GCP Transit VPC

VPC Peering & Internal Load Balancer as Next Hop

Matt McLimans
Public Cloud Consultant Engineer

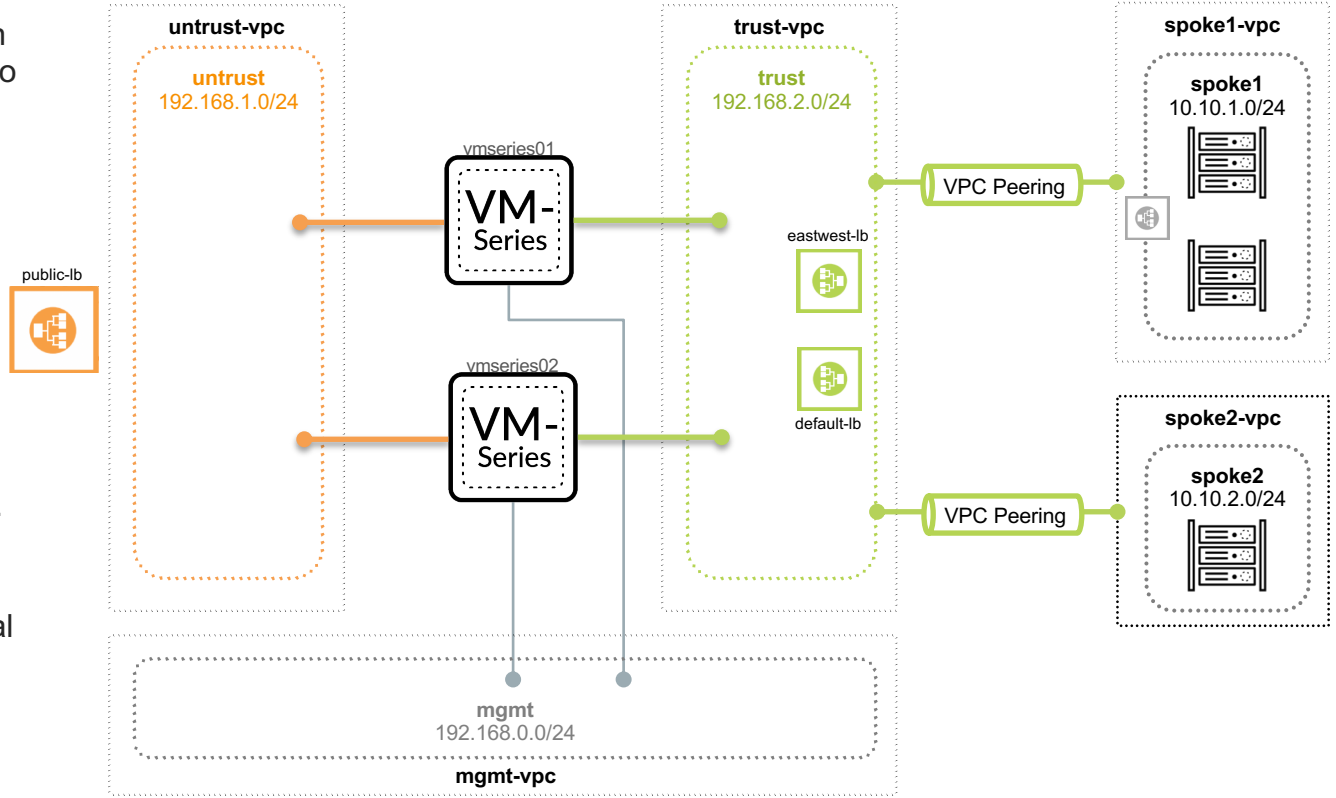
paloalto NETWORKS®

SUPPORT POLICY

This is released under an as-is, best effort, support policy. These scripts should be seen as community supported and Palo Alto Networks will contribute our expertise as and when possible. We do not provide technical support or help in using or troubleshooting the components of the project through our normal support options such as Palo Alto Networks support teams, or ASC (Authorized Support Centers) partners and backline support options. The underlying product used (the VM-Series firewall) by the scripts or templates are still supported, but the support is only for the product functionality and not for help in deploying or using the template or script itself. Unless explicitly tagged, all projects or work posted in our GitHub repository (at https://github.com/PaloAltoNetworks) or sites other than our official Downloads page on https://support.paloaltonetworks.com are provided under the best effort policy.

