a Beta release of Configuring failover for Internal TCP/UDP Load Balancing. This feature is not covered by any SLA c
cation policy and might be subject to backward-incompatible changes.

This guide uses an example to teach you how to configure failover for a Google Cloud internal TCP/UDP load balancer. Before following this guide, familiarize yourself with the following:

- Internal TCP/UDP Load Balancing concepts (/load-balancing/docs/internal/index)
- Failover concepts for Internal TCP/UDP Load Balancing (/load-balancing/docs/internal/failover-overview)
- Firewall rules overview (/vpc/docs/firewalls)
- Health check concepts (/load-balancing/docs/health-check-concepts)

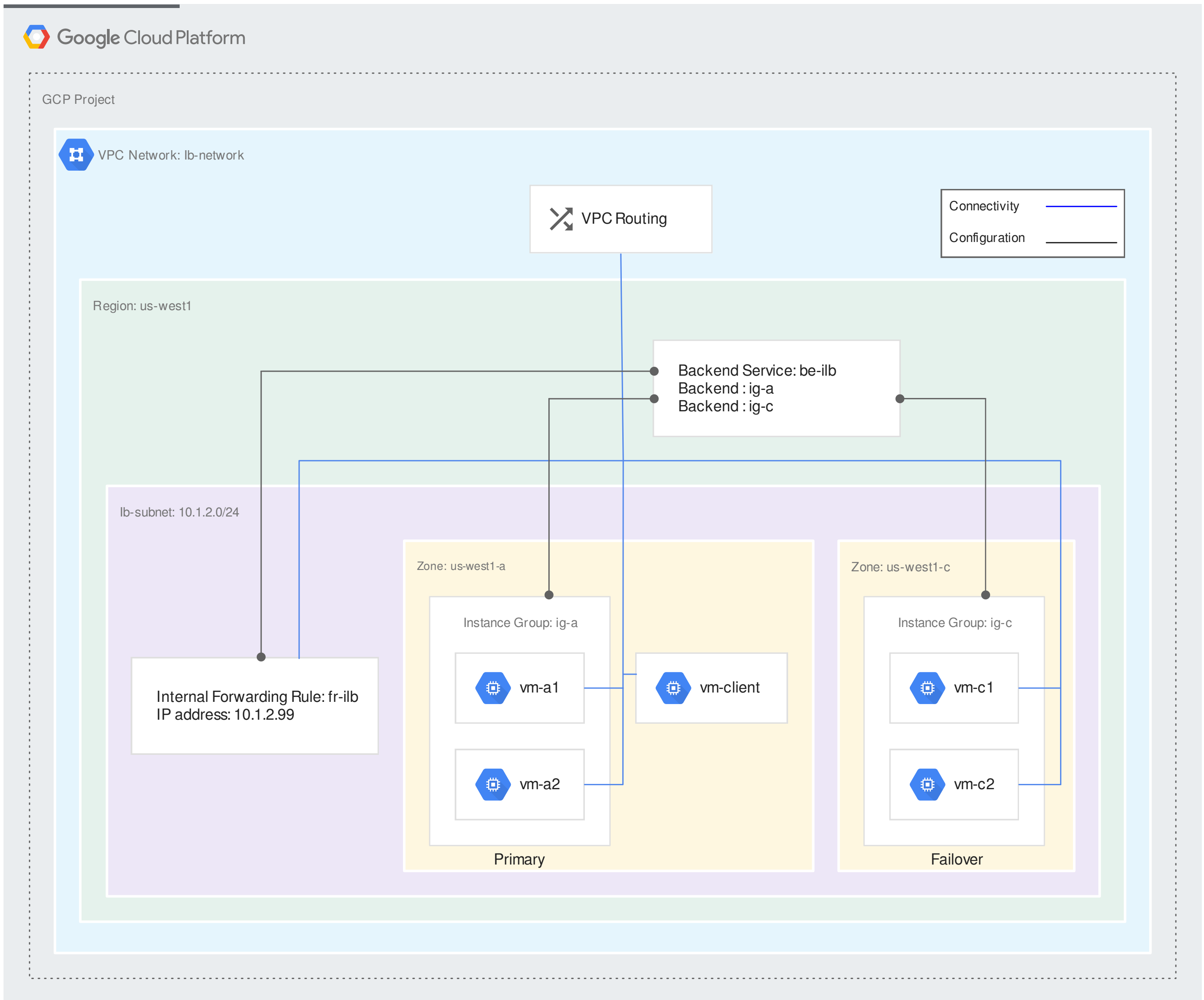
To follow this guide, you need to create instances and modify a network in a project. You should be either a project owner or editor (/iam/docs/understanding-roles#primitive_roles), or you should have all of the following Compute Engine IAM roles (/compute/docs/access/iam):

