Using Private Service Connect to publish and consume services

1. Introduction

Private Service Connect lets a service producer offer services privately to a service consumer. Private Service Connect offers the following benefits:

- A service producer VPC network can support more than one service consumer.
- Each consumer connects to an internal IP address that they define. Private Service Connect performs network address translation (NAT) to route the request to the service producer.

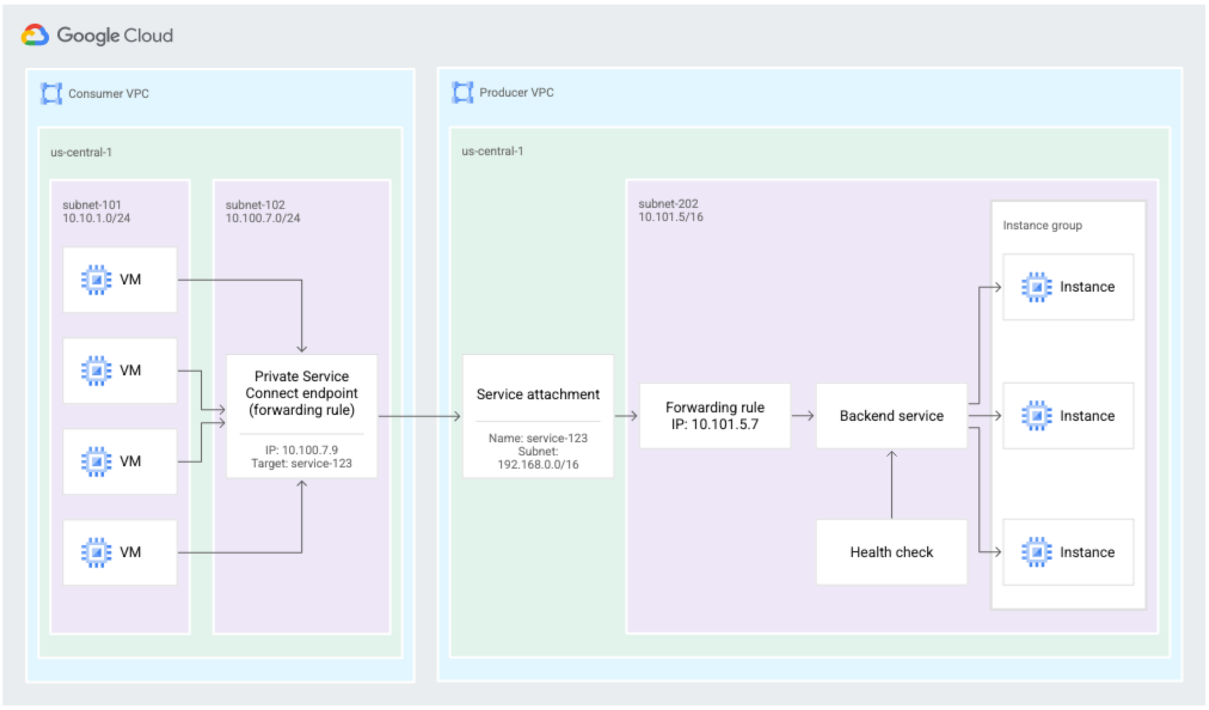


Figure 2. Private Service Connect uses endpoints and service attachments to let service consumers send traffic from the consumer's VPC network to services in the service producer's VPC network (click to enlarge).

What you'll learn

- Private service connect benefits
- Key concepts for service consumers
- Key concepts for service producers
- Create a producer environment
- Expose service (producer environment) through a service attachment
- Create a consumer environment

- Create a forwarding rule in the consumer network
- Validate TCP consumer access
- Enable & validate proxy protocol
- Enable policy access control

What you'll need

- Knowledge of Internal load balancers
- Ability to create VPCs in two projects

2. Private service connect benefits

With PSC, you have several benefits compared to using VPC Peering:

Better control of private IP space

- As a service consumer, you can control the private IP address that is used to connect to the managed service you would like to access.
- As a service consumer, you do not need to worry about reserving private IP address ranges for backend services that are consumed in your VPC. You only need to choose an IP address from your own subnet to connect to the producer services.
- As a service producer, you can choose to deploy a multi-tenant model, where your VPC contains services that serve multiple consumer VPCs. The consumers having overlapping subnet ranges are no longer a problem.
- As a service provider, you can scale your service to as many VM instances as required, without needing to contact your consumer for more IP addresses.

Improved security and isolation

- As a service consumer, only you can initiate communication to the service producer. This uni-directional connectivity drastically simplifies firewall configuration but also reduces risk from rouge traffic coming from the service producer.
- As a service producer, you do not need to change your firewall rules based on the subnet ranges in the consumer's VPC. You can simply create firewall rules for the NAT IP address range configured for your service.

Better Scalability

- PSC enables highly-scalable design by supporting thousands of Consumers, and allows Service Producers to offer highly scalable multi-tenant or single-tenant services.
- As a service consumer using private service connect, you can create resources as required in your VPC. The scale of this is not affected by the number of such resources created in the producer VPC.

3. Key concepts for service consumers

You can use Private Service Connect endpoints to consume services that are outside of your VPC network. Service consumers create Private Service Connect endpoints that connect to a target service.

Endpoints

You use Private Service Connect endpoints to connect to a target service. Endpoints have an internal IP address in your VPC network and are based on the forwarding rule resource.

You send traffic to the endpoint, which forwards it to targets outside of your VPC network.

Targets

Private Service Connect endpoints have a target, which is the service you want to connect to:

- An API bundle:
- All APIs: most Google APIs
- VPC-SC: APIs that VPC Service Controls supports
- A published service in another VPC network. This service can be managed by your own organization or a third party.

Published service

To connect your endpoint to a service producer's service, you need the service attachment for the service. The service attachment URI has this format:
projects/SERVICE_PROJECT/regions/REGION/serviceAttachments/SERVICE_NAME

4. Key concepts for service producers

To make a service available to consumers, you create one or more dedicated subnets to use for network address translation (NAT) of consumer IP addresses. You then create a service attachment which refers to those subnets.

Private Service Connect subnets

To expose a service, the service producer first creates one or more subnets with purpose Private Service Connect.

When a request is sent from a consumer VPC network, the consumer's source IP address is translated using source NAT (SNAT) to an IP address selected from one of the Private Service Connect subnets.

If you want to retain the consumer connection IP address information, see Viewing consumer connection information.

These subnets cannot be used for resources such as VM instances or forwarding rules. The subnets are used only to provide IP addresses for SNAT of incoming consumer connections.

The Private Service Connect subnet must contain at least one IP address for every 63 consumer VMs so that each consumer VM is allocated 1,024 source tuples for network address translation.

The minimum size for a Private Service Connect subnet is /24.

Service attachments

Service producers expose their service through a service attachment.

- To expose a service, a service producer creates a service attachment that refers to the service's load balancer forwarding rule.
- To access a service, a service consumer creates an endpoint that refers to the service attachment.

Connection preferences

When you create a service, you choose how to make it available. There are two options:

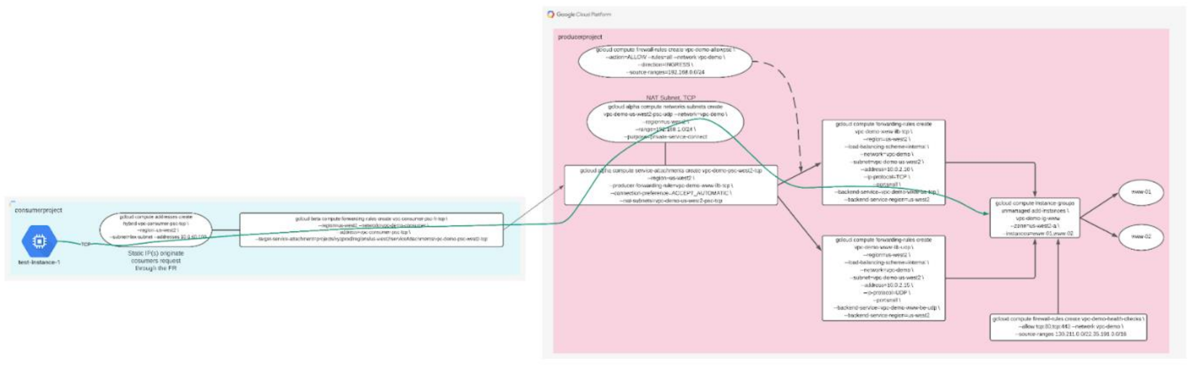
- Automatically accept connections for all projects - any service consumer can configure an endpoint and connect to the service automatically.
- Accept connections for selected projects - service consumers configure an endpoint to connect to the service and the service producer accepts or rejects the connection requests.

