**Ansible**

## **About Ansible**

Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

Ansible’s main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with other transports and pull modes as alternatives), and a language that is designed around auditability by humans–even those not familiar with the program.

**Installing Ansible:**



On CentOS:

$ sudo yum install epel-release

$ sudo yum install ansible

Please follow below

[**Installing Ansible — Ansible Documentation**](https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html)

**amazon-linux-extras install ansible2**

**Yum install ansible2**

Example script to create user:

**#!/bin/bash**

**# Script to add a user to Linux system**

**if [ $(id -u) -eq 0 ]; then**

**$username=johndoe**

**read -s -p "Enter password : " password**

**egrep "^$username" /etc/passwd >/dev/null**

**if [ $? -eq 0 ]; then**

**echo "$username exists!"**

**exit 1**

**else**

**useradd -m -p $password $username**

**[ $? -eq 0 ] && echo "User has been added**

**to system!" || echo "Failed to add a user!"**

**fi**

**fi**

Example ansible playbook to create user:

**- hosts: all\_my\_db\_servers**

**tasks:**

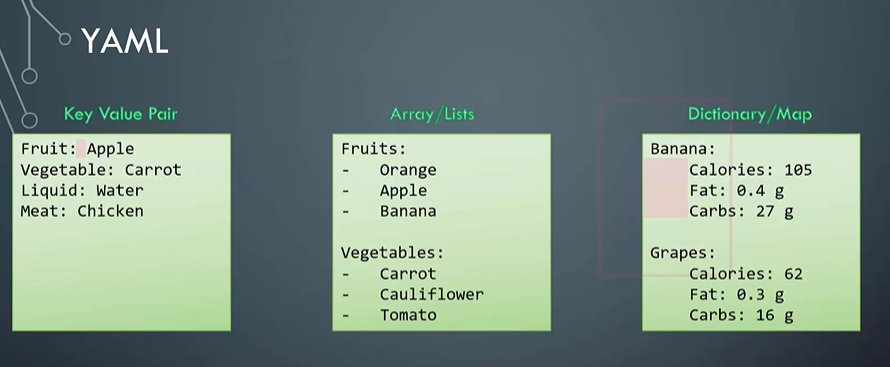
**- user:**

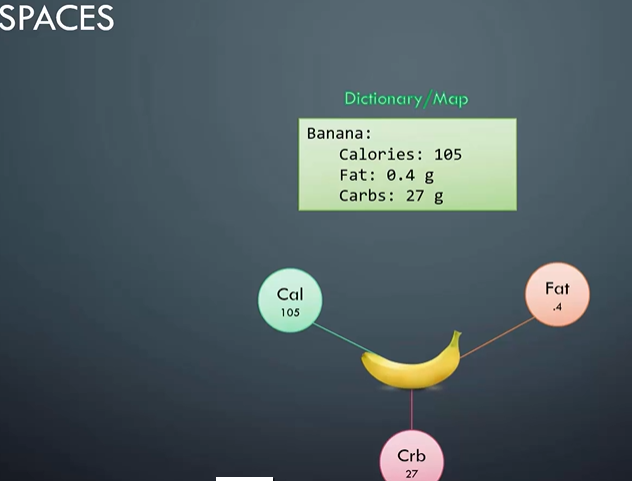
**name: johndoe**

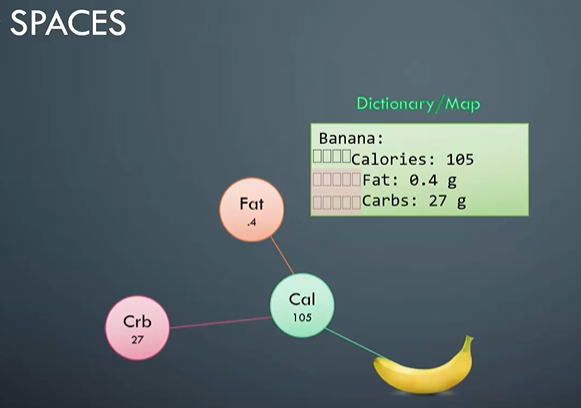
**state: present**

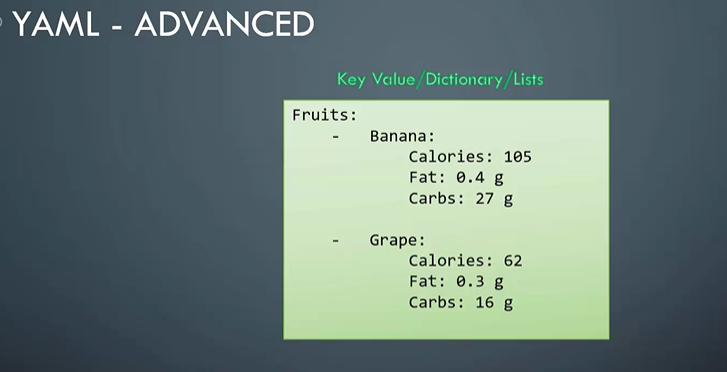
**Yaml:**



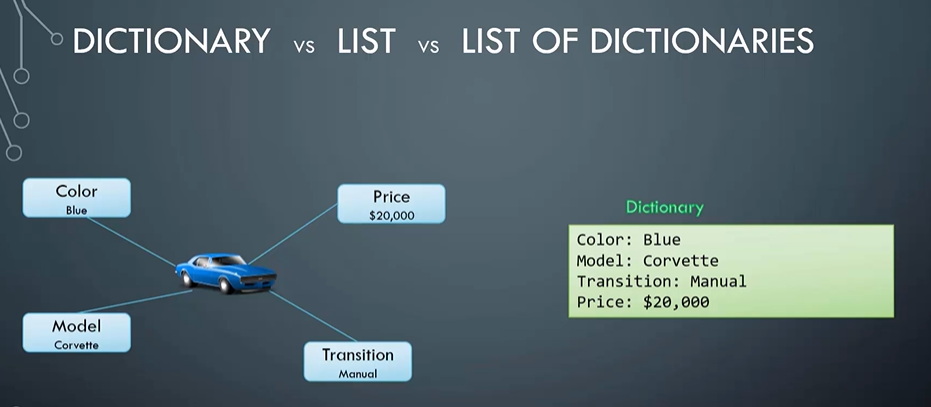


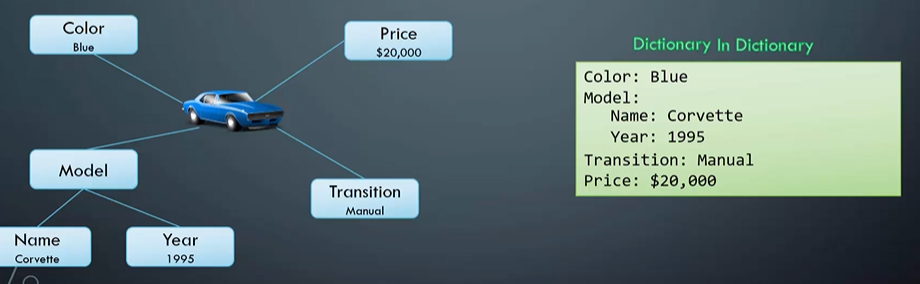






Dictionary vs List vs List of Dictionary:







