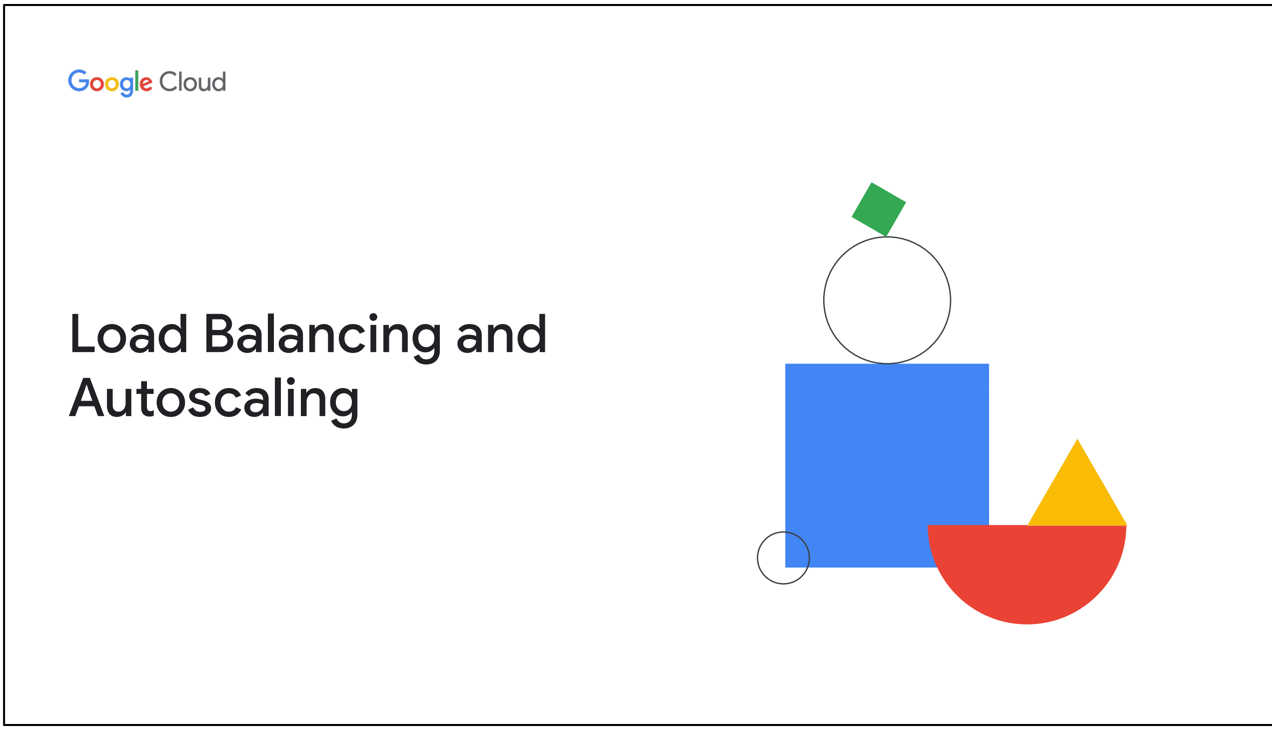
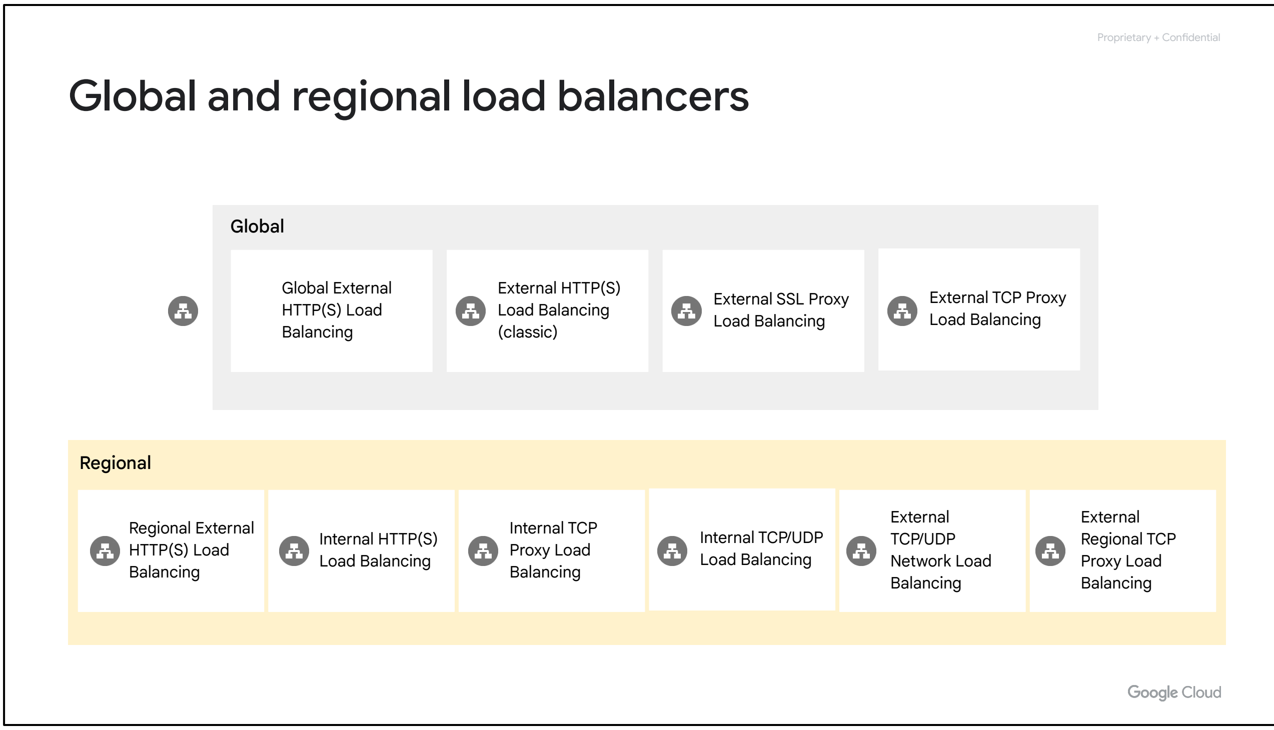


In this module we focus on Load Balancing and Autoscaling.

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.



Google Cloud offers different types of load balancers that can be divided into two categories: global and regional.

The global load balancers are the HTTP(S), SSL proxy, and TCP proxy load balancers. These load balancers leverage the Google frontends, which are software-defined, distributed systems that sit in Google's points of presence and are distributed globally. Therefore, you want to use a global load balancer when your users and instances are globally distributed, your users need access to the same applications and content, and you want to provide access using a single anycast IP address.

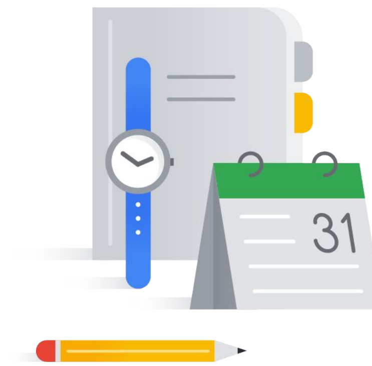
The regional load balancers are external and internal HTTP(S), and TCP Proxy. There are also internal TCP/UDP, and external TCP/UDP Network load balancers.

The internal and network load balancers distribute traffic to instances that are in a single Google Cloud region. The internal load balancer uses Andromeda, which is Google Cloud's software-defined network virtualization stack, and the network load balancer uses Maglev, which is a large, distributed software system.

The internal load balancer for HTTP(S) traffic is a proxy-based, regional Layer 7 load balancer that enables you to run and scale your services behind a private load balancing IP address that is accessible only in the load balancer's region in your VPC network.

