



Enterprise Database Migration

Networking for Secure Database Connectivity

Julianne Cuneo
Data Analytics Specialist,
Google Cloud



Hello, my name is Julianne Cuneo and I'm a Data Analytics Specialist Customer Engineer at Google. Welcome back to Enterprise Database Migration.

Security should always be a top concern, so let's have a look at how to build a secure network for database connectivity.

Learning objectives

- Build secure networks to host databases and database client applications.
- Allow secure communication across networks using VPC Peering, VPNs, and Interconnect.
- Control access to databases using firewall rules.
- Automate network infrastructure using Terraform.



There are various levels of network security we need to focus on: first building a secure network for the database servers, and then the client applications that will connect to them.

Next, in cases where you need to communicate across networks, you can use VPNs and VPC peering.

As always, firewall rules are a useful way to control access to the databases.

And throughout this module, you will automate building the network infrastructure using Terraform.

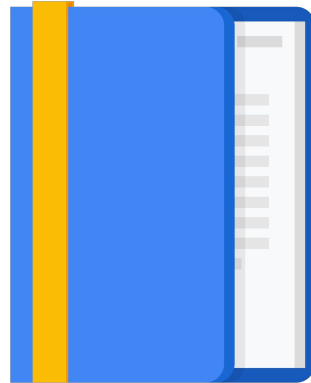
Agenda

Building Secure Networks

Connecting Networks

Enabling Communication across
Networks

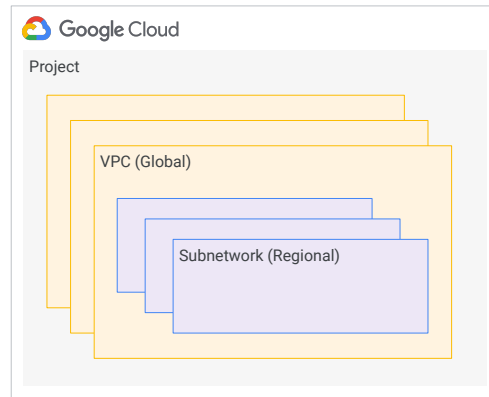
Network Considerations for
Managed Databases



Building a secure network to host the database server is the first place to start.

A database migration project requires a secure network configuration in Google Cloud

- All resources need to be in a project.
- Projects contain one or more VPCs:
 - VPCs provide virtual networking.
 - VPCs in Google Cloud are global resources.
- Each VPC contains one or more subnets:
 - Subnets are regional resources.
 - Create a subnet for each region you want to create VMs in.



In order to build a secure network configuration, all of the resources need to be in a project.

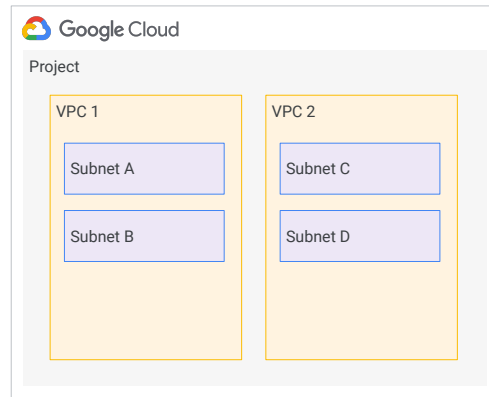
Projects contain one or more VPCs, which provide virtual networking. In Google Cloud, VPCs are a global resource.

Each VPC contains one or more subnets, which are regional resources. You need to create a subnet for each region you want VMs located in.

Use multiple networks to easily isolate machines from each other

By default:

- Machines in the same VPC can communicate via their internal IP address regardless of the subnet (region) they are in; machines in different VPCs can communicate through external IP addresses.
- A VM with no external IP address is only reachable from inside its VPC (*by default*).



By default, machines in the same VPC can communicate with one another via their internal IP address, regardless of the region they are in.

However, machines in different VPCs can communicate through external IP addresses instead. That means that a VM with no public IP address is only reachable by other resources within the same VPC.

You can use multiple networks to control which machines are reachable from other machines and to isolate machines from the outside world.

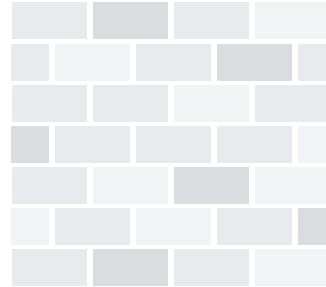
Firewall rules are used to control which machines can communicate using which ports

By default:

- All ports are closed to ingress.
- All ports are open to egress.

When creating firewall rules:

- Targets are used to specify which machines in the VPC the rule applies to.
- Sources specify which machines outside the VPC the rule applies to.
- Use Allow rules to permit ingress.
- Use Deny rules prevent egress.



Firewall rules can control which machines can communicate with one another through designated ports.

- By default, all ports are closed to ingress but open to egress.
- Firewall rules consist of targets, which specify which machines in the VPC the rules apply to, and sources, which specify which machines outside the VPC the rules apply to.
Because ingress is closed by default, you use Allow rules to permit specific ingress, and because egress is open by default, you use Deny rules to prevent specific egress.

A default network is created when you enable the Compute Engine service

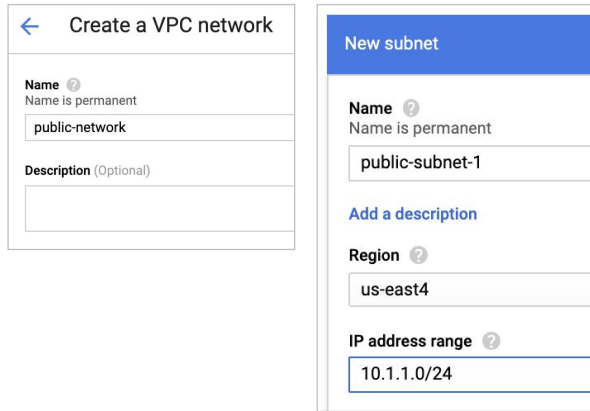
- A subnet is created for every region in the world.
 - Default firewall rules are created for:
 - SSH
 - RDP
 - HTTP
 - HTTPS
 - ICMP
 - All internal traffic is allowed
- The default network makes it easy to get started, but is probably not appropriate for production environments:
 - You probably don't want every subnet to be used.
 - The firewall rules are too permissive (*SSH and RDP are allowed from all sources, for example*).



- When you enable the Compute Engine service in a project, the system automatically creates a default network. This default network contains a subnet for each region in the world. Also, default firewall rules are created for SSH, RDP, HTTP, HTTPS, and ICMP, and all internal traffic is allowed.
- The default network makes it easy to get started, but it should be just that—a starting point. Most production environments will need a modified set of these rules. For example, you probably don't want every subnet to be used, and certainly don't want the overly permissive rules allowing SSH and RDP from all sources.

Creating a custom VPC network

- Give your network a name.
- Add a subnet per region.
- Subnets require an internal IP address range.
- Address ranges in different subnets cannot overlap.



The image displays two screenshots of the Google Cloud console. The left screenshot, titled 'Create a VPC network', shows a form with a 'Name' field containing 'public-network' and a 'Description (Optional)' field. The right screenshot, titled 'New subnet', shows a form with a 'Name' field containing 'public-subnet-1', a 'Region' dropdown set to 'us-east4', and an 'IP address range' field containing '10.1.1.0/24'.

Create a VPC network	New subnet
Name ? Name is permanent public-network	Name ? Name is permanent public-subnet-1
Description (Optional)	Add a description
	Region ? us-east4
	IP address range ? 10.1.1.0/24



The best option is to create your own custom VPC network. You start by giving it a name and then add a subnet for each region you want to use.

Subnets require an internal IP address range. Make sure the address ranges in different subnets don't overlap.

Creating firewall rules illustrated

Network *
public-network

Priority *
1000
Priority can be 0 - 65535 [Check priority of other firewall rules](#)

Direction of traffic ?
☒ Ingress
☐ Egress

Action on match ?
☒ Allow
☐ Deny

Targets
Specified target tags

Target tags *
allow-http

Source filter
IP ranges

Source IP ranges *
0.0.0.0/0 for example, 0.0.0.0/0, 192.168.2.0/24

Create a firewall rule

Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. [Learn more](#)

Name *
public-network-allow-html
Lowercase letters, numbers, hyphens allowed

Protocols and ports ?
☐ Allow all
☒ Specified protocols and ports

☒ tcp : 80
☐ udp : all
☐ Other protocols
protocols, comma separated, e.g. ah, sctp



Here's an example of using the console to fill in the relevant details for creating a firewall rule. The parameters for configuring a firewall rule include name, network, priority, ingress or egress, allow or deny, targets, sources, protocols, and ports.

