Solutions (https://cloud.google.com/solutions/) Solutions

Automated Network Deployment: Building a VPN Between GCP and AWS

(Building a VPN between AWS and GCP with Terraform)

This tutorial shows you how to use Terraform (https://www.terraform.io/) by HashiCorp to create secure, private, site-to-site connections between Google Cloud Platform (GCP) and Amazon Web Services (AWS) using virtual private networks (VPNs). This is a multi-cloud deployment.

This tutorial is the third in a three-part series that demonstrates the automated deployment of common networking resource patterns. The tutorial relies on the authentication and project configuration described in the Automated Network Deployment: Overview (https://cloud.google.com/solutions/automated-network-deployment-overview) tutorial.

In this tutorial, you deploy virtual machine (VM) instances into custom virtual private cloud (VPC) networks in GCP and AWS. You then deploy supporting infrastructure to construct a VPN connection with two Internet Protocol security (IPsec) tunnels between the GCP and AWS VPC networks. The environment and tunnel deployment usually completes within four minutes.

Although this tutorial is an extension of the Automated Network Deployment: Startup (https://cloud.google.com/solutions/automated-network-deployment-startup) tutorial, it does not include a Deployment Manager configuration because resources are deployed to providers outside of GCP. Instead, to deploy resources using multiple public-cloud providers, including GCP, this tutorial uses Terraform configuration files. Multi-cloud deployments are beyond the intended scope of Deployment Manager.

Costs

This tutorial uses the following billable components of GCP and AWS:

- Compute Engine instances and Amazon Elastic Compute Cloud (Amazon EC2) instances
- Multi-vCPU instances to support higher network throughput
- Persistent disk and block storage
- Networking egress
- VPN tunnels

Outbound or egress traffic from VM instances is subject to maximum network egress throughput caps (https://cloud.google.com/vpc/docs/quota#per_instance). This tutorial uses multiple vCPU machine types (https://cloud.google.com/compute/docs/machine-types) to allow headroom for network egress traffic.

Use the pricing calculator (https://cloud.google.com/products/calculator/) to generate a cost estimate based on your projected usage.

Before you begin

1. Select the GCP project named `gcp-automated-networks`.

   GO TO THE PROJECTS PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECT)

