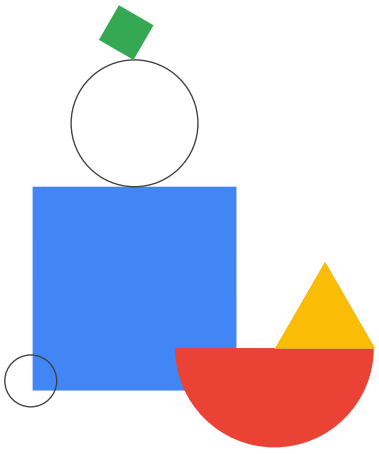
Google Cloud

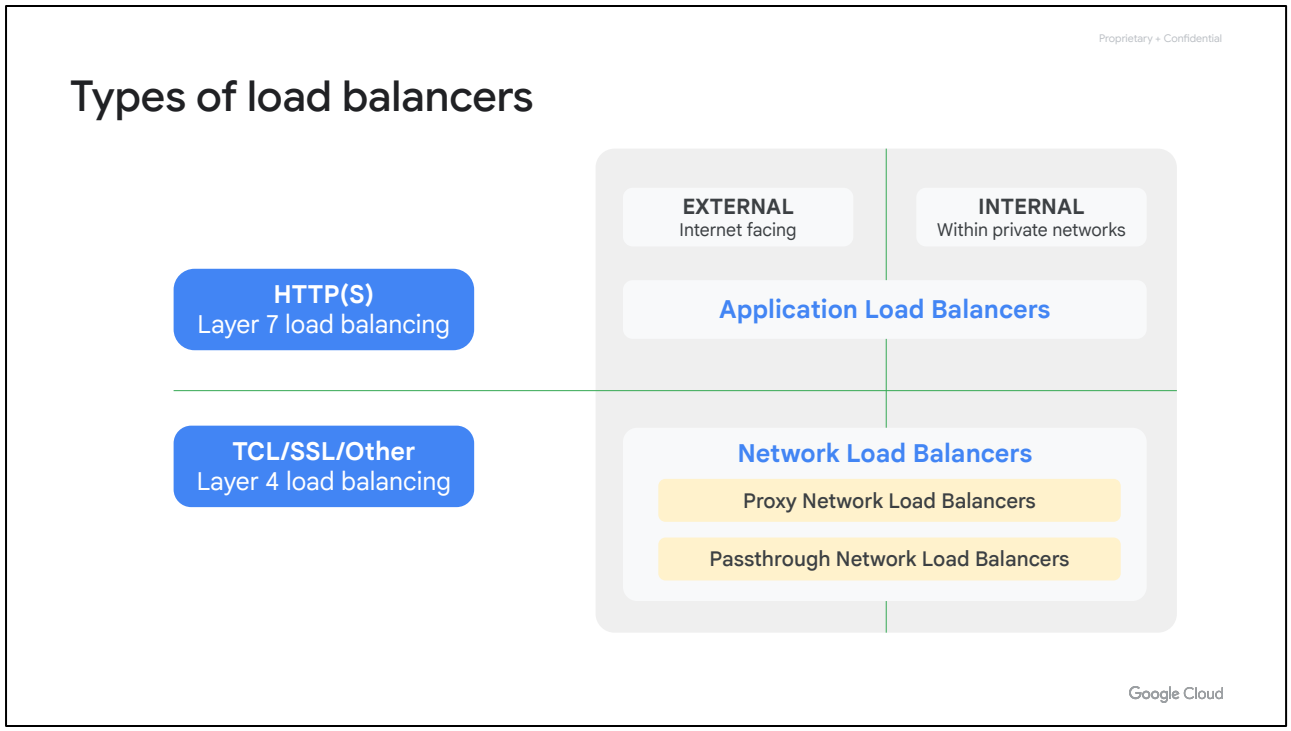
Load Balancing and Autoscaling



In this module we focus on load balancing and autoscaling.

Cloud Load Balancing gives you the ability to distribute load-balanced compute resources in single or multiple regions to meet your high availability requirements, to put your resources behind a single anycast IP address, and to scale your resources up or down with intelligent autoscaling.

Using Cloud Load Balancing, you can serve content as close as possible to your users on a system that can respond to over 1 million queries per second. Cloud Load Balancing is a fully distributed, software-defined, managed service. It isn't instance- or device-based, so you don't need to manage a physical load balancing infrastructure.



Application Load Balancers and **Network Load Balancers** are two primary types of load balancers offered by Google Cloud, each designed for specific use cases.

Application Load Balancers operate at the application layer (Layer 7) of the OSI model. They are ideal for applications that require load balancing based on HTTP(S) headers, cookies, or URL paths. **Application Load Balancers** provide features like SSL/TLS termination, session affinity, and content-based routing.

Network Load Balancers operate at the network layer (Layer 4) of the OSI model. They are suitable for load balancing based on IP addresses and ports. **Network Load Balancers** are often used for TCP and UDP traffic, as well as for scenarios where low latency and high throughput are critical. They support features like TCP/UDP load balancing and health checks.

For more information on load balancers, please refer to Cloud Load Balancing Overview.

Agenda

01	Managed Instance Groups
02	Application Load Balancers Lab: Configure an Application Load Balancer with Autoscaling
03	Cloud CDN
04	Network Load Balancing
05	Internal Load Balancing Lab: Configure an Internal Network Load Balancer
06	Choosing a Load Balancer

In this module, we cover the different types of load balancers that are available in Google Cloud. We also go over managed instance groups and their autoscaling configurations, which can be used by these load balancing configurations.

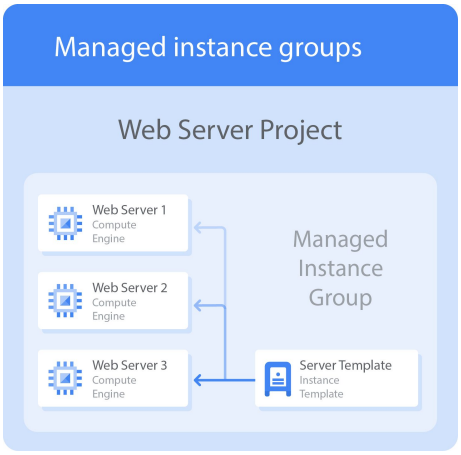
You explore many of the covered features and services throughout the two labs of this module. The module wraps things up by helping you determine which Google Cloud load balancer best meets your needs.

Managed Instance Groups

Let’s start by talking about managed instance groups.

Managed instance groups

- Deploy identical instances based on instance template
- Instance group can be resized
- Manager ensures all instances are RUNNING
- Typically used with autoscaler
- Can be single zone or regional

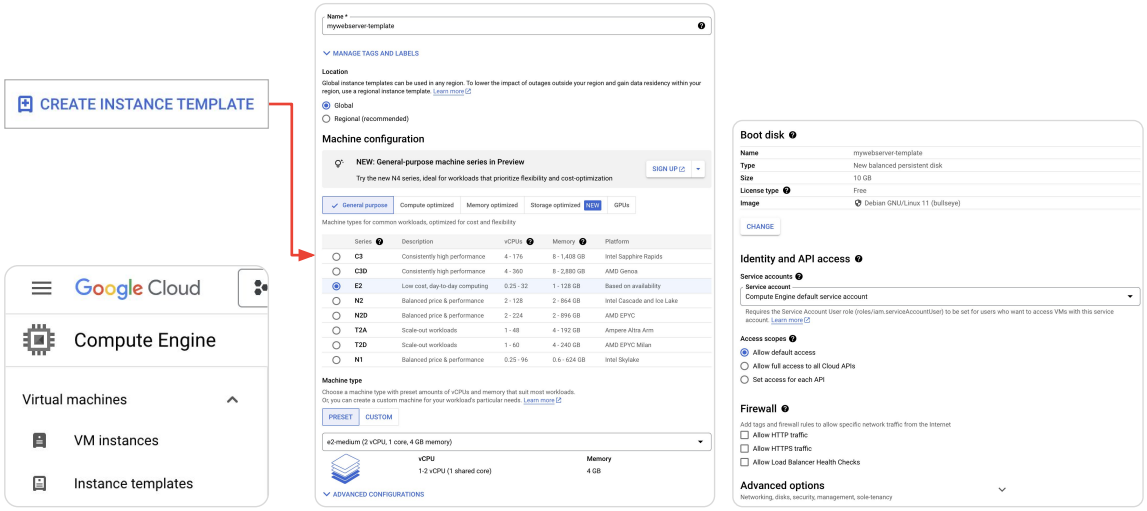


A managed instance group is a collection of identical VM instances that you control as a single entity, using an instance template. You can easily update all the instances in the group by specifying a new template in a rolling update. Also, when your applications require additional compute resources, managed instance groups can automatically scale the number of instances in the group.

Managed instance groups can work with load balancing services to distribute network traffic to all of the instances in the group. If an instance in the group stops, crashes, or is deleted by an action other than the instance group's commands, the managed instance group automatically recreates the instance so it can resume its processing tasks. The recreated instance uses the same name and the same instance template as the previous instance. Managed instance groups can automatically identify and recreate unhealthy instances in a group to ensure that all the instances are running optimally.

Regional managed instance groups are generally recommended over zonal managed instance groups because they allow you to spread the application load across multiple zones instead of confining your application to a single zone or you having to manage multiple instance groups across different zones. This replication protects against zonal failures and unforeseen scenarios where an entire group of instances in a single zone malfunctions. If that happens, your application can continue serving traffic from instances running in another zone of the same region.

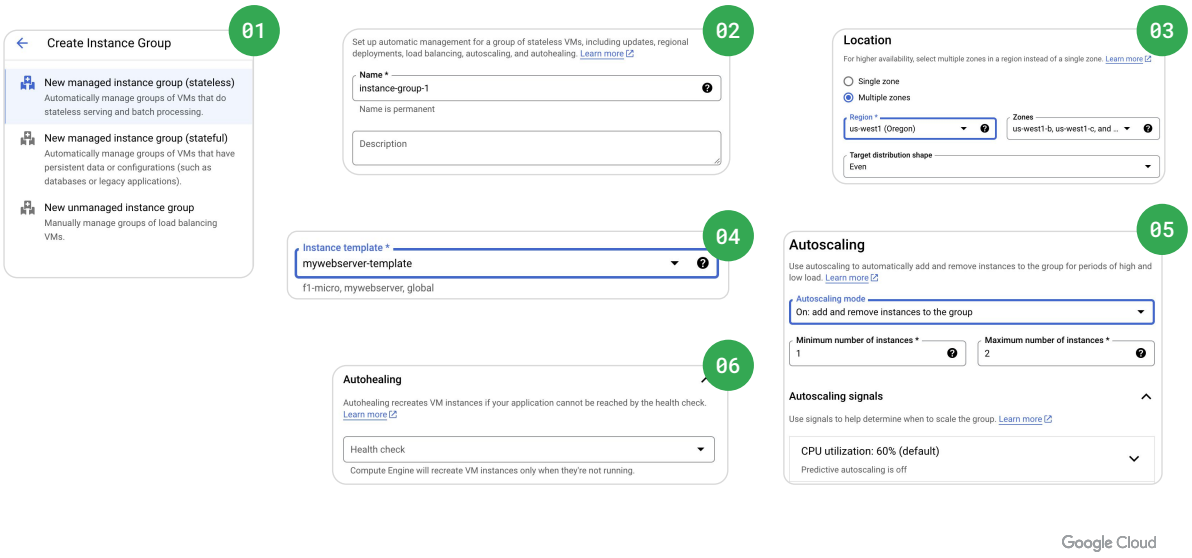
Create an instance template



In order to create a managed instance group, you first need to create an instance template. Next, you're going to create a managed instance group of N specified instances. The instance group manager then automatically populates the instance group based on the instance template.