Task	Required Role
Create networks, subnets, and load balancer components	Network Admin (/compute/docs/access/iam#compute.networkAdmin)
Add and remove firewall rules	Security Admin (/compute/docs/access/iam#compute.securityAdmin)
Create instances	Instance Admin (/compute/docs/access/iam#compute.instanceAdmin)

This guide shows you how to configure and test an internal TCP/UDP load balancer that uses failover. The steps in this section describe how to configure the following:

1. A sample VPC network with custom subnets
2. Firewall rules that allow incoming connections to backend VMs
3. Backend VMs:
   - One primary backend in an unmanaged instance group in zone `us-west1-a`
   - One failover backend in an unmanaged instance group in zone `us-west1-c`
4. One client VM to test connections and observe failover behavior
5. The following internal TCP/UDP load balancer components:
   - A health check for the backend service
   - An internal backend service in the `us-west1` region to manage connection distribution among the backend VMs
   - An internal forwarding rule and internal IP address for the frontend of the load balancer

The architecture for this example looks like this:



(/load-balancing/images/ilb-failover.svg)

Simple Failover Example for Internal TCP/UDP Load Balancing (click to enlarge)

Unmanaged instance groups are used for both the primary and failover backends in this example. For more informat
pported instance groups (/load-balancing/docs/internal/failover-overview#supported_instance_groups).

This example uses the following VPC network, region, and subnet:

- Network: The network is a custom mode VPC network (/vpc/docs/vpc#subnet-ranges) named `lb-network`.
- Region: The region is `us-west1`.

- Subnet: The subnet, `lb-subnet`, uses the `10.1.2.0/24` IP range.

You can change the name of the network, the region, and the parameters for the subnet; however, subsequent steps use the network, region, and subnet parameters as outlined above.

To create the example network and subnet, follow these steps.

This example uses the following firewall rules:

- `fw-allow-lb-subnet`: An ingress rule, applicable to all targets in the VPC network, allowing traffic from sources in the `10.1.2.0/24` range. This rules allows incoming traffic from any source within the `lb-subnet` to the instances (VMs) being load balanced.
- `fw-allow-ssh`: An ingress rule applied to the instances being load balanced, allowing incoming SSH connectivity on TCP port 22 from any address. You can choose a more restrictive source IP range for this rule; for example, you can specify the IP ranges of the systems from which you plan to initiate SSH sessions. This example uses the target tag `allow-ssh` to identify the VMs to which the firewall rule applies.
- `fw-allow-health-check`: An ingress rule, applicable to the instances being load balanced, that allows traffic from the Google Cloud health checking systems (`130.211.0.0/22` and `35.191.0.0/16`). This example uses the target tag `allow-health-check` to identify the instances to which it should apply.

Without these firewall rules, the default deny ingress (/vpc/docs/firewalls#default_firewall_rules) rule blocks incoming traffic to the backend instances.

You must create a firewall rule to allow health checks from the IP ranges of Google Cloud probe systems. See probe I
s (/load-balancing/docs/health-check-concepts) for more information.

