

Ansible Made Easy: Step-by-Step IT Automation

  What we learn in this Blog  

Table Content

1.What is Ansible ??  

2.Configuration Management 

3.Push Based vs Pull Based 

4.How to install Ansible

5.Host Inventory

6.YAML

7.Playbooks

8.Hands on

9.Conclusion.

Unlocking the Power of Ansible: A Brief Guide 🚀

Hey there, tech adventurers! ✨

Ready to level up your IT game? Say hello to Ansible, the magical tool that makes managing your servers as easy as waving a wand. 🪄 Whether you're a sysadmin, developer, or just a curious soul, Ansible is here to sprinkle some automation fairy dust over your infrastructure. Let's dive into why Ansible is your new best friend! 🤖

What is Ansible?

Ansible is an open-source automation tool that simplifies IT orchestration, configuration management, and application deployment. It uses a simple, human-readable language (YAML) to define tasks, making automation accessible to everyone. 📄

Ansible is one among the DevOps Configuration management tools which is famous for its simplicity. It is an open-source software developed by Michael DeHaan and its ownership is on RedHat.

Ansible is an open-source IT configuration Management, Deployment & Orchestration tool.

This tool is very simple to use yet powerful enough to automate complex multi-tier IT application environments.

The main component of ansible are playbooks. configuration management and deployment.

just you want to specify what state you want the system to be in and ansible take care of it.

Ansible was written in python.

Key Features of Ansible

- Agentless: No need to install agents on managed nodes, reducing complexity and overhead. 🖥️🔗
- Versatile: Works seamlessly across diverse platforms including Linux, Windows, and cloud environments. 🌐
- Scalable: From managing a few servers to thousands, Ansible scales effortlessly. 📈

- Community-driven: Continuously evolving with contributions from a vibrant community of users and developers. 🍷

Configuration Management 🌟

Configuration Management: Automate server setups, software installations, and system configurations.

It is a method through which we automate admin task.

Configuration Management tool turns your code into Infrastructure.

so your code would be testable, repeatable and Versionable.

Infrastructure refers to the composite of -

- software
- Network
- People
- Process

Push Based vs Pull Based

Tools like Puppet and chef are pull based

Agent on the server periodically checks for the configuration information from central server (master)

Ansible is Push Based

Central server pushes the configuration information on target servers. You control when the changes are made on the servers

(Ansible send notification to host servers to perform task)

What Ansible can do??

- Configuration Management
- App Deployment
- Continuous Delivery

Inventory File

Ansible's inventory hosts file is used to list and group your servers. its default location is `/etc/ansible/hosts`

Note: In inventory host file we can mention IP address or Hostname also

Sample of Inventory file

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

```
[databases]
192.168.1.20
192.168.1.21
db1.example.com
db2.example.com
```

Important Points about Ansible Inventory File

1. **Default Location:** The default location for the inventory file is `/etc/ansible/hosts`.
2. **Format:** The inventory file can be in INI or YAML format. The INI format is more common and straightforward.
3. **Grouping:** Hosts can be grouped into categories, such as `[webservers]` or `[databases]`, to apply tasks to multiple hosts simultaneously.
4. **Hostnames and IP Addresses:** You can list hosts by their hostnames or IP addresses.
5. **Variables:** You can define variables for groups or individual hosts within the inventory file. These variables can be used in playbooks to customize tasks.
6. **Dynamic Inventory:** Ansible supports dynamic inventory scripts, which can pull inventory data from external sources like cloud providers.
7. **Nested Groups:** Groups can be nested within other groups, allowing for more complex and hierarchical organization of hosts.
8. **Aliases:** Hosts can have aliases for easier reference in playbooks.
9. **Comments:** Lines starting with `#` are treated as comments and are ignored by Ansible.