You can easily create instance templates using the Google Cloud console. The instance template dialog looks and works exactly like creating an instance, except that the choices are recorded so that they can be repeated.

Create a managed instance group



When you create an instance group, you define the specific rules for the instance group.

- First, decide which type of managed instance group you want to create. You can use managed instance groups for stateless serving or batch workloads, such as a website frontend or image processing from a queue, or for stateful applications, such as databases or legacy applications.
- Second provide a name for the instance group.
- Third, decide whether the instance group is going to be single or multi-zoned, and where those locations will be. You can optionally provide port name mapping details..
- Fourth, select the instance template that you want to use.
- Fifth, decide whether you want to autoscale, and under what circumstances.
- Finally, consider creating a health check to determine which instances are healthy and should receive traffic.

Essentially, you're still creating virtual machines, but you're applying more rules to that instance group.

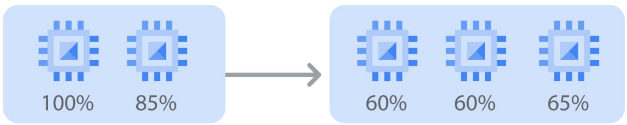
Managed instance groups offer autoscaling capabilities

Dynamically add/remove instances:

- Increases in load
- Decreases in load

Autoscaling policy:

- CPU utilization
- Load balancing capacity
- Monitoring metrics
- Queue-based workload
- Schedule-based



Target CPU utilization = 75%

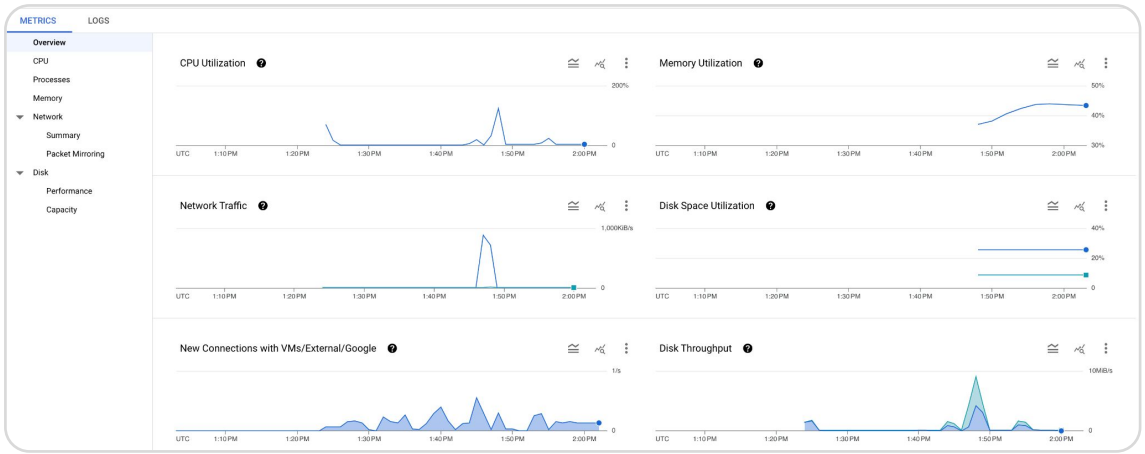
Let me provide more details on the autoscaling and health checks of a managed instance group.

As I mentioned earlier, managed instance groups offer autoscaling capabilities that allow you to automatically add or remove instances from a managed instance group based on increases or decreases in load. Autoscaling helps your applications gracefully handle increases in traffic and reduces cost when the need for resources is lower.

You just define the autoscaling policy, and the autoscaler performs automatic scaling based on the measured load. Applicable autoscaling policies include scaling based on CPU utilization, load balancing capacity, or monitoring metrics, or by a queue-based workload like Pub/Sub or schedule such as start-time, duration and recurrence.

For example, let's assume you have 2 instances that are at 100% and 85% CPU utilization as shown on this slide. If your target CPU utilization is 75%, the autoscaler will add another instance to spread out the CPU load and stay below the 75% target CPU utilization. Similarly, if the overall load is much lower than the target, the autoscaler will remove instances as long as that keeps the overall utilization below the target. Now, you might ask yourself how do I monitor the utilization of my instance group.

VM graph helps set CPU utilization

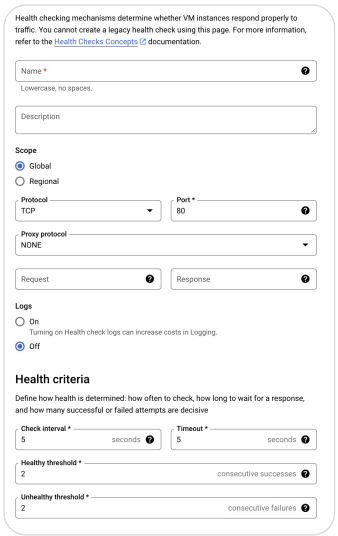


When you click on an instance group (or even an individual VM), you can choose to view different metrics. By default you'll see the CPU utilization over the past hour, but you can change the time frame and visualize other metrics like disk and network usage. These graphs are very useful for monitoring your instances' utilization and for determining how best to configure your Autoscaling policy to meet changing demand.

If you monitor the utilization of your VM instances in Cloud Monitoring, you can even set up alerts through several notification channels.

A link to more information on autoscaling can be found in the Course Resources for this module.

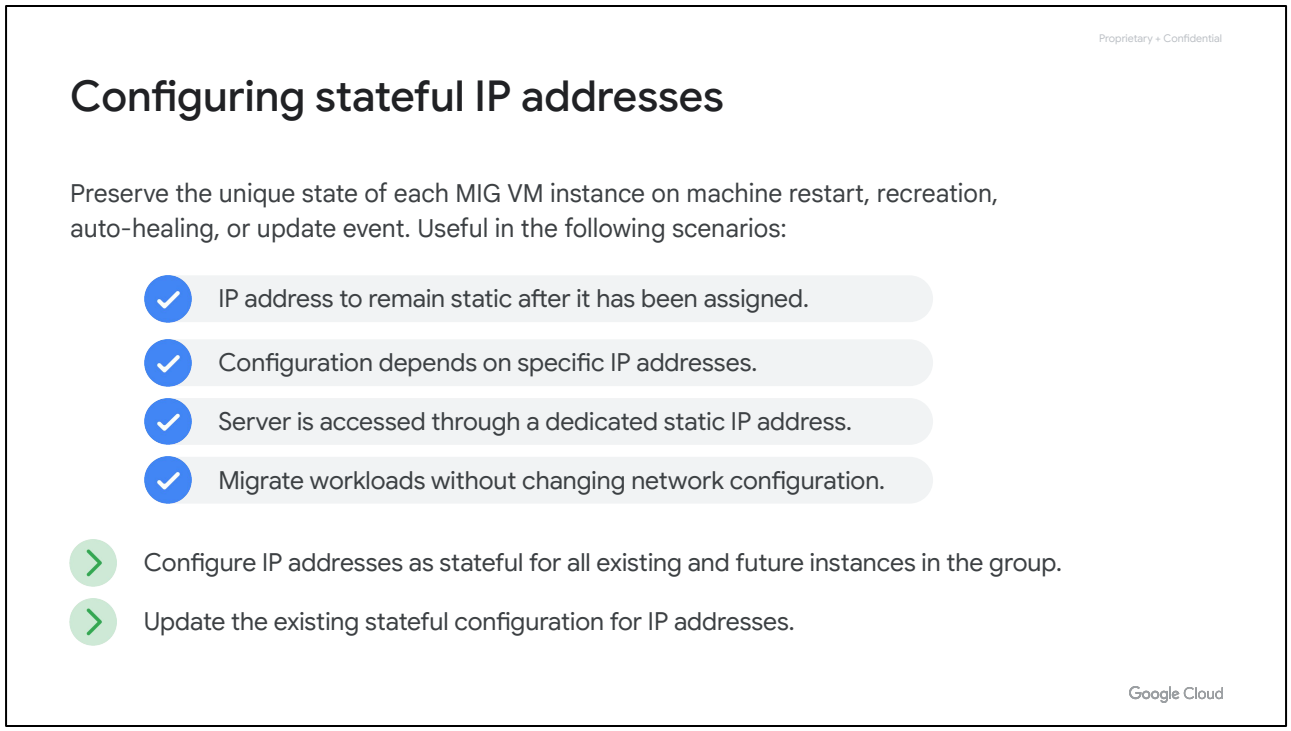
Create a health check



Elapsed time (seconds)	Event duration (seconds)	
1	1	wait
2	2	wait
3	3	wait
4	4	wait
5	5	wait
6	1	**health check #1 starts**
7	2	wait
8	3	wait
9		**health check #1 fails**
10		wait
11	1	**health check #2 starts**
12	2	wait
13	3	wait
14		**health check #2 fails**
15		**Unhealthy threshold reached**

Another important configuration for a managed instance group and load balancer is a health check. A health check is very similar to an uptime check in Cloud Monitoring. You just define a protocol, port, and health criteria, as shown in this screenshot. Based on this configuration, Google Cloud computes a health state for each instance.

The health criteria define how often to check whether an instance is healthy (that's the check interval); how long to wait for a response (that's the timeout); how many successful attempts are decisive (that's the healthy threshold); and how many failed attempts are decisive (that's the unhealthy threshold). In the example on this slide, the health check would have to fail twice over a total of 15 seconds before an instance is considered unhealthy.



Configuring stateful IP addresses in a Managed Instance Group (MIG) ensures that applications continue to function seamlessly during autohealing, update, and recreation events. Both internal and external IPv4 addresses can be preserved. You can configure IP addresses to be assigned automatically or assign specific IP addresses to each VM instance in a MIG.

Preserving an instance's IP addresses is useful in many different scenarios:

- Your application requires an IP address to remain static after it has been assigned.
- Your application's configuration depends on specific IP addresses.
- Users, including other applications, access a server through a dedicated static IP address.
- You need to migrate existing workloads without changing network configuration.

You can do the following operations by configuring stateful policy on an existing MIG:

- Configure IP addresses as stateful for all existing and future instances in the group. This will promote the corresponding ephemeral IP addresses of all existing instances to static IP addresses.
- Update the existing stateful configuration for IP addresses.

Application Load Balancing

Now, let's talk about Application Load Balancing, which acts at Layer 7 of the OSI model. This is the application layer, which deals with the actual content of each message, allowing for routing decisions based on the URL.

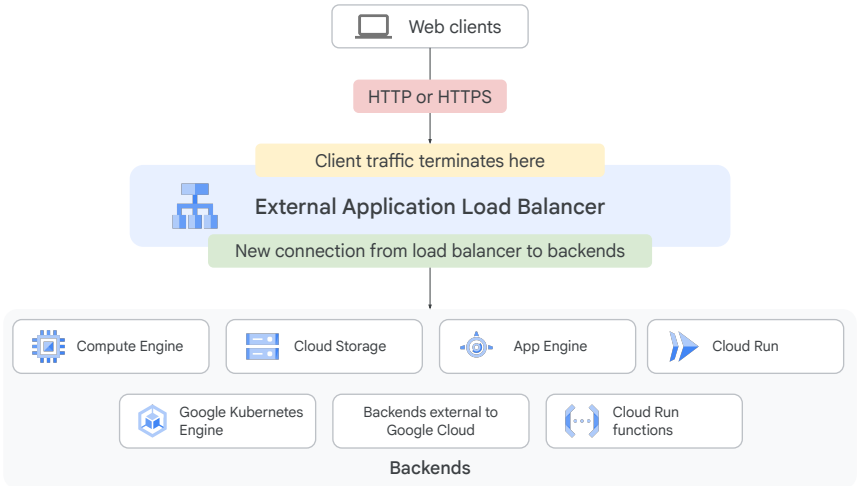
Overview of an Application Load Balancer

Deployment mode	Network service tier	Load balancing scheme	IP address	Frontend ports
Global external	Premium Tier	EXTERNAL_MANAGED	IPv4 IPv6	Can reference exactly one port from 1-65535
Regional external	Premium or Standard Tier	EXTERNAL_MANAGED	IPv4	
Classic	Global in Premium Tier Regional in Standard Tier	EXTERNAL	IPv4 IPv6 (requires Premium Tier)	

The Application Load Balancer distributes HTTP and HTTPS traffic to backends hosted on a variety of Google Cloud platforms—such as Compute Engine, Google Kubernetes Engine (GKE), Cloud Storage, and Cloud Run—as well as external backends connected over the internet or by using hybrid connectivity.

