Ansible – Configuration Management

https://docs.ansible.com/

Topics

- Introduction to Ansible
- Setting up Ansible
- Introduction to YAML
- Inventory Files
- Playbooks
- Variables
- Conditionals
- Loops
- Roles

Control Node

Redhat or CentOS – $ sudo yum install ansible

Fedora – $ sudo dnf install ansible

Ubuntu – $ sudo apt-get install ansible

PIP – $ sudo pip install ansible

Install pip if not present

$ sudo yum install epel-release

$ sudo yum install python-pip

Install Ansible using pip Upgrade Ansible using pip

$ sudo pip install ansible $ sudo pip install --upgrade ansible

Install Specific Version of Ansible using pip

$ sudo pip install ansible==2.4

Ansible Inventory

Inventory contains the list of hosts to be managed/configured

Defult Inventory → /etc/ansible/hosts

server1.example.com

server2.example.com

[db]

server3.example.com

server4.example.com

[web]

server5.example.com

server6.example.com

Connection

Linux – SSH

Windows – Powershell Remoting

e.g:

web1 ansible_host=server1 ansible_connection=ssh ansible_user=root ansible_ssh_pass =xyz

web2 ansible_host=server2 ansible_connection=winrm ansible_user=administrator ansible_password=xyz

Ansible Playbook

- Defines plays containing tasks to be performed on managed hosts.
- File format is YAML
  - Play – Defines a set of activities (tasks) to be run on hosts
    - Task – An action to be performed on the host
      - Execute a command
      - Run a script
      - Install a package

- Shutdown/Restart

Playbook Format

-

name: Play 1

hosts: localhost

tasks:

- name: Execute command ‘date’

command: date

Run

$ ansible-playbook <playbook file name>

Ansible Configuration Files

/etc/ansible/ansible.cfg

[defaults]

[inventory]

[privilege_escalation]

[paramiko_connection]

[ssh_connection]

[persistent_connection]

[colors]

$ ANSIBLE_CONFIG=<path to custom cfg file>

Configuration file Precedence

0 /etc/ansible/ansible.cfg

1 ~/.ansible.cfg

2 ./ansible.cfg

3 ANSIBLE_CONFIG

Single configuration can be set anywhere in the hierarchy of config files:

$ export ANSIBLE_GATHERING=explicit

View Configuration

$ ansible-config list

$ ansible-config view

$ ansible-config dump

$ export ANSIBLE_GATHERING=explicit

$ ansible-config dump | grep GATHERING

DEFAULT_GATHERING(env: ANSIBLE_GATHERING) = explicit

Facts

ttps://docs.ansible.com/ansible/latest/user_guide/playbooks_vars_facts.html

- name: Gather facts

  gather_facts: yes no

  gather_facts: true false

  gather_facts: TRUE FALSE

  gather_facts: True False

Creating and Distributing SSH key

$ ssh-keygen

id_rsa id_rsa.pub

$ ssh-copy-id -i id_ras <user>@<server>

Privilege Escalation

- Become Super user (sudo) → **become: yes**
- Become Method – sudo (pfexec, doas, ksu, runas) → **become_method: <method-name>**
- Become another user → **become_user: <user-name>**

Privilege Escalation in Inventory File

Server1 ansible_ become=yes ansible_ become_user=<user-name>

Privilege Escalation in Configuration File

/etc/ansible/ansible.cfg

become = True

become_method = doas

become_user = <user-name>

Privilege Escalation using command Line

$ ansible-playbook --become --become-method=doas –become-user=<user> --ask-become-pass

Modules

https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html

$ ansible -m <module-name> <hosts>

e.g:

$ ansible -m ping all

$ ansible –a ‘cat /etc/hosts’ all

Check Mode or Dry Run

$ ansible-playbook playbook.yml –-check

Start at

$ ansible-playbook playbook.yml –-start-at-task <task-name>

Tags

$ ansible-playbook playbook.yml –-tags “install”

$ ansible-playbook playbook.yml –-skip-tags “install”