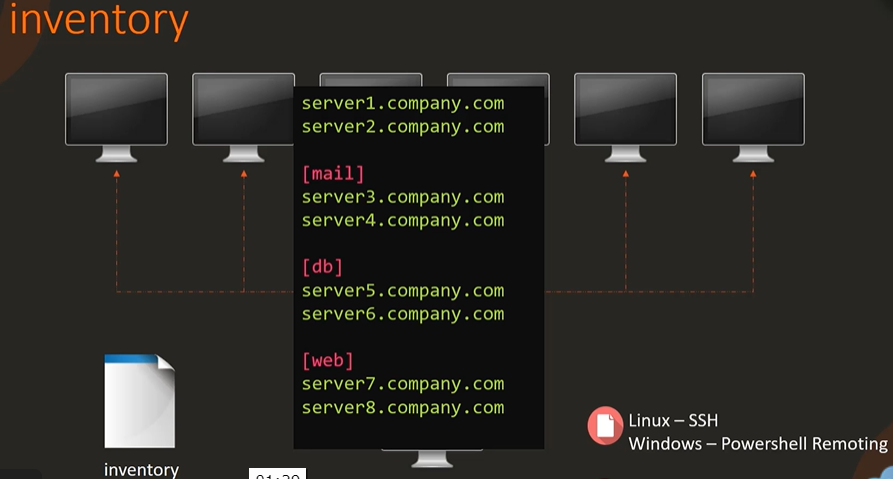


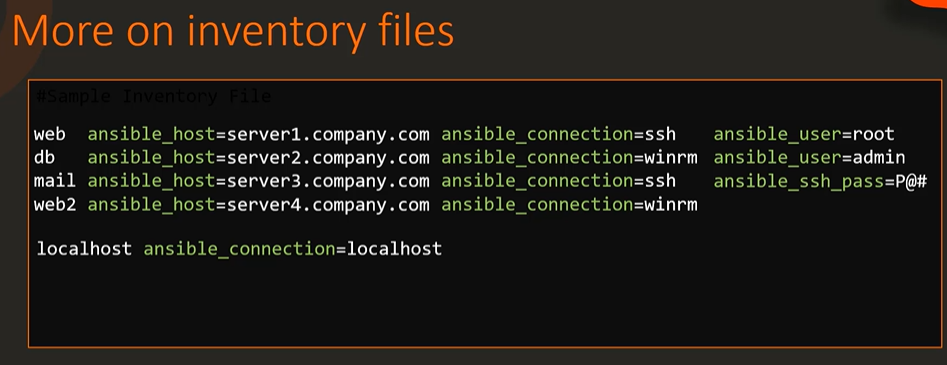


**Ansible Inventory:**

Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as inventory. Once your inventory is defined, you use [patterns](https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html#intro-patterns) to select the hosts or groups you want Ansible to run against.

The default location for inventory is a file called /etc/ansible/hosts. You can specify a different inventory file at the command line using the -i <path> option. You can also use multiple inventory files at the same time as described in [Using multiple inventory sources](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#using-multiple-inventory-sources), and/or pull inventory from dynamic or cloud sources or different formats (YAML, ini, and so on), as described in [Working with dynamic inventory](https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html#intro-dynamic-inventory). Introduced in version 2.4, Ansible has [Inventory plugins](https://docs.ansible.com/ansible/latest/plugins/inventory.html#inventory-plugins) to make this flexible and customizable.





Inventory Parameters:

- ansible\_connection-ssh/wnrm/localhost

- ansible\_port-22/5986

- ansible\_user-root/administrator

- ansible\_ssh\_pass-Password

# 

# **Intro to playbooks**

Ansible Playbooks offer a repeatable, re-usable, simple configuration management and multi-machine deployment system, one that is well suited to deploying complex applications. If you need to execute a task with Ansible more than once, write a playbook and put it under source control. Then you can use the playbook to push out new configuration or confirm the configuration of remote systems. The playbooks in the [ansible-examples repository](https://github.com/ansible/ansible-examples) illustrate many useful techniques. You may want to look at these in another tab as you read the documentation.

Playbooks can:

* declare configurations
* orchestrate steps of any manual ordered process, on multiple sets of machines, in a defined order
* launch tasks synchronously or [asynchronously](https://docs.ansible.com/ansible/latest/user_guide/playbooks_async.html#playbooks-async)

## [**Playbook syntax**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#id3)

Playbooks are expressed in YAML format with a minimum of syntax. If you are not familiar with YAML, look at our overview of [YAML Syntax](https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html#yaml-syntax) and consider installing an add-on for your text editor (see [Other Tools and Programs](https://docs.ansible.com/ansible/latest/community/other_tools_and_programs.html#other-tools-and-programs)) to help you write clean YAML syntax in your playbooks.

A playbook is composed of one or more ‘plays’ in an ordered list. The terms ‘playbook’ and ‘play’ are sports analogies. Each play executes part of the overall goal of the playbook, running one or more tasks. Each task calls an Ansible module

## [**Playbook execution**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#id4)

#ansible-playbook playbook.yml -i <inventory>

Sample playbook:

---

- name: Playbook

hosts: webservers

become: yes

become\_user: root

tasks:

- name: ensure apache is at the latest version

yum:

name: httpd

state: latest

- name: ensure apache is running

service:

name: httpd

state: started

**Ansible Modules:**

# **Introduction to modules**

Modules (also referred to as “task plugins” or “library plugins”) are discrete units of code that can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote target node, and collects return values.

You can execute modules from the command line:

ansible webservers -m service -a "name=httpd state=started"

ansible webservers -m ping

ansible webservers -m command -a "/sbin/reboot -t now" - <inventory>

Each module supports taking arguments. Nearly all modules take key=value arguments, space delimited. Some modules take no arguments, and the command/shell modules simply take the string of the command you want to run.

From playbooks, Ansible modules are executed in a very similar way:

- name: reboot the servers

action: command /sbin/reboot -t now

Which can be abbreviated to:

- name: reboot the servers

command: /sbin/reboot -t now

Another way to pass arguments to a module is using YAML syntax also called ‘complex args’

- name: restart webserver

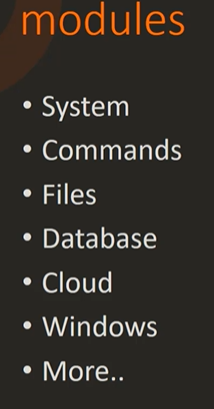
service:

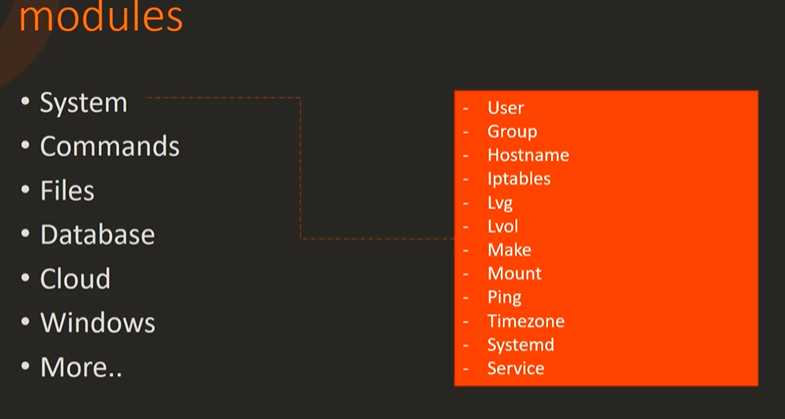
name: httpd

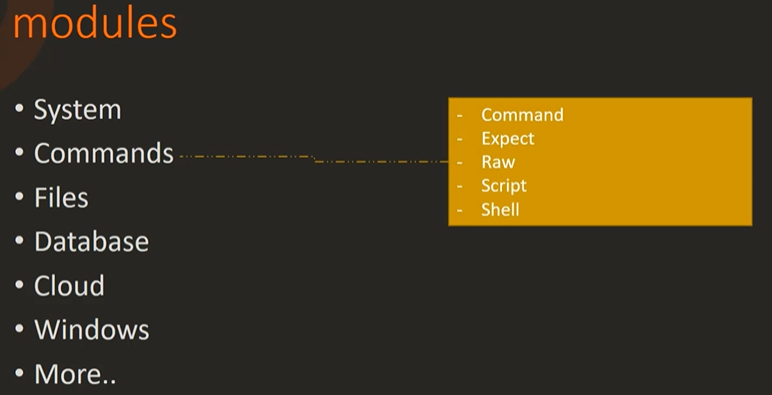
state: restarted

All modules return JSON format data. This means modules can be written in any programming language. Modules should be idempotent, and should avoid making any changes if they detect that the current state matches the desired final state. When used in an Ansible playbook, modules can trigger ‘change events’ in the form of notifying ‘handlers’ to run additional tasks.

Link to find all ansible modules: [**All modules — Ansible Documentation**](https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html)

****

****

****

- name: This command will change the working directory to somedir/.

shell:

cmd: ls -l | grep log

chdir: somedir/

- **name**: This command will change the working directory to somedir/ and will only run when /path/to/database doesn't exist.

**command**: /usr/bin/make\_database.sh arg1 arg2

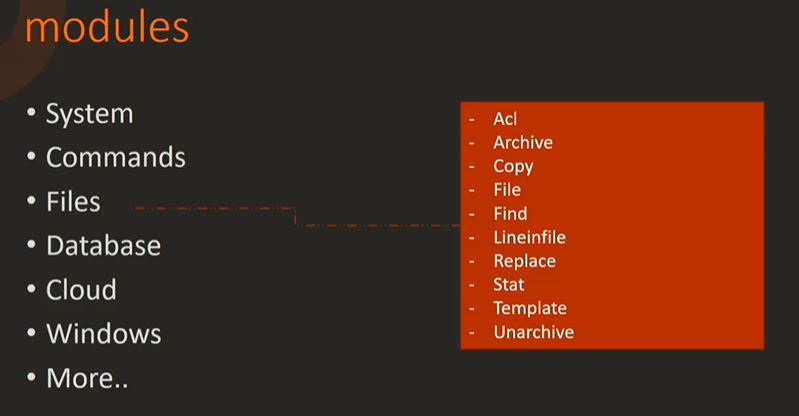
**args**:

**chdir**: somedir/

**creates**: /path/to/database

### Difference between two Ansible modules: Command and Shell module

* In Ansible, the command module is often used for many cases to be executed on the selected nodes or hosts. In the command module, the execution of command is not executed through the shell which results in not working of variables like $HOME, $HOSTNAME and the operations like ‘|’, ‘\*’, ‘<’, ‘>’, ‘&’ and ‘;’. But all these can be achieved and works by using the shell module. So, shell commands are used in such cases.
* A shell module is used for the execution of the shell commands against the target like UNIX-based hosts but it affects the user’s environment which is not a good thing in the real environment. So in this case, the command module is good for the security purpose and to run the required command more securely.
* In Ansible, the command module provides commands to execute on all the listed nodes or hosts. Such executed commands will not be processed through the shell. While the shell module is generally used for the execution of all the shell commands against the target UNIX-based hosts. Such shell modules execute commands on nodes or shell scripts.
* In Ansible, the shell module executes the shell command on the remote hosts. It uses /bin/sh to execute the command by default. Also, commands can be executed using the /bin/bash by passing the executable arguments but in the command module, the execution of command is not executed through the shell

****

- copy:

src: /srv/myfiles/foo.conf

dest: /etc/foo.conf

owner: foo

group: foo

mode: 0644

*# The same example as above, but using a symbolic mode equivalent to 0644*

- copy:

src: /srv/myfiles/foo.conf

dest: /etc/foo.conf

owner: foo

group: foo

mode: u=rw,g=r,o=r

*# Another symbolic mode example, adding some permissions and removing others*

- copy:

src: /srv/myfiles/foo.conf

dest: /etc/foo.conf

owner: foo

group: foo

mode: u+rw,g-wx,o-rwx

*#Add a line to a file if it does not exist, without passing regexp*

- lineinfile:

path: /tmp/testfile

line: '192.168.1.99 foo.lab.net foo'

*# Fully quoted because of the ': ' on the line. See the Gotchas in the YAML docs.*

- lineinfile:

path: /etc/sudoers

state: present

regexp: '^%wheel\s'

line: '%wheel ALL=(ALL) NOPASSWD: ALL'

*# Yaml requires escaping backslashes in double quotes but not in single quotes*

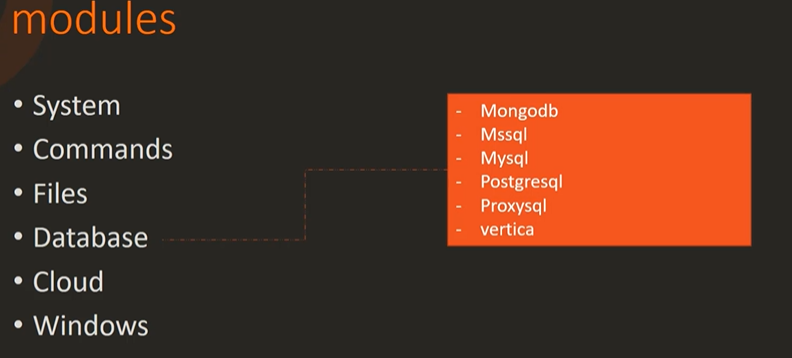
- lineinfile:

path: /opt/jboss-as/bin/standalone.conf

regexp: '^(.\*)Xms(\\d+)m(.\*)$'

line: '\1Xms${xms}m\3'

