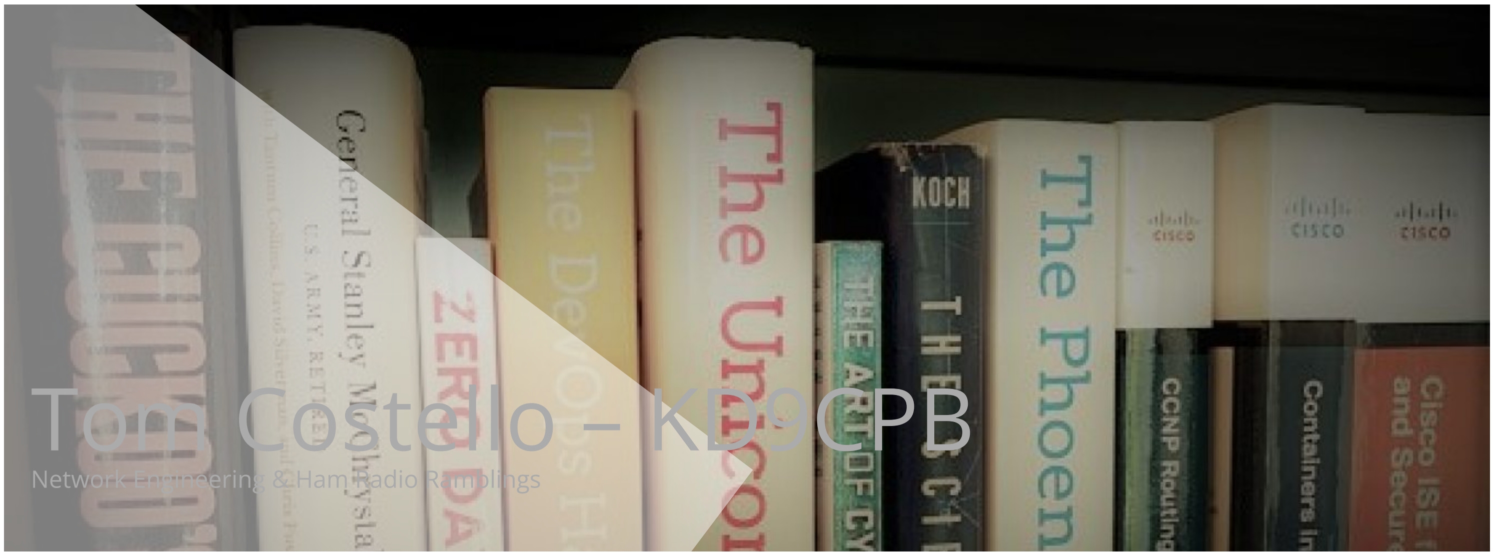


Automating startup/shutdown of a GCP eve-ng homelab with VPN tunnel

Homelab

kd9cpb

> *As always, opinions in this post are solely those of my own, and not necessarily those of any organization I am currently affiliated with or have been in the past.*

First posted 5/7/2021

I really love my Google Cloud Platform (GCP) eve-ng instance paired with an IPSec tunnel from my homelab's OPNsense firewall into GCP Virtual Private Cloud (VPC). It's amazing how much compute power you can access for $0.53 an hour with their 16-core CPU 60gb RAM machine type!

That being said, I really hate when I accidently leave my GCP eve-ng instance and/or IPsec tunnel running when I'm not actively tinkering in the homelab. It's amazing how quickly you can run up the cloud computing bill when you forget to shutdown your 16-core eve-ng instance or keep the IPSec tunnel running 24/7!

Wouldn't it be great if we could automate away the problem of forgetfully leaving a GCP eve-ng instance running at $0.53 an hour? Maybe we could even automatically create/destroy the $0.05 an hour IPSec tunnel into the homelab's OPNsense firewall as needed? I hope you answered yes, because today's post is all about making startup/shutdown of a GCP eve-ng playground much smoother.