Application Load Balancers are available in the following deployment modes, external and internal. You will learn about internal Application Load Balancers later in this module.

Architecture of an external Application Load Balancer



External Application Load Balancers are implemented using Google Front Ends (GFEs) or managed proxies. Global external Application Load Balancers and classic Application Load Balancers use GFEs that are distributed globally, operating together by using Google's global network and control plane. GFEs offer multi-region load balancing in the Premium tier, directing traffic to the closest healthy backend that has capacity and terminating HTTP(S) traffic as close as possible to your users. Global external Application Load Balancers and regional external Application Load Balancers use the open source Envoy proxy software to enable advanced traffic management capabilities.

These load balancers can be deployed in one of the following modes: global, regional, or classic.

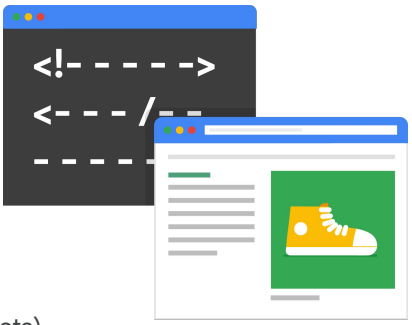
Let me walk through the architecture of an Application Load Balancer, by using this diagram:

- An external forwarding rule specifies an external IP address, port, and target HTTP(S) proxy. Clients use the IP address and port to connect to the load balancer.
- A target HTTP(S) proxy receives a request from the client. The HTTP(S) proxy evaluates the request by using the URL map to make traffic routing decisions.

- The proxy can also authenticate communications by using SSL certificates.
- A backend service distributes requests to healthy backends. The global external Application Load Balancers also support backend buckets. One or more backends must be connected to the backend service or backend bucket.

Backend services

- Health check
- Session affinity (optional)
- Time out setting (30-sec default)
- One or more backends
  - An instance group (managed or unmanaged)
  - A balancing mode (CPU utilization or RPS)
  - A capacity scaler (ceiling percentage of CPU/Rate targets)



The backend services contain a health check, session affinity, a timeout setting, and one or more backends.

A health check polls instances attached to the backend service at configured intervals. Instances that pass the health check are allowed to receive new requests. Unhealthy instances are not sent requests until they are healthy again.

Normally, Application Load Balancing uses a round-robin algorithm to distribute requests among available instances.

This can be overridden with session affinity. Session affinity attempts to send all requests from the same client to the same virtual machine instance.

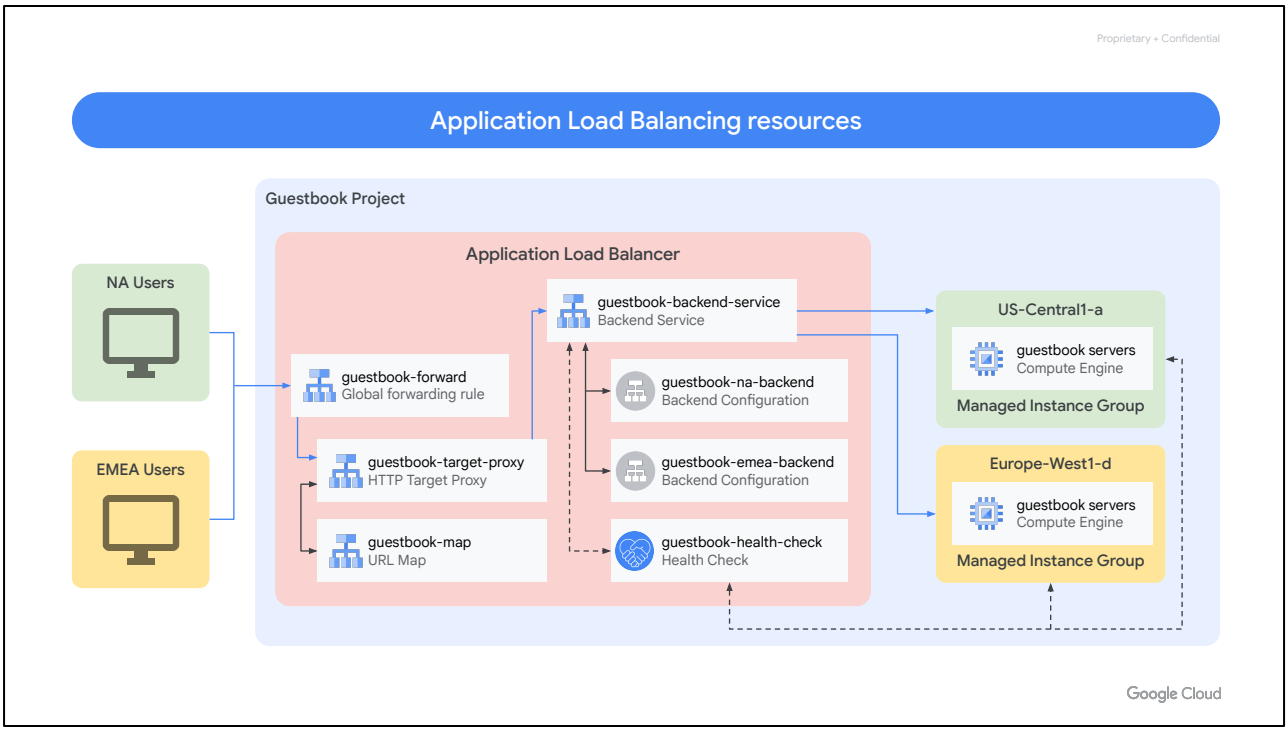
Backend services also have a timeout setting, which is set to 30 seconds by default. This is the amount of time the backend service will wait on the backend before considering the request a failure. This is a fixed timeout, not an idle timeout. If you require longer-lived connections, set this value appropriately.

The backends themselves contain an instance group, a balancing mode, and a capacity scaler.

- An instance group contains virtual machine instances. The instance group may be a managed instance group with or without autoscaling or an unmanaged instance group.

- A balancing mode tells the load balancing system how to determine when the backend is at full usage. If all the backends for the backend service in a region are at full usage, new requests are automatically routed to the nearest region that can still handle requests. The balancing mode can be based on CPU utilization or requests per second (RPS).
- A capacity setting is an additional control that interacts with the balancing mode setting. For example, if you normally want your instances to operate at a maximum of 80% CPU utilization, you would set your balancing mode to 80% CPU utilization and your capacity to 100%. If you want to cut instance utilization in half, you could leave the balancing mode at 80% CPU utilization and set capacity to 50%.

Now, any changes to your backend services are not instantaneous. So, don't be surprised if it takes several minutes for your changes to propagate throughout the network.



Let me walk through an Application Load Balancer in action. The project on this slide has a single global IP address, but users enter the Google Cloud network from two different locations: one in North America and one in EMEA.

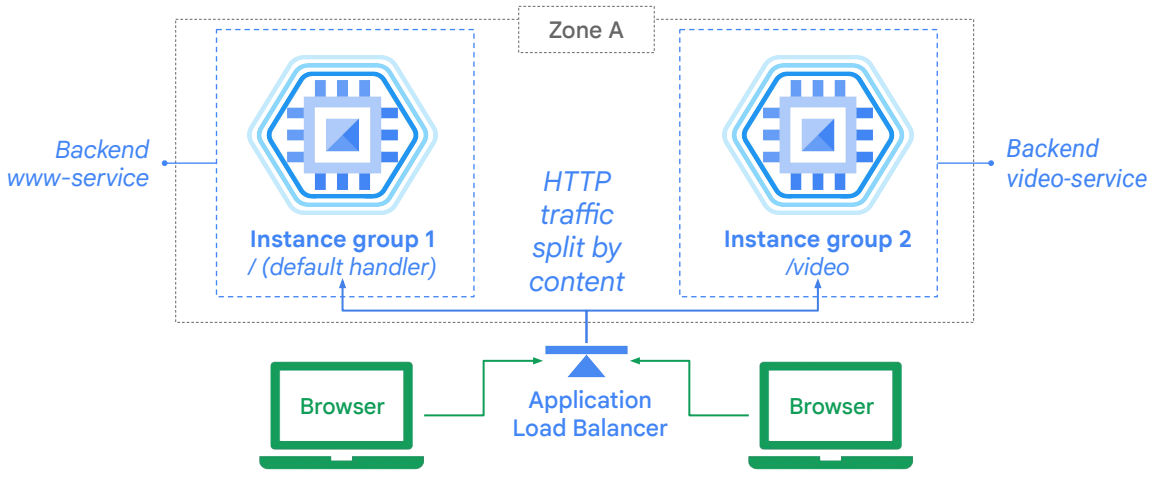
First, the global forwarding rule directs incoming requests to the target HTTP proxy. The proxy checks the URL map to determine the appropriate backend service for the request. In this case, we are serving a guestbook application with only one backend service.

The backend service has two backends: one in us-central1-a and one in europe-west1-d. Each of those backends consists of a managed instance group. Now, when a user request comes in, the load balancing service determines the approximate origin of the request from the source IP address. The load balancing service also knows the locations of the instances owned by the backend service, their overall capacity, and their overall current usage. Therefore, if the instances closest to the user have available capacity, the request is forwarded to that closest set of instances.

In our example, traffic from the user in North America would be forwarded to the managed instance group in us-central1-a, and traffic from the user in EMEA would be forwarded to the managed instance group in europe-west1-d. If there are several users in each region, the incoming requests to the given region are distributed evenly across all available backend services and instances in that region.

If there are no healthy instances with available capacity in a given region, the load balancer instead sends the request to the next closest region with available capacity. Therefore, traffic from the EMEA user could be forwarded to the us-central1-a backend if the europe-west1-d backend does not have capacity or has no healthy instances as determined by the health checker. This is referred to as cross-region load balancing.

Example: Content-based load balancing

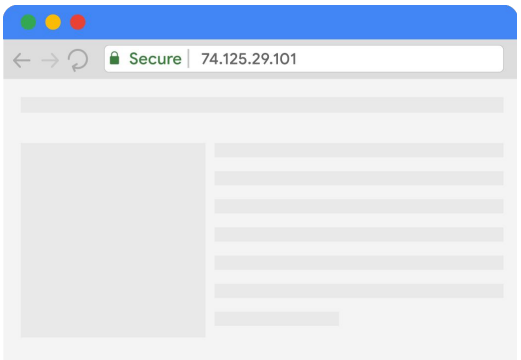


Another example of an Application Load Balancer is a content-based load balancer. In this case, there are two separate backend services that handle either web or video traffic.

The traffic is split by the load balancer based on the URL header as specified in the URL map. If a user is navigating to /video, the traffic is sent to the backend video-service, and if a user is navigating anywhere else, the traffic is sent to the web-service backend. All of that is achieved with a single global IP address.