OVERVIEW

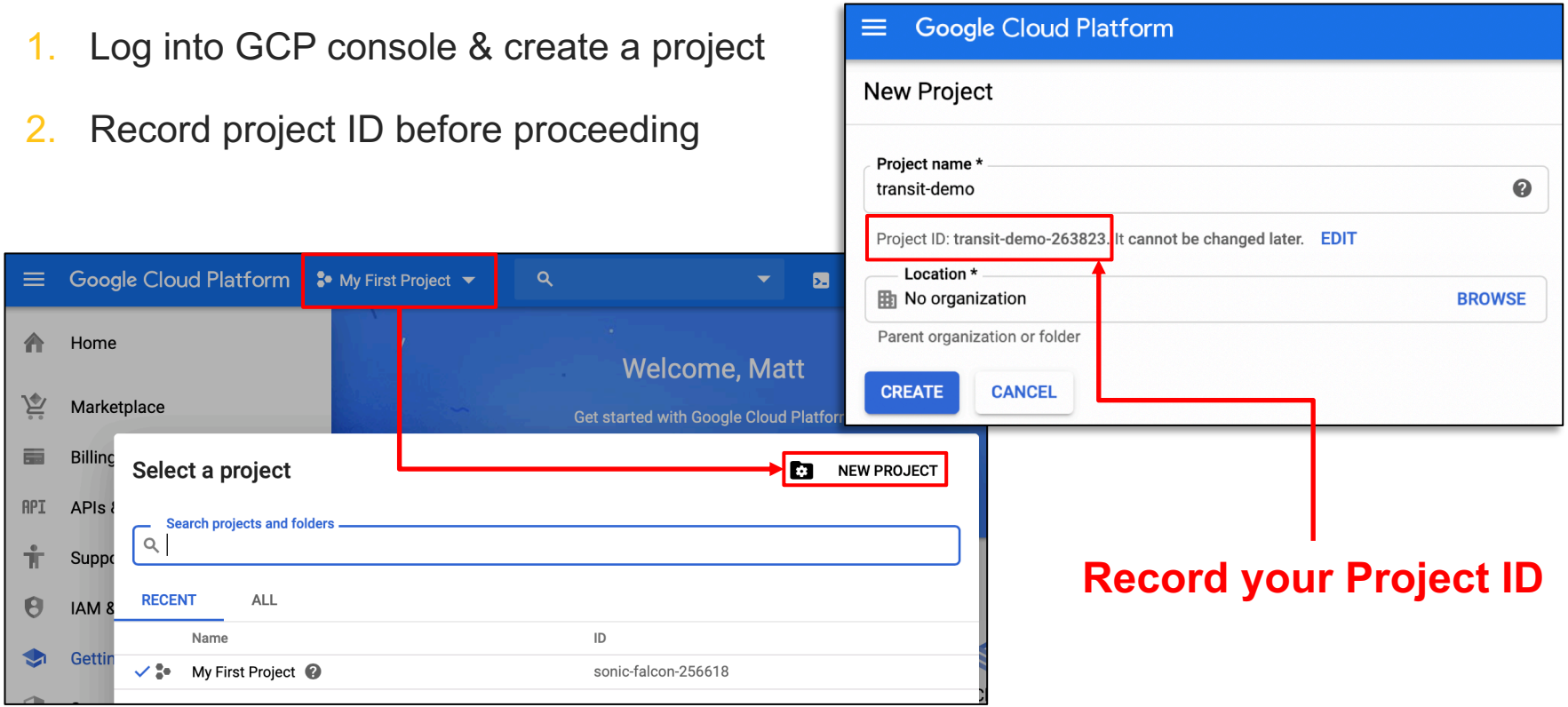
- 2 x VM-Series deployed between public & internal load balancers to secure inbound, outbound, and east-west traffic between two spoke VPCs.
- The spoke VPCs connect to the trust VPC via VPC peering.
- Spoke1 contains 2 x Web VMs with an internal load balancer.
- Spoke2 contains 1 x Ubuntu VM.
- Outbound and East-West traffic flows over the peer link to internal load balancers in the trust VPC.



PART 1
SETUP PROJECT
& BUILD ENVIRONMENT

STEP 1. CREATE A PROJECT

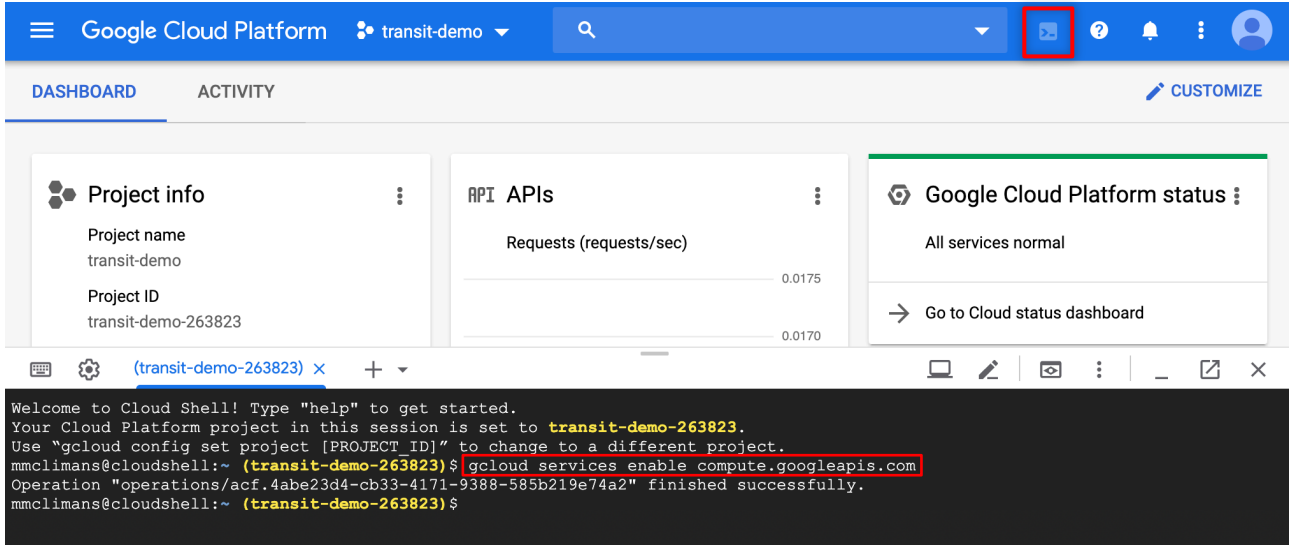
1. Log into GCP console & create a project
2. Record project ID before proceeding