5. Test Environment

Consumer network consists of a TCP static IP address used to originate requests to the service producer, in addition to the target-service-attachment that maps to the producer's service attachment (published service).

Now, let's take a look at the producers network. Notice how the producers network does not have a mapping to the consumers network, instead, the producer network contains a service attachment (published service) that is used by the consumer for services. The producer's service attachment in our lab is a Layer 4 Internal Load Balancer (producer-forwarding-rule) mapped to a backend service supporting a TCP application.

The NAT subnet and associated firewall rules allow communication to the producer application.



```
gcloud compute addresses create
hybrid-vpc-consumer-psc-tcp \
--region=us-west2 \
--subnet=lax-subnet --addresses 10.0.60.100
```

```
gcloud beta compute forwarding-rules create vpc-consumer-psc-fr-tcp \
--region=us-west2 --network=vpc-demo-consumer \
--address=vpc-consumer-psc-tcp \
--target-service-attachment=projects/ypprod/regions/us-west2/serviceAttachments/vpc-demo-psc-west2-tcp
```

```
gcloud compute firewall-rules create vpc-demo-allowpsc \
--action=ALLOW --rules=all --network vpc-demo \
--direction=INGRESS \
--source-ranges=192.168.0.0/24
```

```
gcloud alpha compute networks subnets create
vpc-demo-us-west2-psc-udp --network=vpc-demo \
--region=us-west2 \
--range=192.168.1.0/24 \
--purpose=private-service-connect
```

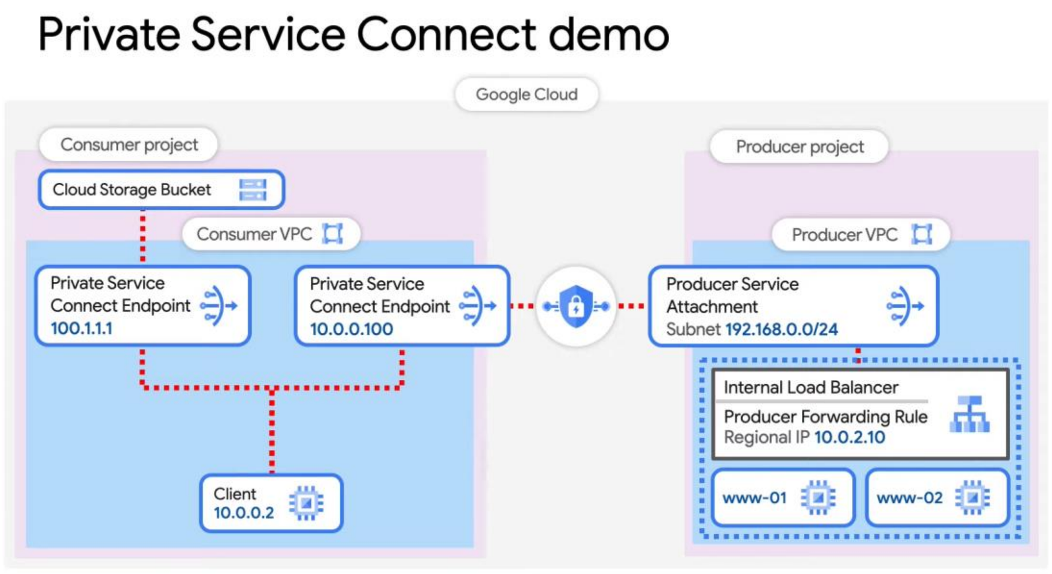
```
gcloud alpha compute service-attachments create vpc-demo-psc-west2-tcp
--region=us-west2 \
--producer-forwarding-rule=vpc-demo-www-ilb-tcp \
--connection-preference=ACCEPT_AUTOMATIC \
--nat-subnets=vpc-demo-us-west2-psc-tcp
```

```
gcloud compute forwarding-rules create
vpc-demo-www-ilb-tcp \
--region=us-west2 \
--load-balancing-scheme=internal \
--network=vpc-demo \
--subnet=vpc-demo-us-west2 \
--address=10.0.2.10 \
--ip-protocol=TCP \
--ports=all \
--backend-service=vpc-demo-www-be-tcp \
--backend-service-region=us-west2
```

```
gcloud compute forwarding-rules create
vpc-demo-www-ilb-udp \
--region=us-west2 \
--load-balancing-scheme=internal \
--network=vpc-demo \
--subnet=vpc-demo-us-west2 \
--address=10.0.2.15 \
--ip-protocol=UDP \
--ports=all \
--backend-service=vpc-demo-www-be-udp \
--backend-service-region=us-west2
```

```
gcloud compute instance-groups
unmanaged add-instances \
vpc-demo-ig-www
--zone=us-west2-a \
--instances=www-01,www-02
```

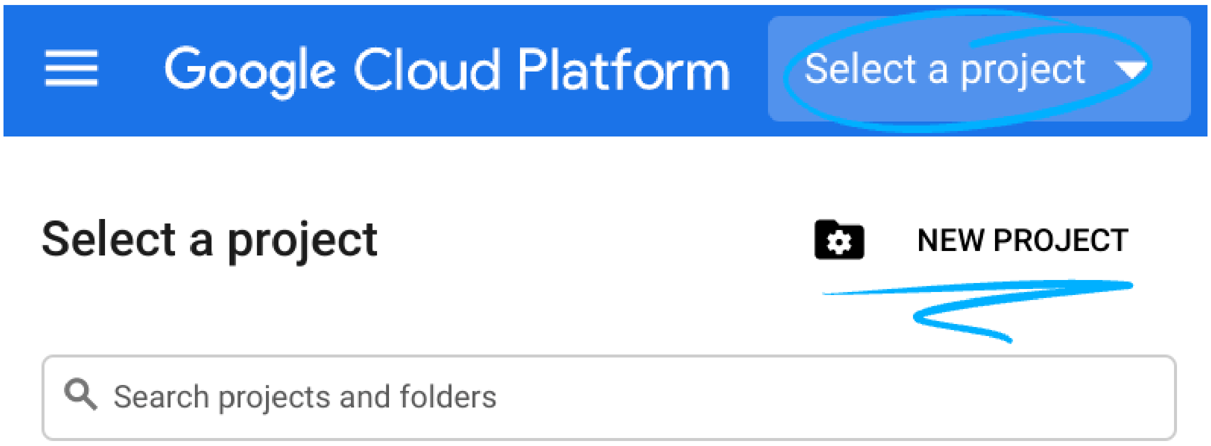
```
gcloud compute firewall-rules create vpc-demo-health-checks \
--allow tcp:80,tcp:443 --network vpc-demo \
--source-ranges 130.211.0.0/22,35.191.0.0/16
```

