**Ansible Tutorial**

I hope you went through my previous blog to learn [***what is Ansible***](http://www.edureka.co/blog/what-is-ansible/?utm_source=blog&utm_medium=content-link&utm_campaign=ansible-tutorial) and the most used terminologies of Ansible. In case you haven’t, please do check it out so that you can get a better understanding of this Ansible Tutorial.

Let me give you an overview of this ‘Ansible Tutorial’:

* You will learn to [write playbooks](https://www.edureka.co/blog/ansible-tutorial/#ansible_playbook)
* You will learn about different [modules in Ansible](https://www.edureka.co/blog/ansible-tutorial/#ansible_modules)
* You will learn to write [Adhoc commands](https://www.edureka.co/blog/ansible-tutorial/" \l "adhoc_commands)
* [Hands on](https://www.edureka.co/blog/ansible-tutorial/#hands_on) with Ansible

**Ansible Playbook Tutorial | DevOps Training**

**Ansible Tutorial – Writing Ansible Playbooks**

Playbooks in Ansible are written in YAML format. It is a human-readable data serialization language. It is commonly used for configuration files. It  can also be used in many applications where data is being stored.

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.

All members of a list are lines beginning at the same indentation level starting with a “- ” (dash and space). More complicated data structures are possible, such as lists of dictionaries or mixed dictionaries whose values are lists or a mix of both.

 e.g. For a list of departments in edureka:

departments:

- marketing

- sales

- solutions

- content writing

- support

- product

Now let me give you an example of a dictionary:

-USA

-continent: North America

-capital: Washington DC

-population: 319 million

**Hosts And Users:**

For each play in a playbook, you get to choose which machines in your infrastructure to target and which remote user to complete the tasks. To include hosts in Ansible inventory, we will be using the IP addresses of the host machines.

Generally the hosts are a list one or more groups or host patterns, separated by colons. The remote user is just the name of the user account.

**Variables:**

Ansible uses variables which are defined previously to enable more flexibility in playbooks and roles. They can be used to loop through a set of given values, access various information like the host name of a system and replace certain strings in templates with specific values.

Ansible already defines a rich set of variables, individual for each system. Whenever Ansible will run on a system, all facts and information about the system are gathered and set as variables.

But there is a *rule* for naming variables. Variable names should be letters, numbers, and underscores. Variables should always start with a letter. E.g. wamp\_21, port5 is valid variable names, whereas 01\_port, \_server are invalid.

**Tasks:**

Tasks allow you to break up bits of configuration policy into smaller files. Task includes pull from other files. Tasks in Ansible go with pretty much the English meaning of it.

E.g: Install <package\_name>, update <software\_name> etc.

**Handlers:**

Handlers are just like regular tasks in an Ansible playbook, but are only run if the Task contains a notify directive and also indicates that it changed something. For example, if a config file is changed, then the task referencing the config file may notify a service restart handler.

Let me give you an example of a playbook which will start the Apache httpd server program:

---

- hosts: webservers

vars:

http\_port: 80

max\_clients: 200

remote\_user: root

tasks:

- name: ensure apache is at the latest version

yum: name=httpd state=latest

- name: write the apache config file

template: src=/srv/httpd.j2 dest=/etc/httpd.conf

notify:

- restart apache

- name: ensure apache is running (and enable it at boot)

service: name=httpd state=started enabled=yes

handlers:

- name: restart apache

service: name=httpd state=restarted

I hope the example will relate you to all the description of the playbook components that I have mentioned above. If it is still not clear to you, don’t worry all your doubts will be clear in the later part of this blog.

This is all about playbooks. The playbooks which will be written by you. But Ansible provides you with a wide range of modules as well, which you can use.

**[Learn DevOps Now](https://goo.gl/3zLUUy?utm_source=blog&utm_medium=button&utm_campaign=ansible-tutorial-learn-devops-now" \t "_blank)**

**Ansible Tutorial – Modules**

Modules in Ansible are idempotent. From a RESTful service standpoint, for an operation (or service call) to be idempotent, clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request.

There are different types of modules in Ansible

* Core modules
* extras modules

**Core Modules**

These are modules that the core Ansible team maintains and will always ship with Ansible itself. They will also receive slightly higher priority for all requests than those in the “extras” repos.

The source of these modules is hosted by Ansible on GitHub in the Ansible-modules-core.

**Extras Modules**

These modules are currently shipped with Ansible, but might be shipped separately in the future. They are also mostly maintained by the Ansible community. Non-core modules are still fully usable, but may receive slightly lower response rates for issues and pull requests.

