# **Golden Gate Bridge - Ansible - Automation Sibelius**

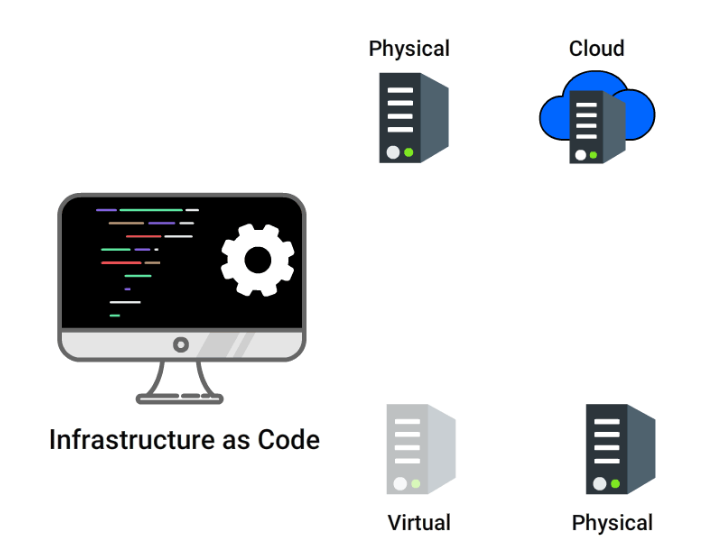
##### **What are You Going to Learn?**

We hope you have gained a good understanding of what is Infrastructure as a code. In this course, you will learn about

* Ansible and its benefits
* Ansible in Infrastructure as code (IaC)
* YAML format
* Few Ad\_Hoc commands
* Different parts of Ansible
* Role of Modules in Ansible
* How to write Playbooks
* How to Control flow of execution in Ansible
* Few points about configuration file (ansible.cfg)
* How to setup environment in local machine and online playground

By the end of this course, you will be in a position to write a few **ad-hoc commands** and **Playbook** of your own.

##### **What Is Infrastructure As Code?**

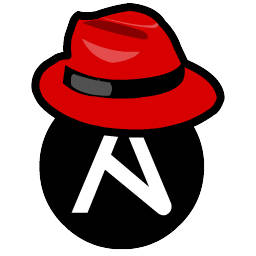


**IaC means writing code for infrastructure i.e. your systems and devices, which are used to run Softwares, are to be treated as software and defined using code(which can be done using a high level or descriptive language).**

For example: version control, testing, small deployments, use of design patterns etc.

**Configuration Management tools** are used to accomplish IaC.

##### **Why are You Here?**



**Infrastructure as Code** can be achieved using **Ansible**, which is one of the popular **Configuration Management tools**. In this course, we will have you explore various capabilities of Ansible.

What is Configuration Management?

**Configuration Management is a process of establishing, tracking and controlling the current design and build state of the system** *(software versions).*

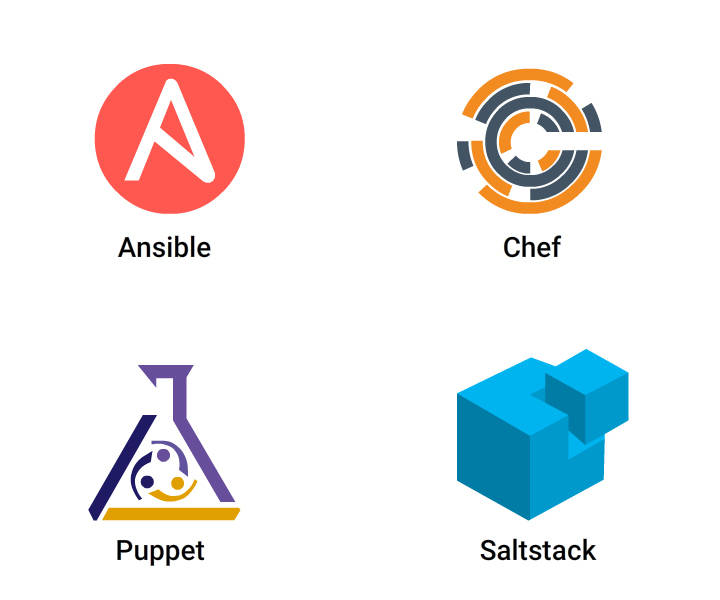
It also ensures that **past records of system state** is easily and accurately accessible which helps in project management, audit purposes, debugging etc.

##### **Before Configuration Management**

Consider you are planning for a **New Year Special Sale** in your e-commerce site. You need to

* **Scale up** your servers
* Then **configure** them(and all other old servers) for special new year sale
* This whole process would take **lot of effort and time**
* What if new configurations did not work as expected? Then you will have to **roll back to previous stable version**, which will add more work and subtract the profits and potential customers while in downtime.

##### **Configuration Management Tools**



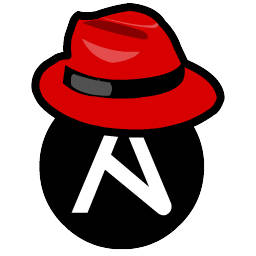
**You need some kind of Configuration Management tool that can automate these tasks:**

* Roll back to stable version with zero downtime
* Provide you with constant computing environment throughout SDLC
* Automatically scale up or down depending upon traffic

Some of the **popular Configuration Management tools are: Ansible, Puppet, Chef and Saltstack.**

You will now learn more about Ansible in upcoming cards.

##### **What is Ansible?**



**Ansible is an open source software, first released by Michael DeHaan in 2012 and owned by Red Hat.**

It is used to **automate**

* configuration management
* application deployment
* software provisioning and other IT needs

##### **Benefits of Ansible**

* **Simple:** Very easy to install, setup and learn. Written in YAML file which is pretty much like reading English.
* **Agentless:** Do not need to install any agents on target nodes.
* **Powerful:** It can model any complex IT workflow as it has 1100+ modules
* **Efficient:** You can customize modules, using any programming language
* **Secure:** Uses SSH for connection

Wondering about YAML language?? Read on to discover

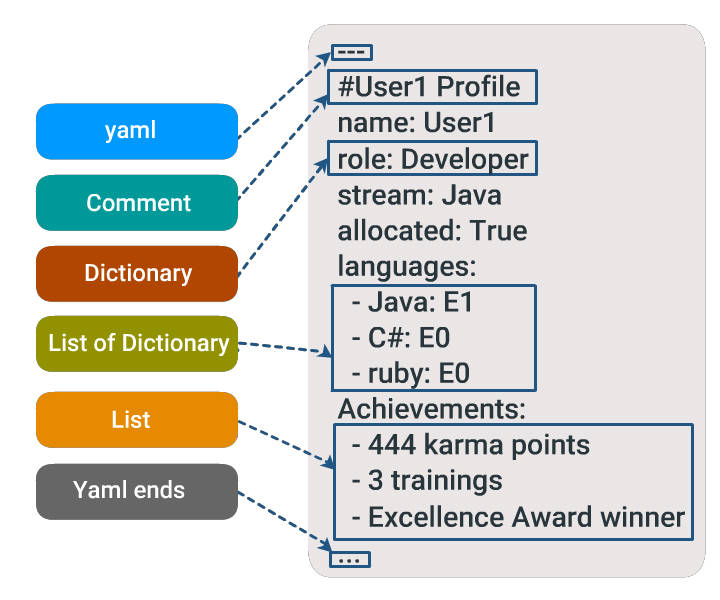
##### **What is YAML?**

**YAML** stands for **Yet Another Markup Language**. It is a data serialization language just like JSON.

***Why YAML, when we already have JSON or XML?***

* YAML files are easy to read and write for humans, similar to English.

##### **Foundation of YAML**



* YAML files should end as .yaml or .yml
* **Begins** with --- and **ends** with …
* # defines comment

##### **YAML is Case and Indentation Sensitive**

* **Members of a list should be at the same indentation level** starting with a dash(-) and space.
* Each item in the list is a **key: value pair (colon must be followed by a space)**, called as **dictionary**.
* At each level, exactly **two spaces are used for indentation**. Using tabs is not recommended here.

##### **Boolean Values**

Variables can be defined in YAML files as shown:

stream: Java

allocated: true

**Variables can be assigned boolean values in different ways** as shown:

allocated: yes

allocated: no

allocated: True

allocated: TRUE

allocated: false

##### **Data Structures**

Complicated data structures are possible in YAML.