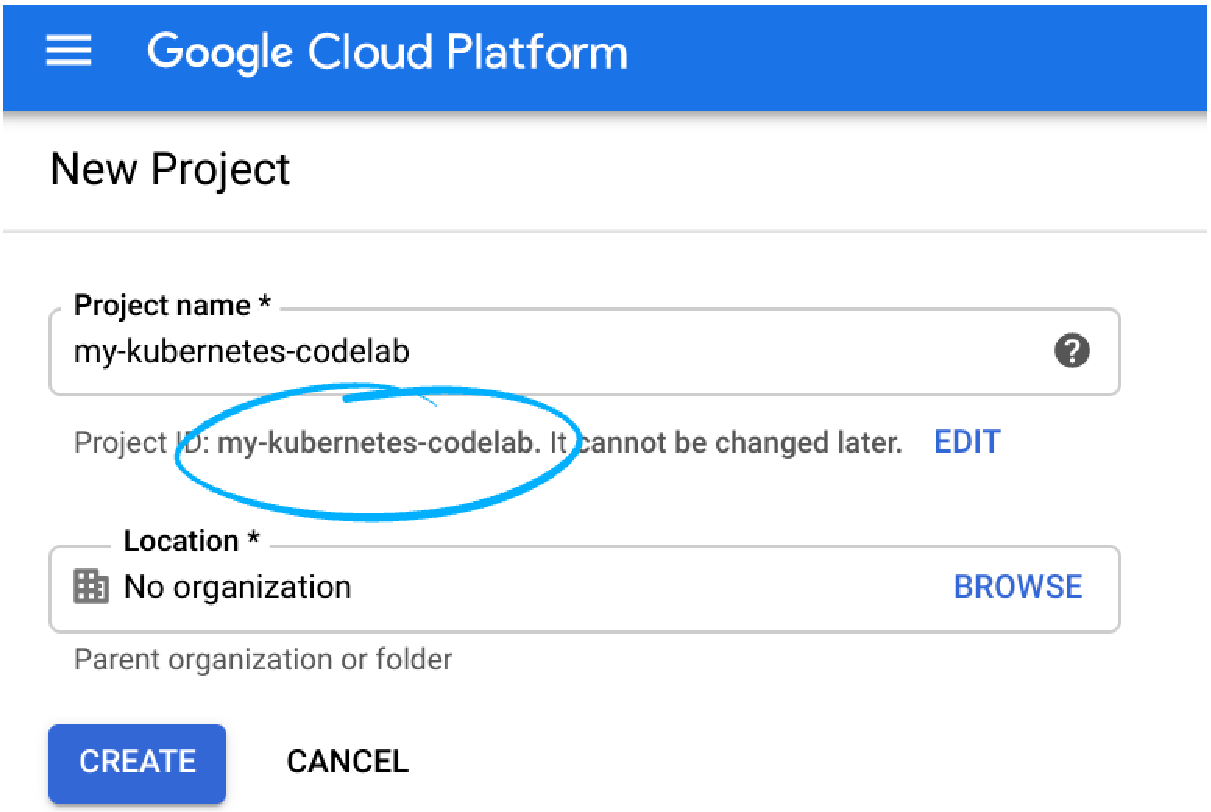

Private Service Connect demo
Google Cloud
Consumer project
Cloud Storage Bucket
Consumer VPC
Private Service Connect Endpoint 100.1.1.1
Private Service Connect Endpoint 10.0.0.100
Client 10.0.0.2
Producer project
Producer VPC
Producer Service Attachment Subnet 192.168.0.0/24
Internal Load Balancer
Producer Forwarding Rule Regional IP 10.0.2.10
www-01
www-02

Self-paced environment setup

1. Sign in to Cloud Console and create a new project or reuse an existing one. If you don't already have a Gmail or Google Workspace account, you must create one.

Note: You can easily access Cloud Console by memorizing its URL, which is console.cloud.google.com.




Google Cloud Platform

New Project

Project name *
my-kubernetes-codelab

Project ID: my-kubernetes-codelab. It cannot be changed later.

Location *
No organization

Parent organization or folder

Remember the project ID, a unique name across all Google Cloud projects (the name above has already been taken and will not work for you, sorry!). It will be referred to later in this codelab as `PROJECT_ID`.

Note: If you're using a Gmail account, you can leave the default location set to **No organization**. If you're using a Google Workspace account, then choose a location that makes sense for your organization.

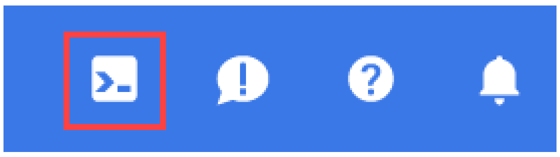
2. Next, you'll need to enable billing in Cloud Console in order to use Google Cloud resources.

Running through this codelab shouldn't cost much, if anything at all. Be sure to to follow any instructions in the "Cleaning up" section which advises you how to shut down resources so you don't incur billing beyond this tutorial. New users of Google Cloud are eligible for the $300 USD Free Trial program.

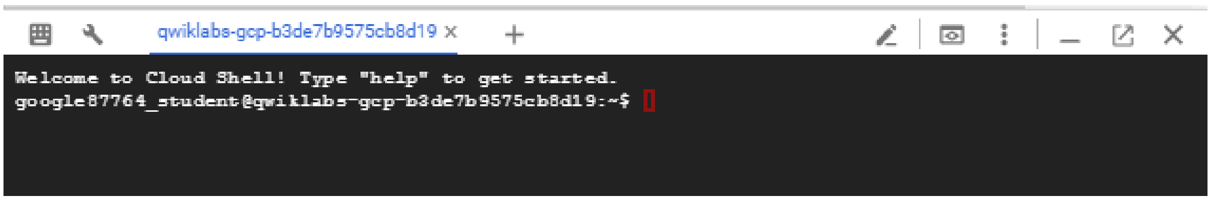
Start Cloud Shell

While Google Cloud can be operated remotely from your laptop, in this codelab you will be using Google Cloud Shell, a command line environment running in the Cloud.

From the GCP Console click the Cloud Shell icon on the top right toolbar:



It should only take a few moments to provision and connect to the environment. When it is finished, you should see something like this:



```
Welcome to Cloud Shell! Type "help" to get started.
google87764_student@qwiklabs-gcp-b3de7b9575cb8d19:~$
```

This virtual machine is loaded with all the development tools you'll need. It offers a persistent 5GB home directory, and runs on Google Cloud, greatly enhancing network performance and authentication. All of your work in this lab can be done with simply a browser.

6. Before you begin

Codelab requires two projects, although not a requirement for PSC. Note the references to support single or multiple projects.

Single Project - Update project to support producer and consumer network

Inside Cloud Shell, make sure that your project id is set up

```
gcloud config list project
gcloud config set project [YOUR-PROJECT-NAME]
prodproject=YOUR-PROJECT-NAME
consumerproject=YOUR-PROJECT-NAME
echo $prodproject
echo $consumerproject
```

Multiple projects - Update project to support producer network

Inside Cloud Shell, make sure that your project id is set up

```
gcloud config list project
gcloud config set project [YOUR-PROJECT-NAME]
prodproject=YOUR-PROJECT-NAME
echo $prodproject
```

7. Create Producers VPC network

Note: In the following section, execute configuration updates in the project that contains your Producer Service

VPC Network

From Cloud Shell

```
gcloud compute networks create vpc-demo-producer --project=$prodproject --subnet-mode=custom
```

Create Subnet

From Cloud Shell

```
gcloud compute networks subnets create vpc-demo-us-west2 --project=$prodproject --
range=10.0.2.0/24 --network=vpc-demo-producer --region=us-west2
```

Create Cloud NAT instance

Cloud NAT is not the same NAT used for PSC. Cloud NAT is used explicitly for internet access to download application packages.

Create Cloud Router

From Cloud Shell

```
gcloud compute routers create crnatprod --network vpc-demo-producer --region us-west2
```

Create Cloud NAT

From Cloud Shell

```
gcloud compute routers nats create cloudnatprod --router=crnatprod --auto-allocate-nat-external-
ips --nat-all-subnet-ip-ranges --enable-logging --region us-west2
```

8. Create compute instances

From Cloud Shell create instance www-01

```
gcloud compute instances create www-01 \
   --zone=us-west2-a \
   --image-family=debian-9 \
   --image-project=debian-cloud \
   --subnet=vpc-demo-us-west2 --no-address \
   --metadata=startup-script='#! /bin/bash
apt-get update
apt-get install tcpdump -y
apt-get install apache2 -y
a2ensite default-ssl
apt-get install iperf3 -y
a2enmod ssl
vm_hostname="$(curl -H "Metadata-Flavor:Google" \
http://169.254.169.254/computeMetadata/v1/instance/name)"
```

```
filter="{print \$NF}"
vm_zone="$(curl -H "Metadata-Flavor:Google" \
http://169.254.169.254/computeMetadata/v1/instance/zone \
| awk -F/ "${filter}")"
echo "Page on $vm_hostname in $vm_zone" | \
tee /var/www/html/index.html
systemctl restart apache2
iperf3 -s -p 5050'
```

From Cloud Shell create instance www-02

```
gcloud compute instances create www-02 \
    --zone=us-west2-a \
    --image-family=debian-9 \
    --image-project=debian-cloud \
    --subnet=vpc-demo-us-west2 --no-address \
    --metadata=startup-script='#! /bin/bash
apt-get update
apt-get install tcpdump -y
apt-get install apache2 -y
a2ensite default-ssl
apt-get install iperf3 -y
a2enmod ssl
vm_hostname="$(curl -H "Metadata-Flavor:Google" \
http://169.254.169.254/computeMetadata/v1/instance/name)"
filter="{print \$NF}"
vm_zone="$(curl -H "Metadata-Flavor:Google" \
http://169.254.169.254/computeMetadata/v1/instance/zone \
| awk -F/ "${filter}")"
echo "Page on $vm_hostname in $vm_zone" | \
tee /var/www/html/index.html
systemctl restart apache2
iperf3 -s -p 5050'
```