Popular “extras” modules may be promoted to core modules over time.

The source for these modules is hosted by Ansible on GitHub in the Ansible-modules-extras.

E.g: The one of the extras module in Remote Management Modules is ipmi\_power module, which is a power manger for the remote machines. It requires python 2.6 or later and pyghmi to run.

You can use this module by writing an adhoc command like the one I have written below:

ipmi\_power : name ="test.domain.com" user="localhost" password="xyz" state="on"

**Ansible Tutorial – Return Values**

Ansible modules normally return a data structure that can be registered into a variable, or seen directly when output by the Ansibleprogram. Each module can optionally document its own unique return values.

Some examples of return values are:

* changed: returns with a boolean value whenever the task makes any change.
* failed: returns a boolean value, if the task is failed
* msg: it returns a string with a generic message relayed to the user.

**Ansible Tutorial – AdHoc Commands**

Adhoc commands are simple one line command to perform some action. Running modules with Ansible commands are adhoc commands.

E.g:

ansible host -m netscaler -a "nsc\_host=nsc.example.com user=apiuser password=apipass"

The above adhoc command uses the netscaler module to disable the server. There are hundreds of modules available in Ansible from where you can refer to and write adhoc commands.

Well, enough with all the theoretical explanations, let me explain you Ansible with some hands on.

**Ansible Tutorial – Hands On**

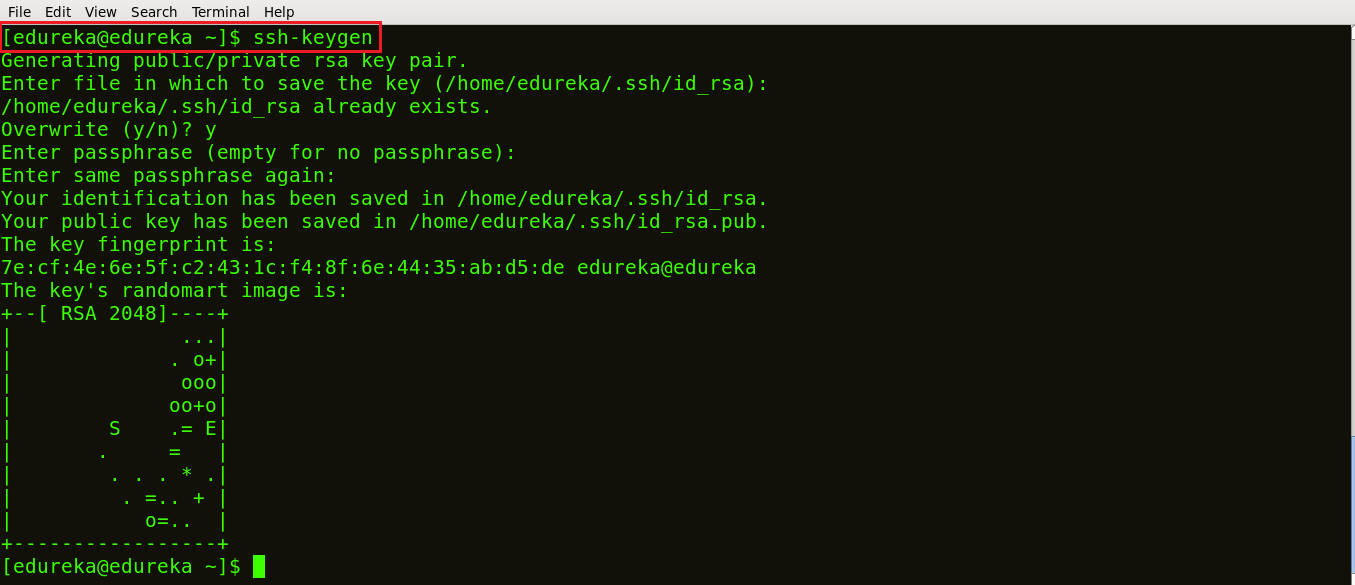
I am going to write a playbook to install Nginx on my node/host machine.