You can define **lists having dictionaries**, **dictionaries having lists** or a **mix** of both.

In the following example,

* name and job are dictionaries. Skill is a **dictionary with a list**.
* David and Amy are **lists having dictionaries**

# Employee records

- David:

name: David Moore

job: Developer

skills:

- python

- sql

- java

- Amy:

name: Amy Brown

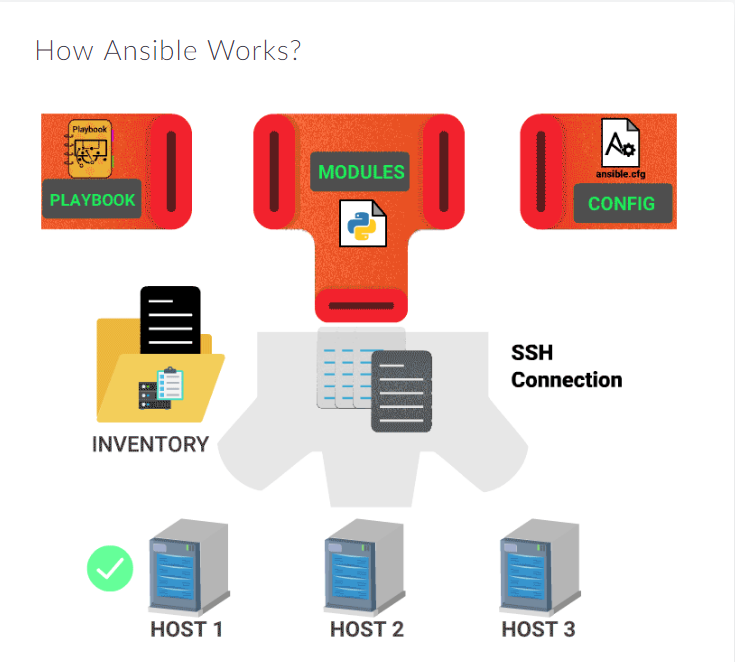
job: Developer

skills:

- angular

- redux

- react

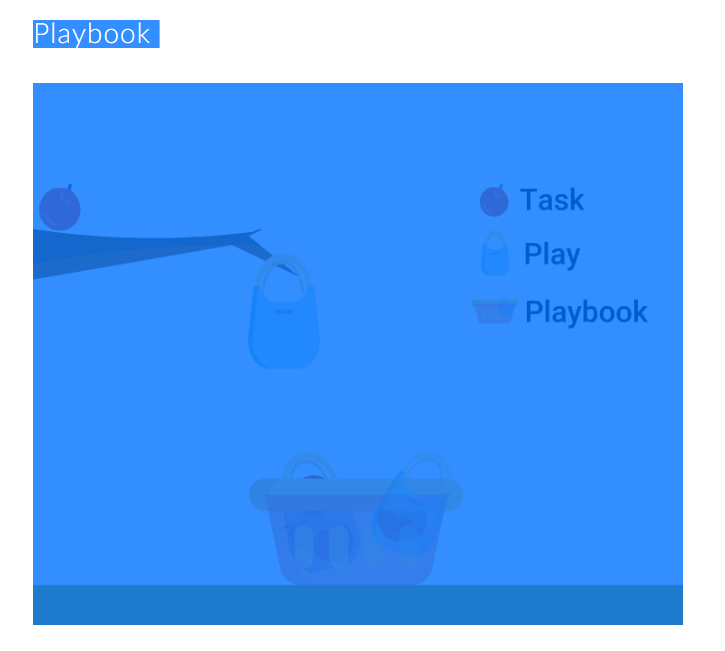


*As already discussed, Ansible is a configuration management tool, based on push-based architecture to automate configuration of your hosts to achieve a****desired state****.*

Following are the components of **Ansible architecture**

* **Inventory** - Defines the list of target hosts
* **Playbook (YAML file)** - Defines list of tasks
* **Module** - A python code invoked from tasks and executed on hosts
* **Control Machine** - Takes playbook and executes each task on particular group of hosts

##### **Playbook**

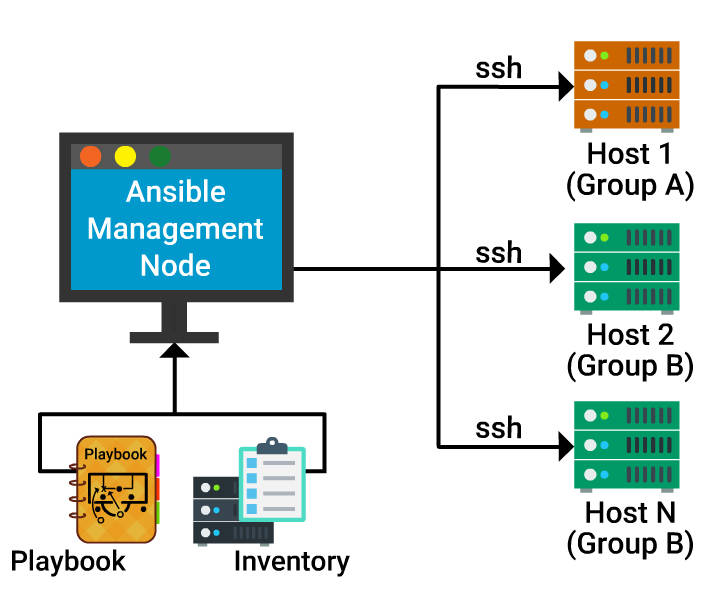


\*\*A Playbook is a file that defines the desired state of your system.

It contains plays, which has a list of tasks to run in sequence against a list of hosts\*\*.

* A play is set of tasks, grouped together to achieve an objective
* A task is an **instruction** you give to Ansible.
* They are written in **YAML** format, a **data serialization language**, that we discussed in previous cards.

##### **Inventory**



So Ansible captured the desired state through Playbook, but how would ansible know which machines it should configure through *Inventory*?

**The Inventory file in Ansible contains the list of all hosts (target systems/servers) that need to be configured. You can also group hosts under different names as shown:**

[group A]

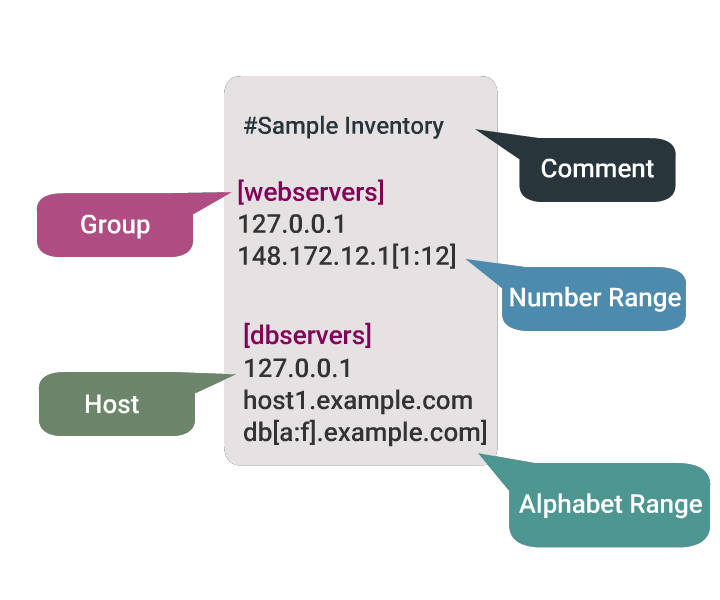
Host 1

[group B]

Host 2

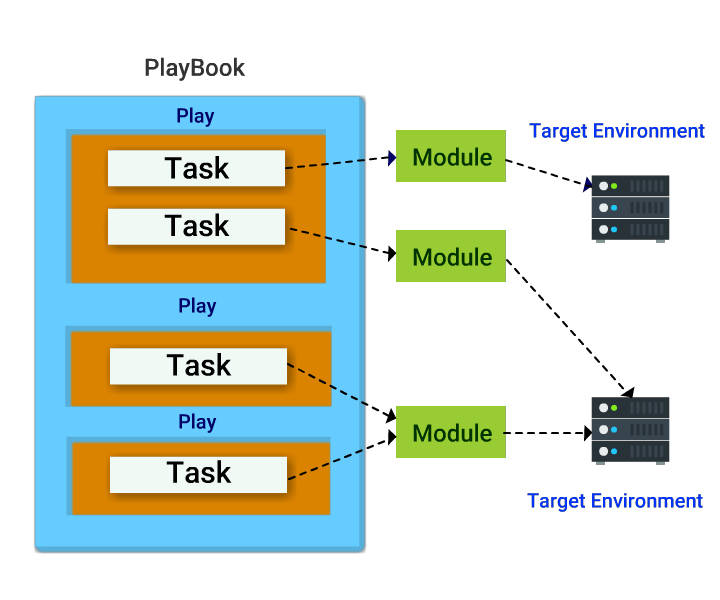
Host N

##### **More on Inventory**



* Default location of Inventory File: /etc/ansible/hosts
* You can define an Inventory file **statically (.ini file)** or **dynamically (.json file)** to Ansible.