STEP 2. ENABLE COMPUTE ENGINE API

1. Inside your project, open a cloud shell terminal
2. Enable the Compute Engine API

```
$ gcloud services enable compute.googleapis.com
```



STEP 3. DOWNLOAD REPO

1. Create an SSH key to enable user authentication to the VMs (password protected key is optional)

```
$ ssh-keygen -f ~/.ssh/gcp-demo -t rsa -C demo
```

```
mmclimans@cloudshell:~ (transit-demo-263823)$
mmclimans@cloudshell:~ (transit-demo-263823)$ ssh-keygen -f ~/.ssh/gcp-demo -t rsa -C demo
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mmclimans/.ssh/gcp-demo.
Your public key has been saved in /home/mmclimans/.ssh/gcp-demo.pub.
The key fingerprint is:
SHA256:8FrpSUW3V2ZaP5umXAWLEI/EYZlnECwEj1ibJEtYJjo demo
The key's randomart image is:
+---[RSA 2048]----+
|  .o= +o.oXB. . =|
| ..+ * =.+==oo O.|
|E   o = ..oo+ +.o|
| .     o o   .  =|
|        S      = |
|       = .  . +  |
|      . o    o   |
|                 |
|                 |
+----[SHA256]-----+
mmclimans@cloudshell:~ (transit-demo-263823)$
```

Record the ***public*** key's full path

STEP 4. DOWNLOAD REPO

1. Clone the Github repo & change directories to `adv_peering_2fw_2spoke_common/`

```
$ git clone https://github.com/wwce/terraform; cd terraform/gcp/adv_peering_2fw_2spoke_common
```

```
mmclimans@cloudshell:~ (transit-demo-263823)$ git clone https://github.com/wwce/terraform; cd terraform/gcp/adv_peering_2fw_2spoke_common
Cloning into 'terraform'...
remote: Enumerating objects: 214, done.
remote: Counting objects: 100% (214/214), done.
remote: Compressing objects: 100% (138/138), done.
remote: Total 5184 (delta 153), reused 131 (delta 75), pack-reused 4970
Receiving objects: 100% (5184/5184), 32.77 MiB | 41.71 MiB/s, done.
Resolving deltas: 100% (3108/3108), done.
mmclimans@cloudshell:~/terraform/gcp/adv peering 2fw 2spoke common (transit-demo-263823)$
```


STEP 5. UPDATE TERRAFORM.TFVARS

1. Open the terraform.tfvars file

```
$ nano terraform.tfvars
```

2. Uncomment (delete # at line start) & set values for **`project_id`**, **`public_key_path`**, & **`fw_panos`**

```
  GNU nano 2.7.4                File: terraform.tfvars                Modified

project_id      = "transit-demo-263823"   <- Project ID from Step 1
public_key_path = "~/.ssh/gcp-demo.pub"   <- SSH public key path from Step 3

#fw_panos       = "byol-904"
fw_panos        = "bundle1-904"           <- Uncomment only 1 line for VM-Series license option
#fw_panos       = "bundle2-904"


#-------------------------------------------------------------------------
```

3. Save file and exit: `ctrl-x` → `y` → `enter`

BYOL LICENSES ONLY

To license the firewall's during deployment perform the following. You can also manually license later.