Agenda

- **01** Managed Instance Groups
- **02** HTTP(S) Load Balancing
  - Lab: Configuring an HTTP Load Balancer with Autoscaling
- **03** Cloud CDN
- **04** SSL Proxy/TCP Proxy Load Balancing
- **05** Network Load Balancing
- **06** Internal Load Balancing
  - Lab: Configuring an Internal Load Balancer
- **07** Choosing a Load Balancer



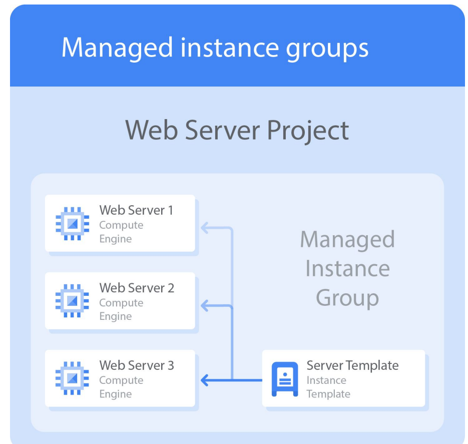
In this module, we cover the different types of load balancers that are available in Google Cloud. We also go over managed instance groups and their autoscaling configurations, which can be used by these load balancing configurations.

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.

Let’s start by talking about managed instance groups.

Managed instance groups

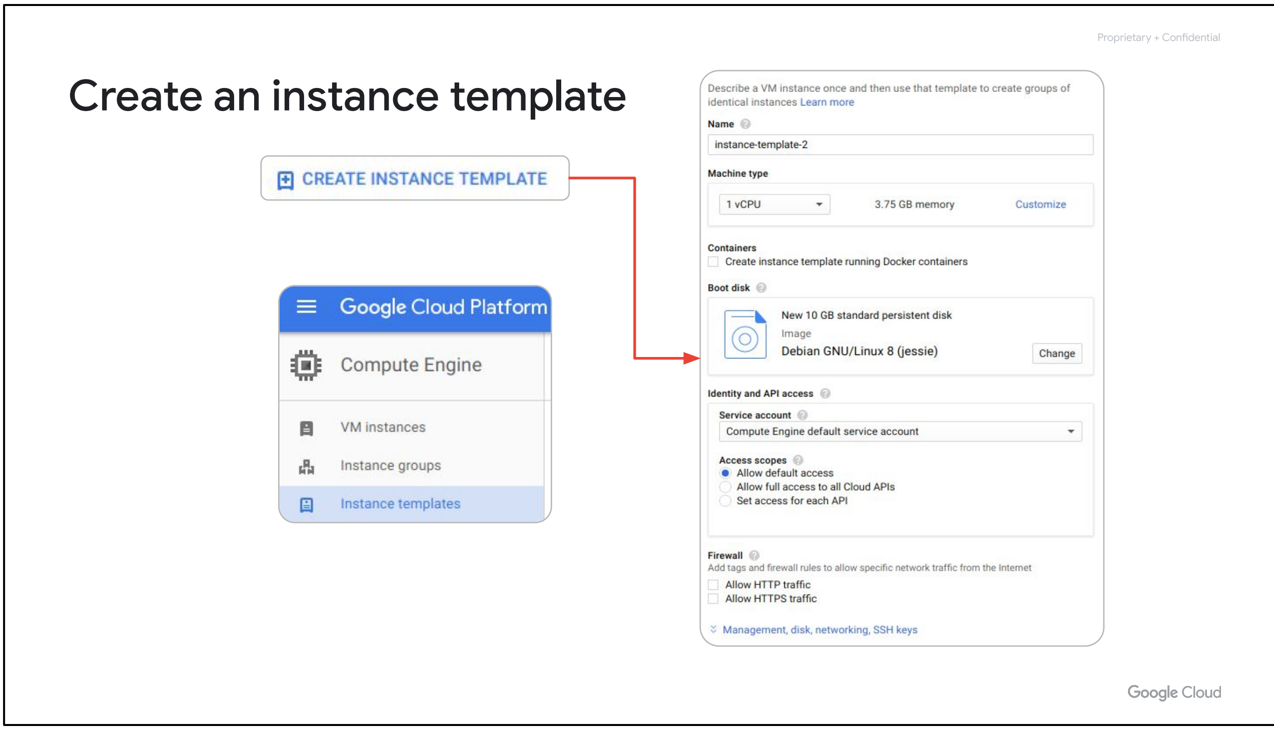
- Deploy identical instances based on instance template
- Instance group can be resized
- Manager ensures all instances are RUNNING
- Typically used with autoscaler
- Can be single zone or regional



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.

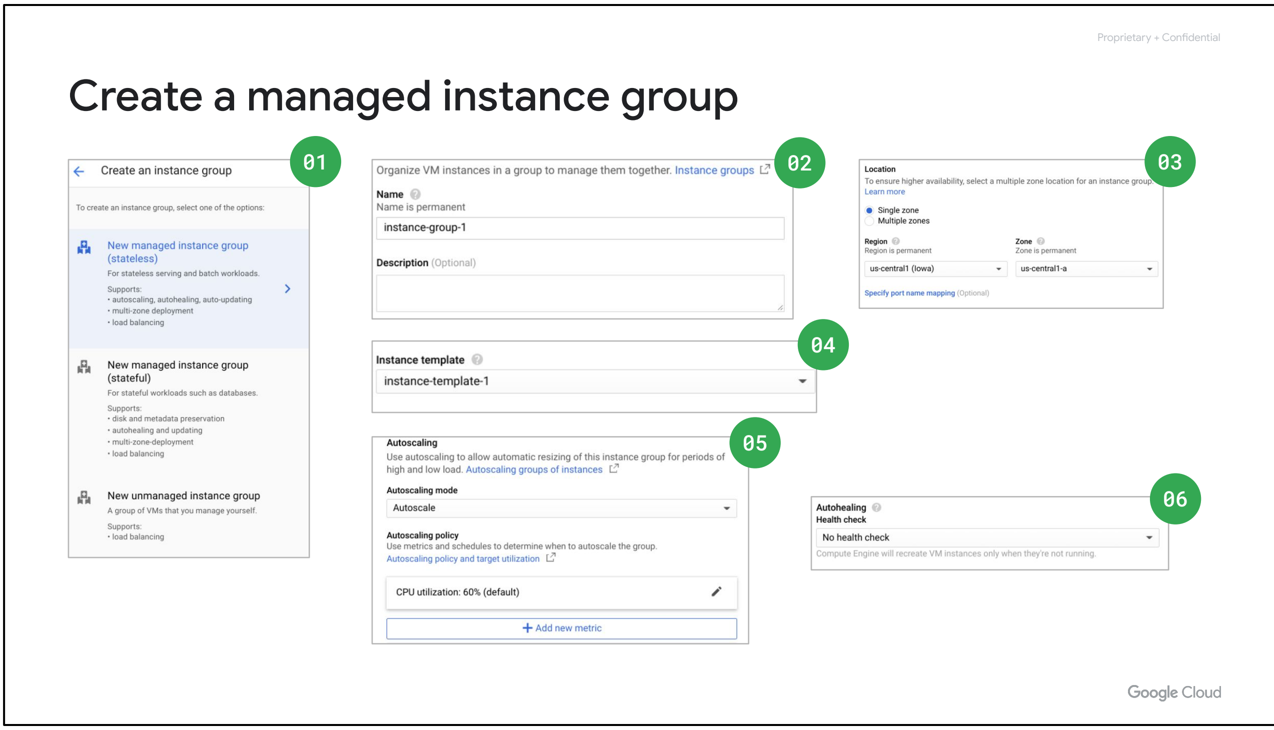
Managed instance groups can work with load balancing services to distribute network traffic to all of the instances in the group. If an instance in the group stops, crashes, or is deleted by an action other than the instance group's commands, the managed instance group automatically recreates the instance 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.



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 GCP 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.



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.

