**What is Ansible?**

[Ansible](https://tekslate.com/tutorials/ansible/) is a software tool to deploy application using ssh without any downtime. It is also used to manage and configure software applications. Ansible is developed by Python language.

**What are the Advantages of Ansible?**

-Agent-less

-Verylow overhead

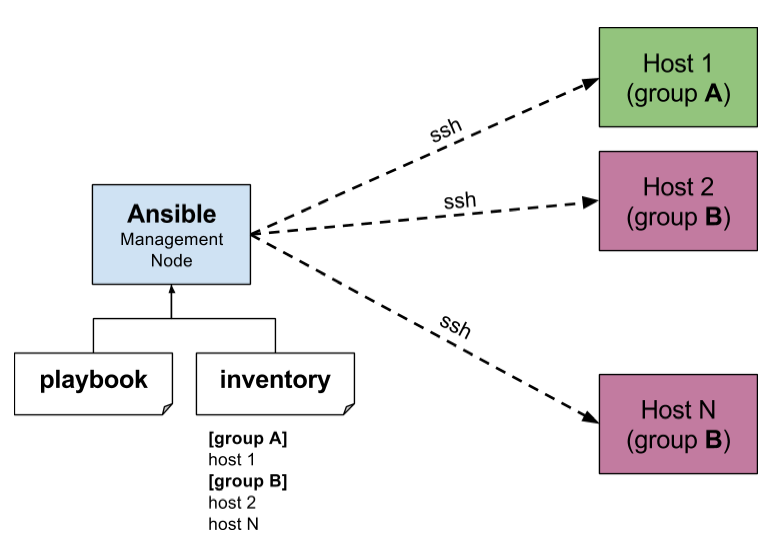
-Good performance

**How Ansible Works?**

There are many similar automation tools available like Puppet, Capistrano, Chef, Salt, Space Walk etc, but Ansible categorize into two types of server: controlling machines and nodes.

The controlling machine, where Ansible is installed and Nodes are managed by this controlling machine over SSH. The location of nodes is specified by controlling machine through its inventory.

The controlling machine (Ansible) deploys modules to nodes using SSH protocol and these modules are stored temporarily on remote nodes and communicate with the Ansible machine through a JSON connection over the standard output.



Ansible is agent-less, that means no need of any agent installation on remote nodes, so it means there are no any background daemons or programs are executing for Ansible, when it’s not managing any nodes.

Ansible can handle 100’s of nodes from a single system over SSH connection and the entire operation can be handled and executed by one single command ‘ansible’. But, in some cases, where you required to execute multiple commands for a deployment, here we can build playbooks.

Playbooks are bunch of commands which can perform multiple tasks and each playbooks are in YAML file format.

**Is there a web interface / REST API / etc?**

Yes, Ansible, Inc makes a great product that makes Ansible even more powerful and easy to use. See [Ansible Tower](http://docs.ansible.com/ansible/tower.html).

**How do I submit a change to the documentation?**

Documentation for Ansible is kept in the main project git repository, and complete instructions for contributing can be found in the docs.

**When should I use {{ }}? Also, how to interpolate variables or dynamic variable names**

A steadfast rule is ‘always use {{ }} except when *when:*‘. Conditionals are always run through Jinja2 as to resolve the expression, so *when:* *failed\_when:* and *changed\_when:* are always templated and you should avoid adding *{{}}*.

In most other cases you should always use the brackets, even if previouslly you could use variables without specifying (like *with\_* clauses), as this made it hard to distinguish between an undefined variable and a string.

Another rule is ‘moustaches don’t stack’. We often see this:

{{ somevar\_{{other\_var}} }}

The above DOES NOT WORK, if you need to use a dynamic variable use the hostvars or vars dictionary as appropriate:

{{ hostvars[inventory\_hostname]['somevar\_' + other\_var] }}

**How to install Ansible**

Installation of Ansible Ubuntu 14.04

The best way to get Ansible for Ubuntu is to add the project’s PPA (personal package archive) to your system.

To do this effectively, we need to install the software-properties-common package, which will give us the ability to work with PPAs easily. (This package was called python-software-properties on older versions of Ubuntu.)

sudo apt-get update

sudo apt-get install software-properties-common

Once the package is installed, we can add the Ansible PPA by typing the following command:

sudo apt-add-repository ppa:ansible/ansible

Press ENTER to accept the PPA addition.

Next, we need to refresh our system’s package index so that it is aware of the packages available in the PPA. Afterwards, we can install the software:

sudo apt-get update

sudo apt-get install ansible

We now have all of the software required to administer our servers through Ansible.

