Ansible Real-time Project

Task 1

Create three (3) EC2 instances on AWS using Ansible loops

- 2 Instances with Ubuntu Distribution (Debian)

- 1 Instance with Amazon linux Distribution (Redhat)

-Hint: Use connection: local on Ansible Control node.

Task 2

Set up authentication between Ansible control node and newly created instances.

-Use ssh keys based authentication for Ubuntu instances

-Use password based authentication for amazon Linux instance.

Task 3

Automate the shutdown of Ubuntu Instances only using Ansible Conditionals

-Hint: Use when condition on ansible gather_facts

Before starting this project, we should have been aware of Ansible provisioning.

What is Provisioning in Ansible?

Provisioning in Ansible involves automating the setup and configuration of servers and infrastructure components. Using playbooks, Ansible executes tasks to deploy software, manage services, and ensure systems are configured as defined. This process enhances consistency, reduces manual effort, and supports scalable infrastructure management across diverse environments.

TASK-1

Control node is our laptop(local machine)

AWS instances are Manage Nodes

Prerequisites:

Install Ansible on local machine

What is Boto3?

Boto3 is the AWS SDK for Python, allowing developers to interact with AWS services programmatically. It simplifies integration by providing APIs for services like S3, EC2, and DynamoDB, offering both high-level abstractions and low-level access to AWS resources with features for automation, data processing, and serverless applications.

Install boto3

pip install boto3

What is AWS Collection?

In Ansible, "AWS collection" refers to a curated set of Ansible roles, modules, and plugins specifically designed to interact with Amazon Web Services (AWS). Collections in Ansible are a way to package and distribute content such as modules and plugins, making it easier to extend Ansible's capabilities beyond its core modules.

Install AWS Collection

ansible-galaxy collection install amazon.aws

Create IAM USER

1. Sign in to AWS Management Console:

- Go to the [AWS Management Console] (https://aws.amazon.com/console/).

- Sign in with your administrative credentials.

2. Navigate to the IAM Console:

- Search for "IAM" in the search bar and select "IAM" to open the IAM dashboard.

3. Create a New User:

- Click on "Users" in the sidebar.

- Click the "Add user" button.

4. Set User Details:

- Enter a username for the new user.

- Select the "Programmatic access" checkbox.

5. Attach Permissions:

- Choose "Attach existing policies directly".

- In the policy list, search for "AmazonEC2FullAccess".

- Select the "AmazonEC2FullAccess" policy.

6. Review and Create User:

- Review the user details and permissions.

- Click "Create user".

7. Download Credentials:

- On the success page, you will see the "Access key ID" and "Secret access key".

- Click "Download .csv" to save the credentials securely.

What is Ansible Vault?

Ansible Vault is a feature in Ansible that allows users to securely store and manage sensitive data, such as passwords and keys, by encrypting them. This ensures that sensitive information is protected within playbooks and can only be accessed by those with the appropriate decryption password.

In this project, the access key and secret access key are sensitive data, which are securely stored using Ansible Vault.

Use Visual Studio Code or Linux Terminal

1.Create a Folder Named "ansible_Project"

2. Create a Password for Vault

Command

```
openssl rand -base64 2048 > vault.pass
```

3. Create Vault File with AWS Credentials

Command

```
ansible-vault create group_vars/all/pass.yml --vault-password-file vault.pass
```

4. Add AWS Credentials:

Enter your AWS credentials in the following format:

Example:

ec2_access_key: YOUR_ACCESS_KEY

ec2_secret_access_key: YOUR_SECRET_ACCESS_KEY

5.If you want to edit ansible vault file use this command

ansible-vault edit group_vars/all/pass.yml --vault-password-file vault.pass

Now, create a file in the same folder named **ec2_ansible_playbook.yaml**

Next, go to the Ansible documentation and search for the module called ec2_instance module.