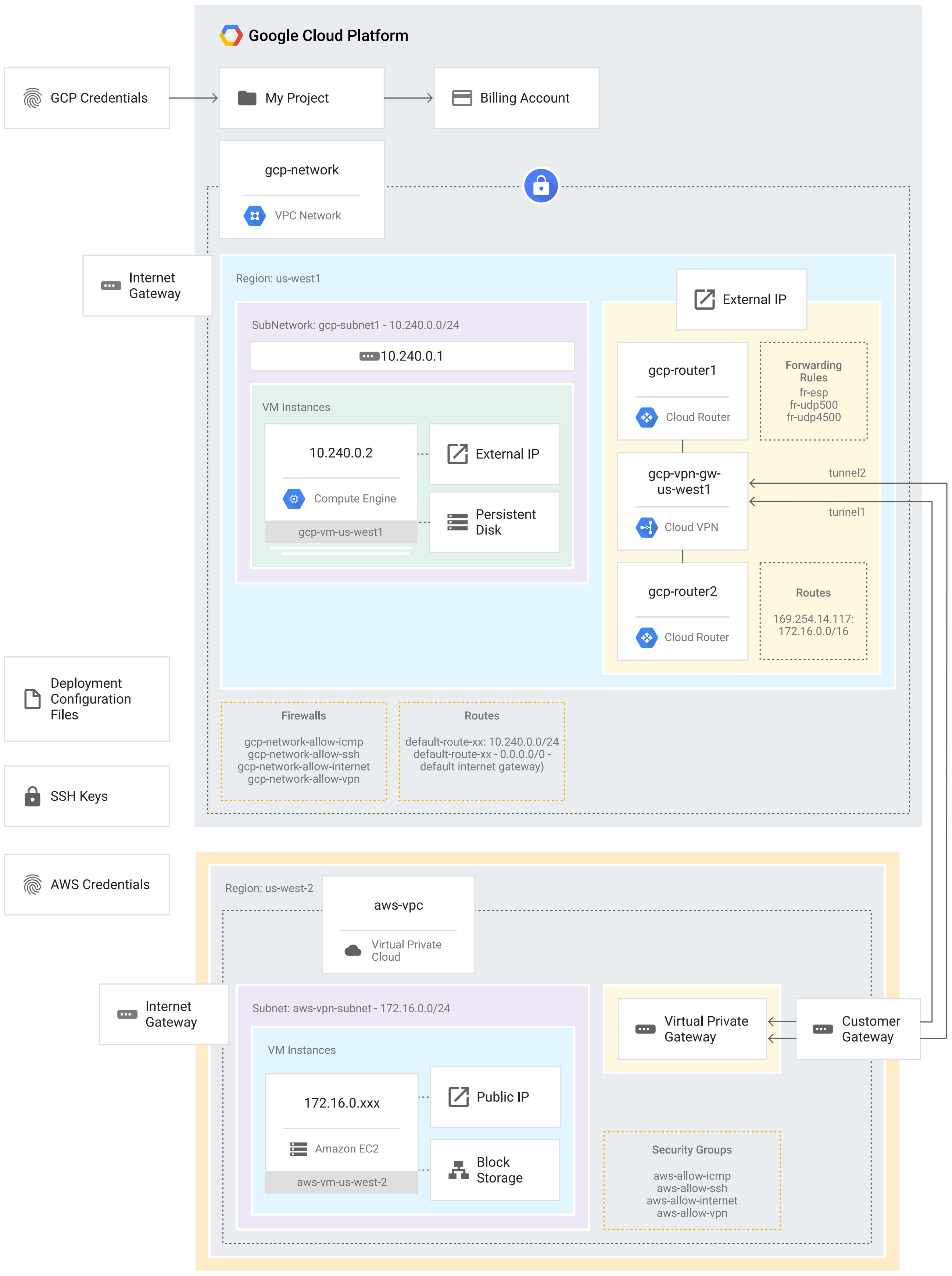
2. Start a Cloud Shell instance. You run all the terminal commands in this tutorial from Cloud Shell.

   OPEN CLOUD SHELL (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/?CLOUDSHELL=TRUE)

Note: This tutorial uses Cloud Shell to get you started quickly. You can also download the Cloud SDK (https://cloud.google.com/sdk/docs/), configure it, and run the command-line tool gcloud (https://cloud.google.com/sdk/gcloud/) from your local command environment.

Deployment architecture

In this tutorial, you build the following deployment environment.


Google Cloud Platform
GCP Credentials
My Project
Billing Account
gcp-network
VPC Network
Internet Gateway
Region: us-west1
SubNetwork: gcp-subnet1 - 10.240.0.0/24
10.240.0.1
VM Instances
10.240.0.2
Compute Engine
gcp-vm-us-west1
External IP
Persistent Disk
External IP
gcp-router1
Cloud Router
Forwarding Rules
fr-esp
fr-udp500
fr-udp4500
gcp-vpn-gw-us-west1
Cloud VPN
tunnel2
tunnel1
gcp-router2
Cloud Router
Routes
169.254.14.117:
172.16.0.0/16
Deployment Configuration Files
Firewalls
gcp-network-allow-icmp
gcp-network-allow-ssh
gcp-network-allow-internet
gcp-network-allow-vpn
Routes
default-route-xx: 10.240.0.0/24
default-route-xx - 0.0.0.0/0 - default internet gateway)
SSH Keys
AWS Credentials
Region: us-west-2
aws-vpc
Virtual Private Cloud
Internet Gateway
Subnet: aws-vpn-subnet - 172.16.0.0/24
VM Instances
172.16.0.xxx
Amazon EC2
aws-vm-us-west-2
Public IP
Block Storage
Virtual Private Gateway
Customer Gateway
Security Groups
aws-allow-icmp
aws-allow-ssh
aws-allow-internet
aws-allow-vpn

This tutorial guides you through:

- Building custom VPC networks (https://cloud.google.com/vpc/) with user-specified CIDR blocks in GCP and AWS.
- Deploying a VM instance (https://cloud.google.com/compute/docs/instances/) in each VPC network.
- Creating VPN gateways (https://cloud.google.com/compute/docs/vpn/overview/) in each VPC network and related resources for two IPsec tunnels.