##### **Modules Perform the Action**



**Modules are a piece of code that gets executed when you run playbook. You use them to describe the state you want the host to be in.**

**Each task in play is made of module and arguments.**

##### **Ad-hoc Keywords**

Before hopping into Ad-Hoc commands, let us first learn ***Ansible keywords***:

* **ansible**: This is a tool that allows you to ***run a single task*** at a time.

$ ansible <host-pattern> [-m module\_name] [-a args] [options]

* **ansible-playbook**: This is the tool used to ***run ansible playbook***

$ ansible-playbook <filename.yml> ... [options]

##### **Ad-hoc Keywords (Contd 1)**

* **ansible-console**: This is a REPL using which you can run ad-hoc commands on chosen inventories.

$ ansible-console <host-pattern> [-m module\_name] [-a args] [options]

* **ansible-pull**: This inverts the default push architecture of Ansible into a pull architecture, which has near-limitless scaling potential.

ansible-pull -U URL [options] [ <filename.yml> ]

* **ansible-doc**: Displays data on modules installed in Ansible libraries.

$ ansible-doc [-M module\_path] [-l] [-s] [module...]

##### **Ad-hoc Keywords (Contd 2)**

* **ansible-vault**: Using this you can ***encrypt*** any structured data file used by Ansible.

$ ansible-vault [create|decrypt|edit|encrypt|rekey] [--help] [options] file\_name

* **ansible-galaxy**: This is a shared ***repository for Ansible roles***. This ansible-galaxy command can be utilized to manage these roles, or to create a skeleton framework for the roles to be uploaded to Galaxy.

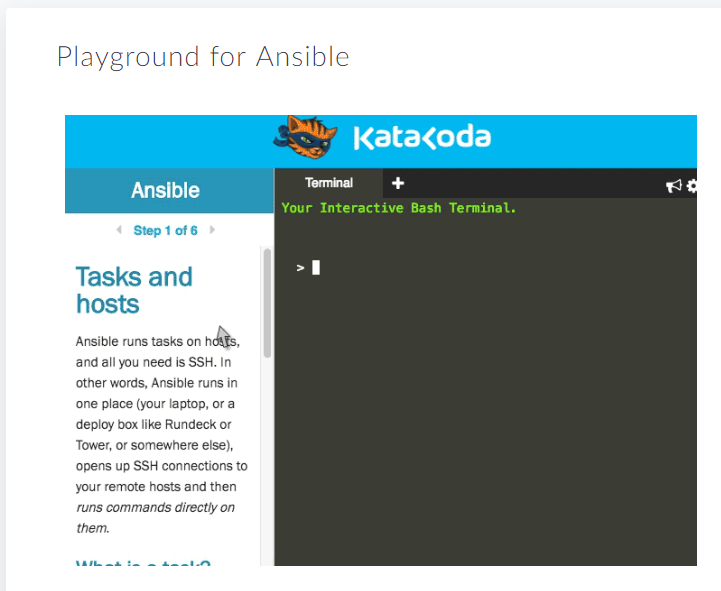
$ ansible-galaxy [delete|import|info|init|install|list|login|remove|search|setup] [--help] [options]

[Source](http://manpages.ubuntu.com/manpages/yakkety/man1/ansible.1.html)

##### **Short Hands In Ansible**

* -a: This tells the **arguments** to pass to the module
* -m: Execute the **module**
* -b: Use **privilege escalation (become)**
* -i: The path to the **inventory**, which defaults to /etc/ansible/hosts
* --version: Show program **version number**
* --help: Shows help message

[Source](http://manpages.ubuntu.com/manpages/yakkety/man1/ansible.1.html)



Throughout this course, you can use **[Katacoda Playground](https://www.katacoda.com/jonatanblue/scenarios/1)** to try out **Ad-hoc Commands**being introduced to you.

Katacoda is an online platform that provides you ***lab for hands-on, right in your browser,*** for you to test your Ansible codes and commands.

You are provided with the terminal of Control Machine and one host machine to test your commands.

To practice as you read, you first need to follow the **two steps shown in GIF to setup [Katacoda](https://www.katacoda.com/jonatanblue/scenarios/1)** environment.

# Welcome!

#### **jonatanblue - Ansible**

**Difficulty:** **Beginner**

**Estimated Time:** **10 minutes**

## What is Ansible?

[Ansible](http://www.ansible.com/how-ansible-works) is a radically simple IT automation engine that automatesprovisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

### What will I learn in this tutorial?

You will learn how to use Ansible to configure a Linux host.

Hint: You can click on many of the commands to run them directly

**Step 1 of 6**

#### Tasks and hosts

Ansible runs tasks on hosts, and all you need is SSH. In other words, Ansible runs in one place (your laptop, or a deploy box like Rundeck or Tower, or somewhere else), opens up SSH connections to your remote hosts and then runs commands directly on them.

### What is a task?

A task can be anything from creating a bucket in AWS S3, to launching an instance in Azure, installing pip on a server, updating a config file, or simply checking the time on a remote host. In this tutorial we focus on tasks to configure a remote Ubuntu host.

For example, this task installs pip on Ubuntu:

- name: ensure pip is installed

apt:

name: python-pip

state: installed

Every task uses a module. Here we use the apt module, a wrapper for the apt package manager, letting you specify what to do in [YAML syntax](http://docs.ansible.com/ansible/YAMLSyntax.html). There are [hundreds of different modules](http://docs.ansible.com/ansible/modules_by_category.html) included in Ansible.

### What is a host?

The host is where the tasks get run. It can be any number of remote hosts that you have SSH access to, or localhost.

Your hosts respective IP addresses or hostnames need to be stored in an inventoryfile for Ansible to be aware of them. Later when you run Ansible, you will specify one or more hosts, or groups of hosts, defined in the inventory file.

#### Creating an inventory

In this tutorial you have access to one host named host01. To let Ansible know about it, put its hostname in an inventory file. Let's call our inventory myhosts, with one group called group1.

echo "[group1]" > myhosts

Then add the host to the group:

echo "host01 ansible\_ssh\_user=ubuntu" >> myhosts

Here we're also passing the username to use for SSH access, as an [inventory parameter](http://docs.ansible.com/ansible/intro_inventory.html#list-of-behavioral-inventory-parameters).

#### Running Ansible

### Ad-hoc Tasks

To run an ad-hoc task, using a single module, use ansible.

For example, to check the current date and time of the remote host:

ansible group1 -i myhosts -m command -a date

(if the command fails the first time, try again in few seconds, as the test host may not yet have been provisioned)

* group1 = group to run against. You can also use the name of an individual host, so host01 would work just as well for this example.
* myhosts = name of the inventory file listing all available hosts
* command = module to use (the [command module](http://docs.ansible.com/ansible/command_module.html) returns the result of a bash command)
* date = argument for the module, in this case a standard shell command date.

When you run the command above, Ansible establishes an SSH connection with the remote host, executes the command, captures the output and returns it.

#### Running a playbook

## Playbook

A playbook is an entirely different way of running Ansible, that is far more powerful.

### What is a play?

A play is a set of tasks mapped to a set of hosts. Plays are organised inside a text file called a playbook.

### Running a playbook

This playbook runs one task, on our one host, host01. Note the indentation - it's important for how the file gets parsed. Blank lines are ignored, but makes the playbook more readable for humans.

---

- hosts: host01

become: true

tasks:

- name: ensure latest sysstat is installed

apt:

name: sysstat

state: latest

The above code is already in a file called site.yml. Let's check the contents:

cat site.yml

To run the playbook, use the ansible-playbook command with the inventory file myhosts:

ansible-playbook -i myhosts site.yml

Ansible should return the result 'Changed=1', indicating that the package was installed.