URL: Ansible EC2 Instance Module

Copy the code from there or use my GitHub code.

https://github.com/RoysonPais/Ansible/blob/main/Ansible_Real-time_Project/ec2_ansible_playbook.yaml

---

- hosts: localhost # Specifies that the tasks will run on the local machine

connection: local # Defines the connection type as local

tasks:

- name: Create EC2 instances # Name of the task

amazon.aws.ec2_instance: # Ansible module to manage EC2 instances

name: "{{ item.name }}" # Name of the EC2 instance, dynamically set from the loop items

key_name: "Ansible-aws-keypair" # Name of the SSH key pair to be used for the instances

instance_type: t2.micro # Type of EC2 instance to be created (t2.micro in this case)

security_group: default # Security group to be assigned to the instances

region: ap-northeast-1 # AWS region where the instances will be created

aws_access_key: "{{ec2_access_key}}" # AWS access key (provided as a variable)

aws_secret_key: "{{ec2_secret_access_key}}" # AWS secret key (provided as a variable)

network:

assign_public_ip: true # Assign a public IP address to the instances

image_id: "{{ item.image }}" # AMI ID for the instance, dynamically set from the loop items

tags:

environment: "{{ item.name }}" # Tag the instance with an environment tag using the name of the instance

loop: # Loop to create multiple instances

- { image: "ami-061a125c7c02edb39", name: "manage-node-1" } # First instance with specific AMI ID and name

- { image: "ami-01bef798938b7644d", name: "manage-node-2" } # Second instance with specific AMI ID and name

- { image: "ami-01bef798938b7644d", name: "manage-node-3" } # Third instance with specific AMI ID and name

To execute this playbook, you need to provide the vault file along with the command.

Command:

ansible-playbook ec2_ansible_playbook.yaml --vault-password-file vault.pass

What is idempotency in Ansible?

In Ansible, idempotency refers to the property of tasks and playbooks such that running them multiple times has the same effect as running them once. This is a key principle in configuration management and automation because it ensures that systems reach and maintain a desired state without unintended changes or side effects when the playbook is re-run.

In the above code, we used two variables for the name and the image because there are two Ubuntu instances sharing the same AMI ID. If we were to use only one variable for each instance, due to Ansible's idempotency, it would ignore the second instance since all the provided information is the same. Therefore, we used an additional variable for the name.

TASK-2

How to setup Authentication EC2 Instances?

ssh keys based authentication for Ubuntu instances

Using Public Key

ssh-copy-id -f "-o IdentityFile <PATH TO PEM FILE>" ubuntu@<INSTANCE-PUBLIC-IP>

ssh-copy-id: This is the command used to copy your public key to a remote machine.

-f: This flag forces the copying of keys, which can be useful if you have keys already set up and want to overwrite them.

"-o IdentityFile ": This option specifies the identity file (private key) to use for the connection. The -o flag passes this option to the underlying ssh command.

ubuntu@: This is the username (ubuntu) and the IP address of the remote server you want to access.

Password based authentication for amazon Linux instance.

Go to the file **/etc/ssh/sshd_config.d/60-cloudimg-settings.conf** (in manage node)

Update PasswordAuthentication yes

Restart SSH -> sudo systemctl restart ssh

Now set password

sudo passwd Ubuntu

In a control machine use this command

Ssh-copy-id Ubuntu@@<INSTANCE-PUBLIC-IP>

TASK3

Automate the shutdown of Ubuntu Instances only using Ansible Conditionals

-Hint: Use when condition on ansible gather facts

Create a file **inventory.ini**

Add public ip of the manage-nodes

Example:

[all]

ubuntu@ 100.0.0.6

ubuntu@ 100.0.0.4

ec2-user@ 203.0.113.6

Here, you can group the Ubuntu instances and easily shut down the instance, but the task clearly mentions using the "when" condition on Ansible gather facts.

Example:

[Manage]

ubuntu@ 100.0.0.6

ubuntu@ 100.0.0.4

Create a file **ec2_shutdown_playbook.yaml**

Direct github link

https://github.com/RoysonPais/Ansible/blob/main/Ansible_Real-time_Project/ec2_shutdown_playbook.yaml

---

This playbook will run on all hosts defined in the inventory

- hosts: all

Elevate privileges to become the root user or another specified user

become: true

tasks:

Define a task to shut down Ubuntu (or Debian-based) instances only

- name: Shutdown Ubuntu instances only

Use the command module to execute the shutdown command

ansible.builtin.command: /sbin/shutdown -t now

Add a condition to run this task only if the operating system family is Debian (which includes Ubuntu)

when: ansible_facts['os_family'] == "Debian"

To execute this playbook, you need to provide the vault file along with the command.

Command:

ansible-playbook ec2_stop_playbook.yaml --vault-password-file vault.pass

Gather Information

Ansible gather facts is a process in which Ansible collects detailed information about the remote hosts or machines it is managing.