9. Create unmanaged instance group

Create a unmanaged instance group consisting of www-01 & www-02

From Cloud Shell

```
gcloud compute instance-groups unmanaged create vpc-demo-ig-www --zone=us-west2-a
```

```
gcloud compute instance-groups unmanaged add-instances vpc-demo-ig-www --zone=us-west2-a --
instances=www-01,www-02
```

```
gcloud compute health-checks create http hc-http-80 --port=80
```

10. Create TCP backend services, forwarding rule & firewall

From Cloud Shell create the backend service

```
gcloud compute backend-services create vpc-demo-www-be-tcp --load-balancing-scheme=internal --protocol=tcp --region=us-west2 --health-checks=hc-http-80
```

```
gcloud compute backend-services add-backend vpc-demo-www-be-tcp --region=us-west2 --instance-group=vpc-demo-ig-www --instance-group-zone=us-west2-a
```

From Cloud Shell create the forwarding rule

```
gcloud compute forwarding-rules create vpc-demo-www-ilb-tcp --region=us-west2 --load-balancing-scheme=internal --network=vpc-demo-producer --subnet=vpc-demo-us-west2 --address=10.0.2.10 --ip-protocol=TCP --ports=all --backend-service=vpc-demo-www-be-tcp --backend-service-region=us-west2
```

From Cloud Shell create a firewall rule to enable backend health checks

```
gcloud compute firewall-rules create vpc-demo-health-checks --allow tcp:80,tcp:443 --network vpc-demo-producer --source-ranges 130.211.0.0/22,35.191.0.0/16 --enable-logging
```

To allow IAP to connect to your VM instances, create a firewall rule that:

- Applies to all VM instances that you want to be accessible by using IAP.
- Allows ingress traffic from the IP range 35.235.240.0/20. This range contains all IP addresses that IAP uses for TCP forwarding.

From Cloud Shell

```
gcloud compute firewall-rules create psclab-iap-prod --network vpc-demo-producer --allow tcp:22 --source-ranges=35.235.240.0/20 --enable-logging
```

11. Create TCP NAT subnet

From Cloud Shell

```
gcloud compute networks subnets create vpc-demo-us-west2-psc-tcp --network=vpc-demo-producer --region=us-west2 --range=192.168.0.0/24 --purpose=private-service-connect
```

12. Create TCP service attachment and firewall rules

From Cloud Shell create the TCP service attachment

```
gcloud compute service-attachments create vpc-demo-psc-west2-tcp --region=us-west2 --producer-forwarding-rule=vpc-demo-www-ilb-tcp --connection-preference=ACCEPT_AUTOMATIC --nat-subnets=vpc-demo-us-west2-psc-tcp
```

Validate the TCP service attachment

```
gcloud compute service-attachments describe vpc-demo-psc-west2-tcp --region us-west2
```

From Cloud Shell create the firewall rule allowing TCP NAT subnet access to the ILB backend

```
gcloud compute --project=$prodproject firewall-rules create vpc-demo-allowpsc-tcp --direction=INGRESS --priority=1000 --network=vpc-demo-producer --action=ALLOW --rules=all --source-ranges=192.168.0.0/24 --enable-logging
```

13. Create Consumers VPC network

Note: In the following section, execute configuration updates in the project that contains your Consumer Service

In the following section the consumer VPC is configured in a separate project. Communication between the consumer and producer network is accomplished through the service attachment defined in the consumers network.

VPC Network

Codelab requires two projects, although not a requirement for PSC. Note the references to support single or multiple projects.

Single Project - Update project to support producer and consumer network

Inside Cloud Shell, make sure that your project id is set up

```
gcloud config list project
gcloud config set project [YOUR-PROJECT-NAME]
consumerproject=YOUR-PROJECT-NAME
prodproject=YOUR-PROJECT-NAME
echo $prodproject
echo $consumerproject
```

Multiple projects - Update project to support consumer a network

Inside Cloud Shell, make sure that your project id is set up

```
gcloud config list project
gcloud config set project [YOUR-PROJECT-NAME]
consumerproject=YOUR-PROJECT-NAME
echo $consumerproject
```

From Cloud Shell

```
gcloud compute networks create vpc-demo-consumer --project=$consumerproject --subnet-mode=custom
```

Create Subnet for PSC

From Cloud Shell

```
gcloud compute networks subnets create consumer-subnet --project=$consumerproject  --range=10.0.60.0/24 --network=vpc-demo-consumer --region=us-west2
```

Create a static IP address for TCP applications

From Cloud Shell

```
gcloud compute addresses create vpc-consumer-psc-tcp --region=us-west2 --subnet=consumer-subnet --addresses 10.0.60.100
```

Create Firewall Rules

To allow IAP to connect to your VM instances, create a firewall rule that:

- Applies to all VM instances that you want to be accessible by using IAP.
- Allows ingress traffic from the IP range 35.235.240.0/20. This range contains all IP addresses that IAP uses for TCP forwarding.

From Cloud Shell

```
gcloud compute firewall-rules create psclab-iap-consumer --network vpc-demo-consumer --allow tcp:22 --source-ranges=35.235.240.0/20 --enable-logging
```

Although not required for PSC create a egress firewall rule to monitor consumer PSC traffic to the producers service attachment

```
gcloud compute --project=$consumerproject firewall-rules create vpc-consumer-psc --direction=EGRESS --priority=1000 --network=vpc-demo-consumer --action=ALLOW --rules=all --destination-ranges=10.0.60.0/24 --enable-logging
```

Create Cloud NAT instance

Cloud NAT is not the same NAT used for PSC. Cloud NAT is used explicitly for internet access to download application packages

Create Cloud Router

From Cloud Shell

```
gcloud compute routers create crnatconsumer --network vpc-demo-consumer --region us-west2
```

Create Cloud NAT

From Cloud Shell

```
gcloud compute routers nats create cloudnatconsumer --router=crnatconsumer --auto-allocate-nat-external-ips --nat-all-subnet-ip-ranges --enable-logging --region us-west2
```

14. Create test instance VM

From Cloud Shell

```
gcloud compute instances create test-instance-1 \
    --zone=us-west2-a \
    --image-family=debian-9 \
    --image-project=debian-cloud \
    --subnet=consumer-subnet --no-address \
    --metadata=startup-script='#! /bin/bash
apt-get update
apt-get install iperf3 -y
apt-get install tcpdump -y'
```

15. Create TCP service attachment

From Cloud Shell

```
gcloud compute forwarding-rules create vpc-consumer-psc-fr-tcp --region=us-west2 --network=vpc-demo-consumer --address=vpc-consumer-psc-tcp --target-service-attachment=projects/$prodproject/regions/us-west2/serviceAttachments/vpc-demo-psc-west2-tcp
```

16. Validation

We will use CURL, TCPDUMP & firewall logs to validate consumer and producer communication.

Within the Consumer's project the static IP addresses are used to originate communication to the Producer. This mapping of static IP address to Consumer forwarding rule is validated by performing the following syntax.

Note: In the following section, execute configuration updates in the project that contains your Consumer Service

From the Consumer VPCs Cloud shell identify the TCP forwarding rule and static IP

```
gcloud compute forwarding-rules describe vpc-consumer-psc-fr-tcp --region us-west2
```

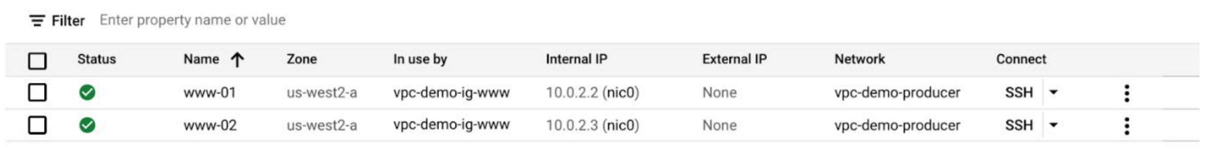
Output:

```
IPAddress: 10.0.60.100
IPProtocol: TCP
creationTimestamp: '2021-07-14T13:34:23.359-07:00'
id: '2768158450402915488'
kind: compute#forwardingRule
labelFingerprint: 42WmSpB8rSM=
name: vpc-consumer-psc-fr-tcp
<snip>
```

17. TCP Validation

Note: In the following section, execute configuration updates in the project that contains your Producer Service

From the **Producer Project,** identify "www-01" & "www-02" and launch one SSH session per instance.



