Ansible Tutorial

Published On: July 30, 2024

Ansible Tutorial

Cloud provisioning, application deployment, intra-service orchestration, and other IT technologies can be automated using Ansible, an open-source IT engine. Begin your learning journey as we provide here the basic and advanced Ansible concepts in this Ansible tutorial.

Download Ansible Tutorial PDF

Introduction to Ansible

One of the best tools used in DevOps culture is Ansible. Since it doesn't require any agents or specialized security infrastructure, it may be easily deployed. The following topics are covered in this Ansible tutorial:

- Overview of Ansible
- Understanding if YAML
- Understanding of Ad hoc commands
- Understanding of Playbooks
- Understanding the Roles
- Understanding of Variables
- Execution and Troubleshooting with Ansible

Ansible Interview Questions

Overview of Ansible

Ansible provides open-source automation that reduces complexity and works everywhere. You can automate almost any task with Ansible. Here are a few typical Ansible usage cases:

- Reduce duplication and streamline processes
- Control and uphold the configuration of the system
- Constantly implement sophisticated software
- Implement rolling updates with zero downtime.

Ansible automates your tasks via playbooks—simple, human-readable scripts. In your playbook, you specify the desired state of a remote or local system. Ansible maintains the system in that configuration.

Principles of Ansible

Ansible is an automation technology built on the following standards:

Agent-less architecture: It minimizes maintenance costs by avoiding the need to add more software to the IT infrastructure.

Simplicity: Automation playbooks have code that reads like documentation in simple YAML format. Moreover, Ansible is decentralized; it accesses remote computers using SSH with pre-existing OS credentials.

Flexibility and scalability: Scale your automated systems quickly and easily with a modular design that works with various network devices, cloud platforms, and operating systems.

Balancing and Forecastability: Even if the playbook runs more than once, Ansible does not alter anything when the system is in the condition your playbook describes.

Ansible Salary

Understanding of YAML

Ansible uses YAML (Yet Another Markup Language) syntax to represent Ansible playbooks. It is easier for humans to read, understand, and write than alternative data formats like XML and JSON.

- Every YAML file for Ansible begins with a list. A list of key-value pairs, sometimes referred to as a “hash” or “dictionary,” makes up each item. Thus, we must learn how to create dictionaries and lists in YAML.
- All YAML files have the ability to start and finish with —.

ASP DOTNET Tutorial
Published On: July 31, 2024

ASP DOTNET Tutorial Microsoft created the web framework known as ASP.NET. It is employed in...

Artificial Intelligence Tutorial
Published On: July 30, 2024

Artificial Intelligence Tutorial Artificial intelligence (AI) is significant since it enhances many facets of society...

Appium Testing Tutorial
Published On: July 30, 2024

Appium Testing Tutorial Designed to make the UI automation of many app platforms easier, Appium...

This denotes the beginning and end of a document and is a component of the YAML format.

Example

A list's members are all lines that start at the same indentation level, "-" (a dash and a space):

—

 # A list of colors

– White

– Orange

– Red

– Black

—

Key-value Pair

The Key-Value pair is the data representation used in YAML. Additionally, the dictionary is explained in the value pair key.

Example: A Student Record

—

 # A student record

Bright:

name: *Bright*

roll no: *10*

class: *12th*

div: *A*

—

Representing List

Lists can also be expressed in YAML. Each entry in the list, or member, must be written on a new line with the same amount of indentation, beginning with "-" (- and space).

Example: Name of Indian Cities

—

#Name of Cities

Cities:

– Delhi

– Chennai

– Bangalore

– Hyderabad

—

List Inside Dictionaries

The value of a key is a list; thus, we may use the list that exists inside dictionaries.

Example: Employee Data

—

 # Employee Data

Raj:

name: Raj

emp no: 35

dept: Development

skills:

– Python

– DevOps

– Agile

—

List of Directories

We can create a list of directories as follows:

Example: Student Records

—

 # student records

– Bright:

name: Bright

roll no: 10

class: 12th

div: A

likes:

– Physics

– Chemistry

– Math

– Hari:

name: Hari

roll no: 11

class: 12th

div: A

likes:

– Biology

– English

—

YAML employs “|” to incorporate newlines when displaying multiple lines and “>” to obstruct newlines when displaying different lines.