In this step, you'll create the backend VMs and unmanaged instance groups:

- The instance group `ig-a` in `us-west1-a` is a primary backend with two VMs:

    - `vm-a1`
    - `vm-a2`
- The instance group `ig-c` in `us-west1-c` is a failover backend with two VMs:
    - `vm-c1`
    - `vm-c2`

The primary and failover backends are placed in separate zones for instructional clarity and to handle failover in case one zone goes down.

Each primary and backup VM is configured to run an Apache web server on TCP ports 80 and 443. Each VM is assigned an internal IP address in the `lb-subnet` for client access and an ephemeral external (public) IP address for SSH access. For information about removing external IP addresses, see removing external IP addresses from backend VMs (/load-balancing/docs/internal/setting-up-internal#remove_external_ip).

By default, Apache is configured to bind to any IP address. Internal TCP/UDP load balancers deliver packets by preserving the destination IP.

Ensure that server software running on your primary and backup VMs is listening on the IP address of the load balancer's internal forwarding rule. If you configure multiple internal forwarding rules (/load-balancing/docs/internal/index#multiple_forwarding_rule), ensure that your software listens to the internal IP address associated with each one. The destination IP address of a packet delivered to a backend VM by an internal TCP/UDP load balancer is the internal IP address of the forwarding rule.

For instructional simplicity, all primary and backup VMs run Debian GNU/Linux 9.

This example creates a client VM (`vm-client`) in the same region as the load balancer. The client is used to demonstrate how failover works.

These steps configure all of the internal TCP/UDP load balancer components (/load-balancing/docs/internal/#components) starting with the health check and backend service, and then the frontend components:

- **Health check**: This example uses an HTTP health check that simply checks for an HTTP `200` (OK) response. For more information, see the health checks section of the Internal TCP/UDP Load Balancing overview (/load-balancing/docs/internal/#health-checking).
- **Backend service**: Because the example passes HTTP traffic through the internal load balancer, the configuration specifies TCP, not UDP. To illustrate failover, this backend service has a failover ratio of `0.75`.
- **Forwarding rule**: This example creates a single internal forwarding rule.
- **Internal IP address**: In this example, we specify an internal IP address, `10.1.2.99`, when we create the forwarding rule.

These tests show how to validate your load balancer configuration and learn about its expected behavior.

This procedure contacts the load balancer from the client VM. You'll use this procedure to complete the other tests.

1. Connect to the client VM instance.
2. Make a web request to the load balancer using `curl` to contact its IP address.

3. Note the text returned by the `curl` command. The name of the backend VM generating the response is displayed in that text; for example: `Page served from: vm-a1`

After you've configured the example load balancer, all four of the backend VMs should be healthy:

- the two primary VMs, `vm-a1` and `vm-a2`
- the two backup VMs, `vm-c1` and `vm-c2`

Follow the client test procedure (#client-curl). Repeat the second step a few times. The expected behavior is for traffic to be served by the two primary VMs, `vm-a1` and `vm-a2`, because both of them are healthy. You should see each primary VM serve a response approximately half of the time because no session affinity has been configured (/load-balancing/docs/internal/index#traffic_distribution) for this load balancer.

This test simulates the failure of `vm-a1` so you can observe failover behavior.

1. Connect to the `vm-a1` VM.

2. Stop the Apache web server. After ten seconds, Google Cloud considers this VM to be unhealthy. (The `hc-http-80` health check that you created in the setup uses the default check interval of five seconds and unhealthy threshold of two consecutive failed probes.)

3. Follow the client test procedure (#client-curl). Repeat the second step a few times. The expected behavior is for traffic to be served by the two backup VMs, `vm-c1` and `vm-c2`. Because only one primary VM, `vm-a2`, is healthy, the ratio of healthy primary VMs to total primary VMs is `0.5`. This number is less than the failover threshold of `0.75`, so Google Cloud reconfigured the load balancer's active pool to use the backup VMs. You should see each backup VM serve a response approximately half of the time as long as no session affinity has been configured (/load-balancing/docs/internal/index#traffic_distribution) for this load balancer.