While GCP uses routes to support equal-cost multi-path (ECMP (https://en.wikipedia.org/wiki/Equal-cost_multi-path_routing)) routing, AWS supports VPN gateways with two tunnels, active and standby, for redundancy and availability.

Routing

The tutorial configuration uses Cloud Router (https://cloud.google.com/compute/docs/cloudrouter) to demonstrate dynamic routing. Cloud Router exchanges your VPC network route updates with your environment in AWS using Border Gateway Protocol (BGP) (https://en.wikipedia.org/wiki/Border_Gateway_Protocol). Dynamic routing by Cloud Router requires a separate Cloud Router for each IPsec tunnel. Alternatively, you can configure a setup with static routes. Both configurations are covered in the Cloud VPN Interop Guide (https://cloud.google.com/files/CloudVPNGuide-UsingCloudVPNwithAmazonWebServices.pdf).

Preparing your GCP working environment

Note: This tutorial relies on the environment configured by the Overview tutorial (https://cloud.google.com/solutions/automated-network-deployment-overview).

In this section, you:

- Clone the tutorial code.
- Verify your GCP region and zone.

Clone the tutorial code

1. Clone the tutorial code from GitHub (https://github.com/):

```
git clone https://github.com/GoogleCloudPlatform/autonetdeploy-multicloudvpn.gi
```

2. Navigate to the tutorial directory:

```
cd autonetdeploy-multicloudvpn
```

Verify the GCP region and zone

Certain cloud resources in this tutorial, including Compute Engine instances, VPN gateways, and Cloud Router, require you to explicitly declare the intended placement region or zone, or both. For more details, see Regions and Zones for GCP (https://cloud.google.com/compute/docs/regions-zones/regions-zones).

This tutorial requires only a single region for each provider. To optimize the connection between the two clouds, choose regions near each other. The following table lists the values set in the tutorial files `terraform/gcp_variables.tf` and `terraform/aws_variables.tf`.

Field name	GCP values	AWS values
`Region Name`	`us-west1`	`US West (us-west-2)`
`Location`	`The Dalles, Oregon, USA`	`Oregon`

Preparing for AWS use

In this section, you verify your AWS region. For details about AWS regions, see Regions and Availability Zones for AWS (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html).

1. Sign in to the AWS Management Console (https://console.aws.amazon.com/), and navigate to the **VPC Dashboard**. Select the **Oregon** region using the pulldown menu.
2. In the **EC2 Dashboard** and the **VPC Dashboard**, review the resources used by this tutorial.

Preparing Terraform

In this section, you download the Terraform executable.

- In Cloud Shell, run the following script:

```
./get_terraform.sh
```

  This script downloads and unpacks the executable binary of the Terraform tool to the `~/terraform` directory. The script output shows an export command to update your `PATH`. After you update your `PATH`, verify that Terraform is working:

```
terraform --help
```

  Resulting output:

```
Usage: terraform [--version] [--help] [command] [args]
...
```

If you need help, see the topics Download Terraform (https://www.terraform.io/downloads.html) and Install Terraform (https://www.terraform.io/intro/getting-started/install.html).

Creating GCP and AWS access credentials

You created credentials in the Overview tutorial (https://cloud.google.com/solutions/automated-network-deployment-overview). However, you must register your credentials with Terraform.

1. Register your GCP credentials with Terraform:

```
./gcp_set_credentials.sh exists
```

   Resulting output:

```
Updated gcp_credentials_file_path in ~/autonetdeploy-multicloudvpn/terraform/te
```

2. Register your AWS credentials with Terraform:

```
./aws_set_credentials.sh exists
```

Resulting output:

```
Updated aws_credentials_file_path in ~/autonetdeploy-multicloudvpn/terraform/te
```

Setting your project

GCP offers several ways to designate the GCP project to be used by the automation tools. For simplicity, instead of pulling the project ID from the environment, the GCP project is explicitly identified by a string variable in the template files.

