Ansible Quick Guide

Ansible is a tool to manage multiple remote systems from a single command center. In Ansible, the single command center is known as the control node and the remote systems to be managed are known as managed nodes. The following describes the 2 nodes:

Control node:

- The command center where Ansible is installed.
- Supported systems are Unix and Unix-like (Linux, BSD, macOS).
- Python and sshd are required.
- Remote systems to be managed are listed in a YAML or INI file called an inventory.
- Tasks to be executed are defined in a YAML file called a playbook.

Managed node:

- The remote systems to be managed.
- Supported systems are Unix/Unix-like, Windows, and Appliances (eg: Cisco, NetApp).
- Python and sshd are required for Unix/Unix-like.
- PowerShell and WinRM are required for Windows.
- In this tutorial we will use Ansible to manage multiple EC2 instances. For simplicity, we are going to provision EC2 instances in the AWS web console. Then we will configure one EC2 as the control node that will be managing multiple EC2 instances as managed nodes.

Prerequisites

For this tutorial we will need the following from AWS:

- An active AWS account.
- EC2 instances with Amazon Linux 2 as the OS.
- AWS Keys for SSH to access control node and managed nodes.
- Security group which allows SSH and HTTP.
- A decent editor such as Vim or Notepad++ to create the inventory and the playbook.

EC2 Instances provisioning

The following are the steps to provision EC2 instances with the AWS web console.

1. Go to AWS Console → EC2 → Launch Instances.
2. Select the Amazon Linux 2 AMI.
3. Select a key pair. If there are no available key pairs, please create one according to Amazon's instructions.
4. Allow SSH and HTTP.
5. Set Number of Instances to 4.
6. Click Launch Instance.

Ansible nodes and SSH keys

In this section we will gather the IP addresses of EC2 instances and set up the SSH keys.

1. Go to AWS Console → EC2 → Launch Instances.
2. Get the Public IPv4 addresses.

3. We will choose the first EC2 to be the Ansible control node and the rest to be the managed nodes:
   - control node:
     - 13.215.159.65
   - managed nodes:
     - 18.138.255.51
     - 13.229.198.36
     - 18.139.0.15
4. Login to the control node using our key pair. For me, it is kaptenjeffry.pem.

```
ssh -i kaptenjeffry.pem ec2-user@13.215.159.65
```

5. Open another terminal and copy the key pair to the control node

```
scp -i kaptenjeffry.pem kaptenjeffry.pem ec2-user@13.215.159.65:~/.ssh
```

6. Go back to the control node terminal. Try to log in from the control node to one of the managed nodes by using the key pair. This is to ensure the key pair is usable to access the managed nodes.

```
ssh -i .ssh/kaptenjeffry.pem ec2-user@18.138.255.51
```

7. Register the rest of the managed nodes as known hosts to the control nodes, in bulk:

```
ssh-keyscan -t ecdsa-sha2-nistp256 13.229.198.36 18.139.0.15 >> .ssh/known_hosts
```

Ansible Installation and Configuration

Install Ansible in the control node and create the inventory file.

1. In the control node, execute the following commands to install Ansible:

```
sudo dnf update -y
sudo dnf install ansible -y
ansible --version
```

2. Create a file named myinventory.ini. Insert the IP addresses that we identified earlier to be the managed nodes in the following format:

```
[mynginx]
red ansible_host=18.138.255.51
green ansible_host=13.229.198.36
blue ansible_host=18.139.0.15
```

   Where:
   - [mynginx] is the group name of the managed nodes,
   - red, green, and blue are the aliases of each managed node, and
   - ansible_host=x.x.x.x sets the IP Address each managed node.
   - myinventory.ini is a basic inventory file in a INI format. An inventory file could be either in INI or YAML format.

For more information on inventory see the Ansible docs.
https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html

Ansible modules and Ansible ad hoc commands

Ansible modules are scripts to do a specific task at managed nodes. For example, there are modules to check availability, copy files, install applications, and lots more.

To get the full list of modules, you can check the official Ansible modules page.

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

A quick way to use Ansible modules is with an ad hoc command. Ad hoc commands use the ansible command-line interface to execute modules at the managed nodes. The usage is as follows:

```
ansible <pattern> -m <module> -a "<module options>" -i <inventory>
```

Where:

```
<pattern> is the IP address, hostname, alias or group name,
-m module is name of the module to be used,
-a "<module options>" sets options for the module, and
-i <inventory> is the inventory of the managed nodes.
```

Change Username in Ansible Inventory for Ad-hoc Commands

When running Ansible ad-hoc commands, you can specify a different username than the default using:

Option 1: Specify Username in Command Line

Use the -u flag to change the username for the connection:

```
ansible all -m ping -u new_user
```

Example:

```
ansible webservers -m ping -u ec2-user
```

Option 2: Define Username in the Inventory File

You can define the username for specific hosts in the inventory file (hosts).

Example Inventory File (inventory.ini)

```
[webservers]
192.168.1.10 ansible_user=new_user
192.168.1.11 ansible_user=ec2-user
```

Then, run:

```
ansible webservers -m ping
```

*Ansible will use the username specified in the inventory.

