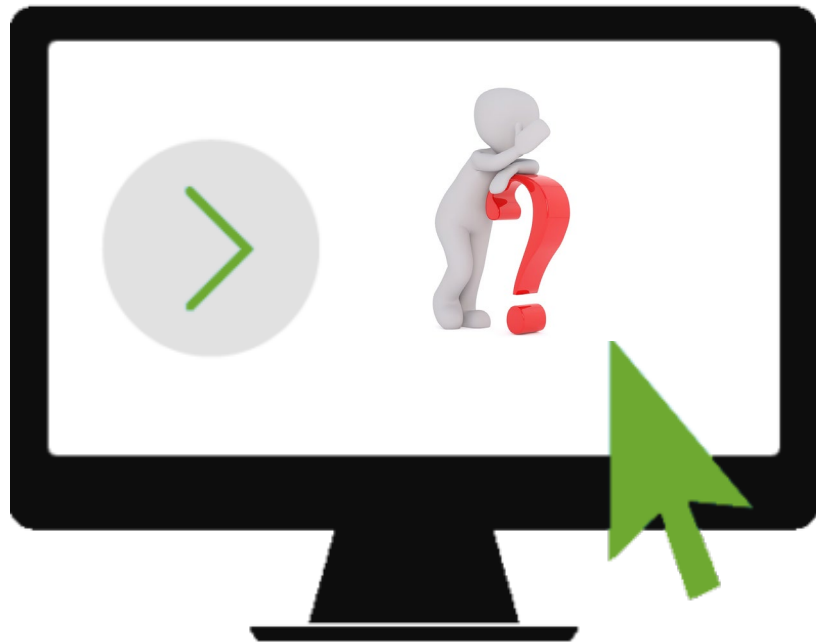
# Installing Ansible

## 3. Install Ansible

```
student:~$ sudo apt install ansible -y
```

## 4. Check Ansible version

```
student:~$ ansible --version
ansible 2.9.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/student/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.15+ (default, Jan 1 2040, 17:39:04) [GCC 7.4.0]
```



# Lab 1: Installing Ansible





Solutions for training the world.



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Installing Ansible

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: Login to Ansible Control Node (hivemaster host)	2
Task 2: Installing prerequisites	2
Task 3: Add Ansible official repository	2
Task 4: Install Ansible	3
Task 5: Check Ansible version	3
Task 6: Explore Ansible command options	3

# Lab Overview and objectives

The purpose of this lab is to perform installation of Ansible on the hivemaster host, in order to be able to fully-manage the other two remote servers automatically.

All of these systems are as close as possible to a fresh install of the system (with the exception of some minor changes, such as creating a "student" user, allowing login using an SSH public key, and disabling firewalld and SELinux).

## Guided Tasks

### Task 1: Login to Ansible Control Node (hivemaster host)

As you already seen in the previous lab, each of you has a pod with 3 machines:

- Ansible server (Control Node) – **ansible-XX-01-hivemaster**
- Ubuntu server (Controlled Node 1) – **ansible-XX-02-ubuntu**
- CentOS server (Controlled Node 2) – **ansible-XX-03-centos**

Assuming that you have already performed the login in the previous lab, you may use that connection, otherwise you can open a new connection to you own **ansible-XX-hivemaster** host.

```
ssh -p 22 user@ansible-XX-hivemaster.vms.sass.ro
```

If the connection is successful, your prompt should look like:

```
student@ansible-00-01-hivemaster:~$
```

### Task 2: Installing prerequisites

Our control node (hivemaster) is running Ubuntu 18.04, and in order to install Ansible we are going to use Ubuntu package manager (APT). Before installing Ansible we have to install some prerequisites and to add Ansible official repository. In order to perform these, we have to run the following command using privilege escalation (using sudo):

```
student@ansible-00-01-hivemaster:~$ sudo apt update
student@ansible-00-01-hivemaster:~$ sudo apt -y install software-properties-common
```

### Task 3: Add Ansible official repository



We are going to add the Ansible official repository using `apt-add-repository` command:

```
student@ansible-00-01-hivemaster:~$ sudo apt-add-repository
ppa:ansible/ansible
[...]

http://ansible.com/
More info: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Press [ENTER] to continue or Ctrl-c to cancel adding it.
[...]
Fetched 17.0 kB in 1s (17.6 kB/s)
Reading package lists... Done
```

Now let's run again `apt update` command:

```
student@ansible-00-01-hivemaster:~$ sudo apt update
```

#### Task 4: Install Ansible

After preparing the host, we can properly install the latest version of Ansible:

```
student@ansible-00-01-hivemaster:~$ sudo apt install ansible -y
```

#### Task 5: Check Ansible version

In order to make sure that we have properly installed the latest version of Ansible (from the repository which we just added) and not an older one from Ubuntu repository, we have to run the following ansible command:

```
student@ansible-00-01-hivemaster:~$ ansible --version
ansible 2.9.1
  config file = /etc/ansible/ansible.cfg
  configured module search path =
[u'/home/student/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-
packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.15+ (default, Jan 1 2040, 17:39:04) [GCC 7.4.0]
```

You Ansible version should be at least (or higher) than the one in the example (2.9.1).

#### Task 6: Explore Ansible command options

As we installed Ansible, let's explore the Ansible command options available, because we are going to use some of them in this training, while others you may need to use in your everyday work:

```
student@ansible-00-01-hivemaster:~$ ansible --help
```

# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



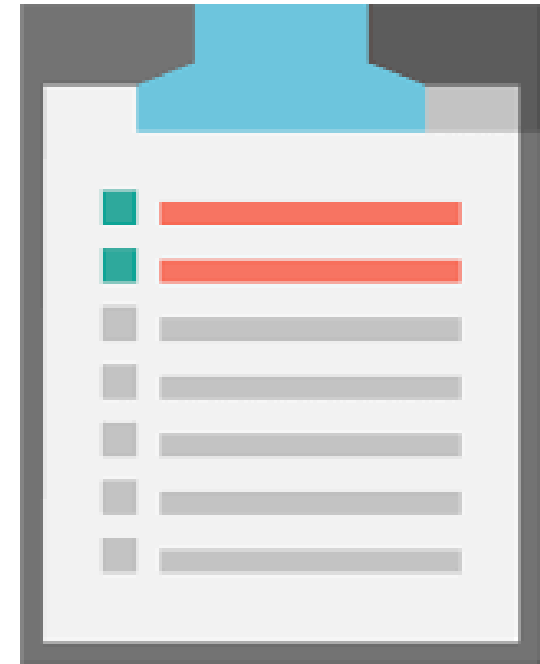
FASTER  
SMARTER  
SAFER

## 2. Running Ad Hoc Commands

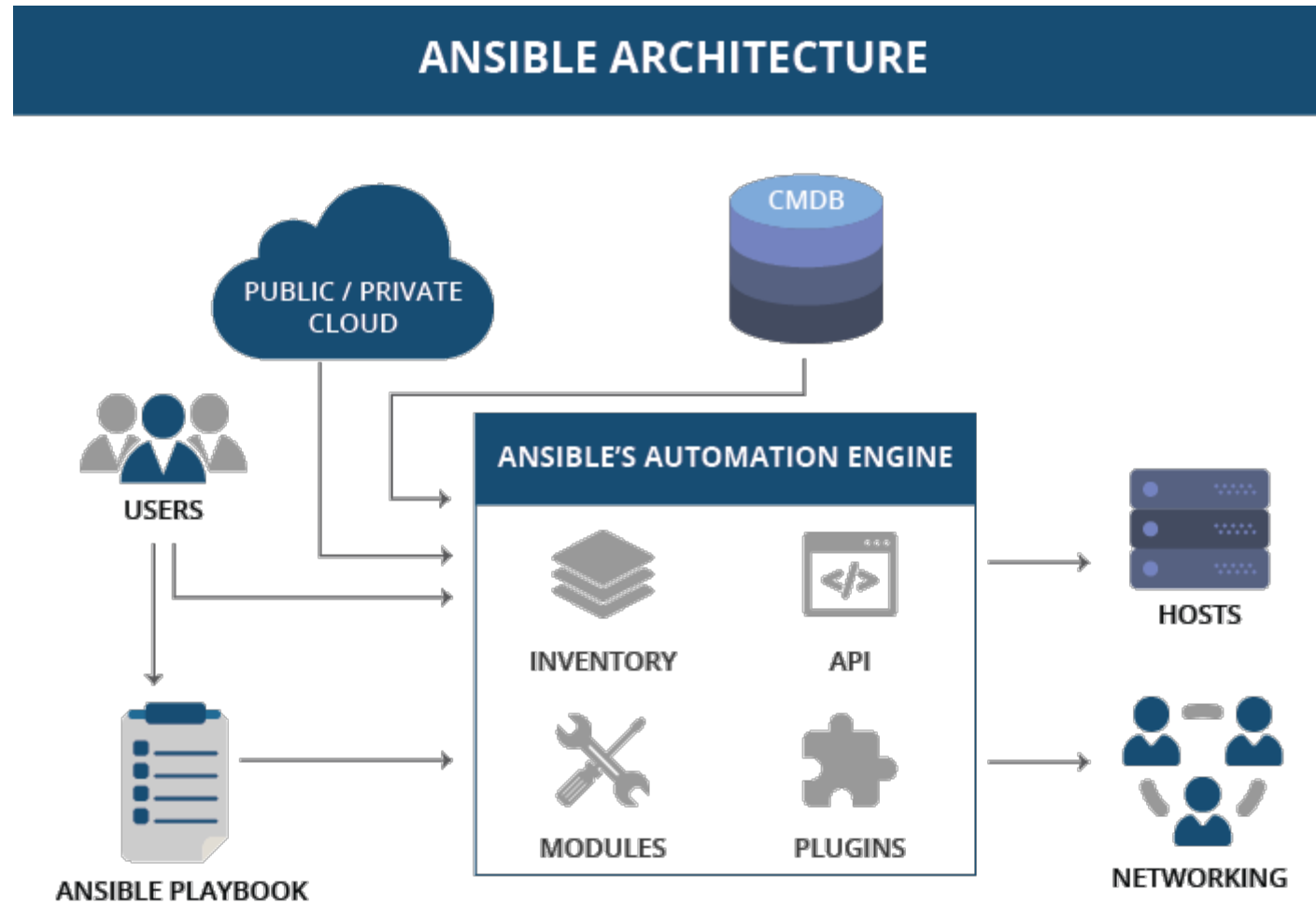


# Topics covered:

- How Ansible works
- Authenticating Ansible Connections



# How Ansible Works



# How Ansible Works (Automation Engine)



INVENTORY

- Inventories are **lists of hosts** ( also called nodes) - servers that **need to be managed**.

- There are two types:
  - Static inventory
  - Dynamic inventory

```
[webservers]
ubuntu
centos
```

```
[dbservers]
ubuntu
hivemaster
```

```
[datacenter:children]
webservers
dbservers
```



# How Ansible Works (Automation Engine)



## MODULES

- Modules are executed directly on remote hosts through playbooks.
- Modules can control system resources, like services, packages, or files, or execute system commands.
- They enable you to manage virtually *everything* that has an API, CLI, or configuration file you can interact with, including network devices like load balancers, switches, firewalls, container orchestrators, containers themselves.



# How Ansible Works



ANSIBLE PLAYBOOK

- Playbooks are simple files written in YAML format which describe the configuration tasks to be applied
- Playbooks can declare configurations, but they can also orchestrate the steps of any manual ordered process.

# How Ansible Works



NETWORKING

- Ansible can also be used to automate different networks.
- It uses a data model that is separate from the Ansible automation engine that easily spans different network hardware.

# How Ansible Works



HOSTS

- The hosts in the Ansible architecture are just node systems which are being configured.
- They can be any kind of machine – Windows, Linux

# How Ansible Works



- CMDB - a repository that acts as a data warehouse for IT installations.
- It holds data relating to a collection of IT assets (configuration items (CI)), as well as relationships between such assets.

# Ansible Configuration File

- Ansible has a number of options that can be adjusted
- The default configuration file: `/etc/ansible/ansible.cfg`
- Notable configuration options include:
  - default inventory file location
  - default remote user
  - default authentication settings

# Ansible Configuration File

- The configuration properties may be overridden using local files.
- The first configuration file found is used **and later files are ignored.**
- Configuration file search order:
  - **ANSIBLE\_CONFIG** (environment variable if set)
  - **ansible.cfg** (in the current directory)
  - **~/.ansible.cfg** (in the home directory)
  - **/etc/ansible/ansible.cfg**

# Ansible Configuration File

- As of version 2.4, Ansible has a new utility called **ansible-config**
- This utility allows users to see:
  - All the configuration setting available
  - Their defaults
  - How to set them
  - Where their current value comes from

# Ansible inventory

- An inventory is a list of hosts that Ansible manages.
- The default Ansible Inventory File is `/etc/ansible/hosts`
- Inventory location may be specified as follows:
  - Default: `/etc/ansible/hosts`
  - On the command line: `ansible -i <filename>`
  - Can be set in `ansible.cfg`



# Ansible inventory

- Inventory files can also use YAML format
- You may want to keep separate inventories for staging and production.

# Ansible inventory file example

```
mail.example.com ansible_port=5556 ansible_host=192.168.0.100
```

```
[webservers]
```

```
httpd1.example.com
```

```
httpd2.example.com
```

```
[labservers]
```

```
lab[02:05]
```

# Ansible inventory file example

this line defines a host (alias)

```
mail.example.com ansible_port=5556 ansible_host=192.168.0.100
```

```
[webservers]
```

```
httpd1.example.com
```

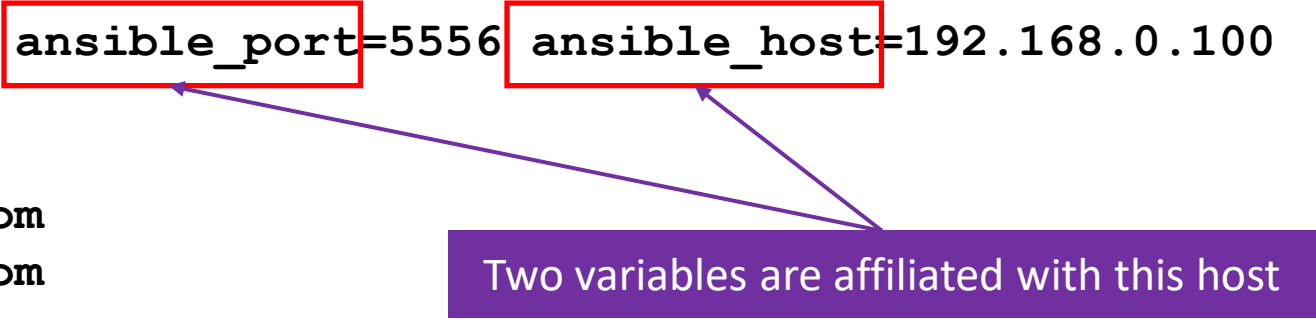
```
httpd2.example.com
```

```
[labservers]
```

```
lab[02:05]
```

# Ansible inventory file example

```
mail.example.com ansible_port=5556 ansible_host=192.168.0.100
```



```
[webservers]  
httpd1.example.com  
httpd2.example.com
```

Two variables are affiliated with this host

```
[labservers]  
lab[02:05]
```

# Ansible inventory file example

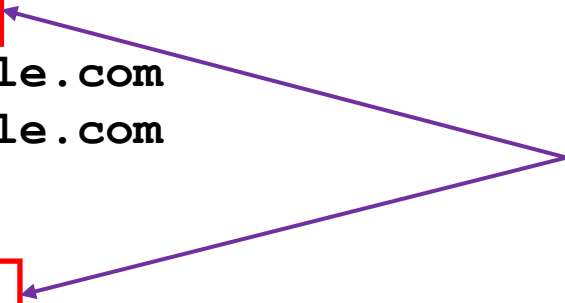
```
mail.example.com ansible_port=5556 ansible_host=192.168.0.100
```

```
[webservers]
```

```
httpd1.example.com  
httpd2.example.com
```

```
[labservers]
```

```
lab[02:05]
```



Two group servers are defined  
(webservers and labservers)

# Ansible inventory file example

```
mail.example.com ansible_port=5556 ansible_host=192.168.0.100
```

```
[webserver]
httpd1.example.com
httpd2.example.com
```

```
[labserver]
lab[02:05]
```

labserver group has 4 hosts defined

the expression lab[02:05] is the same  
as lab02, lab03, lab04, lab05

# Ansible command

- Ansible ad-hoc commands allow us to quickly run a task against a remote system
- Syntax:
  - `ansible <HOST> -b -m <MODULE> -a "<ARG1 ARG2 ARGN>" -f <NUM_FORKS>`

# Ansible command

- Ansible ad-hoc commands analogous to bash commands.

- Syntax:

- `ansible <HOST> -b -m <MODULE> -a "<ARG1 ARG2 ARGN>" -f <NUM_FORKS>`



host or host group defined in  
the inventory file



# Ansible command

- Ansible ad-hoc commands analogous to bash commands.

- Syntax:

- `ansible <HOST> -b -m <MODULE> -a "<ARG1 ARG2 ARGN>" -f <NUM_FORKS>`



- "become"
- Ansible escalates permission to `--become-user` using the method defined by `--become-method`

Default become-user is `root`.  
Default become-method is `sudo`

# Ansible command

- Ansible ad-hoc commands analogous to bash commands.

- Syntax:

- `ansible <HOST> -b -m <MODULE> -a "<ARG1 ARG2 ARGN>" -f <NUM_FORKS>`



-m is for module to use

# Ansible command

- Ansible ad-hoc commands analogous to bash commands.

- Syntax:

- `ansible <HOST> -b -m <MODULE> -a "<ARG1 ARG2 ARGN>" -f <NUM_FORKS>`



-a is for module arguments


Note: If you do not specify "-m", the default module used will be "command"

# Ansible command

- Ansible ad-hoc commands analogous to bash commands.

- Syntax:

- `ansible <HOST> -b -m <MODULE> -a "<ARG1 ARG2 ARGN>" -f <NUM FORKS>`



-f is used to set forks for parallelism, which is how you can have Ansible execute tasks simultaneously on many hosts

# Authenticating Ansible Connections

- You can run ad-hoc commands using:
  - password authentication
  - key-based authentication

# Authenticating Ansible Connections (password)

- To run ansible using password authentication:
  - `-u` for username
  - `--ask-pass` for prompting us to enter the account password

```
student:~$ ansible -i hosts all -m ping -u student --ask-pass
SSH password:
ansible-00-02-ubuntu | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

# Authenticating Ansible Connections (key-based)

- In everyday use, the best practice is to have a public key installed on the remote hosts and it is also recommended to create a dedicated user who should have access using that key.
- The `ssh-keygen` and `ssh-copy-id` command can facilitate creating a pre-shared key for user authentication.

# Authenticating Ansible Connections (key-based)

## 1. Generate a SSH keypair

```
student:~$ ssh-keygen -t rsa -b 2048 -f ansible_key
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ansible_key.
Your public key has been saved in ansible_key.pub.
The key fingerprint is:
SHA256:gm+P8xHI5tl55qCJF88jmDj/9C4Z25rzLTzQsrZmO/4 student@ansible-00-01-hivemaster
The key's randomart image is:
+---[RSA 2048]-----+
|
|
|
|   o .
|  . =.S
|   +++.o
|  . =*&= o
| o o+#BOB
| oo*X&E+o
+-----[SHA256]-----+
```



# Authenticating Ansible Connections (key-based)

## 1. Generate a SSH keypair

- a. The previous command will generate 2 new files in the current directory

```
student:~$ ls | grep ansible  
ansible_key  
ansible_key.pub
```

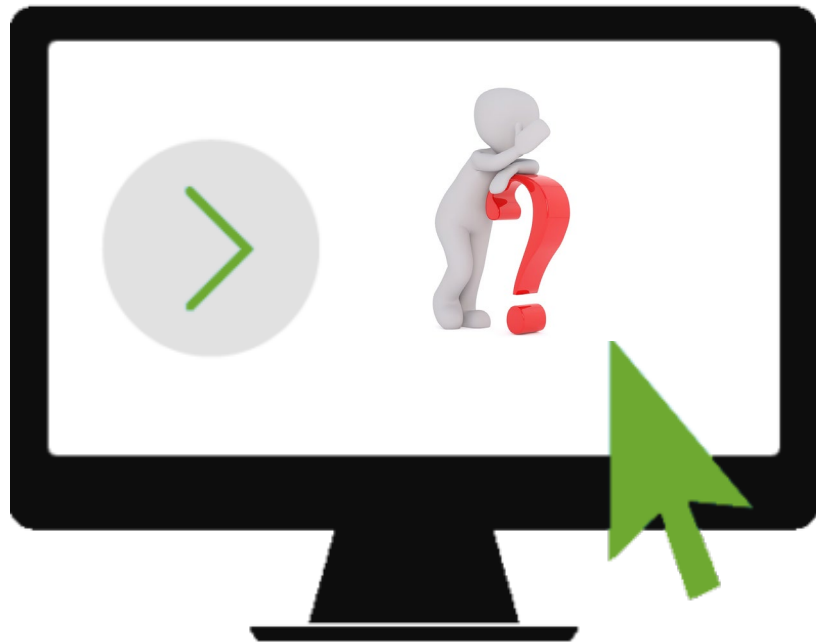
- b. Add the newly-generated public key (ansible\_key.pub) to the remote hosts

```
student:~$ cat ansible_key.pub >> /home/student/.ssh/authorized_keys
```

# Authenticating Ansible Connections (key-based)

## 3. Ping all hosts using key-based auth

```
student:~$ ansible -i hosts all -m ping -u student --private-key  
/home/student/ansible_key  
ansible-00-02-ubuntu | SUCCESS => {  
    "ansible_facts": {  
        "discovered_interpreter_python": "/usr/bin/python3"  
    },  
    "changed": false,  
    "ping": "pong"  
}
```



## Lab 2: Ad-Hoc commands





Solutions for training the world.



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Running Ad-Hoc Commands

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: Testing Ad-Hoc commands on localhost	2
Task 2: Create a basic inventory file	3
Task 3: Running Ad-Hoc commands using password authentication	4
Task 3.1: Ping all hosts – Attempt 1	4
Task 3.2: Ping all hosts – Attempt 2	4
Task 3.3: Fix [DEPRECATION WARNING] for hive master	5
Task 3.4: Test again for hive master	5
Task 3.5: Check the connection user used by Ansible	6
Task 3.6: Accessing system files (privilege escalation)	6
Task 4: Running Ad-Hoc commands using key-based authentication	7
Task 4.1: Generate a SSH keypair	8
Task 4.2: Put publickey into correct place	9
Task 4.3: Ping all hosts (using key-based auth)	10
Task 4.4: Accessing system files (privilege escalation)	10
Task 4.5: Command module	11
Task 4: Setup module in Ad-Hoc commands	11

# Lab Overview and objectives

The purpose of this lab is testing Ansible connection to remote servers using different authentication methods by running Ad-Hoc commands.

Ad-Hoc commands are one line commands that performs one task on the target host. Using Ad-Hoc commands we can explore Ansible itself, without creating a playbook first, but having the opportunity to perform several tasks, such as managing services, rebooting servers, editing configurations, copying files, installing packages and so on.

Basic structure of an Ad-Hoc command:

```
$ ansible <pattern> [options] [module options]
```

Using **patterns** we choose which managed hosts or groups we want to execute against. A pattern can refer to a single host, IP, an inventory group, a set of groups or even all hosts of inventory.

Here are some options available for ansible command:

Option	Description
-i	inventory file (default /etc/ansible/hosts) or comma-separated list
-v	verbose mode (-vvv for more, -vvvv for connection debugging)
-m	module name to execute (default=command)
-a	module arguments
-b	privilege escalation ("become") method to use (default=sudo)
-C	("check") - don't make any changes; instead try to predict some of the changes
-f	number of parallel processes (forks) to use (default=5).

## Guided Tasks

### Task 1: Testing Ad-Hoc commands on localhost

In order to start managing servers we have to tell Ansible what hosts it can manage. The default inventory file is located in `/etc/ansible/hosts` but this file is empty (commented) by default. Before writing



our first ‘lite’ inventory file, we are going to test Ansible on `localhost`, which is “implicit”, so we don’t have to create an inventory file for it.

Let’s run `ansible` command with ‘localhost’ as <pattern> and using ping module:

```
student@ansible-00-01-hivemaster:~$ ansible localhost -m ping
localhost | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

By default, Ansible is trying to run commands as the current user (unless another user is specified). We can see this running “id” command using “raw” module (Ansible should run under `student` user):

```
student@ansible-00-01-hivemaster:~$ ansible localhost -m raw -a "id"
localhost | CHANGED | rc=0 >>
uid=1002(student) gid=1003(student) groups=1003(student)
```

Remember that “raw” module is a little bit different than other modules, as it does not requires that Python should be installed on remote host, so it’s also the tool to perform Python installation in case that it is missing from the (current state of) distribution.