1. Set your project ID:

   ```
   gcloud config set project [YOUR-PROJECT-ID]
   ```

   Resulting output:

   ```
   Updated property [core/project].
   ```

2. Use the provided script to update the project value in your configuration file for Terraform.

   ```
   ./gcp_set_project.sh
   ```

   Resulting output:

   ```
   Updated gcp_project_id in /home/[USER]/autonetdeploy-gcpawsvpn/terraform/terraf
   ```

3. Review the updated file, `terraform/terraform.tfvars`, to verify that your `project-id` has been inserted.

4. Run the one-time `terraform init` (https://www.terraform.io/docs/commands/init.html) command to install the Terraform providers for this deployment.

   ```
   pushd ./terraform && terraform init && popd > /dev/null
   ```

5. Run the Terraform `plan` command to verify your credentials.

   ```
   pushd ./terraform && terraform plan && popd > /dev/null
   ```

If you don't see red error text, your authentication is working properly.

Resulting output:

```
Refreshing Terraform state in-memory prior to plan...
...
 +google_compute_instance.gcp-vm
...
Plan: 34 to add, 0 to change, 0 to destroy.
```

Examining Terraform configuration files

In Terraform, a deployment configuration (https://www.terraform.io/intro/getting-started/build.html) is defined by a directory of files. Although these files can be JSON files, it's better to use the Terraform configuration file (https://www.terraform.io/docs/configuration/index.html) (`.tf`file) syntax, which is easier to read and maintain. This tutorial provides a set of files that illustrate one way of cleanly organizing your resources. This set is a functional deployment and requires no edits to run.