Create Ubuntu system in AWS (free tier eligible) EC2 machine

1. Ansible system
2. Host system (it could 100 Machine according to your requirement)

Install Ansible in control node

Install packages

```
$ sudo apt-add-repository ppa:ansible/ansible
```

update the packages

```
$sudo apt update -y
```

Install ansible

```
$sudo apt install ansible
```

Now check version

```
$ansible --version
```

```
ubuntu@ip-172-31-39-94:~$ ansible --version
ansible [core 2.16.9]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/ubuntu/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.12.3 (main, Apr 10 2024, 05:33:47) [GCC 13.2.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
```

create folder

```
$mkdir keys
```

Now copy the .pem file from your local to remote Machine by using scp command

```
$scp -i "divya.pem" divya.pem ubuntu@ec2-13-127-5-106.ap-south-1.compute.amazonaws.com:/home/ubuntu/keys
```

verify that your .pem file (key_file) is present on keys folder which we create

```
ubuntu@ip-172-31-39-94:~$ mkdir keys
ubuntu@ip-172-31-39-94:~$ cd keys
ubuntu@ip-172-31-39-94:~/keys$ ls
divya.pem
```

Now go to your hosts file (Inventory file) and make some configuration like Host_IP and pass variable (all:vars) where you can specify your key file location

Note: Reverify that you configure all detail correctly

```
$sudo vim /etc/ansible/hosts
```

```
# It should live in /etc/ansible/hosts
#
# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or ip addresses
# - A hostname/ip can be a member of multiple groups

# Ex 1: Ungrouped hosts, specify before any group headers:
[servers]
server_1 ansible_host=15.207.87.227
server_2 ansible_host=13.126.31.55

[all:vars]
ansible_ssh_private_key_file=/home/ubuntu/keys/divya.pem

ansible_user=ubuntu
## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

# Ex 2: A collection of hosts belonging to the 'webservers' group:

## [webservers]
## alpha.example.org
```

Give permission to your key

```
$chmod 600 /home/ubuntu/keys/divya.pem
```

check your connectivity (**that your master node Attach to host node or not**)

```
$ansible all -m ping
```

```
ubuntu@ip-172-31-39-94:~$ sudo vim /etc/ansible/hosts
ubuntu@ip-172-31-39-94:~$ ansible all -m ping
The authenticity of host '13.126.31.55 (13.126.31.55)' can't be established.
ED25519 key fingerprint is SHA256:B5CfvLdLHh9FETcYq8E0oxT+sQBZ78R6UkJZv3zEKyg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? server_1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
yes
server_2 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
ubuntu@ip-172-31-39-94:~$
```

Heyyyyyy hurayyyyyy with this Ansible setup is completed

Ansible Ad Hoc Commands

Ansible ad hoc commands are a powerful feature that allows you to run simple commands or tasks directly on managed nodes without needing to write a full playbook. This can be useful for quick checks, debugging, or performing one-off tasks.

Basic Syntax for Ansible Ad Hoc Commands

The general syntax for running an ad hoc command is:

```
$ansible [Hostgroup,ip,hostname] -m [module] -a "[command]"
```

Where:

- [group,ip,Hostname] is the host or group of hosts on which you want to run the command.
- -m [module] specifies the Ansible module to use (e.g., command, shell, ping).
- -a "[command]" specifies the arguments for the module.



Common Examples of Ansible Ad Hoc Commands

1. **ansible all -m ping** = Use the ping module to check connectivity to the hosts
2. **ansible all -m shell -a "df -h"** =Use the shell module to execute a command. For example, to check the disk usage on all hosts
3. ****ansible all -m shell -a "uptime" =****To check the uptime of all hosts
4. ****ansible all -m copy -a "src=/path/to/file.txt dest=/tmp/file.txt" =****Use the copy module to copy a file from the control node to the managed nodes. For example, to copy file.txt to /tmp on all nodes
5. **ansible all -m apt -a "name=vim state=present"** =Use the apt module (for Debian-based systems) to install a package
6. ****ansible all -m service -a "name=nginx state=restarted" =****Use the service module to start, stop, or restart services. For example, to restart the nginx service on all hosts

7. **ansible all -m apt -a "name=vim state=absent" --become** =Use the yum or dnf module to remove software packages. For example, to uninstall vim
8. ****ansible all -m command -a "systemctl status nginx" =****Check Status of a Service
9. ****ansible all -m command -a "ip a" =****Get Network Interfaces
10. ****ansible all -m file -a "path=/tmp/newdir state=directory" --become =****Create a Directory





YAML: The Friendly Data Serialization Format

What is YAML?