I'll be starting with the finished product from my OPNsense IPSec tunnel into GCP eve-ng post for this homelab. If you did the ZeroTier tunnel setup from Network Collective's glorious

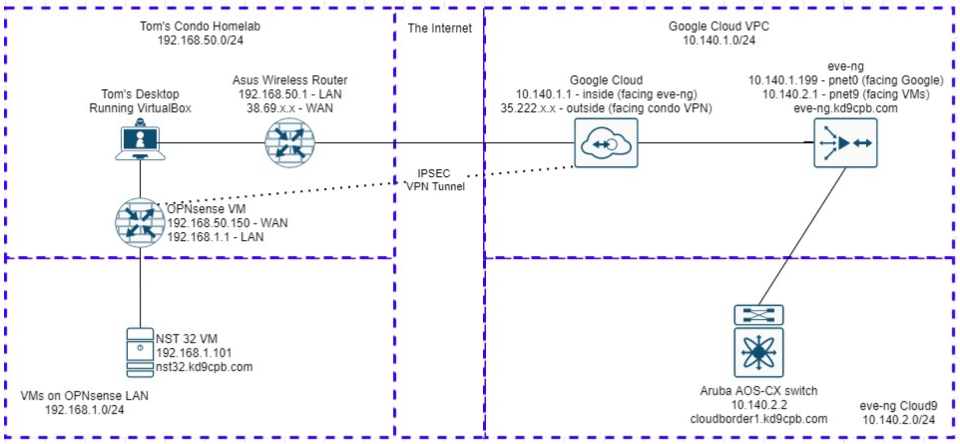
Type and hit enter...

Please click the picture above to vote for kd9cpb.com today!

Most Recent Posts

- RPi 4 WordPress server with Cloudfl
Free via Dynamic WAN IP
- New license, new radio!
- Diving into DMR hotspot traffic with Wireshark
- Homelab Hiatus
- qdmr (DMR code-plug utility) on Raspberry Pi OS & AntiX Linux
- Working 10m FT8 with only a technician license and sub-$100 used radio
- An FT8 attempt was made

YouTube walkthrough instead of an OPNsense IPsec tunnel, be sure to skip the OPNsense stuff in the initial setup section below, and only complete steps 5-9 in the script. Here's a quick diagram of all the topology items I'll be using today (you can substitute NST32 for just about any flavor of Linux if you're following along at home):



Initial Setup

On the NST32 box, I need to have Ansible and Google Cloud API access squared away before I can start the fun automation stuff. I also need to ensure OPNsense will accept the SSH traffic that Ansible will be sending it. As of this writing, the following commands will get you Ansible 2.9, the Google Cloud SDK, and a few other dependencies installed. Your mileage may vary with other Ansible versions and/or Linux distros that are not based on Fedora 32:

```
sudo yum install ansible
sudo pip3 install lxml requests google-auth paramiko
ansible-galaxy install git+https://github.com/naturalis/ansible-
opnsense.git
ansible-galaxy collection install google.cloud
```

I made a new directory called lab10 where I'll be saving all of the scripts in the lab today, the contents of which are available up on GitHub. Let's create a text file named hosts.ini with the following so Ansible knows how to get into the OPNsense firewall:

```
[opnsense]
192.168.1.1

[opnsense:vars]
ansible_user=root
ansible_ssh_pass="not_the_real_ssh_password"
ansible_become_method=sudo
ansible_shell_type=csh
local_config_path='/home/nstuser/lab10/config.xml'
```

Obviously put your proper SSH password in the file (or better yet, use something like Ansible Vault) and change the local config path to the directory of your choosing. Next, make a text file called opntest.yml containing the following:

Using NX-OS 10.1.2's new two-stage configuration commit with Netmiko
Automating startup/shutdown of a GCP eve-ng homelab with VPN tunnel
Don't injure yourself sprinting towar IT certifications
Google Cloud eve-ng setup with OPNsense homelab tunnel in 2021
Network Automation with Jinja2 & 1 Grade Level Ansible

Tom Costello Follow
Network security nerd with a ham radio l Internet plumber by day, https://t.co/8ZnF2jnxUD author by night. Views/Opinions are mine. RT != E

Tom Costello Retweeted
Carla Notarobot 🤖👩🏻‍💻 @CarlaNotarc
13 Jan
Junior dev: "I fucked up bad, I'm so

Senior dev: " I have 3 production ou named after me lol"
3141 34053 Twitter

Tom Costello @kd9cpb · 10 Jan
I made the #ITBlogAwards finalist c Please vote for http://kd9cpb.com t Big thanks to the @CiscoChampion so cool to see the #neteng blogging community keep growing

Cisco Champion @CiscoCham
It is our pleasure to announce the 2021 #ITBlogAwards finalists. Congratulations!