We are able to read and alter lengthy lines as a result. The indentation will be disregarded in both scenarios. Additionally, boolean (True/false) values can be represented in YAML without regard to case.

Example: Student Result

```
---

#a student result

- Bright:

name: Bright

roll no: 10

class: 12th

div: A

likes:

- Physics

- Chemistry

- Math

  result:

    Physics: 70

   Chemistry: 45

Math: 85

Biology: 65

    English: 80

  passed: TRUE

  messageIncludeNewLines: |

    Congratulation!!

    You passed with 79%

  messageExcludeNewLines: >

    Congratulation!!

    You passed with 79%

---
```

Download Ansible Syllabus PDF

Understanding of Ad hoc commands

Ad hoc commands are separate commands that can be used to carry out critical tasks. There is no need to carry out these commands later.

Due to their one-time use, these ad hoc commands are not utilized for deployment and configuration maintenance. Configuration management and deployment are handled using ansible-playbook.

For example,

You must restart each server in your business. The ad hoc commands from "/usr/bin/ansible" will be used for this.

Parallelism and Shell Commands

Reboot the server of your business in 12 parallel forks at once. We must configure SSHagent for a connection to do this.

$ ssh-agent bash

$ ssh-add ~/.ssh/id_rsa

To do a 12-part parallel reboot for every server in your organization, 'abc' -

$ Ansible abc -a "/sbin/reboot" -f 12

Ansible will execute the above ad hoc commands from the current user account by default. To modify this behavior, you must pass the username in Ad-hoc instructions in the following manner:

$ Ansible abc -a "/sbin/reboot" -f 12 -u username

File Transfer

Ad-hoc commands can be used to SCP (Secure Copy Protocol) numerous files in parallel across different workstations.

Transferring a file to numerous computers or servers

$ Ansible abc -m copy -a "src = /etc/yum.conf dest =

/tmp/yum.conf”

New directory creation

$ Ansible abc -m file -a “dest = /path/user1/new mode = 777 owner = user1 group = user1 state = directory”

Deleting the entire directory and files

$ Ansible abc -m file -a “dest = /path/user1/new state = absent”

Managing Packages

You can use the ad-hoc commands for apt and yum. The following yum commands are ad hoc.

The following command does not update the Yum package; it checks to see if it is installed.

$ Ansible abc -m yum -a “name = demo-tomcat-1 state = present”

To verify if the package is not installed, run the following command.

$ Ansible abc -m yum -a “name = demo-tomcat-1 state = absent”

The command that follows verifies that the package’s most recent version is installed.

$ Ansible abc -m yum -a “name = demo-tomcat-1 state = latest”

Gathering Facts

Playbooks can employ facts to implement conditional statements.

The following ad-hoc command allows you to find the ad hoc information for all your facts:

$ Ansible all -m setup

Ansible Project Ideas

Understanding of Playbooks

The foundation of every Ansible use case is playbooks.

These are the files containing the Ansible code. YAML is the writing format used for playbooks.

One of Ansible's main functions is playbooks, which instruct Ansible on what to do. They function similarly to an Ansible to-do list with a list of tasks.

Playbooks include the steps that the user wants to carry out on a certain computer. Playbooks will run sequentially.

Playbook Structure

A playbook consists of one or more plays collected together. Plays are used to organize playbooks. A playbook can contain more than one play. A play's purpose is to map a collection of instructions that are defined against a specific host.

Since YAML is a strictly typed language, special attention must be given when creating YAML files. While there are other YAML editors available, we will stick with a basic editor like Notepad++.

Simply launch Notepad++, copy and paste the YAML below, and select YAML as the language (Language → YAML).

Generally, a YAML begins with — three hyphens.

Create Playbook

First, let's write a sample YAML file. We will go over each section that is written in a file in YAML format.