backrefs: yes



**-** **name:** Create a single index on a collection

**community.mongodb.mongodb\_index:**

**login\_user:** admin

**login\_password:** secret

**indexes:**

**-** **database:** mydb

**collection:** test

**keys:**

**-** **username:** 1

**last\_login:** -1

**options:**

**name:** myindex

**state:** present



**-** name**:** Create a data container

docker\_container**:**

name**:** mydata

image**:** busybox

volumes**:**

**-** /data

**-** name**:** Re-create a redis container

docker\_container**:**

name**:** myredis

image**:** redis

command**:** redis-server --appendonly yes

state**:** present

recreate**:** yes

exposed\_ports**:**

**-** 6379

volumes\_from**:**

**-** mydata

**-** name**:** Restart a container

docker\_container**:**

name**:** myapplication

image**:** someuser/appimage

state**:** started

restart**:** yes

links**:**

**-** "myredis:aliasedredis"

devices**:**

**-** "/dev/sda:/dev/xvda:rwm"

ports**:**

**-** "8080:9000"

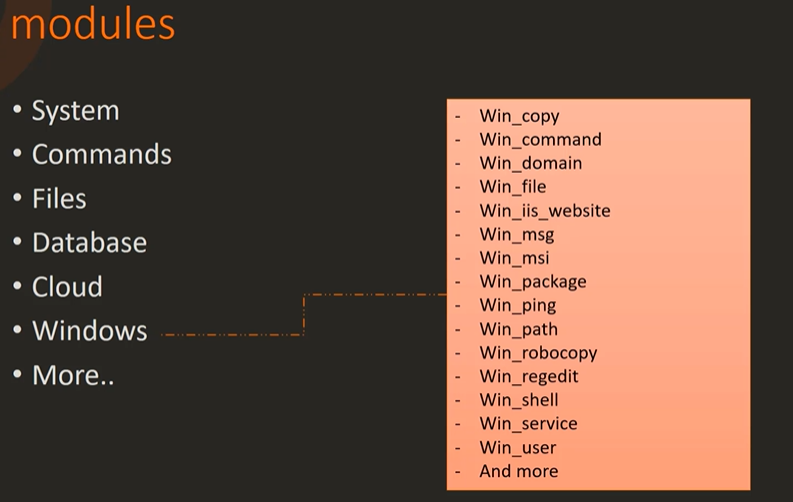
**-** "127.0.0.1:8081:9001/udp"

env**:**

SECRET\_KEY**:** "ssssh"

*# Values which might be parsed as numbers, booleans or other types by the YAML parser need to be quoted*

BOOLEAN\_KEY**:** "yes"



**-** name**:** Touch a file (creates if not present, updates modification time if present)

win\_file**:**

path**:** C:\Temp\foo.conf

state**:** touch

**-** name**:** Remove a file, if present

win\_file**:**

path**:** C:\Temp\foo.conf

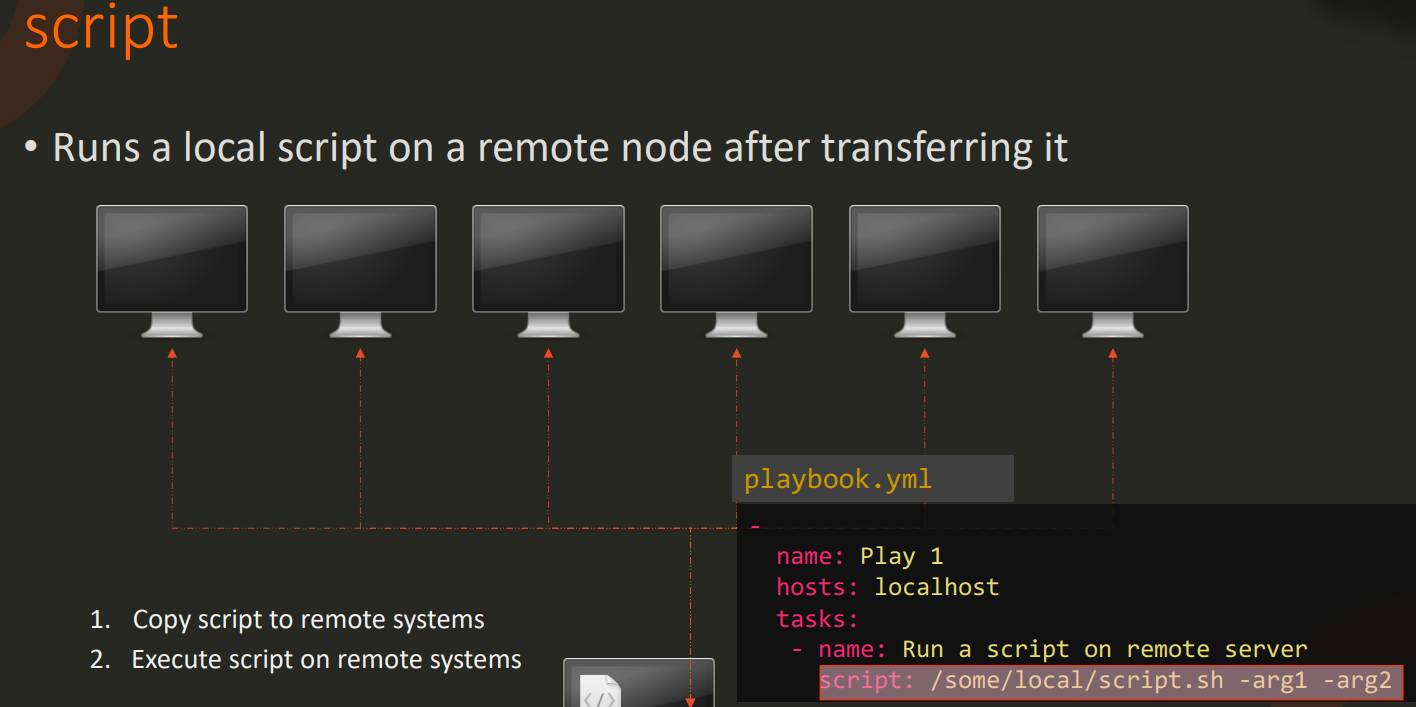
state**:** absent

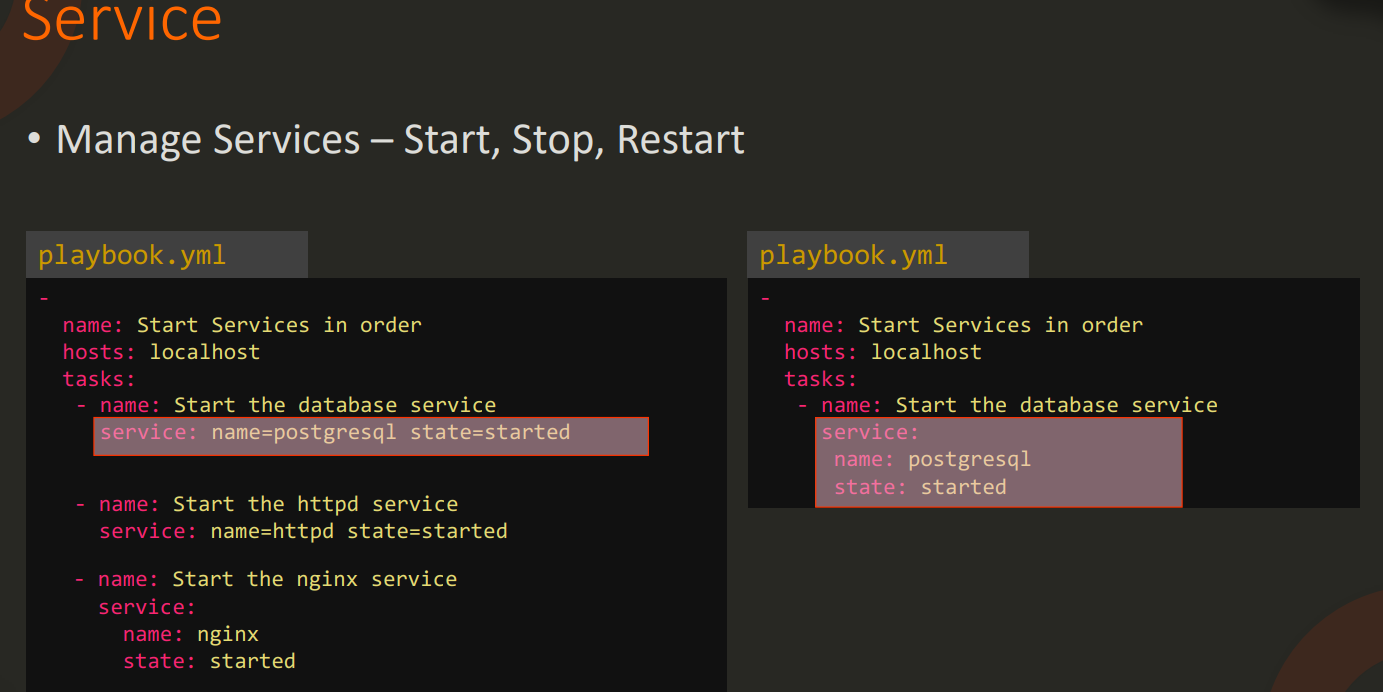
**-** name**:** Create directory structure

