Ansible Dynamic Inventory with AWS EC2 Plugin

Overview

Ansible can pull inventory information from dynamic sources by various dynamic inventory plugins.
The aws_ec2 plugin is a great way to manage AWS EC2 Linux instances without having to maintain a standard local inventory. This will allow for easier Linux automation, configuration management, and infrastructure as code of AWS EC2 instances.

Prerequisites

The below requirements are needed on the local controller node that executes this inventory.

- python3
- python-pip
- boto (pip3 install boto)
- boto3 (pip3 install boto3)
- botocore (pip3 install botocore)

Configure Dynamic Inventory

1. It is needed two users – one for interaction with AWS infrastructure another – for playbook implementation on ec2 instances.

Add user via aws console IAM – adduser
Permissions policies:
- read only ec2 instances – AmazonEc2ReadOnlyAccess
- read only rds instances - AmazonRDSReadOnlyAccess
- read only elastic services – AmazonElastiCacheReadOnlyAccess

There are several ways how to use this credantials:

- Export variable:

```
export AWS_ACCESS_KEY_ID=<YOUR ACCESS KEY ID>
export AWS_SECRET_ACCESS_KEY=<YOUR SECRET ACCESS KEY>
export AWS_DEFAULT_REGION=<YOUR AWS DEFAULT REGION>
```

- Specify AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY in the aws_ec2.yml

```
plugin: aws_ec2
aws_access_key: "AKIA4PVQ6OY4USYZ3MLP"
aws_secret_key: "IKhPhKL0s76b+gHii5CfF/WMKOPS6letgO8WFjfl"
```

- Using Boto3 library

Set up credentials (in e.g. ~/.aws/credentials):
#aws configure
or edit ~/.aws/credentials

```
[default]
aws_access_key_id = YOUR_KEY
aws_secret_access_key = YOUR_SECRET
```

Import the Boto 3 library
#python3
>>> import boto3

!!!Before crate instances in different regioni, check that you have the same key in every region where you are planning to install instances. It is needed for performing different plays in one playbook for instances in different regions, because in config file you can specify only one user and theirs credential (ssh key).**!!!**

Switch to the needed region and open the EC2 console and in the navigation pane under the NETWORK & SECURITY click “Key Pairs”. In the upper right corner of the page, click “Actions” button and choose “import key pair”. or import your key to the all regions via AWS CLI: https://aws.amazon.com/premiumsupport/knowledge-center/ec2-ssh-key-pair-regions/#:~:text=To%20use%20a%20single%20SSH,most%20recent%20AWS%20CLI%20version.

Specify user and key path in ansible.cfg
remote_user = ec2-user
private_key_file=<path to key.pem>
become = true
become_method: sudo
become_user: root

2. Get the Ansible Amazon AWS collection that includes a variety of Ansible content to help automate the management of AWS instances.

   #ansible-galaxy collection install amazon.aws

In the ansible working directory create ./ansible_plugins directory.
This is a handy way to have all Ansible aws_ec2 plugin configuration files in a single directory. If you have multiple AWS accounts you can have multiple Ansible AWS EC2 plugin configuration files within this directory.

Copy from installed amazon.aws collection
.ansible/collections/ansible_collections/amazon/aws/plugins/inventory/aws_ec2.py
.ansible/collections/ansible_collections/amazon/aws/plugins/inventory/__init__.py
to the working ansible directory or to the ./ansible_plugins directory.

3. Add to the ansible.cfg file:

[defaults]
inventory = ./ansible_plugins
enable_plugins = aws_ec2

4. Add ./ansible_plugins /aws_ec2.yml file

For multiple AWS accounts it is better create multiple Ansible AWS EC2 plugin configuration files that sould be naming with requirement - the suffix portion of the name must be: _aws_ec2.yml or _aws_ec2.yaml

For example:
production_useast_aws_ec2.yml
production_uswest_aws_ec2.yml
test_dev_aws_ec2.yml
production_aws_ec2.yml