—

name: install and configure DB

hosts: testServer

become: yes

vars:

oracle_db_port_value : 1521

tasks:

-name: Install the Oracle DB

yum: <code to install the DB>

-name: Ensure the installed service is enabled and running

service:

name: <your service name>

We are attempting to go over the fundamental syntax of a playbook in the sample playbook up above. Save the above text as test.yml in a file. A little caution must be used when developing YAML syntax, as it must adhere to proper indentation.

YAML Tags

The descriptions of each tag are provided below.

Tag Name	**Description**
name	This tag gives the Ansible playbook's name.
hosts	The host field or tag must be filled up. It instructs Ansible on which hosts to execute the tasks on the list.
vars	The variables that you can utilize in your playbook can be defined using the Vars tag.
tasks	Every playbook ought to include assignments or a list of things that need to be done. A task list is a set of things that need to be done. The task name is contained in a task field.

Understanding the Roles

For completely independent or interdependent collections of variables, tasks, files, templates, and modules, roles offer a framework.

- The main method in Ansible for dividing a playbook

into several files is the role. This facilitates the rewriting of intricate playbooks and makes them easier to write.
- By applying the playbook breaking technique, you can logically separate the playbook into reusable sections.
- Although they are little pieces of functionality, roles must be used within playbooks.
- A role cannot be carried out directly. The host to whom a role will apply is not specified explicitly in roles.
- The link between the hosts in your inventory file and the roles that need to be assigned to them is provided by top-level playbooks.

Creating a New Role

To establish a new role, you must have the directory structure for roles.

Role Structure

The file system systematically arranges roles. Although the default structure can be modified, let's stay with it for the time being.

Every role is a self-contained directory tree. The directory name seen in the /roles directory is the role name.

$ ansible-galaxy -h

Usage

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

Role Options

- -h, –help: Display this help message and Exit.
- -v, –verbose: Verbose mode (-vvv to enable connection debugging, -vvvv for more)
- -version: Display the version number of the program and abort.

Creating a Role Directory

The role directories will be created as follows:

$ ansible-galaxy init vivekrole

ERROR! The API server (https://galaxy.ansible.com/api/) is not responding, please try again later.

$ ansible-galaxy init –force –offline vivekrole

– vivekrole was created successfully

$ tree vivekrole/

vivekrole/

├── defaults

│ └── main.yml

├── files ├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── README.md ├── tasks

│ └── main.yml

├── templates ├── tests │ ├── inventory

│ └── test.yml

└── vars

└── main.yml

8 directories, 8 files

Utilizing Roles in Playbook

This is the playbook's code that we wrote for demonstration. The playbook vivek_orchestrate.yml contains this code. The hosts have been defined as follows: the two responsibilities, install-tomcat and start-tomcat, and invoked tomcat-node.

The problem statement is that we need to use Ansible to deploy a war on a machine.

—

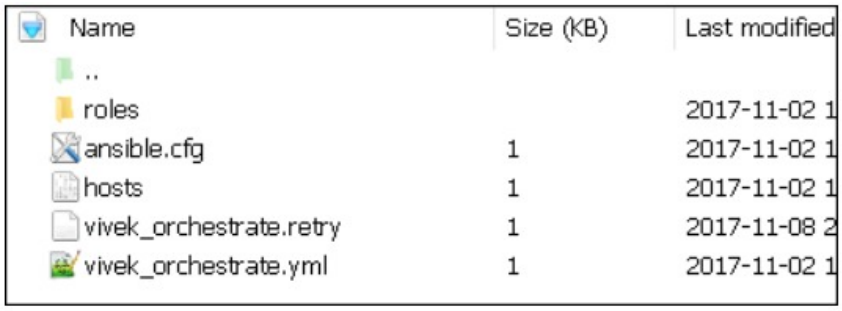
– hosts: tomcat-node

roles:

– *{role: install-tomcat}*

– *{role: start-tomcat}*

Contents of the directory structure that our playbook is operating from.