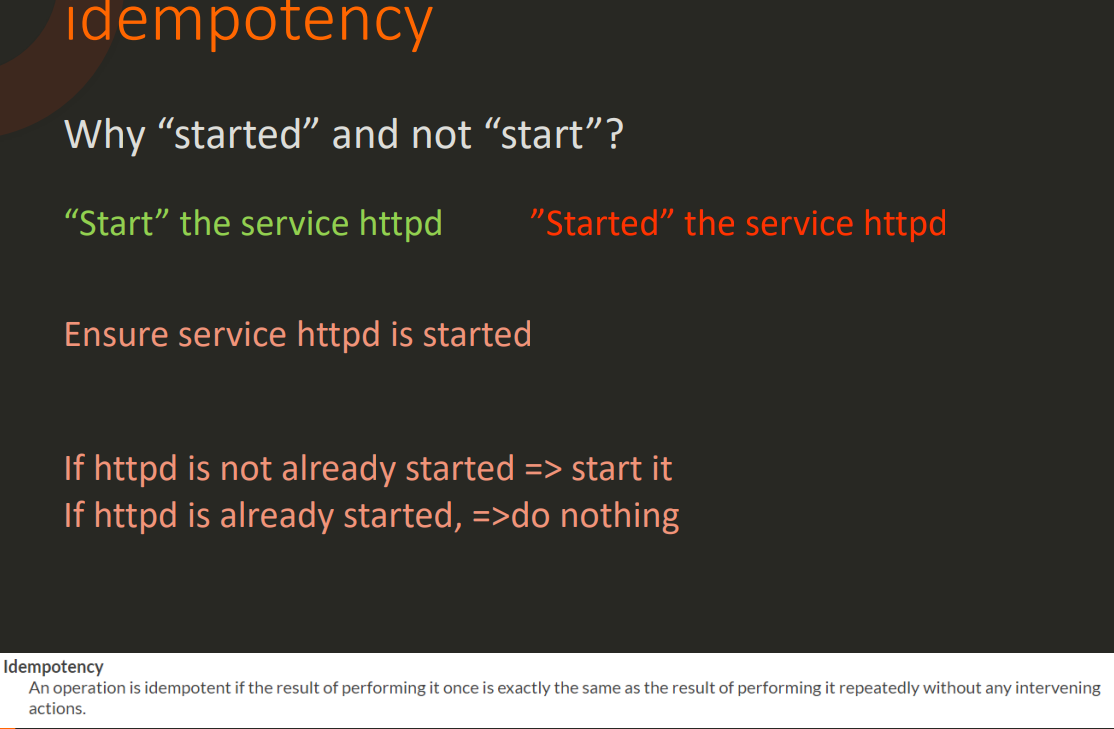
win\_file**:**

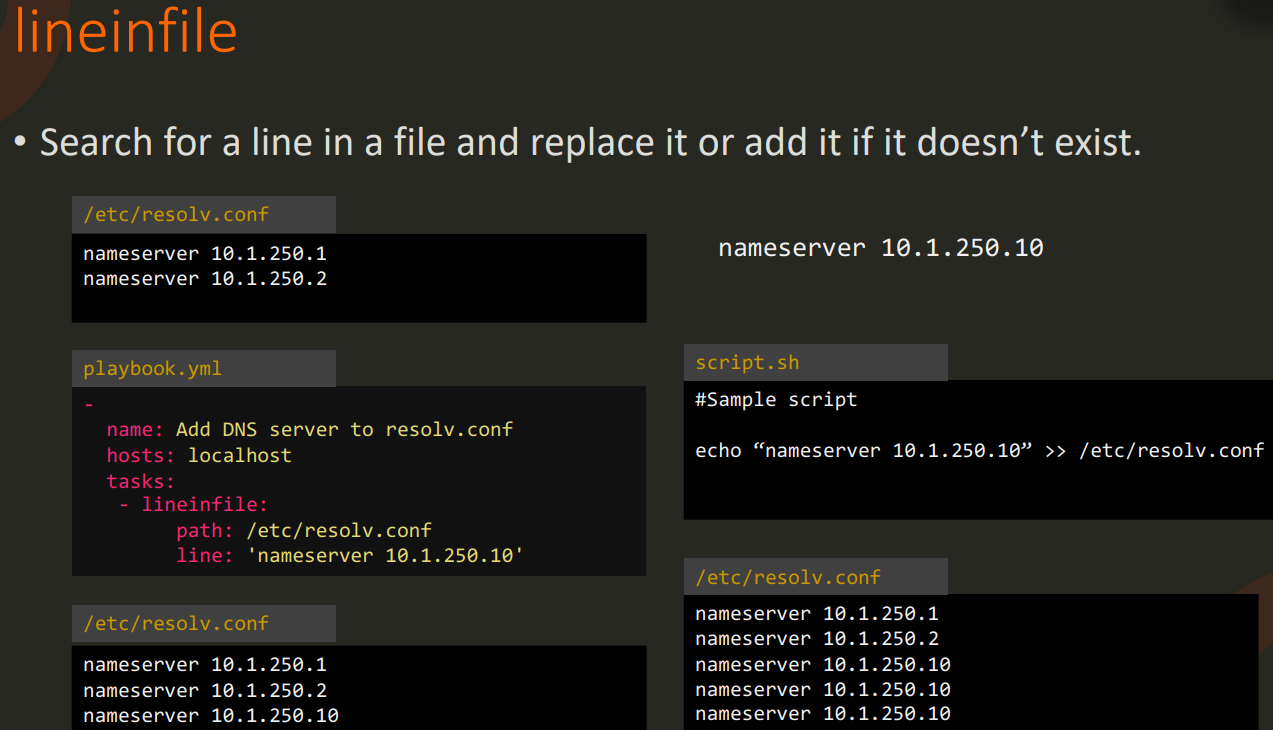
path**:** C:\Temp\folder\subfolder

state**:** directory



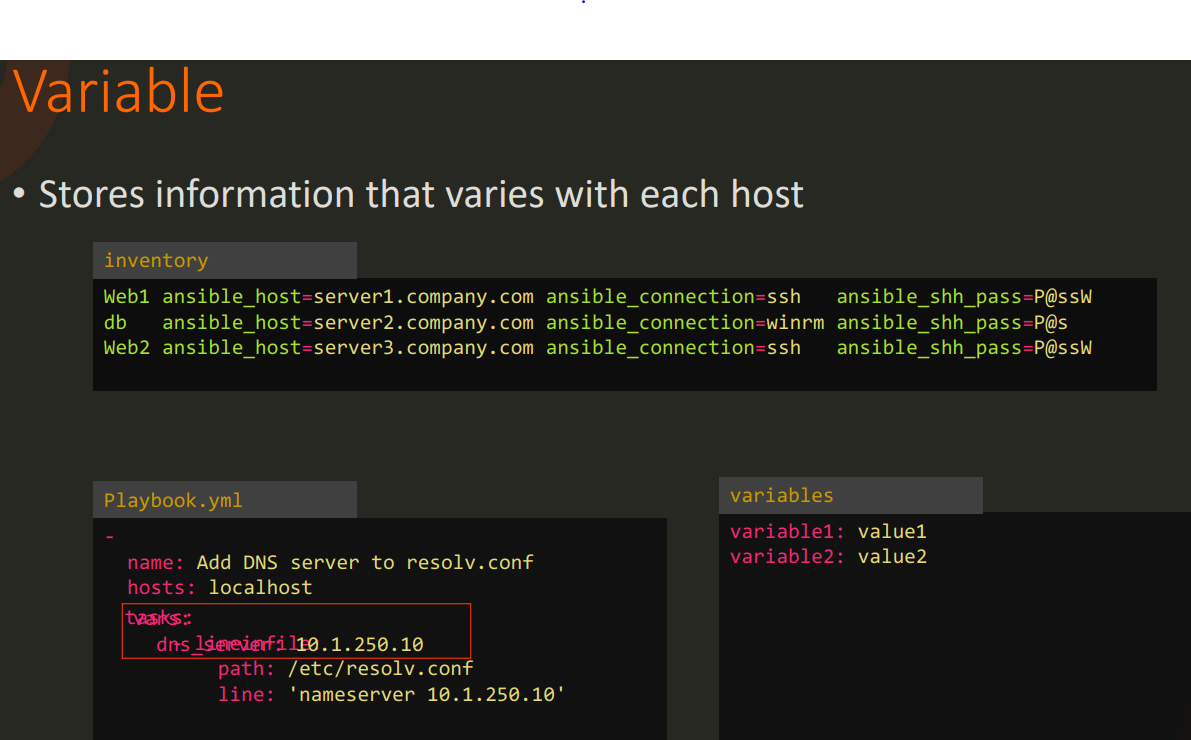


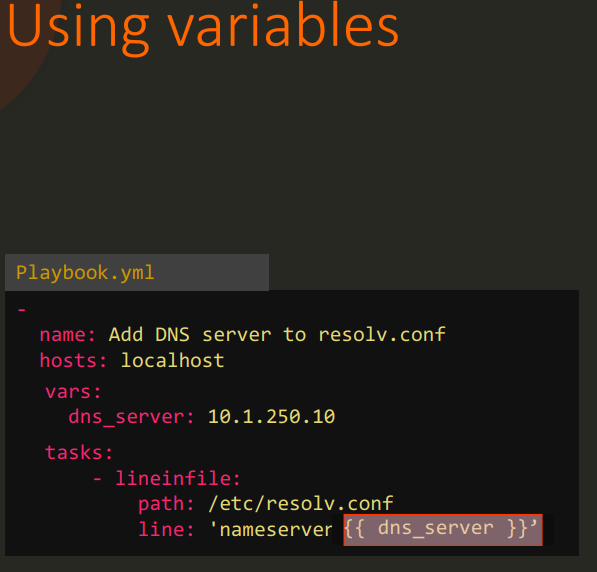




Variables:

Ansible uses variables to manage differences between systems. With Ansible, you can execute tasks and playbooks on multiple different systems with a single command. To represent the variations among those different systems, you can create variables with standard YAML syntax, including lists and dictionaries. You can define these variables in your playbooks, in your [inventory](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#intro-inventory), in re-usable [files](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse.html#playbooks-reuse) or [roles](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html#playbooks-reuse-roles), or at the command line. You can also create variables during a playbook run by registering the return value or values of a task as a new variable.