Let’s begin :)

**Step 1:** Connect to your hosts using SSH. For that, you need to generate a public SSH key.

Use the command below:

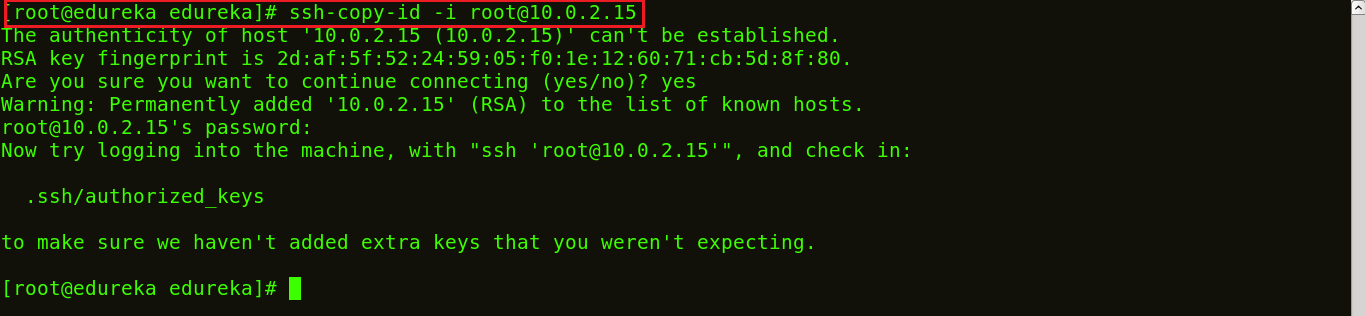
**ssh-keygen**



As you can see in the snapshot above, the command **ssh-keygen** generated a public SSH key.

**Step 2:** Your next task is to copy the public SSH key on your hosts. In order to do that, use the command below:

**ssh-copy-id -i root@<IP address of your host>**

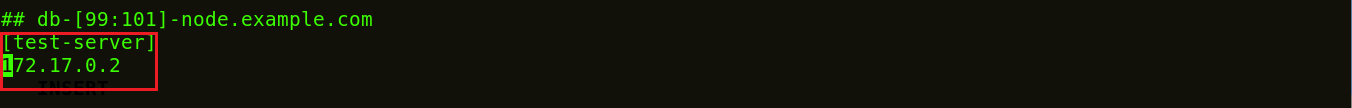


The snapshot above shows the SSH key being copied to the hosts.

**Step 3:** List the IP addresses of your hosts/nodes in your inventory.

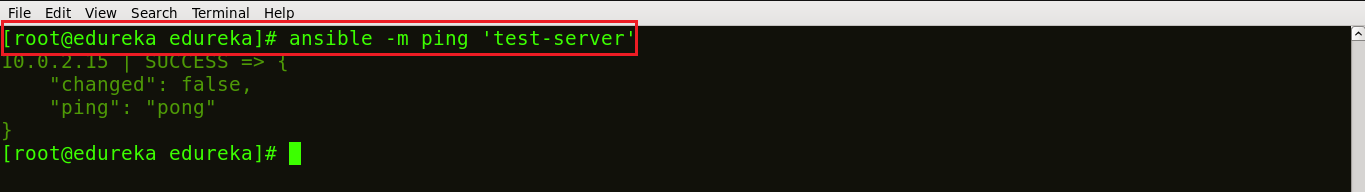
Use the following command:

**vi /etc/ansible/hosts**



This will open a vi editor where you can list down the IP addresses of your hosts. This is now your inventory.

**Step 4:** Let’s ping to ensure a connection has been established.

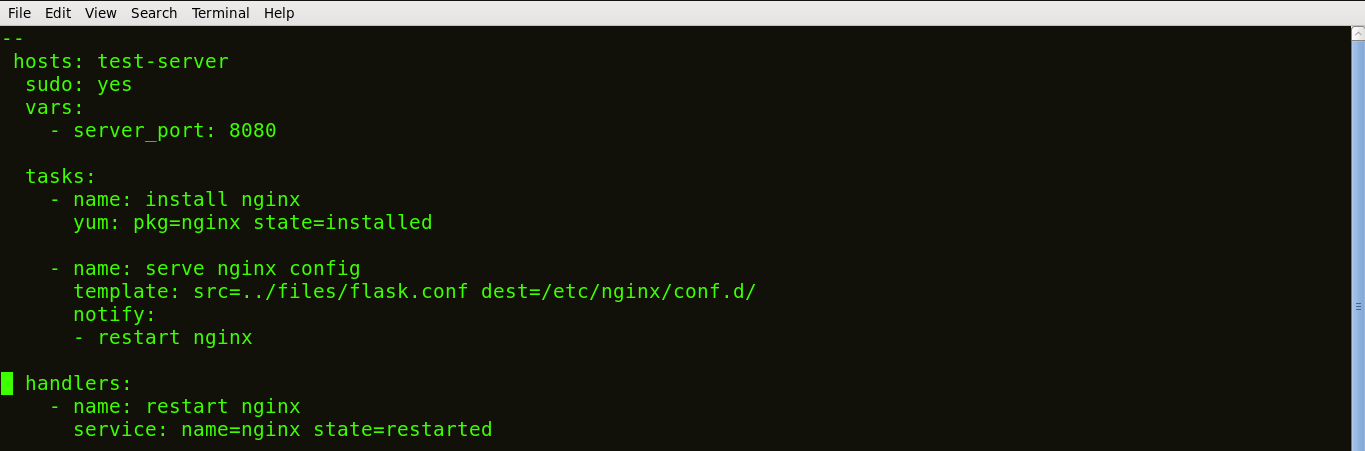


The snapshot above confirms that connection has been made between your control machine and host.