Firewall rule parameters

Name	Standard naming convention is: [network name]-[allow or deny]-[protocol or port]
Network	Firewall rules are scoped to the network specified.
Priority	Used when multiple rules could apply at the same time. A lower number indicates a higher priority.
Direction	Ingress or egress
Action	Allow or deny
Targets	Can be all instances. Or, use network tags to specify a subset of VMs in the network.
Sources	Usually done with IP address ranges. Use CIDR addressing to specify an IP address or ranges of IP addresses.
Protocols and ports	Specify TCP or UDP ports or port ranges.



Most of the configuration is straightforward:

- Each firewall rule has a unique name, and you should use a consistent naming convention.
- Each rule is scoped to a network.
- If multiple rules conflict, the priority determines which rule wins. Lower numbers have higher priority.
- Direction determines whether this is an ingress or egress rule. Recall that ingress is blocked by default, and egress is allowed by default.
- Therefore, the action is allow or deny. If you are creating an ingress rule, it is probably an "allow"; for an egress rule, it is probably a "deny."
- Targets determine which machines in your network the rules apply to. This can be set to all the machines, or you can use tags or service accounts to specify only certain machines.
- Sources are used to determine which machines outside your network the rule applies to. Sources are usually determined using IP address ranges.
- Finally, you specify protocols and ports. Protocols are either TCP, UDP, or

- ICMP (which is ping). You can specify ports or ranges.

It's not so different from defining firewall rules on your own router. The interesting thing to note is priority, which handles conflicts when multiple rules overlap. In that case, the rule with the lower priority number wins and is used over the higher number.

Firewall targets and sources

Three ways to specify targets

- All instances on network
- Specified target tags:
 - Tag is just a string.
 - Tag must be added to VM for rule to apply.
- Specified service account:
 - VM must run under that service account.



You have three ways to specify both targets and sources.

For targets, you can specify all machines on the network. Or you can specify VMs with a particular network tag. A network tag is simply a string that you can assign to VMs. Lastly, you can specify the service account that VMs are assigned when they are created.

Firewall targets and sources

Three ways to specify targets

- All instances on network
- Specified target tags:
 - Tag is just a string.
 - Tag must be added to VM for rule to apply.
- Specified service account:
 - VM must run under that service account.

Three ways to specify sources

- IP ranges
 - 10.1.1.2/32 - Only that IP address
 - 10.1.1.0/24 - IPs begin with 10.1.1.#
 - 10.1.0.0/16 - IPs begin with 10.1.#.#
 - 0.0.0.0/0 - All machines
- Source tags:
 - Source must include tag.
- Service accounts:
 - Source must run under that service account.



Sources are usually specified using IP addresses or ranges using CIDR notation. With CIDR notation, an IP address is specified followed by a slash and a number. The number determines the range and can be from 0 to 32. Lower numbers indicate larger ranges. Thus, a /32 means only that address, a /24 means all IP addresses that begin with the first three numbers, a /16 means all addresses that begin with the first two numbers, and so on. The range 0.0.0.0/0 means all the machines in the world are considered sources for this rule. As with targets, you can also use tags or service accounts to determine sources.

Use Terraform to automate the creation of resources

- Automation is essential so you can effectively test and iterate on a solution.
- Terraform is included in Cloud Shell by default:
 - Installing it or running a separate Terraform server isn't necessary.

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to database-migration-tf.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
me@cs-6000-devshell-vm-96400f18-24ad-4fe9-910d-ab311220e746:~$ terraform --version
Terraform v0.12.24
me@cs-6000-devshell-vm-96400f18-24ad-4fe9-910d-ab311220e746:~$
```



Automating makes it easier to reproduce and test a solution.

Terraform is a tool from HashiCorp that allows you to automate the creation of resources. Each cloud provider has their own version of a similar tool. AWS has Cloud Formation, and Microsoft Azure has Resource Manager. But Terraform is commonly used to automate cloud resources because it is supported on all the major cloud platforms. Thus, many organizations who want to use multiple or hybrid cloud environments prefer it, so they only need to learn one tool.

Terraform is included in the Cloud Shell by default and requires no installation or separate Terraform server in order to run. Using it will make most tasks easier and automated.

Terraform uses a collection of template files to deploy cloud resources

- Providers are available for AWS, Azure Google, OpenStack, and more.
- One or more .tf files are used to define resources.
- All the files in a folder are combined during a deployment.
- A .tfvars file is used to set variable values.
- A .tfstate file tracks existing resources:
 - Used when updating or destroying resources.



Terraform uses files with instructions, and you put all the files for a specific deployment in one folder.

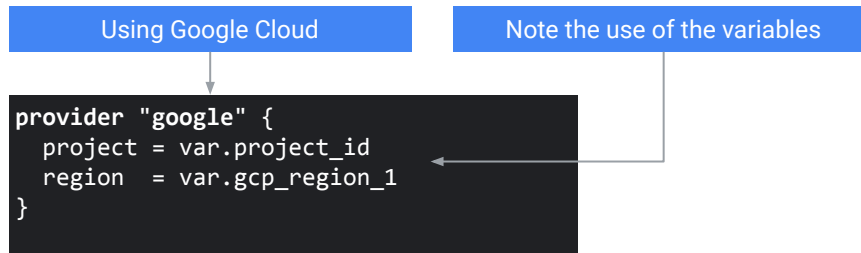
You create templates that describe what you want to create, and there are templates available for creating all types of Google Cloud resources.

You add one or more these .tf files to the folder and combine them for deployment.

If there are variables, you can set them in the .tfvars file.

And the .tfstate file keeps track of existing resources and is used when resources are updated or destroyed.

Use the Google Cloud Terraform provider



There are Terraform providers for every major cloud provider.

This code snippet shows how to use the Google Cloud Terraform provider. Google collaborates with HashiCorp to keep this provider up to date.

In the example, the Google Cloud project ID and region are being set. Note that the values are being set to variables that are created in another `.tf` file. Variables are often a better choice than hard coding values because they promote reuse of the configuration file more easily.

Create variables for your Terraform deployment to more easily change settings

```
variable "project_id" {  
  type = string  
  description = "GCP Project ID"  
}  
variable "gcp_region_1" {  
  type = string  
  description = "GCP Region"  
}  
variable "subnet_cidr_public" {  
  type = string  
  description = "Subnet CIDR for Public Network"  
}
```



In this code block, three variables are being declared: "project_id," "gcp_region_1," and "subnet_cidr_public." This is done in the file variables.tf.

Note that the values are not being set here: the variables are simply being declared. You could however set default values for each variable here if you wanted to.

Variables can be set in a template or at runtime

Set in the terraform.tfvars file

```
project_id    = "database-migration-tf"  
gcp_region_1 = "us-east4"  
subnet_cidr_public  = "10.1.1.0/24"  
subnet_cidr_private = "10.2.2.0/24"
```



You can set the values of your variable in the .tfvars file, or alternatively, you can set them at runtime.

If you do not set the value of a variable in this file, you will be prompted for it when running the template. You could also pass the variable values as parameters when running the template.

Terraform template for creating a VPC and subnet

```
resource "google_compute_network" "public-vpc" {  
  name           = "public-vpc"  
  auto_create_subnetworks = "false"  
  routing_mode    = "GLOBAL"  
}  
  