### Playbook breakdown

What happened here?

* --- denotes the beginning of a YAML file
* hosts: host tells Ansible to run the tasks on the host *host*
* become: true makes all your tasks run as sudo
* - name: is basically a comment, describing what the task does
* apt: specifies the module we want to use
* name: is an argument to the apt module, that specifies the name of the package to install.

To see all arguments for a specific module, allowed values, and other details, you can use the CLI documentation that is included with Ansible:

ansible-doc apt

To close the documentation, enter *q* in the terminal.

*NOTE: This tutorial is using Ubuntu. If your own hosts are RHEL/CentOS, you should use the****yum****module instead.*

#### Ansible ensures

A fundamental principle of Ansible is that it ensures a desired state.

From the previous example:

- name: ensure latest sysstat is installed

apt:

name: sysstat

state: latest

This tells Ansible to check if the latest version of sysstat is installed. If it is, Ansible will do nothing more, since the "latest" state is already achieved. If sysstat is not already there, or of an older version than what is available, Ansible installs the latest version.

If you run the playbook again, Ansible does the former, and instead of "Changed: 1", you will get "OK: 2, Changed 0". Try it out:

ansible-playbook -i myhosts site.yml

#### Ensure a package is removed

The apt module allows you to specify the state you wish the package to be in. If you want a specific version, you append it to the package name, for example:

- name: ensure sysstat is installed at version 10.2.0-1

apt:

name: sysstat=10.2.0-1

state: installed

If you want to ensure that the package is not installed, you can declare that with state: absent, and Ansible will ensure it.

Update the playbook to remove sysstat.

sed -i -e 's/state: latest/state: absent/' -e 's/ensure.\*/ensure sysstat is removed/' site.yml

Then re-run the playbook:

ansible-playbook -i myhosts site.yml

In summary, the same playbook should always be able to run on a host and - assuming there are no errors - produce exactly the desired state, regardless of what state the host started in. For example, should your playbook halt halfway through due to a network error, it would still finish successfully when you re-run it.

##### **Running Your First Ad-hoc Command**

**An ad-hoc command is a single statement to complete a particular task. For example: consider you want to check if you could connect to your hosts.**

Open **[Katacoda](https://www.katacoda.com/jonatanblue/scenarios/1)** and enter the following command:

ansible group1 -i myhosts -m ping

The above statement is a **single task** to **ping** target host and **return pong** if the connection is successful.

* ansible is a **keyword** you need to write before running any ad-hoc command
* group1 is the group name of the **list of hosts**
* -m means **module**, this is followed by the **module name** ping, which will be executed to achieve the task

To know more about each module you can try: *ansible-doc ping*.

##### **Copy a File to the Servers**

You can use **copy module** to copy a file from your **control machine to host** as shown:

$ touch test.txt

* This will create a sample file which could be used to copy

$ ansible host01 -i myhosts -m copy -a "src=test.txt dest=/tmp/"

* copy the file test.txt **from your control machine** (where ansible is installed) **to all the hosts** defined in myhosts inventory group
* -a means **arguments** of that module (here copy module)
* src is **attribute of copy module** that defines the **source path** of file or directory **on control machine**

Similarly, to fetch a file from Host to your Control Machine, you can use ***fetch module***. You may use *ansible-doc fetch* to know about it.

##### **Encrypting Your File**

As you just created a test.txt file, let us now encrypt the same using **ansible-vaultkeyword**.

* $ ansible-vault encrypt test.txt: encrypts the file.

This asks for a password to be set. Give a password and confirm it.

* ansible-vault edit test.txt: to edit the file and add some content.

This opens vi editor. Type some text, then save it(:wq)

* cat test.txt: to view the content inside.

Observe the output carefully

* ansible-vault decrypt test.txt: to decrypt the file, use the password set during encryption
* cat test.txt: now observe the output

##### **Create Directories and Files**

You can use **file module** to create files and directories, manage their permissions and ownership as shown:

ansible host01 -i myhosts -m file -a "dest=/tmp/test mode=644 state=directory"

* This will create **directory /tmp/test** on all the host01 of myhosts group
* mode defines permission of file/directory
* state can take value: file, directory, link, absent, etc

You can set the state to **absent** to delete a file or directory to delete it:

$ ansible host01 -i myhosts -m file -a "dest=/tmp/test state=absent"

##### **Automating with Ansible**

Till now you were executing each task (using Ad-hoc command) to create a folder, copy a file, encrypt or decrypt a file, etc.

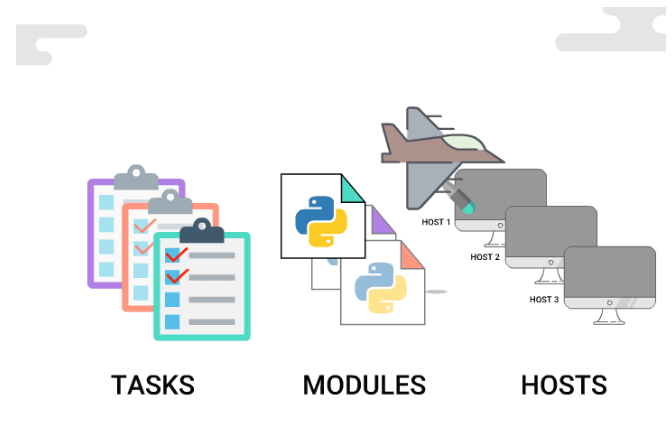
***What if you need to execute, say, \*\*500 tasks\*\* to configure a server?***

**Do not worry, as Ansible got you.**

You just need to define a Playbook that Ansible can play with, and have a popcorn watching Ansible in action.

Read on to find out what Playbook is.

##### **Modules**



**Modules are blocks of Python code that gets executed on remote hosts when you run each task in Playbook.**

Each module is built for a particular task and **you can use arguments to change the behavior of module**.

**state** argument is used to start or stop a service

##### **Currently Running Services**

service --status-all

##### **Stop a Service**

Let us try to stop sudo service:

ansible host01 -i myhosts -m service -a "name=sudo state=stopped"

##### **Start a Service**

Let us now try to start sudo service:

ansible host01 -i myhosts -m service -a "name=sudo state=started"

Following are ***state*** argument values: *running*,*started*,*stopped*,*restarted*,*reloaded*

##### **Things You Should Know About Modules**

* Also referred as **task plugins** or **library plugins**
* **Default location** for Ansible modules is /usr/share/ansible
* Take arguments in **key=value** format (state=stopped)
* **Returns data in JSON format**
* **Modules should be idempotent**, meaning they should not do any changes if current state of system matches with the desired final state
* To access **list of all installed modules** using command line: ansible-doc -l
* To see the **documentation of particular module** using command line: ansible-doc yumwhere yum is the module name
* You can **run modules from the command line or include them in Playbook**.
* Ansible allows you to **write your own module** (this you will learn later in advanced courses of Ansible).

Let us now go through some standard modules: *apt*, *yum*, *shell*, *command*, and *template*.

##### **APT Module**

**APT (Advanced Package Tool) is a command-line tool used to easily manage (install, remove, search, etc.) packages on Ubuntu/Debian based Linux systems.**

**Debian Based OS**: Ubuntu, Kali Linux, SteamOS and much more.

You can try the following in **[Katacoda](https://www.katacoda.com/jonatanblue/scenarios/1)**

$ ansible host01 -i myhosts -m apt -a "name=sudo state=latest"

#This is how you write in Playbook

- name: Upgrade sudo to the latest version

apt:`

name: sudo

state: latest

##### **YUM Module**

**YUM (Yellowdog Updater Modified) is a command-line tool used to easily manage (install, remove, search, etc.) packages on RPM (Red Hat Package Manager) based Linux systems.**

**Red Hat Based OS**: Fedora, Qubes OS, CentOS and much more.

#This is how you write in Playbook

- name: upgrade all packages

yum:

name: sudo

state: latest

##### **shell Module**

In shell module, the command name is followed by arguments, which **runs** on remote hosts through a shell(/bin/sh).

You **can use** various operations(|,<,> etc) and environment variable(#HOME).

ansible host01 -i myhosts -m shell -a "echo $TERM": This displays the terminal name of host machine

#This is how you write in Playbook

- name: Execute the command in remote shell

shell: echo $TERM

##### **command Module**

In command module, the command name is followed by arguments, which **does not run** on remote hosts through a shell(/bin/sh).

