

# Configuring Egress from a Static Outbound IP Address

## Overview

By default, a Cloud Run service will connect to external endpoints on the internet using a dynamic IP address pool. This method is not suitable if the Cloud Run service needs to connect to an external endpoint that requires connections originating from a static IP address, for example, as a database or API using an IP address-based firewall. For those types of connections, you must configure your Cloud Run service to route requests through a static IP address.

This lab teaches the process of configuring a Cloud Run service to send requests using a static IP address.

## Prerequisites

These labs are based on intermediate knowledge of Google Cloud. While the steps required are covered in the content, it would be helpful to have familiarity with any of the following products:

- Static IP Addresses
- Cloud Run

## Objectives

In this lab, you learn to:

- Build and publish a sample Cloud Run service and image using code from the Buildpack samples repository.
- Create a subnetwork and a VPC access connector.
- Configure network address translation (NAT).
- Route your Cloud Run traffic through the VPC network.

## **Task 1. Enable the Cloud Run API and configure your Shell environment for flexibility**

1. From Cloud Shell, enable the Cloud Run API :

2. `gcloud services enable run.googleapis.com`
3. If you are asked to authorize the use of your credentials, do so. You should then see a successful message similar to this one:
4. `Operation "operations/acf.cc11852d-40af-47ad-9d59-477a12847c9e" finished successfully.`
5. **Note:** You can also enable the API using the **APIs & Services** section of the console.
6. Create a LOCATION environment variable:
7. `LOCATION=us-east1`
8. Set the compute region:
9. `gcloud config set compute/region $LOCATION`

## Task 2. Create a service

### Clone the buildpack-samples repository

- Clone the buildpacks samples repository to your local directory using the command below:
- `git clone https://github.com/GoogleCloudPlatform/buildpack-samples.git`

## Copy a sample application

- Copy an existing application that you will use later to create a second service.
- `cp -R buildpack-samples/sample-go`  
`buildpack-samples/sample-go-service`

## Edit the sample application

1. Change to the sample code folder:
2. `cd buildpack-samples/sample-go`
3. Using any editor, edit the file `hello.go` in the `sample-go` folder, and add the `/service` http handler to the `main` function:

```
http.HandleFunc("/service", func(w http.ResponseWriter, r *http.Request) {  
    serviceURL := os.Getenv("SERVICE_B_URL")  
    if serviceURL == "" {  
        log.Fatalln("Service URL environment variable is not set.")  
    }  
    resp, err := http.Get(serviceURL)  
    if err != nil {  
        log.Fatalln(err)  
    }  
    defer resp.Body.Close()  
    body, err := ioutil.ReadAll(resp.Body)  
    if err != nil {  
        log.Fatalln(err)  
    }  
    log.Println(string(body))  
    fmt.Fprint(w, string(body))  
4. })
```

5. The function uses a URL from an environment variable to invoke another internal Cloud Run service.

6. Update the `import` statement to add the `io/ioutil` package:

```
import (  
    "fmt"  
    "log"  
    "net/http"  
    "os"  
    "io/ioutil"  
7. )
```

8. Build an application container for your sample application with the `pack` command. If asked to provide authorization for the Shell, give it permission to run.
9. `pack build --builder=gcr.io/buildpacks/builder sample-go`

## Run your new container in Docker

1. Run the command below to bind the sample app created in the previous step to port 8080, so that it can be accessed via the browser:
2. `docker run -it -e PORT=8080 -p 8080:8080 sample-go`
3. In the Cloud Shell window, click on the **Web Preview** icon and select **Preview on port 8080**.  
  
A new browser window or tab will open showing the "hello, world" message.  
Success!
4. Back in the console window, click `Control C` (Control and C key held down at the same time) to stop the web service and return control of the command line to the Cloud Shell.