**How do I generate crypted passwords for the user module?**

The mkpasswd utility that is available on most Linux systems is a great option:

mkpasswd --method=sha-512

If this utility is not installed on your system (e.g. you are using OS X) then you can still easily generate these passwords using Python. First, ensure that the [Passlib](https://code.google.com/p/passlib/) password hashing library is installed.

pip install passlib

Once the library is ready, SHA512 password values can then be generated as follows:

python -c "from passlib.hash import sha512\_crypt; import getpass; print sha512\_crypt.encrypt(getpass.getpass())"

Use the integrated [Hashing filters](http://docs.ansible.com/ansible/playbooks_filters.html#hash-filters) to generate a hashed version of a password. You shouldn’t put plaintext passwords in your playbook or host\_vars; instead, use [Vault](http://docs.ansible.com/ansible/playbooks_vault.html) to encrypt sensitive data.

**How do I get ansible to reuse connections, enable Kerberized SSH, or have Ansible pay attention to my local SSH config file?**

Switch your default connection type in the configuration file to ‘ssh’, or use ‘-c ssh’ to use Native OpenSSH for connections instead of the python paramiko library. In Ansible 1.2.1 and later, ‘ssh’ will be used by default if OpenSSH is new enough to support ControlPersist as an option.

Paramiko is great for starting out, but the OpenSSH type offers many advanced options. You will want to run Ansible from a machine new enough to support ControlPersist, if you are using this connection type. You can still manage older clients. If you are using RHEL 6, CentOS 6, SLES 10 or SLES 11 the version of OpenSSH is still a bit old, so consider managing from a Fedora or openSUSE client even though you are managing older nodes, or just use paramiko.

We keep paramiko as the default as if you are first installing Ansible on an EL box, it offers a better experience for new users.

**What is the best way to make content reusable/redistributable?**

If you have not done so already, read all about “Roles” in the playbooks documentation. This helps you make playbook content self-contained, and works well with things like git submodules for sharing content with others.

If some of these plugin types look strange to you, see the API documentation for more details about ways Ansible can be extended.

**How do I see all the inventory vars defined for my host?**

You can see the resulting vars you define in inventory running the following command:

ansible -m debug -a "var=hostvars['hostname']" localhost

**How do I copy files recursively onto a target host?**

The “copy” module has a recursive parameter, though if you want to do something more efficient for many files, look at the “synchronize” module instead, which wraps rsync. See the module index for info on both modules.

**What is Ansible Role?**

Ansible can interact with configured clients from the command line with the ansible command, and how you can automate configuration with playbooks run through the ansible-playbook command.

The first step in creating a role is creating its directory structure. To create the base directory structure, we’re going to use a tool bundled with Ansible called ansible-galaxy:

$ ansible-galaxy init azavea.packer

azavea.packer was created successfully

That command will create an azavea.packer directory with the following structure:

├── README.md

├── defaults

│ └── main.yml

├── files

├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── tasks

│ └── main.yml

├── templates

└── vars

└── main.yml

**How do I access a variable name programmatically?**

An example may come up where we need to get the ipv4 address of an arbitrary interface, where the interface to be used may be supplied via a role parameter or other input. Variable names can be built by adding strings together, like so:

{{ hostvars[inventory\_hostname]['ansible\_' + which\_interface]['ipv4']['address'] }}

The trick about going through hostvars is necessary because it’s a dictionary of the entire namespace of variables. ‘inventory\_hostname’ is a magic variable that indicates the current host you are looping over in the host loop.

**How do I access shell environment variables?**

If you just need to access existing variables, use the ‘env’ lookup plugin. For example, to access the value of the HOME environment variable on management machine:

---

# ...

vars:

local\_home: "{{ lookup('env','HOME') }}"

If you need to set environment variables, see the Advanced Playbooks section about environments.

Ansible 1.4 will also make remote environment variables available via facts in the ‘ansible\_env’ variable:

{{ ansible\_env.SOME\_VARIABLE }}

############### 2 #####################

## ****Ansible Playbook Tutorial****

## ****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.

## ****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 Ansible program. 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 oneline 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****