After you create variables, either by defining them in a file, passing them at the command line, or registering the return value or values of a task as a new variable, you can use those variables in module arguments, in [conditional “when” statements](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#playbooks-conditionals), in [templates](https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html#playbooks-templating), and in [loops](https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#playbooks-loops). The [ansible-examples github repository](https://github.com/ansible/ansible-examples) contains many examples of using variables in Ansible.****

****

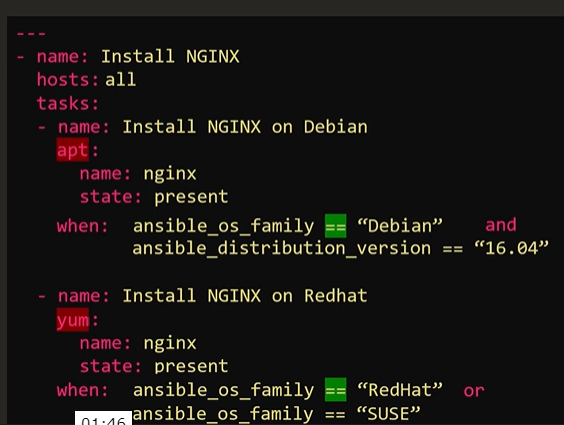
****

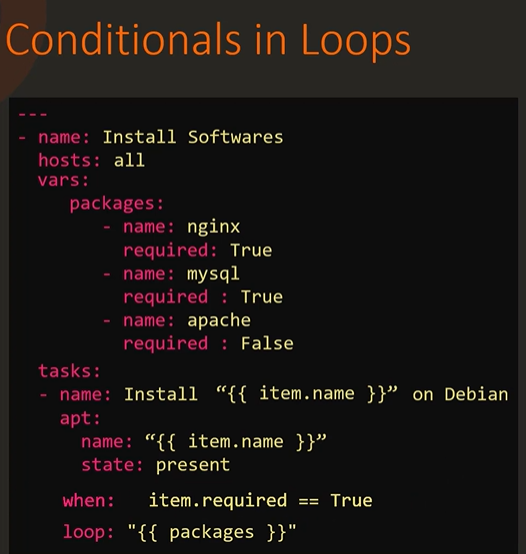
[**https://docs.ansible.com/ansible/latest/user\_guide/playbooks\_templating.html**](https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html)

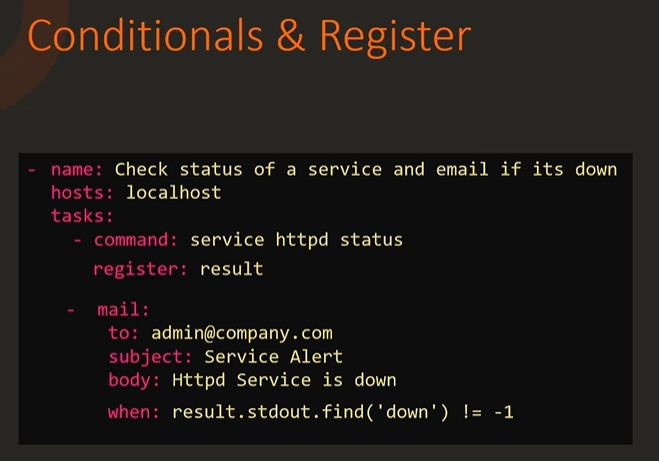
Ansible Conditional Statements

In a playbook, you may want to execute different tasks, or have different goals, depending on the value of a fact (data about the remote system), a variable, or the result of a previous task. You may want the value of some variables to depend on the value of other variables. Or you may want to create additional groups of hosts based on whether the hosts match other criteria. You can do all of these things with conditionals.