## Publish the image

1. First, set the default Buildpacks builder:
2. `pack config default-builder gcr.io/buildpacks/builder:v1`
3. Publish your container image:
4. `pack build --publish gcr.io/$GOOGLE_CLOUD_PROJECT/sample-go`

You are now ready for the next part of the lab.

## Task 3. Create a subnetwork

Creating a subnetwork for your Serverless VPC Access connector to reside in ensures that other compute resources in your VPC, such as Compute Engine VMs or Google Kubernetes Engine clusters, do not accidentally use the static IP you have configured to access the internet.

1. First, you will need to find the name of your VPC network by calling up a list of the networks associated with your account, using Cloud shell:
2. `gcloud compute networks list`
3. You should see something like the following in the output, identifying the networks you can use when attaching your Serverless VPC Access connector.  
For this lab, the only network that will be available is the `default` network, so the information returned will be similar to the output shown below:

NAME: default

SUBNET\_MODE: AUTO

BGP\_ROUTING\_MODE: REGIONAL

IPV4\_RANGE:

4. GATEWAY\_IPV4:

5. Create a subnetwork called **mysubnet** with a range of **192.168.0.0/28** specified in CIDR format, to be used for your Serverless VPC Access connector.

To create this subnetwork in the project's **default** network (note also that ), run the command:

```
gcloud compute networks subnets create mysubnet \
```

6. --range=192.168.0.0/28 --network=default --region=\$LOCATION

7. **Note:** To avoid errors later in the process, use a netmask of 28 for subnets that are used for VPC connectors.

## Task 4. Create a serverless VPC Access connector

To route your Cloud Run service's outbound traffic to a VPC network, you first need to set up a Serverless VPC Access connector.

1. Create a Serverless VPC Access connector named **myconnector** with your previously created subnetwork named **mysubnet** using the command:

```
gcloud compute networks vpc-access connectors create myconnector \
```

- ```
--region=$LOCATION \  
--subnet-project=$GOOGLE_CLOUD_PROJECT \  
2.    --subnet=mysubnet
```
3. If prompted to enable the `vpcaccess.googleapis.com` on your project, type **y**.  
Wait for the process to complete before proceeding.

## Task 5. Configure network address translation (NAT)

To route outbound requests to external endpoints through a static IP (which is the main goal of this lab), you must first configure a Cloud NAT gateway.

1. Create a new Cloud Router to program your NAT gateway:

```
gcloud compute routers create myrouter \  
--network=default \  
2.    --region=$LOCATION
```

3. Next, reserve a static IP address using the command below, where `myoriginip` is the name being assigned to your IP address resource.

A reserved IP address resource retains the underlying IP address when the resource it is associated with is deleted and re-created. Using the same region as your Cloud NAT router will help to minimize latency and network costs.

4. `gcloud compute addresses create myoriginip --region=$LOCATION`
5. To route outbound requests to external endpoints through a static IP, you must configure a Cloud NAT gateway.



6. Bring all of the resources you've just created together to create a Cloud NAT gateway named `mynat`.

To configure your router to route the traffic originating from the VPC network, run the command:

```
gcloud compute routers nats create mynat \  
  --router=myrouter \  
  --region=$LOCATION \  
  --nat-custom-subnet-ip-ranges=mysubnet \  
  7. --nat-external-ip-pool=myoriginip
```

## Task 6. Route Cloud Run traffic through the VPC network

1. After NAT has been configured, deploy your Cloud Run service with the Serverless VPC Access connector, and set VPC egress to route all traffic through the VPC network:

```
gcloud run deploy sample-go \  
  --image=gcr.io/$GOOGLE_CLOUD_PROJECT/sample-go \  
  --vpc-connector=myconnector \  
  --vpc-egress=all-traffic \  
  --region $LOCATION \  
  2. --allow-unauthenticated
```

3. If prompted to enable `run.googleapis.com` for your project, type `y`.

If successful, after a few minutes you should see output similar to that below:

```
Deploying container to Cloud Run service [sample-go] in project
[qwiklabs-gcp-01-825e0adf46fc] region [us-central1]
✓ Deploying new service... Done.
✓ Creating Revision...
✓ Routing traffic...
✓ Setting IAM Policy...
Done.
Service [sample-go] revision [sample-go-00002-yax] has been deployed and
is serving 100 percent of traffic.
```

4. Service URL: `https://sample-go-lawvw7teya-uc.a.run.app`

5. Click on the Service URL.

You should see the same "hello, world" message that you saw displayed in the web preview pane in Task 1.