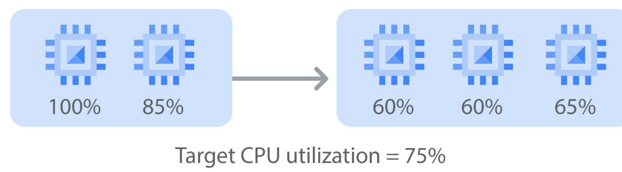
Managed instance groups offer autoscaling capabilities

Dynamically add/remove instances:

- Increases in load
- Decreases in load

Autoscaling policy:

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



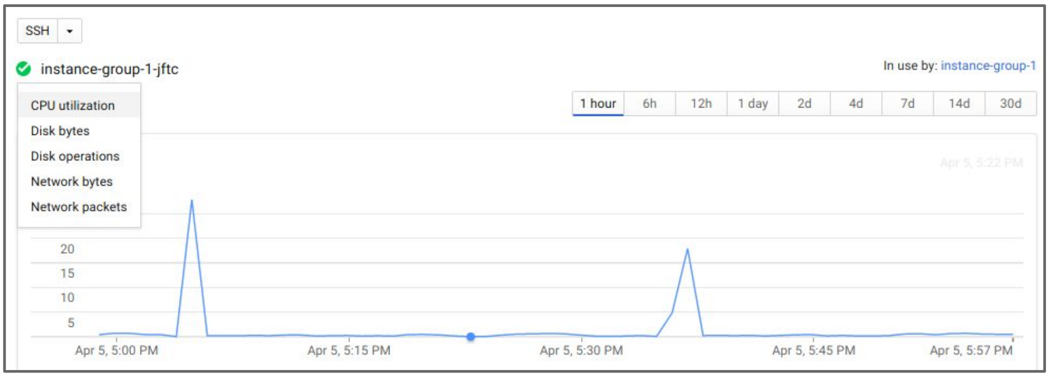
Let me provide more details on the autoscaling and health checks of a managed instance group.

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 Cloud Pub/Sub.

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

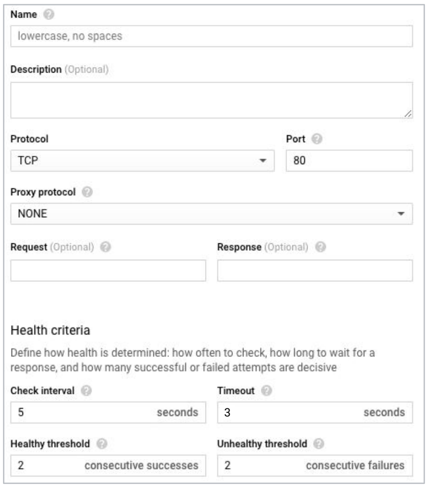


When you click on an instance group (or even an individual VM), a graph is presented. By default you'll see the CPU utilization over the past hour, but you can change the time frame and visualize other metrics like disk and network usage. These graphs are very useful for monitoring your instances' utilization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

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

For more information on autoscaling, see the links section of this video.
[https://cloud.google.com/compute/docs/autoscaler/]

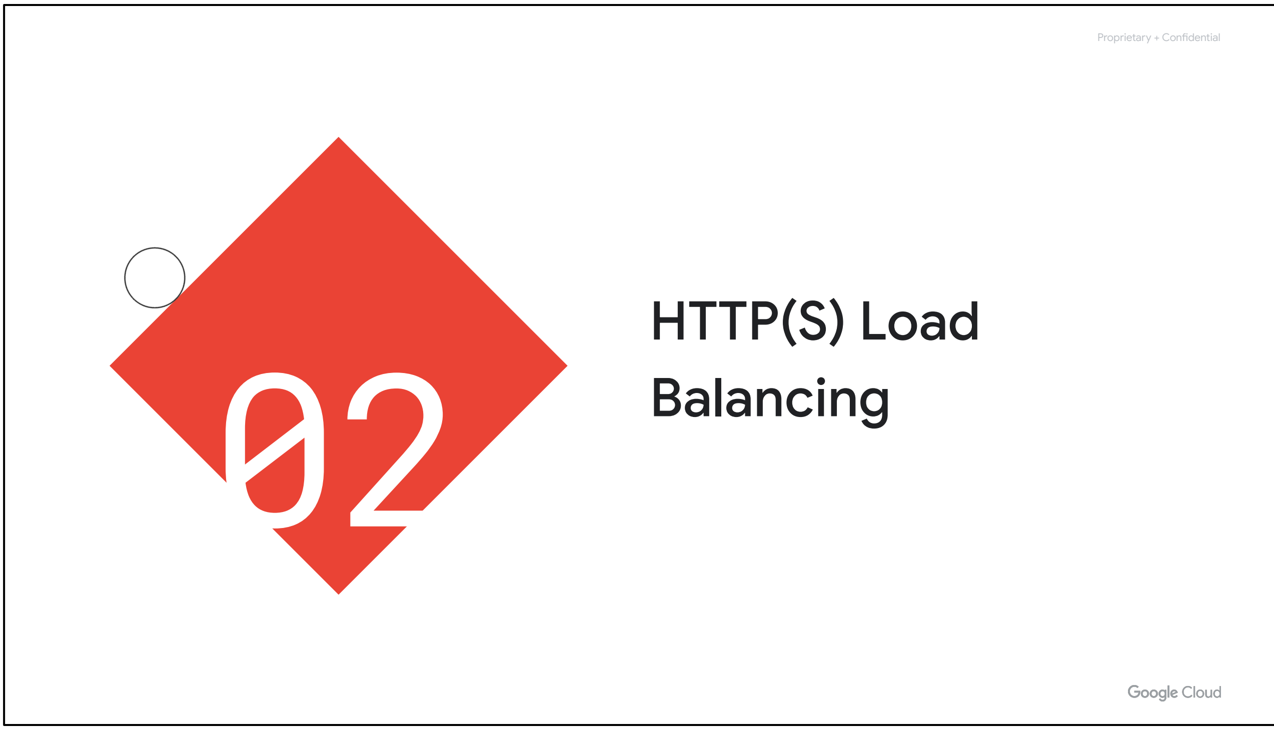
Create a health check



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 Stackdriver. You just define a protocol, port, and health criteria, as shown in this screenshot. Based on this configuration, GCP computes a health state for each instance.

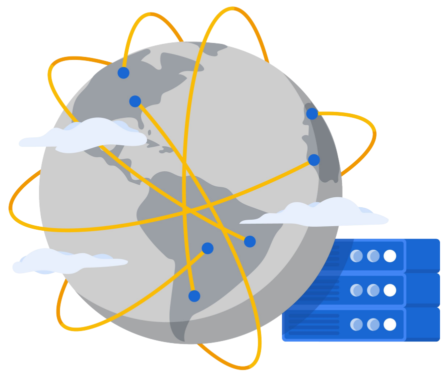
The health criteria define how often to check whether an instance is healthy (that's the check interval); how long to wait for a response (that's the timeout); how many successful attempts are decisive (that's the healthy threshold); and how many failed attempts are decisive (that's the unhealthy threshold). 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.



Now, let's talk about HTTP(S) 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.

HTTP(S) load balancing

- Global load balancing
- Anycast IP address
- HTTP or port 80 or 8080
- HTTPs on port 443
- IPv4 or IPv6
- Autoscaling
- URL maps