It is now up to our wonderful tech community to show the love. Che out who made the list and cast yo vote today. http://cs.co/6010JhKD

2 8 Twitter

Tom Costello Retweeted
Ray [REDACTED] @RayRedacted · 10 Ja
Question: have you ever attended a virtual InfoSec conference where th topics included not only DFIR, SOC2 compliance, and malware analysis, also Farming Fails, the history of LE Improv, and Antique photography?

Block off this Sunday, January 16th, @PancakesCon !
71 258 Twitter

Tom Costello @kd9cpb · 10 Jan

```
---
- name: Verify Ansible can SSH into OPNsense
  hosts: opnsense
  gather_facts: false
  become: true
  roles:
    - ansible-opnsense
```

This is an Ansible playbook simply saying to SSH into the OPNsense VM, become root, and use the ansible-opnsense role to do things. Head to your OPNsense GUI and ensure secure shell is enabled, root login permitted, password login allowed, and listening on the LAN interface by changing the following under Settings -> Administration:

Fire up a terminal in NST32, make sure SSH into the OPNsense firewall works before we start tinkering with Ansible:

I'm super excited to be a backup sp
for @PancakesCon 3!!! Regardless o
whether or not my talk sees the ligh
day, it's going to be an awesome tir
can't wait

PancakesCon 3: Batter Up @PancakesCon
Now, congratulations to our 2022
speakers - we are excited to see y
See the full schedule at
https://pancakescon.com/2022-conference-information/

3 Twitter

Tom Costello Retweeted

Tim McC @juangolbez · 8 Jan
Signal boosting this for someone, if
sounds interesting, check it out!

https://lightbytes.es.net/2022/01/0
for-the-2022-wome...

7 5 Twitter

Load More...

Close out that terminal window, and try running the following in the folder where you've created the Ansible inventory (hosts.ini) and playbook (opntest.yml) files:

```
ansible-playbook -i hosts.ini -c paramiko opntest.yml -t fetch
```

As we can see from the above ls command, we successfully grabbed the config.xml file off our OPNsense firewall using the ansible-opnsense role! Whether it's your first time running an Ansible playbook or your 1000th, I always love that feeling when you first get data back from a new device type, this framework is so powerful 🙂

Unfortunately Ansible gives us an ugly warning that it can't seem to delete temporary files on the OPNsense firewall, but I'm not too concerned about that as it doesn't seem to be impacting our use-case today. Now that we have our initial Ansible plumbing figured out, head over to your GCP Main Menu, select IAM & Admin, then Service Accounts to get the GCP API access ironed out:

Click on the "Create Service Account" button, name it something appropriate (I chose automationuser), and give it owner permissions:

Click on the 3 dots towards the right of your new service account, select “Manage Keys”. Then click on the Add Key button, and select Create New Key:

Create the key in JSON format, then save it to an appropriate place on your machine (I’m using the same lab10 folder).

In the upper left corner of GCP console, click on your project’s name, and take note of the **full** project ID with the dashes & numbers, we’ll need this shortly:

Open up /etc/ansible/ansible.cfg in the editor of your choosing as root, then add the following line under the [inventory] section, **but be careful if you’re using inventories other than ini or yaml, this might break things if you’re a more advanced Ansible user**:

```
enable_plugins = gcp_compute, ini, yaml
```

Head back to the folder containing all your scripts for this lab, and make a new file named gcp.yml with the following contents:

```
plugin: gcp_compute
projects:
  - your_full_project_id
auth_kind: serviceaccount
service_account_file: your_gcp_json_file.json
```

Replace "your_full_project_id" with whatever your full GCP project name is, and change the filename on the last line to whatever you've named your JSON credential file downloaded from GCP. Next, run the following command, and you should be greeted with a nice long list of output about all GCP instances in your project!

```
ansible-inventory --list -i gcp.yml
```

I'll probably do another homelab that does much more with this GCP inventory plugin soon, it's a crazy powerful tool, but for today we are only using it to validate Ansible can login to GCP. Now that we've seen with our own eyes that Ansible can talk with OPNsense and GCP, we can really get the automation party started 🙂