Option 3: Use Ansible Configuration File

Set a global default username in ansible.cfg:

```
[defaults]
remote_user = new_user
```

*Then, Ansible will use this username automatically for all ad-hoc commands.

Configure Ansible Inventory to Use SSH Keys

Modify your inventory file (inventory.ini):

```
[webservers]
192.168.1.10 ansible_user=ec2-user ansible_ssh_private_key_file=~/.ssh/ansible_key.pem
192.168.1.11 ansible_user=ubuntu ansible_ssh_private_key_file=~/.ssh/ansible_key.pem
```

Set SSH Key as Default in ansible.cfg

To avoid specifying --private-key every time, configure ansible.cfg:

```
[defaults]
remote_user = ec2-user
private_key_file = ~/.ssh/ansible_key.pem
```

Ad hoc command examples

The following are some example of Ansible ad hoc commands:

ping checks SSH connectivity and Python interpreter at the managed node. To use the ping module against the mynginx group of servers (all 3 hosts: red, green, and blue), run:

```
ansible mynginx -m ping -i myinventory.ini
```

To copy a text file (/home/ec2-user/hello.txt in our test case) from the Control node to /tmp/ at all managed nodes in the mynginx group, run:

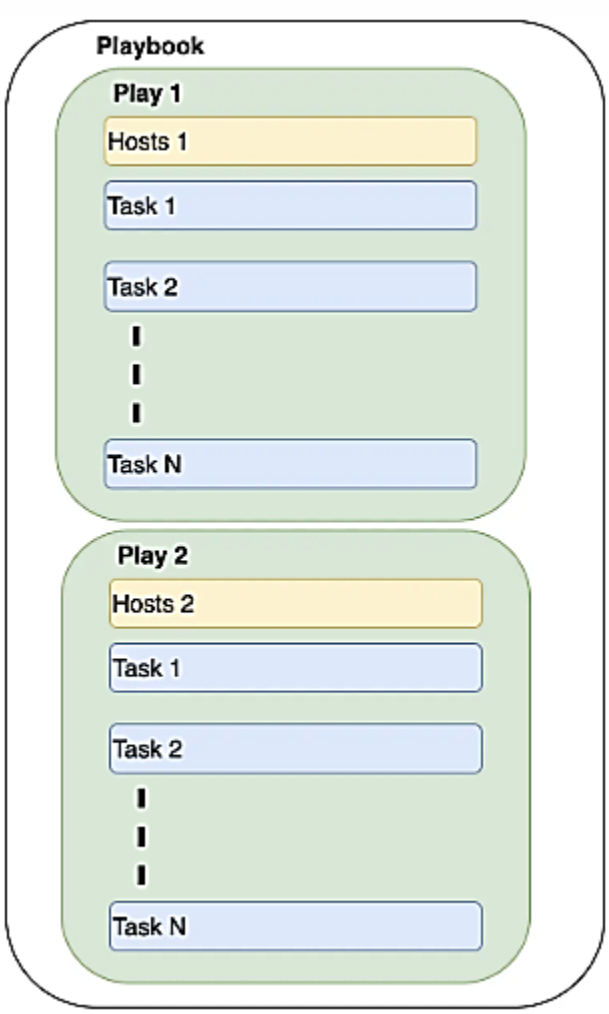
```
ansible mynginx -m copy \
-a 'src=/home/ec2-user/hello.txt dest=/tmp/hello.txt' \
-i myinventory.ini
```

shell executes a shell script at a managed node. To use module shell to execute uptime at all managed nodes in the mynginx group, run:

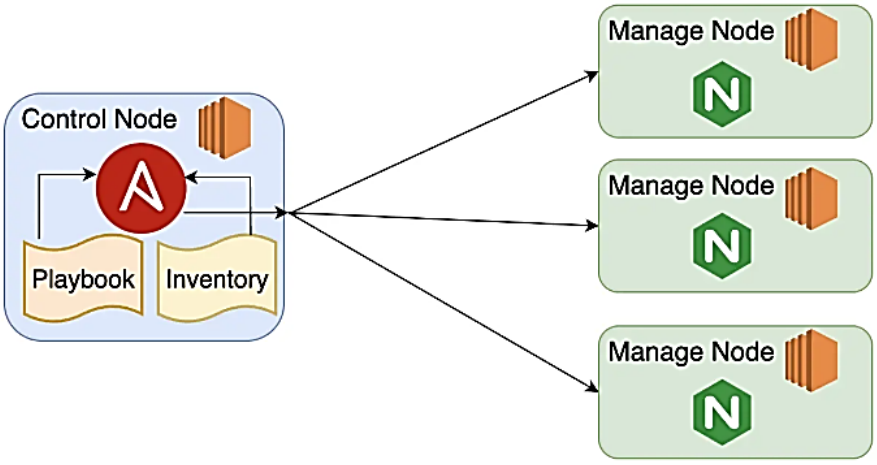
```
ansible mynginx -m shell -a 'uptime' -i myinventory.ini
```

