Google Cloud Platform Guide

Introduction

Ansible + Google have been working together on a set of auto-generated Ansible modules designed to consistently and comprehensively cover the entirety of the Google Cloud Platform (GCP).

Ansible contains modules for managing Google Cloud Platform resources, including creating instances, controlling network access, working with persistent disks, managing load balancers, and a lot more.

These new modules can be found under a new consistent name scheme "gcp_*" (Note: gcp_target_proxy and gcp_url_map are legacy modules, despite the "gcp_*" name. Please use gcp_compute_target_proxy and gcp_compute_url_map instead).

Additionally, the gcp_compute inventory plugin can discover all Google Compute Engine (GCE) instances and make them automatically available in your Ansible inventory.

You may see a collection of other GCP modules that do not conform to this naming convention. These are the original modules primarily developed by the Ansible community. You will find some overlapping functionality such as with the "gce" module and the new "gcp_compute_instance" module. Either can be used, but you may experience issues trying to use them together.

While the community GCP modules are not going away, Google is investing effort into the new "gcp_*" modules. Google is committed to ensuring the Ansible community has a great experience with GCP and therefore recommends adopting these new modules if possible.

Requisites

The GCP modules require both the `requests` and the `google-auth` libraries to be installed.

```
$ pip install requests google-auth
```

Alternatively for RHEL / CentOS, the `python-requests` package is also available to satisfy `requests` libraries.

```
$ yum install python-requests
```

Credentials

It's easy to create a GCP account with credentials for Ansible. You have multiple options to get your credentials - here are two of the most common options:

- Service Accounts (Recommended): Use JSON service accounts with specific permissions.
- Machine Accounts: Use the permissions associated with the GCP Instance you're using Ansible on.

For the following examples, we'll be using service account credentials.

To work with the GCP modules, you'll first need to get some credentials in the JSON format:

1. Create a Service Account
2. Download JSON credentials

Once you have your credentials, there are two different ways to provide them to Ansible:

- by specifying them directly as module parameters
- by setting environment variables

Providing Credentials as Module Parameters

For the GCE modules you can specify the credentials as arguments:

- `auth_kind`: type of authentication being used (choices: machineaccount, serviceaccount, application)
- `service_account_email`: email associated with the project
- `service_account_file`: path to the JSON credentials file
- `project`: id of the project
- `scopes`: The specific scopes that you want the actions to use.

For example, to create a new IP address using the `gcp_compute_address` module, you can use the following configuration:

```
- name: Create IP address
  hosts: localhost
  gather_facts: no

  vars:
    service_account_file: /home/my_account.json
    project: my-project
    auth_kind: serviceaccount
    scopes:
      - https://www.googleapis.com/auth/compute

  tasks:

   - name: Allocate an IP Address
     gcp_compute_address:
         state: present
         name: 'test-address1'
         region: 'us-west1'
         project: "{{ project }}"
         auth_kind: "{{ auth_kind }}"
         service_account_file: "{{ service_account_file }}"
         scopes: "{{ scopes }}"
```

Providing Credentials as Environment Variables

Set the following environment variables before running Ansible in order to configure your credentials:

```
GCP_AUTH_KIND
GCP_SERVICE_ACCOUNT_EMAIL
GCP_SERVICE_ACCOUNT_FILE
GCP_SCOPES
```

GCE Dynamic Inventory

The best way to interact with your hosts is to use the gcp_compute inventory plugin, which dynamically queries GCE and tells Ansible what nodes can be managed.

To be able to use this GCE dynamic inventory plugin, you need to enable it first by specifying the following in the `ansible.cfg` file:

```
[inventory]
enable_plugins = gcp_compute
```

Then, create a file that ends in `.gcp.yml` in your root directory.

The gcp_compute inventory script takes in the same authentication information as any module.

Here's an example of a valid inventory file:

```
plugin: gcp_compute
projects:
  - graphite-playground
auth_kind: serviceaccount
service_account_file: /home/alexstephen/my_account.json
```

Executing `ansible-inventory --list -i <filename>.gcp.yml` will create a list of GCP instances that are ready to be configured using Ansible.

Create an instance

The full range of GCP modules provide the ability to create a wide variety of GCP resources with the full support of the entire GCP API.