## Task 2: Create a basic inventory file

As we mentioned earlier, `ansible` command requires a <pattern>, to match some specific hosts from the inventory file. For the moment, we are going to create a ‘lite’ inventory file, which should contain only hostnames (or IPs) of our servers. You can find the details of hosts from your own pod in `/etc/hosts`:

```
student@ansible-00-01-hivemaster:~$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      ansible-00-01-hivemaster.training.sass.ro ansible-00-01-hivemaster

# The following lines are desirable for IPv6 capable hosts
::1           localhost ip6-localhost ip6-loopback
ff02::1       ip6-allnodes
ff02::2       ip6-allrouters

10.128.0.48    ansible-00-01-hivemaster.training.sass.ro ansible-00-01-hivemaster ansible-00-01-srv01
10.128.0.49    ansible-00-02-ubuntu.training.sass.ro ansible-00-02-ubuntu ansible-00-02-srv02
10.142.15.213  ansible-00-03-centos.training.sass.ro ansible-00-03-centos ansible-00-03-srv03
```

Let’s create the inventory:

```
student@ansible-00-01-hivemaster:~$ vi hosts
```

```
ansible-00-01-hivemaster
ansible-00-02-ubuntu
10.142.15.213
```

Notice that I added for `hivemaster` and `ubuntu` their hostnames and for `centos` its IP address (just to show you that we can use any of them). We didn't specified any groups for our hosts, but Ansible creates by default `all` group, which contains all the hosts in the inventory.

### Task 3: Running Ad-Hoc commands using password authentication

#### Task 3.1: Ping all hosts – Attempt 1

Now that we have an inventory file, we can start running ad-hoc commands, so let's ping all hosts in the inventory:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m ping
ansible-00-01-hivemaster | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: student@ansible-00-01-hivemaster: Permission denied (publickey,password).",
  "unreachable": true
}
ansible-00-02-ubuntu | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: student@ansible-00-02-ubuntu: Permission denied (publickey,password).",
  "unreachable": true
}
10.142.15.213 | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: student@10.142.15.213: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
  "unreachable": true
}
```

#### Task 3.2: Ping all hosts – Attempt 2

You will notice that Ansible failed to connect to any of the hosts in the inventory. As we mentioned in the first task, Ansible is trying to connect using current user (student), but you can see that it failed for all hosts, because we didn't provided the password for student account. So, the proper way to run ansible using password authentication is use the options `-u` for username and `--ask-pass` for prompting us to enter the account password:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m ping -u student --ask-pass
SSH password:
ansible-00-02-ubuntu | SUCCESS => {
  "ansible_facts": {
```

```

        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host ansible-00-01-
hivemaster should use /usr/bin/python3, but is using /usr/bin/python for
backward compatibility with prior Ansible releases. A future Ansible
release will default to using the discovered platform python for this
host. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_di
scovery.html for more information. This feature will be
removed in version 2.12. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.
ansible-00-01-hivemaster | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
10.142.15.213 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}

```

### Task 3.3: Fix [DEPRECATION WARNING] for hivemaster

Notice that in this case we can omit to specify `-u` (for user) because “student” user is the current user that Ansible is trying to use.

Moreover, you can see that for our `hivemaster` server we have a [DEPRECATION WARNING], so we are going to fix this warning, as it is always recommended to do with warnings by adding `interpreter_python = auto` in the defaults section:

```

student@ansible-00-01-hivemaster:~$ sudo vi /etc/ansible/ansible.cfg

[defaults]

# some basic default values...
interpreter_python = auto

[...]

```

### Task 3.4: Test again for hivemaster

We are going to test that the warning is not showing anymore for our `hivemaster` host:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts ansible-00-01-hivemaster -m ping -u student --ask-pass
SSH password:
ansible-00-01-hivemaster | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

### Task 3.5: Check the connection user used by Ansible

To make sure that the user is “student”, we can run also the “id” command using `shell` module:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a "id" -u student --ask-pass
SSH password:
ansible-00-01-hivemaster | CHANGED | rc=0 >>
uid=1002(student) gid=1003(student) groups=1003(student)

ansible-00-02-ubuntu | CHANGED | rc=0 >>
uid=1002(student) gid=1003(student) groups=1003(student)

10.142.15.213 | CHANGED | rc=0 >>
uid=1001(student) gid=1002(student) groups=1002(student)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Notice that we used here another module called `shell` to run the `id` command, instead of `raw` module which we used in [Task 1](#) (we are going to learn more about differences between these modules into modules lab).

### Task 3.6: Accessing system files (privilege escalation)

Firstly, we are going to try accessing `/etc/shadow` file using `student` user:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a "cat /etc/shadow" -u student --ask-pass
SSH password:
ansible-00-02-ubuntu | FAILED | rc=1 >>
cat: /etc/shadow: Permission deniednon-zero return code

ansible-00-01-hivemaster | FAILED | rc=1 >>
cat: /etc/shadow: Permission deniednon-zero return code

10.142.15.213 | FAILED | rc=1 >>
cat: /etc/shadow: Permission deniednon-zero return code
```

As you may already expected, student user cannot access `/etc/shadow` file, so we have to instruct Ansible to become `root` before executing this command:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a
"cat /etc/shadow" -u student --ask-pass --become
SSH password:
ansible-00-01-hivemaster | FAILED | rc=-1 >>
Missing sudo password
ansible-00-02-ubuntu | FAILED | rc=-1 >>
Missing sudo password
10.10.17.146 | FAILED | rc=-1 >>
Missing sudo password
```

It looks like we didn't provided the BECOME password for becoming `root` user. This can be fixed by adding `--ask-become-pass`.

Notice that the `BECOME` password defaults to `SSH` password, so in our case (because we use the same password) we can just press ENTER:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a
"cat /etc/shadow" -u student --ask-pass --become --ask-become-pass
SSH password:
BECOME password[defaults to SSH password]:
ansible-00-01-hivemaster | CHANGED | rc=0 >>
root:$6$27488$DYVSMYCFtpga/pAdoxJM9iyaVI4BE0ku8zb2c7GWrPE28t5K0qwvaMvKez
jWUCIGdwDtWbrHnPn6kYse7SEjb1:18223:0:99999:7:::
daemon*:18213:0:99999:7:::
bin*:18213:0:99999:7:::
[...]

ansible-00-02-ubuntu | CHANGED | rc=0 >>
root:$6$45423$/USc3nLyxSEx0HmuXUCCi6g0TsqtLk9eDUkmvEQHuw1vHXLrpCZa5e3TfK
9De7c2H22Ie/3qvTCPKTX6WTO301:18223:0:99999:7:::
daemon*:18213:0:99999:7:::
[...]

10.142.15.213 | CHANGED | rc=0 >>
root:$6$48153$AYWTuD52gImgiS/lWOyeY0LsTaK9byanOBfkCL97SImtVUYHjVzdying0J
KNgE34bgesc.D2fjtkinDSCrpjI0:18223:0:99999:7:::
bin*:17834:0:99999:7:::
[...]
```

#### Task 4: Running Ad-Hoc commands using key-based authentication

In everyday use the best practice is to have a public key installed on the remote hosts and it is also recommended to create a dedicated user who should have access using that key. Right now, we are going to create a new keypair for our student user and add the public key to the proper file `/home/student/.ssh/authorized_keys`. Notice that you can issue this command on any server you want, the key is provisioned to all 3 servers.

```
student@ansible-00-02-ubuntu:~$ cat /home/student/.ssh/authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAKdlJG4MF1YUglv6cFYDE93gAK7Z3/HeFJ/ZjWgHIYQAt
NCkhHnR0EXlN3kJJUu/xP7zLwcfIbY73pjTVMtmwx+kheb1R4Uqj1SPTHQWrVpeddVDLmflS
h5nCFVJJZ9bdi8YTgyl2NKTbNIQ7U9fXP/AIaNHZBT0rBUcf055qCm+ub2qx+TOvkzalaIPz
+XHo6Cx37+Pkq3WmQBfDmAuOwRByE95bF1FYd02mGGvOiedK1bJM8jxcMTzqZ/0gyFA4cXkh
pQBjbMIEeWqpbE0ZUZ7RY+thHW2JIXfu0tcqjbUnMIkAwN+JDVBolRPuNKJ/4vpIMf3sfysg
n7dgrvS0tw== bogd@cloudbrain
```

Notice that this file already contains a public key, which was used during provisioning of the infrastructure, so be careful just to append the new key to this file.

#### Task 4.1: Generate a SSH keypair

In order to test key-based authentication we are going to generate a new public/private key pair, using RSA type (-t), 2048 length and save it as `ansible_key` in the current directory (without password):

```
student@ansible-00-01-hivemaster:~$ ssh-keygen -t rsa -b 2048 -f
ansible_key
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ansible_key.
Your public key has been saved in ansible_key.pub.
The key fingerprint is:
SHA256:gm+P8xHI5tl55qCJF88jmDj/9C4Z25rzLTzQSrZmO/4 student@ansible-00-
01-hivemaster
The key's randomart image is:
+---[RSA 2048]-----+
|
|
|
|   o .
|  . =.S
|   +*+.o
|  . =*&= o
| o o+#BOB
| oo*X&E+o
+---[SHA256]-----+
```

You will notice that 2 new files were created in the current directory (`/home/student`):

```
student@ansible-00-01-hivemaster:~$ ls | grep ansible
ansible_key
ansible_key.pub
```

Right now we are going to add the new generated public key (`ansible_key.pub`) to our student user on hivemaster host:

```
student@ansible-00-01-hivemaster:~$ cat ansible_key.pub >>
/home/student/.ssh/authorized_keys
```

To make things easier, we are going to use Ansible itself to add the public key also on the other 2 managed hosts (ubuntu and centos):

#### Task 4.2: Put publickey into correct place

As we just generated the new keypair, we need to put it in the corresponding file for student user (/home/student/.ssh/authorized\_keys).

Firstly, we have to copy the file also to the remote servers (we are going to use Ansible for that):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m copy -a
"src=/home/student/ansible_key.pub dest=/home/student/" -u student --
ask-pass --become --ask-become-pass
```

Now that we have `ansible_key.pub` in `/home/student` let's append the key to the `authorized_keys` file from each host:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a
"cat ansible_key.pub >> /home/student/.ssh/authorized_keys" -u student -
-ask-pass --become --ask-become-pass
SSH password:
BECOME password[defaults to SSH password]:
ansible-00-02-ubuntu | CHANGED | rc=0 >>

ansible-00-01-hivemaster | CHANGED | rc=0 >>

10.142.15.213 | CHANGED | rc=0 >>
```

You can check that the public key was properly added:

```
student@ansible-00-01-hivemaster:~$ cat
/home/student/.ssh/authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAKdlJG4MF1YUglv6cFYDE93gAK7Z3/HeFJ/ZjWgHIYQAt
NCkhHnR0EXlN3kJJUt/xP7zLwcfIbY73pjTVMtmwx+kheb1R4Uqj1SPTHQWrVpeddVDLmfLS
h5nCFVJJZ9bdi8YTgyl2NKTbnIQ7U9fXP/AIaNHZBT0rBUcf055qCm+ub2qx+TOvkzalaIPz
+XHo6Cx37+Pkq3WmQBfDmAuOwRByE95bF1FYd02mGGvOiedK1bJM8jxcMTzqZ/0gyFA4cXkh
pQBjbMIEeWqpbE0ZUZ7RY+thHW2JIxfu0tcqjbUnMIkAwN+JDVBolRPuNKJ/4vpIMf3sfysg
n7dgrvS0tw== bogd@thermite
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQD76mY0aD+iWPoR5zc+1Vp7Ve6vGSRbliogXl8ABA0l
urw8GKMnQHNBCAgNMO+u1R2qRRgY/fIBXF1D4iQm1Dd4Z4c3qwbbZAltK7zsoDcY8AZZwj5t
f54yhqN3BGQMySABIBWF7FKjyH1iZ/ut3hA2hqlIDv2rpqzB+U2Bov4+B7L3naWMAYYEnGtK
SD75GKsChsNpjVfZpw5v6V/3tMBw1iPS55IjWDTNwSCKrOmaFsxU45oMN7HnZCh5K1zL3+z1
```

```
w5golr5Smz+JgtMAAzhUgb9gDz9dbaJnrFoU6bS/wKNBebjrk3LPTMHSnW62pi/YtxxpRw2S
Qa5QRQnizgsn student@ansible-00-01-hivemaster
```

#### Task 4.3: Ping all hosts (using key-based auth)

It's time to test the key-based authentication, specifying user (-u) and private key file (--private-key):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m ping -u
student --private-key /home/student/ansible_key
ansible-00-02-ubuntu | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
ansible-00-01-hivemaster | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
10.142.15.213 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": false,
  "ping": "pong"
}
```

#### Task 4.4: Accessing system files (privilege escalation)

Also using key-based authentication we have to instruct Ansible to become root (--become) and to prompt for become password (--ask-become-pass):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a
"cat /etc/shadow" -u student --private-key /home/student/ansible_key --
become --ask-become-pass
ansible-00-02-ubuntu | CHANGED | rc=0 >>
root:$6$45423$/USc3nLyxSEx0HmuXUCCi6g0TsqTLk9eDUkmvEQHuwlVHXLrpCZa5e3TfK
9De7c2H22Ie/3qvTCPKTX6WTO301:18223:0:99999:7:::
daemon*:18213:0:99999:7:::
[...]

ansible-00-01-hivemaster | CHANGED | rc=0 >>
root:$6$27488$DYVSMYCFtpga/pAdoxJM9iyaVI4BE0ku8zb2c7GWrPE28t5K0qwvaMvKez
jWUCIGdwDtWbrHnPn6kYse7SEjb1:18223:0:99999:7:::
[...]
```



```
10.142.15.213 | CHANGED | rc=0 >>
root:$6$48153$AYWTuD52gImgis/lWOyeY0LsTaK9byanOBFkCL97SImtVUyHjVzdyinq0J
KNgE34bgesc.D2fjtkinDSCrpjI0:18223:0:99999:7:::
[...]
```

#### Task 4.5: Command module

We are going to explore this module right now, because command module is the default module for ansible command line utility. Let's test rebooting our ubuntu server (notice that we omitted specifying module name because Ansible defaults to command module):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts ansible-00-02-ubuntu -a "/sbin/reboot" -u student --private-key /home/student/ansible_key --become --ask-become-pass
BECOME password:
ansible-00-02-ubuntu | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ssh: connect to host ansible-00-02-ubuntu port 22: Connection timed out",
  "unreachable": true
}
```

Soon, you will notice a "Connection timed out" error, as the server become UNREACHABLE. Let's wait for server to reboot and test again the ping:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts ansible-00-02-ubuntu -m ping -u student --private-key /home/student/ansible_key
ansible-00-02-ubuntu | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

#### Task 4: Setup module in Ad-Hoc commands

Facts represent discovered variables about hosts gathered by Ansible using setup module. When using ad-hoc commands, we have to explicitly instruct Ansible to gather facts:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts ansible-00-02-ubuntu -m setup -u student --private-key /home/student/ansible_key
ansible-00-02-ubuntu | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "10.128.0.49"
    ],

```

[...]

As you may see in your environment, the amount of gathered facts (just for one host – notice that we run the command only against **ubuntu** host) is enormous, so it very likely that you may want to narrow the results to see only some of them. We can use `filter` option to make this possible:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts ansible-00-02-ubuntu -m setup -a "filter=ansible_env" -u student --private-key /home/student/ansible_key
ansible-00-02-ubuntu | SUCCESS => {
  "ansible_facts": {
    "ansible_env": {
      "HOME": "/home/student",
      "LANG": "C.UTF-8",
      "LOGNAME": "student",
      "MAIL": "/var/mail/student",
      "PATH":
"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games",
      "PWD": "/home/student",
      "SHELL": "/bin/bash",
      "SHLVL": "1",
      "SSH_CLIENT": "10.128.0.48 52772 22",
      "SSH_CONNECTION": "10.128.0.48 52772 10.128.0.49 22",
      "SSH_TTY": "/dev/pts/0",
      "TERM": "xterm",
      "USER": "student",
      "XDG_RUNTIME_DIR": "/run/user/1002",
      "XDG_SESSION_ID": "6",
      "_": "/bin/sh"
    },
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false
}
```

# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



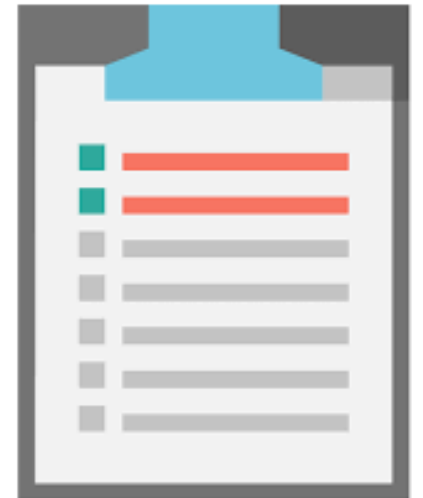
FASTER  
SMARTER  
SAFER

# 3. Preparing Hosts for Ansible



# Topics covered:

- Creating a dedicated user
- Configuring key-based auth
- Default `remote_user` and `private_key` in Ansible
- Static host inventories (INI and YAML)



# Creating a dedicated user

- Ansible is best implemented using a common user across all Ansible controlled systems.
- Only a basic system user with ssh access is needed for Ansible to connect to a host.

# Creating a dedicated user

- The following commands (executed as root) will create a new user on a system called **ansible** and allow for the user password to be set:
  - `useradd ansible`
  - `passwd ansible`
    - Recommended: `passwd -l ansible`
- This step should be performed on every node that will be managed by Ansible
  - Repetitive and time-consuming, you say? Well... not necessarily... 😊



# Creating a dedicated user

- Or you can use Ansible to automate this for all hosts

```
student:~$ ansible -i hosts.ini all --ask-pass --become -u student -m user -a  
"name=ansible state=present"  
SSH password:  
BECOME password[defaults to SSH password]:  
ansible-00-02-ubuntu | CHANGED | rc=0 >>  
[...]
```



# Configuring key-based auth

- a. After you created the user, you have to setup SSH key-based authentication ( learned in previous lesson)
- b. Create `.ssh` folder in `/home/ansible`
  - owner/group **ansible:ansible** and **0700** permissions

```
sudo mkdir /home/ansible/.ssh
sudo chown ansible:ansible /home/ansible/.ssh
sudo chmod 700 /home/ansible/.ssh
```

# Configuring key-based auth

c. Copy public key to `/home/ansible/.ssh/authorized_keys`

```
student:~$ sudo -i
root:~# cat /home/student/ansible_key.pub >> /home/ansible/.ssh/authorized_keys
root:~# chown ansible:ansible /home/ansible/.ssh/authorized_keys
root:~# chmod 644 /home/ansible/.ssh/authorized_keys
```

# Configuring key-based auth

## d. Test key-based auth for 'ansible' user

```
student:~$ ansible -i hosts all -m ping -u ansible --private-key /home/student/ansible_key
ansible-00-02-ubuntu | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

## e. In order to make sure that the correct user was used, we can also run the "id" command

```
student:~$ ansible -i hosts all -a "id" -u ansible --private-key /home/student/ansible_key
ansible-00-02-ubuntu | CHANGED | rc=0 >>
uid=1004(ansible) gid=1005(ansible) groups=1005(ansible)
```

# Giving the User sudo permissions

- The '**ansible**' user should be able to run commands as root, without entering the password
- The recommended way to perform changes on the `sudoers` file is by using the `visudo` command
  - it performs a check before saving, so that you do not get locked out of the system

```
student:~$ sudo visudo /etc/sudoers.d/95-ansible
#add this line
ansible ALL=(ALL:ALL) NOPASSWD: ALL
```

# Default remote\_user and private\_key in Ansible

- When we tested the key-based authentication for our new '**ansible**' user, we specified the name and path to the private key.
- We can configure default values for these parameters in **/etc/ansible/ansible.cfg** in order to avoid writing them every time we run ansible.
- These settings will be applied both when running ad hoc commands, and when running playbooks (discussed later)

# Default remote\_user and private\_key in Ansible

```
student:~$ sudo vi /etc/ansible/ansible.cfg
```

```
#Uncomment and modify (or add) the following lines:  
remote_user = ansible  
private_key_file = /home/student/ansible_key
```



```
student:~$ ansible -i hosts all -m shell -a "whoami"  
ansible-00-02-ubuntu | CHANGED | rc=0 >>  
ansible
```

```
ansible-00-01-hivemaster | CHANGED | rc=0 >>  
ansible
```

```
10.142.15.213 | CHANGED | rc=0 >>  
ansible
```

# Default remote\_user and private\_key in Ansible

- In the same file, **ansible.cfg**, we can also set (enable) some other parameters like **become**, **become\_user**, **become\_method** in **privilege\_escalation** section

```
[privilege_escalation]
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False
```

# Default remote\_user and private\_key in Ansible

- Notice that if we set **become=True** all the tasks will be executed as **become\_user** (root by default):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a "whoami"  
ansible-00-02-ubuntu | CHANGED | rc=0 >>  
root
```

```
ansible-00-01-hivemaster | CHANGED | rc=0 >>  
root
```

```
10.142.15.213 | CHANGED | rc=0 >>  
root
```



# Static host inventories

- In a minimal form, a static inventory is a list of host names and IP addresses that can be managed by Ansible.
- Host can be placed into groups to make it easy to address multiple hosts at once.
- A host can be a member of multiple groups.
- Nested groups are also available.

# Static host inventories

- It is common to work with project-based inventory files.
- Variables can be set from the inventory file – but this does not scale very well
- Ranges can be used:
  - `server[1:20]` matches server 1 up to server 20
  - `192.168.[4:5].[0:255]` matches two full class C subnets
- The most common formats are INI and YAML

# Inventory File Locations

- **/etc/ansible/hosts** is the default inventory.
- Alternative inventory location can be specified through the **ansible.cfg** configuration file
- Or use the **-i inventory** option to specify the location of the inventory file to use.
- It is common practice to put the inventory file in the current project directory.

# Static Inventory Example – INI format

```
[webservers]
web1.example.com
web2.example.com

[fileservers]
file1.example.com
file2.example.com

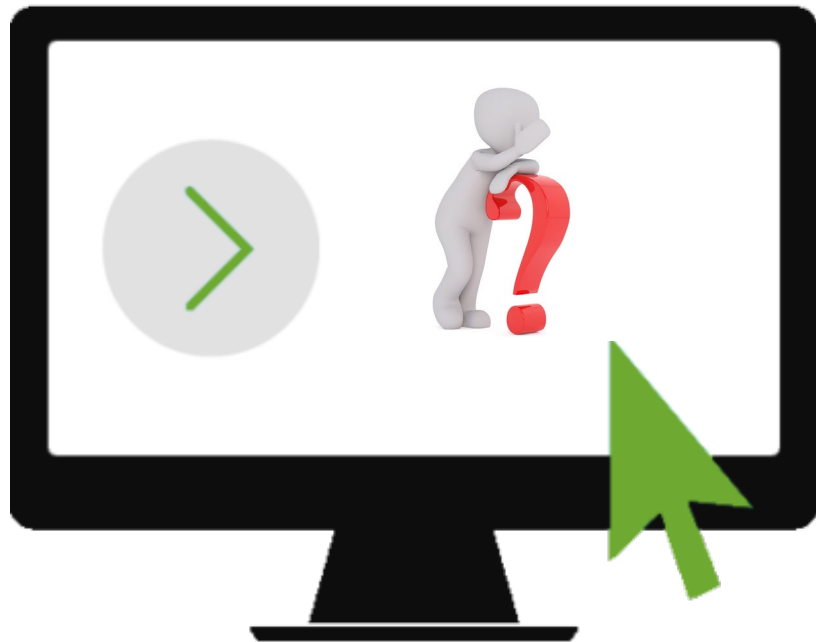
[server:children]
webservers
fileservers
```

group name

host name

# Static Inventory Example – YAML format

```
webservers:
  hosts:
    ubuntu:
      ansible_host: 10.128.0.20
      # We have not yet created ansible user here
      ansible_user: student
    centos:
  vars:
    type: webserver
dbservers:
  hosts:
    ubuntu:
    hivemaster:
datacenter:
  children:
    webservers:
    dbservers:
```



# Lab 3: Preparing hosts for Ansible





Solutions for training the world.





ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

## Confidential Document

Preparing hosts for Ansible use

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: Create a dedicated user	2
Task 2: Configuring key-based auth for “ansible” user	2
Task 2.1: Create .ssh folder in /home/ansible	2
Task 2.2: Copy public key to /home/ansible/.ssh/authorized_keys	4
Task 2.3: Test key-based auth for ‘ansible’ user	5
Task 3: Giving the ‘ansible’ user sudo permissions	6
Task 4: Configuring remote_user and private_key in ansible.cfg	6
Task 5: Configuring static host inventories	8
Task 5.1: Explore inventory file	8
Task 5.2: Create a new inventory file (INI format)	8
Task 5.3: Test the inventory file	9
Task 5.4: Create a new inventory file (YAML format)	11
Task 6: Set the default inventory	12
Task 7: Test “ansible” command	12

# Lab Overview and objectives

In this lab we are going to perform a full setup of our hosts in order to be prepared for managing with Ansible playbooks. We are going to create a dedicated user, configure key-based authentication and also provide **sudo** permissions for this user. Next we will perform some changes in ansible configuration file and we will also create a static host inventory file.

## Guided Tasks

### Task 1: Create a dedicated user

As we mentioned in previous labs, it is highly recommended to have a dedicated user for managing hosts with Ansible, so we are going to create a new user called “ansible”. To perform this task we can use `useradd` command with `-m` to create the user’s home directory and `-s /bin/bash` to set the default shell. You can add the users manually, or you can use Ansible to automate this for all hosts (login with student user using password authentication or key-based authentication):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m user -a  
"name=ansible state=present" --ask-pass --become --ask-become-pass
```

Otherwise, you can login into every host using SSH, and run the command which was provided as argument for `shell` module:

```
sudo useradd -m ansible -s /bin/bash
```

### Task 2: Configuring key-based auth for “ansible” user

Now that we have just created our user, we have to setup SSH key-based authentication. We are going to use the same keypair generated in the previous lab, which is located in `/home/student`.

```
student@ansible-00-01-hivemaster:~$ ls | grep ansible  
ansible_key  
ansible_key.pub
```

Things are similar to the previous task, we can make this manually, or we can use Ansible.

#### Task 2.1: Create `.ssh` folder in `/home/ansible`

As the name of the task says, we are going to create `/home/ansible/.ssh` folder with owner/group `ansible:ansible` and `0700` permissions (`rxw` for owner).

In this subtask we will use Ansible itself to perform the changes, but you can also use the following commands for each host (if you don't feel comfortable with Ansible modules yet):

```
sudo mkdir /home/ansible/.ssh
sudo chown ansible:ansible /home/ansible/.ssh
sudo chmod 700 /home/ansible/.ssh
```

The corresponding Ansible command is:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m file -a
"path=/home/ansible/.ssh state=directory owner=ansible group=ansible
mode=0700" --ask-pass --become --ask-become-pass
SSH password:
BECOME password[defaults to SSH password]:
ansible-00-02-ubuntu | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "gid": 1005,
    "group": "ansible",
    "mode": "0700",
    "owner": "ansible",
    "path": "/home/ansible/.ssh",
    "size": 4096,
    "state": "directory",
    "uid": 1004
}
ansible-00-01-hivemaster | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "gid": 1005,
    "group": "ansible",
    "mode": "0700",
    "owner": "ansible",
    "path": "/home/ansible/.ssh",
    "size": 4096,
    "state": "directory",
    "uid": 1004
}
10.142.15.213 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "gid": 1004,
    "group": "ansible",
    "mode": "0700",
```

```
"owner": "ansible",
"path": "/home/ansible/.ssh",
"secontext": "unconfined_u:object_r:ssh_home_t:s0",
"size": 6,
"state": "directory",
"uid": 1003
}
```

## Task 2.2: Copy public key to /home/ansible/.ssh/authorized\_keys

Remember that we have already copied the public key (ansible\_key.pub) to each server on /home/student. So, we just have to copy it to `authorized_keys` file and set proper owner/group and permissions:

```
student@ansible-00-01-hivemaster:~$ sudo -i
root@ansible-00-01-hivemaster:~# cat /home/student/ansible_key.pub >>
/home/ansible/.ssh/authorized_keys
root@ansible-00-01-hivemaster:~# chown ansible:ansible
/home/ansible/.ssh/authorized_keys
root@ansible-00-01-hivemaster:~# chmod 644
/home/ansible/.ssh/authorized_keys
root@ansible-00-01-hivemaster:~# exit
logout
student@ansible-00-01-hivemaster:~$
```

Notice that we jumped to `root` account to be able to write inside `.ssh` folder of “ansible” user!

Otherwise, this can be also done using Ansible using `copy` module (notice that we used an interesting parameter for this module, called `remote_src`, which specifies if the source file is copied from remote or from the local host to destination):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m copy -a
"src=/home/student/ansible_key.pub remote_src=yes
dest=/home/ansible/.ssh/authorized_keys owner=ansible group=ansible
mode=0644" --ask-pass --become --ask-become-pass
SSH password:
BECOME password[defaults to SSH password]:
ansible-00-02-ubuntu | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": true,
  "checksum": "92e5c547a77ee942116b4b7295d35afb57e8b9db",
  "dest": "/home/ansible/.ssh/authorized_keys",
  "gid": 1005,
  "group": "ansible",
  "md5sum": "d869316f5c31572b8232b1bee60e8313",
  "mode": "0644",
  "owner": "ansible",
  "size": 414,
```

```

    "src": "/home/student/ansible_key.pub",
    "state": "file",
    "uid": 1004
}
ansible-00-01-hivemaster | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": true,
  "checksum": "92e5c547a77ee942116b4b7295d35afb57e8b9db",
  "dest": "/home/ansible/.ssh/authorized_keys",
  "gid": 1005,
  "group": "ansible",
  "md5sum": "d869316f5c31572b8232b1bee60e8313",
  "mode": "0644",
  "owner": "ansible",
  "size": 414,
  "src": "/home/student/ansible_key.pub",
  "state": "file",
  "uid": 1004
}
10.142.15.213 | CHANGED => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": true,
  "checksum": "92e5c547a77ee942116b4b7295d35afb57e8b9db",
  "dest": "/home/ansible/.ssh/authorized_keys",
  "gid": 1004,
  "group": "ansible",
  "md5sum": "d869316f5c31572b8232b1bee60e8313",
  "mode": "0644",
  "owner": "ansible",
  "secontext": "unconfined_u:object_r:ssh_home_t:s0",
  "size": 414,
  "src": "/home/student/ansible_key.pub",
  "state": "file",
  "uid": 1003
}

```

### Task 2.3: Test key-based auth for 'ansible' user

```

student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m ping -u
ansible --private-key /home/student/ansible_key
ansible-00-02-ubuntu | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}

```

```
}
ansible-00-01-hivemaster | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
10.142.15.213 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": false,
  "ping": "pong"
}
```

In order to make sure that the correct user was used, we can also run “id” command:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -a "id" -u
ansible --private-key /home/student/ansible_key
ansible-00-02-ubuntu | CHANGED | rc=0 >>
uid=1004(ansible) gid=1005(ansible) groups=1005(ansible)

ansible-00-01-hivemaster | CHANGED | rc=0 >>
uid=1004(ansible) gid=1005(ansible) groups=1005(ansible)

10.142.15.213 | CHANGED | rc=0 >>
uid=1003(ansible) gid=1004(ansible) groups=1004(ansible)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

### Task 3: Giving the ‘ansible’ user sudo permissions

The ‘ansible’ user should be able to run commands as root, without entering the password, so we have to add it to `/etc/sudoers` file. The recommended way to perform changes on sudoers file is using `visudo` command, as it performs also a check before saving, in order not to mess something and not be able to edit this file anymore:

```
student@ansible-00-01-hivemaster:~$ sudo visudo
#add this line at the end of the file
ansible ALL=(ALL:ALL) NOPASSWD: ALL
```

Make sure to login also in the other 2 servers and perform the same task.

### Task 4: Configuring `remote_user` and `private_key` in `ansible.cfg`

In [Task 2.3](#) we tested key-based authentication for our new ‘ansible’ user, specifying its name and path to the private key. Notice that it is also necessary in case of using playbooks, not just in case of ad-hoc

commands. We can configure default values for these parameters in `/etc/ansible/ansible.cfg` in order to avoid writing them every time we run ansible:

```
student@ansible-00-01-hivemaster:~$ sudo vi /etc/ansible/ansible.cfg

#Uncomment and modify (or add) the following lines:
remote_user = ansible
private_key_file = /home/student/ansible_key
```

Right now we can check that these parameters are set correctly:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a
"whoami"
ansible-00-02-ubuntu | CHANGED | rc=0 >>
ansible

ansible-00-01-hivemaster | CHANGED | rc=0 >>
ansible

10.142.15.213 | CHANGED | rc=0 >>
ansible
```

In the same file, `ansible.cfg`, we can also set (enable) some other parameters like `become`, `become_user`, `become_method` in `privilege_escalation` section:

```
[privilege_escalation]
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False
```

Notice that if we set `become=True` all the tasks will be executed as `become_user` (root by default):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a
"whoami"
ansible-00-02-ubuntu | CHANGED | rc=0 >>
root

ansible-00-01-hivemaster | CHANGED | rc=0 >>
root

10.142.15.213 | CHANGED | rc=0 >>
root
```

Make sure to comment again `become=True` because right now we don't want that all the requests to be executed as root user.



## Task 5: Configuring static host inventories

Until now we worked with a basic inventory file which contains only the hostnames (IP for centos) of our hosts. In this section we are going to make this inventory a little bit more complex (not by adding more hosts) by grouping hosts in groups and setting some variables for them.

### Task 5.1: Explore inventory file

Before performing any changes to our inventory file let's see how Ansible sees our `hosts` file. We can use `ansible-inventory` command using `--graph` or `--list`:

```
student@ansible-00-01-hivemaster:~$ ansible-inventory -i hosts --list
{
  "_meta": {
    "hostvars": {}
  },
  "all": {
    "children": [
      "ungrouped"
    ]
  },
  "ungrouped": {
    "hosts": [
      "10.142.15.213",
      "ansible-00-01-hivemaster",
      "ansible-00-02-ubuntu"
    ]
  }
}
```

```
student@ansible-00-01-hivemaster:~$ ansible-inventory -i hosts --graph
@all:
|--@ungrouped:
|   |--10.142.15.213
|   |--ansible-00-01-hivemaster
|   |--ansible-00-02-ubuntu
```

As you can see from these results, our servers are listed in “ungrouped” group.

### Task 5.2: Create a new inventory file (INI format)

Our first basic inventory file was written in INI format, so we are going to start also with this format for the current inventory. Let's put our hosts in groups according to the following table:

Group	Hosts	Group Vars
webservers	hivemaster, centos	apache_port 80, apache_path /var/www/html/
dbservers	hivemaster, ubuntu	

We can also group these 2 groups into a bigger group (let's name it datacenter).

```
student@ansible-00-01-hivemaster:~$ vi hosts_ini
hivemaster ansible_host="10.128.0.48" ansible_user=ansible
ansible_ssh_private_key_file=/home/student/ansible_key
ubuntu ansible_host="10.128.0.49"
centos ansible_host="10.142.15.213" ansible_user=student

[webservers]
ubuntu
centos

[webservers:vars]
apache_port=80
apache_path=/var/www/html/

[dbservers]
ubuntu
hivemaster

[datacenter:children]
webservers
dbservers
```

Notice some important aspects in this inventory file:

- in the first part we defined the `hosts` (using `inventory_hostnames`) and some `host_vars` (`ansible_host` – this can also be the hostname instead of IP address, `ansible_user`, `ansible_ssh_private_key_file`); these are redundant for the first 2 hosts as they are already defined in `ansible.cfg`;
- we created the specified groups (`webservers`, `dbservers`), included the hosts into them and set the `group_vars`;
- we included both groups into the bigger one, `datacenter`, using “`:children`”, otherwise Ansible would try to include in `datacenter` group 2 hosts called `webservers` and `dbservers`.

### Task 5.3: Test the inventory file

It's time to test our new inventory, so let's filter gathered facts to see some info about IP addresses:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts_ini all -m setup -a
"filter=*ipv4"
ubuntu | SUCCESS => {
  "ansible_facts": {
    "ansible_default_ipv4": {
      "address": "10.128.0.49",
      "alias": "ens4",
      "broadcast": "global",
      "gateway": "10.128.0.1",
```

```

        "interface": "ens4",
        "macaddress": "42:01:0a:80:00:31",
        "mtu": 1460,
        "netmask": "255.255.255.255",
        "network": "10.128.0.49",
        "type": "ether"
    },
    "discovered_interpreter_python": "/usr/bin/python3"
},
"changed": false
}
hivemaster | SUCCESS => {
    "ansible_facts": {
        "ansible_default_ipv4": {
            "address": "10.128.0.48",
            "alias": "ens4",
            "broadcast": "global",
            "gateway": "10.128.0.1",
            "interface": "ens4",
            "macaddress": "42:01:0a:80:00:30",
            "mtu": 1460,
            "netmask": "255.255.255.255",
            "network": "10.128.0.48",
            "type": "ether"
        },
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false
}
centos | SUCCESS => {
    "ansible_facts": {
        "ansible_default_ipv4": {
            "address": "10.142.15.213",
            "alias": "eth0",
            "broadcast": "10.142.15.213",
            "gateway": "10.142.0.1",
            "interface": "eth0",
            "macaddress": "42:01:0a:8e:0f:d5",
            "mtu": 1460,
            "netmask": "255.255.255.255",
            "network": "10.142.15.213",
            "type": "ether"
        },
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}

```

Right now we want also to see what user is using Ansible to connect to every host (remember that for centos we set the “student” user, for hivemaster we set also the `ansible_user` and `ansible_ssh_key`, while for ubuntu we didn’t set any host var):

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts_ini all -a "whoami"
ubuntu | CHANGED | rc=0 >>
ansible

hivemaster | CHANGED | rc=0 >>
ansible

centos | CHANGED | rc=0 >>
student
```

As you can see, for `hivemaster` and `ubuntu` the “**ansible**” user was used (this is also configured in `ansible.cfg`), while for `centos` the “**student**” user was used, so the variable defined in the inventory file has precedence over the `ansible_user` defined in `ansible.cfg`!

Now we want to set also for centos the ansible user (so we have to remove the `ansible_user=student` for centos from inventory file, or replace it with ‘ansible’).

```
student@ansible-00-01-hivemaster:~$ vi hosts_ini
centos ansible_host="10.142.15.213" ansible_user=ansible
```

Test again using `ansible -i hosts_ini all -a "whoami"`.

#### Task 5.4: Create a new inventory file (YAML format)

Right now you just have to create the same inventory file, using YAML format (basically it’s just a rewrite of the same inventory):

```
student@ansible-00-01-hivemaster:~$ vi hosts_yaml

webservers:
  hosts:
    ubuntu:
      ansible_host: "10.128.0.49"
    centos:
      ansible_host: "10.142.15.213"
      ansible_user: ansible #we replaced student with ansible
  vars:
    apache_port: 80
    apache_path: /var/www/html/

dbservers:
  hosts:
    ubuntu:
    hivemaster:
      ansible_host: "10.128.0.48"
      ansible_user: ansible
```

```
ansible_ssh_private_key_file: /home/student/ansible_key

datacenter:
  children:
    webservers:
    dbservers:
```

#### Task 5.5: Explore the new inventory files (INI and YAML)

We can use the same command “ansible-inventory” to explore the list and graph of the new inventory files:

```
student@ansible-00-01-hivemaster:~$ ansible-inventory -i hosts_ini --
list
student@ansible-00-01-hivemaster:~$ ansible-inventory -i hosts_ini --
graph
student@ansible-00-01-hivemaster:~$ ansible-inventory -i hosts_yaml --
list
student@ansible-00-01-hivemaster:~$ ansible-inventory -i hosts_yaml --
graph
```

#### Task 6: Set the default inventory

Ansible uses by default the inventory file located at /etc/ansible/hosts. This file is commented by default and it is also a good start to explore its content in order to make an idea about the various possibilities of writing the inventory file:

```
student@ansible-00-01-hivemaster:~$ cat /etc/ansible/hosts
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
[...]
```

Let's copy the content of our hosts\_ini to /etc/ansible/hosts:

```
student@ansible-00-01-hivemaster:~$ sudo bash -c 'cat hosts_ini >
/etc/ansible/hosts'

student@ansible-00-01-hivemaster:~$ cat /etc/ansible/hosts
hivemaster ansible_host="10.128.0.48" ansible_user=ansible
ansible_ssh_private_key_file=/home/student/ansible_key
ubuntu ansible_host="10.128.0.49"
centos ansible_host="10.142.15.213" ansible_user=ansible
[...]
```

#### Task 7: Test “ansible” command

Finally, we can test using `ansible` command without providing host/user/key options:

```
student@ansible-00-01-hivemaster:~$ ansible all -m ping
ubuntu | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
hivemaster | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
centos | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": false,
  "ping": "pong"
}
```

```
student@ansible-00-01-hivemaster:~$ ansible all -a "id"
ubuntu | CHANGED | rc=0 >>
uid=1004(ansible) gid=1005(ansible) groups=1005(ansible)

hivemaster | CHANGED | rc=0 >>
uid=1004(ansible) gid=1005(ansible) groups=1005(ansible)

centos | CHANGED | rc=0 >>
uid=1003(ansible) gid=1004(ansible) groups=1004(ansible)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```



# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



FASTER  
SMARTER  
SAFER

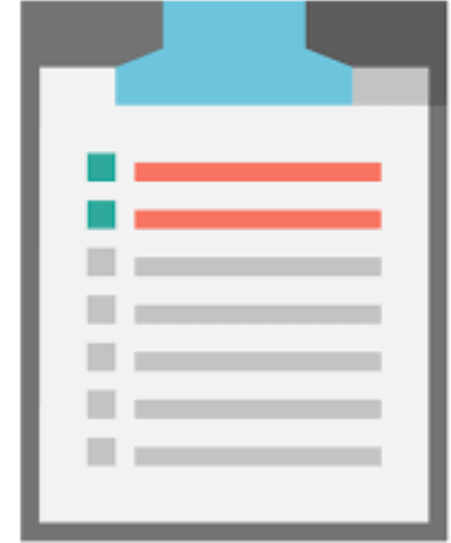
# 4. Basic Playbooks





# Topics covered:

- YAML overview
- Modules, Tasks, Plays, Playbooks
- General Playbook Structure
- Commonly used Modules
- Task Results (OK vs changed vs failed)
- Validating the Result
- Writing Idempotent Tasks



# Why Playbooks?

- Ad-hoc commands can be used to run one or a few tasks
- Ad-hoc commands are convenient to test, or when a complete managed infrastructure hasn't been set up yet.
- Ansible **Playbooks** are used to **run multiple tasks** against **managed hosts** in a scripted way
- In Playbooks, one or multiple plays are started
  - Each play runs one or more tasks
  - In these tasks, different modules are used to perform the actual work
- **Playbooks** are **written** in **YAML**, and have the **.yml** or **.yaml** extension

# YAML Overview

- YAML is Yet Another Markup Language according to some.
- According to others it stands for YAML Ain't Markup Language
- Anyway, it's an easy-to-read format to structure tasks/items that need to be created
- In YAML files, items are using indentation with white spaces to indicate the structure of data
- Data elements at the same level should have the same indentation
- Child items are indented more than the parent items

# Create a Basic Playbook

- A simple playbook which contains just one play (and particularly this play will contain just one task) and performs the ping command to all hosts in the inventory:

```
---  
- name: Ping all hosts  
  hosts: all  
  tasks:  
    - name: Ping task  
      ping:
```

# Run a Playbook

- Use `ansible-playbook <playbook.yml>` to run the playbook
- Notice that a successful run requires the inventory and become parameters to be set correctly, and also requires access to an inventory file
- The output of the **ansible-playbook** command will show what exactly has happened
- Playbooks in general are idempotent, which means that running the same playbook again should lead to the same result
- Notice there is no easy way to undo changes made by a playbook

# Run a Playbook - output

```
student:~$ ansible-playbook playbook.yml
```

```
PLAY [Ping all hosts]
```

```
*****
```

```
TASK [Gathering Facts]
```

```
*****
```

```
ok: [ubuntu]
```

```
ok: [hivemaster]
```

```
ok: [centos]
```

```
TASK [Ping task]
```

```
*****
```

```
ok: [ubuntu]
```

```
ok: [hivemaster]
```

```
ok: [centos]
```

```
PLAY RECAP
```

```
*****
```

centos	: ok=2	changed=0	unreachable=0	failed=0	skipped=0	rescued=0
ignored=0						
hivemaster	: ok=2	changed=0	unreachable=0	failed=0	skipped=0	rescued=0
ignored=0						
ubuntu	: ok=2	changed=0	unreachable=0	failed=0	skipped=0	rescued=0
ignored=0						

# Understanding Modules

- Ansible comes with lots of modules that allow you to perform specific tasks on managed hosts.
- When using Ansible you'll always use modules to tell Ansible what you want to do, in ad-hoc commands as well in playbooks.
- Many modules are provided with Ansible, if required you can develop your own modules.
- Use **ansible-doc -l** for a list of modules currently available.
- All modules work with arguments, **ansible-doc** will show which arguments are available and which are required.

# Understanding Modules

```
[student@workstation modules]$ ansible-doc -l
a10_server          Manage A10 Networks AX/SoftAX/Thunder/vThunder devices
a10_service_group   Manage A10 Networks devices' service groups
a10_virtual_server   Manage A10 Networks devices' virtual servers
acl                 Sets and retrieves file ACL information.
add_host            add a host (and alternatively a group) to the ansible-
playbook in-memory inventory
airbrake_deployment Notify airbrake about app deployments
alternatives        Manages alternative programs for common commands
apache2_module       enables/disables a module of the Apache2 webserver
apk                 Manages apk packages
apt                 Manages apt-packages
...output omitted...
```



# Understanding Modules

```
[student@workstation modules]$ ansible-doc yum
> YUM
```

Installs, upgrade, removes, and lists packages and groups with the `yum` package manager.

Options (= is mandatory):

- `conf_file`  
The remote yum configuration file to use for the transaction.  
[Default: None]
- `disable_gpg_check`  
Whether to disable the GPG checking of signatures of packages being installed. Has an effect only if state is `present' or `latest'. (Choices: yes, no) [Default: no]

...output omitted...

## EXAMPLES:

```
- name: install the latest version of Apache
  yum: name=httpd state=latest
```

```
- name: remove the Apache package
  yum: name=httpd state=absent
```

...output omitted...