YAML stands for "YAML Ain't Markup Language" (a recursive acronym) or "Yet Another Markup Language." It's a human-readable data serialization standard that's often used for configuration files and data exchange. Think of YAML as the cool, more readable cousin of JSON and XML.  

Practical Uses of YAML

Here are some common scenarios where YAML shines:

- **Configuration Files:** Applications like Docker and Kubernetes use YAML to define configurations and deployment setups.  
- **Data Serialization:** YAML is great for storing and exchanging data between applications or systems. 
- **Documentation:** Many documentation tools use YAML to define metadata and configurations. 

Ansible Playbooks

🌟 What is an Ansible Playbook?

An Ansible playbook is a YAML file that defines a set of tasks to be executed on remote systems. It's a way to automate tasks like software installation, configuration changes, and system management. Think of it as a recipe for your infrastructure! 📖🔧

📖 Key Components of a Playbook

1. **Play:** A play defines a set of tasks to be executed on a group of hosts. It's like a chapter in your book of automation. 📖
2. **Tasks:** Tasks are the individual actions within a play. Each task uses an Ansible module to perform an action. 🔧
3. **Modules:** Modules are the building blocks of tasks. They are predefined functions that Ansible uses to perform actions like installing packages, copying files, and more. 🔧
4. **Variables:** Variables store data that can be reused in playbooks. They make your playbooks flexible and dynamic. 📋
5. **Handlers:** Handlers are special tasks that only run when notified by other tasks. They are great for managing services and performing actions only when changes occur. 🔄

🔧 Writing Your First Ansible Playbook

Let's get started with a simple example. We'll create a playbook that installs Nginx webserver on a remote server and ensures the service is running.

Step 1: Define the Playbook

```
---  
  
- name: Setup NGINX Server  
  hosts: webservers  
  become: yes  
  tasks:  
    - name: Install NGINX  
      apt:  
        name: nginx  
        state: present  
        update_cache: yes  
  
    - name: Start and enable NGINX service  
      service:  
        name: nginx  
        state: started  
        enabled: yes  
  
    - name: Deploy custom HTML page  
      copy:  
        src: index.html  
        dest: /var/www/html/index.html  
        owner: www-data  
        group: www-data  
        mode: '0644'
```

Create a file named setup-nginx.yml and add the following content:

Step 2: Create the HTML File

Create a file named index.html in the same directory as your playbook with the content you want to serve. For example: **(you can change html code according to your requirement)**

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Visit Learn with Divya</title>

  <style>

    .button:hover {

      background-color: #4682b4;

    }

  </style>

</head>

<body>

  <div>

    <h1>Visit Learn with Divya!</h1>

    <p>Check out our blog for great content and tutorials.</p>

    <a href="https://learnwithdivya.hashnode.dev/" class="button"
target="_blank">Visit Now</a>

  </div>

</body>

</html>
```

Step 3: Run the Playbook

Execute the playbook using the ansible-playbook command:

```
$ansible-playbook setup-nginx.yml
```

before you access the application open all traffic rule in your Inbound rule

The screenshot shows the 'Inbound rules' configuration page in the AWS Management Console. It displays a table of existing rules and a new rule being added. The new rule is highlighted with a red box.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional	Info
sgr-07d3b108880867e0b	Custom TCP	TCP	8080	Custom		
sgr-0e290fe05d9fd2828	SSH	TCP	22	Custom		
-	All traffic	All	All	Anywhere...		

Buttons: Add rule, Cancel, Preview changes, Save rules

Warning: Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Access your application Now: **URL : http : //**
EC2-VM-Public-IP

Test Results:

Connecting host node success