Application Load Balancing - Target HTTPS proxy

- Target HTTP(S) proxy
- One signed SSL certificate installed (minimum)
- Client SSL session terminates at the load balancer
- Support the QUIC transport layer protocol



An Application Load Balancer that configured to use HTTP(S) has the same structure basic structure when configured to balance using HTTP, but differs in the following ways:

- An Application Load Balancer uses a target HTTPS proxy instead of a target HTTP proxy.
- An Application Load Balancer requires at least one signed SSL certificate installed on the target HTTPS proxy for the load balancer.
- The client SSL session terminates at the load balancer.
- Application Load Balancer supports the QUIC transport layer protocol.

QUIC is a transport layer protocol that allows faster client connection initiation, eliminates head-of-line blocking in multiplexed streams, and supports connection migration when a client's IP address changes. For more information on the QUIC protocol, refer to the documentation - https://www.chromium.org/quic.

SSL certificates

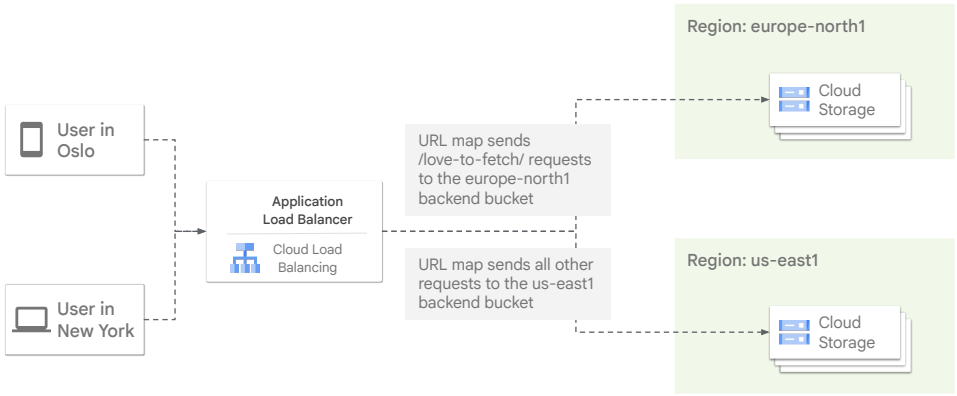
- Required for Application Load Balancing
- Up to 15 SSL certificates (per target proxy)
- Create an SSL certificate resource

To use HTTPS, you must create at least one SSL certificate that can be used by the target proxy for the load balancer.

You can configure the target proxy with up to fifteen SSL certificates.

For each SSL certificate, you first create an SSL certificate resource, which contains the SSL certificate information. SSL certificate resources are used only with load balancing proxies such as a target HTTPS proxy or target SSL proxy, which we will discuss later in this module.

Backend buckets



Backend buckets allow you to use Cloud Storage buckets with Application Load Balancing.

An external Application Load Balancer uses a URL map to direct traffic from specified URLs to either a backend service or a backend bucket.

One common use case is:

- Send requests for dynamic content, such as data, to a backend service.
- Send requests for static content, such as images, to a backend bucket.

In this diagram, the load balancer sends traffic with a path of /love-to-fetch/ to a Cloud Storage bucket in the europe-north region. All other requests go to a Cloud Storage bucket in the us-east region. After you configure a load balancer with the backend buckets, requests to URL paths that begin with /love-to-fetch are sent to the europe-north Cloud Storage bucket, and all other requests are sent to the us-east Cloud Storage bucket.

Network endpoint groups (NEG)

A network endpoint group (NEG) is a configuration object that specifies a group of backend endpoints or services.

There are four types of NEGs:

- Zonal
- Internet
- Serverless
- Hybrid connectivity

A network endpoint group (NEG) is a configuration object that specifies a group of backend endpoints or services. A common use case for this configuration is deploying services in containers. You can also distribute traffic in a granular fashion to applications running on your backend instances.

You can use NEGs as backends for some load balancers and with Traffic Director.

Zonal and internet NEGs define how endpoints should be reached, whether they are reachable, and where they are located. Unlike these NEG types, serverless NEGs don't contain endpoints.

A zonal NEG contains one or more endpoints that can be Compute Engine VMs or services running on the VMs. Each endpoint is specified either by an IP address or an IP:port combination.

An internet NEG contains a single endpoint that is hosted outside of Google Cloud. This endpoint is specified by hostname FQDN:port or IP:port.

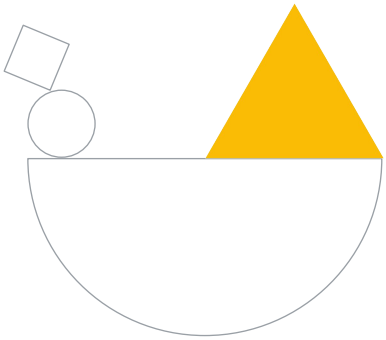
A hybrid connectivity NEG points to Traffic Director services running outside of Google Cloud.

A serverless NEG points to Cloud Run, App Engine, Cloud Run functions services residing in the same region as the NEG.

For more information on using NEG's, please refer to the Network endpoint groups overview page.

Lab Intro

Configure an Application Load Balancer with Autoscaling



Let’s apply what we just covered.

Lab objectives

01 Create HTTP and health check firewall rules

02 Create a custom image for a web server

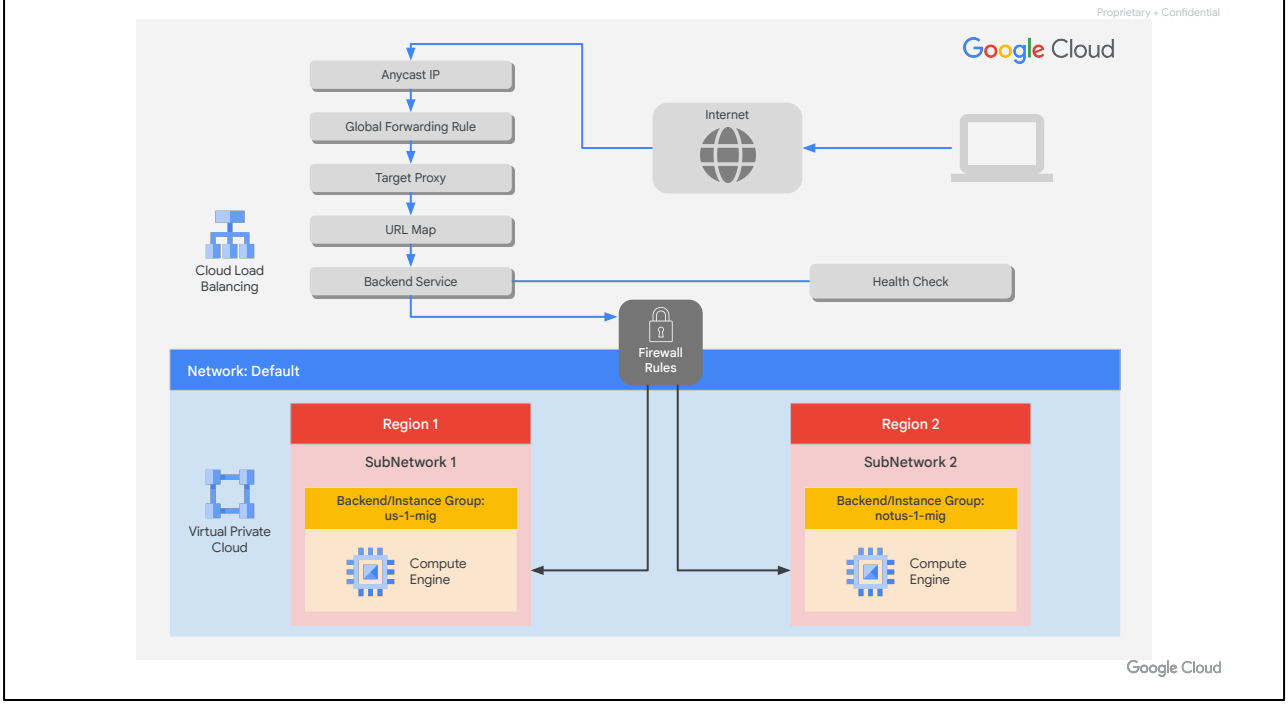
03 Create an instance template based on the custom image

04 Create two managed instance groups

05 Configure an Application Load Balancer with IPv4 and IPv6

06 Stress test an Application Load Balancer

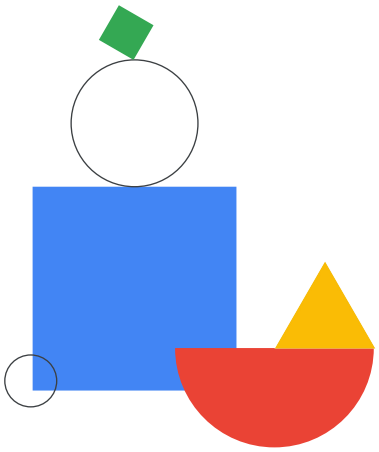
In this lab, you will configure an Application Load Balancer with autoscaling.



Specifically, you create two managed instance groups that serve as backends in two different regions. Then, you create and stress test a load balancer to demonstrate global load balancing and autoscaling.

Lab Review

Configure an Application Load Balancer with Autoscaling

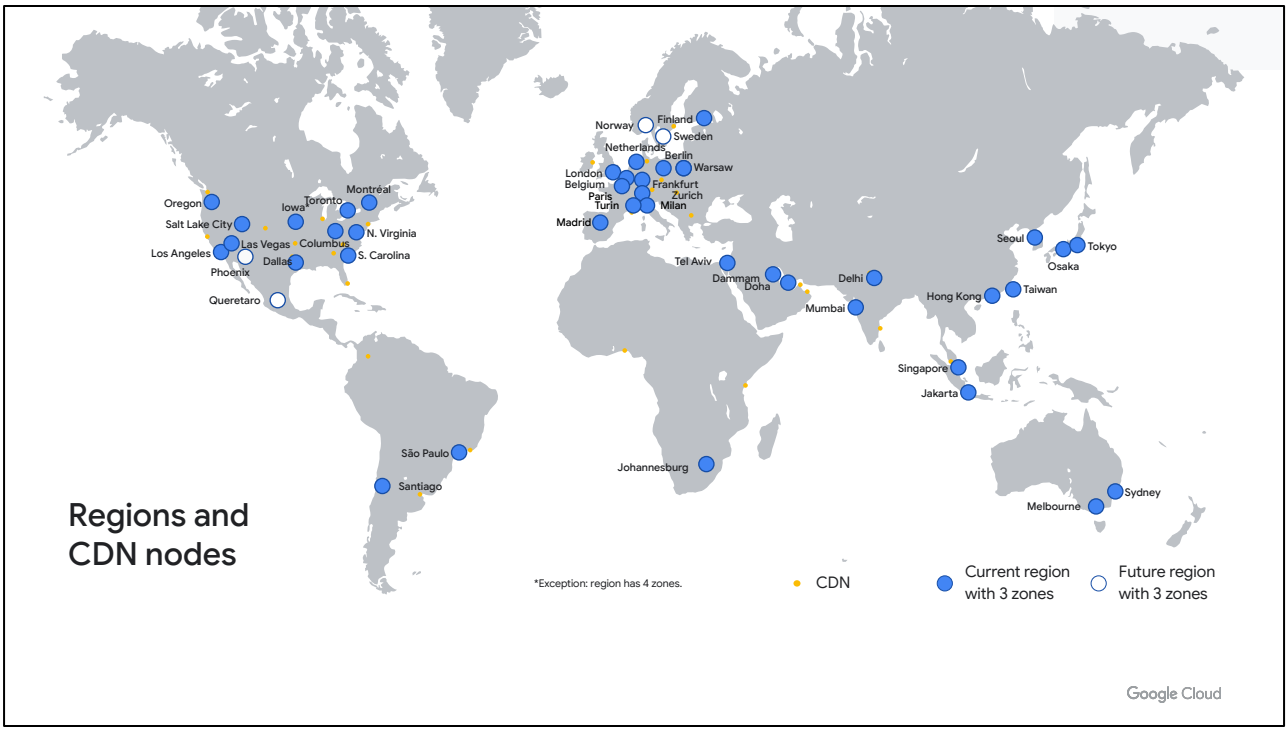


In this lab, you configured an Application Load Balancer with backends in us-central1 and europe-west1. Then you stress-tested the load balancer with a VM to demonstrate global load balancing and autoscaling.