Parameters in aws_ec2.yml|yaml	Description
plugin: aws_ec2	aws ec2 ansible dynamic inventory plugin
aws_access_key: <PUT IN YOUR AWS ACCESS KEY>	set aws_access_key and secret_key.
aws_secret_key: <PUT IN YOUR AWS SECRET KEY>	
regions:	set the regions.
- us-east-1	
strict: False	Ignores 403 errors rather than failing. By default if a 403 (Forbidden) error code is encountered this plugin will fail. You can set this option to False which will allow "#403 errors" to be gracefully skipped. If "True" this will make invalid entries a fatal error
filters:	Filter helps to make a selection of instances that match up some demands. While as "keyed_groups" use for collecting instances in separated groups for working with them.
• availability-zone - The Availability Zone of the instance. • group-id - The ID of the security group for the instance. EC2-Classic only. • group-name - The name of the security group for the instance. EC2-Classic only. • instance-id - The ID of the instance. • instance-state-name - The state of the instance (pending | running | shutting-down | terminated | stopping | stopped). • instance-type - The type of instance (for example, t2.micro). • instance.group-id - The ID of the security group for the instance. • instance.group-name - The name of the security group for the instance. • ip-address - The public IPv4 address of the instance. • tag:<key> - The key/value combination of a tag assigned to the resource. Use the tag key in the filter name and the tag value as the filter value. For example, to find all resources that have a tag with the key Owner and the value TeamA , specify tag:Owner for the filter name and TeamA for the filter value. • tag-key - The key of a tag assigned to the resource. Use this filter to find all resources that have a tag with a specific key, regardless of the tag value. All dev and QA hosts: tag:Environment: - dev - qa • vpc-id - The ID of the VPC that the instance is running in.	If you specify multiple filters, the filters are joined with an AND , and the request returns only results that match all of the specified filters. Filter names are case-sensitive. The filter values. Filter values are case-sensitive. If you specify multiple values for a filter, the values are joined with an OR , and the request returns all results that match any of the specified values.

keyed_groups:	Each aws ec2 instance has it own instance tags. Create a tag variable from those tags for ansible to use. If the ec2 tag Name had the value cygnusx1 the tag variable would be: tag_Name_cygnusx1.
- key: tags prefix: tag - key: architecture prefix: arch - key: tags.Applications separator: '' - key: instance_type prefix: aws_instance_type - key: placement.region prefix: aws_region - key: image_id prefix: aws_image - key: 'security_groups|json_query(""[].group_id"")' prefix: 'security_groups"'	keyed groups are from the aws url: https://docs.aws.amazon.com/cli/latest/reference/ec2/describe-instances.html#options
hostnames:	a list in order of precedence for hostname variables.
- ip-address - dns-name - tag:Name - private-ip-address	
compose:	use if you need to connect via the ec2 private ip address. This is needed for example in a corporate/company environment where ec2 instances don't use a public ip address
ansible_host: private_ip_address	

detailed: https://docs.ansible.com/ansible/latest/collections/amazon/aws/aws_ec2_inventory.html
https://github.com/ansible-collections/amazon.aws/blob/main/plugins/inventory/aws_ec2.py

example:
plugin: aws_ec2
aws_access_key: '' YOUR AWS ACCESS KEY ''
aws_secret_key: '' YOUR AWS SECRET KEY ''
hostnames:
  - ip-address
  - tag:hostname
regions:
filters:
    instance-state-name: running
keyed_groups:
 - key: tags.function
   prefix: function
   separator: "_"

result of the command “ansible-inventory -i aws_ec2.yml –graph” :
@all:
  |--@aws_ec2:
  | |--15.188.62.98
  | |--3.120.238.184
  |--@function_ansible_master:
  | |--3.120.238.184
  |--@function_application:
  | |--15.188.62.98
  |--@ungrouped:

5. To execute the Ansible Playbook we can pass any AWS EC2 Tag or variable to the Playbook’s hosts variable as shown below.

- name: Ansible Test Playbook
  gather_facts: false
  hosts: aws_ec2 (or function_ansible_master , or function_application)
  tasks:

    - name: Run Shell Command
      command: echo "Hello World"

or you can set
hosts: "{{ srv }}"
and call playbook:
ansible-playbook **-e "srv=aws_ec2**" –i aws_ec2.yml playbook.yml
ansible-playbook **-e "srv=function_ansible_master "** –i aws_ec2.yml playbook.yml

Sample for our project

Assign appropriate tag “function” – frontend, json-filter, rabbit-to-bd, rest-api for each pf the ec2 instances according to their functionality.
So we can perform common for all instances plays using tag **aws_ec2** and plays for separate instance using appropriate tag **function_frontend** (**function_json-filter , function_rabbit-to-bd ,function_rest-api**)

./ansible_plugins /aws_ec2.yml file:

```
plugin: aws_ec2
aws_profile: default
hostnames:
  - ip-address
#for using  - tag:hostname this hostname should be resolved to ip
regions:
filters:
    instance-state-name: running
keyed_groups:
 - key: tags.function
   prefix: function
   separator: "_"
compose:
    ansible_host: private_ip_address
```

Testing

ansible-inventory -i aws_ec2.yaml –list
ansible-inventory -i aws_ec2.yml --graph

Recently terminated instances might appear in the returned results. This interval is usually less than one hour.

https://docs.aws.amazon.com/cli/latest/reference/ec2/describe-instances.html#options
https://github.com/ansible-collections/amazon.aws/blob/main/plugins/inventory/aws_ec2.py
https://docs.ansible.com/ansible/latest/collections/amazon/aws/aws_ec2_inventory.html
https://galaxy.ansible.com/amazon/aws?extIdCarryOver=true&sc_cid=701f2000001OH6uAAG
https://pypi.org/project/boto3/