This test simulates failback by restarting the Apache server on `vm-a1`.

1. Connect to the `vm-a1` VM.

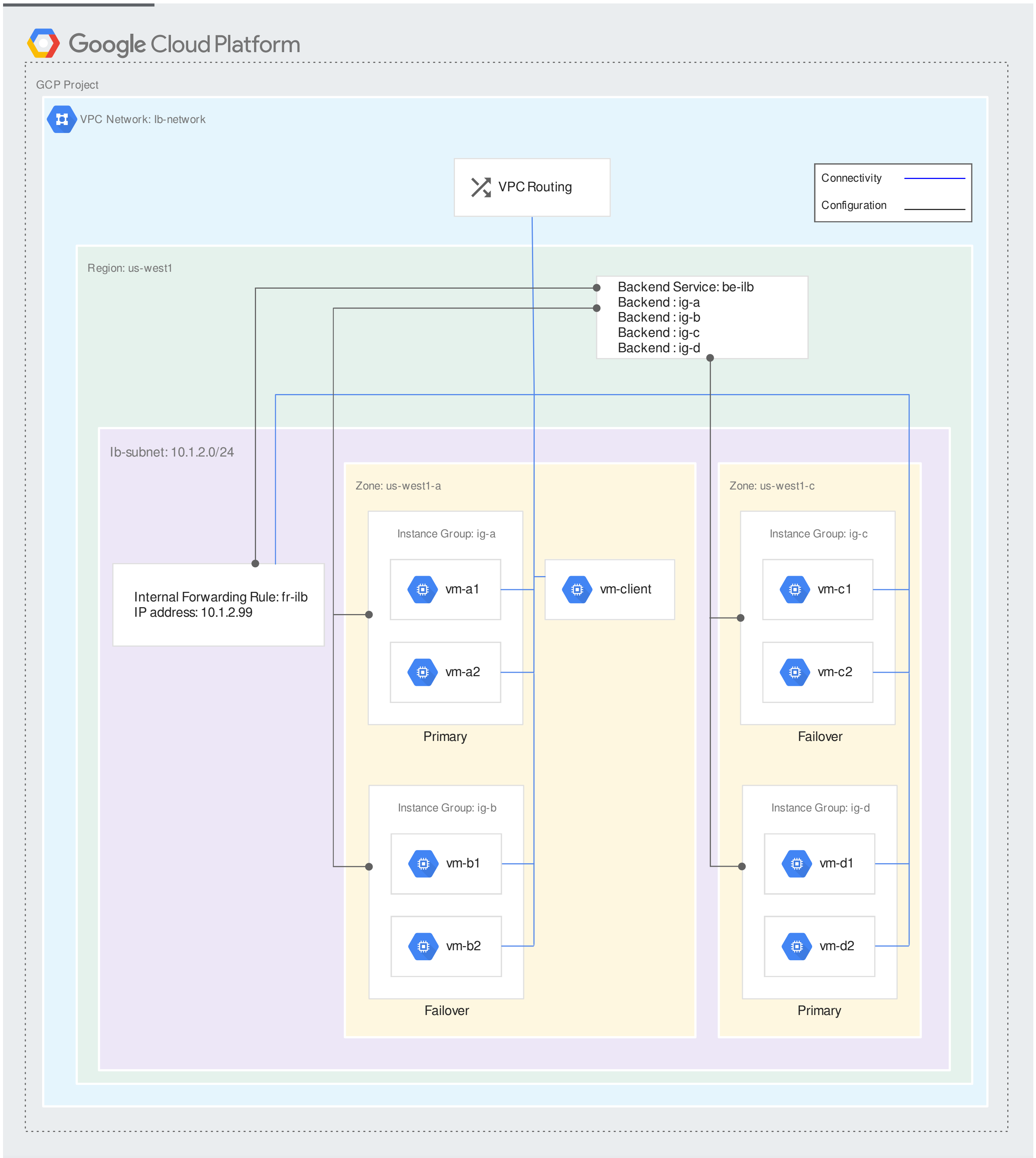
2. Start the Apache web server and wait 10 seconds.

