Ansible Tutorial

Introduction

Jumping into Configuration Management can be tough. Beginners often struggle with the complexity of setting up initial infrastructure, wrestling with cryptic syntax, and managing multiple configuration files, due to traditional tools.

Ansible facilitates this by using simple, human-readable YAML and no agents on target machines. This Ansible tutorial for beginners focuses on its central concepts, which make the automation easy and accessible. Ready to simplify your IT operations? Click here for the full **Ansible course syllabus**, and you can begin automating today!

Why Students or Freshers Learn Ansible

It is very useful to learn Ansible in securing modern, in-demand IT careers:

- **Automation is Key:** As a leading tool used for IT automation, Ansible would be the key skill for DevOps, Cloud, and Infrastructure across the world.

- **Easy to Learn:** It uses simple, human-readable YAML, and has no agents, so the learning curve is much gentler compared to other tools.
- **Various Roles:** Some of the highly paid job titles include DevOps Engineer, Cloud Engineer, and System Administrator, all of which strictly require knowledge in Ansible.
- **Integration with Cloud:** Smooth integration with major cloud platforms such as AWS, Azure, and Google Cloud makes it indispensable for multi-cloud environments.
- **Portfolio Booster:** Automating simple tasks or deploying an application using Ansible for a project adds high value to your resume.

Ready to secure your first automation job? Access the essential **Ansible Interview Questions and Answers** here!

Step-by-Step Ansible Tutorial for Beginners

Ansible is an open-source automation tool that empowers you to manage and configure hundreds of servers from a single control machine. Unlike traditional tools, it's agentless; so, you do not have to install any special software on the machines you're managing, relying on standard SSH (for Linux/Unix) or WinRM (for Windows). This makes it incredibly easy to set up and use.

This Ansible tutorial for beginners will guide you through the essential steps, from installation to running your first playbook.

Part 1. Installation and Initial Setup

The machine where you install and run Ansible is called the Control Node. You can use any Linux or macOS system. In this tutorial, Ubuntu/Debian are used for the Control Node examples.

Step 1.1: Installing Ansible on the Control Node

Ansible is in most of the distribution repositories.

For Debian/Ubuntu:

sudo apt update

sudo apt install software-properties-common

sudo add-apt-repository –yes –update ppa:ansible/ansible

sudo apt install ansible

For RHEL/CentOS/Fedora (using EPEL):

sudo yum install epel-release

sudo yum install ansible

Verification: After installation, check it by verifying the version:

ansible –version

Step 1.2: Set Up SSH Keys (Recommended)

Ansible connects to managed nodes using SSH. Using SSH key-pair authentication instead of passwords is the standard and most secure practice.

Generate a key pair (Ignore if you have it already)

ssh-keygen -t rsa -b 4096

(Accept the default file path and set a strong passphrase.)

Copy the public key:

This command copies your public key securely to the authorized keys file on the target machine, allowing passwordless SSH login from the targeted Managed Nodes.

ssh-copy-id username@target_ip_address

Part 2. The Inventory File: Defining Your Managed Nodes

The Inventory is a text file, in INI format, that describes which machines Ansible is allowed to control. By default Ansible will look for this file at /etc/ansible/hosts. To keep things simple and better organize ourselves, well call ours inventory.ini and place it in your project directory.

Step 2.1: Create the Inventory File

Create a file called *inventory.ini* and include your target server information in it.

[webservers]

web1 ansible_host=192.168.1.10 ansible_user=ubuntu

web2 ansible_host=192.168.1.11 ansible_user=ubuntu

[databases]

db1 ansible_host=192.168.1.20 ansible_user=centos

[all:vars]

ansible_python_interpreter=/usr/bin/python3

- **Groups:** The machines are organized into groups. E.g., [webservers], [databases].
- **Host Names:** web1, web2, and db1 are the aliases you will use in Ansible.
- **Connection Variables:**
    - **ansible_host:** The actual IP address or DNS name.
    - **ansible_user:** The user to SSH into the managed node with.

- **Global Variables ([all:vars]):** Here we define variables that will be applicable to all hosts. We define the path to the Python interpreter; this is frequently needed on newer systems.

Part 3. Running Your First Ad-Hoc Command

Ad-hoc commands are simple, one-line commands used for doing small things like checking connectivity, rebooting servers, or testing if a service is running. They're great for testing but certainly not for complex, repeatable configuration; that's what Playbooks are for.

Step 3.1: Connectivity Test (Ping)

Use the Ansible built-in ping module to make sure Ansible can connect, authenticate, and communicate with your managed nodes.

ansible all -i inventory.ini -m ping

- **all:** Targets all hosts defined in the inventory.
- **-i inventory.ini:** Provides the path to your inventory file.
- **-m ping:** The Ansible module to apply – this is the ping module.