Main Script Layout

To keep things somewhat organized, I'm going to have one main shell script named eve-start.sh, which will perform the following ten actions:

1. Get current public IP Address, save to file (Python)
2. Get current OPNsense config (Ansible)
3. Check if VPN config has correct public IP, update if necessary (Python + Ansible)
4. Create GCP VPN tunnel (Ansible)
5. Start GCP eve-ng instance (Ansible)
6. Begin pinging GCP eve-ng instance over the VPN tunnel, display message once it's alive (Shell)
7. Wait for input to start teardown process (Shell)
8. Gracefully shutdown GCP eve-ng instance (Ansible)
9. Verify GCP eve-ng instance is down (Ansible)

10. Destroy GCP VPN tunnel (Ansible)

Most of the automation heavy-lifting is going to be with Ansible, however I will be using python for a few tasks along with shell scripting to tie everything together. You can also just grab all the code in the GitHub folder and make the appropriate minor password + GCP project specific changes detailed in the steps below, no need to copy/paste from this webpage.

Step 1: Get current public IP address with Python, save to file
Because I have a dynamic WAN IP address from my ISP and my OPNsense VM is running behind NAT on my home wireless router, I need to use a 3rd party for programmatically figuring out what my current WAN IP address is. Copy/paste the following into publicip.py, give it a test run. If it worked, you'll see the public IP on-screen, and it will also save it to ipaddr.txt:

```
from requests import get

# see https://stackoverflow.com/questions/2311510/getting-a-machines-
external-ip-address-with-python/41432835

ip = get('https://api.ipify.org').text
print('My public IP address is: {}'.format(ip))
file = open('ipaddr.txt', 'w')
file.write(ip)
file.close
```

I'm censoring the final two octets of my IP address throughout this lab for privacy reasons, but it looks like the simple script did its job! Note how the bash prompt is immediately after the output of "cat ipaddr.txt". This is because there is no newline at the end of the file our Python script wrote, which is fine for today's homelab. While this isn't the cleanest way to retrieve or store the public IP, it works in even the craziest multiple NAT situations you may find yourself in at a college campus or hotel, so we'll live with it for now.

Step 2: Get current OPNsense config with Ansible
Make a new playbook named opnsense.yml with the following content, it's the same thing as our initial Ansible test playbook into OPNsense, but with a more appropriate name:

```
---
- name: Push/Pull OPNsense config file
  hosts: opnsense
  gather_facts: false
  become: true
  roles:
    - ansible-opnsense
```

Now that we have the tooling to get the public IP and OPNsense config, let's create our main shell script named eve-start.sh, containing the following:

```
#!/bin/bash

# step 1: get fresh public IP
rm -f ipaddr.txt
python3 publicip.py

# step 2: get fresh OPNsense config
rm -f config.xml
ansible-playbook -i hosts.ini -c paramiko opnsense.yml -t fetch
```

Give the script proper permissions with a “chmod +x eve-start.sh”, and go for a test run!

Looks pretty good! Our main script is now cleaning out old results from the public IP retrieval and OPNsense config download tasks, then grabbing fresh data. Again, I’m ignoring that remote temporary file deletion warning as I believe it’s related to an ansible-opnsense dependency I can’t easily install on Red-hat flavored Linux distros. Check out the dependencies section of the ansible-opnsense GitLab page for more info about this if you’re curious.