You **cannot use** various operations(|,<,> etc) and environment variable(#HOME).

* To check the list of files or folder in **remote host**: ansible host01 -i myhosts -m command -a "ls"
* Make a directory in **remote host**: ansible host01 -i myhosts -m command -a "mkdir folder1"
* Check files or folders in **remote host**: ansible host01 -i myhosts -m command -a "ls"

Now to check files or folders in ***your terminal*** use *ls* and observe the output. As you can see, ***using command, you can execute tasks on remote host***.

##### **command Everytime**

**Most of the Ansible modules are idempotent**.

But command module does not exhibit this property as this **runs every time you run playbook**. Hence you will always find the changed status on running the same Playbook again and again.

Consider you wrote a task to copy a file to remote hosts using command module.

Ansible **command module will copy the same file the number of times you run the Playbook**.

Had this been idempotent; Ansible will not copy from the second time, as the file is already present (current state = desired state).

##### **command can be Idempotent**

To overcome this, you can use creates or remove parameter, where you define the filename/pattern.

* creates: if filename **exists**, task will not run
* remove: if filename **does not exist**, task will not run

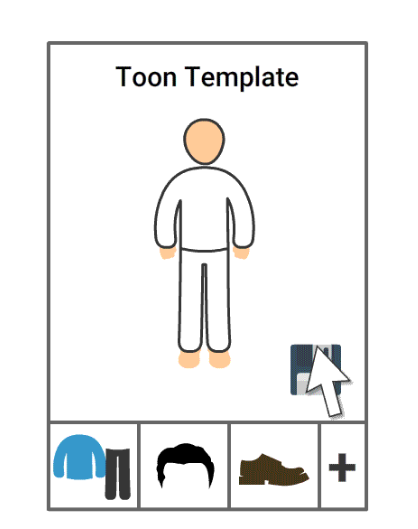
#This is how you write in Playbook

- name: copy a file, but do not copy if the file already exists

command: cp /home/dist/file1.txt /usr/someFolder/ creates=file1.txt

##### **template Module**

**Ansible template is a file having a defined structure with some variables and expressions, which can be replaced with the corresponding values to generate new configuration file.**



* Template files are **written in Jinja2 language** (.j2) (Read on to know more about Jinja2)
* Generally, in Ansible, these files are copied to remote hosts in **JSON or YAML format**

There are basically two files involved to define templates in Ansible:

* playbook(YAML file): here you substitute variables in template file with values
* template(Jinja2 file): here you define the template file in Jinja2 format

Let us take the example of the human body as a template. Depending on the various elements (footwear, dress, hair) you could name the template as a man or a woman.

##### **Intro to Jinja2**

Go through this video to learn about Jinja2. Jinja2(.j2) file gives output in many formats (JSON or YAML being common in Ansible).

##### **file: sample-playbook.yml**

This is where your template and variables are merged.

Let us consider another example of **Payslip as a template**, where a structure of payslip is pre-defined. Depending on the values being passed new payslip is generated for each name.

#This is how you write in Playbook

- hosts: all

vars:

quarter: false,

salary : 30000,

extra : 10000,

names: ["John","David"]

tasks:

- name: Ansible Template

template:

src: ../templates/sample-template.j2

dest: /home/sample-template

* src: defines path where your template file is kept.
* dest: path where you want to copy your file (mostly JSON or YAML formatted file).

##### **file: sample-template.j2**

This is where you define your template.

{% for name in names %}

Hi {{name}}!

{% if quarter -%}

Your pay cheque for this month is {{ salary + extra }}.

{% else -%}

Your pay cheque for this month is {{ salary }}.

{%- endif %}

{% endfor %}

* variable names are given within double curly braces {{ }}

##### **Hands-on Template**

You can test your Jinja2 templates [**online**](https://cryptic-cliffs-32040.herokuapp.com/) with the steps as explained.

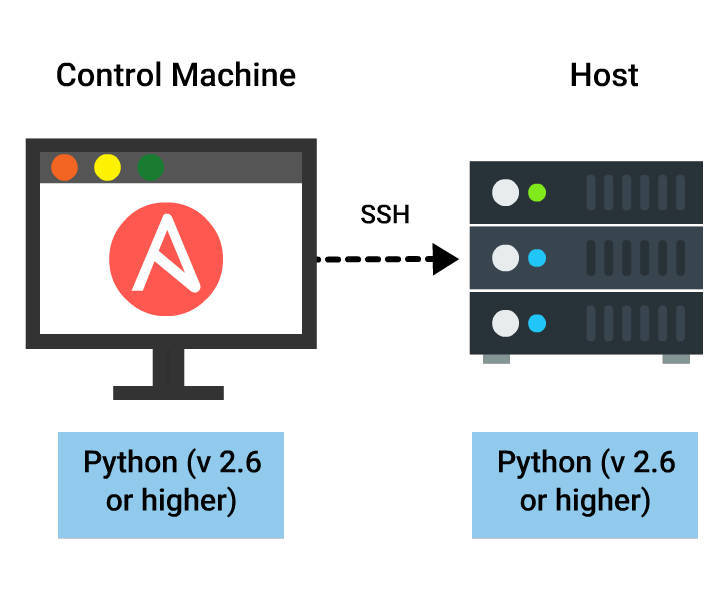
* Copy sample-template.j2 file and paste in **Template** box
* Copy **variables** from sample-playbook.yml and paste in **Values** box
* Click on **Convert**

##### **Try It Out - Template**

Try to make a template to display **Students mark sheet** that displays name of student, marks in Physics, Chemistry and Maths.

Provide relevant values via variables.

##### **Environment at a Glance**



**Control Machine Requirements**

* Linux/Mac OS (Windows not supported)
* Python (2.6 or later version) should be installed.
* Install Ansible.

**Host Requirements**

* Only Python needs to be installed.
* Uses SSH to connect to control machine.
* SFTP is used else scp can be configured.

##### **Step By Step**

You need to follow these steps to setup environment on your system:

1. Install **[VirtualBox](https://www.virtualbox.org/wiki/Downloads)**.
2. Install Vagrant. [Vagrant](http://www.slashroot.in/what-vagrant-and-how-does-it-work) enables virtual software development environment.
3. Install Ansible.

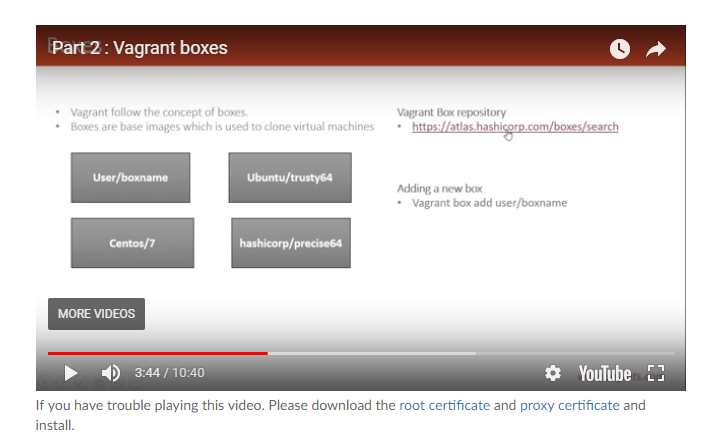
##### **Install Vagrant**

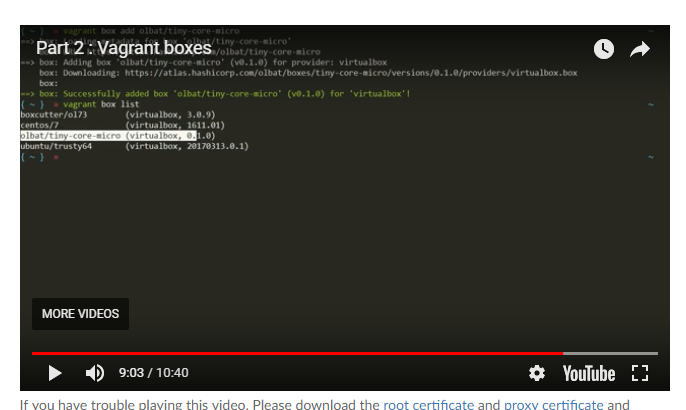
* [**Download and install Vagrant**](https://www.vagrantup.com/downloads.html)
* **Identify the OS** you want to install in your virtual system. Accordingly, you need to find the [**Vagrant Box**](https://atlas.hashicorp.com/boxes/search) (let us consider Ubuntu on both our control and host machines).
* **Create a folder on your local machine**, to keep your Vagrant Boxes.