[Conditionals — Ansible Documentation](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html)





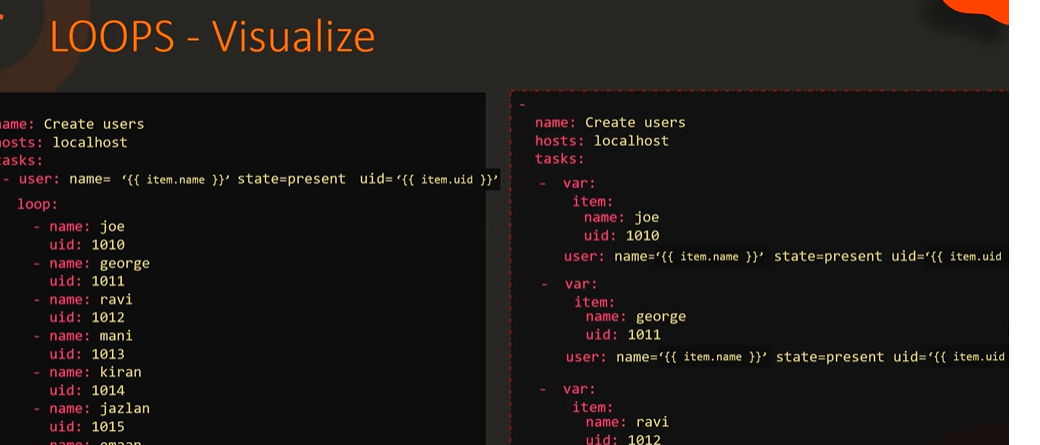


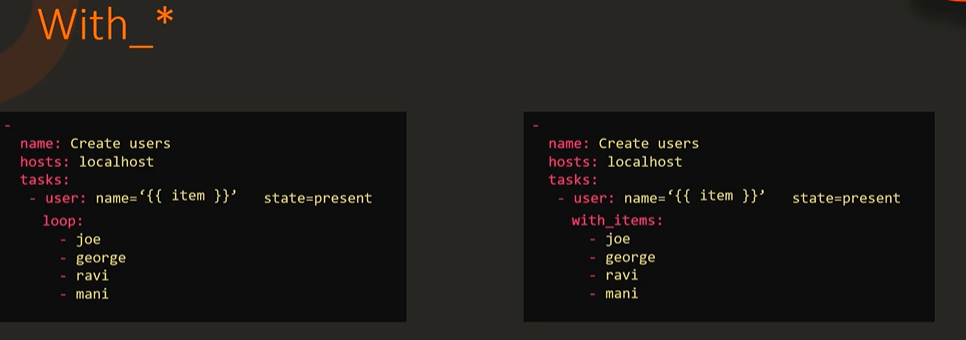
# **Loops**

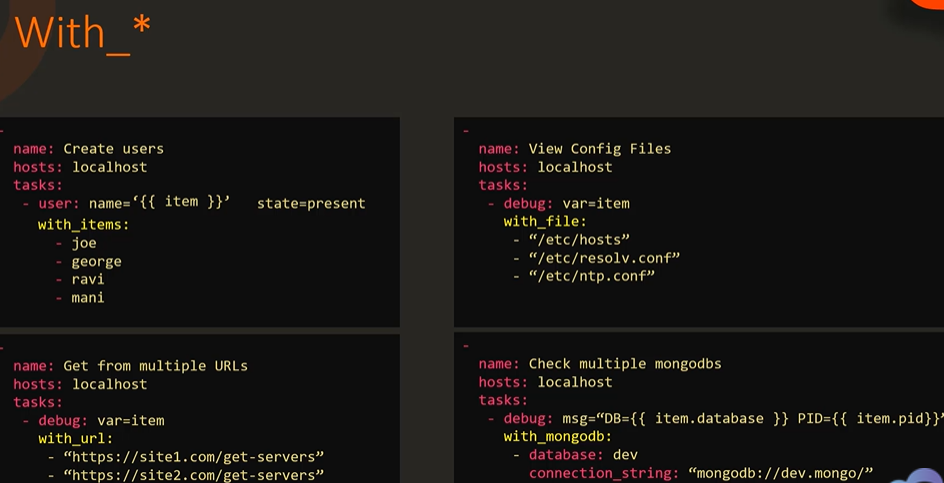
Ansible offers the loop, with\_<lookup>, and until keywords to execute a task multiple times. Examples of commonly-used loops include changing ownership on several files and/or directories with the [file module](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html#file-module), creating multiple users with the [user module](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/user_module.html#user-module), and repeating a polling step until a certain result is reached.

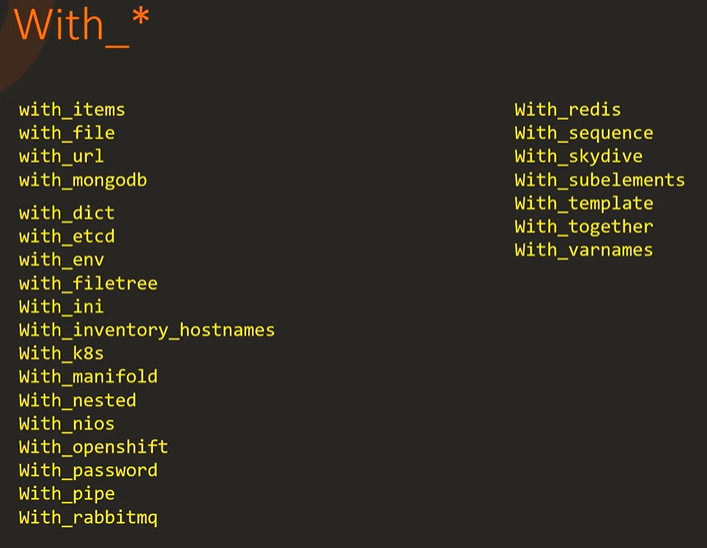
<https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html>







