**Dynamic Inventory**

**Topics**

* Dynamic Inventory
  + Example: The Cobbler External Inventory Script
  + Example: AWS EC2 External Inventory Script
  + Example: OpenStack External Inventory Script
    - Explicit use of inventory script
    - Implicit use of inventory script
    - Refresh the cache
  + Other inventory scripts
  + Using Inventory Directories and Multiple Inventory Sources
  + Static Groups of Dynamic Groups

Often a user of a configuration management system will want to keep inventory in a different software system. Ansible provides a basic text-based system as described in Inventory but what if you want to use something else?

Frequent examples include pulling inventory from a cloud provider, LDAP, Cobbler, or a piece of expensive enterprisey CMDB software.

Ansible easily supports all of these options via an external inventory system. The contrib/inventory directory contains some of these already – including options for EC2/Eucalyptus, Rackspace Cloud, and OpenStack, examples of some of which will be detailed below.

Ansible Tower also provides a database to store inventory results that is both web and REST Accessible. Tower syncs with all Ansible dynamic inventory sources you might be using, and also includes a graphical inventory editor. By having a database record of all of your hosts, it’s easy to correlate past event history and see which ones have had failures on their last playbook runs.

For information about writing your own dynamic inventory source, see Developing Dynamic Inventory Sources.

**Example: The Cobbler External Inventory Script**

It is expected that many Ansible users with a reasonable amount of physical hardware may also be Cobbler users. (note: Cobbler was originally written by Michael DeHaan and is now led by James Cammarata, who also works for Ansible, Inc).

While primarily used to kickoff OS installations and manage DHCP and DNS, Cobbler has a generic layer that allows it to represent data for multiple configuration management systems (even at the same time), and has been referred to as a ‘lightweight CMDB’ by some admins.

To tie Ansible’s inventory to Cobbler (optional), copy this script to /etc/ansible and chmod +x the file. cobblerd will now need to be running when you are using Ansible and you’ll need to use Ansible’s -i command line option (e.g. -i /etc/ansible/cobbler.py). This particular script will communicate with Cobbler using Cobbler’s XMLRPC API.

Also a cobbler.ini file should be added to /etc/ansible so Ansible knows where the Cobbler server is and some cache improvements can be used. For example:

[cobbler]

*# Set Cobbler's hostname or IP address*

host = http://127.0.0.1/cobbler\_api

# API calls to Cobbler can be slow. For this reason, we cache the results of an API

# call. Set this to the path you want cache files to be written to. Two files

# will be written to this directory:

*# - ansible-cobbler.cache*

*# - ansible-cobbler.index*

cache\_path = /tmp

*# The number of seconds a cache file is considered valid. After this many*

*# seconds, a new API call will be made, and the cache file will be updated.*

cache\_max\_age = 900

First test the script by running /etc/ansible/cobbler.py directly. You should see some JSON data output, but it may not have anything in it just yet.

Let’s explore what this does. In Cobbler, assume a scenario somewhat like the following:

cobbler profile add --name=webserver --distro=CentOS6-x86\_64

cobbler profile edit --name=webserver --mgmt-classes="webserver" --ksmeta="a=2 b=3"

cobbler system edit --name=foo --dns-name="foo.example.com" --mgmt-classes="atlanta" --ksmeta="c=4"

cobbler system edit --name=bar --dns-name="bar.example.com" --mgmt-classes="atlanta" --ksmeta="c=5"

In the example above, the system ‘foo.example.com’ will be addressable by ansible directly, but will also be addressable when using the group names ‘webserver’ or ‘atlanta’. Since Ansible uses SSH, we’ll try to contact system foo over ‘foo.example.com’, only, never just ‘foo’. Similarly, if you try “ansible foo” it wouldn’t find the system... but “ansible ‘foo\*’” would, because the system DNS name starts with ‘foo’.

The script doesn’t just provide host and group info. In addition, as a bonus, when the ‘setup’ module is run (which happens automatically when using playbooks), the variables ‘a’, ‘b’, and ‘c’ will all be auto-populated in the templates:

*# file: /srv/motd.j2*

Welcome, I am templated with a value of a=**{{** a **}}**, b=**{{** b **}}**, and c=**{{** c **}}**

Which could be executed just like this:

ansible webserver -m setup

ansible webserver -m template -a "src=/tmp/motd.j2 dest=/etc/motd"

**Note**

The name ‘webserver’ came from Cobbler, as did the variables for the config file. You can still pass in your own variables like normal in Ansible, but variables from the external inventory script will override any that have the same name.

So, with the template above (motd.j2), this would result in the following data being written to /etc/motd for system ‘foo’:

Welcome, I am templated with a value of a=2, b=3, and c=4

And on system ‘bar’ (bar.example.com):

Welcome, I am templated with a value of a=2, b=3, and c=5

And technically, though there is no major good reason to do it, this also works too:

ansible webserver -m shell -a "echo **{{** a **}}**"

So in other words, you can use those variables in arguments/actions as well.

**Example: AWS EC2 External Inventory Script**

If you use Amazon Web Services EC2, maintaining an inventory file might not be the best approach, because hosts may come and go over time, be managed by external applications, or you might even be using AWS autoscaling. For this reason, you can use the EC2 external inventory script.