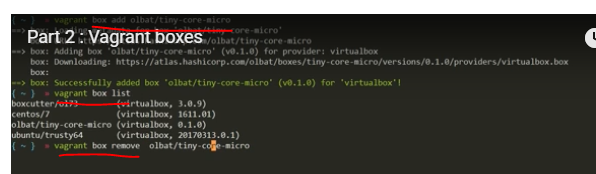
Now, **open your terminal** and move to the path where you created your folder for Vagrant boxes. Then run the following commands:

* $ vagrant init ubuntu/trusty64: Initialises vagrant file inside the folder
* $ vagrant up --provider VirtualBox: will download necessary files of your virtual machine
* $ vagrant ssh: will make secure connection with virtual machine

<https://www.hashicorp.com/boxes/search> - there u can find the vagrant boxes to download







##### **Install Ansible**

Once you are done with Virtual Box and Vagrant setup, you only need to run one last command in your terminal to install Ansible on the control machine.

### **On Ubuntu/Debian/Linux Mint**

$ sudo apt-get install ansible

### **On RHEL/CentOS/Fedora**

$ sudo yum install epel-release

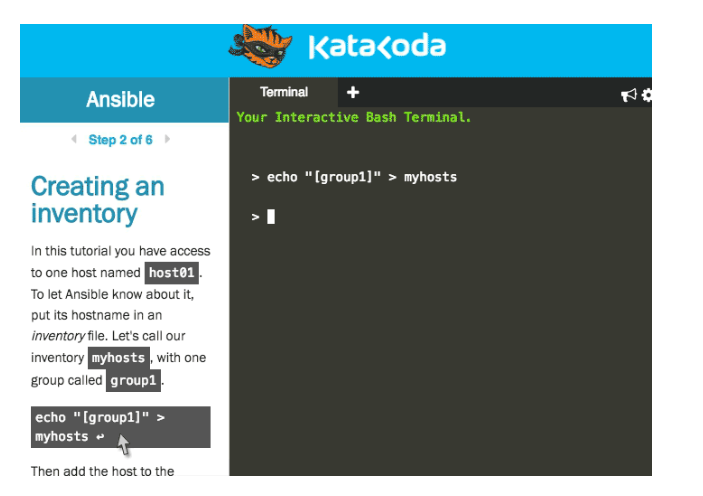
$ sudo yum install ansible

To verify if Ansible is successfully installed or not: ansible --version

You can follow *[Ansible Official Docs](http://docs.ansible.com/ansible/intro_installation.html" \l "latest-releases-on-mac-osx)* for detailed instructions.

##### **Writing Playbook**

Let us now write a Playbook to print **Hello World**. You can follow the steps parallelly in **[Katacoda](https://www.katacoda.com/jonatanblue/scenarios/1)**.



##### **Step By Step**

* **Open [Katacoda](https://www.katacoda.com/jonatanblue/scenarios/1)**
* Enter **commands given in Step 2 of 6**. This makes an SSH connection to your host.
* Open Playbook using the **vi editor** (eg: vi demo.yml).
* **Write** your playbook (given in next card).
* **Press Escape** key on your keyboard.
* **Save and close editor** using **:wq**(w: write, q: quit).
* **Run your playbook: $ ansible-playbook -i myhosts demo.yml**.
* **Step 1 of 6**

#### Tasks and hosts

* Ansible runs tasks on hosts, and all you need is SSH. In other words, Ansible runs in one place (your laptop, or a deploy box like Rundeck or Tower, or somewhere else), opens up SSH connections to your remote hosts and then runs commands directly on them.

### What is a task?

* A task can be anything from creating a bucket in AWS S3, to launching an instance in Azure, installing pip on a server, updating a config file, or simply checking the time on a remote host. In this tutorial we focus on tasks to configure a remote Ubuntu host.
* For example, this task installs pip on Ubuntu:
* - name: ensure pip is installed
* apt:
* name: python-pip
* state: installed
* Every task uses a module. Here we use the apt module, a wrapper for the apt package manager, letting you specify what to do in [YAML syntax](http://docs.ansible.com/ansible/YAMLSyntax.html). There are [hundreds of different modules](http://docs.ansible.com/ansible/modules_by_category.html) included in Ansible.

### What is a host?

* The host is where the tasks get run. It can be any number of remote hosts that you have SSH access to, or localhost.
* Your hosts respective IP addresses or hostnames need to be stored in an inventory file for Ansible to be aware of them. Later when you run Ansible, you will specify one or more hosts, or groups of hosts, defined in the inventory file.

#### Creating an inventory

In this tutorial you have access to one host named host01. To let Ansible know about it, put its hostname in an inventory file. Let's call our inventory myhosts, with one group called group1.

echo "[group1]" > myhosts

Then add the host to the group:

echo "host01 ansible\_ssh\_user=ubuntu" >> myhosts

Here we're also passing the username to use for SSH access, as an [inventory parameter](http://docs.ansible.com/ansible/intro_inventory.html#list-of-behavioral-inventory-parameters).

#### Running Ansible

### Ad-hoc Tasks

To run an ad-hoc task, using a single module, use ansible.

For example, to check the current date and time of the remote host:

ansible group1 -i myhosts -m command -a date

(if the command fails the first time, try again in few seconds, as the test host may not yet have been provisioned)

* group1 = group to run against. You can also use the name of an individual host, so host01 would work just as well for this example.
* myhosts = name of the inventory file listing all available hosts
* command = module to use (the [command module](http://docs.ansible.com/ansible/command_module.html) returns the result of a bash command)
* date = argument for the module, in this case a standard shell command date.

When you run the command above, Ansible establishes an SSH connection with the remote host, executes the command, captures the output and returns it.

#### Running a playbook

## Playbook

A playbook is an entirely different way of running Ansible, that is far more powerful.

### What is a play?

A play is a set of tasks mapped to a set of hosts. Plays are organised inside a text file called a playbook.

### Running a playbook

This playbook runs one task, on our one host, host01. Note the indentation - it's important for how the file gets parsed. Blank lines are ignored, but makes the playbook more readable for humans.

---

- hosts: host01

become: true

tasks:

- name: ensure latest sysstat is installed

apt:

name: sysstat

state: latest

The above code is already in a file called site.yml. Let's check the contents:

cat site.yml

To run the playbook, use the ansible-playbook command with the inventory file myhosts:

ansible-playbook -i myhosts site.yml

Ansible should return the result 'Changed=1', indicating that the package was installed.

### Playbook breakdown

What happened here?

* --- denotes the beginning of a YAML file
* hosts: host tells Ansible to run the tasks on the host host
* become: true makes all your tasks run as sudo
* - name: is basically a comment, describing what the task does
* apt: specifies the module we want to use
* name: is an argument to the apt module, that specifies the name of the package to install.

To see all arguments for a specific module, allowed values, and other details, you can use the CLI documentation that is included with Ansible:

ansible-doc apt

To close the documentation, enter q in the terminal.

#### Ansible ensures

A fundamental principle of Ansible is that it ensures a desired state.

From the previous example:

- name: ensure latest sysstat is installed

apt:

name: sysstat

state: latest

This tells Ansible to check if the latest version of sysstat is installed. If it is, Ansible will do nothing more, since the "latest" state is already achieved. If sysstat is not already there, or of an older version than what is available, Ansible installs the latest version.

If you run the playbook again, Ansible does the former, and instead of "Changed: 1", you will get "OK: 2, Changed 0". Try it out:

ansible-playbook -i myhosts site.yml

#### Ensure a package is removed

The apt module allows you to specify the state you wish the package to be in. If you want a specific version, you append it to the package name, for example:

- name: ensure sysstat is installed at version 10.2.0-1

apt:

name: sysstat=10.2.0-1

state: installed

If you want to ensure that the package is not installed, you can declare that with state: absent, and Ansible will ensure it.

Update the playbook to remove sysstat.

sed -i -e 's/state: latest/state: absent/' -e 's/ensure.\*/ensure sysstat is removed/' site.yml

Then re-run the playbook:

ansible-playbook -i myhosts site.yml

In summary, the same playbook should always be able to run on a host and - assuming there are no errors - produce exactly the desired state, regardless of what state the host started in. For example, should your playbook halt halfway through due to a network error, it would still finish successfully when you re-run it.