Step 3: Check if VPN config has correct public IP, update if necessary

Because of google’s VPN rules on things behind NAT, it’s critical that OPNsense has the proper WAN IP configured in the “My Identifier” field of your VPN settings to bring up the GCP tunnel. This next step assumes you’ve already manually configured a tunnel from OPNsense to GCP like the one in my previous homelab write-up, so you’ll want to do that if you haven’t already. Make a new file named ipcheck.py, entering in the following code:

```
import re
import fileinput
import os

print("Checking OPNsense VPN config's myid_data IP address")
initial_data = []

# read the OPNsense config for the myid_data line
# help from https://stackoverflow.com/questions/51427543/print-lines-
from-file-that-contain-a-string/51427710
with open('config.xml', 'r') as f:
    config = f.readlines()
for line in config:
    if line.__contains__('myid_data'):
        initial_data.append(line)

# cleanup the remote-gateway line to only have the IP address
# help from https://stackoverflow.com/questions/2890896/how-to-extract-
an-ip-address-from-an-html-string
ip_addrs = re.findall( r'[0-9]+(?:\.[0-9]+){3}', initial_data[0] )
```

```
# read the public IP address from file
with open('ipaddr.txt', 'r') as f:
    public_ip = f.readline()
ip_addrs.append(public_ip)

print("OPNsense config's remote-gateway IP is: " + ip_addrs[0])

if ip_addrs[0] == ip_addrs[1]:
    print("We have a match, continuing to GCP tunnel config!")
else:
    print("No match, updating config.xml")
    # help from https://stackoverflow.com/questions/17140886/how-to-
search-and-replace-text-in-a-file
    with fileinput.FileInput('config.xml', inplace=True) as file:
        for line in file:
            print(line.replace(ip_addrs[0], ip_addrs[1]), end='')
    os.system("ansible-playbook -i hosts.ini -c paramiko opntest.yml -t
copy,reload")
```

As you can see, it took me three Stackoverflow articles and some shady stuff to pull off parsing + updating the IP address from that big OPNsense XML config file, this is not my cleanest code 🙂 If the Python script decides it needs to update the OPNsense config with Ansible, it will also reload the configuration so the changes are applied immediately. Honestly, this script is a bit nasty as it only supports one VPN tunnel configured on your OPNsense firewall, so be careful if you have more than one tunnel! All things considered, this lame Python is good enough for today's use-case, so I'll add the following two lines to the bottom of eve-start.sh, rolling it into our main script:

```
# step 3: run IP checker, which will update OPNsense config if necessary
python3 ipcheck.py
```

To ensure the IP address change logic works, login to your OPNsense firewall's webGUI. Under VPN -> IPSec -> Tunnel Settings, click the little Edit pencil next to your GCP tunnel. Change the "My Identifier" field to a ridiculous IP address. This way, we'll know the current WAN IP isn't going to match what our Python script gets as our current WAN IP:

Save & apply the config, then kick off the eve-start.sh script again. If all goes well, you should be greeted by output similar to the following, and the "My Identifier" field at the webGUI changing to the proper public IP address!

Step 4: Create Google Cloud VPN tunnel with Ansible

Before we get too in-depth with Ansible talking to GCP, go ahead and delete any VPN tunnels you've setup to your OPNsense homelab in the past under the GCP Main Menu -> Hybrid Connectivity -> VPN.

Ensure you still have a good Cloud VPN gateway setup too. If you don't have one setup like in the screenshot below, create one:

Note that my gateway name is named vpn-1, we'll need this name in our Ansible Playbook. If your gateway name is something different, **make sure to change the last part of the HTTPS links below too**. Create a new file named gcpvpn.yml, and enter the following, substituting your project ID, region and JSON account file name to fit your needs. This playbook will create the VPN tunnel, and also fix routing towards the tunnel:

```
---
- name: Create VPN tunnel to GCP
  hosts: localhost
  gather_facts: false
  vars:
    public_ip: "{{ lookup('file', 'ipaddr.txt') }}"
  tasks:
    - name: Create tunnel
      google.cloud.gcp_compute_vpn_tunnel:
        name: vpn-opnsense
        target_vpn_gateway:
          selfLink:
https://www.googleapis.com/compute/v1/projects/your_full_project_id/regio
ns/us-central1/targetVpnGateways/vpn-1
        shared_secret: not_the_real_key
        local_traffic_selector: 10.140.0.0/16
        remote_traffic_selector: 192.168.1.0/24
        peer_ip: "{{ public_ip }}"
        region: us-central1
        project: your_full_project_id
        auth_kind: serviceaccount
        service_account_file: your_gcp_json_file.json
    - name: Create Route
      google.cloud.gcp_compute_route:
        name: to-opnsense
        next_hop_vpn_tunnel:
          selfLink:
https://www.googleapis.com/compute/v1/projects/your_full_project_id/regio
ns/us-central1/vpnTunnels/vpn-opnsense
        dest_range: 192.168.1.0/24
        network:
          selfLink:
https://www.googleapis.com/compute/v1/projects/your_full_project_id/globa
l/networks/default
        project: your_full_project_id
        auth_kind: serviceaccount
        service_account_file: your_gcp_json_file.json
```

