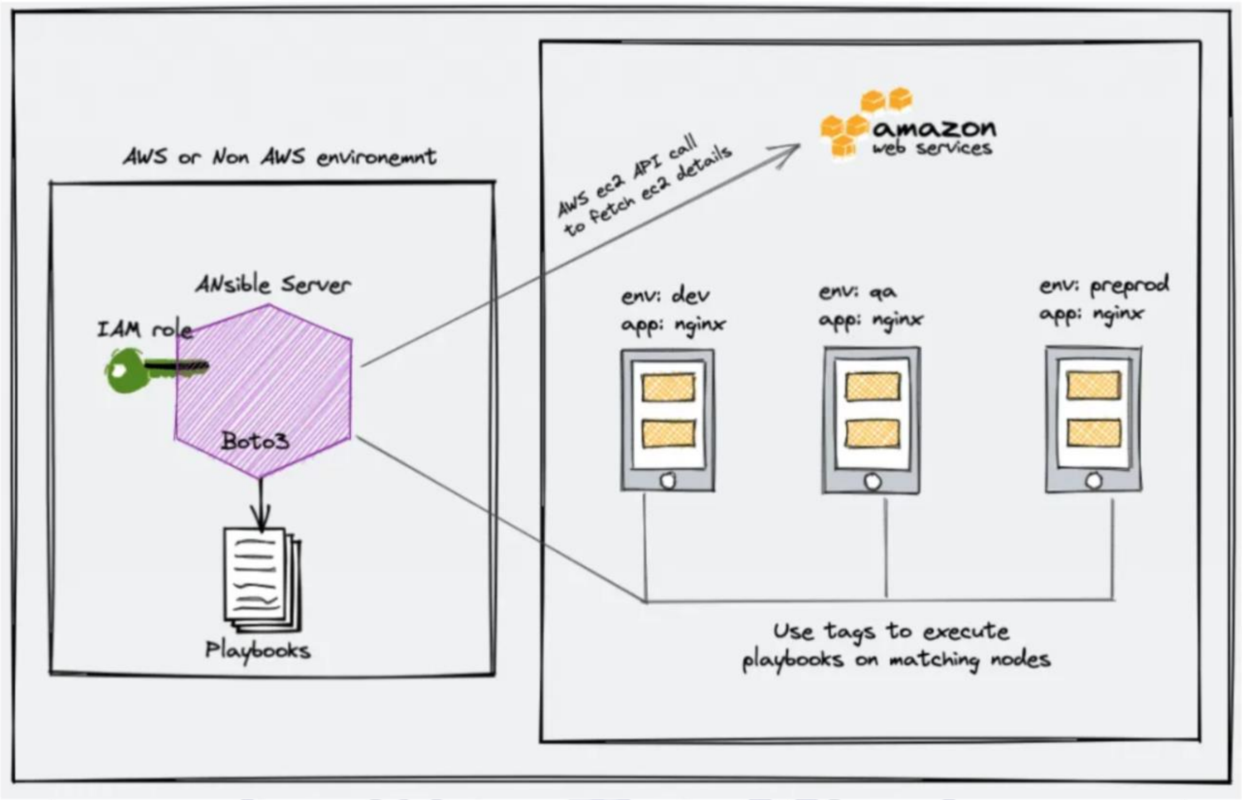
Ansible Dynamic Inventory

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Session - 24 Agenda:
1. Ansible Dynamic Inventory

Ansible Dynamic Inventory:
When you are using Ansible with any cloud like AWS, maintaining the inventory file will be a hectic task as AWS has frequently changed IPs, autoscaling instances, and much more.

However, there is an easy solution called ansible dynamic inventory. Dynamic inventory is an ansible plugin that makes an API call to AWS to get the instance information in the run time. It gives you the ec2 instance details dynamically to manage the AWS infrastructure.



In the beginning people were using the Dynamic inventory, in form of a Python file, later it became an Ansible plugin.

Let's learn how can we use dynamic inventory in our ansible control node which is running on RHEL 9. Here we are going to use our local machine which is running in VMWARE as the ansible control node to manage the nodes which are EC2 instances running on AWS.