##### **Sample Playbook 1 - "demo.yml"**

---

- name: this play displays "hello world"

hosts: all

tasks:

- name: displaying "hello world"

shell: echo "hello world"

- name: displaying wishes for the day

shell: echo "have a good day"

##### **Playbook Step 1 - Name and Hosts**

---

- name: this play displays "hello world"

hosts: all

* A Playbook always starts with three dashes ---.
* name tells the name of the play.
* hosts tell the list of hosts on which this play will be played.

##### **Playbook Step 2 - Tasks**

tasks:

- name: displaying "hello world"

shell: echo "hello world"

- name: displaying wishes for the day

shell: echo "have a good day"

* name is optional but is always recommended as it improves readability
* shell: echo "hello world" is a single task. This executes **shell module** and calls its **echo argument** to display the message written

##### **Playbook Step 3 - Run Your Playbook**

Type ansible-playbook -i myhosts demo.yml from the terminal to run your Playbook.

* ansible-playbook is the command to execute your Playbook
* -i myhosts tell the inventory name is myhosts
* demo.yml is the playbook that needs to be executed

##### **Output Of "demo.yml"**

This is how your Playbook would run

PLAY [this play displays "hello world"] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GATHERING FACTS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [host01]

TASK: [displaying "hello world"] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [host01]

TASK: [displaying wishes for the day] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [host01]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

host01 : ok=3 changed=2 unreachable=0 failed=0

##### **Sample Playbook 2 - Install Apache**

**Let us now take another example to install apache in your hosts**.

This is how your Playbook would look:

---

- name: install apache

hosts: all

sudo: yes

tasks:

- name: install apache2

apt: name=apache2 update\_cache=yes state=latest

Save and run this playbook as *apache.yml* in Katacoda and check the output.

**Please note that for installing in RHEL/CentOS/Fedora,yum module is used instead of apt.**

##### **Restructuring Playbook**

You can define the same playbook as:

---

- name: install apache

hosts: all

sudo: yes

tasks:

- name: install apache2

apt:

name: apache2

update\_cache: yes

state: latest

* observe **colon : is used while structuring arguments vertically**, whereas equal to sign = is used while structuring arguments horizontally

Both ways of structuring your playbook is fine.

* **This vertical structuring of arguments is not a list, as list starts with dash sign -**.

Here **name, update\_cache and state are arguments of module apt**, hence they do not start with -.

##### **Handlers**

**Handlers are special tasks that run only on certain triggers like notify keyword.**

* Handlers run mostly at the **end of the play**.
* Handlers **run only once** even if you run the playbook multiple times.

This is how a handler would look:

---

- name: install apache

...................

tasks:

- name: install apache2

.....................

notify:

- start Apache

............

handlers:

- name: start Apache

..........

##### **Conditionals In Ansible**



Recollect while reading the section on setting up your environment or while installing an application ***, you need different modules (apt and yum) to execute task***.

What will you do, if you have to configure 50 such servers with different OS and requirements?

**This is where conditionals can be used to decide and control the execution flow in Ansible.**

##### **Conditionals - When Clause**

**When clause in Ansible is a raw jinja2 expression that defines the condition which will be evaluated for TRUE or FALSE.**

tasks:

- name: "shutdown Debian flavored systems"

command: /sbin/shutdown -t now

when: ansible\_os\_family == "Debian"

##### **Conditionals - When Clause...**

* You can **group conditions using parenthesis ()**

tasks:

- name: "shutdown CentOS 6 and Debian 7 systems"

command: /sbin/shutdown -t now

when: (ansible\_distribution == "CentOS" and ansible\_distribution\_major\_version == "6") or

(ansible\_distribution == "Debian" and ansible\_distribution\_major\_version == "7")+

* You can **define multiple conditions**, where all of them should be true to execute the tasks

tasks:

- name: "shut down CentOS 6 systems"

command: /sbin/shutdown -t now

when:

- ansible\_distribution == "CentOS"

- ansible\_distribution\_major\_version == "6"

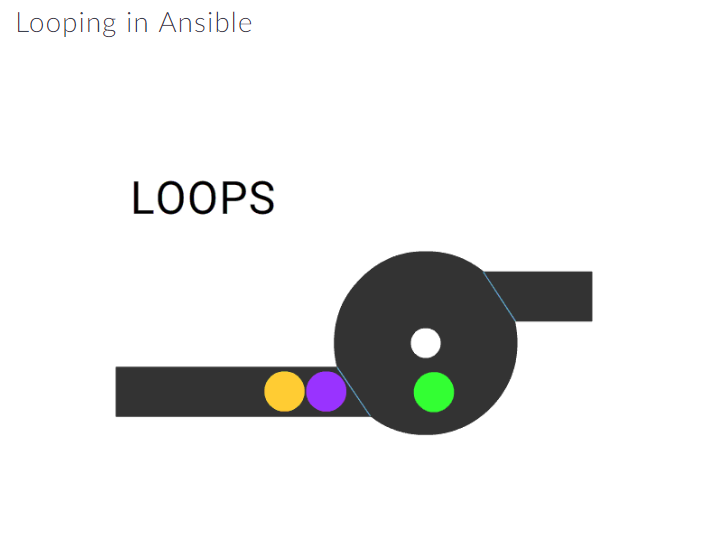
### Ansible sibelius - Try It Out-conditional (when clause)

**Step 1 of 4**

#### Step 1

Create a folder in "Host" machine folder1 in /home/ubuntu/ location, using ansible Ad-Hoc command (and not ansible-playbook).

Hint: ansible host01 -i myhosts -m command -a "mkdir folder1"



**If you need to repeat the same task with different items (loop through), you can use with\_items clause**

*For example, consider you want to add several users*

- name: add several users

user:

name: "{{ item }}"

state: present

groups: "developer"

with\_items:

- raj

- david

- john

- lauren

##### **Looping Through The Inventory**

**You can loop through the Inventory file to list all hosts or hosts from a play, using with\_items with the play\_hosts or groups variables,**

# show all the hosts in the inventory

- debug:

msg: "{{ item }}"

with\_items:

- "{{ groups['all'] }}"

# show all the hosts in the current play

- debug:

msg: "{{ item }}"

with\_items:

- "{{ play\_hosts }}"

##### **Sample Playbook 3 - Conditions And Loop**

This Playbook will add Java Packages to different systems (handling Ubuntu/Debian OS)

- name: debian | ubuntu | add java ppa repo

apt\_repository:

repo=ppa:webupd8team/java

state=present

become: yes

when: ansible\_distribution == 'Ubuntu'

- name: debian | ensure the webupd8 launchpad apt repository is present

apt\_repository:

repo="{{ item }} http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main"

update\_cache=yes

state=present

with\_items:

- deb

- deb-src

become: yes

when: ansible\_distribution == 'Debian'

# Learn how to install softwares or packages using Playbook.

#### Step 1

create a playbook test.yml that will install

apache2

sqlite3

git

##### **ansible.cfg**

You told how your hosts should behave (via Playbook). But how do you tell Ansible (your control machine) should behave?

through ansible.cfg

**ansble.cfg is a configuration file that defines how Ansible should behave.**

It tells

* how to establish an ssh connection
* duration of ssh connection with the host
* how to run the playbook
* where to log errors that might occur while playing playbook on hosts etc.

##### **Ansible Configuration Settings**

**Your ansible.cfg file will have the following settings:**

* [defaults]
* [privilege\_escalation]
* [paramiko\_connection]
* [ssh\_connection]
* [accelerate]

You will now learn the frequently used settings in upcoming cards. For detailed understanding, you may refer to the *[Ansible Official Docs](http://docs.ansible.com/ansible/intro_configuration.html" \l "id79)*.

<https://www.katacoda.com/jonatanblue/scenarios/1> - session

<https://docs.ansible.com/ansible/latest/installation_guide/intro_configuration.html#id79> - Ansible Configuration Information web url.

##### **[defaults]**

### **poll\_interval**

For **asynchronous tasks in Ansible, this tells the frequency of checking the status of task completion**. The default value is 15 seconds; which implies, for every 15 sec, it will check if the task is completed.

poll\_interval = 15

### **sudo\_user**

This is the default user to sudo. If --sudo-user is not specified in an Ansible playbook, the default is the most logical: ‘root’:

sudo\_user = root

##### **[defaults]**

### **ask\_pass**

This tells **if Ansible Playbook should ask for a password** by default. If SSH key is used for authentication, the **default behavior is no.**

ask\_pass = True

### **ask\_sudo\_pass**

Just like ask\_pass, this asks for **sudo password** by default while sudoing.

ask\_sudo\_pass = True

### **remote\_port**

If your systems did not define an alternate value in Inventory for SSH port, this sets the default value to 22.

remote\_port = 22

##### **[privilege\_escalation]**

\*For some tasks to execute in Ansible, you require administrative access to the system.