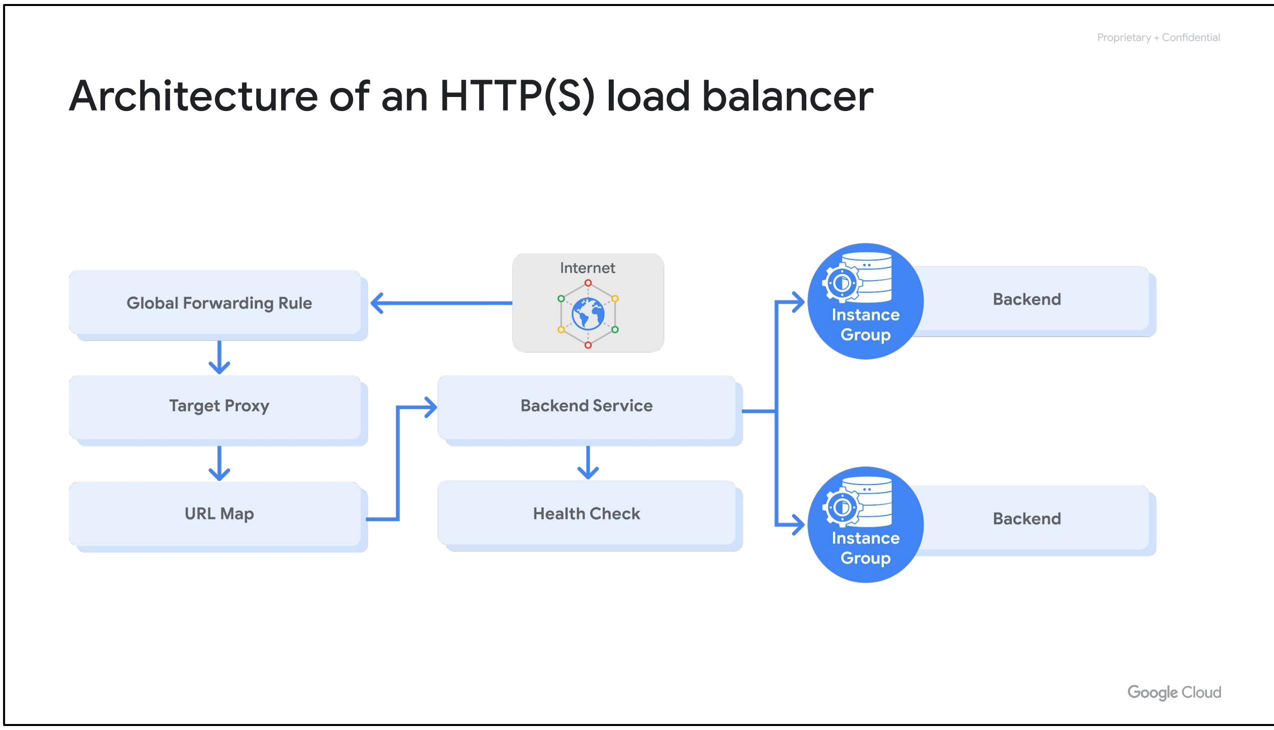
HTTP(S) Load Balancing

GCP's HTTP(S) load balancing provides global load balancing for HTTP(S) requests destined for your instances. This means that your applications are available to your customers at a single anycast IP address, which simplifies your DNS setup. HTTP(S) load balancing balances HTTP and HTTPS traffic across multiple backend instances and across multiple regions.

HTTP requests are load balanced on port 80 or 8080, and HTTPS requests are load balanced on port 443.

This load balancer supports both IPv4 and IPv6 clients, is scalable, requires no pre-warming, and enables content-based and cross-region load balancing.

You can configure URL maps that route some URLs to one set of instances and route other URLs to other instances. Requests are generally routed to the instance group that is closest to the user. If the closest instance group does not have sufficient capacity, the request is sent to the next closest instance group that does have capacity. You will get to explore most of these benefits in the first lab of the module.

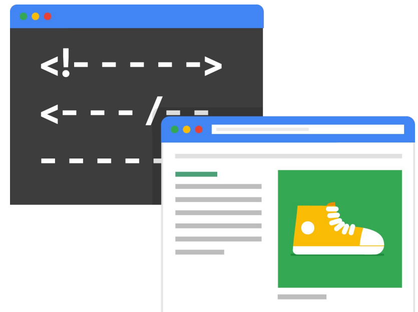


Let me walk through the complete architecture of an HTTP(S) load balancer, by using this diagram:

- A global forwarding rule directs incoming requests from the internet to a target HTTP proxy.
- The target HTTP proxy checks each request against a URL map to determine the appropriate backend service for the request. For example, you can send requests for www.example.com/audio to one backend service, which contains instances configured to deliver audio files, and requests for www.example.com/video to another backend service, which contains instances configured to deliver video files.
- The backend service directs each request to an appropriate backend based on serving capacity, zone, and instance health of its attached backends.

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, HTTP(S) 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.

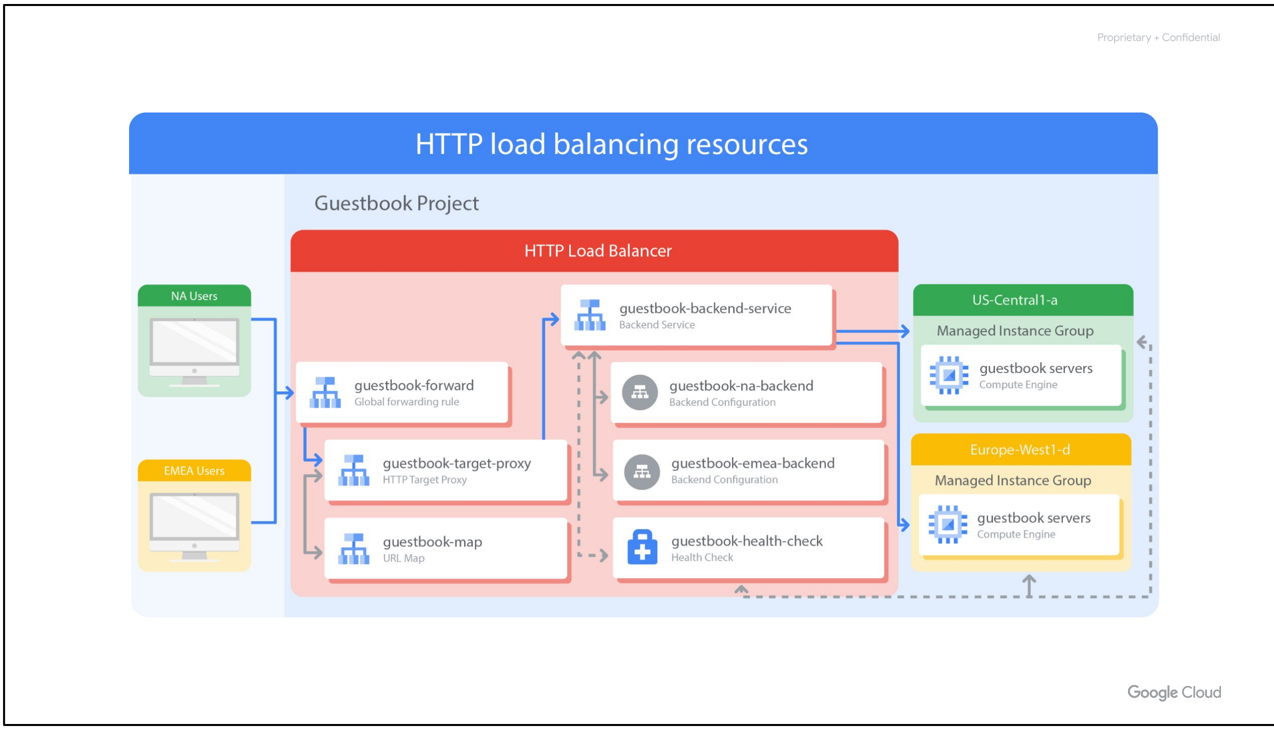
Backend services also have a timeout setting, which is set to 30 seconds by default. This is the amount of time the backend service will wait on the backend before considering the request a failure. This is a fixed timeout, not an idle timeout. If you require longer-lived connections, set this value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