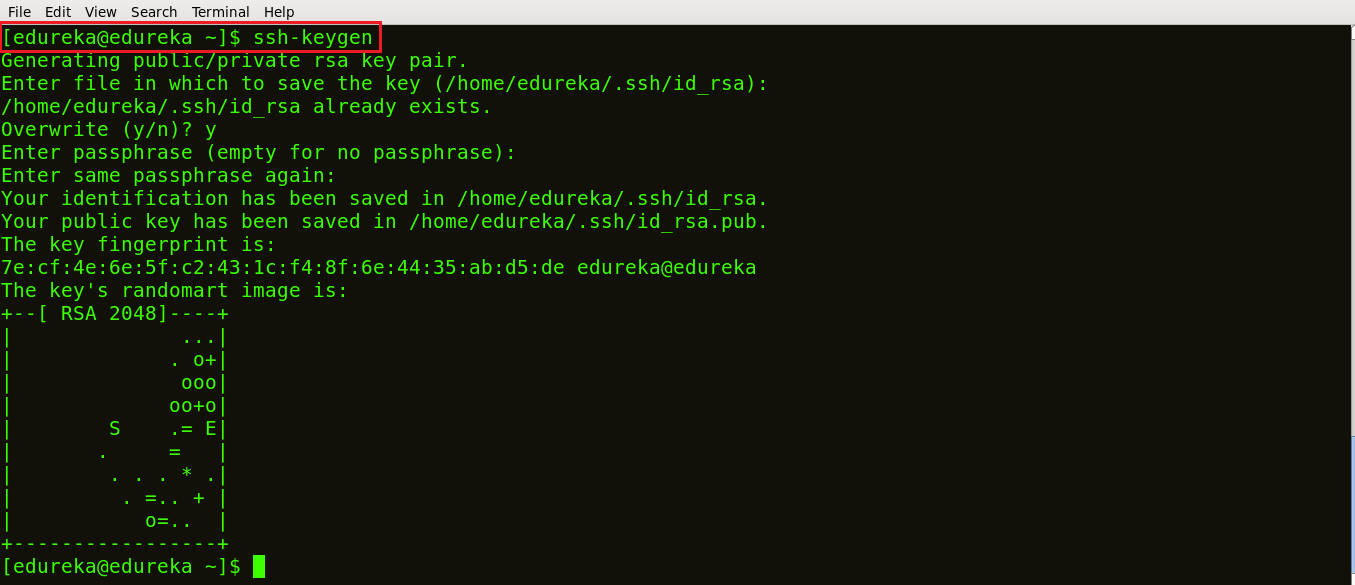
I am going to write a playbook to install Nginx on my node/host machine.

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:

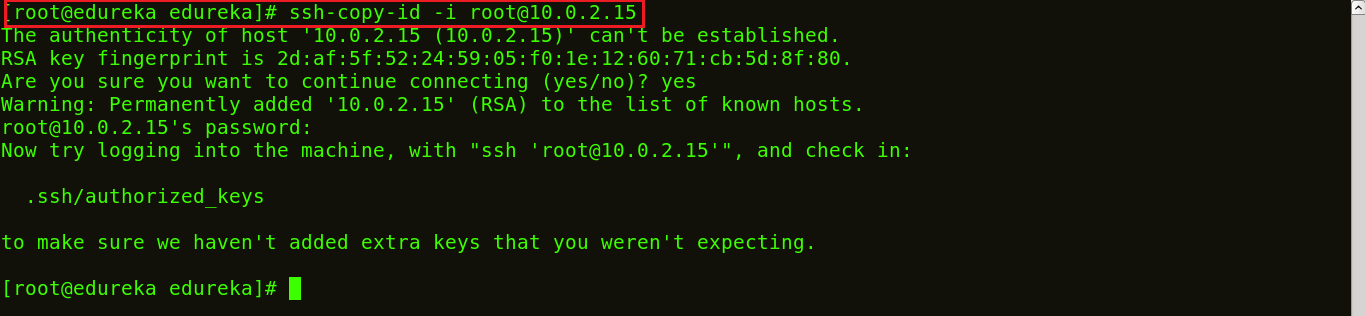
**ssh-keygen**



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>**

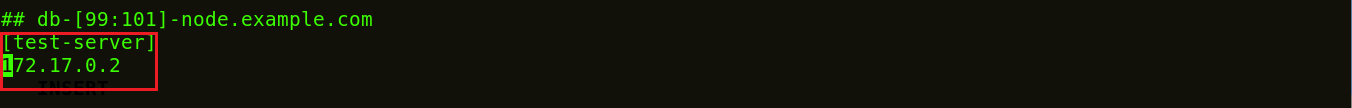


The snapshot above shows the SSH key being copied to the hosts.

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

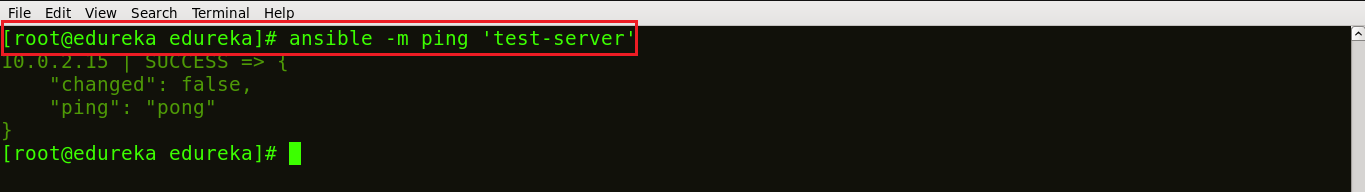
Use the following command:

**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.

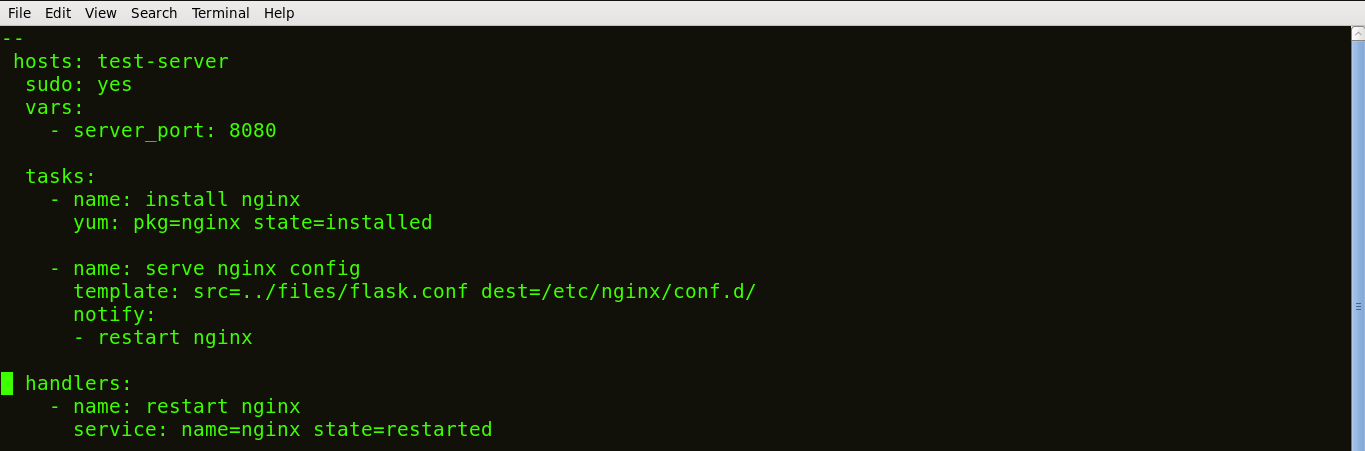


The snapshot above confirms that connection has been made between your control machine and host.

**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.

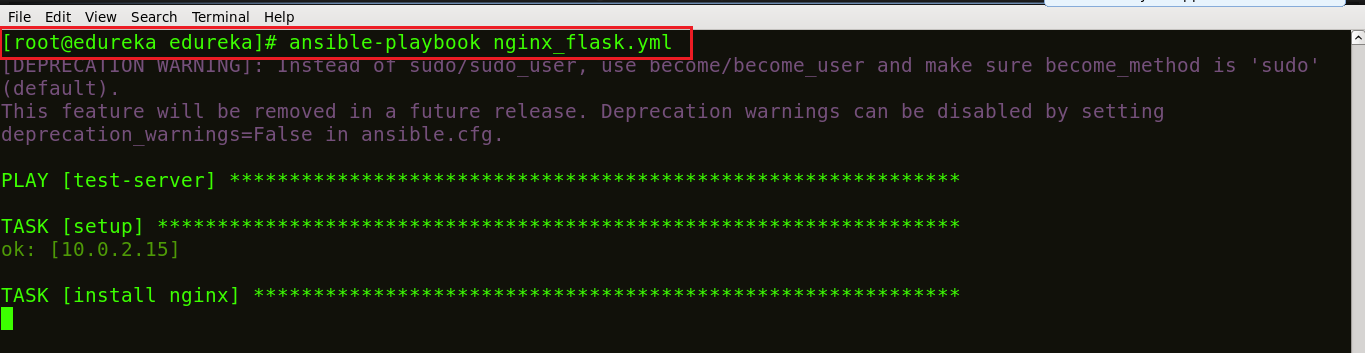
Here, the template is a file you can deploy on hosts. However, template files also include some reference variables which are pulled from variables defined as part of an Ansible playbook or facts gathered from the hosts. Facts containing the configuration details are being pulled from a source directory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

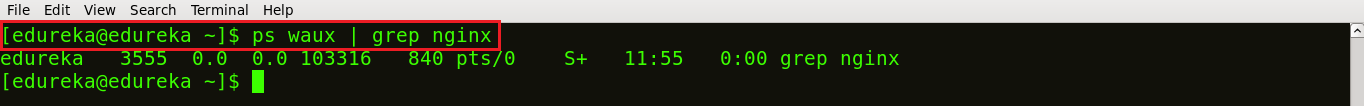
**ansible-playbook <name of your file>.yml**



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.