You can use this script in one of two ways. The easiest is to use Ansible’s -i command line option and specify the path to the script after marking it executable:

ansible -i ec2.py -u ubuntu us-east-1d -m ping

The second option is to copy the script to */etc/ansible/hosts* and *chmod +x* it. You will also need to copy the ec2.ini file to */etc/ansible/ec2.ini*. Then you can run ansible as you would normally.

To successfully make an API call to AWS, you will need to configure Boto (the Python interface to AWS). There are a variety of methods available, but the simplest is just to export two environment variables:

export AWS\_ACCESS\_KEY\_ID='AK123'

export AWS\_SECRET\_ACCESS\_KEY='abc123'

You can test the script by itself to make sure your config is correct:

cd contrib/inventory

./ec2.py --list

After a few moments, you should see your entire EC2 inventory across all regions in JSON.

If you use Boto profiles to manage multiple AWS accounts, you can pass --profile PROFILE name to the ec2.py script. An example profile might be:

[profile dev]

aws\_access\_key\_id = <dev access key>

aws\_secret\_access\_key = <dev secret key>

[profile prod]

aws\_access\_key\_id = <prod access key>

aws\_secret\_access\_key = <prod secret key>

You can then run ec2.py --profile prod to get the inventory for the prod account, although this option is not supported by ansible-playbook. You can also use the AWS\_PROFILE variable - for example: AWS\_PROFILE=prod ansible-playbook -i ec2.py myplaybook.yml

Since each region requires its own API call, if you are only using a small set of regions, you can edit the ec2.ini file and comment out the regions you are not using.

There are other config options in ec2.ini, including cache control and destination variables. By default, the ec2.ini file is configured for **all Amazon cloud services**, but you can comment out any features that aren’t applicable. For example, if you don’t have RDS or elasticache, you can set them to False

[ec2]

...

*# To exclude RDS instances from the inventory, uncomment and set to False.*

rds = False

# To exclude ElastiCache instances from the inventory, uncomment and set to False.

elasticache = False

...

At their heart, inventory files are simply a mapping from some name to a destination address. The default ec2.ini settings are configured for running Ansible from outside EC2 (from your laptop for example) – and this is not the most efficient way to manage EC2.

If you are running Ansible from within EC2, internal DNS names and IP addresses may make more sense than public DNS names. In this case, you can modify the destination\_variable in ec2.ini to be the private DNS name of an instance. This is particularly important when running Ansible within a private subnet inside a VPC, where the only way to access an instance is via its private IP address. For VPC instances, *vpc\_destination\_variable* in ec2.ini provides a means of using which ever boto.ec2.instance variable makes the most sense for your use case.

The EC2 external inventory provides mappings to instances from several groups:

**Global**

All instances are in group ec2.

**Instance ID**

These are groups of one since instance IDs are unique. e.g. i-00112233 i-a1b1c1d1

**Region**

A group of all instances in an AWS region. e.g. us-east-1 us-west-2

**Availability Zone**

A group of all instances in an availability zone. e.g. us-east-1a us-east-1b

**Security Group**

Instances belong to one or more security groups. A group is created for each security group, with all characters except alphanumerics, converted to underscores (\_). Each group is prefixed by security\_group\_. Currently, dashes (-) are also converted to underscores (\_). You can change using the replace\_dash\_in\_groups setting in ec2.ini (this has changed across several versions so check the ec2.ini for details). e.g. security\_group\_defaultsecurity\_group\_webservers security\_group\_Pete\_s\_Fancy\_Group

**Tags**

Each instance can have a variety of key/value pairs associated with it called Tags. The most common tag key is ‘Name’, though anything is possible. Each key/value pair is its own group of instances, again with special characters converted to underscores, in the format tag\_KEY\_VALUE e.g.tag\_Name\_Web can be used as is tag\_Name\_redis-master-001 becomes tag\_Name\_redis\_master\_001 tag\_aws\_cloudformation\_logical-id\_WebServerGroup becomes tag\_aws\_cloudformation\_logical\_id\_WebServerGroup

When the Ansible is interacting with a specific server, the EC2 inventory script is called again with the --host HOST option. This looks up the HOST in the index cache to get the instance ID, and then makes an API call to AWS to get information about that specific instance. It then makes information about that instance available as variables to your playbooks. Each variable is prefixed by ec2\_. Here are some of the variables available:

* ec2\_architecture
* ec2\_description
* ec2\_dns\_name
* ec2\_id
* ec2\_image\_id
* ec2\_instance\_type
* ec2\_ip\_address
* ec2\_kernel
* ec2\_key\_name
* ec2\_launch\_time
* ec2\_monitored
* ec2\_ownerId
* ec2\_placement
* ec2\_platform
* ec2\_previous\_state
* ec2\_private\_dns\_name
* ec2\_private\_ip\_address
* ec2\_public\_dns\_name
* ec2\_ramdisk
* ec2\_region
* ec2\_root\_device\_name
* ec2\_root\_device\_type
* ec2\_security\_group\_ids
* ec2\_security\_group\_names
* ec2\_spot\_instance\_request\_id
* ec2\_state
* ec2\_state\_code
* ec2\_state\_reason
* ec2\_status
* ec2\_subnet\_id
* ec2\_tag\_Name
* ec2\_tenancy
* ec2\_virtualization\_type
* ec2\_vpc\_id

Both ec2\_security\_group\_ids and ec2\_security\_group\_names are comma-separated lists of all security groups. Each EC2 tag is a variable in the format ec2\_tag\_KEY.