resource "google_compute_subnetwork" "public-subnet_1" {  
  name           = "public-subnet-1"  
  ip_cidr_range  = var.subnet_cidr_public  
  network        = google_compute_network.public-vpc.name  
  region         = var.gcp_region_1  
}
```

Refers to the network defined above



Here's an example template for creating a VPC with one subnet.

In the first code block, a network is created called "public-vpc." It is a resource of the type "google_compute_network."

The network is then referenced in the second code block when creating the subnet. Notice the network property of the subnet. It refers to the network created above it using the resource type and name. A network also has a property called "name" which is being used to set the value of the subnet's network property.

Terraform commands

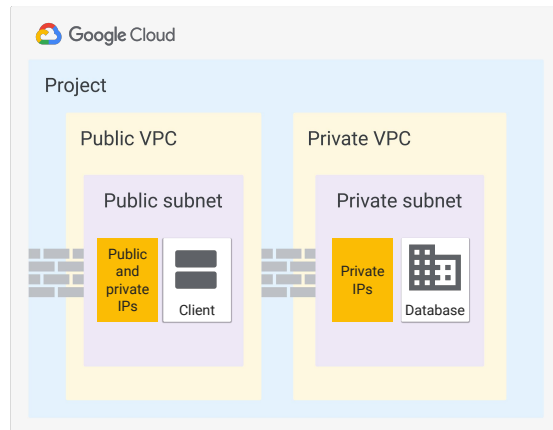
<code>terraform init</code>	Initialize a Terraform working directory. Run from the folder with the Terraform files.
<code>terraform plan</code>	Generate and show an execution plan. Terraform controls the order in which resources are created.
<code>terraform apply</code> <code>-auto-approve</code>	Builds or changes infrastructure described in the templates. Use <code>-auto-approve</code> parameter to prevent from being prompted to approve the plan.
<code>terraform destroy</code> <code>-auto-approve</code>	Destroy Terraform-managed infrastructure. Tracked in the <code>.tfstate</code> file.



Using Terraform involves using the templates to make your own config scripts, then running several Terraform commands.

- The command `terraform init` should be run from the folder where the Terraform files are; it sets the working directory. You only need to do this once whenever you change directories.
- `terraform plan` generates and displays an execution plan. Terraform decides on the order in which to create resources independently of how you ordered them in your configuration. This is part of the Terraform logic, and dependencies are known by the provider you are using.
- The `apply` command builds or modifies the infrastructure according to the plan that was developed from your templates. If you don't want to be prompted to approve the plan, you can include the `-auto-approve` option when you apply.
- Finally, `destroy` would get rid of the resources defined in a template. This is all tracked in the `.tfstate` file. There's also an `-auto-approve` option here.

A secure environment allows only known clients to access the database



Google Cloud

A good practice in securing your environment is to only allow known clients to have access to your database server through firewall rules.

Creating VMs with Terraform

```
resource "google_compute_instance" "sql-client" {
  name          = "sql-client-${random_id.instance_id.hex}"
  machine_type  = "f1-micro"
  zone          = var.gcp_zone_1
  tags          = ["allow-ssh"]
  boot_disk {
    initialize_params {
      image = "ubuntu-os-cloud/ubuntu-1604-xenial-v20200429"
    }
  }

  metadata_startup_script = "sudo apt-get update;"
  network_interface {
    network        = google_compute_network.public-vpc.name
    subnetwork     = google_compute_subnetwork.public-subnet_1.name
    access_config { }
  }
}
```



Here is an example template for creating a virtual machine based on the Ubuntu image. The network tag "allow-ssh" is used as the target tag for the SSH firewall rule. Note the various properties that are being set for the machines. These are the same properties you would set in the web console; you are just doing it with some code.

Creating firewall rules with Terraform