Ansible playbooks

Ansible playbooks are configuration files in a YAML format that tell Ansible what to do. A playbook executes its assigned tasks sequentially from top to bottom. Tasks in playbooks are grouped by a block of instructions called a play. The following diagram shows the high level structure of a playbook:



Now we are going to use a playbook to install Nginx at our three managed nodes as depicted in the following diagram:



1. Create the following YAML file and name it nginx-playbook.yaml. This is a playbook with one play that will install and configure Nginx service at the managed node.

```
- name: Installing and Managing Nginx Server
  hosts: mynginx
  become: True
  vars:
    nginx_version: 1
    nginx_html: /usr/share/nginx/html
    user_home: /home/ec2-user
    index_html: index.html
  tasks:
    - name: Install the latest version of nginx
      command: amazon-linux-extras install nginx{{ nginx_version }}=latest -y

    - name: Start nginx service
      service:
        name: nginx
        state: started

    - name: Enable nginx service
      service:
        name: nginx
        enabled: yes
    - name: Copy index.html to managed nodes
      copy:
        src:  "{{ user_home }}/{{ index_html }}"
        dest: "{{ nginx_html }}"
```

Where:

name	(top most) is the name of this play,
hosts	specifies the managed nodes for this play,
become	says whether to use superuser privilege (sudo for Linux),
vars	defines variables for this play,
tasks	is the start of the task section,
name	(under task section) specifies the name of each task, and
name	(in a service section) specifies the name of a module.

2. Let's try to execute this playbook. Firstly, we need to create the source index.html to be copied to managed nodes.

```
echo 'Hello World!' > index.html
```

3. Execute ansible-playbook against our playbook. Just like the ad hoc command, we need to specify the inventory with the -i switch.

```
ansible-playbook nginx-playbook.yaml -i myinventory.ini
```

4. Now we can curl our managed nodes to check on the Nginx service and the custom index.html.

```
curl 18.138.255.51
curl 13.229.198.36
curl 18.139.0.15
```

Managing EC2 with Ansible Playbooks

Ansible can manage AWS EC2 instances using the AWS Ansible Collection and boto3 (AWS SDK for Python).

Step 1: Prerequisites

Before running an Ansible playbook for EC2, ensure you have:

AWS CLI Installed

```
sudo dnf install awscli -y
```

Configure AWS CLI:

```
aws configure
```

Enter:

- ✓ AWS Access Key
- ✓ AWS Secret Key
- ✓ Region (e.g., us-east-1)
- ✓ Output format (default: json)

Install Required Ansible Collections

```
ansible-galaxy collection install amazon.aws
```

Install boto3 and botocore

```
pip install boto3 botocore
```

Ensure Ansible Can Communicate with AWS Test connectivity:

```
aws ec2 describe-instances
```

Step 2: Create an Ansible Playbook to Manage EC2

Playbook to Launch an EC2 Instance

Create a file launch_ec2.yml:

```
- name: Launch an EC2 instance
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Launch EC2 Instance
      amazon.aws.ec2_instance:
```

```
    name: "Ansible-Managed-EC2"
    key_name: "my-key-pair"
    instance_type: "t2.micro"
    security_group: "default"
    image_id: "ami-0c55b159cbfafe1f0"  # Change based on your region
    region: "us-east-1"
    state: running
    wait: yes
   register: ec2_info

  - name: Display EC2 Instance Details
    debug:
     msg: "EC2 instance {{ ec2_info.instance_ids }} has been created."
```

Run the playbook:

```
ansible-playbook launch_ec2.yml
```

Playbook to Stop an EC2 Instance

Create stop_ec2.yml:

```
- name: Stop an EC2 instance
 hosts: localhost
 gather_facts: no
 tasks:
  - name: Stop the EC2 instance
    amazon.aws.ec2_instance:
     region: "us-east-1"
     instance_ids: ["i-xxxxxxxxxxxxxxxxx"]  # Replace with your instance ID
     state: stopped
```

Run it:

```
ansible-playbook stop_ec2.yml
```

Playbook to Terminate an EC2 Instance

Create terminate_ec2.yml:

```
- name: Terminate an EC2 instance
 hosts: localhost
 gather_facts: no
 tasks:
  - name: Terminate EC2 instance
    amazon.aws.ec2_instance:
     region: "us-east-1"
     instance_ids: ["i-xxxxxxxxxxxxxxxxx"]
     state: terminated
```

Run it:

```
ansible-playbook terminate_ec2.yml
```

Step 3: Automating with Roles

If you want a structured approach, create a role for EC2 management:

```
ansible-galaxy init roles/ec2_manager
```

Then structure it like this:

```
roles/ec2_manager/
 ├── tasks
│    ├── launch.yml
│    ├── stop.yml
│    ├── terminate.yml
│    ├── main.yml
 ├── defaults
│    ├── main.yml
└── meta
    ├── main.yml
```

Modify roles/ec2_manager/tasks/main.yml to include:

```
- import_tasks: launch.yml
- import_tasks: stop.yml
- import_tasks: terminate.yml
```

Then call it in ec2_playbook.yml:

```
- name: EC2 Management Playbook
  hosts: localhost
  roles:
    - ec2_manager
```

Run:

```
ansible-playbook ec2_playbook.yml
```