Variables

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

- Define variables in inventory/playbook file
- Use the variable in playbook in Jinja format I.e withing {{}}

inventory

web1 ansible_host=172.20.1.100

web2 ansible_host=172.20.1.101 dns_server=10.5.5.4

[web_servers]

web1

Web2

[web_servers:vars]

dns_server=10.5.5.3

Playbook

```
vars:
        dns_server: 10.5.5.5
tasks:
        - nsupdate:
                server: ‘{{ dns_server }}’
```

$ ansible-playbook playbook.yml –-extra-vars “dns_server = = 10.5.5.6”

Variable Precedence

1. Role Defaults
2. Group vars
3. Host vars
4. Host Facts
5. Play vars
6. Role vars
7. Include vars
8. Set Facts
9. Extra vars

Variable Scope

- Host
- Group
- Play
- Global/Playbbok

Register Variables

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#registering-variables

- Create variables from the output of an Ansible task with the task keyword **`register`**.
- Use registered variables in any later tasks in the play.

Magic Variables

https://docs.ansible.com/ansible/latest/user_guide/playbooks_vars_facts.html#information-about-ansible-magic-variables

- `hostvars`
- `groups`

- group_names
- inventory_hostname

e.g:

msg: ‘{{ hostvars['<hostname>'].ansible_host }}’

msg: ‘{{ hostvars['<hostname>'].ansible_facts.architecture }}’

msg: ‘{{ hostvars['<hostname>'].ansible_facts.devices }}’

msg: ‘{{ hostvars['<hostname>'].ansible_facts.mounts }}’

msg: ‘{{ hostvars['<hostname>'].ansible_facts.processor }}’

msg: ‘{{ hostvars['<hostname>']['ansible_facts']['processor'] }}’

Conditionals

- when

Operators

- Or
- and

Loops

- Loop keyword to iterate over a list

Blocks

- Groups tasks

Error Handling

- Rescue block for action to be taken in case of failure
- always block to be executed at the end irrespective of task status

Task failure

- any_errors_fatal: true
- max_fail_percentage: 30
- ignore_errors: yes # to be specified with task
- failed_when: <check> # at task level

Ansible Filters

https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

Templates

- Use Jinja 2 format for creating templates
- Use templates instead of copy module to copy the files to servers with interpolated values

Includes

- Use include_vars module to import variable from a file
- Create an inventory hierarchy as below:
  - **Inventory**
    - inventory
    - **host_vars**
      - <hostname alias>.yml
    - **group_vars**
      - <group name>.yml

```
$ ansible-inventory –i inventory/ -y
```

- **include_tasks** to include tasks from other .yml file

  e.g:

tasks: - name: Install MySQL Packages

- include_tasks: tasks/db.yml

- include_tasks: tasks/web.yml

Roles

- Code Reusablty
- Initialize a role using below command:

  ```
  $ ansible-galaxy init mysql
  ```

- Roles contains below folders
  - tasks
  - vars
  - defaults
  - handlers
  - templates
- Use roles in a playbook as below

  ```
  - name: Install and Configure MySQL
    hosts: db-server 1......db-server100
    roles:
      - mysql
  ```

- Look for ansible roles at https://galaxy.ansible.com/
- Use ansible-galaxy command to serach a role using CLI

  ```
  $ ansible-galaxy search <keyword>
  ```

- Install a role using below command:

  ```
  $ ansible-galaxy install <role-name>
  $ ansible-galaxy install geerlingguy.mysql –p ./roles
  ```

- List roles

  ```
  $ ansible-galaxy list
  ```

Strategy

- Linear (Default)
- Free
  - strategy: free
- Batch
  - serial: 3
  - forks value in .cfg file

What is CI/CD?

CI/CD automates much or all of the manual human intervention traditionally needed to get new code from a commit into production such as build, test, and deploy, as well as infrastructure provisioning. With a CI/CD pipeline, developers can make changes to code that are then automatically tested and pushed out for delivery and deployment.

What is Jenkins

Jenkins is one of the most popular automation tool used worldwide for continuous integration

and continuous delivery.

Why Jenkins?

When working on a project with different teams, developers often face issues with different teams using different CI tools, version management, and other tools. Setting up a CI/CD toolchain for each new project will lead to certain challenges like:

- Slower Releases
- Manual Builds
- Non-repeatable processes
- No Automations

Jenkins is the solution to those challenges. It provides:

- Automated builds
- Automated Tests
- Automated CI/CD pipelines
- Automated deployments
- Ability to install Jenkins locally
- Jenkins support and Plugins

Jenkins is well tested and provide several integrations with 1800+ plugins to support build, deployment and automation for the project.