1. Ensure authcode is registered with the Palo Alto Networks support site.
2. Open bootstrap_files/authcodes & enter your authcode.

```
$ nano bootstrap_files/authcodes
```

3. Save file and exit: `ctrl-x → y → enter`

STEP 6. DEPLOY BUILD

Initialize and apply the Terraform

```
$ terraform init
```

```
mmclimans@cloudshell:~/terraform/gcp/adv_peering_2fw_2spoke_common (transit-demo-263823)$ terraform init
Initializing modules...


Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
mmclimans@cloudshell:~/terraform/gcp/adv_peering_2fw_2spoke_common (transit-demo-263823)
```

```
$ terraform apply
```

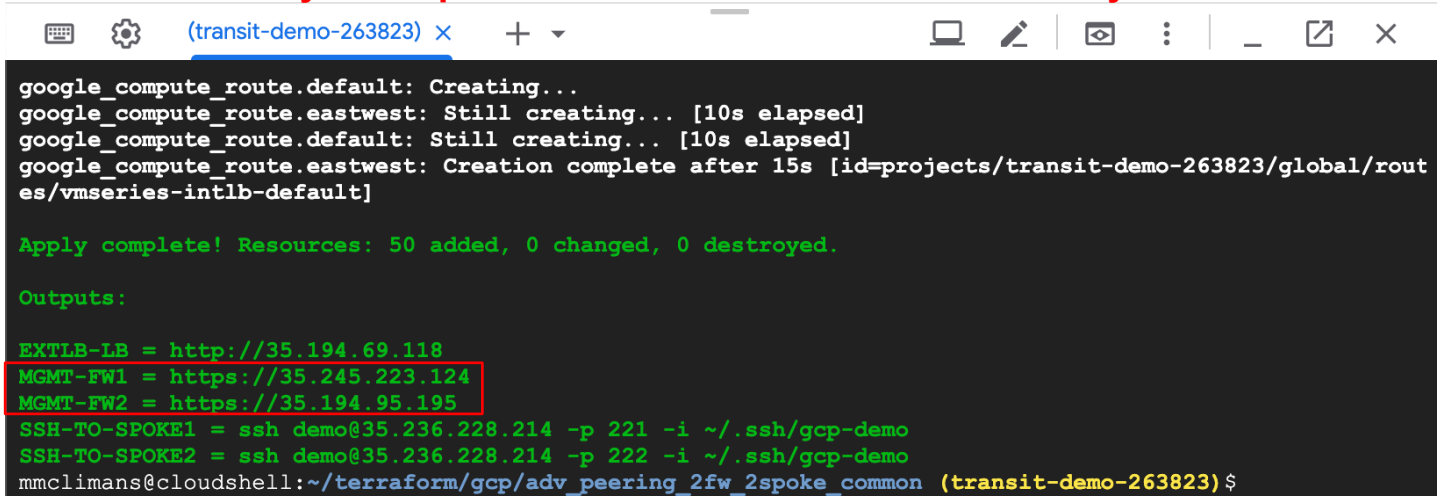
```
mmclimans@cloudshell:~/terraform/gcp/adv_peering_2fw_2spoke_common (transit-demo-263823)$ terraform apply
data.google_compute_zones.available: Refreshing state...


Plan: 50 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

← Enter yes to confirm

STEP 7. BUILD OUTPUT

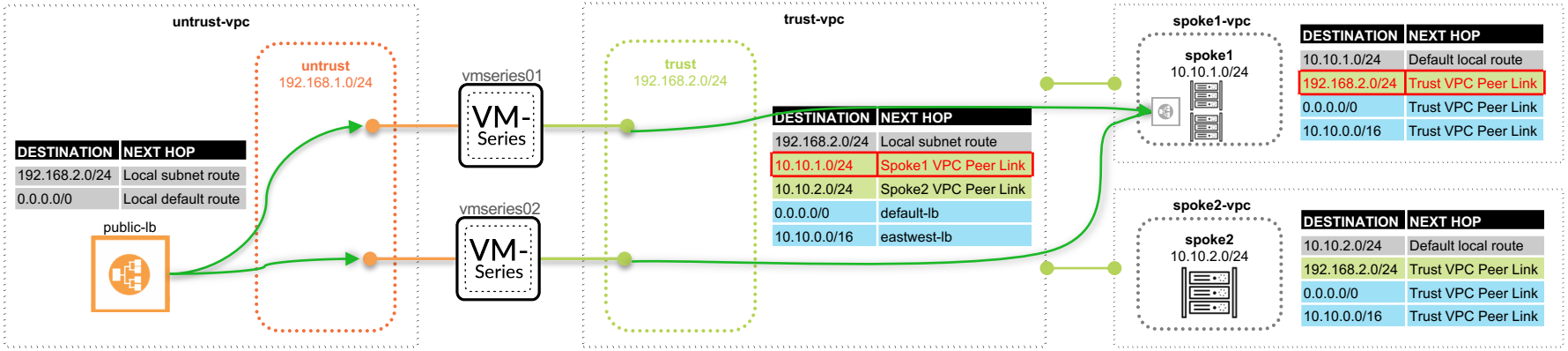
- Once the build completes, the following output will be displayed