The following playbook creates a GCE Instance. This instance relies on other GCP resources like Disk. By creating other resources separately, we can give as much detail as necessary about how we want to configure the other resources, for example formatting of the Disk. By registering it to a variable, we can simply insert the variable into the instance task. The gcp_compute_instance module will figure out the rest.

```
- name: Create an instance
  hosts: localhost
  gather_facts: no
  vars:
      gcp_project: my-project
      gcp_cred_kind: serviceaccount
      gcp_cred_file: /home/my_account.json
      zone: "us-central1-a"
      region: "us-central1"

  tasks:
   - name: create a disk
     gcp_compute_disk:
         name: 'disk-instance'
         size_gb: 50
         source_image: 'projects/ubuntu-os-cloud/global/images/family/ubuntu-1604-lts'
         zone: "{{ zone }}"
         project: "{{ gcp_project }}"
         auth_kind: "{{ gcp_cred_kind }}"
```

```
          service_account_file: "{{ gcp_cred_file }}"
          scopes:
            - https://www.googleapis.com/auth/compute
          state: present
      register: disk
    - name: create a address
      gcp_compute_address:
          name: 'address-instance'
          region: "{{ region }}"
          project: "{{ gcp_project }}"
          auth_kind: "{{ gcp_cred_kind }}"
          service_account_file: "{{ gcp_cred_file }}"
          scopes:
            - https://www.googleapis.com/auth/compute
          state: present
      register: address
    - name: create a instance
      gcp_compute_instance:
          state: present
          name: test-vm
          machine_type: n1-standard-1
          disks:
            - auto_delete: true
              boot: true
              source: "{{ disk }}"
          network_interfaces:
              - network: null # use default
                access_configs:
                  - name: 'External NAT'
                    nat_ip: "{{ address }}"
                    type: 'ONE_TO_ONE_NAT'
          zone: "{{ zone }}"
          project: "{{ gcp_project }}"
          auth_kind: "{{ gcp_cred_kind }}"
          service_account_file: "{{ gcp_cred_file }}"
          scopes:
            - https://www.googleapis.com/auth/compute
      register: instance

    - name: Wait for SSH to come up
      wait_for: host={{ address.address }} port=22 delay=10 timeout=60

    - name: Add host to groupname
      add_host: hostname={{ address.address }} groupname=new_instances


- name: Manage new instances
  hosts: new_instances
  connection: ssh
  become: True
  roles:
    - base_configuration
    - production_server
```

Note that use of the "add_host" module above creates a temporary, in-memory group. This means that a play in the same playbook can then manage machines in the 'new_instances' group, if so desired. Any sort of arbitrary configuration is possible at this point.

For more information about Google Cloud, please visit the Google Cloud website.

Migration Guides

gce.py -> gcp_compute_instance.py

As of Ansible 2.8, we're encouraging everyone to move from the `gce` module to the `gcp_compute_instance` module. The `gcp_compute_instance` module has better support for all of GCP's features, fewer dependencies, more flexibility, and better supports GCP's authentication systems.

The `gcp_compute_instance` module supports all of the features of the `gce` module (and more!). Below is a mapping of `gce` fields over to `gcp_compute_instance` fields.

gce.py	gcp_compute_instance.py	Notes
state	state/status	State on gce has multiple values: "present", "absent", "stopped", "started", "terminated". State on gcp_compute_instance is used to describe if the instance exists (present) or does not (absent). Status is used to describe if the instance is "started", "stopped" or "terminated".

gce.py	gcp_compute_instance.py	Notes
image	disks[].initialize_params.source_image	You'll need to create a single disk using the disks[] parameter and set it to be the boot disk (disks[].boot = true)
image_family	disks[].initialize_params.source_image	See above.
external_projects	disks[].initialize_params.source_image	The name of the source_image will include the name of the project.
instance_names	Use a loop or multiple tasks.	Using loops is a more Ansible-centric way of creating multiple instances and gives you the most flexibility.
service_account_email	service_accounts[].email	This is the service_account email address that you want the instance to be associated with. It is not the service_account email address that is used for the credentials necessary to create the instance.
service_account_permissions	service_accounts[].scopes	These are the permissions you want to grant to the instance.
pem_file	Not supported.	We recommend using JSON service account credentials instead of PEM files.
credentials_file	service_account_file	
project_id	project	
name	name	This field does not accept an array of names. Use a loop to create multiple instances.
num_instances	Use a loop	For maximum flexibility, we're encouraging users to use Ansible features to create multiple instances, rather than letting the module do it for you.
network	network_interfaces[].network	
subnetwork	network_interfaces[].subnetwork	
persistent_boot_disk	disks[].type = 'PERSISTENT'	
disks	disks[]	
ip_forward	can_ip_forward	
external_ip	network_interfaces[].access_configs.nat_ip	This field takes multiple types of values. You can create an IP address with `gcp_compute_address` and place the name/output of the address here. You can also place the string value of the IP address's GCP name or the actual IP address.
disks_auto_delete	disks[].auto_delete	
preemptible	scheduling.preemptible	
disk_size	disks[].initialize_params.disk_size_gb	

An example playbook is below:

```
gcp_compute_instance:
    name: "{{ item }}"
    machine_type: n1-standard-1
    ... # any other settings
    zone: us-central1-a
    project: "my-project"
    auth_kind: "service_account_file"
    service_account_file: "~/my_account.json"
    state: present
loop:
  - instance-1
  - instance-2
```