```
resource "google_compute_firewall" "public-allow-ssh" {  
  name      = "${google_compute_network.public-vpc.name}-allow-ssh"  
  network   = "${google_compute_network.public-vpc.name}"  
  allow {  
    protocol = "tcp"  
    ports    = ["22"]  
  }  
  source_ranges = [  
    "0.0.0.0/0"  
  ]  
  target_tags = ["allow-ssh"]  
}
```



Here's an example of how to create a firewall rule using Terraform. In this case, all sources can connect via tcp through port 22, but only to targets with the tag "allow-ssh."

Avoid common pitfalls when configuring networks

- Use care when configuring firewall rules:
 - Only use source 0.0.0.0/0 when appropriate.
 - Use tags or Service accounts to specify targets.
- Don't use external IP addresses on VMs that don't need them.
 - External IPs also cost money and increase egress cost.
- Use CIDR ranges that are small when configuring subnets.
 - Reduces chance of IP conflict when connecting networks.
- Always use SSL when communicating from outside Google Cloud.
 - Traffic within Google Cloud is encrypted by default.
- Use a VPN or Cloud Interconnect to connect on-premises networks to Google.



Perhaps that last example was a bit too permissive. In general, you should avoid allowing all sources, so watch out for using 0.0.0.0/0, unless you have to. For example, allowing HTTPS access to web servers from all sources is probably what you want, but use more restrictive rules for dangerous protocols like SSH, RDP, or database access. Instead, use tags or service accounts.

Also avoid using external IP addresses on VMs if you don't need them. In addition to weakening security, they cost money and also increase egress cost.

When configuring subnets, try to use CIDR ranges that are small to reduce the chance of IP overlap conflicts.

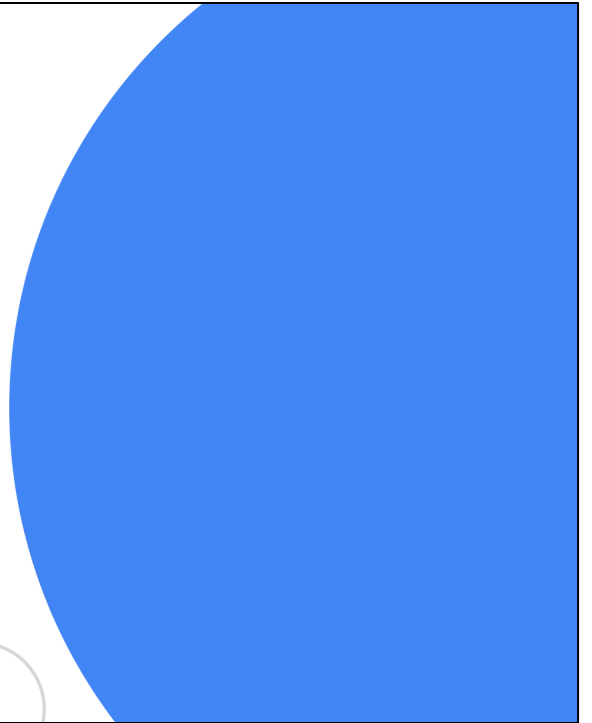
Always use SSL when communicating between Google Cloud and the outside world. Within Google Cloud, all traffic is encrypted by default, but outside of the Google network, you need to encrypt it.

And when you need to connect on-premises networks to Google resources, the best option is to use VPN or Cloud Interconnect, not external IPs.

Lab Intro

Using Terraform to Create Networks and Firewalls

- Use Terraform to automate infrastructure creation.
- Create two VPCs: one for public servers and one for databases.
- Create firewalls for each network.



In this lab, you use Terraform to automate the creation of network infrastructure. You will create two VPCs, one for public service and one for the database server, and create firewall rules for each network.

Lab review

Using Terraform to Create Networks and Firewalls

In this lab, you:

- Use Terraform to automate infrastructure creation.
- Create two VPCs: one for public servers and one for databases.
- Create firewalls for each network.



In this lab:

You used Terraform to automate infrastructure creation.

Created two VPCs: one for public servers and one for databases.

And created firewalls for each network.

The two key takeaways are:

One, how you can use multiple networks and firewall rules to control access to machines and databases.

And two, how you can use Terraform to automate resource creation in Google Cloud.

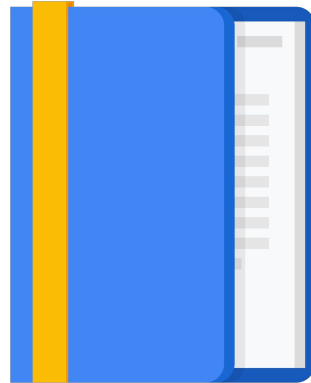
Agenda

Building Secure Networks

Connecting Networks

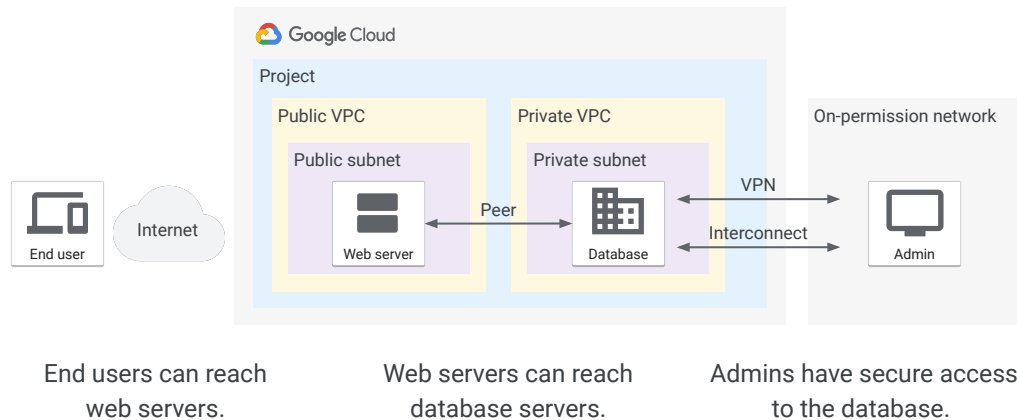
Enabling Communication across
Networks

Network Considerations for
Managed Databases



After you have networks, you want to connect resources with them.

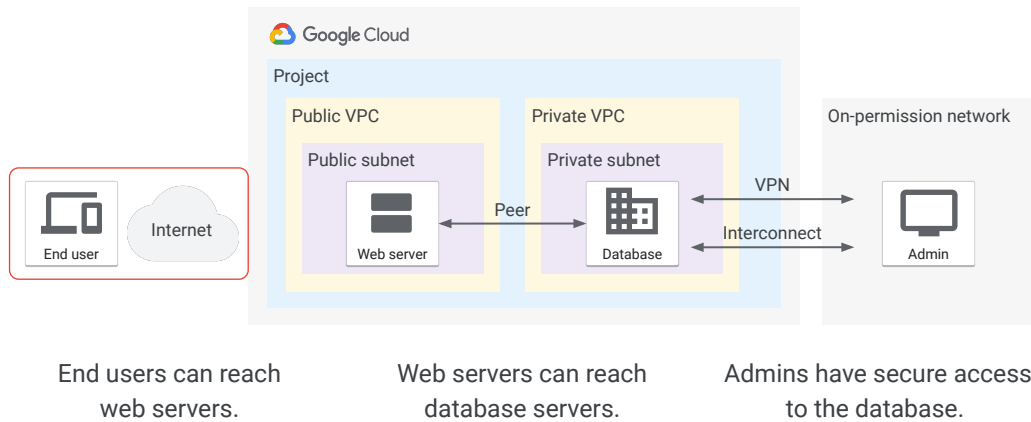
Communication services must be set up so the right computers can talk to each other



 Google Cloud

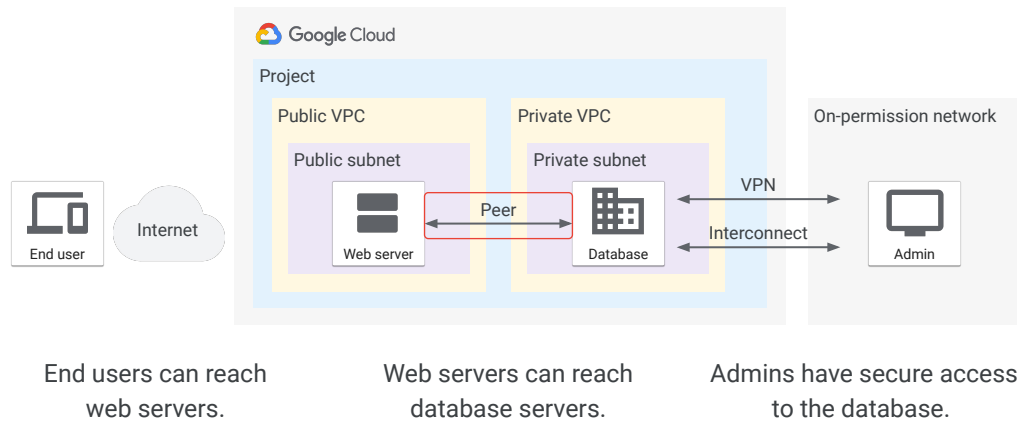
The whole point of networks is to allow resources to connect to one another. But you want to make sure you set this up right, because different types of resources need to talk to one another through different network protocols to make the system secure and to minimize costs.

Communication services must be set up so the right computers can talk to each other



End users will use the public internet to connect to a web server that hosts your app.

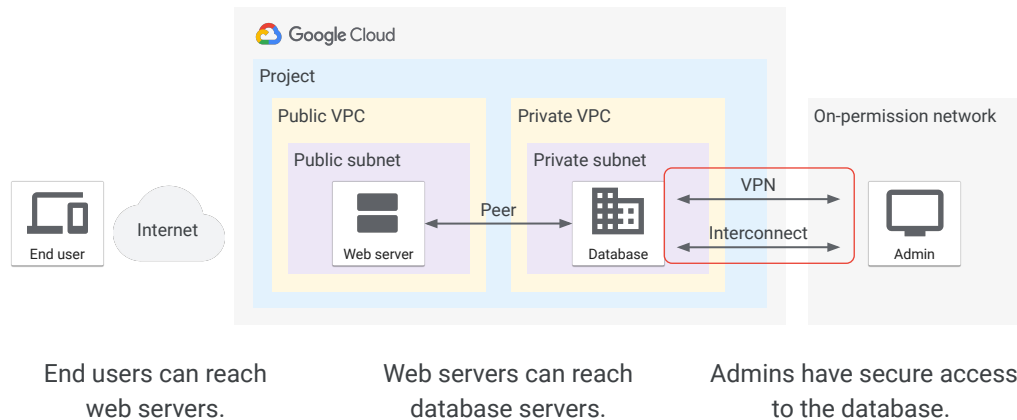
Communication services must be set up so the right computers can talk to each other



 Google Cloud

But that web server will need to talk to a database, which should only allow connections from the same project through VPC.

Communication services must be set up so the right computers can talk to each other



 Google Cloud

You don't want the general public to be able to reach your database server. However, you will have some on-premises resources, such as your admin or data analysts, who will need to connect to the database. For them, a VPN or Interconnect is the more appropriate way to connect versus a public IP.

Cloud VPN securely connects your on-premises network to your Google Cloud VPC network

- Useful for low-volume data connections
- Classic VPN: 99.9% SLA
- High-availability (HA) VPN: 99.99% SLA
- Supports:
 - Site-to-site VPN
 - Static routes (Classic VPN)
 - Dynamic routes (Cloud Router)
 - IKEv1 and IKEv2 ciphers

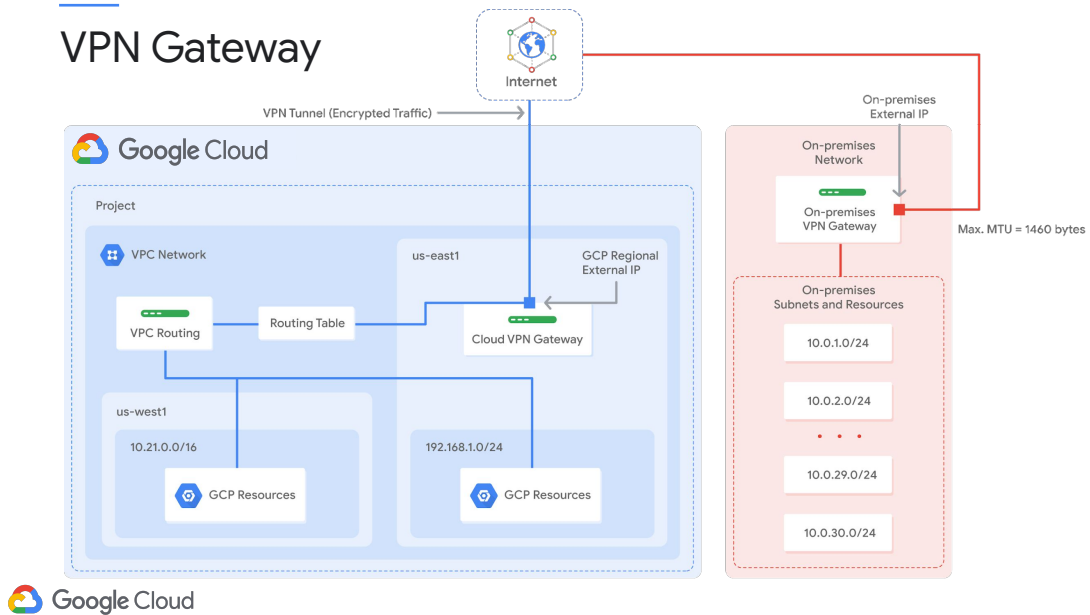


Cloud VPN is the way to connect your on-premises network to your Google Cloud VPC network. It's highly available and secure.

Cloud VPN can support connections up to about 3 Gigabits per second, and It can be configured two ways. The Classic configuration consists of a single VPN tunnel and offers a 99.9% availability SLA. A high-availability configuration uses a second tunnel to achieve 99.99% availability.

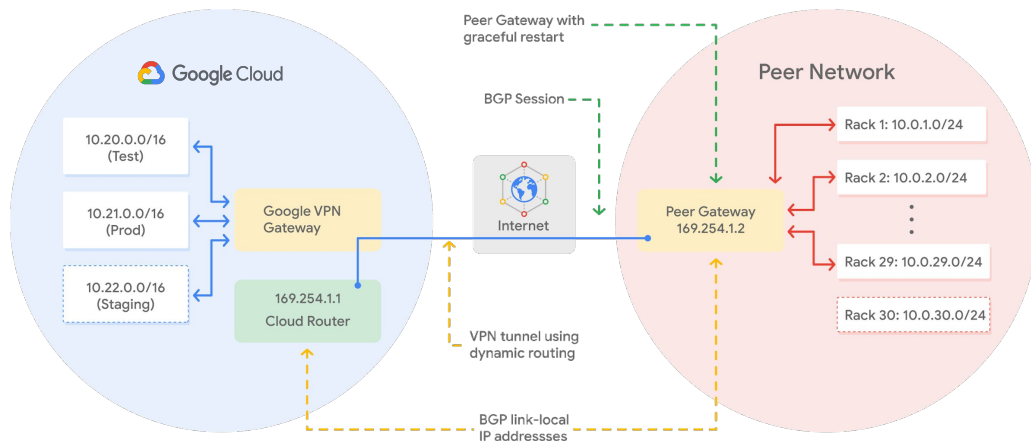
Cloud VPN can also be configured for both static or dynamic routing using Cloud Router.

VPN Gateway



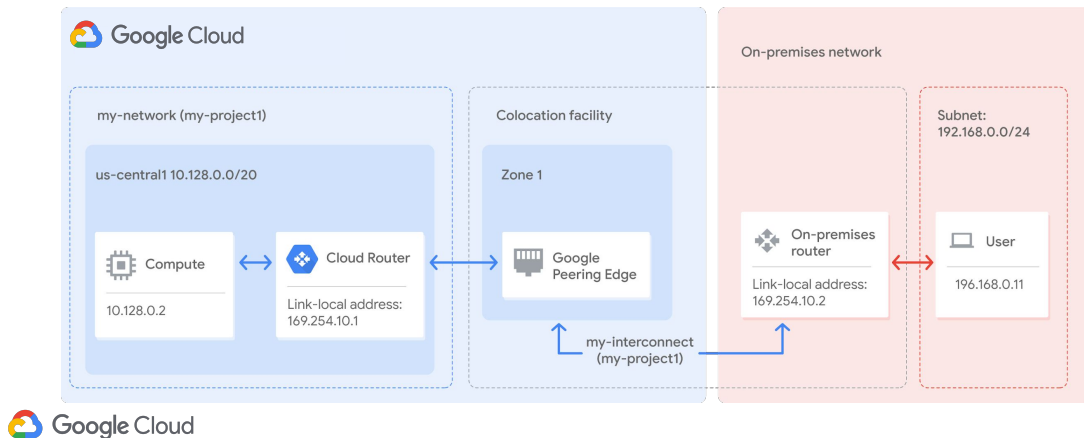
Google Cloud provides a VPN Gateway as a service that you use on-premises to connect your local network to the VPC Network. You set up a VPN gateway in your on-premises network and Cloud VPN in your Google Cloud VPC, and connect the two. This provides an encrypted connection for secure communication between the networks.

Cloud Router enables dynamic routes using Border Gateway Protocol (BGP)



Cloud Router is not a physical router but is instead a service that works over Cloud VPN or Cloud Interconnect connections to provide dynamic routing by using the Border Gateway Protocol. Dynamic routing just means that if the network on either side changes, the connection between the two is automatically updated.

Dedicated Interconnect provides direct physical connections

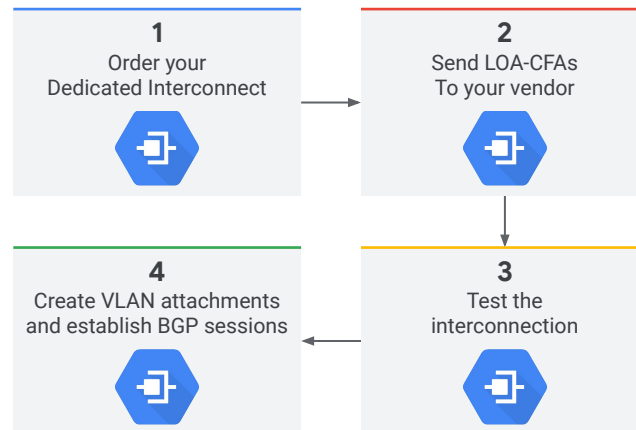


Dedicated Interconnect provides direct physical connections between your on-premises network and Google's network. This enables you to transfer large amounts of data between networks, which can be more cost-effective than purchasing additional bandwidth over the public internet.

In order to use Dedicated Interconnect, you need to provision a cross connect between the Google network and your own router in a common colocation facility, as shown in this diagram. To exchange routes between the networks, you configure a BGP session over the interconnect between the Cloud Router and the on-premises router. This will allow user traffic from the on-premises network to reach Google Cloud resources on the VPC network, and vice versa.

Dedicated Interconnect can be configured to offer a 99.9% or a 99.99% uptime SLA. You can refer to the Dedicated Interconnect documentation for details on how to achieve these SLAs

Create a Dedicated Interconnect connection



Creating a Dedicated Interconnect Connection is as simple as these four steps:

- First, order your Dedicated Interconnect.
- Second, send LOA-CFAs to your vendor.
- Third, test the Interconnect.
- And finally fourth, create VLAN attachments and establish BGP sessions.

Colocation facility locations

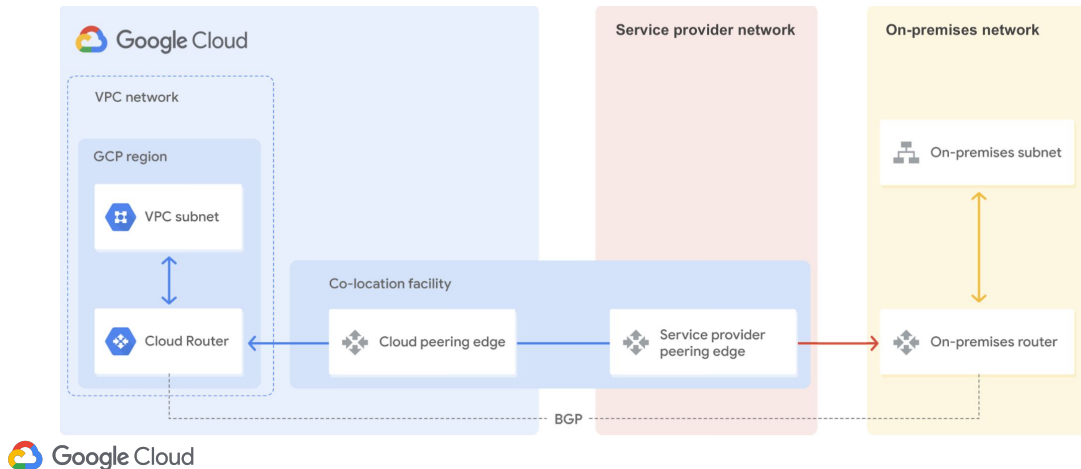
○ Dedicated Interconnect



In order to use Dedicated Interconnect, your network must physically meet Google's network in a supported colocation facility. This map shows the locations where you can currently create dedicated connections.

Now you might look at this map and say, "Well I am nowhere near one of those locations." That's when you want to consider Partner Interconnect.

Partner Interconnect provides connectivity through a supported service provider