Filter Enter property name or value

	Status	Name ↑	Zone	In use by	Internal IP	External IP	Network	Connect
☐	✓	www-01	us-west2-a	vpc-demo-ig-www	10.0.2.2 (nic0)	None	vpc-demo-producer	SSH
☐	✓	www-02	us-west2-a	vpc-demo-ig-www	10.0.2.3 (nic0)	None	vpc-demo-producer	SSH

From "www-01" perform TCPDUMP to monitor NAT

```
sudo tcpdump -i any net 192.168.0.0/16 -n
```

From "www-02" perform TCPDUMP to monitor NAT

```
sudo tcpdump -i any net 192.168.0.0/16 -n
```

Note: In the following section, execute configuration updates in the project that contains your Consumer Service

From the **Consumer Project,** identify "test-instance-1" and launch two sessions.

From "test-instance-1" session one perform TCPDUMP to monitor consumer

```
sudo tcpdump -i any host 10.0.60.100 -n
```

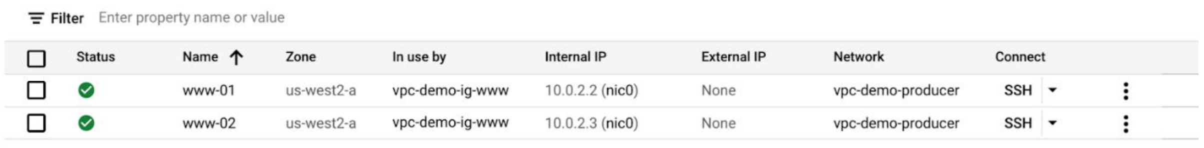
From "test-instance-1" session two perform TCP validation

```
curl -v 10.0.60.100
```

17. TCP Validation

Note: In the following section, execute configuration updates in the project that contains your Producer Service

From the **Producer Project,** identify "www-01" & "www-02" and launch one SSH session per instance.


Filter Enter property name or value

	Status	Name ↑	Zone	In use by	Internal IP	External IP	Network	Connect
☐	✓	www-01	us-west2-a	vpc-demo-ig-www	10.0.2.2 (nic0)	None	vpc-demo-producer	SSH
☐	✓	www-02	us-west2-a	vpc-demo-ig-www	10.0.2.3 (nic0)	None	vpc-demo-producer	SSH

From "www-01" perform TCPDUMP to monitor NAT

```
sudo tcpdump -i any net 192.168.0.0/16 -n
```

From "www-02" perform TCPDUMP to monitor NAT

```
sudo tcpdump -i any net 192.168.0.0/16 -n
```

Note: In the following section, execute configuration updates in the project that contains your Consumer Service

From the **Consumer Project,** identify "test-instance-1" and launch two sessions.

From "test-instance-1" session one perform TCPDUMP to monitor consumer

```
sudo tcpdump -i any host 10.0.60.100 -n
```

From "test-instance-1" session two perform TCP validation

```
curl -v 10.0.60.100
```

18. Observations - Consumer

From "test-instance-1" session two CURL is successful and yields 200 OK.

```
@test-instance-1:~$ curl -v 10.0.60.100
* Rebuilt URL to: 10.0.60.100/
*   Trying 10.0.60.100...
* TCP_NODELAY set
* Connected to 10.0.60.100 (10.0.60.100) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.0.60.100
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Wed, 14 Jul 2021 21:20:22 GMT
< Server: Apache/2.4.25 (Debian)
< Last-Modified: Wed, 14 Jul 2021 20:09:09 GMT
< ETag: "1d-5c71aed5edabd"
< Accept-Ranges: bytes
< Content-Length: 29
< Content-Type: text/html
<
Page on www-01 in us-west2-a
* Curl_http_done: called premature == 0
* Connection #0 to host 10.0.60.100 left intact
```

From "test-instance-1" session one TCPDUMP identifies VM Instance → TCP Static IP communication and response

```
21:20:22.572052 IP 10.0.60.2.59432 > 10.0.60.100.80: Flags [P.], seq 1:76, ack 1, win 222, options
[nop,nop,TS val 634554 ecr 998739], length 75: HTTP: GET / HTTP/1.1

21:20:22.572688 IP 10.0.60.100.80 > 10.0.60.2.59432: Flags [P.], seq 1:257, ack 76, win 220, options
[nop,nop,TS val 998739 ecr 634554], length 256: HTTP: HTTP/1.1 200 OK
```

Firewall logging

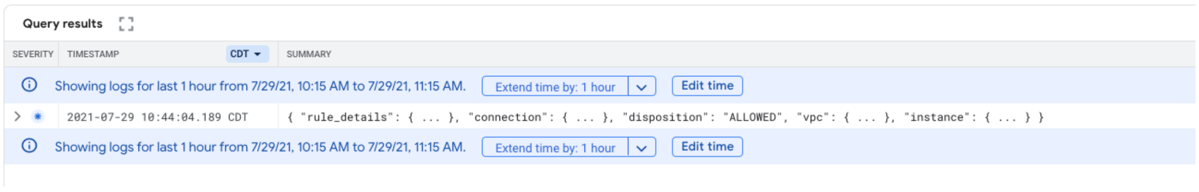
Using Logs Explorer validate firewall rule "vpc-consumner-psc" is capturing the flow between the VM instance and static IP

1. From Cloud Console, Identify Operations Logging → Log Explorer

2. In the Query field update the entry below with yourconsumerproject and select "Run Query"

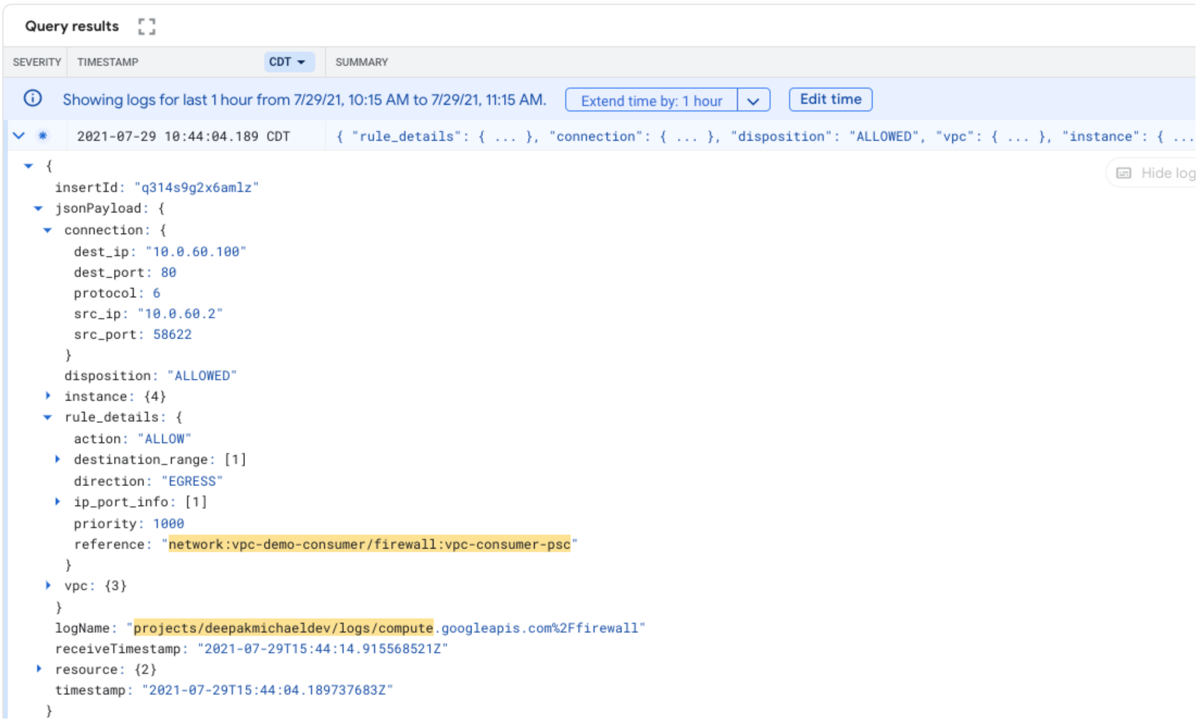
logName:(projects/yourconsumerproject/logs/compute.googleapis.com%2Ffirewall) AND jsonPayload.rule_details.reference:("network:vpc-demo-consumer/firewall:vpc-consumer-psc")