- Log into each firewall by pasting the **MGMT-FW1** & **MGMT-FW2** outputs into a web-browser.
  - **UN:** `paloalto`
  - **PW:** `Pal0Alt0@123`

It may take up to 15 minutes for the firewalls to fully boot.



PART 2
TEST ENVIRONMENT

INBOUND FLOW



1. A request from the internet hits the public-lb's frontend.

2. The LB forwards the traffic to the firewall's untrust NICs.

3. The firewall's inspect and perform SNAT to trust interface and DNAT to spoke1's internal LB (**10.10.1.100**).

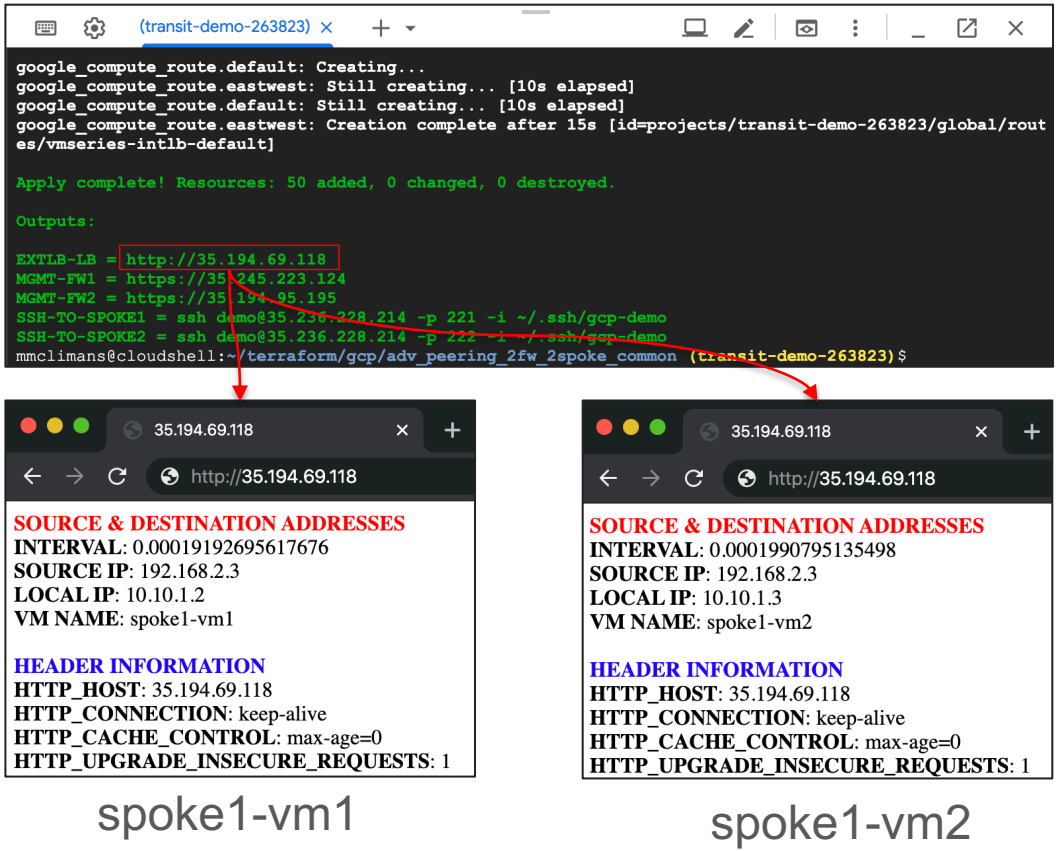
4. When the traffic leaves the trust interface, the spoke1-vpc peering route is taken.

5. The response back from spoke1 uses the trust-vpc peer route.

TEST INBOUND FLOW

1. Copy the **GLB_ADDR** output into a web browser.

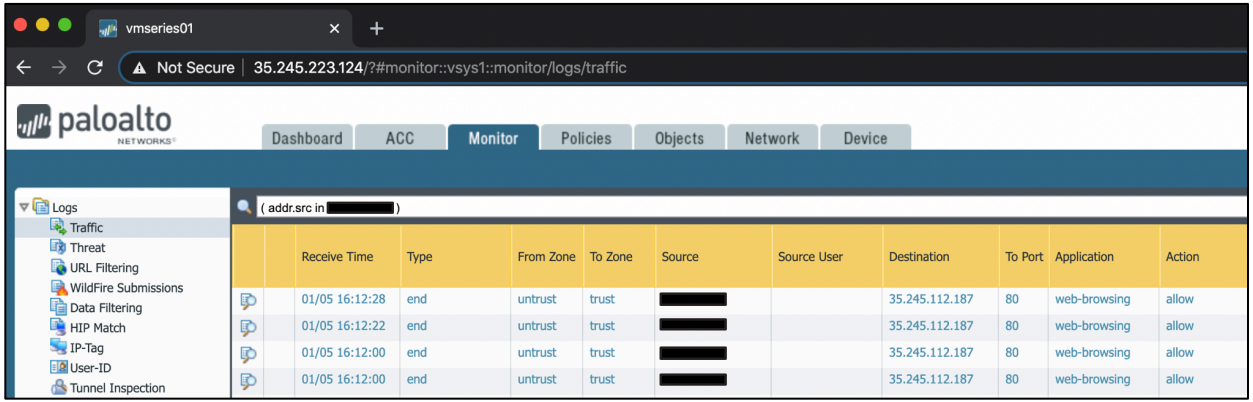
2. Observe inbound load balancing by refreshing the page. The value of **LOCAL IP** and **VM NAME** will change between the web VMs in spoke1.



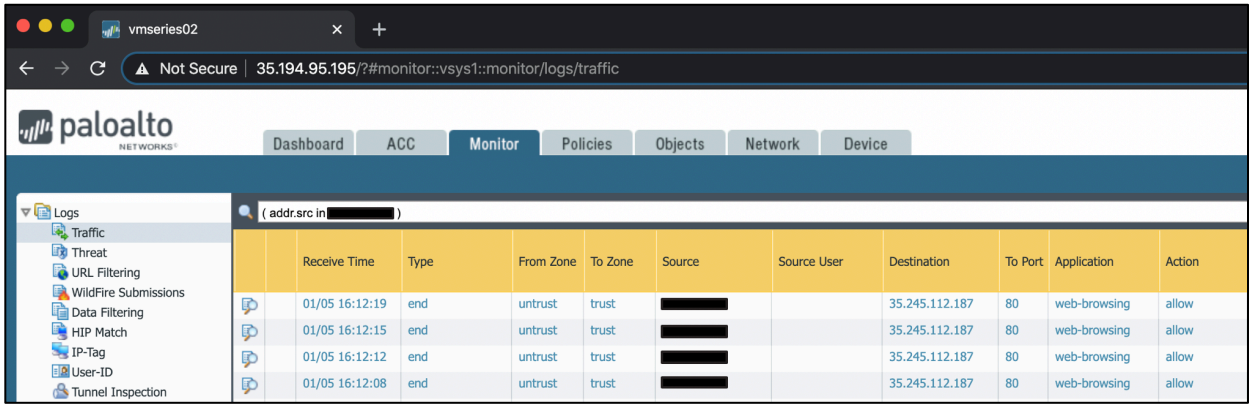
VIEW INBOUND TRAFFIC LOGS

- View the inbound traffic on both firewalls (Monitor → Traffic)