# Tasks

- Modules (with various settings) that are executed on remote hosts
- Configuration is declarative
  - We are telling the system what the desired end state needs to be
  - If the remote host configuration is already correct, nothing gets changed

- Example:

```
- name: Task to start the apache service
  service:
    name: httpd
    state: started
```

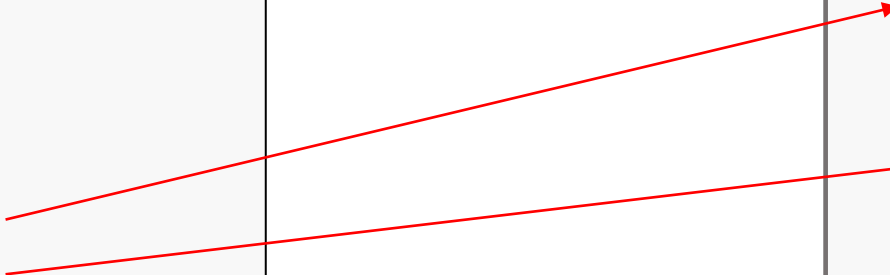
# Handlers

- Used to only execute some actions when they are needed (due to a change)
- These “notify” actions are triggered at the end of each block of tasks in a play, and will only be triggered once even if notified by multiple different tasks.
  - Ex.: Multiple resources may indicate that apache needs to be restarted because they have changed a config file, but apache will only be restarted once.
- Handlers are lists of tasks that are referenced by a globally unique name and are notified as needed.
- If nothing notifies a handler, it will not run.

# Handlers

```
- name: template configuration file
  template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
    - restart memcached
    - restart apache
```

```
handlers:
  - name: restart memcached
    service:
      name: memcached
      state: restarted
  - name: restart apache
    service:
      name: apache
      state: restarted
```



# Verifying Playbook Syntax

- `ansible-playbook -syntax-check <playbook.yml>` will perform a syntax check
- Use `-v[vvv]` to increase output verbosity
  - `-v` will show task results
  - `-vv` will show task results and task configuration
  - `-vvv` also shows information about connections to managed hosts
  - `-vvvv` adds information about plug-ins, users used to run scripts and names of scripts that are executed
- Use the `-C` option to perform a dry run

# Run a Playbook – with -vv output

```
student:~$ ansible-playbook playbook.yml -vv
ansible-playbook 2.9.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/student/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible-playbook
  python version = 2.7.15+ (default, Oct 7 2019, 17:39:04) [GCC 7.4.0]
Using /etc/ansible/ansible.cfg as config file

PLAYBOOK: playbook.yml
*****
1 plays in playbook.yml

PLAY [Ping all hosts]
*****

TASK [Gathering Facts]
*****
task path: /home/student/playbook.yml:2
ok: [ubuntu]
META: ran handlers

TASK [Ping task] *****
task path: /home/student/playbook.yml:5
ok: [ubuntu] => {"changed": false, "ping": "pong"}
META: ran handlers
[---]
```

# Understanding Plays

- A play is a series of tasks that are executed against selected hosts from the inventory, using specific credentials.
- Using multiple plays allows running tasks on different hosts, using different credentials from the same playbook.
- Within a play definition, escalation parameters can be defined:
  - **remote\_user**: the name of the remote user
  - **become**: to enable or disable privilege escalation
  - **become\_method**: to allow using an alternative escalation solution
  - **become\_user**: the target user used for privilege escalation

# Understanding Plays

```
---
- name: Ping all hosts
  hosts: all
  tasks:
    - name: Ping task
      ping:

- name: Deploy mongodb for dbservers group
  hosts: dbservers
  become: true
  tasks:
    - name: Install mongo
      package:
        name: mongodb
        state: latest
      notify: restart mongodb

handlers:
- name: restart mongodb
  service:
    name: mongodb
    state: restarted
    enabled: yes
```



# Commonly used Modules: Package management

- There is a module for most popular package managers, such as YUM and APT, to enable you to install any package on a system.
- Functionality depends entirely on the package manager, but usually these modules can install, upgrade, downgrade, remove, and list packages

```
- name: Install a list of packages
  yum:
    name:
      - nginx
      - postgresql
      - postgresql-server
    state: present
```

# Commonly used Modules: Service

- After installing a package, you need a module to start it.
- The service module enables you to start, stop, and reload installed packages.

```
- name: Start service foo, based on running process /usr/bin/foo
  service:
    name: foo
    pattern: /usr/bin/foo
    state: started
```

# Commonly used Modules: Copy

- The **copy module** copies a file from the local or remote machine to a location on the remote machine.

```
- name: Copy a new "ntp.conf" file into place, backing up the original if it differs from  
the copied version  
  copy:  
    src: /mine/ntp.conf  
    dest: /etc/ntp.conf  
    owner: root  
    group: root  
    mode: '0644'  
    backup: yes
```

# Commonly used Modules: Debug

- The **debug module** prints statements during execution and can be useful for debugging variables or expressions without having to halt the playbook.

```
- name: Display all variables/facts known for a host
  debug:
    var: hostvars[inventory_hostname]
    verbosity: 4
```

# Commonly used Modules: File

- The **file module** manages the file and its properties.
  - It sets attributes of files, symlinks, or directories.
  - It also removes files, symlinks, or directories.

```
- name: Change file ownership, group and permissions
  file:
    path: /etc/foo.conf
    owner: foo
    group: foo
    mode: '0644'
```

# Commonly used Modules: Lineinfile

- The **lineinfile** module manages lines in a text file.
  - It ensures a particular line is in a file or replaces an existing line using a back-referenced regular expression.
  - It's primarily useful when you want to change just a single line in a file.

```
- name: Ensure SELinux is set to enforcing mode
  lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: SELINUX=enforcing
```

# Commonly used Modules: Git

- The **git module** manages git checkouts of repositories to deploy files or software.

```
# Example Create git archive from repo
- git:
    repo: https://github.com/ansible/ansible-examples.git
    dest: /src/ansible-examples
    archive: /tmp/ansible-examples.zip
```

# Commonly used Modules: Cli\_command

- The **cli\_command module**, provides a platform-agnostic way of pushing text-based configurations to network devices over the **network\_cli connection** plugin.

```
- name: commit with comment
  cli_config:
    config: set system host-name foo
    commit_comment: this is a test
```



# Commonly used Modules: Archive

- The **archive module** creates a compressed archive of one or more files.
- By default, it assumes the compression source exists on the target.

```
- name: Compress directory /path/to/foo/ into /path/to/foo.tgz
  archive:
    path: /path/to/foo
    dest: /path/to/foo.tgz
```

# Commonly used Modules: Command

- One of the most basic but useful modules, the **command module** takes the command name followed by a list of space-delimited arguments.

```
- name: return motd to registered var  
  command: cat /etc/motd  
  register: mymotd
```

# Commonly used Modules: User

- The module manages users accounts with their attributes.
- This is handy do to the fact that the user properties and attributes can all be configured from this Ansible module.

```
# Add the user 'student' with a specific uid and a primary group of 'admin'
- user:
    name: student
    comment: "Example User"
    uid: 1040
    group: admin

# Remove the user 'student'
- user:
    name: student
    state: absent
    remove: yes
```

# OK vs Changed vs Failed

- As you already seen, every task of a play returns a status.
- This status is aiming to give the user a feedback about whether the task succeeded or not for a certain host

```
PLAY [playbook] *****
... Output omitted ...
TASK: [Install a service] *****
ok: [demoservera]
ok: [demoserverb]
```

```
PLAY RECAP *****
demoservera :          ok=2      changed=0      unreachable=0      failed=0
demoserverb :          ok=2      changed=0      unreachable=0      failed=0
```

# Writing Idempotent Tasks

- When possible, try to avoid the **command**, **shell**, and **raw** modules in playbooks, as simple as they may seem to use.
- Because these take arbitrary commands, it is very easy to write non-idempotent playbooks with these modules.

# Writing Idempotent Tasks

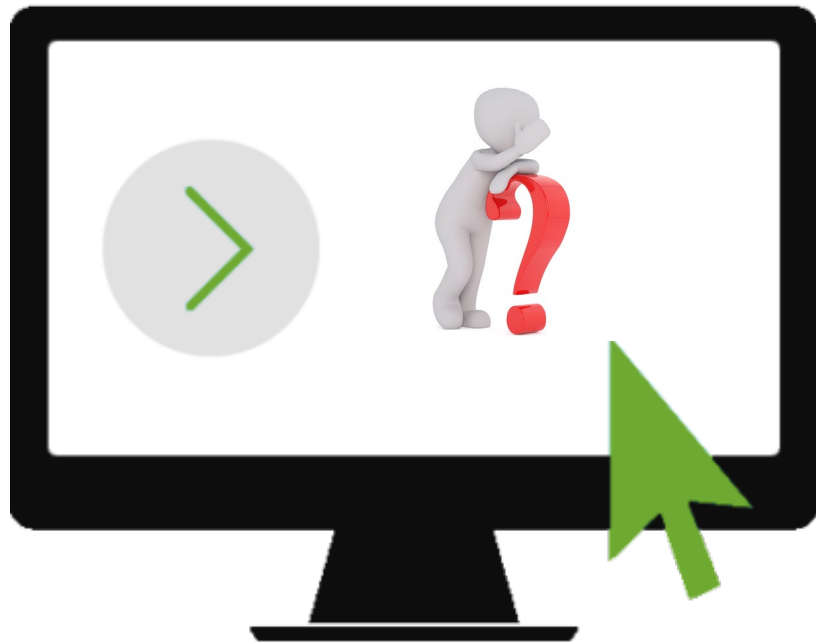
- When possible, try to avoid the **command**, **shell**, and **raw** modules in playbooks, as simple as they may seem to use.
- Because these take arbitrary commands, it is very easy to write non-idempotent playbooks with these modules.
- In order to achieve idempotency an useful feature is “creates” (used in **command** and **shell** modules).
- **Creates** (there is also **removes**) won't create the file if it already exists

# Writing Idempotent Tasks

```
---
- name: Command modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Raw module
      raw: cat /etc/hosts
      register: raw_output

    - name: Shell module
      shell: ls -l /var/log | grep log > /tmp/tmp.log
      args:
        creates: /tmp/tmp.log

    - name: Command module
      command: cat /etc/shadow
      register: cmd_output
```





# Lab 4: Basic Playbooks





Solutions for training the world.



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Basic Playbooks

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: Create your first playbook	2
Task 2: Add another play to existing playbook	4
Commonly used modules	5
Task 2: User and group modules	5
Task 2.1: Create a new user account	5
Task 2.2: Set bash for "testuser"	6
Task 2.3: Create a new group	6
Task 2.4: Add user to groups	7
Task 3: File module	7
Task 3.1: Create .ssh folder for "testuser"	7
Task 3.2: Create authorized_keys file inside .ssh	8
Task 4: Copy module	8
Task 4.1: Copy ansible_key.pub to authorized_keys file	9
Task 4.2: Test authentication	9
Task 5: Lineinfile module	9
Task 5.1: Add "testgroup" to sudoers file	9
Raw vs Command vs Shell	10
Task 6: Command module	10
Task 7: Raw module	11
Task 7: Shell module	11
Task 8: Command line modules in Playbooks	12
Task results	13
Task 9: OK vs Changed vs Failed	13
Task 10: Setting password for a user	15

# Lab Overview and objectives

The purpose of this lab is learning how to write and use Ansible Playbooks, which are a completely different way to use Ansible than ad-hoc commands, being more powerful and useful in configuration management.

## Guided Tasks

### Task 1: Create your first playbook

Let's start by creating a simple playbook which contains just one play (and particularly this play will contain just one task) and performs the ping command to all hosts in the inventory:

```
student@ansible-00-01-hivemaster:~$ vi playbook.yml
---
- name: Ping all hosts
  hosts: all
  tasks:
    - name: Ping task
      ping:
```

It is recommended that all the plays and tasks should have a name, in order to be easier to watch the execution steps in case of debugging or errors.

Run the playbook using `ansible-playbook` command:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook playbook.yml

PLAY [Ping all hosts]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Ping task]
*****
*****
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]
```

## PLAY RECAP

```
*****
*****
centos           : ok=2    changed=0    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
hivemaster      : ok=2    changed=0    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
ubuntu         : ok=2    changed=0    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
```

As you can see, the ping command succeeded for all 3 hosts in the inventory. If you want to see more details about playbook execution, you can specify `-v` (you can also use double, triple v) argument to set verbose mode.

```
student@ansible-00-01-hivemaster:~$ ansible-playbook playbook.yml -vv
ansible-playbook 2.9.1
  config file = /etc/ansible/ansible.cfg
  configured module search path =
[u'/home/student/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-
packages/ansible
  executable location = /usr/bin/ansible-playbook
  python version = 2.7.15+ (default, Oct  7 2019, 17:39:04) [GCC 7.4.0]
Using /etc/ansible/ansible.cfg as config file
```

## PLAYBOOK: playbook.yml

```
*****
*****
1 plays in playbook.yml
```

## PLAY [Ping all hosts]

```
*****
*****
```

## TASK [Gathering Facts]

```
*****
*****
```

```
task path: /home/student/playbook.yml:2
```

```
ok: [ubuntu]
```

```
ok: [hivemaster]
```

```
ok: [centos]
```

```
META: ran handlers
```

## TASK [Ping task]

```
*****
*****
```

```
task path: /home/student/playbook.yml:5
```

```
ok: [ubuntu] => {"changed": false, "ping": "pong"}
```

```
ok: [hivemaster] => {"changed": false, "ping": "pong"}
ok: [centos] => {"changed": false, "ping": "pong"}
META: ran handlers
META: ran handlers

PLAY RECAP
*****
*****
centos                : ok=2    changed=0    unreachable=0
failed=0      skipped=0      rescued=0      ignored=0
hivemaster        : ok=2    changed=0    unreachable=0
failed=0      skipped=0      rescued=0      ignored=0
ubuntu            : ok=2    changed=0    unreachable=0
failed=0      skipped=0      rescued=0      ignored=0
```

## Task 2: Add another play to existing playbook

You learned during lecture that a playbook contains at least one play, so we are going to add one more play to our playbook. Let's create a play which performs mongodb installation on our `dbservers`:

```
---
- name: Ping all hosts
  hosts: all
  tasks:
    - name: Ping task
      ping:

- name: Deploy mongodb for dbservers group
  hosts: dbservers
  become: true
  tasks:
    - name: Install mongo
      package:
        name: mongodb
        state: latest
      notify: restart mongodb

  handlers:
    - name: restart mongodb
      service:
        name: mongodb
        state: restarted
        enabled: yes
```

Notice that we introduced some new things in this play: we used `package` module to install mongodb, we targeted only `dbservers` group of hosts, the `become` parameter is set to "true", because we need a privileged user to perform package installation and we used a handler to restart and enable `mongodb` after installation.

## Commonly used modules

### Task 2: User and group modules

We are going to create a new playbook (called `modules.yml`) and we will start exploring different modules available in Ansible. Let's start with "user" module, which provides an option to manage user accounts and user attributes.

#### Task 2.1: Create a new user account

Let's add the first task of `modules.yml` playbook to create a new user account called "testuser" (set the playbook to affect all hosts in the inventory):

```
student@ansible-00-01-hivemaster:~$ vi modules.yml
---
- name: Modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Create "testuser"
      user:
        name: testuser
        state: present
```

Notice "become: yes" as we need to be root in order to add a new user.  
Run the playbook and watch the results:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modules.yml

PLAY [Modules Playbook]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Create "testuser"]
*****
*****
changed: [hivemaster]
changed: [ubuntu]
changed: [centos]
```



## PLAY RECAP

```
*****
*****
centos           : ok=2    changed=1    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
hivemaster       : ok=2    changed=1    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
ubuntu          : ok=2    changed=1    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
```

## Task 2.2: Set bash for “testuser”

Let’s edit the previous task and add also the `/bin/bash` as the default bash for our “testuser”:

```
student@ansible-00-01-hivemaster:~$ vi modules.yml
---
- name: Modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Create "testuser"
      user:
        name: testuser
        state: present
        shell: /bin/bash
```

Run the playbook again and see how many hosts are affected.

## Task 2.3: Create a new group

Using “user” module we can create new groups. So, we will create a new group called “testgroup”:

```
student@ansible-00-01-hivemaster:~$ vi modules.yml
---
- name: Modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Create "testuser"
      user:
        name: testuser
        state: present
        shell: /bin/bash

    - name: Create "testgroup"
      group:
        name: testgroup
        state: present
```

## Task 2.4: Add user to groups

In this task we will go back to the user module, in order to add “testuser” to the “testgroup”. The task will look like:

```
- name: Create "testuser"
  user:
    name: testuser
    state: present
    shell: /bin/bash
    groups: testgroup
```

Notice that we used “groups”, not just “group” – this sets the primary group.

We can also set the primary group for “testuser” to be “ansible” group:

```
- name: Create "testuser"
  user:
    name: testuser
    state: present
    shell: /bin/bash
    groups: testgroup
    group: ansible
```

Run the playbook and explore the tasks:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modules.yml
[...]

TASK [Create "testuser"]
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]
```

## Task 3: File module

Using “file” module we can create, set attributes or remove files and directories.

### Task 3.1: Create .ssh folder for “testuser”

We are going to create `/home/testuser/.ssh` folder and set owner/group and proper permissions (700) for it:

```
- name: Create '.ssh' folder
  file:
    path: /home/testuser/.ssh
```

```
state: directory
owner: testuser
group: ansible
mode: '0700'
```

Run the playbook and explore the tasks:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modules.yml
[...]

TASK [Create ".ssh" dir]
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]
```

### Task 3.2: Create `authorized_keys` file inside `.ssh`

Next, we have to create the `authorized_keys` file for our “testuser” in `/home/testuser/.ssh` and set owner/group and proper permissions (644) for it:

```
- name: Create 'authorized_keys' file under '.ssh'
  file:
    path: /home/testuser/.ssh/authorized_keys
    state: touch
    owner: testuser
    group: ansible
    mode: '0640'
```

Run the playbook and explore the tasks:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modules.yml
[...]

TASK [Create 'authorized_keys' file under '.ssh']
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]
```

### Task 4: Copy module

This module copies a file from `local` or `remote` source to a `remote` destination. The source location can be specified using `remote_src` parameter which by default is set to no (so it copies from the local machine).

#### Task 4.1: Copy ansible\_key.pub to authorized\_keys file

This is the last step to provide to our “testuser” the option to be able to login using SSH:

```
- name: Copy key to 'authorized_keys'
  copy:
    src: /home/student/ansible_key.pub
    dest: /home/testuser/.ssh/authorized_keys
    owner: testuser
    group: ansible
```

#### Task 4.2: Test authentication

You can test SSH authentication for “testuser” from command line:

```
student@ansible-00-01-hivemaster:~$ ssh -i ansible_key testuser@ansible-00-02-ubuntu
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.0.0-1025-gcp x86_64)
[...]
Last login: Mon Jan 01 23:14:53 2020 from 10.128.0.48
testuser@ansible-00-02-ubuntu:~$
testuser@ansible-00-02-ubuntu:~$ exit
```

If you want to test using Ansible you have to edit in /etc/ansible/hosts the ansible\_user variable for our hosts (in this example I will add this variable for ubuntu host only):

```
student@ansible-00-01-hivemaster:~$ sudo vi /etc/ansible/hosts
ubuntu ansible_host="10.128.0.49" ansible_user=testuser
```

Now let's see if Ansible can login using “testuser”:

```
student@ansible-00-01-hivemaster:~$ ansible ubuntu -a "id"
ubuntu | CHANGED | rc=0 >>
uid=1005(testuser) gid=1005(ansible)
groups=1005(ansible),1007(testgroup)
```

Revert back to the original user (ansible) for in /etc/ansible/hosts.

#### Task 5: Lineinfile module

Using this module, we can add (or replace) a line to a file, using regular expressions to make sure that if the line is not going to be duplicated if it exists.

##### Task 5.1: Add “testgroup” to sudoers file

Because editing sudoers file can get us locked out of the system, we are going to use also a validation tool for this file, using `visudo`:

```
- name: Validate the sudoers file before saving
  lineinfile:
    path: /etc/sudoers
    state: present
    regexp: '^%testgroup\s'
    line: '%testgroup ALL=(ALL) NOPASSWD: ALL'
    validate: /usr/sbin/visudo -cf %s
```

After running the playbook, you can try to use “testuser” to “cat” the contents of “/etc/shadow”. Notice that you have to temporary modify `ansible_user` (from `/etc/ansible/hosts`) to be “testuser” for at least one host. Same steps like in [Task 4.2](#)

Otherwise you can use the following ad-hoc command:

```
student@ansible-00-01-hivemaster:~$ ansible -i hosts all -m shell -a
"cat /etc/shadow" -u testuser --private-key /home/student/ansible_key --
become
```

## Raw vs Command vs Shell

These are the most common modules for executing commands on remote hosts. Each module has its own advantages and disadvantages. We also used them until now during this training, but right now we are going to see the main differences between them.

### Task 6: Command module

This module can only execute binaries from the remote hosts and it is the default module in Ad-Hoc mode. Command module won't be impacted by local shell variables since it bypasses the shell. At the same time, it may not be able to run “shell” inbuilt functions (like `set`) and redirection (which is also shell's inbuilt functionality).

```
student@ansible-00-01-hivemaster:~$ ansible all -a "who"
hivemaster | CHANGED | rc=0 >>
student   pts/0          2019-11-25 15:01 (82.137.13.156)
student   pts/1          2019-11-25 15:44 (82.137.13.156)
ansible   pts/4          2019-11-26 14:35 (10.128.0.48)

ubuntu | CHANGED | rc=0 >>
ansible   pts/0          2019-11-26 14:35 (10.128.0.48)

centos | CHANGED | rc=0 >>
ansible   pts/0          Nov 26 14:35 (ansible-00-01-
hivemaster.training.sass.ro)
```

This command executed the `/bin/who` binary from the remote servers and returned the results. Notice that Python is required, otherwise command module will fail.

### Task 7: Raw module

There are several distributions which doesn't have Python preinstalled, and managing them with Ansible would be impossible without this module which doesn't require Python to be installed (in fact, raw module is usually used to install Python on remote hosts), because it executes a low-down SSH command over the network. Another possible usage is in case of managing network devices that don't have Python installed.

```
student@ansible-00-01-hivemaster:~$ ansible all -m raw -a "who"
hivemaster | CHANGED | rc=0 >>
student pts/0      2019-11-25 15:01 (82.137.13.156)
student pts/1      2019-11-25 15:44 (82.137.13.156)
ansible pts/5      2019-11-26 14:48 (10.128.0.48)
Shared connection to 10.128.0.48 closed.

centos | CHANGED | rc=0 >>
ansible pts/0      Nov 26 14:48 (ansible-00-01-
hivemaster.training.sass.ro)
Shared connection to 10.142.15.213 closed.

ubuntu | CHANGED | rc=0 >>
ansible pts/0      2019-11-26 14:48 (10.128.0.48)
Shared connection to 10.128.0.49 closed.
```

### Task 7: Shell module

Shell module is very useful when you want to use redirections and shell's inbuilt functionality, as the command module doesn't support these.

```
student@ansible-00-01-hivemaster:~$ ansible all -m shell -a "df -h |
grep sda"
hivemaster | CHANGED | rc=0 >>
/dev/sda1      20G  1.6G   18G    9% /
/dev/sda15     105M  3.6M  101M    4% /boot/efi

ubuntu | CHANGED | rc=0 >>
/dev/sda1      20G  1.4G   18G    8% /
/dev/sda15     105M  3.6M  101M    4% /boot/efi

centos | CHANGED | rc=0 >>
/dev/sda1      20G  2.0G   19G   10% /
```

Let's try to run the same command using command module:

```
student@ansible-00-01-hivemaster:~$ ansible all -a "df -h | grep sda1"
hivemaster | FAILED | rc=1 >>
df: '|': No such file or directory
[...]
```

Now let's try also to redirect the output of a command:

```
student@ansible-00-01-hivemaster:~$ ansible all -m shell -a "cat
/etc/hosts > /tmp/hosts.txt"
hivemaster | CHANGED | rc=0 >>

ubuntu | CHANGED | rc=0 >>

centos | CHANGED | rc=0 >>
```

You may also try this using command module.

#### Task 8: "Command-line" modules in Playbooks

We used these 3 modules in Ad-Hoc mode until now, but you have to know that it is also possible to use them in playbooks. Let's create a new playbook and add some tasks:

```
student@ansible-00-01-hivemaster:~$ vi command_modules.yml
---
- name: Command modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Raw module
      raw: cat /etc/hosts
      register: raw_output

    - name: Shell module
      shell: ls -l /var/log | grep log > /tmp/tmp.log

    - name: Command module
      command: cat /etc/shadow
      register: cmd_output

    - name: Print raw output
      debug:
        var: raw_output

    - name: Print cmd output
      debug:
        var: cmd_output
```

Run the playbook and explore the content of `/tmp/tmp.log` for shell module output. Notice that for `raw` and `command` we registered the output in 2 variables and then printed those variables. You are going to learn more about variables in the next lab.

## Task results

### Task 9: OK vs Changed vs Failed

As you already seen, every task of a play returns a status. This status is aiming to give the user a feedback about whether the task succeeded or not for a certain host (also if it performed a change or not).

Let's start with the playbook from previous task (`command_modules.yml`). Running the playbook again you will notice that it returns that all 3 command modules returned the `changed` status.