```
ubuntu@master:~$ ansible all -m ping
The authenticity of host '13.233.10.164 (13.233.10.164)' can't be established.
ED25519 key fingerprint is SHA256:71eIjeeYAK6qOjtx0ZmQH3Q+ri021n0SHBDHHmRIJ/w.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
server_1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Ansible playbook works: Task completed

```
ubuntu@master:~$ ansible-playbook setup-nginx.yml

PLAY [Setup NGINX Server] *****

TASK [Gathering Facts] *****
ok: [server_1]

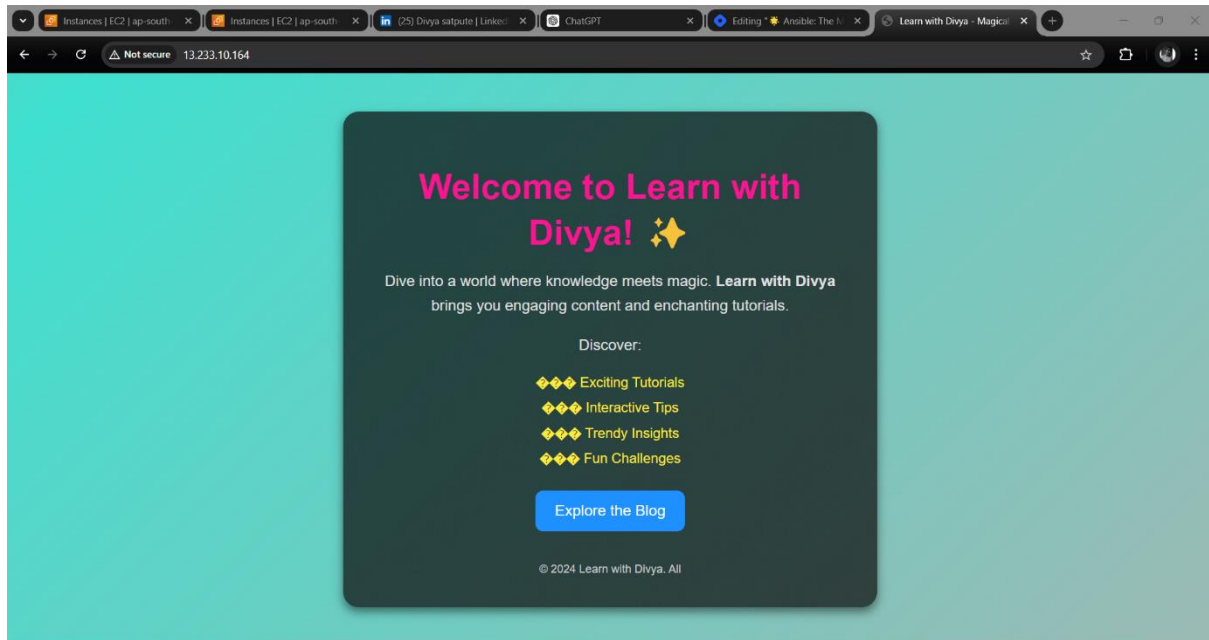
TASK [Install NGINX] *****
changed: [server_1]

TASK [Start and enable NGINX service] *****
ok: [server_1]

TASK [Deploy custom HTML page] *****
changed: [server_1]

PLAY RECAP *****
server_1 : ok=4 changed=2 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

Access your application paste host node IP address on browser: website Deploy



That's a wrap on our magical journey through Ansible! 🚀 ✨ From the basics to advanced features, Ansible is a powerful tool that can transform your IT operations. Dive in, explore, and let automation make your life easier and more fun! 🎉 ✨

Feel free to leave your thoughts, questions, or Ansible tips in the comments below. Happy automating!



Thank You for Choosing My Notes! 🌟📖

Hey Awesome Learner,

Thank you for picking up my notes! 🎉 I'm thrilled to be part of your learning journey. May these notes spark your curiosity and fuel your success. Keep reaching for the stars!

Feel free to reach out if you need anything. Happy learning! 🚀🌟