**Step 5:** Let us now write a playbook to install Nginx on the host machine. You can write your playbook in the vi editor. For that, simply create your playbook, using the command:

**vi <name of your file>.yml**

The below snapshot shows my playbook to install Nginx written in YAML format.



The tasks of a playbook are defined in YAML as a list of dictionaries and executed from top to bottom. If we have several hosts, then each task is tried for each host before moving on to the next one. Each task is defined as a dictionary that can have several keys, such as “name” or “sudo” which signify the name of the task and whether it requires sudo privileges.

A variable *server\_port* is set that listens on TCP port **8080** for incoming requests.

Here, the first task is to get the necessary package for installation of Nginx and then install it. Internally, Ansible will check if the directory exists and create it if it’s not, otherwise it will do nothing.

The next task is to configure Nginx. In Nginx, contexts contain configuration details.

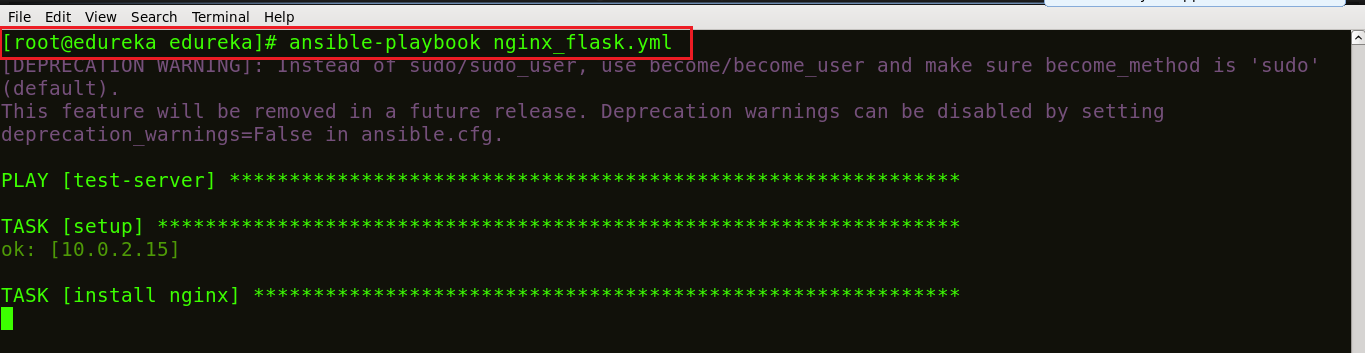
Here, the template is a file you can deploy on hosts. However, template files also include some reference variables which are pulled from variables defined as part of an Ansible playbook or facts gathered from the hosts. Facts containing the configuration details are being pulled from a source directory and being copied to a destination directory.

Handlers here define the action to be performed only upon notification of tasks or state changes. In this playbook, we defined, notify: restart Nginx handler which will restart Nginx once the files and templates are copied to hosts.

Now, save the file and exit.

**Step 6:** Now let’s run this playbook, using the command below:

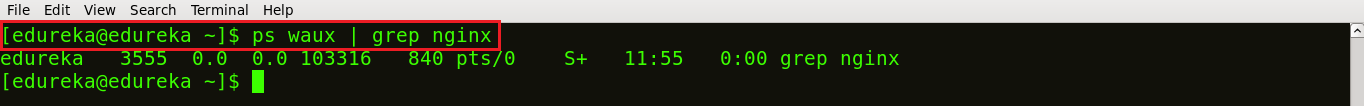
**ansible-playbook <name of your file>.yml**



We can see in the screenshot above that our task is getting executed; Nginx being installed.

**Step 7:** Let’s check if Nginx is installed on my host machine. Use the command below:

**ps waux | grep nginx**



You can see in the screenshot above, that different process ids 3555 and 103316 are running which ensures that Nginx is running on your host machines.

**Congratulations!** You have successfully deployed Nginx on your host using Ansible playbooks. I hope you have enjoyed reading this Ansible Tutorial blog. Please let me know if you have any queries in the comment section below