You can stay for a lab walkthrough, but remember that Google Cloud's user interface can change, so your environment might look slightly different.

Cloud CDN

Cloud CDN, or Content Delivery Network, uses Google's globally distributed edge points of presence to cache HTTP(S) load-balanced content close to your users.



Specifically, content can be cached at CDN nodes as shown on this map.

There are over 90 of these cache sites spread across metropolitan areas in Asia Pacific, Americas, and EMEA.

For an up-to-date list, please refer to the Cloud CDN documentation link in the course resources.

Now, why should you consider using Cloud CDN?

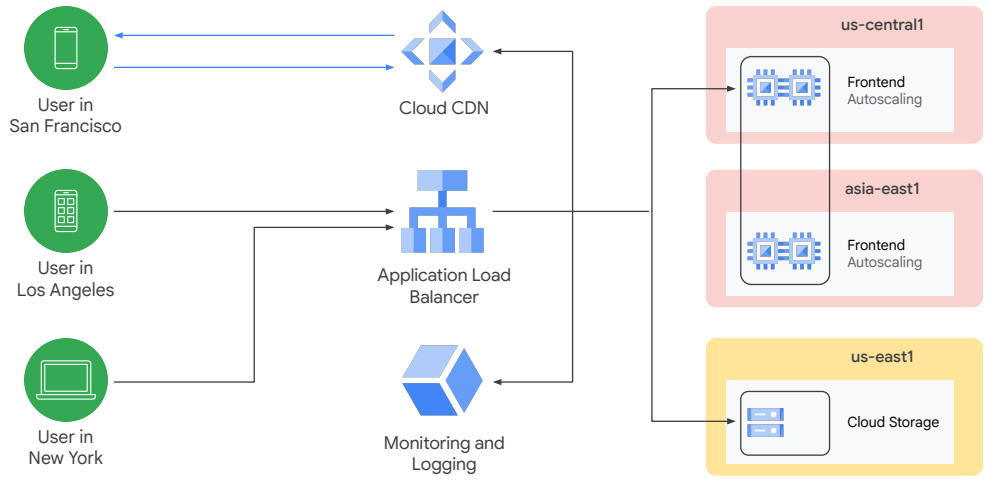
Well, Cloud CDN caches content at the edges of Google's network providing faster delivery of content to your users while reducing serving costs.

You can enable Cloud CDN with a simple checkbox when setting up the backend service of your Application Load Balancer.

So it’s easy to enable and benefits you and your users but how does Cloud CDN do all of this?

[Additional reading: For Cloud CDN performance measured by Cedexis, please refer to these reports: https://itm.cloud.com/google-reports/]

Caching content with Cloud CDN



Let's walk through the Cloud CDN response flow with this diagram.

In this example, the Application Load Balancer has two types of backends. There are managed VM instance groups in the us-central1 and asia-east1 regions, and there is a Cloud Storage bucket in us-east1. A URL map will decide which backend to send the content to: the Cloud Storage bucket could be used to serve static content and the instance groups could handle PHP traffic.

Now, when a user in San Francisco is the first to access a piece of content, the cache site in San Francisco sees that it can't fulfill the request. This is called a cache miss. The cache might attempt to get the content from a nearby cache, for example if a user in Los Angeles has already accessed the content. Otherwise, the request is forwarded to the Application Load Balancer, which in turn forwards the request to one of your backends.

Depending on what content is being served, the request will be forwarded to the us-central1 instance group or the us-east1 storage bucket.

If the content from the backend is cacheable, the cache site in San Francisco can store it for future requests. In other words, if another user requests the same content in San Francisco, the cache site might now be able to serve that content. This shortens the round trip time and saves the origin server from having to process the request. This is called a cache hit.

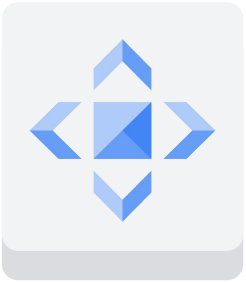
For more information on what content can be cached, please refer to the documentation link in the course resources.

Now, each Cloud CDN request is automatically logged within Google Cloud. These logs will indicate a “Cache Hit” or “Cache Miss” status for each HTTP request of the load balancer. You will explore such logs in the next lab.

But how do you know how Cloud CDN will cache your content? How do you control this? This is where cache modes are useful.

Cloud CDN cache modes

- Cache modes control the factors that determine whether or not Cloud CDN caches your content.
- Cloud CDN offers three cache modes:
  - USE_ORIGIN_HEADERS
  - CACHE_ALL_STATIC
  - FORCE_CACHE_ALL



Using cache modes, you can control the factors that determine whether or not Cloud CDN caches your content by using cache modes.

Cloud CDN offers three cache modes, which define how responses are cached, whether or not Cloud CDN respects cache directives sent by the origin, and how cache TTLs are applied.

The available cache modes are USE_ORIGIN_HEADERS, CACHE_ALL_STATIC and FORCE_CACHE_ALL.

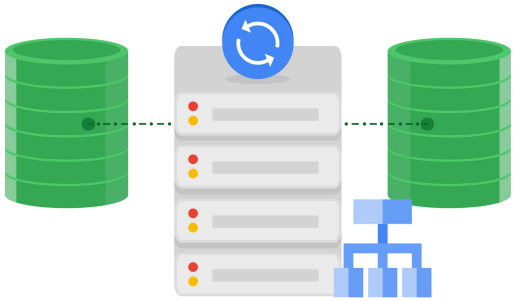
- USE_ORIGIN_HEADERS mode requires origin responses to set valid cache directives and valid caching headers.
- CACHE_ALL_STATIC mode automatically caches static content that doesn't have the no-store, private, or no-cache directive. Origin responses that set valid caching directives are also cached.
- FORCE_CACHE_ALL mode unconditionally caches responses, overriding any cache directives set by the origin. You should make sure not to cache private, per-user content (such as dynamic HTML or API responses) if using a shared backend with this mode configured.

Network Load Balancing

Network Load Balancers are Layer 4 load balancers that can distribute traffic to backends located either in a single region or across multiple regions. Let’s discuss these next.

Network load balancing

- Architecture:
  - Proxy
  - Passthrough
- Traffic:
  - TCP/SSL ports
  - UDP, ESP, GRE
  - ICMP, ICMPv6



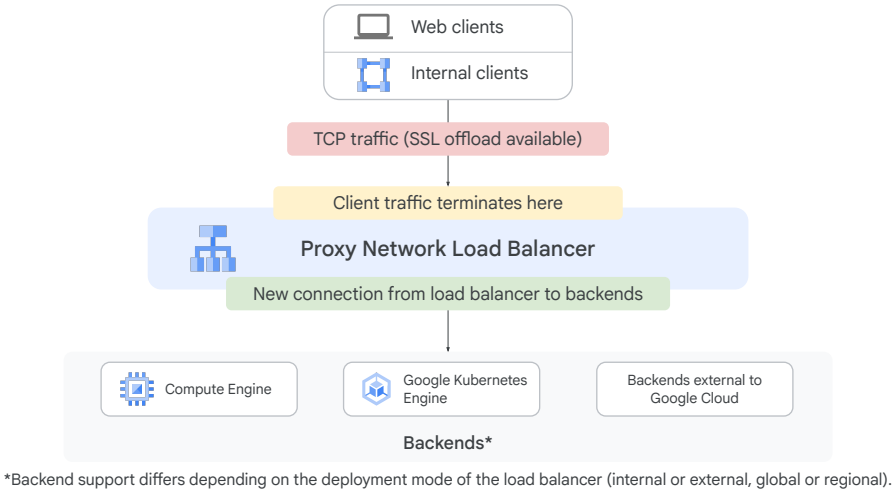
Network Load Balancers are Layer 4 load balancers that can handle TCP, UDP, or other IP protocol traffic. These load balancers are available as either proxy load balancers or passthrough load balancers. You can pick a load balancer depending on the needs of your application and the type of traffic that it needs to handle.

Choose a proxy Network Load Balancer if you want to configure a reverse proxy load balancer with support for advanced traffic controls and backends on-premises and in other cloud environments.

Choose a passthrough Network Load Balancer if you want to preserve the source IP address of the client packets, you prefer direct server return for responses, or you want to handle a variety of IP protocols such as TCP, UDP, ESP, GRE, ICMP, and ICMPv6.

Let's explore these in more detail.

Architecture of a Proxy Network Load Balancer



*Backend support differs depending on the deployment mode of the load balancer (internal or external, global or regional).

Proxy Network Load Balancers are Layer 4 reverse proxy load balancers that distribute TCP traffic to virtual machine (VM) instances in your Google Cloud VPC network. Traffic is terminated at the load balancing layer and then forwarded to the closest available backend by using TCP.

Proxy Network Load Balancers can be deployed externally or internally depending on whether your application is internet-facing or internal. We will discuss internal proxy Network Load Balancers later in this module.

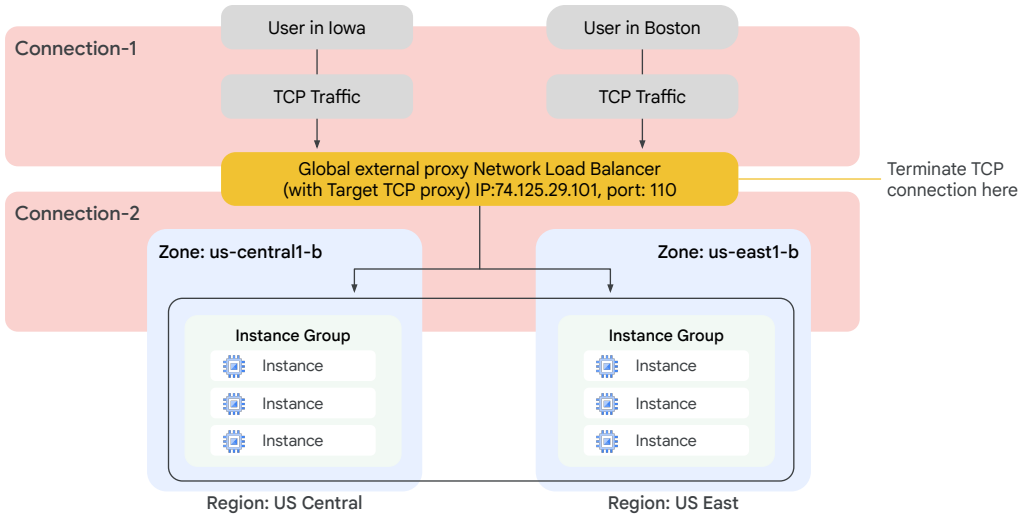
External proxy Network Load Balancers are Layer 4 load balancers that distribute traffic that comes from the internet to backends in your Google Cloud VPC network, on-premises, or in other cloud environments. These load balancers are built on either Google Front Ends (GFEs) or Envoy proxies.

These load balancers can be deployed in the following modes: global, regional, or classic.

Proxy Network Load Balancers are intended for TCP traffic only, with or without SSL. For HTTP(S) traffic, we recommend that you use an Application Load Balancer instead.