$ ls

ansible.cfg hosts roles vivek_orchestrate.retry vivek_orchestrate.yml

Every directory has a tasks directory with a main.yml file in it. The contents of install-tomcat's main.yml are:

—

#Install vivek artifacts

–

block:

– name: Install Tomcat artifacts

action: >

yum name = "demo-tomcat-1" state = present

register: Output

always:

– debug:

msg:

– "Install Tomcat artifacts task ended with message:

{{Output}}”

– “Installed Tomcat artifacts – {{Output.changed}}”

The following are the contents of Start Tomcat’s main.yml:

#Start Tomcat

–

block:

– name: *Start Tomcat*

command: *<path of tomcat>/bin/startup.sh”*

register: *output*

become: *true*

always:

– debug:

msg:

– “Start Tomcat task ended with message: *{{output}}”*

– “Tomcat started – {{output.changed}}”

Breaking a Playbook into Role

Should the roles not be needed, the contents of the corresponding role’s main.yml can be duplicated within the playbook.yml file. But roles were developed in order to have modularity.

A logical entity can be transferred to a role if it can be used again as a reusable function.

Example

-vvv option for verbose output – verbose output

$ cd vivek-playbook/

This command launches the playbook.

$ sudo ansible-playbook -i hosts vivek_orchestrate.yml – vvv

———————————————————

———————————————————————

Ansible Training

Understanding of Variables

Playbook variables work in a very similar way to variables used in any programming language. It facilitates using variables, allowing you to give them values and use them throughout the playbook.

The variables can have conditions applied to their values, and the script can use them appropriately.

Example

– hosts : <your hosts>

vars:

tomcat_port : 8080

You can utilize the variable tomcat_port, which has the value 8080 assigned to it in the example above, anywhere in your playbook. The code from one of the roles, *install-tomcat*, is as follows.

block:

– name: Install Tomcat artifacts

action: >

yum name = "demo-tomcat-1" state = present

register: Output

always:

– debug:

msg:

– "Install Tomcat artifacts task ended with message: {{Output}}"

– "Installed Tomcat artifacts – {{Output.changed}}"

The variable utilized in this case is the output.

Let's go over each keyword that is utilized in the code above.

Keyword	Description
block	Ansible syntax to execute a given block.
name	The block's pertinent name; this is used for logging and aids in debugging to determine whether or not every block was run correctly.
action	The code that appears next to the action tag indicates what needs to be done. Again, the action is a YAML keyword that is utilized in Ansible.
register	The action's output is registered using the register keyword, and the action's output is stored in the variable named output.
always	This is another Ansible keyword that indicates the following will always be carried out.
msg	It shows the message.

Usage of Variable: { {output} }

This will retrieve the output variable's value. It will print the output variable's value as well, just like it does in the message tab.

The variable's subproperties are also available for use. It is an example of determining whether the output has changed by checking {{Output.changed}} and using it appropriately.

Exception Handling in Playbooks

Any programming language's exception handling mechanism is the same in Ansible.

Example

tasks:

– *name*: *Name of the task to be executed*

block:

– *debug*: *msg = 'Just a debug message , relevant for logging'*

– *command*: *<the command to execute>*

rescue:

– *debug*: *msg = 'There was an exception.. '*

– *command*: *<Rescue mechanism for the above exception occurred)*

always:

– *debug*: *msg = "this will execute in all scenarios. Always will get logged"*

The syntax for handling exceptions is as follows:

- The terms **"rescue" and "always"** are unique to exception handling.
- Code (or anything to be executed on the Unix computer) is written in **blocks**.
- The execution reaches the rescue block and is carried out if the command written inside the block feature fails. Rescue won't be carried out if the command under the block feature is error-free.
- **Always** carried out in every situation.
- Thus, it is comparable to '***try, catch, and finally block'*** if we compare it to Java.
- Block in this context is comparable to the **try block**, where you write the code to be run; rescue is comparable to the **catch block**; and **finally**, **always,** is comparable to finally.

Loops

The ‘with_items’ syntax is being used to create a loop.

with_items: “{{output.stdout_lines}}” –> output.stdout_lines provides the output line by line, which we then loop over using Ansible’s with_items command.