```
student@ansible-00-01-hivemaster:~$ ansible-playbook command_modules.yml

PLAY [Command modules Playbook]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [hivemaster]
ok: [ubuntu]
ok: [centos]

TASK [Raw module]
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]

TASK [Shell module]
*****
*****
changed: [hivemaster]
changed: [ubuntu]
changed: [centos]

TASK [Command module]
*****
*****
changed: [ubuntu]
changed: [hivemaster]
changed: [centos]
```



# PLAY RECAP

```
*****
*****
centos      : ok=4    changed=3    unreachable=0
failed=0    skipped=0  rescued=0    ignored=0
hivemaster  : ok=4    changed=3    unreachable=0
failed=0    skipped=0  rescued=0    ignored=0
ubuntu     : ok=4    changed=3    unreachable=0
failed=0    skipped=0  rescued=0    ignored=0
```

This happens also if we run the playbook several times, which means that our tasks are not idempotent. In case of command and shell modules it is more difficult to make them idempotent, as they run a binary on the remote host.

In order to achieve idempotency an useful feature is “creates” (used in this case for shell module – it can also be used for command module). `Creates` (there is also `removes`) won’t create the file if it already exists (so if you run the playbook 2 times, you will notice that second time will just return `ok`:

```
---
- name: Command modules Playbook
  hosts: all
  become: yes
  tasks:
    - name: Raw module
      raw: cat /etc/hosts
      register: raw_output

    - name: Shell module
      shell: ls -l /var/log | grep log > /tmp/tmp.log
      args:
        creates: /tmp/tmp.log

    - name: Command module
      command: cat /etc/shadow
      register: cmd_output

# - name: Print raw output
#   debug:
#     var: raw_output

# - name: Print cmd output
#   debug:
#     var: cmd_output
```

Notice that I commented the variables printing tasks (in order to be easier to watch the playbook results).

Run again the playbook:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook command_modules.yml
```

```
PLAY [Command modules Playbook]
*****

TASK [Gathering Facts]
*****

ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Raw module]
*****

changed: [ubuntu]
changed: [hivemaster]
changed: [centos]

TASK [Shell module]
*****

ok: [hivemaster]
ok: [ubuntu]
ok: [centos]

TASK [Command module]
*****

changed: [ubuntu]
changed: [hivemaster]
changed: [centos]

PLAY RECAP
*****
centos                : ok=4    changed=2    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
hivemaster         : ok=4    changed=2    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
ubuntu             : ok=4    changed=2    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
```

Notice that usually Ansible modules are idempotent, which means that running a task several task will report changed just for the first run.

## Task 10: Setting password for a user

Create a new playbook and a task which performs the setting of a password for out “testuser”:

```
student@ansible-00-01-hivemaster:~$ vi password.yml
---
- name: User management
  hosts: all
  become: yes
  vars:
    user_pass: 123abc
  tasks:
    - name:
      user:
        name: testuser
        password: "{{ user_pass | password_hash('sha512') }}"
```

Run the playbook (several times) and explore tasks reports:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook password.yml

PLAY [User management]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [hivemaster]
ok: [ubuntu]
ok: [centos]

TASK [Setting password]
*****
*****
changed: [hivemaster]
changed: [ubuntu]
changed: [centos]

PLAY RECAP
*****
*****
centos                : ok=2    changed=1    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
hivemaster       : ok=2    changed=1    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
ubuntu          : ok=2    changed=1    unreachable=0
failed=0      skipped=0    rescued=0    ignored=0
```

You probably noticed that every time the task “Setting password” returns changed, even if the password is the same (and I said earlier that Ansible modules are idempotent, so it should return OK from the second run). This is due to the fact that in Linux passwords are salted with a random `salt`, so, the salted-password is in fact a new one.

# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



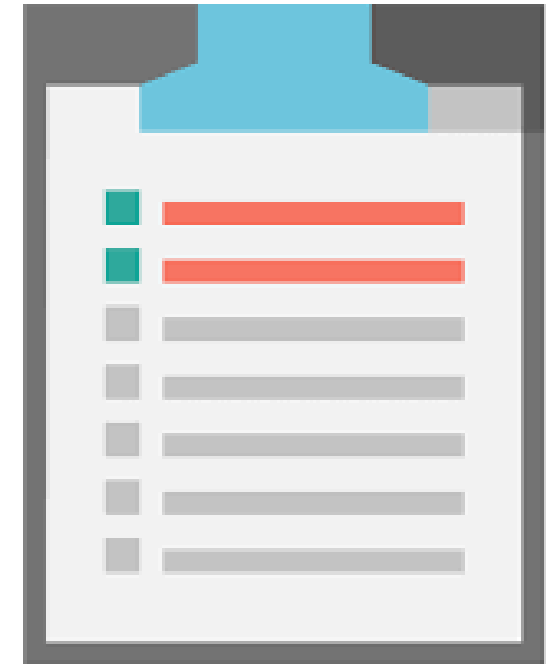
FASTER  
SMARTER  
SAFER

# 5. Facts, Variables



# Topics covered:

- Facts - definition
- Fact gathering
- Disabling fact gathering
- Custom facts
- Variable definition



# What are Facts?

- Ansible Facts are variables that are automatically set and discovered by Ansible on managed hosts.
- Facts contain information about hosts that can be used in conditionals.
- For instance, before installing specific software you can check that a managed host runs a specific kernel version.



# Managing Fact Gathering

- By default, all playbooks perform fact gathering before running the actual plays
- You can run fact gathering manually by using the **setup** module
- To show facts, use the debug module to print the value of the **ansible\_facts** variable



# Displaying Fact Names

- In Ansible 2.4 and before, Ansible facts were stored as individual variables, such as **ansible\_hostname** and **ansible\_interfaces**.
- In Ansible 2.5 and later, all facts are stored in one variable with the name **ansible\_facts**, and referring to specific facts happens in a different way:
  - **ansible\_facts['hostname']**
  - **ansible\_facts['interfaces']**
  - ...

# Displaying Fact Names

```
student:~$ ansible all -m setup -a "filter=*ipv4"
hivemaster | SUCCESS => {
  "ansible_facts": {
    "ansible_default_ipv4": {
      "address": "10.128.0.48",
      "alias": "ens4",
      "broadcast": "global",
      "gateway": "10.128.0.1",
      "interface": "ens4",
      "macaddress": "42:01:0a:80:00:30",
      "mtu": 1460,
      "netmask": "255.255.255.255",
      "network": "10.128.0.48",
      "type": "ether"
    },
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false
}
[...]
```

# Turning Off Fact Gathering

- Disabling fact gathering can significantly speed up playbook execution
- Use `gather_facts: no` in the play header to disable.
- If you need to use facts, you can collect them manually by running the `setup` module in a task.

```
- name: Disable facts
  hosts: all
  become: yes
  gather_facts: no
  tasks:
    - name: Print message
      debug:
        msg: "Fact gathering is disabled. Playbook is running faster"
```

# Using Custom Facts

- Custom facts allow administrators to dynamically generate variables which are stored as facts
- Custom facts are stored in an ini or json file in the `/etc/ansible/facts.d` directory on the managed host
  - The name of these files must end in `.fact`
- Custom facts are stored in the **`ansible_facts.ansible_local`** variable
- Use `ansible hostname -m setup -a "filter=ansible_local"` to display local facts

# Using Custom Facts

- Custom Facts Example File

```
[packages]
web_package = httpd
db_package = mariadb-server
[users]
user1 = joe
user2 = jane
```

```
[student ~]$ ansible demo1.example.com -m setup -a
'filter=ansible_local'
demo1.lab.example.com | SUCCESS => {
  "ansible_facts": {
    "ansible_local": {
      "custom": {
        "packages": {
          "db_package": "mariadb-server",
          "web_package": "httpd"
        },
        "users": {
          "user1": "joe",
          "user2": "jane"
        }
      }
    }
  },
  "changed": false
}
```

# Variables

- A variable is a label that is assigned to a specific value to make it easy to refer to that value throughout the playbook
- Variables can be defined by administrators at different levels
- Variables are particularly useful when dealing with managed hosts where specifics are different
  - Set a variable `web_service` on Ubuntu and Red Hat
  - Refer to that variable `web_service` instead of the specific service name

# Variables Definition

- Variables can be set at different levels
  - In a playbook
  - In the inventory file (not recommended for large numbers of variables)
  - In specific variable files
- Variable names have some requirements
  - The name must start with a letter
  - Variable names can only contain letters, numbers, and underscores

# Variables Definition

- Variables can be defined in a vars block in the beginning of a playbook

```
- hosts: all
  vars:
    web_package: httpd
```

- Alternatively, variables can be defined in a variable file, which will be included from the playbook

```
- hosts: all
  vars_files:
    - vars/users.yml
```



# Variables Definition

- After defining the variables, they can be used later in the playbook
- Referring to a variable:
  - `{{ web_package }}`
- If the variable is the first element, using quotes is mandatory!

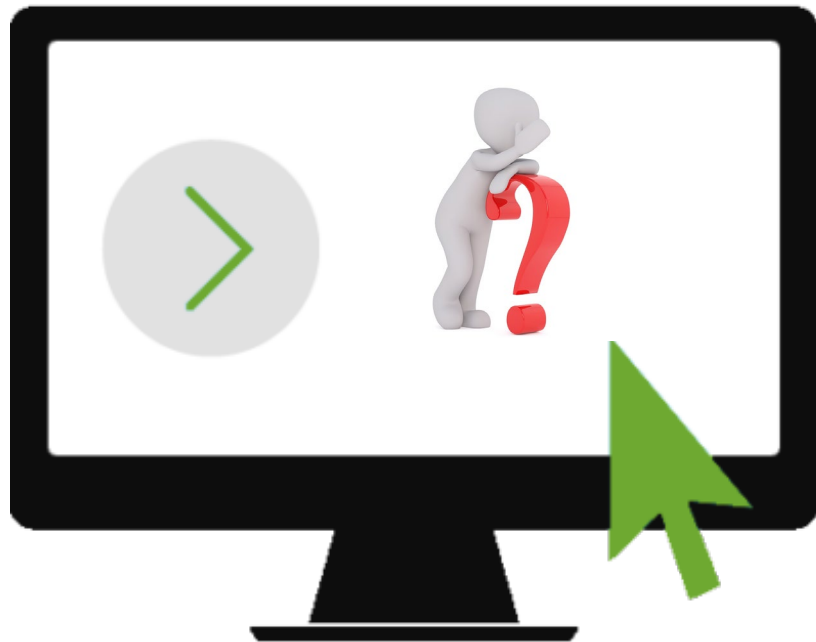
```
---  
- name: create a user using a variable  
  hosts: all  
  vars:  
    user: ben  
  tasks:  
    - name: create a user {{ user }}  
      user:  
        name: "{{ user }}"
```

# Variables Definition

- Variables can be set with different types of scope
  - **Global scope:** set from config, envvars, or command line
  - **Play scope:** set from the playbook (and included files)
  - **Host scope:** set from the inventory, collected facts, or registered task outputs
- When the same variable is set at different levels, the most specific level gets precedence
- When a variable is set from the command line, it will overwrite anything else
  - IF set with -e "var=value"!

# Variables Definition

- Some variables are build in and cannot be used for anything else
  - `hostvars`
  - `inventory_hostname`
  - `inventory_hostname_short`
  - `groups`
  - `group_names`
  - `ansible_check_mode`
  - `ansible_play_batch`
  - `ansible_play_hosts`
  - `ansible_version`



# Lab 5: Facts, Variables





Solutions for training the world.



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Facts. Variables.

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: Fact gathering	2
Task 2: Disable fact gathering in Playbooks	2
Task 3: Custom facts	3
Task 4: Host and group variables	4
Task 5: Defining variables	5
Task 6: Including variables	6
Task 6: Extra variables	7
Task 6: Variables vs Facts	8



# Lab Overview and objectives

The purpose of this lab is to learn how to use facts and variables in playbooks. Facts are automatically gathered using “setup” module and they represent useful variables about remote hosts that can be used in playbooks. Variables are defined by user and not gathered by ansible, but they have basically the same purpose – making easier the process of managing different hosts.

## Guided Tasks

### Task 1: Fact gathering

We already tested the “setup” module even from the first labs, using Ansible ad-hoc command:

```
student@ansible-00-01-hivemaster:~$ ansible all -m setup -a
"filter=*ipv4"
hivemaster | SUCCESS => {
  "ansible_facts": {
    "ansible_default_ipv4": {
      "address": "10.128.0.48",
      "alias": "ens4",
      "broadcast": "global",
      "gateway": "10.128.0.1",
      "interface": "ens4",
      "macaddress": "42:01:0a:80:00:30",
      "mtu": 1460,
      "netmask": "255.255.255.255",
      "network": "10.128.0.48",
      "type": "ether"
    },
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false
}
[...]
```

### Task 2: Disable fact gathering in Playbooks

As you have already seen, at the beginning of the execution of a playbook, the first “implicit” task is `gather_facts`. This task can be disabled if we don’t need these variables collected about hosts, as this task is taking some additional time:

```
vi disable_facts.yml
---
```

```
- name: Disable facts
  hosts: all
  become: yes
  gather_facts: no
  tasks:
    - name: Print message
      debug:
        msg: "Fact gathering is disabled. Playbook is running faster"
```

```
student@ansible-00-01-hivemaster:~$ ansible-playbook disable_facts.yml
```

```
PLAY [Disable facts]
*****
TASK [Print message]
*****
ok: [ubuntu] => {
  "msg": "Fact gathering is disabled. Playbook is running faster"
}
[...]
```

### Task 3: Custom facts

Apart from the facts collected by Ansible from remote hosts, we can also add custom facts that Ansible can fetch from these hosts. To add custom facts we have to create a new directory “/etc/ansible/facts.d” and put inside it one (or more) \*.fact file (s), that return JSON formatted data, which will then be included in the raft of facts that Ansible gathers at the start of each playbook run.

We are going to create a shell script which returns some info about the current logged-in user and his IP address in JSON format (we are also going to make this file executable):

```
student@ansible-00-01-hivemaster:~$ sudo mkdir /etc/ansible/facts.d
student@ansible-00-01-hivemaster:~$ sudo vi
/etc/ansible/facts.d/test.fact

#!/bin/bash
user=$(who | cut -f 1 -d " " | tail -1 )
ipuser=$(who -u | cut -f 2 -d "(" | cut -f 1 -d ")" | head -1)

cat << EOF
{
  "user" : "$user",
  "ipuser" : "$ipuser"
}
student@ansible-00-01-hivemaster:~$ sudo chmod +x
/etc/ansible/facts.d/test.fact

student@ansible-00-01-hivemaster:~$ ansible hivemaster -m setup | grep
ipuser
"ipuser": "92.137.23.156",
```

## Task 4: Host and group variables

Before starting to use variables it is important to know that variable names should contain only letters, numbers and underscores, and should always start with a letter.

We already defined variables in inventory files (remember `ansible_host`, `ansible_user` etc) and we are going to access them in a playbook:

```
student@ansible-00-01-hivemaster:~$ vi print_vars.yml
---
- name: Host and group vars
  hosts: all
  become: true
  gather_facts: no
  tasks:
    - name: Print ansible_hostname
      debug:
        var: ansible_host

    - name: Print ansible_user
      debug:
        var: ansible_user

    - name: Print ansible_ssh_private_key_file
      debug:
        var: ansible_ssh_private_key_file
```

Notice that we disabled gathering facts! Run the playbook and explore the values of variables:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook print_vars.yml

PLAY [Vars from inventory]
*****
TASK [Print ansible_hostname]
*****
ok: [ubuntu] => {
  "ansible_host": "10.128.0.49"
}
ok: [centos] => {
  "ansible_host": "10.142.15.213"
}
ok: [hivemaster] => {
  "ansible_host": "10.128.0.48"
}

TASK [Print ansible_user]
*****
ok: [ubuntu] => {
  "ansible_user": "ansible"
}
ok: [centos] => {
  "ansible_user": "ansible"
```

```

}
ok: [hivemaster] => {
  "ansible_user": "ansible"
}

TASK [Print ansible_ssh_private_key_file]
*****
ok: [ubuntu] => {
  "ansible_ssh_private_key_file": "/home/student/ansible_key"
}
ok: [centos] => {
  "ansible_ssh_private_key_file": "/home/student/ansible_key"
}
ok: [hivemaster] => {
  "ansible_ssh_private_key_file": "/home/student/ansible_key"
}

```

We also defined in the inventory file some variables (`apache_port` and `apache_path`) just for `webserver` group (ubuntu and centos), so let's add 2 more debug tasks:

```

- name: Print apache_port
  debug:
    var: apache_port

- name: Print apache_path
  debug:
    var: apache_path

```

Running the playbook you will notice that for our hivemaster server these 2 variables will be listed as “NOT DEFINED”, because this host is not inside the webserver group. We can also change the targeted hosts of this playbook to affect only webserver group:

```

---
- name: Host and group vars
  hosts: webserver
  become: true
  gather_facts: no

```

## Task 5: Defining variables

To begin with, we are going to define some variables and assign some values from facts to those variables:

```

student@ansible-00-01-hivemaster:~$ vi variables.yml
---
- name: Facts - Variables
  hosts: all
  become: true
  vars:
    - ubuntu_ip: "{{ hostvars['ubuntu']['ansible_default_ipv4']['address'] }}"
    - ubuntu_hostname: "{{ hostvars['ubuntu']['ansible_hostname'] }}"

```

```
- centos_ip: "{{ hostvars['centos']['ansible_default_ipv4']['address'] }}"
- centos_hostname: "{{ hostvars['centos']['ansible_hostname'] }}"
- hivemaster_ip: "{{
hostvars['hivemaster']['ansible_default_ipv4']['address'] }}"
- hivemaster_hostname: "{{ hostvars['hivemaster']['ansible_hostname'] }}"

tasks:
- name: Print ansible hostname
  debug:
    msg: "{{ ansible_hostname }}"

- name: Print IP address
  debug:
    msg: "{{ ansible_default_ipv4.address }}"

- name: Print inventory hostname
  debug:
    msg: "{{ inventory_hostname }}"
```

Running the playbook will print the information about defined variables, populated with values from facts:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook variables.yml

PLAY [Facts - Variables]
*****
TASK [Gathering Facts]
*****
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Print ansible hostname]
*****
ok: [ubuntu] => {
  "msg": "ansible-00-02-ubuntu"
}
ok: [centos] => {
  "msg": "ansible-00-03-centos"
}
ok: [hivemaster] => {
  "msg": "ansible-00-01-hivemaster"
}
[...]
```

## Task 6: Including variables

We can also include variables from other files using `include_vars` directive. Let's create first 2 var files:

```
student@ansible-00-01-hivemaster:~$ vi v1.yml
---
variable1: "Hello "
```

```
student@ansible-00-01-hivemaster:~$ vi v2.yml
---
variable2: "WORLD"
```

Now create a playbook and include these 2 files:

```
student@ansible-00-01-hivemaster:~$ vi include_vars.yml
---
- name: Including vars
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: include v1
      include_vars: "/home/student/v1.yml"

    - name: include v2
      include_vars: "/home/student/v2.yml"

- debug:
    msg: "{{ variable1 + variable2 }}"
```

Run the playbook:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook include_vars.yml
[...]
TASK [debug]
*****
ok: [hivemaster] => {
    "msg": "Hello WORLD"
}
[...]
```

You are going to learn in Ansible Vault lab how to include more files at once in a playbook.

### Task 7: Extra variables

Firstly, let's create a simple playbook with a basic task (return the sum of 2 vars):

```
student@ansible-00-01-hivemaster:~$ vi sum.yml
---
- name: Sum of 2 vars
  hosts: hivemaster
  gather_facts: no
  vars:
    var1: 2
    var2: 3
  tasks:
    - name: Print sum of vars
```

```
debug:
  msg: "{{ var1 + var2 }}"
```

Basically, this task just performs the sum of 2 vars:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook sum.yml

PLAY [Sum]
*****
TASK [Print sum of vars]
*****
ok: [hivemaster] => {
  "msg": "5"
}
```

Right now we are going to pass values for `var1` and `var2` from command line while running the playbook as extra vars:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook sum.yml -e
"var1=20" -e "var2=30"
```

You probably expect that the new values for our variables will replace the older ones (defined in the playbook) as the extra vars has precedence, but what you (probably) don't expect is that extra vars are treated like "strings" and the output is concatenation of these 2 strings:

```
TASK [Print sum of vars]
*****
ok: [hivemaster] => {
  "msg": "2030"
}
```

Notice that we are going to fix this behavior during "Jinja filters" section in the Templates lab!