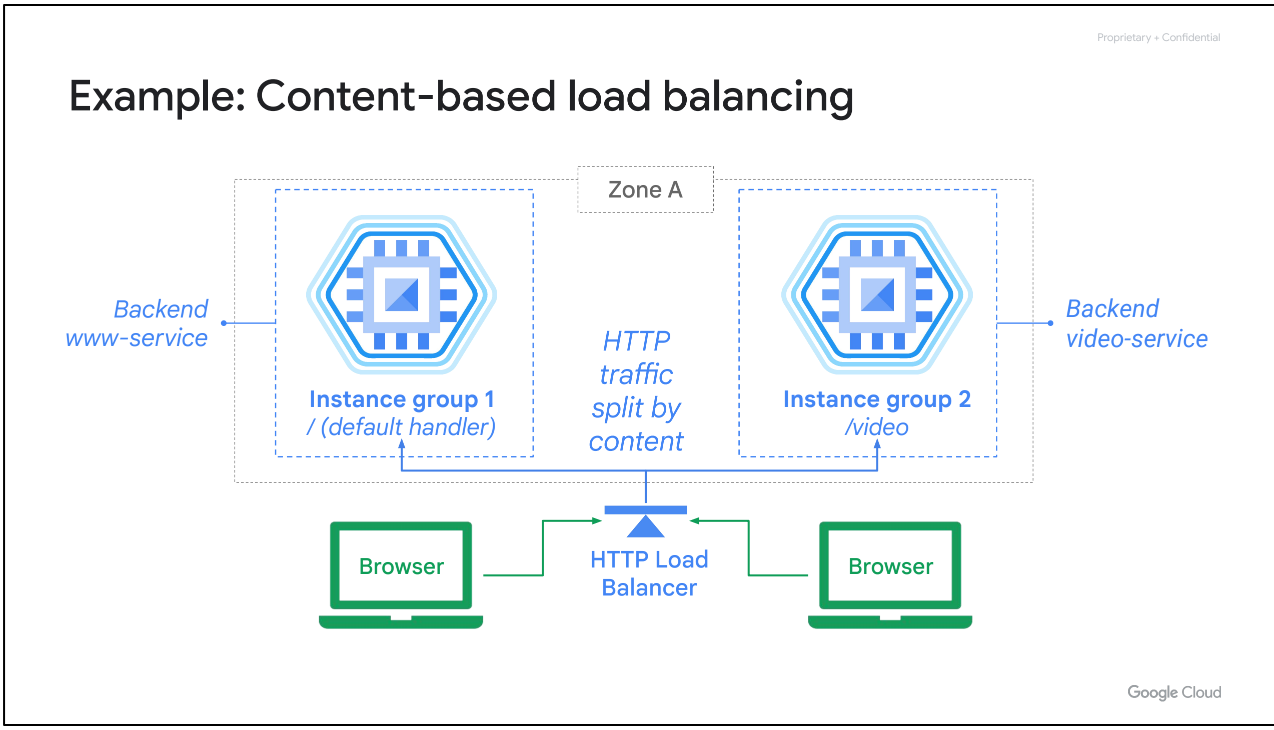
Let me walk through an HTTP load balancer in action. The project on this slide has a single global IP address, but users enter the Google Cloud network from two different locations: one in North America and one in EMEA.

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.

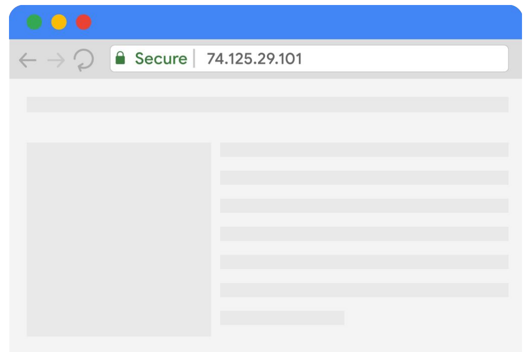


Another example of an HTTP load balancer is a content-based load balancer. In this case, there are two separate backend services that handle either web or video traffic.

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.

HTTP(S) load balancing

- 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 HTTP(S) load balancer has the same basic structure as an HTTP load balancer, but differs in the following ways:

- An HTTP(S) load balancer uses a target HTTPS proxy instead of a target HTTP proxy.
- An HTTP(S) 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.
- HTTP(S) load balancers support 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, see the link in the Course Resources. [https://www.chromium.org/quic]

SSL certificates

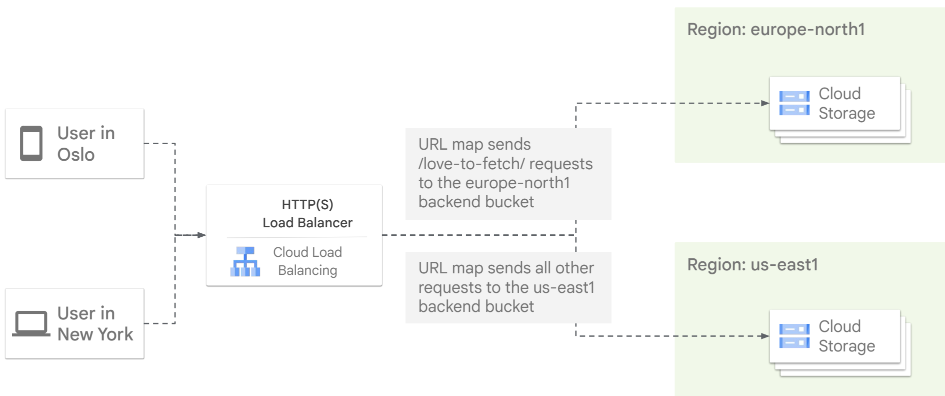
- Required for HTTP(S) load balancing
- Up to 15 SSL certificates (per target proxy)
- Create an SSL certificate resource

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



Google Cloud

Backend buckets allow you to use Google Cloud Storage buckets with HTTP(S) Load Balancing.

An external HTTP(S) 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 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

Configuring an HTTP 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

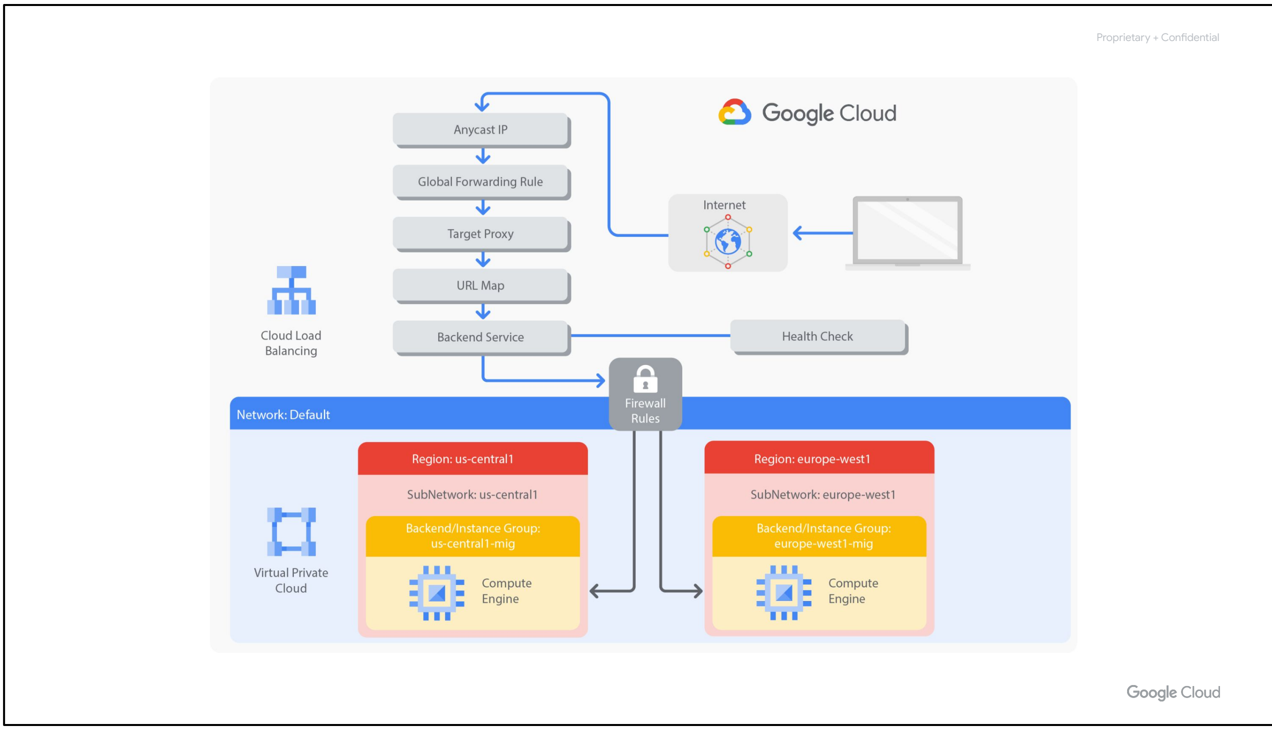
03 Create an instance template based on the custom image