To see the complete list of variables available for an instance, run the script by itself:

cd contrib/inventory

./ec2.py --host ec2-12-12-12-12.compute-1.amazonaws.com

Note that the AWS inventory script will cache results to avoid repeated API calls, and this cache setting is configurable in ec2.ini. To explicitly clear the cache, you can run the ec2.py script with the --refresh-cache parameter:

./ec2.py --refresh-cache

**Example: OpenStack External Inventory Script**

If you use an OpenStack based cloud, instead of manually maintaining your own inventory file, you can use the openstack.py dynamic inventory to pull information about your compute instances directly from OpenStack.

You can download the latest version of the OpenStack inventory script here

You can use the inventory script explicitly (by passing the *-i openstack.py* argument to Ansible) or implicitly (by placing the script at */etc/ansible/hosts*).

**Explicit use of inventory script**

Download the latest version of the OpenStack dynamic inventory script and make it executable:

wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/openstack.py

chmod +x openstack.py

Source an OpenStack RC file:

source openstack.rc

**Note**

An OpenStack RC file contains the environment variables required by the client tools to establish a connection with the cloud provider, such as the authentication URL, user name, password and region name. For more information on how to download, create or source an OpenStack RC file, please refer to Set environment variables using the OpenStack RC file.

You can confirm the file has been successfully sourced by running a simple command, such as *nova list* and ensuring it return no errors.

**Note**

The OpenStack command line clients are required to run the *nova list* command. For more information on how to install them, please refer to Install the OpenStack command-line clients.

You can test the OpenStack dynamic inventory script manually to confirm it is working as expected:

./openstack.py --list

After a few moments you should see some JSON output with information about your compute instances.

Once you confirm the dynamic inventory script is working as expected, you can tell Ansible to use the *openstack.py* script as an inventory file, as illustrated below:

ansible -i openstack.py all -m ping

**Implicit use of inventory script**

Download the latest version of the OpenStack dynamic inventory script, make it executable and copy it to */etc/ansible/hosts*:

wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/openstack.py

chmod +x openstack.py

sudo cp openstack.py /etc/ansible/hosts

Download the sample configuration file, modify it to suit your needs and copy it to */etc/ansible/openstack.yml*:

wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/openstack.yml

vi openstack.yml

sudo cp openstack.yml /etc/ansible/

You can test the OpenStack dynamic inventory script manually to confirm it is working as expected:

/etc/ansible/hosts --list

After a few moments you should see some JSON output with information about your compute instances.

**Refresh the cache**

Note that the OpenStack dynamic inventory script will cache results to avoid repeated API calls. To explicitly clear the cache, you can run the openstack.py (or hosts) script with the --refresh parameter:

./openstack.py --refresh --list

**Other inventory scripts**

In addition to Cobbler and EC2, inventory scripts are also available for:

BSD Jails

DigitalOcean

Google Compute Engine

Linode

OpenShift

OpenStack Nova

Ovirt

SpaceWalk

Vagrant (not to be confused with the provisioner in vagrant, which is preferred)

Zabbix

Sections on how to use these in more detail will be added over time, but by looking at the “contrib/inventory” directory of the Ansible checkout it should be very obvious how to use them. The process for the AWS inventory script is the same.

If you develop an interesting inventory script that might be general purpose, please submit a pull request – we’d likely be glad to include it in the project.

**Using Inventory Directories and Multiple Inventory Sources**

If the location given to -i in Ansible is a directory (or as so configured in ansible.cfg), Ansible can use multiple inventory sources at the same time. When doing so, it is possible to mix both dynamic and statically managed inventory sources in the same ansible run. Instant hybrid cloud!

In an inventory directory, executable files will be treated as dynamic inventory sources and most other files as static sources. Files which end with any of the following will be ignored:

~, .orig, .bak, .ini, .cfg, .retry, .pyc, .pyo

You can replace this list with your own selection by configuring an inventory\_ignore\_extensions list in ansible.cfg, or setting the ANSIBLE\_INVENTORY\_IGNOREenvironment variable. The value in either case should be a comma-separated list of patterns, as shown above.

Any group\_vars and host\_vars subdirectories in an inventory directory will be interpreted as expected, making inventory directories a powerful way to organize different sets of configurations.

**Static Groups of Dynamic Groups**

When defining groups of groups in the static inventory file, the child groups must also be defined in the static inventory file, or ansible will return an error. If you want to define a static group of dynamic child groups, define the dynamic groups as empty in the static inventory file. For example:

[tag\_Name\_staging\_foo]

[tag\_Name\_staging\_bar]

[staging:children]

tag\_Name\_staging\_foo

tag\_Name\_staging\_bar