## Task 8: Variables vs Facts

There is a main difference between variables and facts and we are going to see it by creating 2 separate playbooks and compare them:

```
student@ansible-00-01-hivemaster:~$ vi vars_vs_facts_1.yml
---
- name: Facts vs Vars
  hosts: hivemaster
  vars:
    var_time: "Var: {{lookup('pipe', 'date \"+%H:%M:%S\"')}}"
```

```
tasks:
  - name: Print var_time first time
    debug:
      msg: "{{ var_time }}"
```

```
- name: Sleep a little bit
  pause:
    seconds: 5

- name: Print var_time second time
  debug:
    msg: "{{ var_time }}"
```

Running this playbook will result in printing the `var_time` variable 2 times with a 5 seconds delay. Each time the variable is evaluated:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook vars_vs_facts_1.yml

PLAY [Facts vs Vars]
*****
TASK [Gathering Facts]
*****
ok: [hivemaster]

TASK [Print var_time first time]
*****
ok: [hivemaster] => {
  "msg": "Var: 19:09:29"
}

TASK [Sleep a little bit]
*****
Pausing for 5 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [hivemaster]

TASK [Print var_time second time]
*****
ok: [hivemaster] => {
  "msg": "Var: 19:09:34"
}
```

In the second playbook we are going to set a fact instead of a variable:

```
student@ansible-00-01-hivemaster:~$ vi vars_vs_facts_2.yml
---
- name: Facts vs Vars
  hosts: hivemaster
  tasks:
    - name: Set fact task
      set_fact:
        fact_time: "{{lookup('pipe', 'date \"+%H:%M:%S\"')}}"
    - name: Print fact_time first time
      debug:
        msg: "{{ fact_time }}"

    - name: Sleep a little bit
```



```
    pause:
      seconds: 5

- name: Print fact_time second time
  debug:
    msg: "{{ fact_time }}"
```

Running this playbook you will notice that the same value is printed both times (which means that the fact is set once during the execution and its value is not changed anymore):

```
student@ansible-00-01-hivemaster:~$ ansible-playbook vars_vs_facts_2.yml

PLAY [Facts vs Vars]
*****

TASK [Gathering Facts]
*****
ok: [hivemaster]

TASK [Set fact task]
*****
ok: [hivemaster]

TASK [Print fact_time first time]
*****
ok: [hivemaster] => {
  "msg": "19:11:25"
}

TASK [Sleep a little bit]
*****
Pausing for 5 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [hivemaster]

TASK [Print fact_time second time]
*****
ok: [hivemaster] => {
  "msg": "19:11:25"
}
```

# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



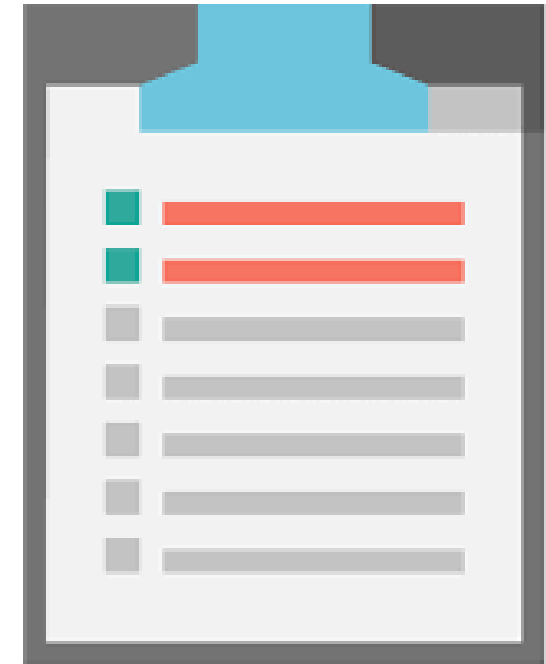
FASTER  
SMARTER  
SAFER

# 6. Loops and Conditionals



# Topics covered:

- What are Loops
- Using Conditionals
- Multiple Conditionals
- Combining Loops and Conditionals



# What are Loops?

- The **loop** keyword allows you to iterate through a simple list of items
- Before Ansible 2.5, the **with\_X** syntax was used instead
  - with\_items, with\_cartesian, with\_subelement, etc

```
-name: start some services
service:
  name: "{{ item }}"
  state: started
loop:
  - vsftpd
  - httpd
```

# Using Variables to Define a Loop

- The list that loop is using can be defined by a variable:

```
vars:
  my_services:
    - httpd
    - vsftpd
tasks:
  - name: start some services
    service:
      name: "{{ item }}"
      state: started
    loop: "{{ my_services }}"
```

# Using Hashes/Dictionary in Loops

- Each item in a loop can be a hash/dictionary with multiple keys

```
- name: create users using a loop
hosts: all
tasks:
- name: create users
  user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  loop:
    - name: anna
      groups: wheel
    - name: linda
      groups: users
```

# loop vs. with\_\*

- The **loop** keyword is the current keyword
- In previous version of Ansible, the **with\_\*** keyword was used for the same purpose
- It is recommended to move towards the **loop** syntax.
  - **with\_list**: equivalent to the loop keyword
  - **with\_items**: equivalent to **loop** + the **flatten** filter
  - **with\_random**: equivalent to **loop** + the **random** filter
  - More details:  
[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_loops.html#migrating-to-loop](https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#migrating-to-loop)



# Using Conditionals

- A condition can be used to run a task only if specific conditions are true.
- Playbook variables, registered variables, and fact can be used in conditions and make sure that tasks only run if specific conditions are true.
- For instance, check if a task has run successfully, a certain amount of memory is available, a file exists, etc.
- **when** statements are used to run a task conditionally

# Defining Simple Conditionals

- The simplest example of a condition, is to check whether a Boolean variable is true or false.
  - Note that **when** conditions are always evaluated in Jinja context, so the double braces are **not** required!
- You can also check and see if a non-Boolean variable has a value and use that value in the conditional.
- Or use a conditional in which you compare the value of a fact to the specific value of a variable.

# Simple Conditionals Example

- **ansible\_machine == "x86\_64"**
- **ansible\_distribution\_major\_version == "8"**
- **ansible\_memfree\_mb == 1024**
- **ansible\_memfree\_mb < 256**
- **ansible\_memfree\_mb > 256**
- **ansible\_memfree\_mb <= 256**
- **ansible\_memfree\_mb != 512**
- **my\_variable is defined**
- **my\_variable is not defined**
- **ansible\_distribution in supported\_distros**

# Simple Conditions Example

```
---
- name: when example
  host: all
  vars:
    supported_distros:
      - CentOS
      - Fedora
      - RedHat
  tasks:
    - name: install RH family specific packages
      yum:
        name: nginx
        state: present
      when: ansible_distribution in supported_distros
```

# Multiple Conditions

- **when** can be used to test multiple conditions as well
- Use **and/or** and group the conditions with parentheses
  - `when: ansible_distribution == "CentOS" or ansible_distribution == "RedHat"`
  - `when: ansible_machine == "x86_64" and ansible_distribution == "CentOS"`
- The `when` keyword also supports a list and when using a list, all of the conditions must be true.
  - Much easier to read than using **and**!
- Complex conditional statements can group conditions using parentheses

# Multiple Conditions - Example

---

- name: Using conditionals
  - hosts: centos
  - gather\_facts: yes
  - become: true
  - vars:
    - backup: true
  - tasks:
    - stat:
      - path: /etc/ssh/sshd\_config
      - register: result
- name: Backup ssh configuration
  - fetch:
    - src: /etc/ssh/sshd\_config
    - dest: ./sshd\_config-{{ ansible\_hostname }}
    - flat: yes
  - when: result.stat.exists and result.stat.isreg and backup == true

# Multiple Conditions - Example

---

```
- name: when example
  host: all
  tasks:
    - package:
        name: httpd
        state: installed
      when: >
        ( ansible_distribution == "RedHat" and
          ansible_memfree_mb > 512 )
        or
        ( ansible_distribution == "CentOS" and
          ansible_memfree_mb > 1024 )
```

# Combining Loops and Conditionals

- Loops and conditionals can be combined
- For instance, you can iterate through a list of dictionaries and apply the conditional statement only if a dictionary is found that matches the condition



# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - command: echo {{ item }}
      loop: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix
      ignore_errors: yes
      register: result
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix 1
      ignore_errors: yes
      register: result
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

1 Is Postfix running or not?

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix 1
      ignore_errors: yes 2
      register: result
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

- 1 Is Postfix running or not?      2 If it is not running and the command “fails”, do not stop processing

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix 1
      ignore_errors: yes 2
      register: result 3
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

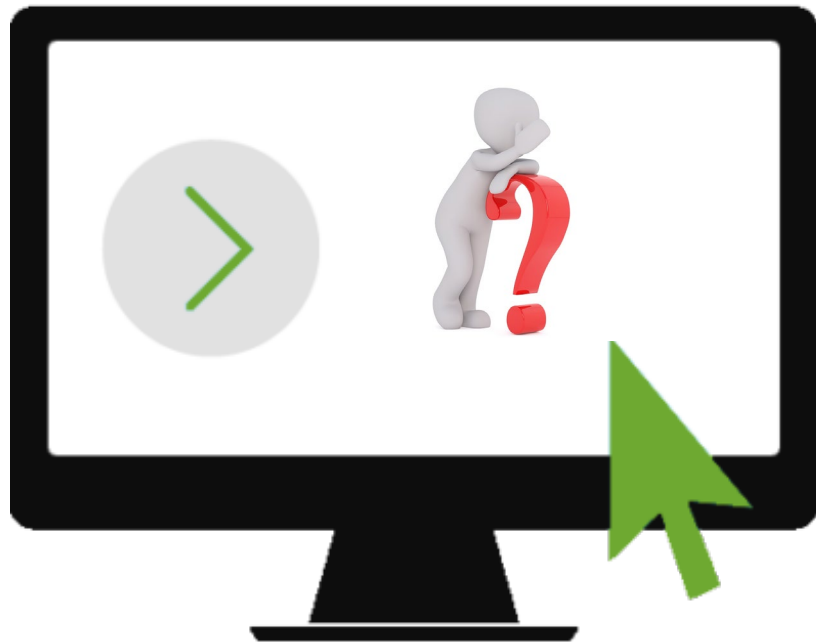
- 1 Is Postfix running or not?
- 2 If it is not running and the command “fails”, do not stop processing
- 3 Save information on the module’s result in a variable named **result**

# Combining Loops and Conditionals

```
- hosts: all
  tasks:
    - name: Postfix server status
      command: /usr/bin/systemctl is-active postfix ❶
      ignore_errors: yes ❷
      register: result ❸
    - name: Restart Apache HTTPD if Postfix running
      service:
        name: httpd
        state: restarted
      when: result.rc == 0 ❹
```

- ❶ Is Postfix running or not?
- ❷ If it is not running and the command “fails”, do not stop processing
- ❸ Save information on the module’s result in a variable named **result**

- ❹ Evaluates the output of the Postfix task. If the exit code of the **systemctl** command is 0, then Postfix is active and this task will restart the **httpd** service



# Lab 6: Loop Conditionals







Solutions for training the world.



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Loops. Conditionals.

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: loop vs with_<lookup>	2
Task 1.1: with_items	2
Task 1.2: loop	3
Task 1.3: Registering variables with a loop	4
Task 1.4: Iterate over inventory file	5
Task 2: Conditionals	6
Task 2.1: "when: var == true"	6
Task 2.2: "when" file exists	7
Task 2.3: "when" multiple conditions	7
Task 2.4: "when" with Facts	8
Task 2.5: Loop and "when" on the same task using multiple conditions	10
Task 2.6: Setup webserver and dbserver	11

# Lab Overview and objectives

The purpose of this lab is to get used with Ansible loops and conditionals. You are probably familiar with the concept of loop from computer programming, but Ansible loops include changing ownership on files or directories with the file module, creating multiple users with the user module and also taking decisions based on conditional clauses.

## Guided Tasks

### Task 1: loop vs with\_<lookup>

Ansible provides two methods for creating loops: `loop` and `with_<lookup>`. `loop` was added in Ansible 2.5 and it doesn't replaced the older `with_<lookup>` method (until now), but it is the recommended way to implement loops.

#### Task 1.1: with\_items

Let's use `with_items` to install/uninstall a list of packages from our `hivemaster` host:

```
student@ansible-00-01-hivemaster:~$ vi with_items.yml
---
- name: Manage packages
  hosts: hivemaster
  become: true
  tasks:
    - name: Install packages
      apt:
        name: "{{ item.name }}"
        state: "{{ item.state }}"
      with_items:
        - { name: wget, state: present }
        - { name: nmap, state: present }
        - { name: apache2, state: absent }
```

Notice the apt module used right here, which is the package manager for Debian distribution.

Running the playbook should result in installing nmap (as wget is already installed and apache2 is absent):

```
student@ansible-00-01-hivemaster:~$ ansible-playbook with_items.yml

PLAY [Manage packages]
*****
```

```

TASK [Gathering Facts]
*****
ok: [hivemaster]

TASK [Install packages]
*****
ok: [hivemaster] => (item={u'state': u'present', u'name': u'wget'})
changed: [hivemaster] => (item={u'state': u'present', u'name': u'nmap'})
ok: [hivemaster] => (item={u'state': u'absent', u'name': u'apache2'})

PLAY RECAP
*****
hivemaster          : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

```

## Task 1.2: loop

We are going to use `loop` in this example to iterate through a simple list and also through a dictionary:

```

student@ansible-00-01-hivemaster:~$ vi loop_1.yml
---
- name: Test loop over list and dict
  hosts: hivemaster
  gather_facts: no
  vars:
    my_grocery_list:
      - milk
      - bread
      - cereal
      - beer
    my_car_preferences:
      brand: mercedes
      model: G
      year: 2018
  tasks:
    - name: Loop over list
      debug:
        msg: "{{ item }}"
      loop: "{{ my_grocery_list }}"

    - name: Loop over dict
      debug:
        msg: "{{ item.key }} -> {{ item.value }}"
      loop: "{{ my_car_preferences | dict2items }}"

```

Run the playbook:

```

student@ansible-00-01-hivemaster:~$ ansible-playbook loop_1.yml

PLAY [Test loop over list and dict]
*****

```

```
TASK [Loop over list]
*****
ok: [hivemaster] => (item=milk) => {
  "msg": "milk"
}
ok: [hivemaster] => (item=bread) => {
  "msg": "bread"
}
ok: [hivemaster] => (item=cereal) => {
  "msg": "cereal"
}
ok: [hivemaster] => (item=beer) => {
  "msg": "beer"
}

TASK [Loop over dict]
*****
ok: [hivemaster] => (item={'key': u'brand', 'value': u'mercedes'}) => {
  "msg": "brand -> mercedes"
}
ok: [hivemaster] => (item={'key': u'model', 'value': u'G'}) => {
  "msg": "model -> G"
}
ok: [hivemaster] => (item={'key': u'year', 'value': 2018}) => {
  "msg": "year -> 2018"
}
```

### Task 1.3: Registering variables with a loop

When you use `register` with a `loop`, the variable registered will contain a list of all responses from the module. This differs from the data structure returned when using `register` without a loop:

```
student@ansible-00-01-hivemaster:~$ vi register_loop.yml
---
- name: Register var with a loop
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: Print some values
      shell: "echo {{ item }}"
      loop:
        - value1
        - value2
        - value3
      register: output

    - name: Print output
      debug:
        var: output
```

```
student@ansible-00-01-hivemaster:~$ ansible-playbook register_loop.yml
```

```
PLAY [Register var with a loop]
*****
TASK [Print some values]
*****
changed: [hivemaster] => (item=value1)
changed: [hivemaster] => (item=value2)
changed: [hivemaster] => (item=value3)

TASK [Print output]
*****
ok: [hivemaster] => {
  "output": {
    "changed": true,
    "msg": "All items completed",
    "results": [
      {
        "ansible_facts": {
          "discovered_interpreter_python": "/usr/bin/python3"
        },
        "ansible_loop_var": "item",
        "changed": true,
        "cmd": "echo value1",
        "delta": "0:00:00.002954",
        "end": "2019-11-26 20:57:40.161126",
        "failed": false,
        "invocation": {
          "module_args": {
            "_raw_params": "echo value1",
            "_uses_shell": true,
            "argv": null,
            "chdir": null,
            "creates": null,
            "executable": null,
            "removes": null,
            "stdin": null,
            "stdin_add_newline": true,
            "strip_empty_ends": true,
            "warn": true
          }
        },
        "item": "value1",
        "rc": 0,
        "start": "2019-11-26 20:57:40.158172",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "value1",
        "stdout_lines": [
          "value1"
        ]
      }
    ]
  }
}
```

#### Task 1.4: Iterate over inventory file

We can also use loop to iterate over our hosts from the inventory file (all of them or just some groups):

```
student@ansible-00-01-hivemaster:~$ vi loop_inventory.yml
---
- name: Loop over inventory file
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: Print inventory hosts
      debug:
        msg: "{{ item }}"
      loop: "{{ groups['webservers'] }}"
```

We looped over hosts from ‘webservers’ group:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook loop_inventory.yml

PLAY [Loop over inventory file]
*****
TASK [Print inventory hosts]
*****
ok: [hivemaster] => (item=ubuntu) => {
    "msg": "ubuntu"
}
ok: [hivemaster] => (item=centos) => {
    "msg": "centos"
}
[...]
```

## Task 2: Conditionals

Ansible provides a method for implementing conditionals using the “when” clause. The syntax is simple “when: condition” and we can use this with any module to control when they are executed.

### Task 2.1: “when: var == true”

Let’s suppose that we want to backup SSH configuration for a host when some variable (called “backup” is set to “true”). Also make sure that you customize the backup including the name of the host on the backup file so that the file doesn’t get overwritten if we run the playbook against several hosts on the same time.

```
student@ansible-00-01-hivemaster:~$ vi when_clause_1.yml
---
- name: Using conditionals
  hosts: centos
  gather_facts: yes
  become: true
  vars:
    backup: true
```



```
tasks:
- name: Backup ssh configuration
  fetch:
    src: /etc/ssh/sshd_config
    dest: ./sshd_config-{{ ansible_hostname }}
    flat: yes
    when: backup == true
```

### Task 2.2: “when” file exists

We can check if a file (or directory) exists using “stat” module. Then we can customize the execution of the playbook using the result from the stat module. So, let’s start from the previous example and create the backup of SSH configuration only if the sshd\_config file exists (make sure that you remove the previous backup from your hivemaster host, otherwise Ansible will just return ok):

```
student@ansible-00-01-hivemaster:~$ rm -rf sshd_config-ansible-00-03-centos
student@ansible-00-01-hivemaster:~$ vi when_clause_2.yml
---
- name: Using conditionals
  hosts: centos
  gather_facts: yes
  become: true
  vars:
    backup: true
  tasks:
  - stat:
    path: /etc/ssh/sshd_config
    register: result

  - name: Backup ssh configuration
    fetch:
      src: /etc/ssh/sshd_config
      dest: ./sshd_config-{{ ansible_hostname }}
      flat: yes
      when: result.stat.exists
```

We used in this example parameters “exists”, which returns if the destination path exists or not. There are also other parameters like “isdir”, “isreg” which returns if the path is a directory or a regular file. You can read more about stat module at [https://docs.ansible.com/ansible/latest/modules/stat\\_module.html](https://docs.ansible.com/ansible/latest/modules/stat_module.html)

### Task 2.3: “when” multiple conditions

Starting from the same example, let’s condition the backup on more conditions at once:

```
---
- name: Using conditionals
  hosts: centos
```

```
gather_facts: yes
become: true
vars:
  backup: true
tasks:
- stat:
    path: /etc/ssh/sshd_config
    register: result

- name: Backup ssh configuration
  fetch:
    src: /etc/ssh/sshd_config
    dest: ./sshd_config-{{ ansible_hostname }}
    flat: yes
    when: result.stat.exists and result.stat.isreg and backup == true
```

In this example we used “**and**” operator, but it also possible (as you expect) to use “**or**” operator with **when** clause. So, let’s suppose that we want to make the backup on every “27” day of the month even if the **backup** variable is not set to true:

```
---
- name: Using conditionals
  hosts: centos
  gather_facts: yes
  become: true
  vars:
    backup: false
  tasks:
- stat:
    path: /etc/ssh/sshd_config
    register: result

- name: Backup ssh configuration
  fetch:
    src: /etc/ssh/sshd_config
    dest: ./sshd_config-{{ ansible_hostname }}
    flat: yes
    when: (result.stat.exists and result.stat.isreg and backup == true)
or (ansible_date_time.day == "27")
```

#### Task 2.4: “when” with Facts

We can also use **Ansible facts** as conditions for “when” clause, so let’s reboot all hosts from inventory that are running CentOS version 7:

```
student@ansible-00-01-hivemaster:~$ vi when_clause_3.yml
---
- name: Reboot all hosts running CentOS 7
  hosts: all
```

```
gather_facts: yes
become: true
tasks:
- name: Rebooting...
  reboot:
    msg: "System is going to reboot now!"
    reboot_timeout: 40
    test_command: uptime
    when: ansible_facts['distribution'] == "CentOS" and
ansible_facts['distribution_major_version'] == "7"
    register: result

- name: Print registered result
  debug:
    var: result
```

In this example we used the reboot module, with a few parameters: `msg` is the message to be displayed for users, `reboot_timeout` is the period of time (in seconds) to wait for the machine to reboot and respond to a test command, and `test_command` which is set by default to `"whoami"`. Notice that we also registered the output of the task as a variable called `result` and displayed in the next task its content:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook when_clause_3.yml

PLAY [Reboot all hosts running CentOS 7]
*****

TASK [Gathering Facts]
*****
ok: [centos]
ok: [ubuntu]
ok: [hivemaster]

TASK [Rebooting...]
*****
skipping: [ubuntu]
skipping: [hivemaster]
changed: [centos]

TASK [Print registered result]
*****
ok: [ubuntu] => {
  "result": {
    "changed": false,
    "skip_reason": "Conditional result was False",
    "skipped": true
  }
}
ok: [centos] => {
  "result": {
    "changed": true,
    "elapsed": 27,
```

```

        "failed": false,
        "rebooted": true
    }
}
ok: [hivemaster] => {
    "result": {
        "changed": false,
        "skip_reason": "Conditional result was False",
        "skipped": true
    }
}

```

#### PLAY RECAP

```

*****
centos           : ok=3    changed=1    unreachable=0    failed=0
skipped=0       rescued=0  ignored=0
hivemaster       : ok=2    changed=0    unreachable=0    failed=0
skipped=1       rescued=0  ignored=0
ubuntu          : ok=2    changed=0    unreachable=0    failed=0
skipped=1       rescued=0  ignored=0

```

Running this playbook you will notice another output status “**skipped**” for `ubuntu` and `hivemaster` (we already know “**OK**”, “**changed**” and “**failed**” from a previous lab) which is returned because “**Conditional result was False**”, but this is exactly what we intended to do (apply the reboot task only for the host(s) running CentOS 7).

#### Task 2.5: Loop and “when” on the same task using multiple conditions

```

---
- name: Install several packages on web servers
  hosts: web servers
  gather_facts: yes
  vars:
    my_var: nginx
    my_list:
      - curl
      - wget
      - htop
      - nginx

  tasks:
    - name: Loop with conditional clause
      debug:
        msg: "{{ item }}"
      loop: "{{ my_list }}"
      when: item == my_var

```

Notice that this playbook does not install these packages, it just iterates through `my_list` and prints the `item` name when condition is met.

## Task 2.6: Setup webserver and dbserver

Our hosts have 2 different distributions (Debian and RedHat). As you already seen, they use different package managers (`apt` and `yum`), so we have to apply some conditions if we want to run the same installation playbook against both of them. There is also a module called `package` which can be used for both distributions, but just in case that the package has the same name.

Let's say that we want to install **Apache** on our `webserver`s (Apache package is called `apache2` on Ubuntu and `httpd` on CentOS). For our `dbserver`s we are going to install `MySQL`.

```
student@ansible-00-01-hivemaster:~$ vi installer.yml
---
- name: Install packages on webserver and dbserver
  hosts: all
  become: yes
  gather_facts: yes
  tasks:
    - name: Set proper name for apache package
      set_fact:
        apache_package: "apache2"
      when: ansible_facts['os_family'] == "Debian"

    - name: Set proper name for apache package
      set_fact:
        apache_package: "httpd"
      when: ansible_facts['os_family'] == "RedHat"

    - name: Install packages for webserver
      package:
        name:
          - "{{ apache_package }}"
          - wget
          - curl
          - htop
        state: latest
        notify: start apache
        when: "'webserver' in group_names"

    - name: Install packages for dbserver
      package:
        name:
          - python-mysqldb
          - mysql-server
        update_cache: yes
        state: latest
        when: "'dbserver' in group_names"

  handlers:
    - name: start apache
      service:
        name: "{{ apache_package }}"
```

```
enabled: yes
state: started
```

This is a more complex playbook, because it performs the installation for all packages necessary for our groups. We set `apache_package` fact for every host according to the distribution and installed the packages using `general_package` module.

Notice that we didn't use a loop explicitly, but we passed a list of packages to the `package` module, as this is the recommended way to install several several packages at one time (loop is considerably slower in this case).

We also used conditional clauses to install packages just for the hosts that belong to certain groups.

Running the playbook should perform the proper installation of all packages:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook installer.yml

PLAY [Install packages on webserver and dbserver]
*****

TASK [Gathering Facts]
*****
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Set proper name for apache package]
*****
ok: [ubuntu]
skipping: [centos]
ok: [hivemaster]

TASK [Set proper name for apache package]
*****
skipping: [ubuntu]
ok: [centos]
skipping: [hivemaster]

TASK [Install packages for webserver]
*****
skipping: [hivemaster]
changed: [ubuntu]
changed: [centos]

TASK [Install packages for dbserver]
*****
skipping: [centos]
changed: [hivemaster]
changed: [ubuntu]
[...]
```

Notice that we also added a `handler` to start and enable `apache` after installation. This handler is called only when the task is returning changed!

If you encounter any errors with `ubuntu` host during installation of packages please login using ssh and perform a “sudo apt update”, because this is not available using package module.



# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



FASTER  
SMARTER  
SAFER



## 7. Working with Templates



# Templates Basics

- Templates give the ability to provide a skeletal file that can be dynamically completed using variables.
- The most common template use case is configuration file management.
- Templates are generally used by providing a template file on the ansible control node, and then using the template module within your playbook to deploy the file to a target server or group.
- Templates are processed using the Jinja2 template language.

# Template Module

- The template module is used to deploy template files
- There are two required parameters:
  - **src** : the template to use (on the ansible control host)
  - **dest** : where the resulting file should be located (on the target host)
- All templating takes place **on the control host!**

# Template Module

- A useful optional parameter is **validate** which requires a successful validation command to run against the result file prior to deployment
- It is also possible to set basic file properties using the template module

# Template Module

```
- name: Update sshd configuration safely, avoid locking yourself out
  template:
    src: etc/ssh/sshd_config.j2
    dest: /etc/ssh/sshd_config
    owner: root
    group: root
    mode: '0600'
    validate: /usr/sbin/sshd -t -f %s
    backup: yes
```

# Template Module

- These are some of the other **parameters** which we can use to change some default behavior of template module:
  - **force** – If the destination file already exists, then this parameter decides whether it should be replaced or not. By default, the value is 'yes'.
  - **mode** – If you want to set the permissions for the destination file explicitly, then you can use this parameter.
  - **backup** – If you want a backup file to be created in the destination directory, you should set the value of the backup parameter to 'yes'. By default, the value is 'no'.
  - **group** – Name of the group that should own the file/directory.