Example

—

```
#Tsting

– hosts: tomcat-node

  tasks:

    – name: Install Apache

    shell: “ls *.war”

    register: output

    args:

      chdir: /opt/ansible/tomcat/demo/webapps

    – file:

      src: ‘/opt/ansible/tomcat/demo/webapps/{{ item }}’

      dest: ‘/users/demo/vivek/{{ item }}’

      state: link

    with_items: “{{output.stdout_lines}}”
```

Conditionals

When a particular step needs to be performed in response to a condition, conditionals are utilized.

—

```
#Tsting

– hosts: all

  vars:

    test1: “Hello Sam”
```

tasks:

– name: Testing Ansible variable

debug:

msg: "Equals"

when: test1 == "Hello Sam"

Since the test1 variable is equal in this instance, as specified by the when condition, equals will be printed. We can use logical AND and OR conditions.

To observe the results, simply change the value of the test1 variable from Hello Sam to, say, Hello World.

Ansible Execution

This is a crucial execution technique in which the playbook as a whole need not be executed; just one execution is needed.

Limit Execution by Tasks

We can assign distinct tags to various roles (which have tasks attached to them). As a result, only the designated role or task is executed, depending on the tags supplied by the executor.

Example: Add Tags

– {role: start-tomcat, tags: ['install']}}

Commands that help using tags:

ansible-playbook -i hosts <your yaml> –tags "install" -vvv

Only the start-tomcat role will be invoked when using the

aforementioned command. The given tag has case sensitivity. Make sure that the command is receiving an exact match.

Limit Execution by Hosts

There are two methods to accomplish the execution of particular actions on particular hosts. One specifies the hosts for a given role, determining which particular hosts the role should be executed on.

– hosts: <A>

environment: "{{your env}}"

pre_tasks:

– debug: msg = "Started deployment.

Current time is {{ansible_date_time.date}} {{ansible_date_time.time}} "

roles:

– {role: <your role>, tags: ['<respective tag>']}

post_tasks:

– debug: msg = "Completed deployment.

Current time is {{ansible_date_time.date}} {{ansible_date_time.time}}"

– hosts: <B>

pre_tasks:

– debug: msg = "started….

Current time is {{ansible_date_time.date}} {{ansible_date_time.time}} "

roles:

– {role: <your role>, tags: ['<respective tag>']}

post_tasks:

– debug: msg = "Completed the task..

Current time is {{ansible_date_time.date}}

{{ansible_date_time.time}}”

Running the Playbook

ansible-playbook user.yml –extra-vars “target = “<your host variable>”

Should {{ target }} be undefined, the playbook remains inactive. If necessary, a group from the hosts file can also be passed through. If the extra variables are not supplied, nothing bad will happen.

Playbook targeting a single host

$ ansible-playbook user.yml –extra-vars “target = <your hosts variable>” –listhosts

Conclusion

This Ansible tutorial will be helpful for you to understand the fundamentals. Learn comprehensively with hands-on exposure in our **Ansible training in Chennai.**

Share on your Social Media

Softlogic Academy

Softlogic Systems

KK Nagar [Corporate Office]

No.10, PT Rajan Salai, K.K. Nagar, Chennai – 600 078.
Landmark: Karnataka Bank Building
Phone: +91 86818 84318
Email: enquiry@softlogicsys.in
Map: Google Maps Link

OMR

No. E1-A10, RTS Food Street
92, Rajiv Gandhi Salai (OMR),
Navalur, Chennai - 600 130.

Landmark: Adj. to AGS Cinemas

Phone: +91 89256 88858

Email: info@softlogicsys.in

Map: Google Maps Link

Disclaimer

Privacy Policy

Terms and Conditions

Courses

- Python
- Software Testing
- Full Stack Developer
- Java
- Power BI
- Clinical SAS
- Data Science
- Embedded
- Cloud Computing
- Hardware and Networking
- VBA Macros
- Mobile App Development
- DevOps

Social Media Links

Review Sources

- Google
- Trustpilot
- Glassdoor
- Mouthshut
- Sulekha
- Justdial
- Ambitionbox
- Indeed
- Software Suggest
- Sitejabber