CI/CD in simple words is a process to take a code, package it up and deploy it to a system that can be serverless, a VM, or a container. CI/CD can be broken down into 3 steps:

- CI – Continuous Integration
- CD – Continuous Delivery
- CD – Continuous Deployment

Key Processes of Continuous Integration

- Package up the code
- Test the code (run unit tests, integration tests, etc)
- Run security checks against the code

Think of the Continuous Integration process like a gift you're wrapping

- The gift comes in pieces
- You put the gift together (maybe a toy chest/box)
- The gift gets wrapped in wrapping paper
- You put it in the car and deliver it to the person.

The basic difference between Continuous Delivery and Continuous Deployment is that in Continuous Delivery to deploy the code after the CI process you have to manually trigger it via some button to deploy on the system whereas in Continuous Deployment this process is automatic.

Key Pieces of CD:

- Ensure you're authenticated to the system or wherever you're deploying
- Ensure that the code that's being deployed is working as expected once it's deployed

Installing Jenkins

https://www.jenkins.io/doc/book/installing/linux/

https://www.jenkins.io/doc/book/installing/linux/#fedora

Jenkins Plugins

Plugins are used in Jenkins to enhance Jenkins functionality and cater to user-specific needs. It allow us to connect one service to other services and work with other products.

Install Plugins

To install a new plugin in Jenkins

1) Go to Manage Jenkins -> Manager Plugins

2) Click Available and search for the desired plugin.

3) Select the desired plugin and Install.

Note: Few plugins may need a restart

To restart Jenkins

$ sudo systemctl restart jenkins

Jenkins Jobs

Different types of jobs that can be created in Jenkins:

Freestyle project

This is a central feature of Jenkins. It will build the project, combine SCM with the build system. It can also be used for things other than building applications.

Pipeline

This is used to create a pipeline

Multi-configuration project

This is great if you need a large number of Jenkins configurations if you need multiple environments like Dev/ UAT.

Folder

This creates containers and stores nested items. It is useful in grouping, creating a namespace, etc.

Organisation folder

Creates a multibranch project for all different subfolders that are available.

Multibranch Pipeline

It sets up pipeline projects for different repositories.

Jenkinsfile

Jenkinsfile is a text file that contains definitions. This could be templates or instructions. It tells pipelines what they should be doing and what services and plugins they should be interacting with.

Components of Jenkinsfile:

Pipeline – The task you are trying to accomplish

Build Agent –The place where you run your pipeline

Stages – Staging/Production/UAT

Steps –Work done in the pipeline

Build Agents

Build Agents are systems that run the processes throughout the pipelines. Build agents help in building codes, deploying, and running automated tests. It is a system that runs the entire workload.

Jenkins Security

Jenkins access control is split into two parts:

1. **Authentication** (users prove who they are) is done using a security realm. The security realm determines user identity and group memberships.
2. **Authorization** (users are permitted to do something) is done by an authorization strategy. This controls whether a user (directly or through group memberships) has a permission

Objectives

- Introduction to IaC
- Types of IaC Tools
- Why Terraform?
- HCL Basics
- Provision, Update & Destroy

Infrastructure as Code

Configuration Management

- Ansible
- SaltStack
- Puppet

Server Templting

- docker
- Packer
- Vagrant

Provisioning Tools

- Terraform
- CloudFormation

-

Why Terraform?

https://developer.hashicorp.com/terraform/intro

Automate creation, management and tracking of Infrastructure in a collaborative environment.

Terraform comes with rich set of plugins called **providers,** which enables it to connect to remote system/infrastructure.

https://registry.terraform.io

HCL

HCL is the syntactical specification of Terraform language, which is a rich language designed to be relatively easy for humans to read and write. The constructs in the Terraform language can also be expressed in JSON syntax.