Partner Interconnect provides connectivity between your on-premises network and your VPC network through a supported service provider. This is useful if your data center is in a physical location that can't reach a Dedicated Interconnect colocation facility or if your data needs don't warrant a Dedicated Interconnect. In order to use Partner Interconnect, you work with a supported service provider to connect your VPC and on-premises networks. You can refer to the documentation for a full list of providers.

These service providers have existing physical connections to Google's network that they make available for their customers to use. After you establish connectivity with a service provider, you can request a Partner Interconnect connection from your service provider. Then, you establish a BGP session between your Cloud Router and on-premises router to start passing traffic between your networks via the service provider's network.

Partner Interconnect can be configured to offer a 99.9% or a 99.99% uptime SLA between Google and the service provider. See the Partner Interconnect documentation for details on how to achieve these SLAs

Comparison of Interconnect options

Connection	Provides	Capacity	Requirements	Access Type
IPsec VPN tunnel	Encrypted tunnel to VPC networks through the public internet	1.5–3 Gbps per tunnel	On-premises VPN gateway	Internal IP addresses
Dedicated Interconnect	Dedicated, direct connection to VPC networks	10 Gbps or 100 Gbps per link	Connection in colocation facility	
Partner Interconnect	Dedicated bandwidth, connection to VPC network through a service provider	50 Mbps – 10 Gbps per connection	Service provider	



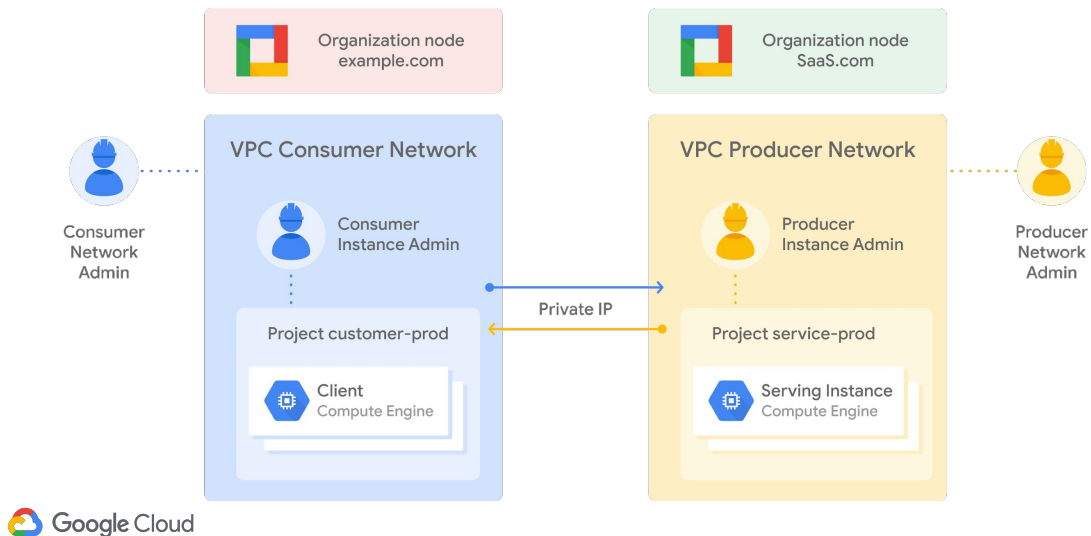
Now let's compare the interconnect options that we just discussed.

- The **IPsec VPN tunnels** that Cloud VPN offers have a capacity of 1.5 to 3 Gbps per tunnel and require a VPN device on your on-premises network. The 1.5-Gbps capacity applies to traffic that traverses the public internet, and the 3-Gbps capacity applies to traffic that is traversing a direct peering link. You can configure multiple tunnels if you want to scale this capacity.
- **Dedicated Interconnect** has a capacity of 10 Gbps or 100 Gbps per link and requires you to have a connection in a Google-supported colocation facility. You can have up to 8 links to achieve multiples of 10 Gbps, or up to 2 links to achieve multiples of 100 Gbps, but 10 Gbps is the minimum capacity.
- **Partner Interconnect** has a capacity of 50 Mbps to 10 Gbps per connection, and requirements depend on the service provider.
- All of these options provide internal IP address access between resources in your on-premises network and in your VPC network. The main differences are the connection capacity and the requirements for using a service.

My recommendation is to start with VPN tunnels. When you need enterprise-grade

connections to Google Cloud, switch to Dedicated Interconnect or Partner Interconnect, depending on your proximity to a colocation facility and your capacity requirements.

VPC peering is used to connect two Google Cloud VPCs



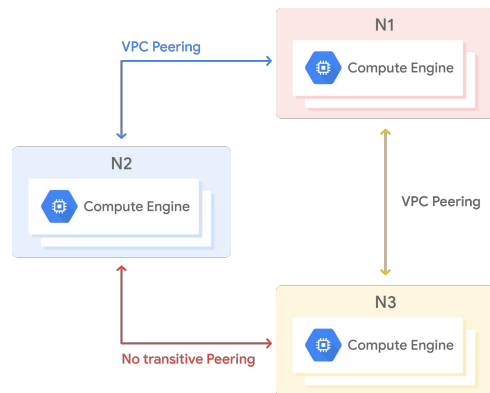
VPC Network Peering allows private RFC 1918 connectivity across two VPC networks, regardless of whether they belong to the same project or the same organization. Now remember that each VPC network will have firewall rules that define what traffic is allowed or denied between the networks.

For example, in this diagram there are two organizations that represent a consumer and a producer. Each organization has its own organization node, VPC network, VM instances, Network Admin, and Instance Admin. In order for VPC Network Peering to be established successfully, the Producer Network Admin needs to peer the Producer Network with the Consumer Network, and the Consumer Network Admin needs to peer the Consumer Network with the Producer Network. When both peering connections are created, the VPC Network Peering session becomes Active and routes are exchanged. This allows the VM instances to communicate privately using their internal IP addresses.

VPC Network Peering is a decentralized or distributed approach to multi-project networking, because each VPC network may remain under the control of separate administrator groups and maintains its own global firewall and routing tables. Historically, such projects would consider external IP addresses or VPNs to facilitate private communication between VPC networks. However, VPC Network Peering does not incur the network latency, security, and cost drawbacks that are present when using external IP addresses or VPNs.

Remember when using VPC peering

- Peered networks can be in the same project or in different projects.
- Peered networks can be owned by different organizations.
- Peered VPC networks remain administratively separate.
- Subnet IP ranges cannot overlap across peered VPC networks.
- Transitive peering is not supported.



VPC Network Peering works with Compute Engine, Kubernetes Engine, and App Engine flexible environments.

Remember a few things when using VPC Network Peering:

- Peered networks can be in the same project or in different projects.
- Peered networks can be owned by different organizations.
- Each side of a peering association is set up independently. Peering will be active only when the configuration from both sides matches. This allows either side to delete the peering association at any time.
- A subnet CIDR prefix in one peered VPC network cannot overlap with a subnet CIDR prefix in another peered network. This means that two auto mode VPC networks that only have the default subnets cannot peer.
- And Only directly peered networks can communicate, which means that transitive peering is not supported. In other words, if VPC network N1 is peered with N2 and N3, but N2 and N3 are not directly connected, VPC network N2 cannot communicate with VPC network N3 over the peering. This is critical if N1 is a SaaS organization offering services to N2 and N3.

A peering request has to be made from both networks

<p>Name *</p> <p>public-private-network-peer</p> <p>Lowercase letters, numbers, hyphens allowed</p> <p>Your VPC network *</p> <p>public-network</p> <p>Peered VPC network</p> <p><input checked="" type="radio"/> In project database-migration-275815</p> <p><input type="radio"/> In another project</p> <p>VPC network name *</p> <p>private-network</p> <p>✓ EXCHANGE CUSTOM ROUTES</p> <p>CREATE CANCEL</p>	<p>Name *</p> <p>private-public-network-peer</p> <p>Lowercase letters, numbers, hyphens allowed</p> <p>Your VPC network *</p> <p>private-network</p> <p>Peered VPC network</p> <p><input checked="" type="radio"/> In project database-migration-275815</p> <p><input type="radio"/> In another project</p> <p>VPC network name *</p> <p>public-network</p> <p>✓ EXCHANGE CUSTOM ROUTES</p> <p>CREATE CANCEL</p>
--	--



As pointed out on the last slide, each side of the peering association needs to be set up separately. So note here how the public and private networks each reciprocally connect to the other. If you only do one, it is not going to automatically allow the other.

Internal IP address ranges in peered networks cannot overlap

private-network	2	Custom		
us-central1	iowa		10.0.2.0/24	10.0.2.1
us-east4	virginia		10.0.1.0/24	10.0.1.1
public-network	2	Custom		
us-central1	iowa-public		10.2.2.0/24	10.2.2.1
us-east4	virginia-public		10.2.1.0/24	10.2.1.1



As always, make sure that you don't have overlapping IP address ranges between the peered networks.

Routes are automatically generated that allow traffic to flow between peered networks

Routes			
CREATE ROUTE REFRESH DELETE			
ALL DYNAMIC PEERING			
<input type="checkbox"/>	Name ↓	Description	Destination IP range
<input type="checkbox"/>	peering-route-d6250c5a972ebe1f	Auto generated route via peering [public-private-network-peer].	10.0.1.0/24
<input type="checkbox"/>	peering-route-931a7533a0bae938	Auto generated route via peering [public-private-network-peer].	10.0.2.0/24
<input type="checkbox"/>	peering-route-8f14d463b538cd6e	Auto generated route via peering [private-public-network-peer].	10.2.1.0/24
<input type="checkbox"/>	peering-route-45e033e403e95979	Auto generated route via peering [private-public-network-peer].	10.2.2.0/24



Creating the peering requests automatically generates the necessary routes to allow the traffic to flow between the peered networks.

Use Terraform to automate the creation of peered networks