Expected Output (Success):

web1 | SUCCESS => {

"changed": false,

"ping": "pong"

}

db1 | SUCCESS => {

“changed”: false,

“ping”: “pong”

}

 # … and so on

Step 3.2: Execute a Shell Command

Execute any shell command on the managed nodes using the shell or command module.

Check Disk Space on webservers group:

ansible webservers -i inventory.ini -m shell -a ‘df -h’

- **webservers:** Targets only the hosts in the webservers group.
- **-m shell:** shell module will be called.
- **-a ‘df -h’:** Passes the arguments (the shell command) to the module.

Part 4. Introduction to Playbooks

Playbooks constitute the heart of Ansible. These are well-structured YAML files that define a set of steps, normally called tasks that must be performed on a particular group of hosts. Playbooks offer features such as repeatability, version control, and the management of complexity.

Step 4.1: Create Your First Playbook

Create a file called site.yml. This playbook will install the Nginx web server and ensure the service is running.

—

```
– name: Configure Webservers and Install Nginx
  hosts: webservers
  become: yes  # Allows tasks to run with root privileges (sudo)
  tasks:
    – name: Ensure Nginx is installed (using apt module)
      ansible.builtin.apt:
        name: nginx
        state: present
        update_cache: yes
      when: ansible_os_family == “Debian” # Only run on Debian/Ubuntu systems
    – name: Start and enable Nginx service
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: yes
    – name: Deploy a simple index.html page
      ansible.builtin.copy:
```

content: “<h1>Welcome to Ansible Automation! Host: {{ ansible_hostname }}</h1>”

dest: /var/www/html/index.html

mode: ‘0644’

Key Playbook Components:

- **—:** Standard YAML document start marker.
- **– name::** A human-readable description for the playbook itself.
- **hosts: webservers:** Specifies which hosts (from your inventory) this playbook applies to.
- **become: yes:** Instructs Ansible to escalate privileges (like using sudo) for all tasks in this play.
- **tasks::** The list of actions to be executed.
- **– name::** A description for the individual task.
- **ansible.builtin.apt::** The module name. Ansible has hundreds of modules for nearly every task (e.g., apt, yum, service, copy).
- **when::** A conditional statement. ansible_os_family is a fact (system information gathered by Ansible) used to target specific operating systems.
- **{{ ansible_hostname }}:** An example of using an Ansible variable (specifically a gathered fact) inside the task.

Step 4.2 Run the Playbook

To run the playbook from your control node, use the ansible-playbook command:

ansible-playbook -i inventory.ini site.yml

Expected Output:

Ansible will display a recap of which tasks ran and what their change state was:

- **ok=X:** Tasks that ran successfully but made no changes (idempotency).
- **changed=Y:** Tasks that successfully ran and resulted in a change to the system.
- **unreachable=Z:** Hosts Ansible couldn't connect to.
- **failed=W:** Tasks that encountered an error.

The first run will likely show changed for the installation and service tasks. Subsequent runs will show ok because the desired state (Nginx installed and running) is already met. This is called idempotency, one of the core principles of Ansible.

Part 5. Key Higher-Level Concepts

5.1: Handlers (Notifying Services)

If the copy task changed the index.html file in your playbook, you may want to restart the Nginx service so that the changes take effect, but do not affect other tasks. You do this using Handlers. Handlers are tasks that only run when explicitly notified by a task.

Modify site.yml:

 # … (inside the tasks section)

– name: Deploy a simple index.html page

ansible.builtin.copy:

content: "<h1>Welcome to Ansible Automation! Host: {{ ansible_hostname }}</h1>"

dest: /var/www/html/index.html

mode: '0644'

notify: restart nginx # <— NEW: This notifies the handler

 # — NEW: Define handlers outside of the 'tasks' block —

```
handlers:
  – name: restart nginx
    ansible.builtin.service:
      name: nginx
      state: restarted
```

Now, the handler to restart nginx will only run if the copy task reports a changed status.

5.2: Vault (Security)

Ansible Vault encrypts sensitive data/passwords, API keys, or private keys that you want to keep in your playbook files or variables.

Create an Encrypted File:

ansible-vault create vars/secret.yml

You will be prompted to set a password. An editor will then open. Add sensitive variables here.

View or Edit:

ansible-vault edit vars/secret.yml

ansible-vault view vars/secret.yml

Run a Playbook Using the Encrypted File:

ansible-playbook -i inventory.ini site.yml –ask-vault-pass

You have installed Ansible, prepared an inventory, ran ad-hoc commands, and created your first idempotent playbook using modules, facts, and basic flow control such as notify and when. These are the core building blocks for automating complex infrastructure.