04 Create two managed instance groups

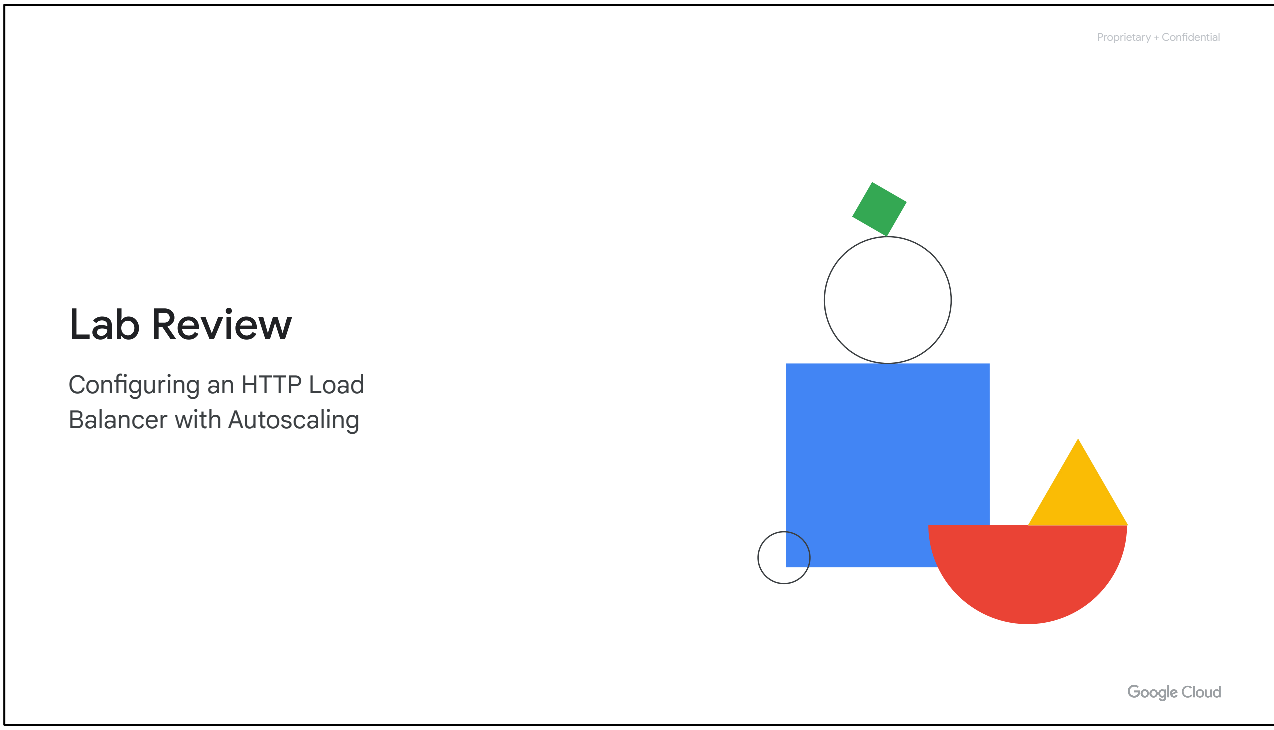
05 Configure an HTTP load balancer with IPv4 and IPv6

06 Stress test an HTTP load balancer

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



Specifically, you create two managed instance groups that serve as backends in us-central1 and europe-west1. Then, you create and stress test a load balancer to demonstrate global load balancing and autoscaling.



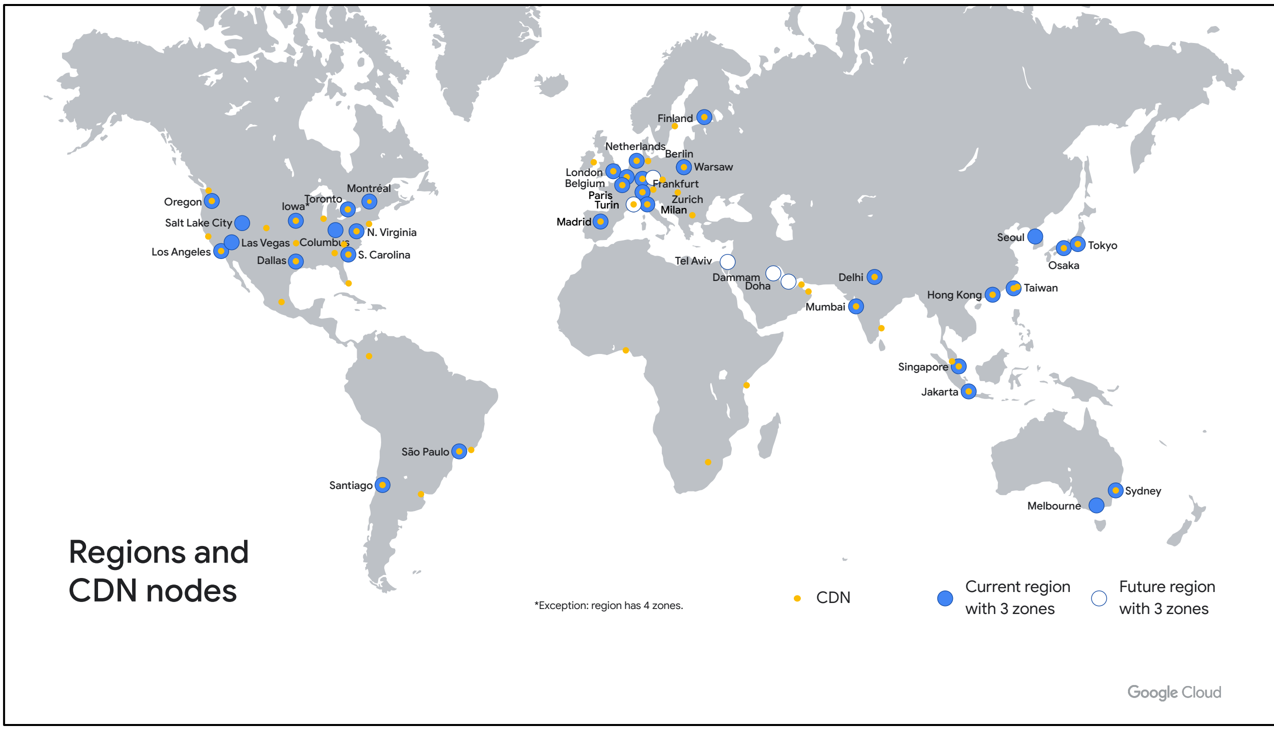
In this lab, you configured an HTTP load balancer with backends in us-central1 and europe-west1. Then you stress-tested the load balancer with a VM to demonstrate global load balancing and autoscaling.

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

03

Cloud CDN

Cloud CDN (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.