# Template Module

- Additional variables that can be used in templates:
  - **ansible\_managed**: contains a string which can be used to describe the template name, host, modification time of the template file and the owner uid.
  - **template\_host**: contains the node name of the template's machine.
  - **template\_uid**: is the numeric user id of the owner.
  - **template\_path**: is the path of the template.
  - **template\_fullpath**: is the absolute path of the template.
  - **template\_destpath**: is the path of the template on the remote system
  - **template\_run\_date**: is the date that the template was rendered.

# Template File

- Template file are essentially little more than text files
- Template files are designed with a file extension of `.j2`
- Template files have access to the same variables that the play that calls them does



# Template File Example

```
student@ansible-00-hivemaster:~$ vi template.j2
Nickname: {{ nickname }}
Email Address: {{ email }}
Description: {{ description }}
Role: {{ role }}
Organization: {{ org }}
```

Note: From the next lab.

# Filters

- Ansible uses Jinja2 filters for transforming data inside a template expression.
- Take into account that templating happens **on the Ansible controller**, **not** on the task's target host.
- Ansible adds some additional filters to the Jinja2 default ones, and you can also write your own custom filters

# Filter Examples



# Formatting Data

- Reading and writing JSON/YAML:

```
{{ variable | to_json }}
```

```
{{ variable | to_nice_json(indent=2) }}
```

```
{{ variable | to_yaml }}
```

```
{{ variable | to_nice_yaml(indent=4) }}
```

```
{{ variable | from_json }}
```

```
{{ variable | from_yaml }}
```

# Mandatory and Default

- By default, ansible will fail when an undefined variable is used
  - This can be turned off from ansible.cfg ( `error_on_undefined_vars = False` )
  - With the setting turned off, we can still force some variables to be defined:

```
{{ variable | mandatory }}
```

- Alternatively, we can set defaults for undefined vars:

```
{{ variable | default('abc') }}
```

# Dictionary to Items

- Convert a dictionary into a list of key/value items, suitable for looping:

```
{{ dict | dict2items }}
```

```
{{ dict | dict2items(key_name='file_name', value_name='file_path') }}
```

New in Ansible 2.8!

```
files:  
  users: /etc/passwd  
  groups: /etc/group
```

```
- file_name: users  
  file_path: /etc/passwd  
- file_name: groups  
  file_path: /etc/group
```

# Subelements

- Product of an object with subelement values of that object (similar to the *subelements* lookup):

```
users:
- name: alice
  authorized:
    - /tmp/alice/onekey.pub
    - /tmp/alice/twokey.pub
  groups:
    - wheel
    - docker
- name: bob
  authorized:
    - /tmp/bob/id_rsa.pub
  groups:
    - docker
```

```
{{ users | subelements('authorized') }}
```

```
-
- name: alice
  groups:
    - wheel
    - docker
  authorized:
    - /tmp/alice/onekey.pub
    - /tmp/alice/twokey.pub
    - /tmp/alice/onekey.pub
-
- name: alice
  groups:
    - wheel
    - docker
  authorized:
    - /tmp/alice/onekey.pub
    - /tmp/alice/twokey.pub
    - /tmp/alice/twokey.pub
-
- name: bob
  authorized:
    - /tmp/bob/id_rsa.pub
  groups:
    - docker
    - /tmp/bob/id_rsa.pub
```

# Network Utilities

- Random Mac Address Filter

- This filter can be used to generate a random MAC address from a string prefix.

```
{{ 'aa:bb:cc' | random_mac }}  
# => 'aa:bb:cc:13:a4:b3'
```

- IP Address Filters

- Check if a string is a valid IP address

```
{{ var | ipaddr }}
```

- Extract IP addresses from a CIDR block

```
{{ '10.11.12.13/24' | ipaddr(2) }}  
# => '10.11.12.2/24'
```



# Other Filters

- Display the underlying Python type of a variable

```
{{ myvar | type_debug }}
```

- Ternary operator (a?b:c)

```
{{ a | ternary(b,c) }}
```

# Checking a Template. The 'template' Lookup

```
> cat template_test.yml
- name: Test a template
  hosts: localhost
  vars:
    mylist:
      - 1
      - 2
      - a
      - b
      - c
  gather_facts: no

  tasks:
    - name: Print templated output
      debug:
        msg: "{{ lookup('template', 'template.j2').split('\n') }}"
```

# Loops in Templates

```
{% for i in mylist %}
{{i}}
{% endfor %}
```

```
> ansible-playbook template_test.yml
```

```
TASK [Print templated output]
*****
ok: [localhost] => {
  "msg": [
    "1",
    "2",
    "a",
    "b",
    "c",
    ""
  ]
}
```

# Trimming Whitespace

```
{% for i in mylist %}  
    {{i}}  
{%- endfor %}
```

```
> ansible-playbook template_test.yml
```

```
TASK [Print templated output]
```

```
*****
```

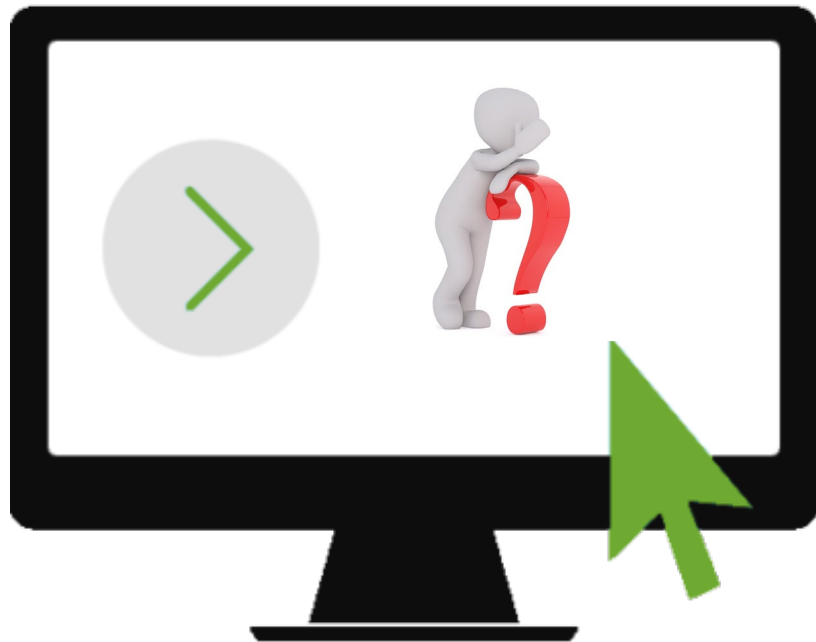
```
ok: [localhost] => {
```

```
  "msg": [
```

```
    " 1 2 a b c",
```

```
  ]
```

```
}
```



# Lab 07: Templates





Solutions for training the world.



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Working with Templates



## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Template Module	2
Task 1: Template module – basic usage	2
Task 2: Customize webservers home page using a template	3
Task 3: Using “for” in Jija2 template	4
Jinja filters	5
Task 4: Number filters (int, abs, round)	5
Task 5: Forcing variables to be defined	6
Task 6: Defaulting variables	7
Task 7: List filters	7
Task 8: Random and Shuffle filters	9
Task 9: IP Address filters	10
Task 10: Hashing filters	10

# Lab Overview and objectives

The purpose of this lab is to learn how to use Jinja2 templates in Ansible in order to apply custom modifications on the content of your files. For simple modifications **lineinfile** and **blockinfile** modules can be used, but for advanced modifications, which contains also variables and facts using Jinja2 templates is what we need.

## Guided Tasks

### Template Module

#### Task 1: Template module – basic usage

In order to use Jinja2 templates in Ansible to perform modifications in files we have to use **template** module, which has mainly two parameters: **src** and **dest**, exactly like **copy** module, because the usage is similar somehow, except that the **src** file contains references to variables or facts. Let's make a simple example to see how templating works. We are going to create first the template file (**src** file), which usually has the **.j2** extension just to know that is a Jinja2 template:

```
student@ansible-00-01-hivemaster:~$ vi template.j2
Nickname: {{ nickname }}
Email Address: {{ email }}
Description: {{ description }}
Role: {{ role }}
Organization: {{ org }}
```

Next, we create the playbook:

```
student@ansible-00-01-hivemaster:~$ vi templating.yml
---
- hosts: hivemaster
  become: yes
  gather_facts: no
  vars:
    nickname: bogdan
    email: bogdan@sass.ro
    description: IT guy
    role: admin
    org: SASS Training
  tasks:
    - name: deploy file from template
      template:
        src: template.j2
```

```
dest: /tmp/info.txt
```

Run the playbook (`ansible-playbook templating.yml`) and explore the results in `/tmp/info.txt`:

```
student@ansible-00-01-hivemaster:~$ cat /tmp/info.txt
Nickname: bogdan
Email Address: bogdan@sass.ro
Description: IT guy
Role: admin
Organization: SASS Training
```

## Task 2: Customize webserver's home page using a template

In the previous lab we installed Apache on our hosts from `webserver` group. It's time to customize index page for each host, starting from a Jinja2 template. Let's create the template first:

```
student@ansible-00-01-hivemaster:~$ vi index.html.j2
<html>
<head>
<title>Welcome to Ansible</title>
</head>
<body>
<p>
Server details:
Hostname: {{ ansible_hostname }}
OS: {{ ansible_distribution }} {{ ansible_distribution_version }}
</p>
</body>
</html>
```

Now let's create a playbook to deploy this template to webserver's:

```
student@ansible-00-01-hivemaster:~$ vi customize_homepage.yml
---
- name: Customize homepage
  hosts: webserver
  become: yes
  tasks:
    - name: Deploy file from j2 template
      template:
        src: index.html.j2
        dest: /var/www/html/index.html
```

Run the playbook and then test the results using `curl`:

```
student@ansible-00-01-hivemaster:~$ curl ansible-00-02-ubuntu
<html>
```

```
<head>
<title>Welcome to Ansible</title>
</head>
<body>
<p>
Server details:
Hostname: ansible-00-02-ubuntu
OS: Ubuntu 18.04
</p>
</body>
</html>
```

```
student@ansible-00-01-hivemaster:~$ curl ansible-00-03-centos
```

```
<html>
<head>
<title>Welcome to Ansible</title>
</head>
<body>
<p>
Server details:
Hostname: ansible-00-03-centos
OS: CentOS 7.7
</p>
</body>
</html>
```

At this point you may encounter an error trying to access the webserver from centos server. This is because the centos server has the firewall active. The default firewall software on centos is firewalld. So, make sure that you add a rule or stop the firewall in order to be able to access the webpage. Here is an example on how you can disable firewalld using ansible:

```
---
- name: Disable firewall
  hosts: webservers
  become: yes
  tasks:
    - name: Disable and stop firewalld
      service:
        name: firewalld
        enabled: no
        state: stopped
      when: ansible_facts['os_family'] == "RedHat"
```

### Task 3: Using “for” in Jija2 template

There may be some tasks when you need to iterate through a list of variables in a Jinja2 template. In order to do this we have to use some special delimiters `{% ... %}`:

```
student@ansible-00-01-hivemaster:~$ vi iterate.j2
```

```
{% for item in my_list %}
  {{ item }}
{% endfor %}
```

```
student@ansible-00-01-hivemaster:~$ vi for_in_jinja.yml
---
- name: For loop in J2
  hosts: hivemaster
  gather_facts: no
  vars:
    my_list: [Ford, Audi, BMW, Skoda, Jaguar, Citroen]
  tasks:
    - name: Deploy file from template
      template:
        src: iterate.j2
        dest: /tmp/file.txt
        mode: 0644
```

Now let's cat the content of file.txt:

```
student@ansible-00-01-hivemaster:~$ cat /tmp/file.txt
Ford
Audi
BMW
Skoda
Jaguar
Citroen
```

You can read more about Jinja2 Templates at <https://jinja.palletsprojects.com/en/2.10.x/templates/>

## Jinja2 filters

Jinja2 filters are a way to transform expressions from one type of data to another. The syntax to apply Jinja2 filters to template variables is the vertical bar character |, also called a 'pipe' in Unix environments (ex: `{{variable|filter}}`). It's worth mentioning you can apply multiple filters to the same variable (ex: `{{variable|filter|filter}}`).

### Task 4: Number filters (int, abs, round)

During “Facts and Variables” lab we used extra variables for Task 7 and you noticed that instead of calculating the correct sum of the 2 vars, Ansible returned the concatenation of the 2 values as strings. We are going to fix that right now using Jinja2 `int` filter:

```
student@ansible-00-01-hivemaster:~$ vi sum.yml
---
- name: Sum
  hosts: hivemaster
  gather_facts: no
```

```
vars:
  var1: 2
  var2: 3
tasks:
- name: Print sum of vars
  debug:
    msg: "{{ var1|int + var2|int }}"
```

```
student@ansible-00-01-hivemaster:~$ ansible-playbook sum.yml -e
"var1=20" -e "var2=30"
```

```
PLAY [Sum]
*****
TASK [Print sum of vars]
*****
ok: [hivemaster] => {
  "msg": "50"
}
```

Let's try to apply `abs` (absolute value) filter over our sum, made now from an integer and a negative float and rounded number:

```
---
- name: Sum
  hosts: hivemaster
  gather_facts: no
  vars:
    var1: 2
    var2: 3
  tasks:
  - name: Print sum of vars
    debug:
      msg: "{{ (var1|int + var2|float|round)|abs }}"
```

```
student@ansible-00-01-hivemaster:~$ ansible-playbook sum.yml -e
"var1=20" -e "var2=-30.33"
TASK [Print sum of vars]
*****
ok: [hivemaster] => {
  "msg": "10.0"
}
```

## Task 5: Forcing variables to be defined

By default Ansible fails if a variable is not defined, but this can be disabled in `ansible.cfg` by uncommenting line `error_on_undefined_vars = False`. After that, we can explicitly specify that a variable should be defined using `{{ variable | mandatory }}`. Add this task to previous playbook:

```
- name: Mandatory var
  debug:
    msg: "{{ var3 | mandatory }}"
```

```
student@ansible-00-01-hivemaster:~$ ansible-playbook sum.yml -e
"var1=20" -e "var2=-30.33"
```

```
PLAY [Sum]
*****

TASK [Print sum of vars]
*****
ok: [hivemaster] => {
  "msg": "10.0"
}

TASK [Mandatory var]
*****
fatal: [hivemaster]: FAILED! => {"msg": "Mandatory variable 'var3' not
defined."}
```

## Task 6: Defaulting variables

Using the default filter we can specify a default value for undefined variables. Modify previous task:

```
- name: Mandatory var -> defaulting var
  debug:
    msg: "{{ var3 | default(50) }}"
```

## Task 7: List filters

**min, max** filters:

```
student@ansible-00-01-hivemaster:~$ vi list_filters.yml
---
- name: List filters Playbook
  hosts: hivemaster
  gather_facts: no
  vars:
    list1:
      - 77
      - 50
      - 100
      - -400
      - 100
    list2:
      - 400
      - 77
```

```
- 300
- 100
- 77

tasks:
- name: Get minimum value of list
  debug:
    msg: "{{ list1|min }}"
```

```
student@ansible-00-01-hivemaster:~$ ansible-playbook list_filters.yml
TASK [Get minimum value of list]
*****
ok: [hivemaster] => {
  "msg": "-400"
}
```

`unique`, `union`, `intersect` filters:

```
- name: Get unique values of list
  debug:
    msg: "{{ list1|unique }}"
```

```
TASK [Get unique values of list]
*****
ok: [hivemaster] => {
  "msg": [
    77,
    50,
    100,
    -400
  ]
}
```

```
- name: Get union of two lists
  debug:
    msg: "{{ list1|union(list2) }}"
```

```
TASK [Get union of two lists]
*****
ok: [hivemaster] => {
  "msg": [
    77,
    50,
    100,
    -400,
    400,
    300
  ]
}
```



```
- name: Get intersection of two lists
  debug:
    msg: "{{ list1|intersect(list2) }}"
```

```
TASK [Get intersection of two lists]
*****
ok: [hivemaster] => {
  "msg": [
    77,
    100
  ]
}
```

## Task 8: Random and Shuffle filters

Add these tasks to the same playbook (list\_filters.yml):

```
- name: Get random number from a list
  debug:
    msg: "{{ (list1|intersect(list2))|random }}"
```

```
TASK [Get random number from a list]
*****
ok: [hivemaster] => {
  "msg": "100"
}
```

```
- name: Get random number with start and step
  debug:
    msg: "{{ 666|random(start=10, step=7) }}"
```

```
TASK [Get random number with start and step]
*****
ok: [hivemaster] => {
  "msg": "206"
}
```

```
- name: Get a list shuffled
  debug:
    msg: "{{ (list1|union(list2))|shuffle }}"
```

```
TASK [Get a list shuffled]
*****
ok: [hivemaster] => {
  "msg": [
    77,
    400,
    300,
    100,
```

```

        50,
        -400
    ]
}
```

## Task 9: IP Address filters

This is a useful filter to test if a string is a valid IP address. Notice that you have to install `netaddr` for this filter to work:

```
sudo apt-get install -y python-netaddr
```

```

- name: Test string for IP Addr
  debug:
    msg: "{{ '192.168.100.260'|ipaddr }}"
```

```

TASK [Test string for IP Addr]
*****
ok: [hivemaster] => {
  "msg": false
}
```

`ipv4`, `ipv6` can also be used to test specific protocol IP addresses.

## Task 10: Hashing filters

We already used hash filter during a previous lab when we set the password for a Linux user that we created.

```

student@ansible-00-01-hivemaster:~$ vi password.yml
---
- name: User management
  hosts: all
  become: yes
  vars:
    user_pass: 123abc
  tasks:
    - name:
      user:
        name: testuser
        password: "{{ user_pass | password_hash('sha512') }}"
```

Remember that this task always returned “changed” even if we set the same password (because the password is salted in Linux). We can set a default `salt` for this task, but it is recommended to make it particular at least for each host (not to use the same salt for the entire infrastructure). So we will use `random` filter with a host-specific seed:

```
password:  "{{ user_pass | password_hash('sha512', 65534 |
random(seed=inventory_hostname) | string) }}"
```

There are also other types of hashes available: `md5`, `checksum`, `blowfish` etc. Go back to the previous yml file and add the following tasks:

```
- name: Get md5 hash
  debug:
    msg: "{{ list1.0 | hash('md5') }}"
```

```
TASK [Get md5 hash]
*****
ok: [hivemaster] => {
  "msg": "28dd2c7955ce926456240b2ff0100bde"
}
```

## Task 11: Other useful filters

Using `quote` filter we can add quotes to a variable (useful for shell usage):

```
- name: Add quotes
  debug:
    msg: "{{ item | quote }}"
  loop: "{{ list1 }}"
```

```
TASK [Add quotes for shell usage]
*****
ok: [hivemaster] => (item=77) => {
  "msg": "77"
}
ok: [hivemaster] => (item=50) => {
  "msg": "50"
}
ok: [hivemaster] => (item=100) => {
  "msg": "100"
}
ok: [hivemaster] => (item=-400) => {
  "msg": "-400"
}
ok: [hivemaster] => (item=100) => {
  "msg": "100"
}
```

Concatenate a list into a string:

```
- name: Concatenate list into string
  debug:
    msg: "{{ list1 | join(' ') }}"
```

```
TASK [Concatenate list into string]
*****
ok: [hivemaster] => {
  "msg": "77 50 100 -400 100"
}
```

Get the file name (basename) from a path:

```
- name: Get basename of path
  debug:
    msg: "{{ '/etc/ansible/ansible.cfg' | basename }}"
```

```
TASK [Get basename of path]
*****
ok: [hivemaster] => {
  "msg": "ansible.cfg"
}
```

# Ansible Basic

An Ansible Training Course

PRESENTED BY:

Bittnet DevOps Team

ANSIBLE

We Make Custom Trainings



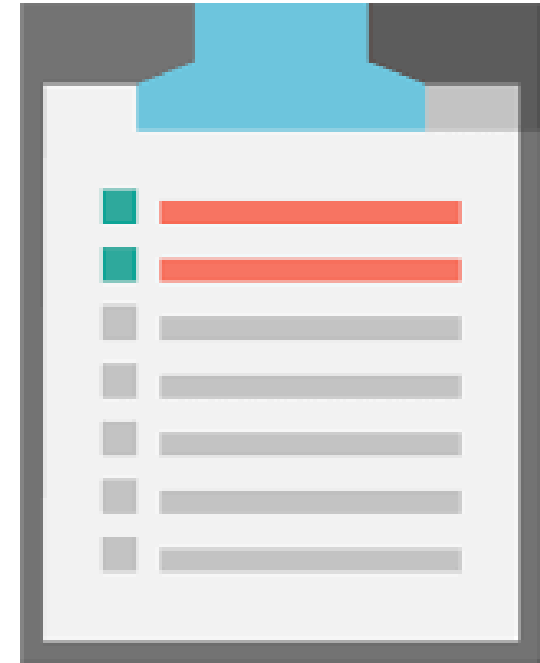
FASTER  
SMARTER  
SAFER

# 8. Advanced Topics



# Topics covered:

- Ansible Vault
- Roles: file structure, simple example



# 8.1. Ansible Vault





# Ansible Vault

- Some modules require sensitive data to be processed
- This may include private keys, passwords, and more
- To process sensitive data in a secure way, Ansible Vault can be used
- **Ansible Vault** is used to **encrypt** and **decrypt** files
- To manage this process, the **ansible-vault** command is used

# Creating an Encrypted File

- To create an encrypted file use:
  - `ansible-vault create playbook.yml`
- This command will prompt for a new vault password, and opens the file in **vi** for further editing
- As an alternative for entering password on the prompt, a vault password file may be used, but you'll have to make sure this file is protected in another way:
  - `ansible-vault create -vault-password-file=vault-pass playbook.yml`

# Creating an Encrypted File - Example

```
student:~$ ansible-vault create secret.yml
New Vault password: 123
Confirm New Vault password: 123
Encryption successful
```

```
student:~$ cat secret.yml
$ANSIBLE_VAULT;1.1;AES256
39656563336538323136363032613366323263613237613333633735623832326631313834643138
3036353434303664316132663439626262336330626166650a626664653636346539623339653631
313633333396435646631623563626132383264343165326635343935633764373735613162613034
6166333861383763370a326561366432323236396131666336373637343136616233313661303561
62643639356139353665346433333663396539363461393862313365333561363663313231376633
3931303634386262643639376233343130363438353334383162
```

# Creating an Encrypted File

- To view a vault encrypted file:
  - `ansible-vault view playbook.yml`
- To edit:
  - `ansible-vault edit playbook.yml`
- Use `ansible-vault encrypt playbook.yml` to encrypt an existing file, and use `ansible-vault decrypt playbook.yml` to decrypt it
- To change a password on an existing file, use `ansible-vault rekey`

# Using Playbooks with Vault

- To run a playbook that accesses Vault encrypted files, you need to use **--vault-id @prompt** option to be prompted for a password
- Alternatively, you can store the password as a single-line string in a password file, and access that using the **--vault-password-file=vault-file** option

# Include Multiple Vaults

- Until Ansible 2.4 we could include more vault files only if they had the same password.
- Starting with that version, a new option called **`vault-id`** was introduced.
  - This provides the option to include multiple vault files with different passwords.