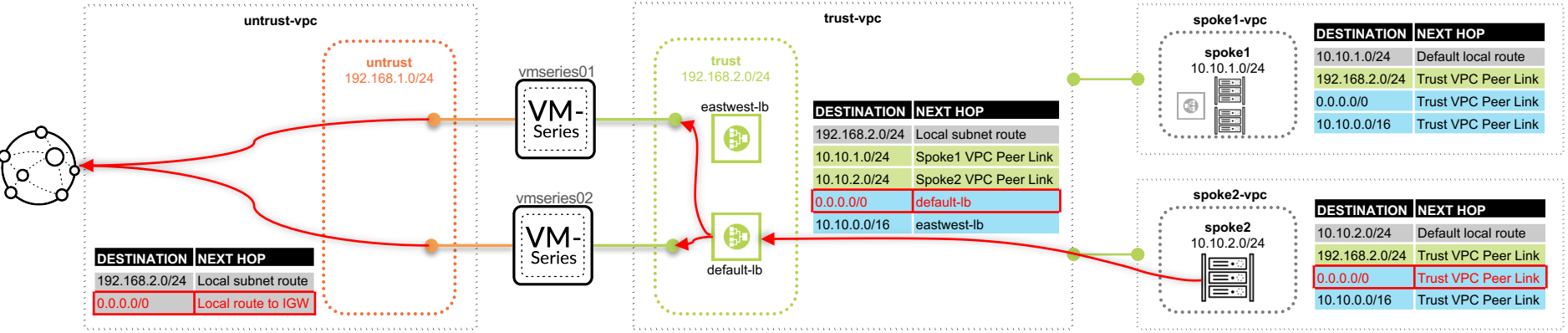
vmseries01



vmseries02



OUTBOUND FLOW



6. The firewall inspects and SNATs the traffic to the untrust interface.

7. When traffic leaves the untrust interface, it uses the untrust-vpc's default route to its internet gateway.

4. Traffic destined for internet from Spoke2 uses the custom static route to the default-lb.

5. The default-lb sends all TCP/UDP to the FW's trust interfaces (active/active)

1. Spoke2 VM (**10.10.2.4**) makes a request to internet.

2. Spoke2 uses route imported from the trust-vpc's route table.

3. The route's next-hop is to the trust-vpc's peering link.

TEST OUTBOUND FLOW

1. Log into spoke2-vm. Copy the **SSH-TO-SPOKE2** output and paste it into your terminal

2. From the spoke2-vm, test outbound connectivity:

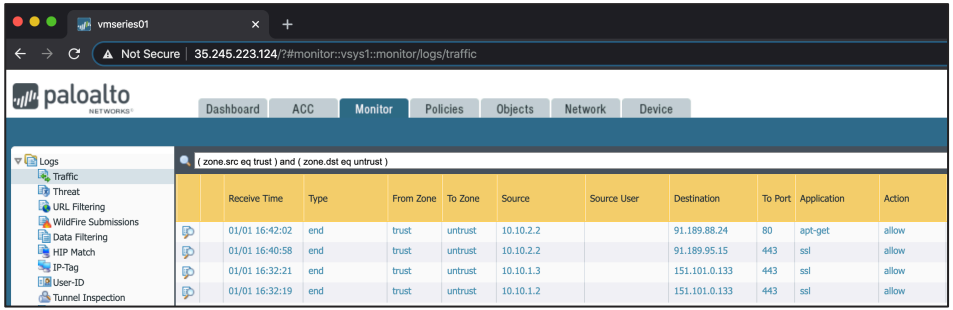
   ```
   $ sudo apt-get update
   ```

3. View the outbound traffic on both firewalls (Monitor → Traffic)

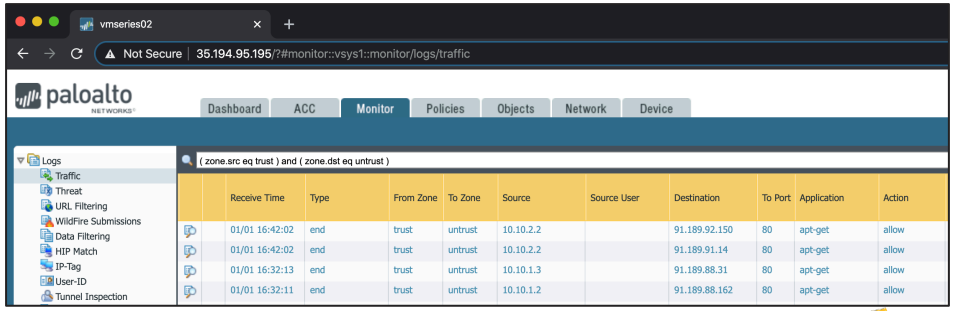
```
Apply complete! Resources: 50 added, 0 changed, 0 destroyed.