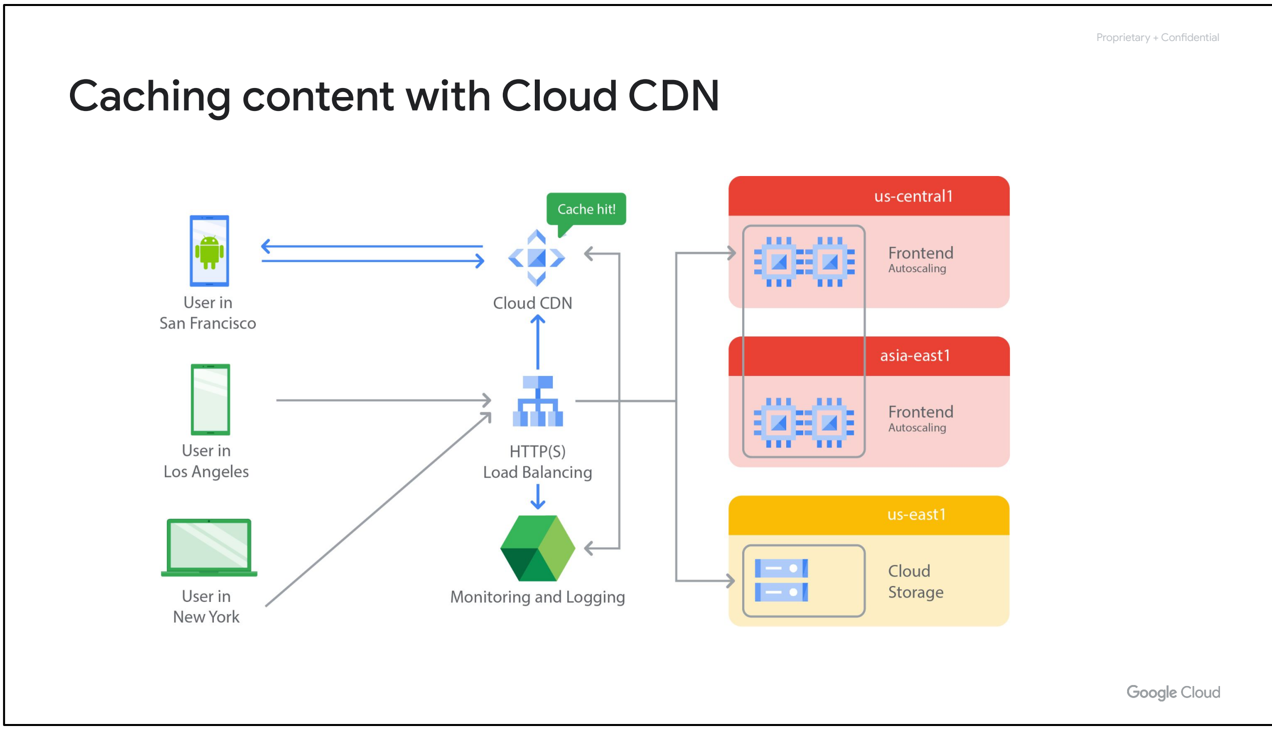
Now, why should you consider using Cloud CDN?

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 HTTP(S) 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/]



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

In this example, the HTTP(S) 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 HTTP(S) 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.

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

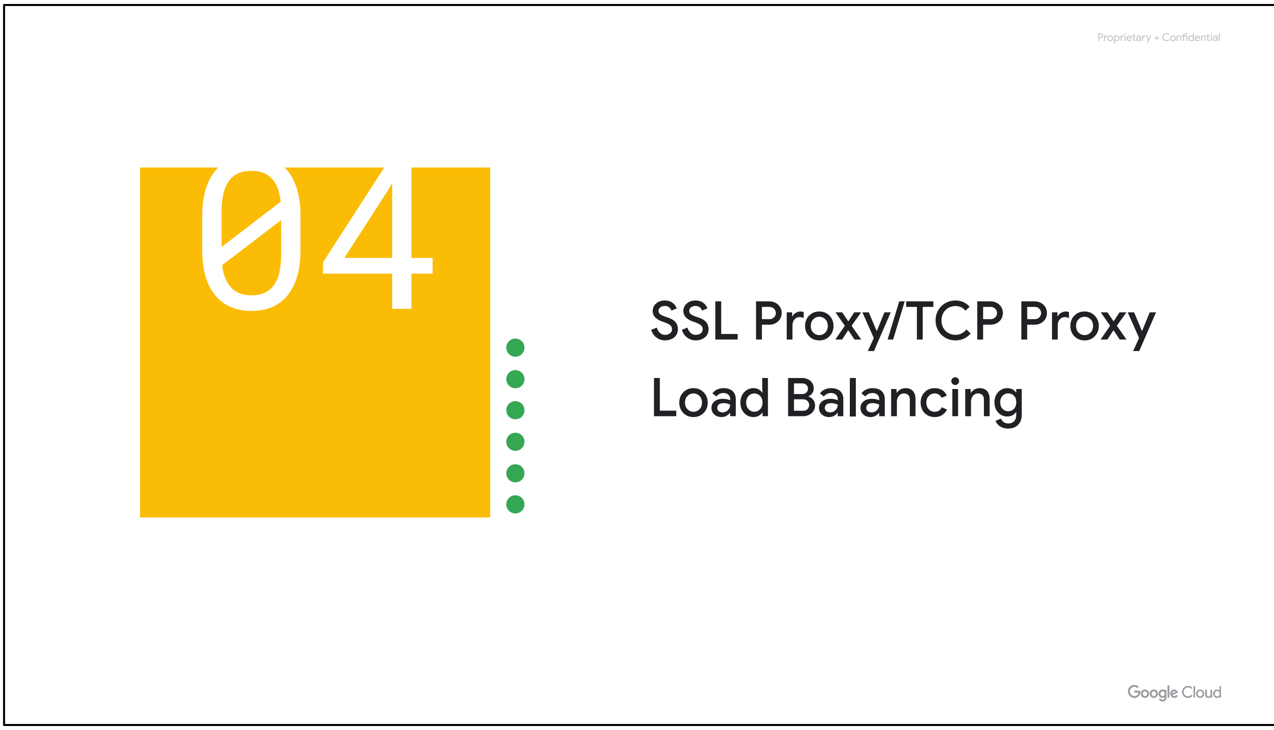
- Cache modes control the factors that determine whether or not Cloud CDN caches your content.
- Cloud CDN offers three cache modes:
  - USE_ORIGIN_HEADERS
  - CACHE_ALL_STATIC
  - FORCE_CACHE_ALL

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

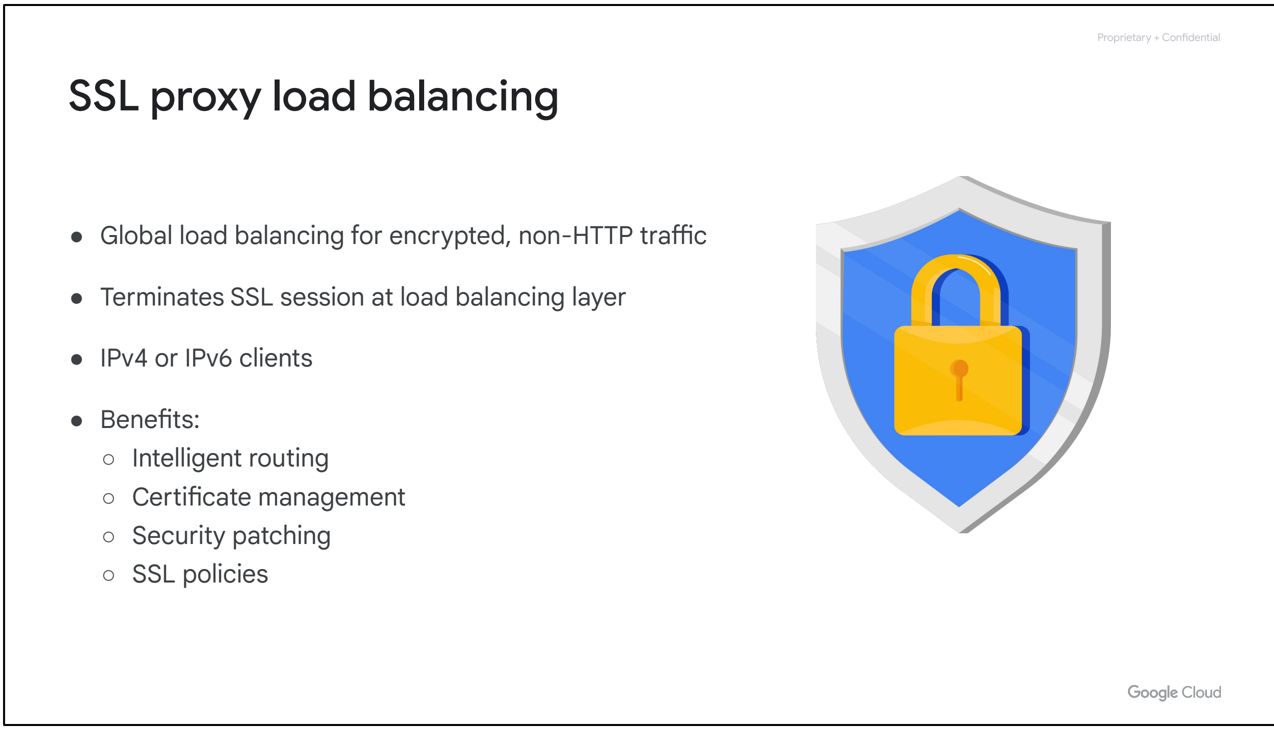
Cloud CDN offers three cache modes, which define how responses are cached, whether or not Cloud CDN respects cache directives sent by the origin, and how cache TTLs are applied.