The next stage of truly understanding Ansible is the ability to apply these concepts to real-world situations. Ready to put your skills to the test and automate sophisticated infrastructure? Take our practical **Ansible Challenges and Solutions** to elevate your automation expertise!

Real Time Examples for Ansible Tutorial for Learners

The best way to solidify how Ansible works is to see how it solves common, real-world IT problems. The following examples combine several of the modules so far, and show some of the efficiency of Ansible.

Enforcing System Security and Compliance

- **Real-World Use:** Ensure all of your servers meet the organizational security standards, such as disabling the root user login or installing security patches.
- **Ansible Action:** Create a playbook to automatically install security updates, create a non-root administrative user, and modify the sshd_config file to disallow password-based root login.
- **Services Learned:** apt/yum module for package management, user module to create users, and the lineinfile module to manage particular lines in configuration files.

Deploying a Multi-Tier Application

- **Real-World Use:** Providing a full-featured web application environment, comprising a web server such as Nginx and a database server such as MySQL, on groups of machines.

- **Ansible Action:** Use one playbook that targets the webservers group to install Nginx and deploy code, and another section for databases group to install MySQL and create the required database schema.
- **Services Learned:** How to utilize the template module to create configuration files from Jinja2 templates; how to use the mysql_db module for database management. Multi-group orchestration.

Service Health Management and Monitoring

- **Real-World Use:** Verifying that a critical service, such as a caching server like Redis, is running and configured correctly on every production server.
- **Ansible Action:** A playbook, upon invocation of the service module, checks whether the service is running and, in case a config change causes a service restart, a handler triggers and informs a monitoring system via a webhook or API call.
- **Services Learned:** how to master the service module, invoke Handlers to perform controlled restarts, and utilize the uri module to integrate with other monitoring tools.

Ready to implement these concepts yourself and create your automated infrastructure? Learn about our selection of **Ansible project ideas** that will help you build an outstanding professional portfolio!

FAQs About Ansible Tutorial for Beginners

1.What is Ansible used for?

Ansible is an IT automation tool mainly utilized to perform configuration management, deploy applications, provision cloud resources, and orchestrate complex workflows across many servers.

2.Is Ansible using YAML or JSON?

Basically, Ansible uses YAML, which stands for YAML Ain't Markup Language, to create its playbooks, which define what automation to execute. It uses YAML because it is designed to be human-readable and expressive.

3.Is Ansible easy to learn?

Yes, Ansible is generally easy to learn for beginners because of its agentless architecture that makes it easier to set up, and it uses simple, human-readable YAML syntax in its Playbooks.

4.Is Ansible push or pull?

Ansible is by and large a push style because the control node connects to and executes commands on the managed nodes through SSH or WinRM. It also comes with an option for pull-based configurations using ansible-pull.

5.Do DevOps use Ansible?

Yes, DevOps teams heavily rely on Ansible for automating and streamlining tasks such as continuous configuration, infrastructure provisioning, and application deployment within CI/CD pipelines.

6.Which protocol does Ansible use?

Ansible uses standard protocols. On Linux/Unix systems, it uses SSH for connection and execution. And for Windows, it is WinRM.

7.Is Python necessary for Ansible?

Yes, it requires Python. The core of Ansible is written in Python, and though the Control Node requires Python, most of the Managed Nodes require only Python to be installed, version 2 or 3, to run the modules.

8.Which is faster, JSON or YAML?

In general, JSON is faster to parse and process since its structure is less complex than that of YAML. However, for human readability, Ansible's Playbooks use YAML.

9.Is Terraform or Ansible better?

Neither is strictly "better." Both of these are complementary. Terraform is better at Infrastructure Provisioning (Day 0), creating the cloud resources. Ansible is better at Configuration Management (Day 1), configuring software on those resources.

10.Is Ansible relevant in 2025?

Yes, Ansible remains very relevant in the year 2025. It always has a ranking as the top-in-demand IT automation skill by employers, especially for jobs that concern DevOps, cloud engineering, and configuration management. Explore **Ansible Salary for DevOps Professionals**.

Conclusion

You have successfully finished the basic steps of Ansible, from installation to the running of your first playbook. You would now have an understanding of how its agentless, YAML-based approach simplifies configuration management and orchestration. Keep in mind that mastering Ansible will get you the best jobs in DevOps and Cloud Engineering. Practice is what really unlocks automation. Are you ready to go beyond the basics and master complex enterprise automation workflows? Enroll in our comprehensive **Ansible course in Chennai** today and transform your infrastructure management skills!