```
resource "google_compute_network_peering" "public-private" {
  name      = "peering1"
  network   = google_compute_network.public-vpc.self_link
  peer_network = google_compute_network.private-vpc.self_link
}

resource "google_compute_network_peering" "private-public" {
  name      = "peering2"
  network   = google_compute_network.private-vpc.self_link
  peer_network = google_compute_network.public-vpc.self_link
}
```



Terraform templates make automating this easier. In the code shown here, note that there is a public-private request and then the reciprocal private-public request. It's simple: you just make two peering requests that are the inverse of each other.

Lab Intro

Using Terraform to Create a Network Peering

- Peer two networks using Terraform.



Multiple networks allow you to easily isolate machines and control which machines have access to other ones. In the course example, you want databases isolated in their own network with no external IP addresses. Those servers, by default, will only be accessible from machines in the same network. You can then put database clients like web servers in another network. Then peer the two networks allowing communication via only internal addresses.

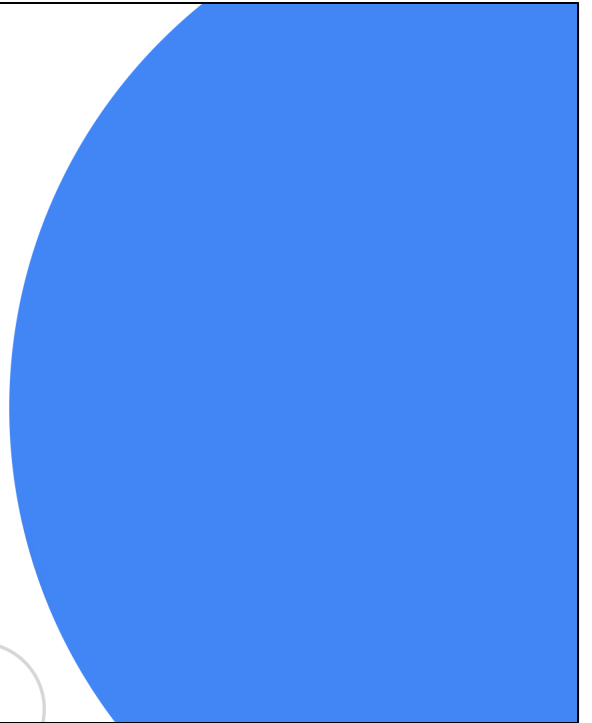
In this lab, you peer two networks using Terraform.

Lab review

Using Terraform to Create a Network Peering

In this lab, you:

- Peered two networks using Terraform.



In this lab, you learned how to use Terraform to set up network peering. The peering allows machines in two different networks to communicate using internal IP addresses. This allows you to protect your databases by controlling exactly which machines have access to them and from which networks.

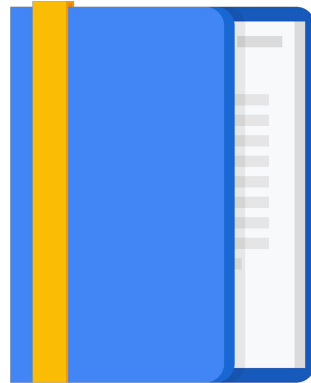
Agenda

Building Secure Networks

Connecting Networks

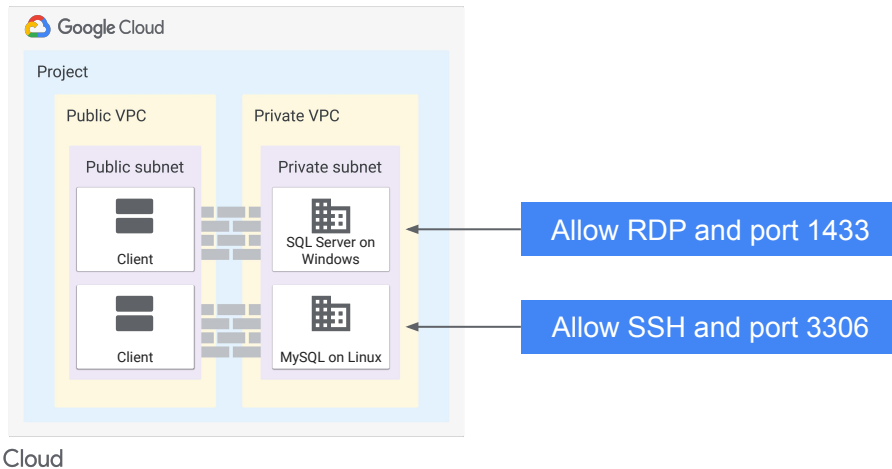
Enabling Communication across
Networks

Network Considerations for
Managed Databases



Now that the networks are in place and connected to each other, you need to set up the appropriate communications between them.

Configure firewall rules to allow communication to the database server from the peered network



First, allow clients from the peered network to talk to the database server. For SQL Server, if you are using a Windows server, you would use the default port of 1433 and RDP. For MySQL, assuming you are running a Linux server, you would open port 3306 and SSH.

Example firewall rule to allow traffic to SQL Server from the peered network