Depending on the type of traffic your application needs to handle, you can configure an external proxy Network Load Balancer with either a target TCP proxy or a target SSL proxy.

Proxy Network Load Balancer - Target TCP proxy

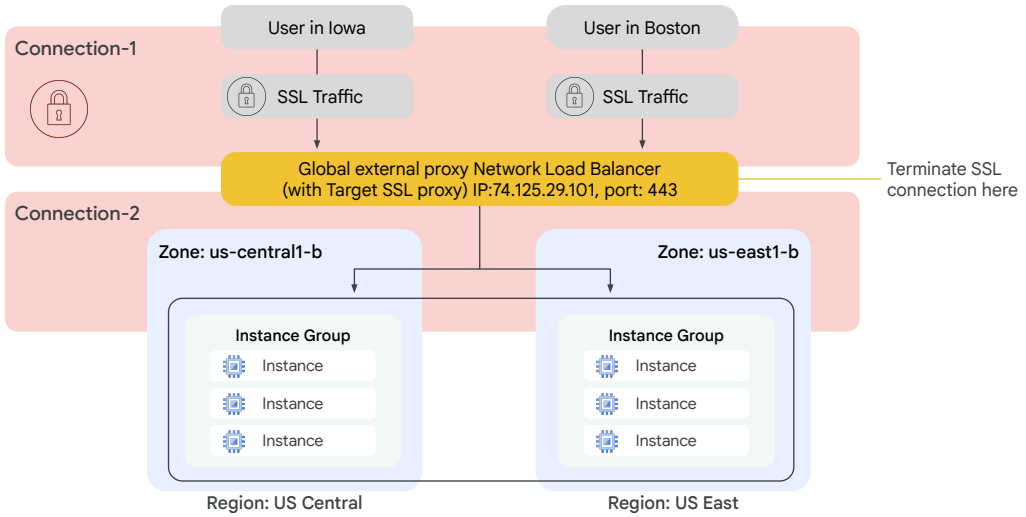


This network diagram illustrates an external proxy Network Load Balancer configured with a target TCP proxy.

In this example, traffic from users in Iowa and Boston is terminated at the global external proxy Network Load Balancer layer. From there, a separate connection is established to the closest backend instance. As in the target SSL proxy example in the next slide, the user in Boston would reach the us east region, and the user in Iowa would reach the us central region, if there is enough capacity.

Now, the traffic between the proxy and the backends can use SSL or TCP, and we also recommend using SSL here.

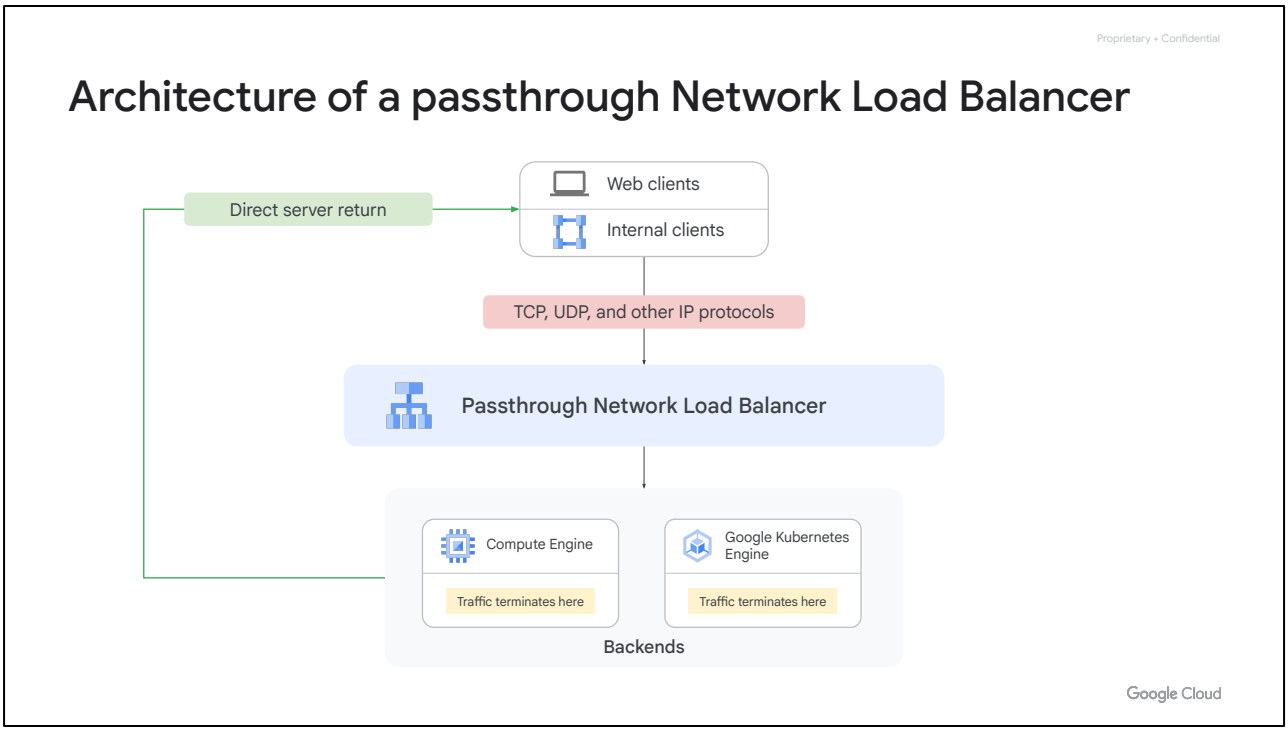
Proxy Network Load Balancer - Target SSL proxy



This network diagram illustrates an external proxy Network configured with a target SSL proxy.

In this example, traffic from users in Iowa and Boston is terminated at the global external proxy Network Load Balancer layer. From there, a separate connection is established to the closest backend instance. In other words, the user in Boston would reach the us east region, and the user in Iowa would reach the us central region, if there is enough capacity.

Now, the traffic between the proxy and the backends can use SSL or TCP. We recommend using SSL. For HTTP(S) traffic, we recommend that you use an external Application Load Balancer.



Passthrough Network Load Balancers are Layer 4 regional, passthrough load balancers. These load balancers distribute traffic among backends in the same region as the load balancer. They are implemented by using Andromeda virtual networking and Google Maglev. These load balancers are not proxies. Load-balanced packets are received by backend VMs with the packet's source and destination IP addresses, protocol, and, if the protocol is port-based, the source and destination ports unchanged. Load-balanced connections are terminated at the backends. Responses from the backend VMs go directly to the clients, not back through the load balancer. The industry term for this is direct server return (DSR).

These load balancers are deployed in two modes, depending on whether the load balancer is internet-facing or internal: We will discuss internal passthrough Network Load Balancers later in this module.

External passthrough Network Load Balancers are built on Maglev. Clients can connect to these load balancers from anywhere on the internet regardless of their Network Service Tiers. The load balancer can also receive traffic from Google Cloud VMs with external IP addresses or from Google Cloud VMs that have internet access through Cloud NAT or instance-based NAT. Backends for external passthrough Network Load Balancers can be deployed using either a backend service or a target pool. For new deployments, we recommend using backend services.

The architecture of an external passthrough Network Load Balancer depends on whether you use a backend service or a target pool to set up the backend.

Backend service-based architecture

- Regional backend service
- Defines the behavior of the load balancer and how it distributes traffic to its backend instance groups
- Support for IPv4 and IPv6 traffic
- Multiple protocols
- Managed and unmanaged instance groups
- Non legacy health checks

New network load balancers can be created with a regional backend service that defines the behavior of the load balancer and how it distributes traffic to its backend instance groups.

Backend service-based external passthrough Network Load Balancers support IPv4 and IPv6 traffic, multiple protocols (TCP, UDP, ESP, GRE, ICMP, and ICMPv6), managed and unmanaged instance group backends, zonal network endpoint group (NEG) backends with GCE_VM_IP endpoints, fine-grained traffic distribution controls, failover policies, and let you use non-legacy health checks that match the type of traffic (TCP, SSL, HTTP, HTTPS, or HTTP/2) that you are distributing.

You can also transition an existing target pool-based network load balancer to use a backend service instead. But what is a target pool resource?

Target pool-based architecture

- Forwarding rules (TCP and UDP)
- Up to 50 per project
- One health check
- Instances must be in the same region

A target pool is the legacy backend supported with external passthrough Network Load Balancers.

A target pool resource defines a group of instances that receive incoming traffic from forwarding rules. When a forwarding rule directs traffic to a target pool, the load balancer picks an instance from these target pools based on a hash of the source IP and port and the destination IP and port. These target pools can only be used with forwarding rules that handle TCP and UDP traffic.

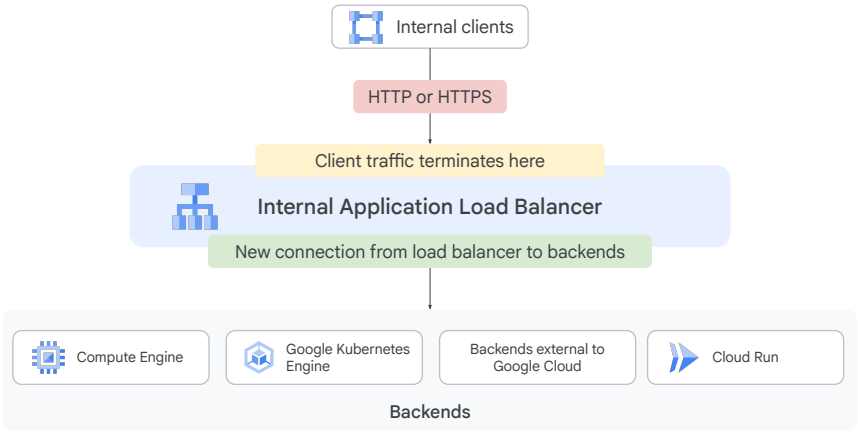
Now, each project can have up to 50 target pools, and each target pool can have only one health check.

Also, all the instances of a target pool must be in the same region, which is the same limitation as for the Network Load Balancer.

Internal Load Balancing

Next, let's talk about internal load balancing.

Architecture of an internal Application Load Balancer



Internal Application Load Balancers are Envoy proxy-based regional Layer 7 load balancers that enable you to run and scale your HTTP application traffic behind an internal IP address. Internal Application Load Balancers support backends in one region, but can be configured to be globally accessible by clients from any Google Cloud region.

You can configure an internal Application Load Balancer in either regional, or cross-region, internal Application Load Balancer mode.

Internal Application Load Balancers

Deployment mode	Network service tier	Load balancing scheme	IP address	Frontend ports
Regional internal	Premium Tier	INTERNAL_MANAGED	IPv4	Can reference exactly one port from 1-65535
Cross-region internal	Premium Tier	INTERNAL_MANAGED	IPv4	

Internal Application Load Balancers optimize traffic distribution within your VPC network or networks connected to your VPC network. You can configure an internal Application Load Balancer in the following modes, regional internal or cross-region.

A regional internal Application Load Balancer is implemented as a managed service based on the open-source Envoy proxy. Regional mode ensures that all clients and backends are from a specified region, which helps when you need regional compliance. This load balancer is enabled with rich traffic control capabilities based on HTTP(S) parameters. After the load balancer is configured, it automatically allocates Envoy proxies to meet your traffic needs.

A cross-region internal Application Load Balancer is a multi-region load balancer that is implemented as a managed service based on the open-source Envoy proxy. The cross-region mode enables you to load balance traffic to backend services that are globally distributed, including traffic management that ensures traffic is directed to the closest backend. This load balancer also enables high availability. Placing backends in multiple regions helps avoid failures in a single region. If one region's backends are down, traffic can fail over to another region.