Outputs:

EXTLB-LB = http://35.194.69.118
MGMT-FW1 = https://35.245.223.124
MGMT-FW2 = https://35.194.95.195
SSH-TO-SPOKE1 = ssh demo@35.236.228.214 -p 221 -i ~/.ssh/gcp-demo
SSH-TO-SPOKE2 = ssh demo@35.236.228.214 -p 222 -i ~/.ssh/gcp-demo
```

```
(transit-demo-263823)$ ssh demo@35.236.228.214 -p 222 -i ~/.ssh/gcp-demo

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

demo@spoke2-vm1:~$ sudo apt-get update
```

vmseries01



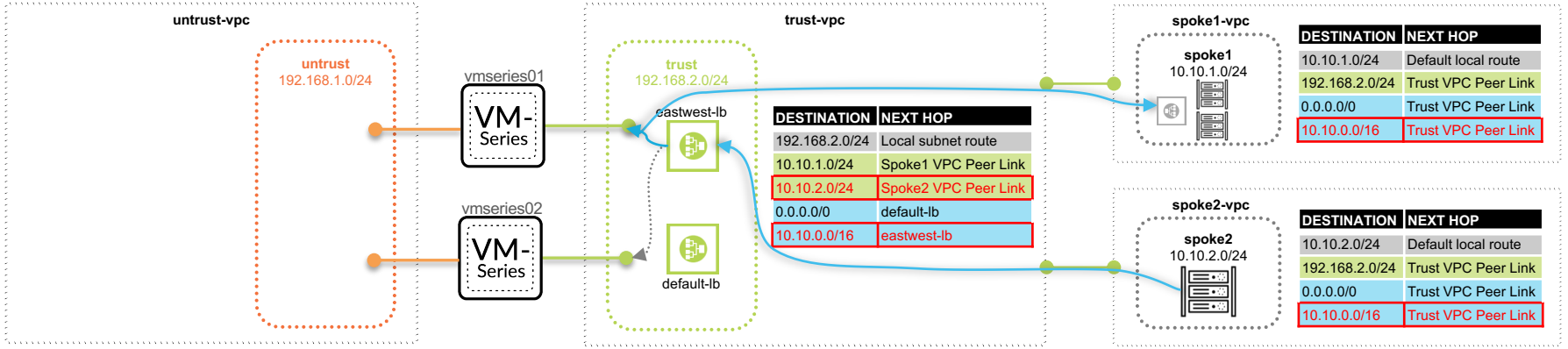
vmseries02



EAST-WEST FLOW

7. vmseries01 inspects and sends the traffic back out its trust NIC

8. The traffic then uses spoke1-vpc's peering link route as the next hop



4. When traffic arrives in the trust-vpc, the custom route to **10.10.0.0/16** to **eastwest-lb** is taken

5. The eastwest-lb sends all TCP/UDP to vmseries01

Note: The use of active/passive backends avoids SNAT for east/west traffic. The LB uses vmseries02 as a failover device (F/O determined by LB health-check).

1. Spoke2 VM (**10.10.2.4**) makes a request to internal LB in spoke1 (**10.10.1.100**)

2. Spoke2 uses the imported route from the trust-vpc's route table to **10.10.0.0/16**

3. The route's next-hop is to the trust-vpc peering link.

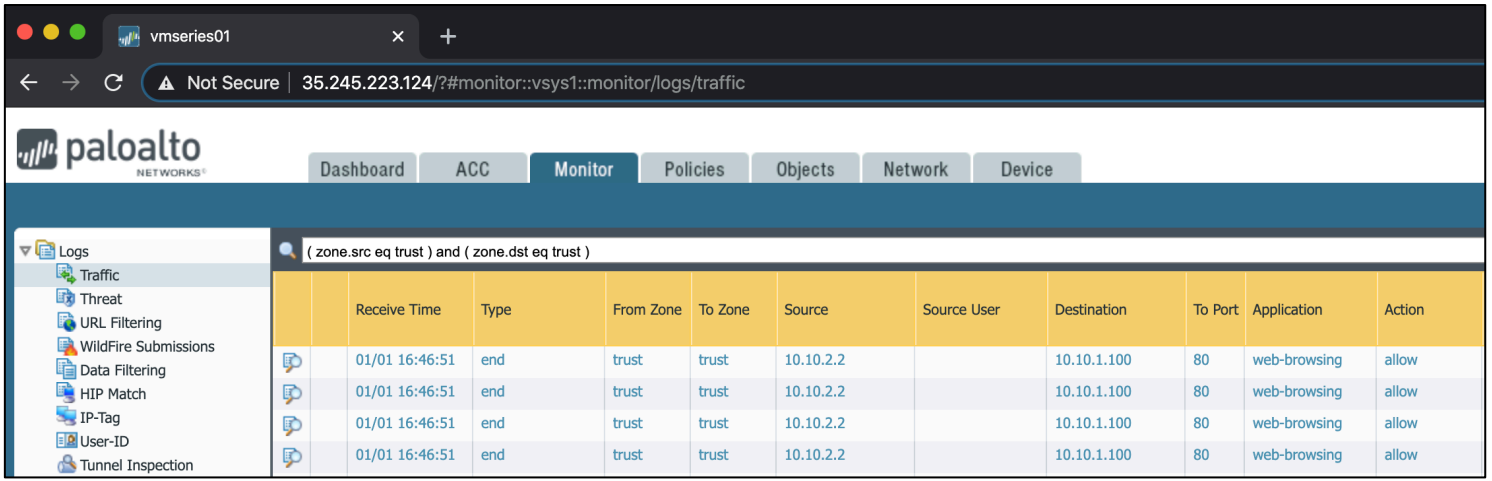
TEST EAST-WEST FLOW

1. Stay inside spoke2-vm1 (**10.10.2.4**)
2. Run a curl command to the web server's internal load balancer in spoke1 (**10.10.1.100**)

```
$ curl http://10.10.1.100/?[1-1000]
```

```
demo@spoke2-vm1:~$ curl http://10.10.1.100/?[1-1000]
```

3. View the east-west traffic on **vmseries01 only** (Monitor → Traffic)