Create a file called **ec2_gather_fact_playbook.yaml**

Direct github link

https://github.com/RoysonPais/Ansible/blob/main/Ansible_Real-time_Project/ec2_gather_fact_playbook.yaml

---

This playbook will run on all hosts defined in the inventory

- hosts: all

Elevate privileges to become the root user or another specified user

become: true

tasks:

Define a task to print all the gathered Ansible facts

- name: Print all the ansible gathered ansible_facts

Use the debug module to display the value of ansible_facts

ansible.builtin.debug:

var: ansible_facts

To execute this playbook, you need to provide the vault file along with the command.

Command:

ansible-playbook ec2_gather_fact_playbook.yaml --vault-password-file vault.pass

Some important topics in this project

Benefits of Idempotency in Ansible

1.Consistency: Ensures that the system configuration remains consistent across multiple runs.

2.Reliability: Reduces the risk of unintended changes or disruptions when playbooks are re-executed.

3.Error Recovery: Makes it easier to recover from errors by safely re-running playbooks.

4.Simplified Management: Simplifies the management of large-scale deployments by providing predictable and repeatable results.

Idempotency comparison between shell scripting and ansible

Shell Script Example

A shell script to ensure Apache is installed and running:

```
#!/bin/bash
# Bash script to check Apache installation and status

# Check if Apache is installed
if ! dpkg -l | grep -q apache2; then
   echo "Apache is not installed. Installing..."
   sudo apt-get update
   sudo apt-get install -y apache2
else
   echo "Apache is already installed."
fi
```

```
# Check if Apache is running
if ! systemctl is-active --quiet apache2; then
    echo "Apache is not running. Starting..."
    sudo systemctl start apache2
else
    echo "Apache is already running."
fi

# Ensure Apache starts on boot
sudo systemctl enable apache2
```

Explanation:

- The script checks if Apache (`apache2`) is installed by searching through the list of installed packages.
- If Apache is not installed, it installs it using `apt-get`.
- It then checks if Apache is running using `systemctl is-active`.
- If Apache is not running, it starts the service.
- Finally, it ensures Apache is enabled to start on boot with `systemctl enable`.

This script achieves idempotency by checking the current state before performing any actions, ensuring that running the script multiple times will not reinstall or restart Apache unnecessarily.

Ansible Playbook Example

An Ansible playbook to achieve the same:

```
---
- name: Ensure Apache is installed and running  # Playbook to ensure Apache installation and service management
  hosts: webservers  # Target hosts for the playbook execution

  tasks:
    - name: Ensure Apache is installed  # Task to ensure Apache installation
      apt:  # Ansible module to manage packages on Debian-based systems
        name: apache2  # Package name to be installed
        state: present  # Ensure the package is present on the system
        update_cache: yes  # Update the package cache before installing

    - name: Ensure Apache is started and enabled  # Task to start and enable Apache service
      service:  # Ansible module to manage services
        name: apache2  # Service name to be managed
        state: started  # Ensure the service is started
        enabled: yes  # Ensure the service is enabled to start on boot
```

Explanation:

- The `apt` module ensures that Apache (`apache2`) is installed. If it's already installed, Ansible does nothing.

- The `service` module ensures that Apache is started and enabled to start on boot. If Apache is already running and enabled, Ansible does nothing.

In this playbook, idempotency is built into the Ansible modules (`apt` and `service`). Running the playbook multiple times will not cause Apache to be reinstalled or restarted if it is already in the desired state.

Comparison

Shell Script:

- Manual Checks: Requires explicit checks to determine the current state before making changes.

- Idempotency: Achieved through conditional logic and checks.

- Complexity: More code and potential for errors.

Ansible:

- Built-in Idempotency: Modules inherently understand and manage the desired state.

- Simplicity: Less code, easier to read and maintain.

- Reliability: Less prone to errors due to the declarative nature.

Key Points about AWS Collection in Ansible:

1. Purpose: AWS collections provide a structured and organized way to manage Ansible modules and plugins for AWS services.

2. Content: Typically includes modules for interacting with various AWS services like EC2, S3, RDS, IAM, and more. It may also include plugins for inventory management, dynamic inventory, and connection plugins specific to AWS.

3. Installation: AWS collections can be installed using the ansible-galaxy command line tool or by including them in an Ansible playbook or role dependencies.

4. Usage: Once installed, AWS collections allow Ansible playbooks to provision, manage, and automate AWS resources directly through Ansible tasks, leveraging the capabilities of AWS without needing to write complex API calls or scripts.

5. Maintenance and Updates: AWS collections are maintained by the Ansible community and AWS itself, ensuring compatibility with new AWS services and updates to existing ones.