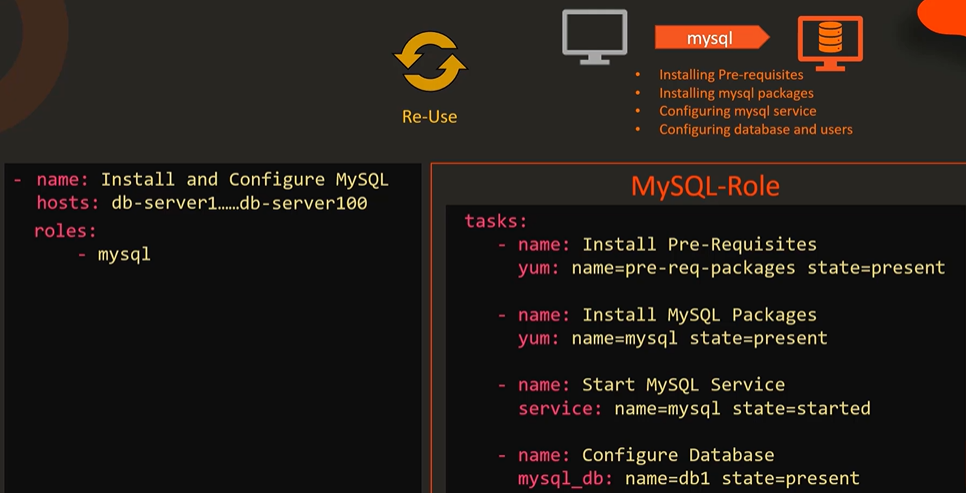


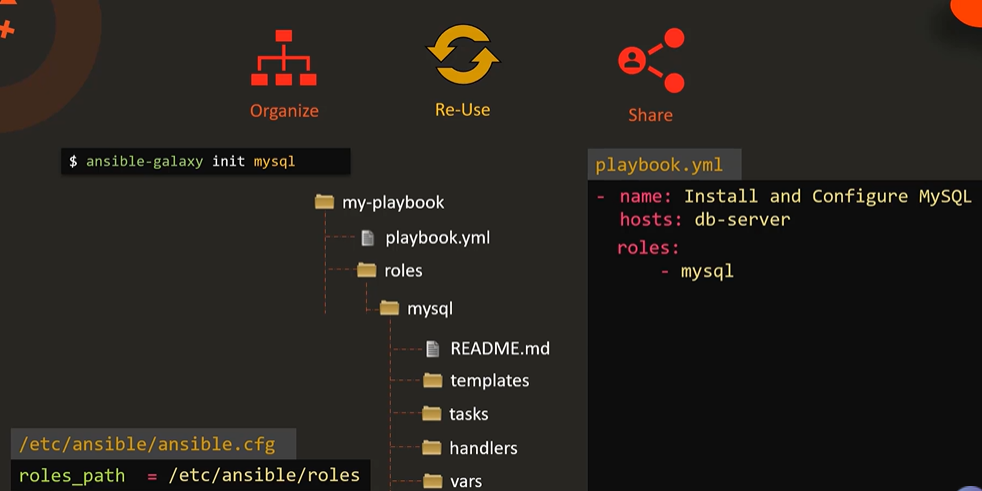


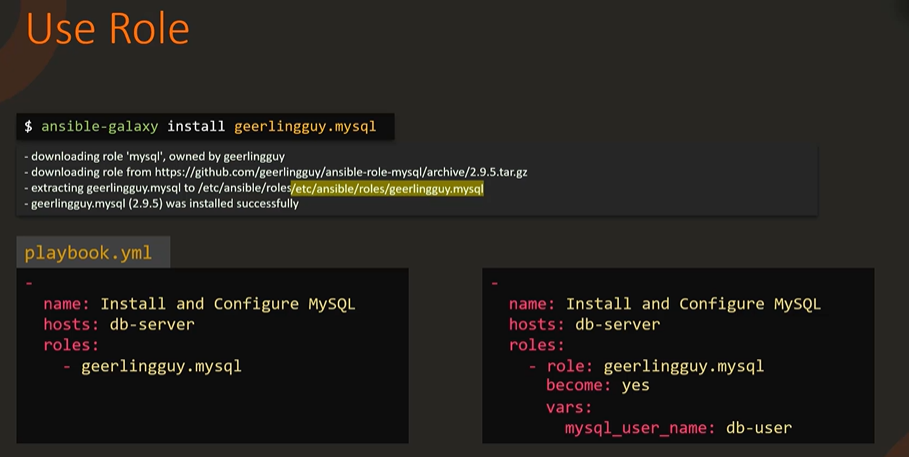
# **Roles**

Roles let you automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structure. After you group your content in roles, you can easily reuse them and share them with other users.

<https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html> <https://galaxy.ansible.com/>







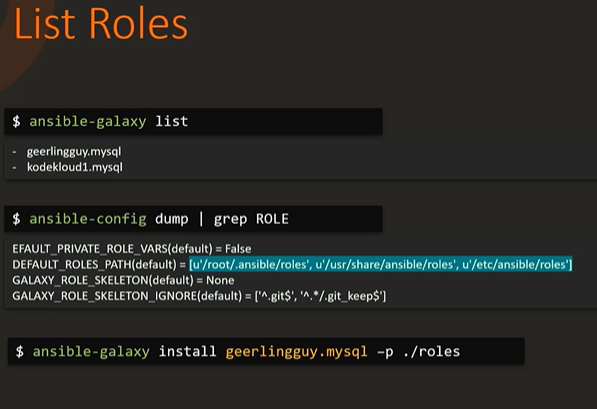
ansible-galaxy list

ansible-config dump |grep ROLE

ansible-galaxy init mysql

ansible-galaxy search

ansible-galaxy install <rolename>



Setting up a windows host:

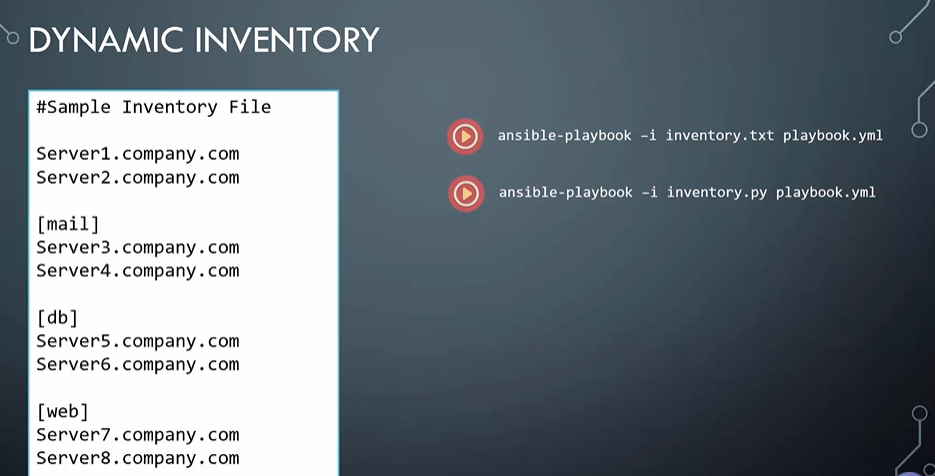
[Setting up a Windows Host — Ansible Documentation](https://docs.ansible.com/ansible/latest/user_guide/windows_setup.html#winrm-setup)

Ansible Patterns: <https://docs.ansible.com/ansible/latest/user_guide/intro_patterns.html>



Dynamic Inventory:

[Working with dynamic inventory — Ansible Documentation](https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html)



# **Developing Ansible modules**

A module is a reusable, standalone script that Ansible runs on your behalf, either locally or remotely. Modules interact with your local machine, an API, or a remote system to perform specific tasks like changing a database password or spinning up a cloud instance. Each module can be used by the Ansible API, or by the **ansible** or **ansible-playbook** programs. A module provides a defined interface, accepts arguments, and returns information to Ansible by printing a JSON string to stdout before exiting.

If you need functionality that is not available in any of the thousands of Ansible modules found in collections, you can easily write your own custom module. When you write a module for local use, you can choose any programming language and follow your own rules. Use this topic to learn how to create an Ansible module in Python. After you create a module, you must add it locally to the appropriate directory so that Ansible can find and execute it. For details about adding a module locally, see [Adding modules and plugins locally](https://docs.ansible.com/ansible/latest/dev_guide/developing_locally.html#developing-locally).

<https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_general.html>

Terraform integration with ansible:

[Integrating Terraform With Ansible Tutorial | Cprime](https://www.cprime.com/resources/blog/terraform-and-ansible-tutorial-integrating-terraform-managed-instances-with-ansible-control-nodes/#:~:text=Terraform%20and%20Ansible,-So%2C%20why%20do&text=Terraform%20is%20designed%20to%20provision,necessary%20applications%20on%20that%20machine.)