DESTROY ENVIRONMENT

DESTROY BUILD

Perform the following to destroy the deployment.

1. Verify you are in the `adv_peering_4fw_2spoke` directory and run the following. Enter `yes` to confirm.

```
$ terraform destroy
```

```
mmclimans@cloudshell:~/terraform/gcp/adv_peering_2fw_2spoke_common (transit-demo-263823)$
mmclimans@cloudshell:~/terraform/gcp/adv_peering_2fw_2spoke_common (transit-demo-263823)$ terraform destroy
Plan: 0 to add, 0 to change, 50 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

2. Once the destroy is complete, you will receive the following output:

```
module.vpc_untrust.google_compute_network.default: Destruction complete after 15s
module.vpc_mgmt.google_compute_network.default: Destruction complete after 15s
module.vpc_trust.google_compute_network.default: Still destroying... [id=projects/transit-demo-263823/global/networks/trust-vpc, 20s elapsed]
module.vpc_trust.google_compute_network.default: Destruction complete after 26s

Destroy complete! Resources: 50 destroyed.
mmclimans@cloudshell:~/terraform/gcp/adv_peering_2fw_2spoke_common (transit-demo-263823)$
```

DEPLOYMENT CHEAT SHEET

Part 1: Setup Project

```
$ gcloud services enable compute.googleapis.com
$ ssh-keygen -f ~/.ssh/gcp-demo -t rsa -C demo
$ git clone https://github.com/wwce/terraform; cd terraform/gcp/adv_peering_2fw_2spoke_common
```

Part 2: Run Build

Edit terraform.tfvars to match project ID, SSH key, and PAN-OS license.

```
$ terraform init
$ terraform apply
```

Part 3: Destroy Build

```
$ terraform destroy
$ rm ~/.ssh/gcp-demo*
```