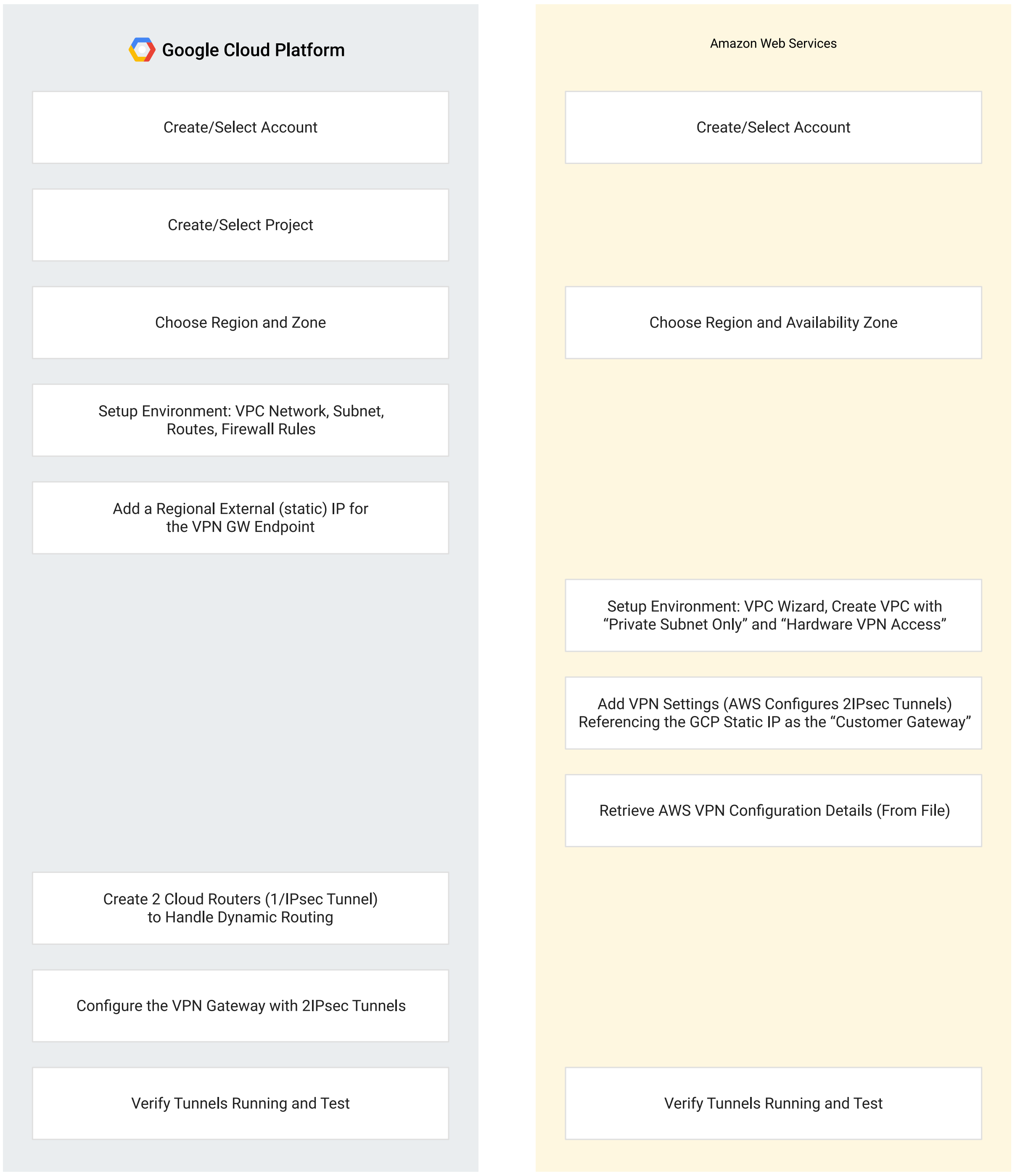
Filename	Purpose
`main.tf`	Defines your providers, and specifies which clouds to deploy in this configuration. Also reads your credentials, project name, and selected regions.
`gcp_variables.tf`, `aws_variables.tf`	Declares variables used to parameterize and customize the deployment—for example, `gcp_region` and `gcp_instance_type`.
`gcp_compute.tf`, `aws_compute.tf`	Defines the compute resources used in your deployment—for example, `google_compute_instance`.
`vm_userdata.sh`	Specifies the script to run when starting up VM instances. Automatically sets up the `iperf3` test tool and some wrapper scripts.
`gcp_networking.tf`, `aws_networking.tf`	Defines networking resources, including `google_compute_network`, `google_compute_subnetwork`, `google_compute_address`, `google_compute_vpn_gateway`, and `google_compute_vpn_tunnel`.
`gcp_security.tf`, `aws_security.tf`	Defines resources for allowing test traffic in the GCP or AWS environment, including `google_compute_firewall` rules and `aws_security_group` resources.

Filename	Purpose
`gcp_outputs.tf`, `aws_outputs.tf`	Defines variables to be output upon completion of the deployment—for example, the **`external_ip`** and **`internal_ip`** of the deployed VM instance.
`terraform.tfstate`	[OUTPUT]. Specifies the file used by Terraform to store the client-side resource state after checking with the cloud. For more details, see Managing GCP Projects with Terraform (https://cloud.google.com/community/tutorials/managing-gcp-projects-with-terraform) .
`run_graph.sh`	Shell script for generating a PNG file from Terraform that shows resource dependencies. You can see the output of this script in images/gcpawsvpn_plan_graph.png (https://github.com/GoogleCloudPlatform/autonetdeploy-multicloudvpn/blob/master/images/gcpawsvpn_plan_graph.png) .

In this tutorial, you use scripts to create the `terraform.tfvars` file, which includes user-specific setup for `credentials` and `gcp_project_id`.

Deploying VPC networks, VM instances, VPN gateways, and IPsec tunnels

Constructing connections between multiple clouds is complex. You can deploy many resources in parallel in both environments, but when you are building IPsec tunnels, you need to order interdependencies carefully. For this reason, establishing a stable deployment configuration in code is a helpful way to scale your deployment knowledge. The following figure summarizes the steps required to create this deployment configuration across multiple providers.