## Task 7. Create a receiving service

To test service to service communication, you create a second service that will respond to requests on its service URL. The service will be configured to only serve internal traffic routed to it using the VPC service connector.

### Create the service

1. Change to the sample code folder:
2. `cd ~/buildpack-samples/sample-go-service`

3. Using any editor, edit the file `hello.go` in the `sample-go-service` folder, and update the `/` http handler in the `main` function:

```
http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {  
    fmt.Fprint(w, "hello, from service B")  
4. })
```

5. Edit the `go.mod` file in this folder to update the module entry:

6. module

```
github.com/GoogleCloudPlatform/buildpack-samples/sample-go-service
```

## Build and deploy the service to Cloud Run

1. Build the second service from source, and deploy it to Cloud Run with the Serverless VPC Access connector allowing internal traffic only:

```
gcloud run deploy sample-go-service \  
--source . \  
--port=8081 \  
--vpc-connector=myconnector \  
--ingress=internal \  
--region $LOCATION \  
2. --allow-unauthenticated
```

3. To create the Artifact registry repository, type **Y** at the prompt.

If successful, after a few minutes you should see output from the command that contains the service URL.

4. Click on the Service URL.

You should not receive any response from the service since the service is configured to accept internal requests only from within the VPC network or VPC service control perimeter.

## Task 8. Test service communication

To verify internal service communication, test the services by making a request to the original Cloud Run service. You'll also verify the originating IP address of the calling service.

### Verify service to service communication

1. First, update the original Cloud Run service with an environment variable that contains the URL of the second Cloud Run service:

```
gcloud run services update sample-go \
--region $LOCATION \
```

2.

```
--set-env-vars=SERVICE_B_URL=https://sample-go-service-ktyskcmazq-uc
.a.run.app
```

3. **Note:** Replace the SERVICE\_URL value with the URL to your own second Cloud Run service from the previous step.
4. Use curl with the original service URL, and append the `/service` request path:
5. `curl https://sample-go-lawvw7teya-uc.a.run.app/service`
6. **Note:** Replace the curl URL with the URL to your own original Cloud Run service.
7. The output from the curl command contains the response from the second Cloud Run service B, which was invoked by the original calling service:

8. `hello, from service B`

## Verify the originating IP address of the calling service

1. Update the service URL environment variable to a service that verifies the originating IP address:

```
gcloud run services update sample-go \
--region $LOCATION \
```

2. `--set-env-vars=SERVICE_B_URL=http://curlmyip.org`

3. Copied!


4. `content_copy`

5. Use curl with the original service URL, and append the `/service` request path:

6. `curl https://sample-go-lawvw7teya-uc.a.run.app/service`

7. **Note:** Replace the curl URL with the URL to your own original Cloud Run service.

8. The output from the curl command contains the originating IP address.

9. In the Google Cloud console, use the **Navigation menu** () , and navigate to **VPC Network > IP Addresses**.

10. Verify that the `myoriginip` static IP address value matches the IP address in the output of the curl command.

# Congratulations!

After completing the above steps, you have:

- Set up Cloud NAT on your VPC network with a predefined static IP address.
- Routed all of your Cloud Run service's outbound traffic into your VPC network.
- Sent requests from your Cloud Run service to internal resources (another Cloud Run service) in your VPC network.
- Confirmed that 100% of your traffic for the sample-go service is now flowing through your new VPC connector.