HCL is also used by configuration languages in other applications, and in particular other HashiCorp products.

https://developer.hashicorp.com/terraform/language/syntax/configuration

https://github.com/hashicorp/hcl/blob/main/hclsyntax/spec.md

Block

```
resource "aws_instance" "example" {
  ami = "abc123"

  network_interface {
    # ...
  }
}
```

Terraform Commads

init	Prepare your working directory for other commands
plan	Show changes required by the current configuration
apply	Create or update infrastructure

To list all terraform commands, fetch the help manual

terraform -h

Filename	Purpose
main.tf	Main configuration file containing resource definition
variables.tf	Contains variable declarations
outputs.tf	Contains outputs from resources
provider.tf	Contains Provider definition

Variable Type

- **string**
- **number**
- **bool**
- **any**
- **list**
- **map**
- **set**
- **object**
- **tuple**

List

```
variable "prefix" {
default = ["Mr", "Mrs", "Sir"]
type = list   0       1       2
}
```

Map

```
variable file-content {
  type = map
 default = {
    "statement1" = "We love pets!"
     "statement2" = “We love animals!"
 }
}
```

Set

```
variable "prefix" {
 default = ["Mr", "Mrs", "Sir"]
 type = set(string)
}
```

Objects

```
variable "cindrella" {
 type = object({
        name = string
        color = string
        age = number
        food = list(string)
        favorite_pet = bool
  })
 default = {
        name = "cindrella"
        color = "brown"
        age = 7
```

```
    food = ["fish", "chicken", "turkey"]
    favorite_pet = true
  }
}
```

Tuples

```
variable kitty {
    type = tuple([string, number, bool])
    default = ["cat", 7, true]
}
```

Set Variable values at runtime

1. Don’t set default values in variables.tf file
2. Specify the values with `terraform apply` command as below:

   terraform apply -var "filename=/root/pets.txt" -var "content=We love Pets!" -var "prefix=Mrs" -var "separator=." -var "length=2"
3. *Set the vlues using environment variables:*

   $ export TF_VAR_filename="/root/pets.txt"

   $ export TF_VAR_content="We love pets!"

   $ export TF_VAR_prefix="Mrs"

   $ export TF_VAR_separator="."

   $ export TF_VAR_length="2"

   $ terraform apply
4. *Set the values in terraform.tfvars files:*

   filename = "/root/pets.txt"

   content = "We love pets!"

   prefix = "Mrs"

   separator = "."

   length = "2"

Now apply it as below:

$ terraform apply -var-file variables.tfvars

Variable files with below name are automatically loaded:

terraform.tfvars | terraform.tfvars.json

**.auto.tfvars | *.auto.tfvars.json*

Variable Precedence

*0 *.tf variable block*

1 Environment Variables

2 terraform.tfvars

*3 *.auto.tfvars (alphabetical order)*

4 -var or –var-file (command-line flags)

Lifecycle Rules

- create_before_destroy
- prevent_destroy
- ignore_changes

Data Source

```
data <prvider-resource> <resource-name> {
  # arguements
}
```

State

$ terraform state list

$ terraform state show <resorce>

Provisioner

https://developer.hashicorp.com/terraform/language/resources/provisioners/syntax

Taint

https://developer.hashicorp.com/terraform/cli/commands/taint

Debugging

Log Level

- Info
- Warning
- Error
- Debug
- Trace

Variables

- TF_LOG
- TF_LOG_PATH

Imports

$ terraform import <resource_type>.<resource-name> id

Modules

module "dev-webserver" {

source = "../aws-instance"

}

Advantages

- Simpler Configuration Files
- Lower Risk
- Re-Usability
- Standardized Configuration

https://registry.terraform.io/browse/modules

Functions

- Numeric Functions
- String Functions
- Collection Functions
- Type Conversion Functions

Numeric Functions

- Max
- min
- ceil
- floor

String Functions

- Split
- lower
- upper
- title
- substr
- join

Collection Functions

- Length

- index
- element
- contains

Map Functions

- Keys
- values
- lookup

Operators & Conditional Expressions

- Numeric Operators → **+, -, *, /**
- Equality Operator → **==, !=**
- Comparison Operator → **>, >=, <, <=**
- Logical Operator → **&&, ||, !**

condition ? true_val : false_val

Workspace

$ terraform workspace new <workspace-name>

$ terraform workspace list

Provision VM on KVM

https://computingforgeeks.com/how-to-provision-vms-on-kvm-with-terraform/