Go to your ansible control node and verify the OS version.

```
$ cat /etc/redhat-release
```

To use dynamic inventory, we would require python boto packages (boto & boto3) which can be installed in our ansible server using pip command. First of all we have to install ansible-core, python-pip & python-devel packages in our machine which we want to use as ansible control node for our dynamic inventory.

```
$ sudo yum install ansible-core python-pip python-devel -y
```

Verify the installation by checking the ansible and python versions.

```
$ ansible --version
```

After that we can install boto & boto3 packages by running the pip install command.

Boto is the Amazon Web Services (AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services, such as EC2 and S3. Boto provides an easy to use, object-oriented API, as well as low-level access to AWS services.

```
$ pip install boto
```

Ansible AWS modules are written on top of boto3. Boto3 is the Python SDK for the AWS that allows users to manage AWS infrastructure: EC2, VPC, Security Groups, etc.

```
$ pip install boto3
```

By default, ansible-core doesn't provide the modules required for managing the nodes running on AWS, to managed AWS nodes we require amazon.aws collection which we can install from ansible galaxy.

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

Ansible dynamic inventory for AWS requires dynamic inventory scripts (ec2.py & ec2.ini) which we can easily download from GitHub or any website. We have already created a repository on GitHub which contains these files. Now, we can clone this repository in our machine.

```
$ git clone https://github.com/ansible22/aws.git
```

Now, move to this aws directory which you have cloned earlier.

```
$ cd aws/
```

List the contents there to verify the files.

```
$ ls -lh
```

Apply the execute permissions on both of these files.

```
$ chmod +x ec2.*
```

Go to AWS and launch an EC2 instance on AWS, also download the pem key file. Then, go to IAM service in AWS and create a IAM User with EC2 full access. After that go to user security credentials and generate access key and secret key for the user.

```
AKIA6KYVK7QF2BKWCBUCD2iA&3#5
ovLIABzjHz2ZpcEAMGoUh79pnwgOe+rC+Iz9aDCL
```

Then open the ec2.py python script file in vi editor and make the below mentioned changes and then save this file.

```
$ vim ec2.py
# from ansible.module_utils import ec2 as ec2_utils #line number 37
'elasticache': 'False',   #line number 64
```

After that open the ec2.ini file in vi editor and go to line 217 to add the access key which you have created earlier, then go to line number 218 to add the secret key and then save this file.

```
$ vim ec2.ini
AKIA6KYVK7QF2BKWCBUCD2iA&3#5              #line number 217
ovLIABzjHz2ZpcEAMGoUh79pnwgOe+rC+Iz9aDCL  #line number 218
```

Then open the ansible configuration file in the vi editor to add the ec2.py file path so use this file as the default inventory file in your ansible server.

```
$ sudo vim /etc/ansible/ansible.cfg
[defaults]
inventory = /home/vikasnehra/aws/ec2.py
```

Now, execute the ec2.py file, it will you show the facts output on the screen. It will also display the ip addresses of the nodes running on AWS which you can manage from your ansible server.

```
$ ./ec2.py
```

You can list the managed nodes by using the below command as well.

```
$ ansible all --list-hosts
```

Now, launch some more instances on AWS just to verify whether you are able to list them using above command as the managed nodes or not using ansible.

```
$ ansible all --list-hosts
```

Now, copy pem file to ansible server in the same aws folder where you have the inventory files and apply the below permissions on it.

```
$ sudo chmod 400 ansible.pem
$ sudo chown vikasnehra:vikasnehra ansible.pem
```

You can again open the ansible configuration file in the vi editor and define additional settings as well to push the tasks on the managed nodes.

```
$ sudo vim /etc/ansible/ansible.cfg
[defaults]
inventory = /home/vikasnehra/aws/ec2.py
remote_user = ec2-user
ask_pass = false
private_key_file = /home/vikasnehra/aws/ansible.pem
[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Check the connectivity with the managed nodes using ping ad-hoc command.

```
$ ansible -m ping all
```

Now, we can push the tasks using playbooks. Let’s create a playbook to install the VSFTP package on all the managed nodes.

```
$ vim playbook.yml
---
- name: VSFTPD package installation playbook
  hosts: all
  tasks:
    - name: Installing VSFTPD package
      dnf:
        name: vsftpd
        state: latest
...
```

Execute the playbook to install the VSFTP package on the managed nodes.

```
$ ansible-playbook playbook.yml
```

Verify the same using ansible ad-hoc command.

```
$ ansible all -m command -a 'rpm -qi vsftpd'
```

Thank You