Fire off the playbook with “ansible-playbook -v gcpvpn.yml”, if all goes well you should see output similar to the following:

Step 5: Power on our GCP eve-ng instance with Ansible

Before we continue, check your Compute Engine -> VM instances menu for the exact name and zone of your GCP eve-ng instance, we'll need this in our Ansible playbook. Also go ahead and turn off your GCP eve-ng instance, Ansible will be powering it back on for you shortly 🙂

Make another file named eve-powerup.yml, paste in the following, change the name to match your instance's if necessary. The magic part of this playbook is the "status: RUNNING" section, which effectively flips the power switch on for the GCP eve-ng instance:

```
- name: turn on eve-ng instance
  hosts: localhost
  gather_facts: false
  tasks:
    - name: turn on eve-ng instance
      google.cloud.gcp_compute_instance:
        name: eveng
        status: RUNNING
        zone: us-central1-a
        project: your_full_project_id
        auth_kind: serviceaccount
        service_account_file: your_gcp_json_file.json
```

If all goes well, you'll see output similar to the following when you run "ansible-playbook -v eve-powerup.yml", and you can ping your eve-ng instance once it's booted up!

Step 6: Ping GCP eve-ng instance over the VPN tunnel, display message once it's alive

Now that we've gotten all our OPNsense & GCP setup tasks automated away, we need to add steps 4 & 5 into the main script, and a ping check to ensure the eve-ng instance is reachable over the tunnel. I also want to display a counter showing how long the script has been running, that way we don't get FOMO about hitting Ctrl+C it if things sit idle for too long. We'll accomplish this by adding the following into the bottom of eve-start.sh:

```
# step 4: setup GCP VPN
ansible-playbook gcpvpn.yml

# step 5: power on eve-ng GCP instance
ansible-playbook eve-powerup.yml

# step 6: ping check to validate VPN + GCP eve-ng reachability
# get the time so we can use it as a counter thanks to
https://unix.stackexchange.com/questions/52313/how-to-get-execution-time-
of-a-script-effectively idea
start=`date +%s`

# ping check logic from https://www.cyberciti.biz/tips/simple-linux-and-
unix-system-monitoring-with-ping-command-and-scripts.html
count=$(ping -c 5 10.140.1.199 | grep 'received' | awk -F',' '{ print $2
}' | awk '{ print $1 }')

while [ $count -eq 0 ]; do
    end=`date +%s`
    runtime=$((end-start))
    minutes=$((runtime/60))
    # use trick from
```

```
https://stackoverflow.com/questions/11283625/overwrite-last-line-on-
terminal
    echo -e "\e[1A\e[K eve-ng instance is not pinging yet, I've been
running for $((minutes)) minutes"
    sleep 5
    count=$(ping -c 5 10.140.1.199 | grep 'received' | awk -F',' '{ print
$2 }' | awk '{ print $1 }')
done
  if [ $count -gt 0 ]; then
    echo "Eve-ng is alive via the VPN tunnel! Press any key to shut
everything down."
  fi
```

If you delete the VPN tunnel under the GCP Main Menu -> Hybrid Connectivity -> VPN, then reboot OPNsense for good measure (this is not 100% necessary, but I like to do this to avoid “stale tunnel” headaches), you should be greeted with the following when running the script! In under a minute, our GCP eve-ng instance gets bought back to life, along with the VPN tunnel into the homelab:

Step 7: Wait for input to start teardown process

So you may have noticed that “Press any key to shut everything down” message above is fake news; our script just completes and leaves you at the terminal for now. To fix this, add the following into the end of eve-start.sh, it’ll simply wait for you to press a key before kicking off all the logic to gracefully shut down all the GCP things:

```
# step 7: Wait for input to start teardown process
read -n1 -s
```

Step 8: Gracefully shutdown GCP eve-ng instance with Ansible

Add the following to your hosts.ini, make sure to follow the comments regarding placement of the sections (or just look at the finished product on GitHub) and change the password accordingly:

```
# put this at the top underneath the [opnsense] section:
[eveng]
10.140.1.199

# put this at the bottom underneath the [opnsense:vars] section:
[eveng:vars]
ansible_user=root
ansible_ssh_pass='not_the_real_password'
ansible_become_method=sudo
```

This adds our eve-ng host to the Ansible inventory, giving us the ability to run ad-hoc commands. Since we just want to send a one-liner shutting down the eve-ng instance, add this into the bottom of eve-start.sh, no need for us to cook up a full playbook:

```
# step 8: shutdown eve-ng instance gracefully
ansible -i hosts.ini eveng -a "shutdown -h +1"
```

That shutdown syntax includes a delay of one minute, allowing plenty of time for Ansible to report back that the system will be shutting down shortly. A nice side effect of this is you also get a chance to abort the shutdown if the script kicks off accidentally. If you just do a "shutdown now -h", the system will turn off so quickly that Ansible will disconnect with an error message!