Internal passthrough Network Load Balancers

- Regional, private load balancing
  - VM instances in same region
  - RFC 1918 IP addresses
- TCP, UDP, ICMP, ICMPv6, SCTP, ESP, AH, and GRE traffic
- Reduced latency, simpler configuration
- Software-defined, fully distributed load balancing

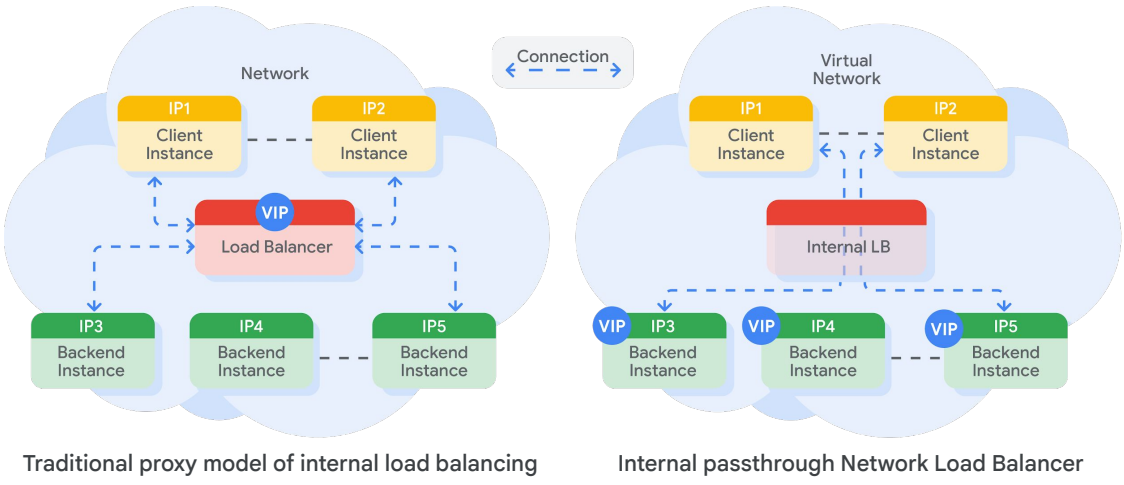
The internal passthrough Network Load Balancer is a regional, private load balancing service for when you need to load balance TCP, UDP, ICMP, ICMPv6, SCTP, ESP, AH, and GRE traffic, or when you need to load balance a TCP port that isn't supported by other load balancers.

In other words, this load balancer enables you to run and scale your services behind a private load balancing IP address This means that it is only accessible through the internal IP addresses of virtual machine instances that are in the same region.

Therefore, configure an internal passthrough Network Load Balancer IP address to act as the frontend to your private backend instances. Because you don't need a public IP for your load-balanced service, your internal client requests stay internal to your VPC network and region. This often results in lowered latency, because all your load-balanced traffic will stay within Google's network, making your configuration much simpler.

Let's talk more about the benefit of using a software-defined internal passthrough Network Load Balancer service.

Software-defined, fully distributed load balancing



Traditional proxy model of internal load balancing

Internal passthrough Network Load Balancer

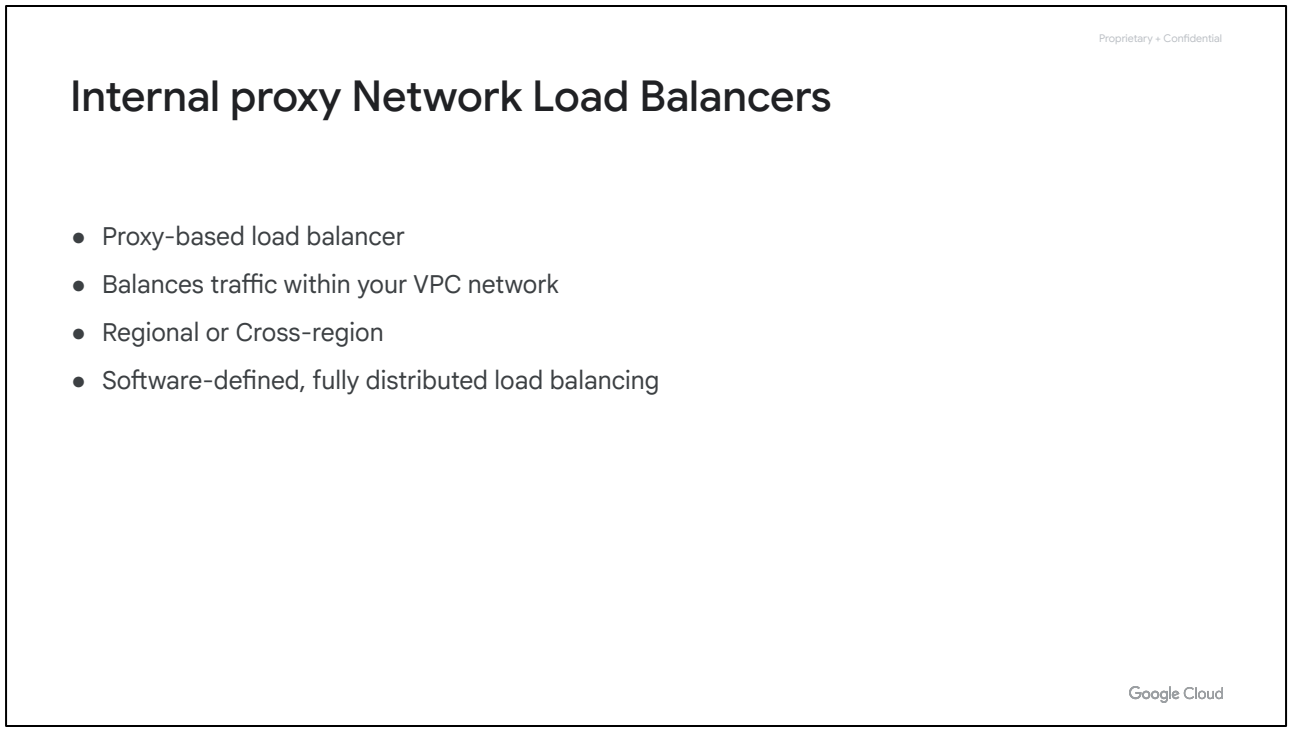
Google Cloud internal load balancing is not based on a device or a VM instance. Instead, it is a software-defined, fully distributed load balancing solution.

In the traditional proxy model of internal load balancing, as shown on the left, you configure an internal IP address on a load balancing device or instances, and your client instance connects to this IP address. Traffic coming to the IP address is terminated at the load balancer, and the load balancer selects a backend to establish a new connection to. Essentially, there are two connections: one between the Client and the Load Balancer, and one between the Load Balancer and the Backend.

Google Cloud internal passthrough Network Load Balancing distributes client instance requests to the backends using a different approach, as shown on the right. It uses lightweight load balancing built on top of Andromeda (Google's network virtualization stack) to provide software-defined load balancing that directly delivers the traffic from the client instance to a backend instance.

For more information on Andromeda, refer to the link in the Course Resources. [https://cloudplatform.googleblog.com/2014/04/enter-andromeda-zone-google-cloud-platforms-latest-networking-stack.html]

Now, let's take a look at internal proxy Network Load Balancers.



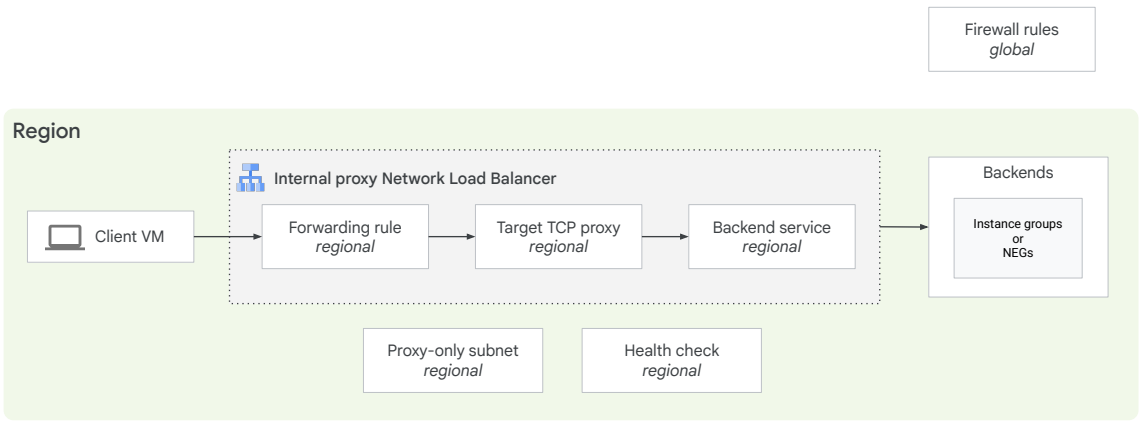
The Google Cloud internal proxy Network Load Balancer is a proxy-based load balancer powered by open source Envoy proxy software and the Andromeda network virtualization stack. It load balances traffic within your VPC network or networks connected to your VPC network.

The internal proxy Network Load Balancer is a layer 4 load balancer that enables you to run and scale your TCP service traffic behind a regional internal IP address that is accessible only to clients in the same VPC network or clients connected to your VPC network. The load balancer first terminates the TCP connection between the client and the load balancer at an Envoy proxy. The proxy opens a second TCP connection to backends hosted in Google Cloud, on-premises, or other cloud environments.

Internal proxy Network Load Balancers are available in regional internal or cross region internal deployment modes.

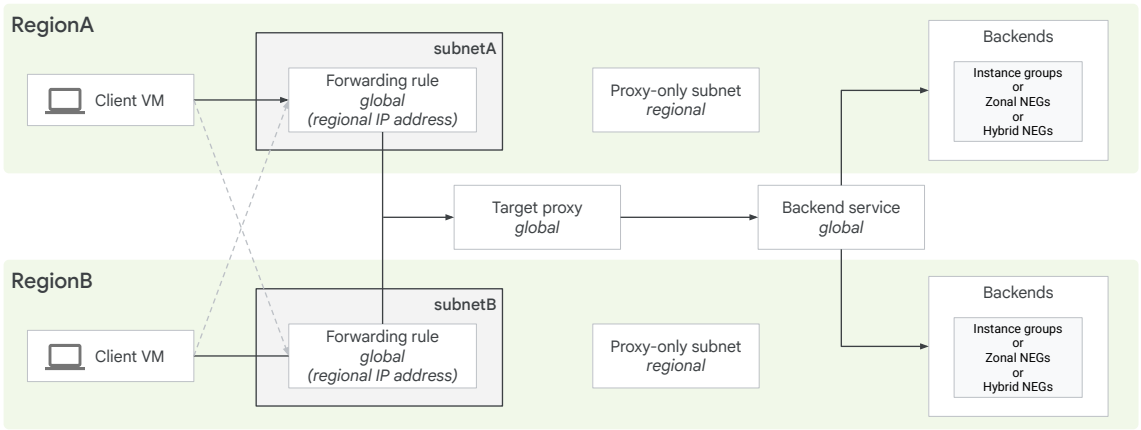
For more use cases, see Proxy Network Load Balancer overview.

Architecture of a regional internal proxy Network Load Balancer



A regional internal proxy Network Load Balancer is implemented as a managed service based on the open source Envoy proxy. Regional mode ensures that all clients and backends are from a specified region, which helps when you need regional compliance. This diagram shows the components of a regional internal proxy Network Load Balancer deployment in Premium Tier.