```
resource "google_compute_firewall" "private-allow-sql" {
  name     = "${google_compute_network.private-vpc.name}-allow-sql"
  network  = "${google_compute_network.private-vpc.name}"
  allow {
    protocol = "tcp"
    ports    = ["1433"]
  }
  source_ranges = [
    "${var.subnet_cidr_public}"
  ]
  target_tags = ["allow-sql"]
}
```

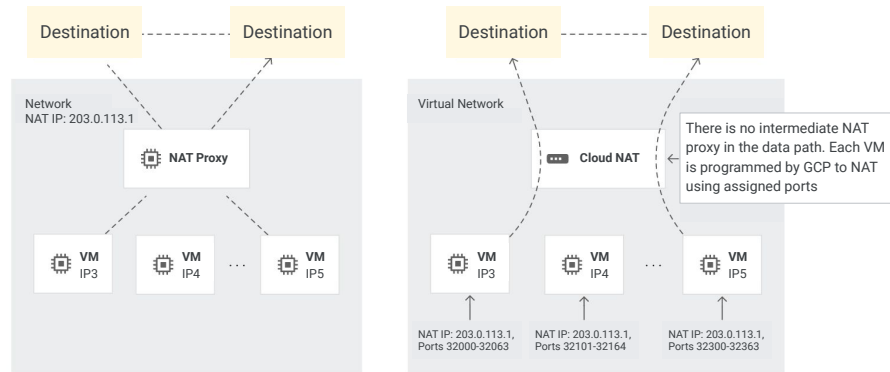


Here's an example Terraform script to allow clients to connect to the SQL Server instance through port 1433. Note that this code is using the variable that sets the internal IP address range of the client network to set the `source_ranges` property of the firewall rule. Thus, the SQL Server instance is in a private network that is only accessible from the client network. Users can connect to the web servers in the public network, but only the servers in the public network can connect to the SQL Server instance.

Also, note the `target_tags` variable. This rule only applies to servers tagged with the string "allow-sql".

A NAT proxy is required to provide internet access to machines with no external IP access

- Can create a NAT using a Compute Engine VM or the Cloud NAT service



When you have a machine with no public IP address, which is usually the preferred configuration, in order for that machine to reach the internet, you would need a NAT proxy. You can create a NAT proxy using either a custom Compute Engine VM or by using the Cloud NAT service.

Create a Cloud NAT gateway in the console or Terraform