Step 9: Verify instance is down with Ansible

In a perfect world, our GCP eve-ng instance will get shutdown 100% of the time after issuing the above ad-hoc command. But what if the machine gets hung on shutdown? What if something crazy happens that results in the instance staying online forever? To mitigate this GCP billing risk, we'll "double-tap" the eve-ng instance, ensuring it's in a shutdown (aka terminated) state directly using another GCP Ansible playbook. Unlike the playbook that bought our eve-ng instance to life via status: RUNNING, this one ensures it's dead by ensuring it's set to status: TERMINATED! Make a new file named eve-powercheck.yml, and put the following into it:

```
---
- name: validate eve-ng instance is off
  hosts: localhost
  gather_facts: false
  tasks:
    - name: turn off eve-ng instance
      google.cloud.gcp_compute_instance:
        name: eveng
        status: TERMINATED
        zone: us-central1-a
        project: your_full_project_id
        auth_kind: serviceaccount
        service_account_file: your_gcp_json_file.json
```

As always, change the name of your instance, full project ID, etc. accordingly. Then add the following into the bottom of eve-start.sh:

```
# step 9: double tap the eve-ng instance
echo "Waiting 90 seconds before double checking the GCP eve-ng instance
is shutdown"
sleep 90
ansible-playbook eve-powercheck.yml
```

Step 10: Destroy GCP VPN tunnel with Ansible

We've finally reached the end of the road, time to teardown the GCP side of the OPNsense <> GCP tunnel so we don't get charged a dime while we're not using it. Create a final playbook named gcpvpndelete.yml, and get the following into it:

```
---
- name: Delete VPN tunnel to GCP
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Delete tunnel
      google.cloud.gcp_compute_vpn_tunnel:
        name: vpn-opnsense
        state: absent
        region: us-central1
        shared_secret: not_the_real_key
        project: your_full_project_id
        auth_kind: serviceaccount
        service_account_file: your_gcp_json_file.json
```

Add the final two lines to eve-start.sh too:

```
# step 10: Delete the VPN tunnel
ansible-playbook gcpvpndelete.yml
echo "Script complete, check GCP console if any above output looks
shady."
```

Conclusion

Rather than bore you with a summary of everything we've done in the above ten steps, I thought it would be more fun to do a demo video. Trust me, it's worth the 13ish minutes of your time if you have a GCP eve-ng instance of your own:

There's a lot of things that could be smoother in this homelab, especially regarding the Python code & Ansible playbooks. As many of you know, I blew off learning how to automate things far longer than I'd like to admit. That really shows in parts of this homelab, some of these Ansible playbooks are dirty 🙂 But that's ok, maybe someday I'll come back and clean up this lab, or maybe someone will put in a GitLab pull request to make everything less jankety. Regardless of where the future takes us, I really hope this lab saves someone a few dollars due to automatically shutting down a GCP eve-ng instance that would have otherwise been left idling. Please let me know in the comments if you're in that boat. Happy Homelabbing!

You've reached the end of the post! Click here to go back to the list of all Homelab posts.

Don't injure yourself sprinting towards IT certifications

Using NX-OS 10.1.2's new two-stage configuration commit with Netmiko

2 thoughts on "Automating startup/shutdown of a GCP eve-ng homelab with VPN tunnel"

Pingback: Google Cloud eve-ng setup with OPNsense homelab tunnel in 2021 - Tom Costello - KD9CPB

Pingback: Initial Setup: Homelab to Google Cloud eve-ng instance via ASA VTI - Tom Costello - KD9CPB

Leave a Reply

Logged in as kd9cpb. Log out?

Comment

POST COMMENT

This site uses Akismet to reduce spam. Learn how your comment data is processed.