Google Cloud Platform
Create/Select Account
Create/Select Project
Choose Region and Zone
Setup Environment: VPC Network, Subnet, Routes, Firewall Rules
Add a Regional External (static) IP for the VPN GW Endpoint
Create 2 Cloud Routers (1/IPsec Tunnel) to Handle Dynamic Routing
Configure the VPN Gateway with 2IPsec Tunnels
Verify Tunnels Running and Test
Amazon Web Services
Create/Select Account
Choose Region and Availability Zone
Setup Environment: VPC Wizard, Create VPC with “Private Subnet Only” and “Hardware VPN Access”
Add VPN Settings (AWS Configures 2IPsec Tunnels) Referencing the GCP Static IP as the “Customer Gateway”
Retrieve AWS VPN Configuration Details (From File)
Verify Tunnels Running and Test

Deploy with Terraform

Terraform uses the `terraform.tfstate` file to capture the resource state. To view the current resource state in a readable form, you can run `terraform show` (https://www.terraform.io/docs/commands/show.html).

In order for Terraform to authenticate correctly with your providers, you must first complete the Overview tutorial (https://cloud.google.com/solutions/automated-network-deployment-overview). The following steps assume that you have already configured Terraform.

1. In Cloud Shell, navigate to the `terraform` directory:

   ```
   pushd terraform
   ```

2. Use the Terraform `validate` (https://www.terraform.io/docs/commands/validate.html) command to validate the syntax of your configuration files. This validation check is simpler than those performed as part of the `plan` and `apply` commands in subsequent steps. The `validate` command does not authenticate with any providers.

   ```
   terraform validate
   ```

   If you don't see an error message, you have completed an initial validation of your file syntax and basic semantics. If you do see an error message, validation failed.

3. Use the Terraform `plan` (https://www.terraform.io/docs/commands/plan.html) command to review the deployment without instantiating resources in the cloud. The `plan` command requires successful authentication with all providers specified in the configuration.

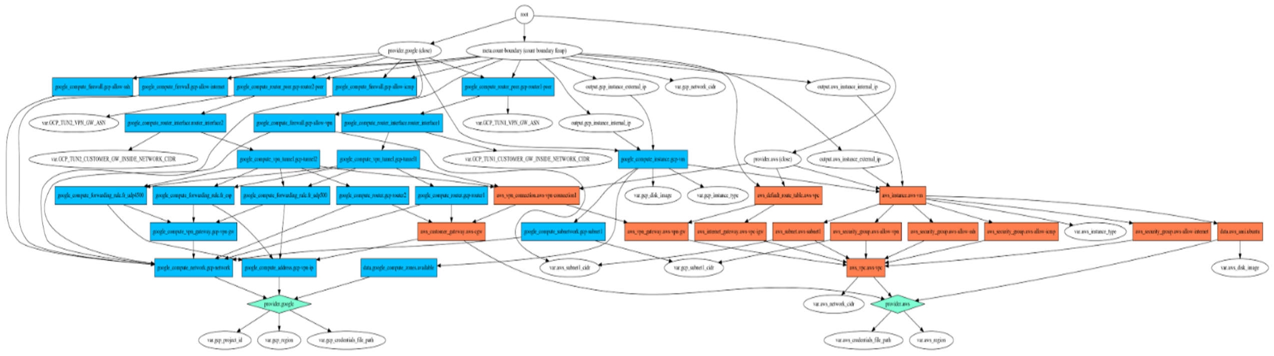
   ```
   terraform plan
   ```

   The `plan` command returns an output listing of resources to be added, removed, or updated. The last line of the `plan` output shows a count of resources to be added, changed, or destroyed:

   ```
   Refreshing Terraform state in-memory prior to plan...
   ...
   Plan: 34 to add, 0 to change, 0 to destroy.
   ```

4. Optionally, visualize your resource dependencies by using the Terraform `graph` (https://www.terraform.io/docs/commands/graph.html) command. The dependency graph helps you analyze your deployed resources. You can view a previously prepared version of the output file at `images/gcpawsvpn_plan_graph.png.`

The `run_graph.sh` script creates the `PNG` file `./gcpawsvpn_plan_graph.png`, which looks similar to the following:



The `run_graph.sh` script relies on the [`graphviz`](http://www.graphviz.org/) (http://www.graphviz.org/) package. If `graphviz` is not installed, you will see a `dot: command not found` message. In Cloud Shell, you can install `graphviz` by using the following command:

```
sudo apt-get install graphviz
```

5. Use the Terraform [`apply`](https://www.terraform.io/docs/commands/apply.html) (https://www.terraform.io/docs/commands/apply.html) command to create a deployment:

```
terraform apply
```

The `apply` command creates a deployment with backing resources in the cloud. In around four minutes, `apply` creates 30+ resources for you, including GCP and AWS VPC networks (https://www.terraform.io/docs/providers/google/r/compute_network.html), VM instances (https://www.terraform.io/docs/providers/google/r/compute_instance.html), VPN gateways (https://www.terraform.io/docs/providers/google/r/compute_vpn_gateway.html), and IPsec tunnels (https://www.terraform.io/docs/providers/google/r/compute_vpn_tunnel.html). The output of the `apply` command includes details of the resources deployed and the output variables defined by the configuration.

```
data.google_compute_zones.available: Refreshing state...
...
Apply complete! Resources: 34 added, 0 changed, 0 destroyed.
...
Outputs:


aws_instance_external_ip = [AWS_EXTERNAL_IP]
aws_instance_internal_ip = 172.16.0.100
```

```
gcp_instance_external_ip = [GCP_EXTERNAL_IP]
gcp_instance_internal_ip = 10.240.0.100
```

6. If you need to update the expected end state of your configuration, edit your `.tf` files. Add port 23 to your `google_compute_firewall gcp_allow-ssh` rule by editing `gcp_security.tf`:

   ```
   ports = ["22", "23"]
   ```

   Terraform analyzes the edits and identifies the minimum changes needed to update your deployment state to match.

   ```
   terraform apply
   ```

   Terraform modifies `gcp-network-gcp-allow-ssh` to the updated configuration.

7. Your deployments can emit output variables to aid your workflow. In this tutorial, the assigned internal and external IP addresses of VM instances have been identified as output variables by the `gcp_outputs.tf` and `aws_outputs.tf` files. These addresses are printed automatically when the `apply` step completes. If, later in your workflow, you want to redisplay the output variable values, use the `output` (https://www.terraform.io/docs/commands/output.html) command.

   ```
   terraform output
   ```

   The output variables defined by this configuration include the internal and external IP addresses for your VM instances. To use the `ssh` command for network validation, you need the external IP addresses to connect to the VM instances.

   ```
   aws_instance_external_ip = [AWS_EXTERNAL_IP]
   aws_instance_internal_ip = 172.16.0.100
   gcp_instance_external_ip = [GCP_EXTERNAL_IP]
   gcp_instance_internal_ip = 10.240.0.100
   ```

8. Use the Terraform show (https://www.terraform.io/docs/commands/show.html) command to inspect the deployed resources and verify the current state.

   ```
   terraform show
   ```

Resulting output:

```
...
google_compute_instance.gcp-vm:
...
Outputs:
...
```

9. To review your instances, use `gcloud compute` `instances list` (https://cloud.google.com/sdk/gcloud/reference/compute/instances/list?hl=bg): or use the Cloud Console, on the **VM instances** panel.

```
gcloud compute instances list
```

Resulting output:

```
NAME              ZONE        MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP    EXTERNAL
gcp-vm-us-west1  us-west1-a  n1-highmem-8              10.240.0.100   [EXTERNA
```

10. Verify that your GCP VM instance is functioning by using the `ssh` command to connect to it:

```
ssh -i ~/.ssh/vm-ssh-key [GCP_EXTERNAL_IP]
```

11. Run the `ping` and `curl` commands in your `ssh` session:

```
ping -c 5 google.com
curl ifconfig.co/ip
```

12. Run simple network performance checks from the GCP VM instance. Use pre-installed scripts to run a test on each network interface, both external and internal.

    a. Over external IPs:

    ```
    /tmp/run_iperf_to_ext.sh
    ```

    This script runs a 30-second performance test that produces summary data about network performance.