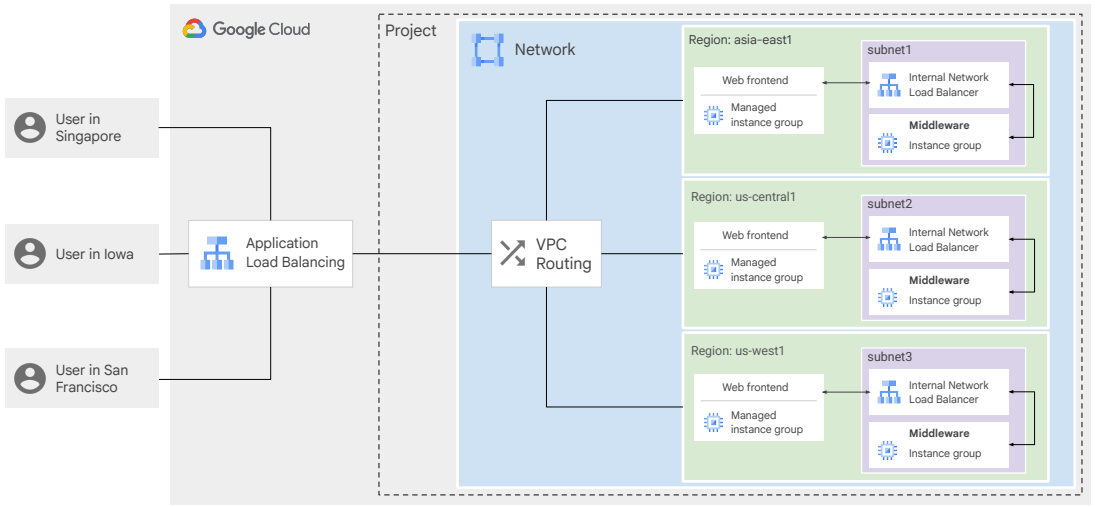
Architecture of a cross-region internal proxy Network Load Balancer



This diagram shows the components of a cross-region internal proxy Network Load Balancer deployment in Premium Tier within the same VPC network. Each global forwarding rule uses a regional IP address that the clients use to connect.

This is a multi-region load balancer that is implemented as a managed service based on the open source Envoy proxy. The cross-region mode lets you load balance traffic to backend services that are globally distributed, including traffic management that ensures traffic is directed to the closest backend. This load balancer also enables high availability. Placing backends in multiple regions helps avoid failures in a single region. If one region's backends are down, traffic can fail over to another region.

Internal load balancing supports 3-tier web services



Now, internal load balancing enables you to support use cases such as the traditional 3-tier web services.

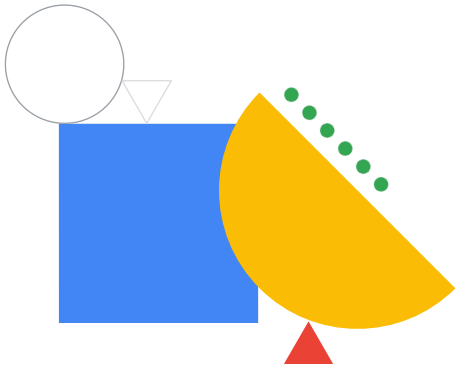
In this example, the web tier uses an external Application Load Balancer that provides a single global IP address for users in San Francisco, Iowa, Singapore, and so on. The backends of this load balancer are located in the us-west1, us-central1, and asia-east1 regions because this is a global load balancer.

These backends then access an internal Network Load Balancer in each region as the application or internal tier. The backends of this internal tier are located in us-west1-a, us-central1-b, and asia-east1-b. The last tier is the database tier in each of those zones.

The benefit of this 3-tier approach is that neither the database tier nor the application tier is exposed externally. This simplifies security and network pricing.

Lab Intro

Configure an Internal Network Load Balancer



Let’s apply some of the internal load balancer concepts that we just discussed in a lab.

Lab objectives

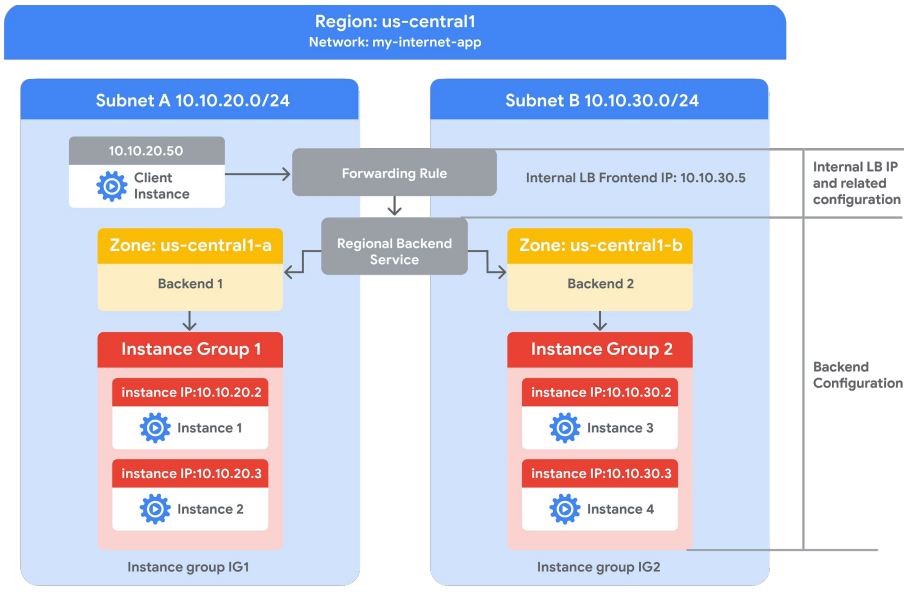
01 Create HTTP and health check firewall rules

02 Configure two instance templates

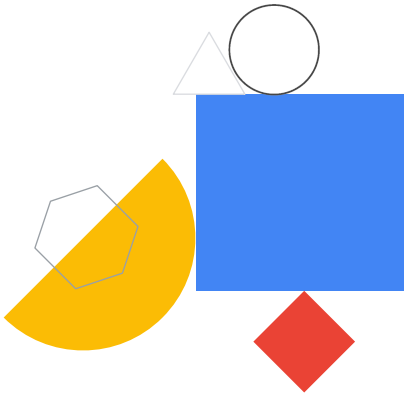
03 Create two managed instance groups

04 Configure and test an internal Network Load Balancer

In this lab, you will create two managed instance groups in the same region. Then, you configure and test an internal Network Load Balancer with the instances groups as the backends, as shown in this network diagram.



Review: Configure an Internal Load Balancer



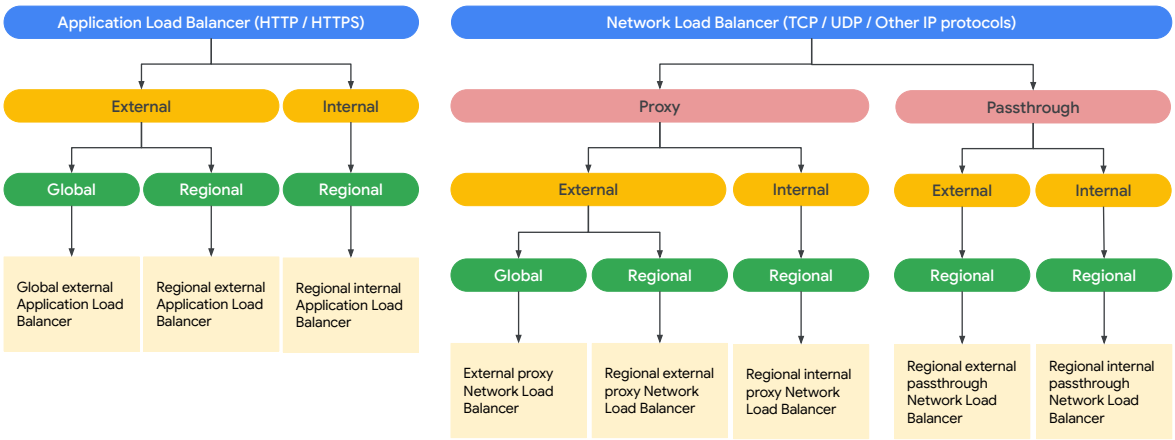
In this lab, you created two managed instance groups in the us-central1 region, along with firewall rules to allow HTTP traffic to those instances and TCP traffic from the Google Cloud health checker. Then, you configured and tested an internal load balancer for those instance groups.

You can stay for a lab walkthrough, but remember that Google Cloud's user interface can change, so your environment might look slightly different.

Choosing a Load Balancer

Now that we've discussed all the different load balancing services within Google Cloud, let me help you determine which load balancer best meets your needs.

Deployment modes available for Cloud Load Balancing



To determine which Cloud Load Balancing product to use, you must first determine what traffic type your load balancers must handle. As a general rule, you'd choose an Application Load Balancer when you need a flexible feature set for your applications with HTTP(S) traffic. You'd choose a proxy Network Load Balancer to implement TLS offload, TCP proxy, or support for external load balancing to backends in multiple regions. You'd choose a passthrough Network Load Balancer to preserve client source IP addresses, avoid the overhead of proxies, and to support additional protocols like UDP, ESP, and ICMP, or if you need to expose client IP addresses to your applications.

You can further narrow down your choices depending on your application's requirements: whether your application is external (internet-facing) or internal and whether you need backends deployed globally or regionally.

Summary of Google Cloud load balancers

Load balancer	Deployment mode	Traffic type	Network Service Tier	Load-balancing scheme
Application Load Balancers	Global external	HTTP or HTTPS	Premium	EXTERNAL_MANAGED
	Regional external	HTTP or HTTPS	Standard	EXTERNAL_MANAGED
	Classic	HTTP or HTTPS	Global in Premium Regional in Standard	EXTERNAL
	Internal Always regional	HTTP or HTTPS	Premium	INTERNAL_MANAGED
Proxy Network Load Balancers	Global external	TCP with optional SSL offload	Global in Premium Regional in Standard	EXTERNAL
	Regional external	TCP	Standard only	EXTERNAL_MANAGED
	Internal Always regional	TCP without SSL offload	Premium only	INTERNAL_MANAGED
Passthrough Network Load Balancers	External Always regional	TCP, UDP, ESP, GRE, ICMP, and ICMPv6	Premium or Standard	EXTERNAL
	Internal Always regional	TCP or UDP	Premium only	INTERNAL

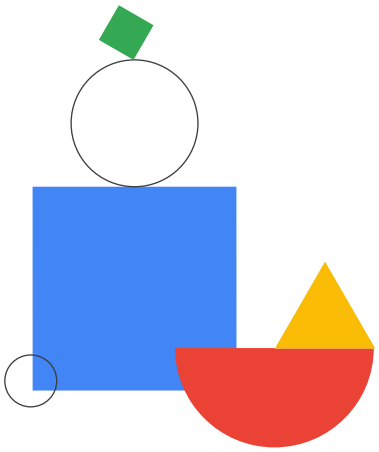
If you prefer a table over a flow chart, we recommend this summary table.

The load-balancing scheme is an attribute on the forwarding rule and the backend service of a load balancer and indicates whether the load balancer can be used for internal or external traffic.

The term MANAGED in the load-balancing scheme indicates that the load balancer is implemented as a managed service either on Google Front Ends or on the open source Envoy proxy. In a load-balancing scheme that is MANAGED, requests are routed either to the Google Front End or to the Envoy proxy.

For more information on Network Service Tiers, refer to the documentation link in the Course Resources.

Review: Load Balancing and Autoscaling



In this module, we looked at the different types of load balancers that are available in Google Cloud, along with managed instance groups and autoscaling. You were able to apply most of the covered services and concepts by working through the two labs of this module.

We also discussed the criteria for choosing between the different load balancers and looked at a flow chart and a summary table to help you pick the right load balancers. Remember, sometimes it's useful to combine an internal and an external load balancer to support 3-tier web services.