3. Query results provide the following per screenshot provided



```
Query results
SEVERITY  TIMESTAMP  CDT  SUMMARY
Showing logs for last 1 hour from 7/29/21, 10:15 AM to 7/29/21, 11:15 AM.  Extend time by: 1 hour  Edit time
2021-07-29 10:44:04.189 CDT   { "rule_details": { ... }, "connection": { ... }, "disposition": "ALLOWED", "vpc": { ... }, "instance": { ... } }
Showing logs for last 1 hour from 7/29/21, 10:15 AM to 7/29/21, 11:15 AM.  Extend time by: 1 hour  Edit time
```

4. Expand the log and identify the output provided below. Take note of the dest_ip: 10.0.60.100 is the STATIC TCP IP and src_ip: 10.0.60.2 is the VM instance IP address



```
Query results
SEVERITY  TIMESTAMP  CDT  SUMMARY
Showing logs for last 1 hour from 7/29/21, 10:15 AM to 7/29/21, 11:15 AM.  Extend time by: 1 hour  Edit time
2021-07-29 10:44:04.189 CDT   { "rule_details": { ... }, "connection": { ... }, "disposition": "ALLOWED", "vpc": { ... }, "instance": { ...
{
  insertId: "q314s9g2x6amlz"
  jsonPayload: {
   connection: {
     dest_ip: "10.0.60.100"
     dest_port: 80
     protocol: 6
     src_ip: "10.0.60.2"
     src_port: 58622
   }
   disposition: "ALLOWED"
   instance: {4}
   rule_details: {
     action: "ALLOW"
     destination_range: [1]
     direction: "EGRESS"
     ip_port_info: [1]
     priority: 1000
     reference: "network:vpc-demo-consumer/firewall:vpc-consumer-psc"
   }
   vpc: {3}
  }
  logName: "projects/deepakmichaeldev/logs/compute.googleapis.com%2Ffirewall"
  receiveTimestamp: "2021-07-29T15:44:14.915568521Z"
  resource: {2}
  timestamp: "2021-07-29T15:44:04.189737683Z"
}
```

19. Observations - Producer

From the backend instance "www-01" or "www-02" the following communication between the TCP NAT subnet and TCP ILB is observed.

21:20:22.572186 IP 192.168.0.2.1024 > 10.0.2.10.80: Flags [P.], seq 1:76, ack 1, win 222, options [nop,nop,TS val 634554 ecr 998739], length 75: HTTP: GET / HTTP/1.1

21:20:22.572679 IP 10.0.2.10.80 > 192.168.0.2.1024: Flags [P.], seq 1:257, ack 76, win 220, options [nop,nop,TS val 998739 ecr 634554], length 256: HTTP: HTTP/1.1 200 OK

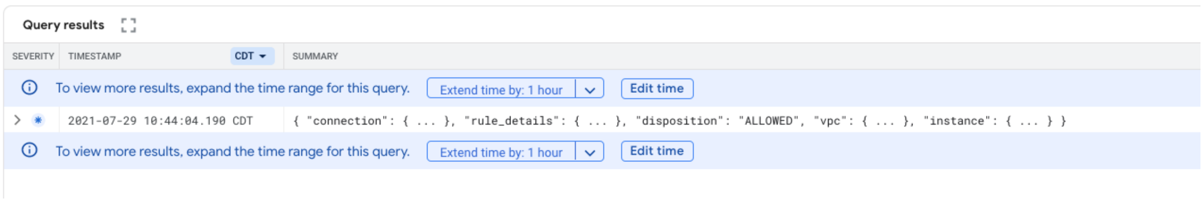
20. Firewall logging

Using Logs Explorer validate firewall rule "vpc-demo-allowpsc-tcp" is capturing the TCP NAT & TCP ILB flow by performing the following steps:

1. From Cloud Console, Identify Operations Logging → Log Explorer
2. In the Query field update the entry below with yourprodproject and select "Run Query"

logName:(projects/yourprodproject/logs/compute.googleapis.com%2Ffirewall) AND jsonPayload.rule_details.reference:("network:vpc-demo-producer/firewall:vpc-demo-allowpsc-tcp")

3. Query results provide the following per screenshot provided

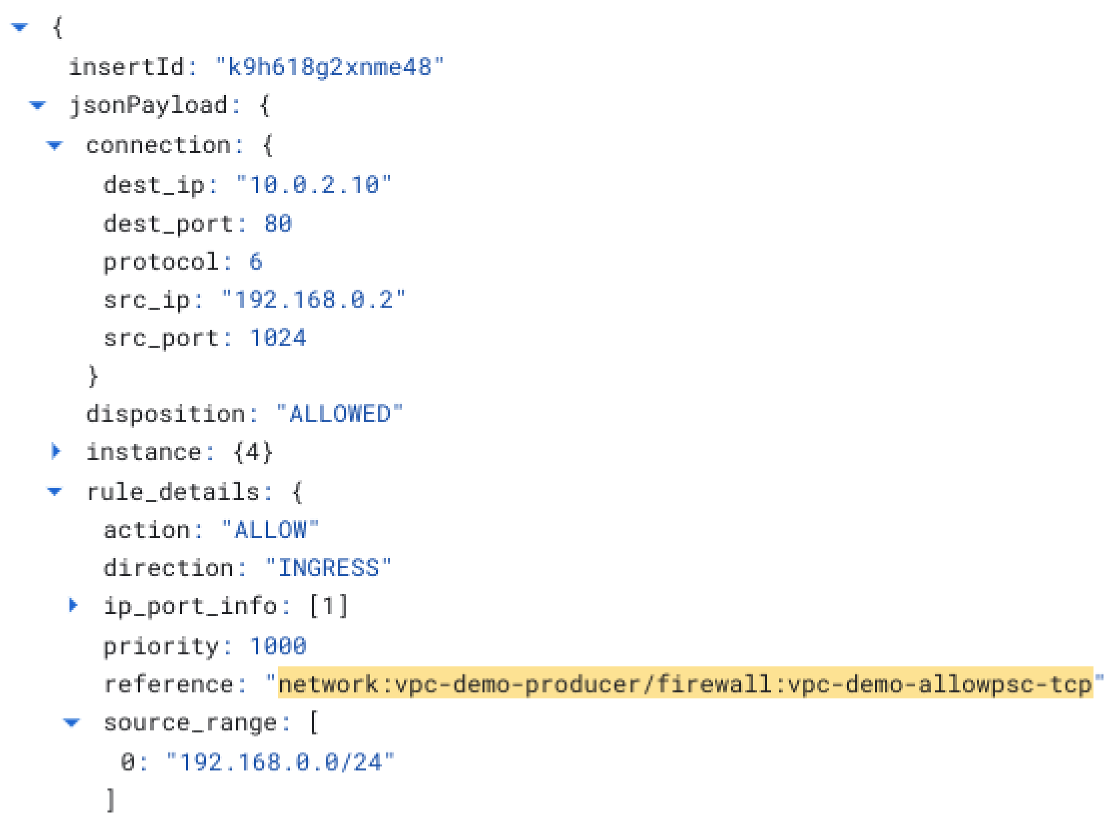


To view more results, expand the time range for this query.

```
2021-07-29 10:44:04.190 CDT    { "connection": { ... }, "rule_details": { ... }, "disposition": "ALLOWED", "vpc": { ... }, "instance": { ... } }
```

To view more results, expand the time range for this query.

4. Expand the log and identify the output provided below. Take note of the TCP ILB dest_ip: 10.0.2.10 and NAT TCP source_range (192.168.0.0/24) & the respective src_ip: 192.168.0.2.



```
▾ {
    insertId: "k9h618g2xnme48"
  ▾ jsonPayload: {
    ▾ connection: {
        dest_ip: "10.0.2.10"
        dest_port: 80
        protocol: 6
        src_ip: "192.168.0.2"
        src_port: 1024
      }
      disposition: "ALLOWED"
    ▸ instance: {4}
    ▾ rule_details: {
        action: "ALLOW"
        direction: "INGRESS"
      ▸ ip_port_info: [1]
        priority: 1000
        reference: "network:vpc-demo-producer/firewall:vpc-demo-allowpsc-tcp"
      ▾ source_range: [
          0: "192.168.0.0/24"
        ]
```

21. Enable Proxy Protocol

By default, Private Service Connect translates the consumer's source IP address to an address in one of the Private Service Connect subnets in the service producer's VPC network. If you want to see the consumer's original source IP address instead, you can enable PROXY protocol. If PROXY protocol is enabled, you can get the consumer's source IP address and PSC connection ID from the PROXY protocol header

Important: Due to a known issue with Preview, if you disable or enable PROXY protocol after a service attachment is created, the configuration does not change. However, the status shown in the service attachment details incorrectly shows that the status has changed. To enable or disable PROXY protocol, delete the service attachment and recreate it with the correct PROXY protocol configuration.