    Resulting output:

```
...
[SUM]   ... sender
[SUM]   ... receiver
...
```

b. Over VPN (internal IPs):

```
/tmp/run_iperf_to_int.sh
```

This script runs a 30-second performance test that produces summary data about network performance.

```
...
[SUM]   ... sender
[SUM]   ... receiver
...
```

13. When you complete the checks from the GCP VM instance, enter the following command:

```
exit
```

14. To verify that your AWS VM instance is functioning, use the `ssh` command to connect to it:

```
ssh -i ~/.ssh/vm-ssh-key ubuntu@[AWS_EXTERNAL_IP]
```

15. Run the `ping` and `curl` commands in your `ssh` session:

```
ping -c 5 google.com
curl ifconfig.co/ip
```

16. Run simple network performance checks from the AWS VM instance. Use pre-installed scripts to run a test on each network interface, both external and internal.

a. Over external IPs:

```
/tmp/run_iperf_to_ext.sh
```

This script runs a 30-second performance test producing summary data about network performance.

```
...
[SUM]   ... sender
[SUM]   ... receiver
...
```

b. Over VPN (internal IPs):

```
/tmp/run_iperf_to_int.sh
```

This script runs a 30-second performance test producing summary data about network performance.

```
...
[SUM]   ... sender
[SUM]   ... receiver
...
```

17. When you complete the checks from the AWS VM instance, enter the following command:

```
exit
```

You have successfully deployed a secure, private, site-to-site connection between GCP and AWS using VPN.

Cleaning up

Clean up the deployed resources. You will continue to be billed for your VM instances until you run the `destroy` deployment command.

1. Run the optional `plan -destroy` command to review the resources affected by `destroy`:

```
terraform plan -destroy
```

Resulting output:

```
Refreshing Terraform state in-memory prior to plan...
...
Plan: 0 to add, 0 to change, 34 to destroy.
```

2. Because the **destroy** (https://www.terraform.io/docs/commands/destroy.html) command will permanently delete your resources, you must confirm your intentions by entering `yes`. The `destroy` command usually completes in around 100 seconds:

   ```
   terraform destroy
   ```

   Resulting output:

   ```
   Do you really want to destroy?
     Terraform will delete all your managed infrastructure.
     There is no undo. Only 'yes' will be accepted to confirm.
       Enter a value: yes
   ```

   Enter `yes` to confirm the destruction of the resource you created.

   ```
   Destroy complete! Resources: 34 destroyed.
   ```

3. Run the show (https://www.terraform.io/docs/commands/show.html) command to display the state of the resources:

   ```
   terraform show
   ```

   Because you destroyed all resources, the `show` command outputs no lines, indicating that no resources remain deployed.

4. Finally, restore your directory:

   ```
   popd > /dev/null
   ```

You have now successfully completed deployment and cleanup of a secure, private, site-to-site connection between instances in GCP and AWS.

What's next

- To learn about a more advanced method of storing your Terraform state file in Cloud Storage, read Managing GCP Projects with Terraform (https://cloud.google.com/community/tutorials/managing-gcp-projects-with-terraform).

- To learn more about Open Cloud, read What if you could run the same, everywhere? (How to escape lock-in with a multi-cloud stack) (https://cloudplatform.googleblog.com/2016/07/how-to-escape-lock-in-with-a-multi-cloud-stack26.html)
.
- To learn more about Open Cloud and hybrid setups, watch the Day 3 Keynote (with Vint Cerf and Sam Ramji) (https://www.youtube.com/watch?v=h9FSqVbdHis) at Google Cloud Next from March 2017.
- To review details on Cloud VPN, see:
  - Cloud VPN Overview (https://cloud.google.com/compute/docs/vpn/overview)
  - Creating a VPN (https://cloud.google.com/compute/docs/vpn/creating-vpns#configure_firewall_rules)
  - VPN Advanced Configurations (https://cloud.google.com/compute/docs/vpn/advanced)
  - Cloud VPN Interop Guide: Using Cloud VPN with Amazon Web Services Virtual Private Gateway (https://cloud.google.com/files/CloudVPNGuide-UsingCloudVPNwithAmazonWebServices.pdf)
- Try out other Google Cloud Platform features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).