3. Follow the client test procedure (#client-curl). Repeat the second step a few times. The expected behavior is for traffic to be served by the two primary VMs, `vm-a1` and `vm-a2`. With both primary VMs being healthy, the ratio of healthy primary VMs to total primary VMs is `1.0`, greater than the failover threshold of `0.75`, so Google Cloud configured the active pool to use the primary VMs again.

This section extends the example configuration by adding more primary and backup VMs to the load balancer. It does so by creating two more backend instance groups to demonstrate that you can distribute primary and backup VMs among multiple zones in the same region:

- A third instance group, `ig-d` in `us-west1-c`, serves as a primary backend with two VMs:
  - `vm-d1`
  - `vm-d2`
- A fourth instance group, `ig-b` in `us-west1-a`, serves as a failover backend with two VMs:
  - `vm-b1`
  - `vm-b2`

The modified architecture for this example looks like this:



(/load-balancing/images/ilb-failover-expanded.svg)

Multi-zone Internal TCP/UDP Load Balancing failover (click to enlarge)

Follow these steps to create the additional primary and backup VMs and their corresponding unmanaged instance groups.

You can use this procedure as a template for how to add an unmanaged instance group to an existing internal TCP/UDP load balancer's backend service as a primary backend. For the example configuration, this procedure shows you how to add instance group `ig-d` as a primary backend to the `be-ilb` load balancer.

You can use this procedure as a template for how to add an unmanaged instance group to an existing internal TCP/UDP load balancer's backend service as a failover backend. For the example configuration, this procedure shows you how to add instance group `ig-b` as a failover backend to the `be-ilb` load balancer.

You can use convert a primary backend to a failover backend, or vice versa, without having to remove the instance group from the internal TCP/UDP load balancer's backend service.

This section describes how to manage a failover policy (/load-balancing/docs/internal/failover-overview#failover_policy) for an internal TCP/UDP load balancer's backend service. A failover policy consists of the:

- Failover ratio
- Dropping traffic when all backend VMs are unhealthy
- Connection draining on failover

For more information on the parameters of a failover policy, see:

- Setting a failover ratio (/load-balancing/docs/internal/failover-overview#failover_ratio)
- Dropping traffic when there is no healthy VM (/load-balancing/docs/internal/failover-overview#drop_traffic)
- Draining connections on failover and failback (/load-balancing/docs/internal/failover-overview#connection_draining)

The following instructions describe how to define the failover policy for an existing internal TCP/UDP load balancer.

The following instructions describe how to view the existing failover policy for an internal TCP/UDP load balancer.

- See Internal TCP/UDP Load Balancing Concepts (/load-balancing/docs/internal/index) for important fundamentals.
- See Failover concepts for Internal TCP/UDP Load Balancing (/load-balancing/docs/internal/failover-overview) for important information about failover.
- See Setting Up Internal TCP/UDP Load Balancing (/load-balancing/docs/internal/setting-up-internal) for an example internal TCP/UDP load balancer configuration.
- See Internal TCP/UDP Load Balancing Logging and Monitoring (/load-balancing/docs/internal/internal-logging-monitoring) for information on configuring Stackdriver logging and monitoring for Internal TCP/UDP Load Balancing.
- See Internal TCP/UDP Load Balancing and Connected Networks (/load-balancing/docs/internal/internal-lb-and-other-networks) for information about accessing internal TCP/UDP load balancers from peer networks connected to your VPC network.
- See Troubleshooting Internal TCP/UDP Load Balancing (/load-balancing/docs/internal/troubleshooting-ilb) for information on how to troubleshoot issues

with your internal TCP/UDP load balancer.