Reference to documentation

Delete the Producers Published Services

Note: In the following section, execute configuration updates in the project that contains your Producer Service

From cloud shell delete the TCP service attachments

```
gcloud compute service-attachments delete vpc-demo-psc-west2-tcp --region=us-west2 --quiet
```

From cloud shell validate service attachments are deleted (Listed 0 items)

```
gcloud compute service-attachments list
```

Create TCP service attachment with proxy protocol enabled

```
gcloud compute service-attachments create vpc-demo-psc-west2-tcp --region=us-west2 \
--producer-forwarding-rule=vpc-demo-www-ilb-tcp \
--connection-preference=ACCEPT_AUTOMATIC \
--nat-subnets=vpc-demo-us-west2-psc-tcp \
--enable-proxy-protocol
```

From cloud shell validate service attachments are created with proxy protocol enabled (true)

```
gcloud compute service-attachments describe vpc-demo-psc-west2-tcp --region=us-west2 | grep -i
enableProxyProtocol:
```

Note: In the following section, execute configuration updates in the project that contains your Consumer Service

From cloud shell delete the TCP forwarding rules

```
gcloud compute forwarding-rules delete vpc-consumer-psc-fr-tcp --region=us-west2 --quiet
```

Recreate the TCP forwarding rules to associate with the previously created (producer) service attachment

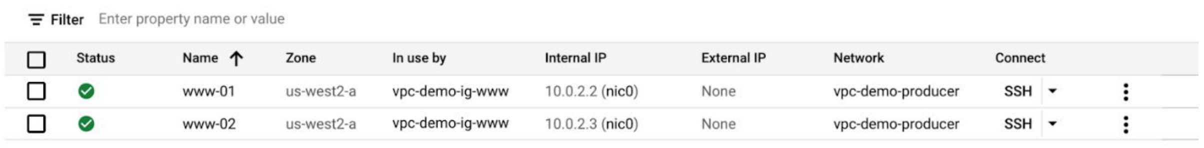
From cloud shell create the TCP forwarding rule

```
gcloud compute forwarding-rules create vpc-consumer-psc-fr-tcp \
--region=us-west2 --network=vpc-demo-consumer \
--address=vpc-consumer-psc-tcp \
--target-service-attachment=projects/$prodproject/regions/us-west2/serviceAttachments/vpc-
demo-psc-west2-tcp
```

Proxy Protocol Validation

Note: In the following section, execute configuration updates in the project that contains your Producer Service

From the **Producer Project,** identify "www-01" & "www-02" and launch one session per instance.


Filter Enter property name or value

	Status	Name ↑	Zone	In use by	Internal IP	External IP	Network	Connect
☐	✓	www-01	us-west2-a	vpc-demo-ig-www	10.0.2.2 (nic0)	None	vpc-demo-producer	SSH
☐	✓	www-02	us-west2-a	vpc-demo-ig-www	10.0.2.3 (nic0)	None	vpc-demo-producer	SSH

From "www-01" perform TCPDUMP to monitor NAT

```
sudo tcpdump -nnvvXSs 1514 net 192.168.0.0/16
```

From "www-02" perform TCPDUMP to monitor NAT

```
sudo tcpdump -nnvvXSs 1514 net 192.168.0.0/16
```

Note: In the following section, execute configuration updates in the project that contains your Consumer Service

From the **Consumer Project,** identify "test-instance-1" and launch a single session

From "test-instance-1" session perform a curl

```
curl 10.0.60.100
```

Observations - Consumer

Note that if PROXY Protocol v2 is enabled but the application is not configured to support it, an error message will be displayed if connecting from the client as per the example below. Apache updates are required to accommodate the additional proxy v2 header & not covered in the codelab.

From "test-instance-1" session CURL will produce an expected 400 Bad requests although the backend query is successful.

```
@test-instance-1:~$ curl 10.0.60.100
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.25 (Debian) Server at www-02.c.deepakmichaelprod.internal Port
80</address>
```

Observations - Consumer

From the backend instance "www-01" or "www-02" the following communication between the TCP NAT subnet and TCP ILB are observed with proxy protocol embedded in the capture.

In most cases, the 3rd packet in the tcpdump contains the relevant proxy protocol information elements (IE). Optionally, identify the packet with 39 bytes that contain proxy protocol IE.

```
192.168.0.3.1025 > 10.0.2.10.80: Flags [P.], cksum 0xb617 (correct), seq 2729454396:2729454435,
ack 1311105819, win 28160, length 39: HTTP
	0x0000:  4500 004f 0000 4000 4006 6df4 c0a8 0003  E..O..@.@.m.....
	0x0010:  0a00 020a 0401 0050 a2b0 2b3c 4e25 e31b  .......P..+<N%..
	0x0020:  5018 6e00 b617 0000 0d0a 0d0a 000d 0a51  P.n............Q
	0x0030:  5549 540a 2111 0017 0a00 3c02 0a00 3c64  UIT.!.....<...<d
	0x0040:  8138 0050 e000 0800 9b34 d70a 003c 64    .8.P.....4...<d
```

Identify the PROXY Protocol Signature: 0d0a0d0a000d0a515549540a in the packet capture

By identifying the PROXY Protocol Signature, it's possible to break it down and decode the fields as below:

PROXY Protocol Signature: 0d0a0d0a000d0a515549540a

Other PROXY Protocol Fields: 21 11 00 17

IPs and Ports: 0a003c02 0a003c64 8138 0050

TLV Type: e0

TLV Length: 00 08

pscConnection ID: 009b34d70a003c64

	Hex	Decimal / IP
PROXY Protocol Signature	0d0a0d0a000d0a515549540a	
Version, Protocol, Length	21 11 0017	
Src IP	0a003c02	10.0.60.2
Dst IP	0a003c64	10.0.60.100
Src Port	8138	33080
Dst Port	0050	80
TLV Type (PP2_TYPE_GCP)	e0	

TLV Length	0008	
pscConnectionId	00004dde290a003c64	43686719580552292

The pscConnectionId can also be validated by describing the consumer forwarding rule as below, and making sure it matches:

Note: In the following section, execute configuration updates in the project that contains your Consumer Service

From cloud shell describe the TCP forwarding rules

```
gcloud compute forwarding-rules describe vpc-consumer-psc-fr-tcp --region=us-west2
```

Output describing the pscConnectionID

```
$ gcloud compute forwarding-rules describe vpc-consumer-psc-fr-tcp --region=us-west2
IPAddress: 10.0.60.100
IPProtocol: TCP
creationTimestamp: '2021-07-14T16:50:31.766-07:00'
id: '4443494505307621032'
kind: compute#forwardingRule
labelFingerprint: 42WmSpB8rSM=
name: vpc-consumer-psc-fr-tcp
network:
https://www.googleapis.com/compute/v1/projects/deepakmichaeldev/global/networks/vpc-
demo-consumer
networkTier: PREMIUM
pscConnectionId: '43686719580552292'
pscConnectionStatus: ACCEPTED
```

22. Connection Policy

You can switch between automatic and explicit project acceptance for a published service.

Changing from automatic acceptance to explicit acceptance does not affect consumer endpoints that had connected to the service before this change. Existing consumer endpoints can connect to the published service until the service attachment is deleted. New consumer endpoints must be accepted before they can connect to the service. See Managing requests for access to a published service for more information.