```
resource "google_compute_router_nat" "private-nat" {
  name           = "private-nat"
  router         = google_compute_router.nat-router.name
  region         = google_compute_router.nat-router.region
  nat_ip_allocate_option = "AUTO_ONLY"
  source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"

  log_config {
    enable = true
    filter = "ERRORS_ONLY"
  }
}
```



Here's a Terraform example using the Cloud NAT gateway service. Again, the NAT gateway will allow machines with no External IP address to access the internet. Note the region and router parameters. These parameters refer to a Cloud Router, which also must be configured.

Cloud Router allows for dynamic routing using BGP (Border Gateway Protocol)

Used by Cloud NAT

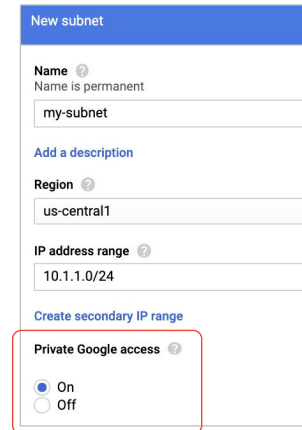
```
resource "google_compute_router" "nat-router" {  
  name      = "nat-router"  
  region    = google_compute_subnetwork.private-subnet_1.region  
  network   = google_compute_network.private-vpc.id  
  
  bgp {  
    asn = 64514  
  }  
}
```



Here the Cloud Router configuration is shown. Note, Cloud Router is a regional service. By default, Cloud Router advertises subnets in its region for regional dynamic routing, or all subnets in a VPC network for global dynamic routing. New subnets are automatically advertised by Cloud Router. Also note, the network that the router serves must be specified.

Private Google access allows machines without external IP addresses to access Google services

- Set when creating subnets.
- For example, a database server needs access to Cloud Storage for backups.



The screenshot shows the 'New subnet' configuration form. The 'Name' field is 'my-subnet'. The 'Region' is 'us-central1'. The 'IP address range' is '10.1.1.0/24'. The 'Private Google access' section is highlighted with a red box, showing the 'On' radio button selected.

New subnet	
Name ?	Name is permanent
my-subnet	
Add a description	
Region ?	us-central1
IP address range ?	10.1.1.0/24
Create secondary IP range	
Private Google access ?	
<input checked="" type="radio"/> On	
<input type="radio"/> Off	



So, if a machine with no public IP address needs internet access, you now know how to fix that Using Cloud NAT.

But what about letting that VM access Google Cloud services like Cloud Storage or BigQuery? In order to permit that, you need to turn on Private Google access. This is done when configuring subnets. Private Google Access permits access to Cloud and Developer APIs and most Google Cloud services, with a few exceptions.

A real-world database use case might be allowing the database server access to Cloud Storage for storing backups.

Lab intro

Using Terraform to Create Clients and Servers

- Create client and server VMs in both public and private networks.
- Set up communication between those services.



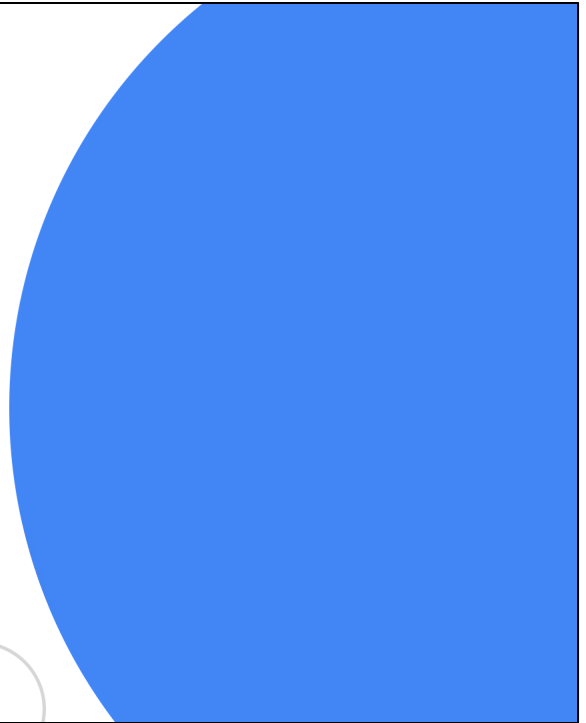
In this lab, you use Terraform to create client and server VMs in both public and private networks and set up the communications between them to allow them to communicate with each other.

Lab review

Using Terraform to Create Clients and Servers

In this lab, you:

- Created client and server VMs in both public and private networks.
- Set up communication between those services.



In this lab, you used Terraform to create client and server VMs in both public and private networks and set up the communications between them to allow them to communicate with each other. At this point, you have learned how to automate the setup of secure networks. This is very important when migrating databases to the cloud.

Agenda

Building Secure Networks

Connecting Networks

Enabling Communication across
Networks

Network Considerations for
Managed Databases



We've looked at network considerations with VM-based database servers, so now let's look at network issues when using Google Cloud managed database services.

Cloud SQL allows for private IPs, public IPs, or both

1 Connectivity

Choose how you would like to connect to your database instance.
For extra security, consider using the Cloud SQL proxy to connect to your instances after creation. [Learn more](#)

☒ **Private IP**

Private IP connectivity requires additional APIs and permissions. You may need to contact your organization's administrator for help enabling or using this feature. Currently, Private IP cannot be disabled once it has been enabled.

Associated networking
Select a network to create a private connection

default

This instance will use the existing managed service connection

☒ **Public IP**

ⓘ You have not authorized any external networks to connect to your Cloud SQL instance. External applications can still connect to the instance through the Cloud SQL Proxy. [Learn more](#)

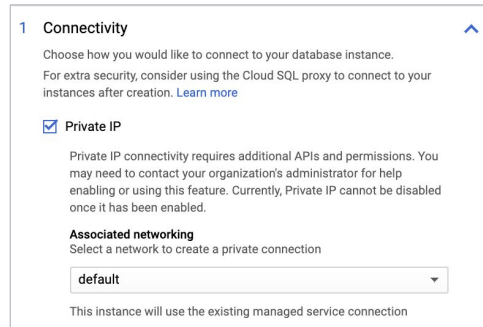
Authorized networks
Authorize a network or use a Proxy to connect to your instance. Networks will only be authorized via these addresses. [Learn more](#)



Cloud SQL allows you to choose either private or public IP addresses, or both. Also make note that once a private IP address is enabled, it cannot be disabled.

If a private IP is chosen, a network peering is created from Google's network to the network specified

- Cloud SQL database is managed by Google in a network that they own.
- You choose the network to associate with Google's network.
- Cloud SQL database is available from the peered network using the private IP address.



The screenshot shows the 'Connectivity' tab in the Google Cloud console. It has a title '1 Connectivity' with an upward arrow. Below the title, it says 'Choose how you would like to connect to your database instance.' and 'For extra security, consider using the Cloud SQL proxy to connect to your instances after creation. [Learn more](#)'. There is a checked checkbox for 'Private IP'. Below this, a note states: 'Private IP connectivity requires additional APIs and permissions. You may need to contact your organization's administrator for help enabling or using this feature. Currently, Private IP cannot be disabled once it has been enabled.' Under the heading 'Associated networking', it says 'Select a network to create a private connection' and shows a dropdown menu with 'default' selected. At the bottom, it says 'This instance will use the existing managed service connection'.



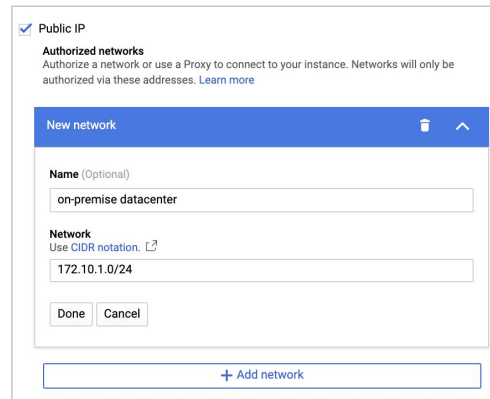
When you enable a Private IP, Google automatically creates a network peering between your network (*which you specify*) and their network where your Cloud SQL database is running. This is just like what you set up earlier in the course. The difference is that Google manages the Cloud SQL network.

After the private IP is enabled, machines in the peered network can communicate without needing a public IP address. You can clear the Public IP option, and the server will not be given one.

A Cloud SQL instance will be in Google's own managed network. You may need to allow resources in your project to connect to that Cloud SQL instance, so you would choose a private IP to do that. This will associate your chosen network with Google's network and peer them.

If a public IP is chosen, the database is protected by a firewall

- By default, only applications in the current project can access the database.
- You authorize one or more additional networks using IP addresses or ranges.
- Create a VPN or Cloud Interconnect to connect networks outside of Google Cloud.



☒ Public IP

Authorized networks
Authorize a network or use a Proxy to connect to your instance. Networks will only be authorized via these addresses. [Learn more](#)

New network

Name (Optional)
on-premise datacenter

Network
Use CIDR notation. [?](#)
172.10.1.0/24

Done Cancel

+ Add network



If you choose a public IP, your database server is protected by firewall rules instead. By default, only apps in the current project can access the database, but you can authorize additional networks to have access by using IP addresses or ranges, just like you do when specifying source IPs in a firewall rule.

You can also create a VPN or Cloud Interconnect to connect networks outside Google Cloud. This would be the preferred way to connect your on-premises network to the manage database instance.

Module review



In this module, you:

Built secure networks to host databases and database client applications.

Allowed secure communication across networks using VPC Peering, VPNs, and Interconnect.

Controlled access to databases using firewall rules.

And Automated network infrastructure using Terraform.