# Include Multiple Vaults - Example

```
student:~$ vi testvault_v2.yml
---
- name: Ansible Vault Playbook
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: Include var from vault file
      include_vars: "/home/student/secret.yml"

    - name: Include var from another vault file
      include_vars: "/home/student/anothersecret.yml"

    - name: Print var from vault1
      debug:
        msg: "{{ secret_var }}"

    - name: Print var from vault2
      debug:
        msg: "{{ another_secret_var }}"
```



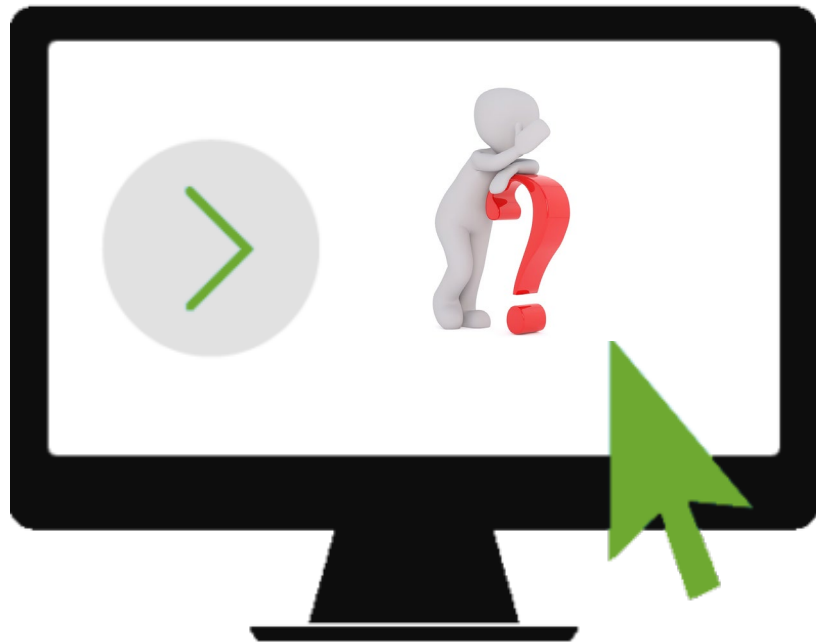
```
student:~$ ansible-playbook testvault_v2.yml --vault-id label1@prompt --vault-id
label2@prompt
```

# Managing Vault Files

- When setting up projects with Vault encrypted files, it makes sense to use separate files to store encrypted and non-encrypted variables
- To store host or host-group related variable files, you can use the following structure:

```
| -group_vars
|   |--dbservers
|       |-- vars
|       |-- vault
```





# Lab 8: Ansible Vault



## 8.2. Roles



# Organizing Ansible Contents

- When working with Ansible, it is recommended to use project directories so that contents can be organized in a consistent way
- Each project directory may have its own `ansible.cfg`, inventory as well as playbooks
- If the directory grows bigger, variable files and other include files may be used
- And finally, roles can be used to standardize and easily re-use specific parts of Ansible
- For now, consider a role a complete project dedicated to a specific task that is going to be included in the main playbook

# Directory Layout – Best Practices

```
production          # inventory file for production servers
staging             # inventory file for staging environment

group_vars/
  group1.yml         # here we assign variables to particular groups
  group2.yml
host_vars/
  hostname1.yml      # here we assign variables to particular systems
  hostname2.yml

library/            # if any custom modules, put them here (optional)
module_utils/       # if any custom module_utils to support modules, put them here
(optional)
filter_plugins/     # if any custom filter plugins, put them here (optional)

site.yml            # master playbook
webservers.yml      # playbook for webserver tier
dbservers.yml       # playbook for dbserver tier
```

# What are Roles?

- Ansible Playbooks can be very similar: code used in one playbook can be useful in other playbooks also.
- To make it easy to reuse code, roles can be used.
- A role is a collection of tasks, variables, files, templates and other resources in a fixed directory structure that, as such, can easily be included from a playbook.

# What are Roles?

- Roles should be written in a generic way, such that play specifics can be defined as variables in the play, and overwrite the default variables that should be set in the role
- Using Roles makes working with large project more manageable

```
roles/  
  common/  
    tasks/  
    handlers/  
    files/  
    templates/  
    vars/  
    defaults/  
    meta/
```

# Roles Default Structure

- **defaults** contains default values of role variables. If variables are set at the play level as well, these default values are overwritten
- **files** may contain static files that are needed from the role tasks
- **handlers** has a `main.yml` that defines handlers used in the role
- **meta** has a `main.yml` that may be used to include role metadata, such as information about author, license, dependencies and more



# Roles Default Structure

- **tasks** contains a `main.yml` that defines the role task definitions
- **templates** is used to store Jinja2 templates
- **tests** may contain an optional inventory file, as well as a `test.yml` playbook that can be used to test the role
- **vars** may contain a `main.yml` with standard variables for the role (which are not meant to be overwritten by playbook variables)

# Role Variables

- Variables can be defined at different levels in a role.
- **vars/main.yml** has the role default variables, which are used in default role functioning. They are not intended to be overwritten.
- **defaults/main.yml** can contain default variables. These have a low precedence, and can be overwritten by variables with the same name that are set in the playbook and which have higher precedence.

# Role Variables

- Playbook variables will always overwrite the variables as set in the role. Site-specific variables such as secrets and vault encrypted data should always be managed from the playbook, as role variables are intended to be generic
- Role variables are defined in the playbook when calling the role and they have the highest precedence and overwrite playbook variables and well as inventory variables

# Role Location

- Roles can be obtained in many ways
  - You can write your own roles
  - For Red Hat Enterprise Linux, the rhel-system-roles package is available
  - The community provides roles through the Ansible Galaxy website
- Roles can be stored at a default location, and from there can easily be used from playbooks
  - **./roles** has highest precedence
  - **~/.ansible/roles** is checked after that
  - **/etc/ansible/roles** is checked next
  - **/usr/share/ansible/** roles is checked last

# Roles in a Playbook

- Roles are referred to from playbooks

- Old syntax:

```
- name: role demo
  hosts: all
  roles:
    - role1
    - role2
```

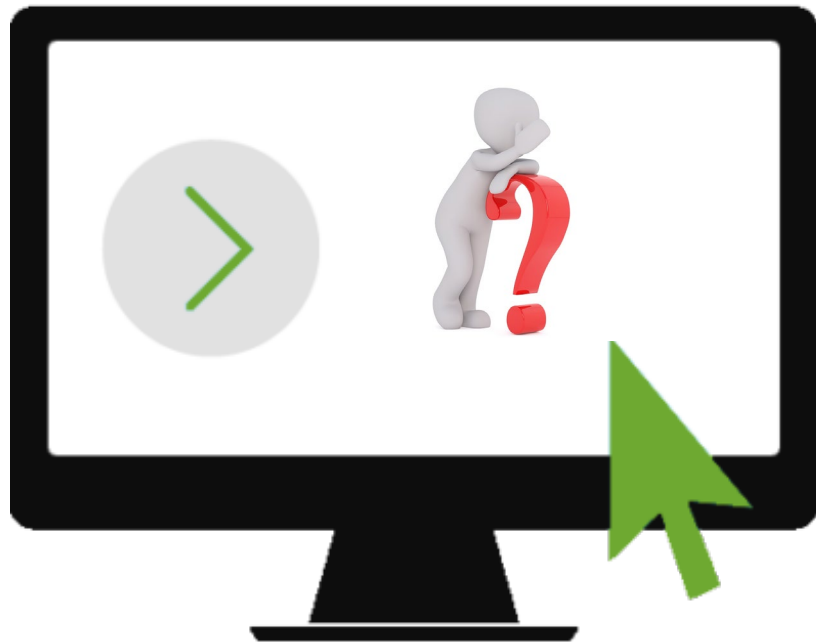
- Newer (recommended) syntax:

```
- hosts: webservers
  tasks:
    - debug:
        msg: "before we run our role"
    - import_role:
        name: example
    - include_role:
        name: example
    - debug:
        msg: "after we ran our role"
```

# Role Variables

- When calling a role, role variables can be defined

```
---  
- name: role demo  
  hosts: all  
  roles:  
    - role1  
    - role2  
    var1: cake  
    var2: cow
```



## Lab 9: Roles







Solutions for training the world.



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Ansible Vault

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: Create a vault file	2
Task 1.1: Encrypt	2
Task 1.2: Create	2
Task 2: View a vault file	3
Task 3: Edit a vault file	3
Task 4: Include vault file in Playbook	3
Task 5: Include multiple Vaults	4
Task 5: Include several files simultaneously	5
<i>Challenge:</i>	6
Part 1:	6
Part 2:	6
<i>Solution:</i>	7
Part 1:	7
Part 2:	7

# Lab Overview and objectives

The purpose of this lab is to learn how to use Ansible Vault, which is a tool that allows us to keep sensitive data like passwords or keys in encrypted files. To access this tool we can use `ansible-vault` command and specify `--ask-vault-pass`, `--vault-password-file` or `--vault-id` while running the playbook (to provide the vault password).

## Guided Tasks

### Task 1: Create a vault file

We can create a vault file `encrypting` an existing file or `creating` a new one from scratch. We will use both options in the next tasks.

#### Task 1.1: Encrypt

Let's create a simple YAML file:

```
student@ansible-00-01-hivemaster:~$ vi secret.yml
---
secret_var: "ThisIsABigSecret"
```

Now let's encrypt this file using `ansible-vault encrypt` command:

```
student@ansible-00-01-hivemaster:~$ ansible-vault encrypt secret.yml
New Vault password: 123
Confirm New Vault password: 123
Encryption successful
```

If we cat the content of `secret.yml` right now, the content should be encrypted:

```
student@ansible-00-01-hivemaster:~$ cat secret.yml
$ANSIBLE_VAULT;1.1;AES256
39656563336538323136363032613366323263613237613333633735623832326631313834643138
3036353434303664316132663439626262336330626166650a626664653636346539623339653631
31363333396435646631623563626132383264343165326635343935633764373735613162613034
6166333861383763370a326561366432323236396131666336373637343136616233313661303561
62643639356139353665346433333663396539363461393862313365333561363663313231376633
3931303634386262643639376233343130363438353334383162
```

Notice that there is also `ansible-vault decrypt` command, which performs the opposite operation.

#### Task 1.2: Create

We can also use `ansible-vault create` command to create a vault file (make sure to choose another password for this vault):

```
student@ansible-00-01-hivemaster:~$ ansible-vault create
anothersecret.yml
New Vault password: 789
Confirm New Vault password: 789
---
another_secret_var: "ThisIsAnotherBigSecret"
```

#### Task 2: View a vault file

We can use `ansible-vault view` command to see the unencrypted content of a vault file:

```
student@ansible-00-01-hivemaster:~$ ansible-vault view secret.yml
Vault password:
---
secret_var: "ThisIsABigSecret"
```

#### Task 3: Edit a vault file

We can use `ansible-vault edit` command to edit the content of a vault file:

```
student@ansible-00-01-hivemaster:~$ ansible-vault edit secret.yml
Vault password:
```

You can just exit from editor, if you don't want to change the variable.

#### Task 4: Include vault file in Playbook

Let's create a playbook and include a variable from a vault file:

```
student@ansible-00-01-hivemaster:~$ vi testvault.yml
---
- name: Ansible Vault Playbook
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: Include var from vault file
      include_vars: "/home/student/secret.yml"

    - name: Print var from vault
      debug:
        msg: "{{ secret_var }}"
```

Run the playbook using `--ask-vault-pass` option:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook testvault.yml --ask-vault-pass
Vault password:

PLAY [Ansible Vault Playbook]
*****
TASK [Include var from vault file]
*****
ok: [hivemaster]

TASK [Print var from vault]
*****
ok: [hivemaster] => {
  "msg": "ThisIsABigSecret"
}
[...]
```

Otherwise, we can write the password of vault in a file and pass that file as argument `--vault-password-file`:

```
student@ansible-00-01-hivemaster:~$ echo '123' > password
student@ansible-00-01-hivemaster:~$ ansible-playbook testvault.yml --vault-password-file=password
```

### Task 5: Include multiple Vaults

Until Ansible v2.4 we could include more vault files only if they had the same password. Starting with that version, a new option called `vault-id` was introduced, and this provides the option to include multiple vault files with different passwords:

```
student@ansible-00-01-hivemaster:~$ vi testvault_v2.yml
---
- name: Ansible Vault Playbook
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: Include var from vault file
      include_vars: "/home/student/secret.yml"

    - name: Include var from another vault file
      include_vars: "/home/student/anothersecret.yml"

    - name: Print var from vault1
      debug:
        msg: "{{ secret_var }}"

    - name: Print var from vault2
```

```
debug:
  msg: "{{ another_secret_var }}"
```

```
student@ansible-00-01-hivemaster:~$ ansible-playbook testvault_v2.yml --
vault-id @prompt --vault-id @prompt
```

Notice that we used @prompt for vault-id, so Ansible will ask us to provide the passwords for the 2 vaults (the order doesn't matter because Ansible tries every possible combination).

#### Task 5: Include several files simultaneously

As you have already seen, we used 2 tasks to include 2 vault files. In case that we have several files (not only vaults files) we can put them inside a directory and include the entire directory at once:

```
student@ansible-00-01-hivemaster:~$ mkdir my_var_files
student@ansible-00-01-hivemaster:~$ cp *secret.yml my_var_files/
student@ansible-00-01-hivemaster:~$ ls my_var_files/
anothersecret.yml  secret.yml
```

Now let's adjust the previous playbook:

```
---
- name: Ansible Vault Playbook
  hosts: hivemaster
  gather_facts: no
  tasks:
    - name: Include directory
      include_vars:
        dir: "/home/student/my_var_files/"

    - name: Print var from vault1
      debug:
        msg: "{{ secret_var }}"

    - name: Print var from vault2
      debug:
        msg: "{{ another_secret_var }}"
```

## Challenge:

### Part 1:

Create a new vault called `mybanner.yml` (you can create it directly using `ansible-vault create`):

```
---  
my_ssh_banner: "THIS IS A RESTRICTED AREA AND ANY UNAUTH ACCESS WILL BE  
PROSECUTED!"
```

### Part 2:

Create a new playbook, called `modifybanner.yml` in which we are going to change the banner for SSH connections (only for `hivemaster`), which is located in `(/etc/ssh/banner)` with the variable `my_ssh_banner` imported from the vault. We also have to enable `ssh` banner in `/etc/sshd/sshd_config`.

Notice that you may not want to use a secret text (from a vault) as you banner in real life experience, but this task is just for educational purpose.



## Solution:

### Part 1:

```
student@ansible-00-01-hivemaster:~$ ansible-vault create mysshbanner.yml
New Vault password:
Confirm New Vault password:

---
my_ssh_banner: "THIS IS A RESTRICTED AREA AND ANY UNAUTH ACCESS WILL BE
PROSECUTED!"
```

### Part 2:

```
student@ansible-00-01-hivemaster:~$ vi modifybanner.yml
---
- name: Modify SSH banner
  hosts: hivemaster
  become: true
  tasks:
    - name: Include var from vault
      include_vars: "/home/student/mysshbanner.yml"

    - name: Create banner file
      copy:
        content: "{{ my_ssh_banner }}"
        dest: /home/student/banner.txt
        owner: student

    - name: Insert a newline at the end of file
      lineinfile:
        path: /home/student/banner.txt
        line: ''

    - name: Copy file to proper location
      copy:
        src: banner.txt
        dest: /etc/ssh/banner

    - name: Modify sshd_config
      lineinfile:
        dest: /etc/ssh/sshd_config
        regexp: "^Banner"
        line: "Banner /etc/ssh/banner"
        state: present

    - name: Restart SSH service
```

```
service:
  name: ssh
  state: restarted
```

```
student@ansible-00-01-hivemaster:~$ ansible-playbook modifybanner.yml --
vault-id @prompt
Vault password (default):
```

```
PLAY [Modify SSH banner]
```

```
*****
```

```
TASK [Gathering Facts]
```

```
*****
```

```
ok: [hivemaster]
```

```
TASK [Include var from vault]
```

```
*****
```

```
ok: [hivemaster]
```

```
TASK [Create banner file]
```

```
*****
```

```
changed: [hivemaster]
```

```
TASK [Insert a newline at the end of file]
```

```
*****
```

```
changed: [hivemaster]
```

```
TASK [Copy file to proper location]
```

```
*****
```

```
ok: [hivemaster]
```

```
TASK [Modify sshd_config]
```

```
*****
```

```
changed: [hivemaster]
```

```
TASK [Restart SSH service]
```

```
*****
```

```
changed: [hivemaster]
```

```
PLAY RECAP
```

```
*****
```

```
hivemaster      : ok=7    changed=4    unreachable=0
failed=0        skipped=0  rescued=0    ignored=0
```



ANSIBLE

# ANSIBLE BASIC LAB MANUAL

Student Lab Kit v1.1

## ABSTRACT

This lab manual is designed for students who are interested in Ansible Basic Automation

**Confidential Document**

Ansible Roles

## Table of Contents

<b>Lab Overview and objectives</b>	<b>2</b>
<i>Guided Tasks</i>	2
Task 1: Role directory structure	2
Task 2: Create directory structure	2
Task 3: Create tasks – part 1	3
Task 4: Create handlers	3
Task 5: Create tasks – part 2	3
Task 6: Create handlers	4
Task 7: Create templates	4
Task 8: Invoke role in a playbook	4
Task 9: Test role	5

# Lab Overview and objectives

The purpose of this lab is to understand Ansible role usage and also to practice writing a role for managing our lab environment hosts.

Ansible roles are a feature that facilitates reuse of some files or playbooks, by grouping multiple tasks together into one container in order to do the automation in an effective manner with clean directory structures.

## Guided Tasks

### Task 1: Role directory structure

There is a predefined structure for an Ansible role, starting from its root directory which is also considered the role name. Inside, it has some directories which contain “.yaml” files. The following directories may be found inside a role:

- `tasks` - contains the main list of tasks to be executed by the role;
- `handlers` - contains handlers, which may be used by this role or even anywhere outside this role;
- `defaults` - default variables for the role;
- `vars` - other variables for the role;
- `files` - contains files which can be deployed via this role;
- `templates` - contains templates which can be deployed via this role;
- `meta` - defines some meta data for this role. See below for more details.

Ansible provides an option to start creating your own role using `ansible-galaxy init` command, which will create the entire hierarchy of directories and “.yaml” files inside your role directory.

In practice, you don't use all of those directories and it is recommended to create your directory structure by hand, with only the necessary files. Notice that a role should contain at least one directory from the previous list!

### Task 2: Create directory structure

We have right now in our environment 2 webserver (ubuntu and centos) with apache installed and custom index pages. Let's suppose that we want to set our `hivemaster` host as a reverse proxy for our `webserver`s, which should balance between the requests between the 2 backend servers, so let's create a role for installing NGINX and configuring load-balancing. Let's start with creating necessary directories:

```
student@ansible-00-01-hivemaster:~/test$ mkdir -p nginx_role/defaults
student@ansible-00-01-hivemaster:~/test$ mkdir -p nginx_role/handlers
student@ansible-00-01-hivemaster:~/test$ mkdir -p nginx_role/tasks
student@ansible-00-01-hivemaster:~/test$ mkdir -p nginx_role/templates
```

### Task 3: Create tasks – part 1

Now, that we have the directories let's move on to the .yaml files. Let's create tasks/main.yaml file:

```
student@ansible-00-01-hivemaster:~$ vi nginx_role/tasks/main.yaml
---
- name: install nginx
  package:
    name: nginx
    state: latest
  when: ansible_os_family == 'Debian'
  notify: restart nginx

- name: Configure NGINX
  import_tasks: configure.yaml
```

The first task is to install `nginx` (we customized the installation just for “Debian” distribution).

### Task 4: Create handlers

As you can see, we also called a handler (“`restart nginx`”), so let's write it in the appropriate location:

```
student@ansible-00-01-hivemaster:~$ vi nginx_role/handlers/main.yaml
---
- name: restart nginx
  service:
    name: nginx
    state: restarted
    enabled: yes

- name: reload nginx
  service:
    name: nginx
    state: reloaded
```

We added a “reload nginx” handler, which performs a reload of configuration without stopping the NGINX service (reload != restart).

### Task 5: Create tasks – part 2

We are going to continue with adding tasks necessary for NGINX configuration. Because we want to implement also the concept of reusable playbooks, we imported the configuration file (`configure.yaml`) into `main.yaml` file:

```
student@ansible-00-01-hivemaster:~$ vi nginx_role/tasks/configure.yaml
---
```

```
- name: copy config file
  template:
    src: load_balancer.conf.j2
    dest: "{{ nginx_conf_path }}/load_balancer.conf"
  notify: restart nginx

- name: Remove default home page served by nginx
  file:
    path: "{{ nginx_path }}/sites-enabled/default"
    state: absent

- name: test nginx configuration
  command: nginx -t
  changed_when: false
  notify: reload nginx
```

### Task 6: Create handlers

We used the variables `nginx_conf_path` and `nginx_path` in this playbook which by default points to `/etc/nginx/conf.d` and `/etc/nginx`. We will define these variables in `defaults/main.yml`:

```
student@ansible-00-01-hivemaster:~$ vi nginx_role/defaults/main.yml
---
nginx_conf_path: /etc/nginx/conf.d
nginx_path: /etc/nginx
```

### Task 7: Create templates

We also have to create the template for our load balancer:

```
student@ansible-00-01-hivemaster:~$ vi
nginx_role/templates/load_balancer.conf.j2
upstream backend {
    server {{ backend_srv_01 }};
    server {{ backend_srv_02 }};
}
server {
    listen {{ listen_port }};
    location / {
        proxy_pass http://backend;
    }
}
```

### Task 8: Invoke role in a playbook

Now let's create a playbook (in `/home/student`) to test the usage of the role:

```
student@ansible-00-01-hivemaster:~$ cd
student@ansible-00-01-hivemaster:~$ vi roletest.yml
---
- name: Test NGINX role
  hosts: hivemaster
  become: true
  vars:
    backend_srv_01: 10.128.0.49
    backend_srv_02: 10.142.15.213
    listen_port: 8081

  tasks:
  pre_tasks:
    - debug:
        msg: 'Starting NGINX installation process.'

  roles:
    - nginx_role

  post_tasks:
    - debug:
        msg: 'Web server has been configured.'
```

Notice that we used `pre_tasks` and `post_tasks` for this role: these ensures that the tasks listed in `pre_tasks` section are executed before running the role and the tasks listed in `post_tasks` after the execution. We also defined the variables necessary for the role: IP addresses for our backend servers and also the listening IP address for NGINX server (`hivemaster`).

\*make sure to replace the IPs with your hosts ones

## Task 9: Test role

Run the playbook and explore playbook recap:

```
student@ansible-00-01-hivemaster:~$ ansible-playbook roletest.yml

PLAY [Test NGINX role]
*****

TASK [Gathering Facts]
*****
ok: [hivemaster]

TASK [debug]
*****
ok: [hivemaster] => {
  "msg": "Starting NGINX installation process."
}

TASK [nginx_role : install nginx]
*****
```



```

changed: [hivemaster]

TASK [nginx_role : copy config file]
*****
changed: [hivemaster]

TASK [nginx_role : Remove default home page served by nginx]
*****
changed: [hivemaster]

TASK [nginx_role : test nginx configuration]
*****
ok: [hivemaster]

RUNNING HANDLER [nginx_role : restart nginx]
*****
changed: [hivemaster]

TASK [debug]
*****
ok: [hivemaster] => {
    "msg": "Web server has been configured."
}

PLAY RECAP
*****
hivemaster          : ok=8    changed=4    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

```

Now let's test that our NGINX server is listening on specified port (8081). Running the curl command twice you should get 2 different responses (1 from each backend server) because the balancing algorithm is default (Round Robin):

```

student@ansible-00-01-hivemaster:~$ curl localhost:8081
<html>
<head>
<title>Welcome to Ansible</title>
</head>
<body>
<p>
Server details:
Hostname: ansible-00-02-ubuntu
OS: Ubuntu 18.04
</p>
</body>
</html>

student@ansible-00-01-hivemaster:~$ curl localhost:8081
<html>
<head>
<title>Welcome to Ansible</title>
</head>
<body>
<p>
Server details:

```

```
Hostname: ansible-00-03-centos
OS: CentOS 7.7
</p>
</body>
</html>
```

If your curl command doesn't work, please check that your NGINX is listening on port 8081 using `netstat` command:

```
student@ansible-00-01-hivemaster:~$ sudo netstat -napt | grep LISTEN
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
25491/mysqld
tcp        0      0 0.0.0.0:8081            0.0.0.0:*               LISTEN
16129/nginx: master
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
964/systemd-resolve
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
5240/sshd
```