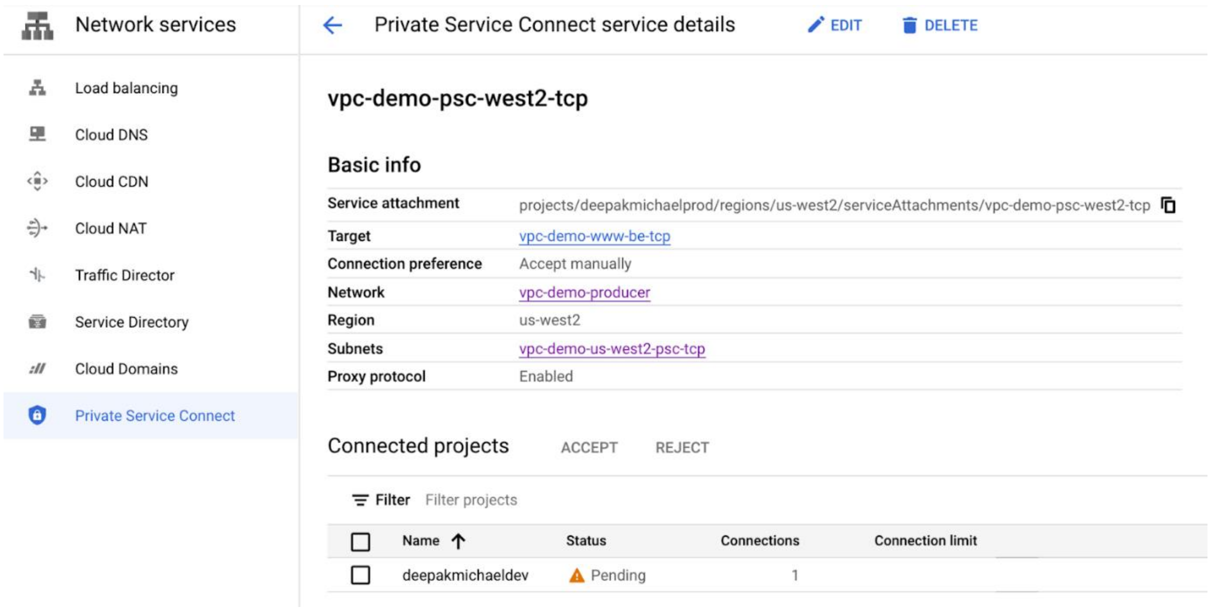
In this section of the lab, you will modify the producer's connection policy to explicitly approve the consumer's service attachment.

Note: In the following section, execute configuration updates in the project that contains your Producer Service

From the producers service cloud shell update the connection-preference policy from accept automatically to **accept manually**

```
gcloud compute service-attachments update vpc-demo-psc-west2-tcp --region=us-west2 --
connection-preference ACCEPT_MANUAL
```

Identify the endpoint status by navigating to Network Services → Private Service Connect → Published Services → vpc-demo-psc-west2-tcp → Connected Projects



vpc-demo-psc-west2-tcp

Basic info

Service attachment	projects/deepakmichaelprod/regions/us-west2/serviceAttachments/vpc-demo-psc-west2-tcp
Target	vpc-demo-www-be-tcp
Connection preference	Accept manually
Network	vpc-demo-producer
Region	us-west2
Subnets	vpc-demo-us-west2-psc-tcp
Proxy protocol	Enabled

Connected projects ACCEPT REJECT

Filter Filter projects

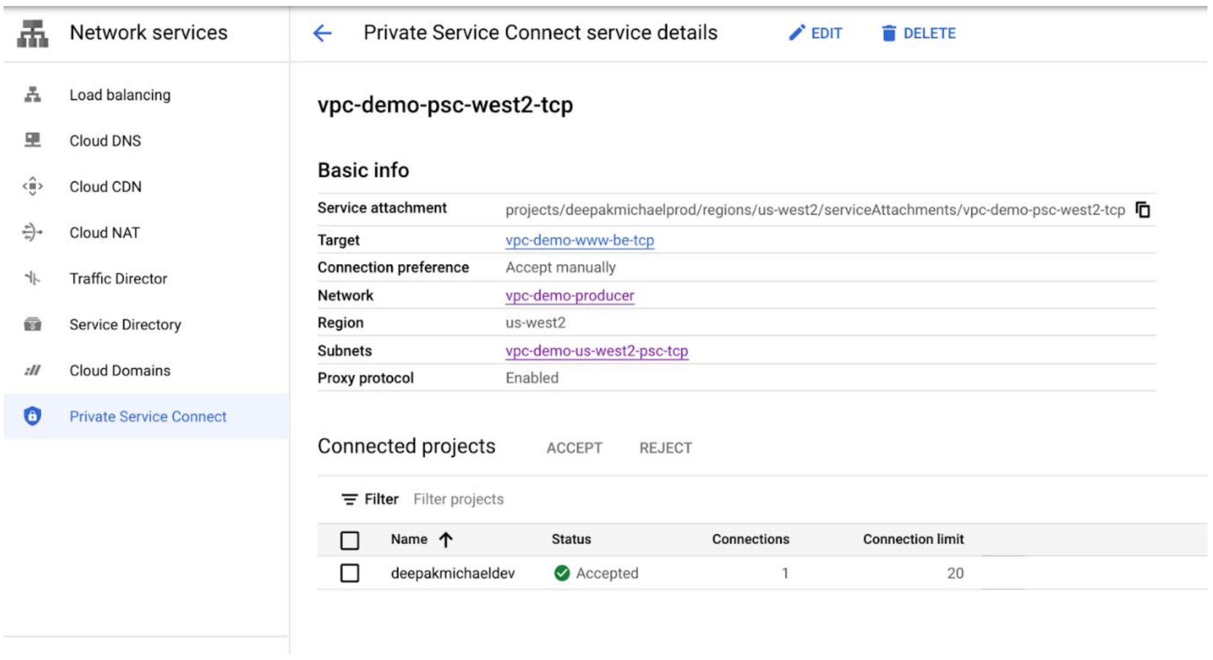
Name	Status	Connections	Connection limit
deepakmichaeldev	Pending	1	

Notice, Consumer project changed to "Pending" status under Connected Projects.

Accept the consumers project by executing the following in cloud shell, ensure to update with the appropriate project name

```
gcloud compute service-attachments update vpc-demo-psc-west2-tcp --region=us-west2 --
consumer-accept-list $consumerproject=20
```

Identify the endpoint status by navigating to Network Services → Private Service Connect → Published Services → vpc-demo-psc-west2-tcp → Connected Projects



vpc-demo-psc-west2-tcp

Basic info

Service attachment	projects/deepakmichaelprod/regions/us-west2/serviceAttachments/vpc-demo-psc-west2-tcp
Target	vpc-demo-www-be-tcp
Connection preference	Accept manually
Network	vpc-demo-producer
Region	us-west2
Subnets	vpc-demo-us-west2-psc-tcp
Proxy protocol	Enabled

Connected projects ACCEPT REJECT

Filter Filter projects

Name	Status	Connections	Connection limit
deepakmichaeldev	Accepted	1	20

Notice, Consumer project changed to "Accepted" status under Connected Projects.

23. Cleanup steps

Producer network clean up steps

Note: In the following section, execute configuration updates in the project that contains your Producer Service

From a single cloud shell in the Producer project terminal delete lab components

```
gcloud compute routers nats delete cloudnatprod --router=crnatprod --region=us-west2 --quiet

gcloud compute routers delete crnatprod --region=us-west2 --quiet

gcloud compute instances delete www-01 --zone=us-west2-a --quiet

gcloud compute instances delete www-02 --zone=us-west2-a --quiet

gcloud compute service-attachments delete vpc-demo-psc-west2-tcp --region=us-west2 --quiet

gcloud compute forwarding-rules delete vpc-demo-www-ilb-tcp --region=us-west2 --quiet

gcloud compute backend-services delete vpc-demo-www-be-tcp --region=us-west2 --quiet

gcloud compute instance-groups unmanaged delete vpc-demo-ig-www --zone=us-west2-a --quiet
gcloud compute health-checks delete hc-http-80 --quiet

gcloud compute firewall-rules delete vpc-demo-allowpsc-tcp --quiet

gcloud compute firewall-rules delete vpc-demo-health-checks --quiet

gcloud compute firewall-rules delete psclab-iap-prod --quiet

gcloud compute networks subnets delete vpc-demo-us-west2 --region=us-west2 --quiet

gcloud compute networks subnets delete vpc-demo-us-west2-psc-tcp --region=us-west2 --quiet

gcloud compute networks delete vpc-demo-producer --quiet
```

Note: In the following section, execute configuration updates in the project that contains your Consumer Service

Consumer network clean up steps

From a single cloud shell in the Producer project terminal delete lab components

```
gcloud compute routers nats delete cloudnatconsumer --router=crnatconsumer --region=us-west2 --
quiet

gcloud compute routers delete crnatconsumer --region=us-west2 --quiet

gcloud compute instances delete test-instance-1 --zone=us-west2-a --quiet

gcloud compute forwarding-rules delete vpc-consumer-psc-fr-tcp --region=us-west2 --quiet

gcloud compute addresses delete vpc-consumer-psc-tcp --region=us-west2 --quiet

gcloud compute firewall-rules delete psclab-iap-consumer --quiet

gcloud compute networks subnets delete consumer-subnet --region=us-west2 --quiet

gcloud compute firewall-rules delete vpc-consumer-psc --quiet

gcloud compute networks delete vpc-demo-consumer --quiet
```