The settings under [privilege\_escalation] will escalate your privileges.\*

### **become**

If set to true or yes, you **activate privilege escalation**.

Default behavior is *no*.

become=True

### **become\_method**

You can define the **method for privilege escalation**.

Default is *sudo*, other options are su, pbrun, pfexec, doas, ksu.

become\_method=sudo

##### **[privilege\_escalation]**

### **become\_user**

This allows you to **become the user over privilege escalation.**

Default is *root*.

become\_user=root

### **become\_ask\_pass**

Asks the **password for privilege escalation**, the default is *False*.

become\_ask\_pass=False

##### **Other Settings**

### **accelerate\_timeout**

This setting will **close the socket connection** if no data is received from the client for the defined time duration.

default: *accelerate\_timeout = 30*

### **record\_host\_keys**

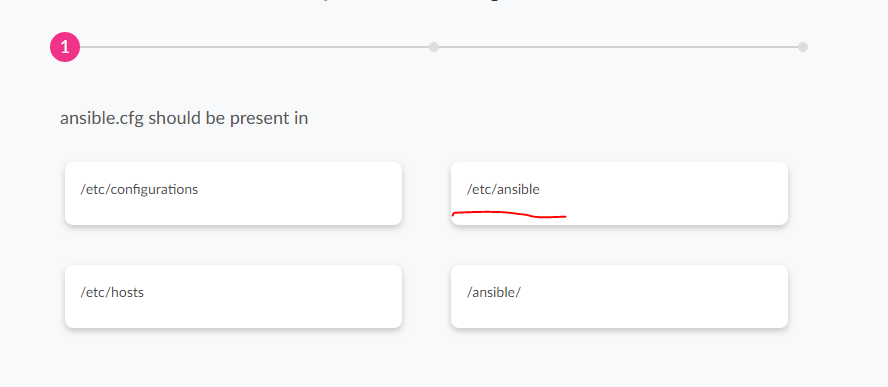
If the setting is True(default value), this will **record new hosts on the user's host file**, provided **host key checking is enabled**. Setting it to False, the performance will improve and which is recommended when **host key checking is disabled**.

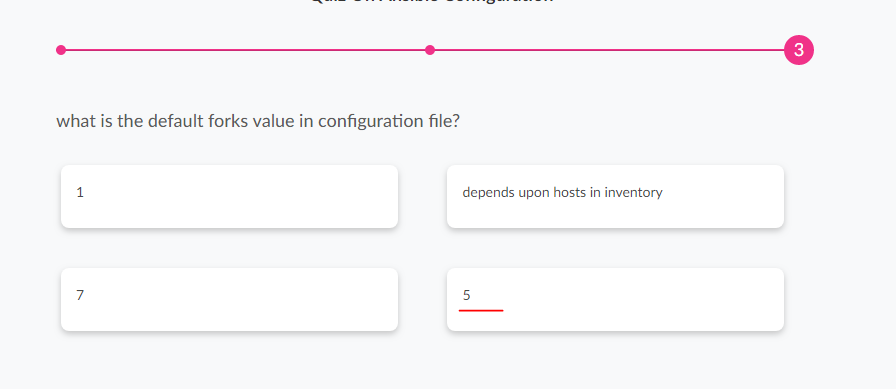
record\_host\_keys=False

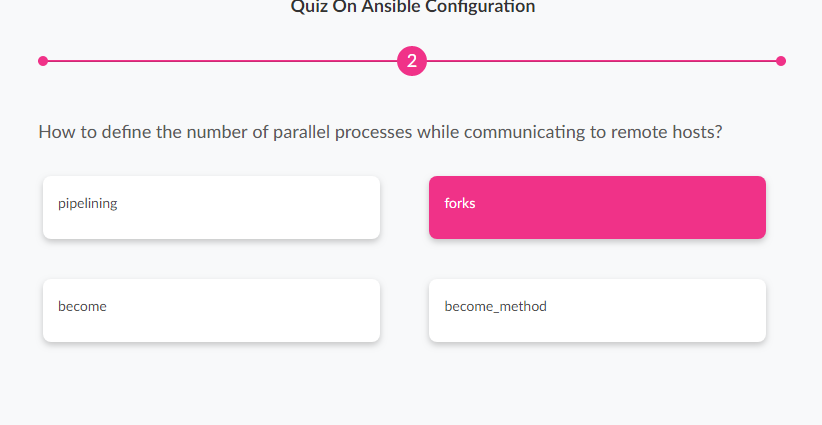
### **pipelining**

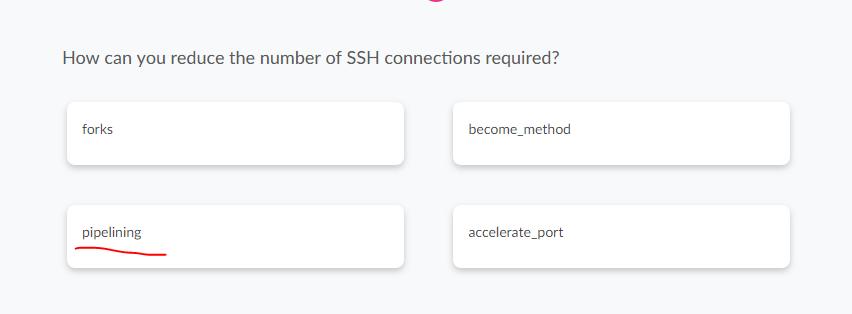
If enabled, the **number of SSH connections** required for execution of a module on the remote server is **reduced**, improving the performance significantly.

By default this is disabled: *pipelining = False*.









# Welcome!

#### **besilbal - Ansible sibelius - Try It Out-Write A Playbook**

**Difficulty:** **Beginner**

**Estimated Time:** **10 minutes**

# Learn how to write a Playbook that will have two plays to configure Nginx and PostgreSQL.

#### Step 1

Write a Playbook test.yml in the following format

Play 1 - Configuring Nginx.

* Task 1: Install Nginx.
* Task 2: Ensure Nginx is running.

Play 2 - Configuring PostgreSQL.

* Task 1: Install PostgreSQL.
* Task 2: Ensuring PostgreSQL is running.

[Hint: you can use apt and service modules]

##### **Expected Output - Try It Out**

On executing ansible-playbook -i myhosts demo.yml, you can expect the output as shown:

PLAY [play1 install and start nginx] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GATHERING FACTS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [host01]

TASK: [Installs nginx web server] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [host01]

TASK: [start nginx] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [host01]

PLAY [play2 install PostgreSQL] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GATHERING FACTS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [host01]

TASK: [Installing PostgreSQL] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [host01]

TASK: [ensure PostgreSQL is running] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [host01]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

host01 : ok=6 changed=2 unreachable=0 failed=0

##### **Ansible Is Idempotent**

**Try executing the same playbook for the second time and observe the output. Thus, Ansible is Idempotent**.

An *Idempotent* operation results to the same output, if executed multiple times (no changes).

##### **Ansible Is Idempotent**

**Try executing the same playbook for the second time and observe the output. Thus, Ansible is Idempotent**.

An *Idempotent* operation results to the same output, if executed multiple times (no changes).

Example of playbook

Vi test.yml

---

- hosts: app

become: **true**

tasks:

- name: Setup nginx conf

template:

src=nginx.conf.tpl

dest=/etc/nginx/nginx.conf

notify: restart nginx

handlers:

- name: restart nginx

service:

name=nginx

state=restarted

execute the playbook

ansible-playbook –I myhosts test.yml

# [**create a playbook test.yml that will install apache2 sqlite3 git?**](https://stackoverflow.com/questions/47175473/create-a-playbook-test-yml-that-will-install-apache2-sqlite3-git)

Test.yml

---

- name: install apache2, sqlite3, git pn remote server

hosts: host01

sudo: yes

tasks:

- name: Install list of packages

action: apt pkg={{item}} state=installed

with\_items:

- apache2

- sqlite3

- git