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.



Let’s talk about SSL proxy and TCP proxy load balancing.



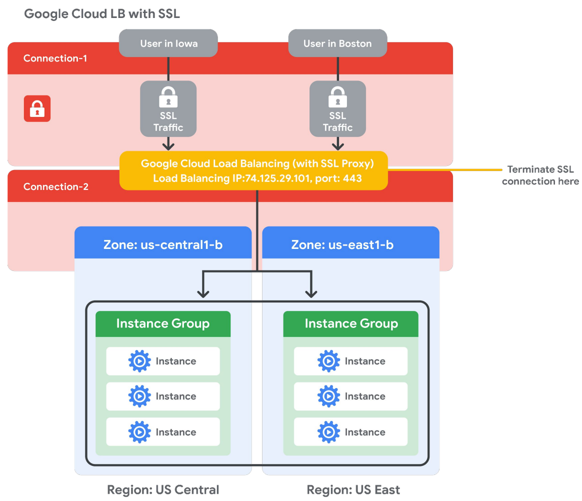
SSL proxy is a global load balancing service for encrypted, non-HTTP traffic. This load balancer terminates user SSL connections at the load balancing layer, then balances the connections across your instances using the SSL or TCP protocols. These instances can be in multiple regions, and the load balancer automatically directs traffic to the closest region that has capacity.

SSL proxy load balancing supports both IPv4 and IPv6 addresses for client traffic and provides Intelligent routing, Certificate management, Security patching and SSL policies.

- Intelligent routing means that this load balances can route requests to backend locations where there is capacity.
- From a certificate management perspective, you only need to update your customer-facing certificate in one place when you need to switch certificates. Also, you can reduce the management overhead for your virtual machine instances by using self-signed certificates on your instances.
- In addition, if vulnerabilities arise in the SSL or TCP stack, GCP will apply patches at the load balancer automatically in order to keep your instances safe.

For the full list of ports supported by SSL proxy load balancing and other benefits, please refer to the links section of this video.
[https://cloud.google.com/load-balancing/docs/ssl/#overview]

Example: SSL proxy load balancing



This network diagram illustrates SSL proxy load balancing. In this example, traffic from users in Iowa and Boston is terminated at the global load balancing 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. I recommend using SSL.

Proprietary + Confidential

TCP proxy load balancing

- Global load balancing for unencrypted, non-HTTP traffic
- Terminates TCP sessions at load balancing layer
- IPv4 or IPv6 clients
- Benefits:
  - Intelligent routing
  - Security patching

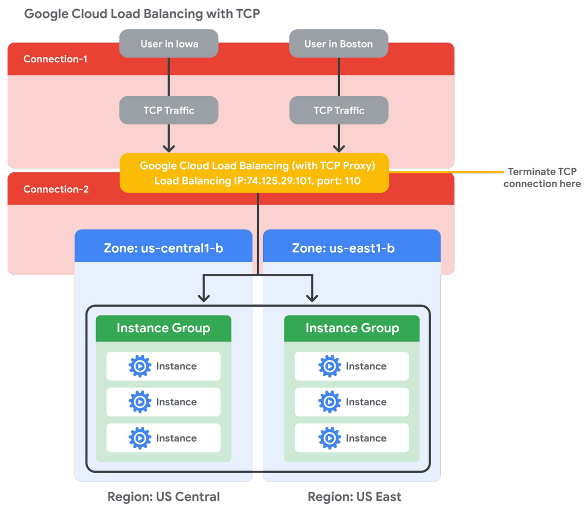
Google Cloud

TCP proxy is a global load balancing service for unencrypted, non-HTTP traffic. This load balancer terminates your customers' TCP sessions at the load balancing layer, then forwards the traffic to your virtual machine instances using TCP or SSL. These instances can be in multiple regions, and the load balancer automatically directs traffic to the closest region that has capacity.

TCP proxy load balancing supports both IPv4 and IPv6 addresses for client traffic. Similar to the SSL proxy load balancer, the TCP proxy load balancer provides Intelligent routing and Security patching.

For the full list of ports supported by TCP proxy load balancing and other benefits, please refer to the links section of this video.
[https://cloud.google.com/load-balancing/docs/tcp/#overview]

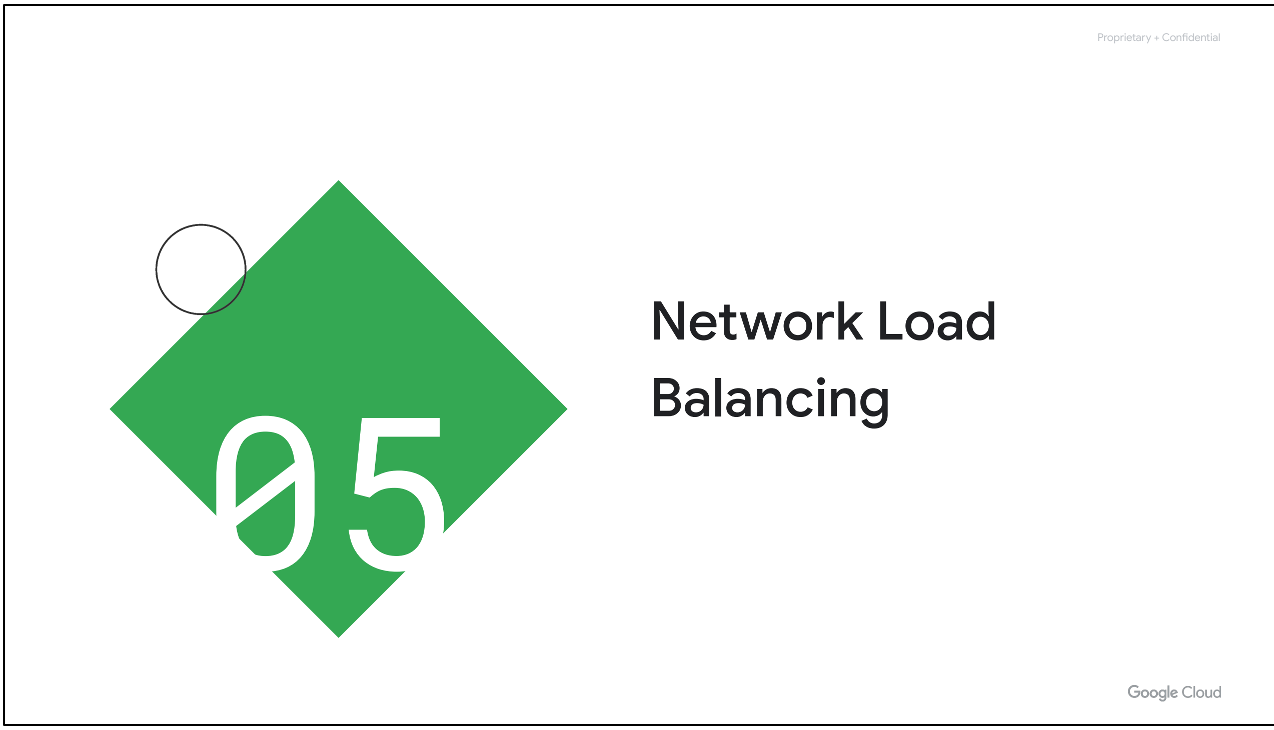
Example: TCP proxy load balancing



Google Cloud

This network diagram illustrates TCP proxy load balancing. In this example, traffic from users in Iowa and Boston is terminated at the global load balancing layer. From there, a separate connection is established to the closest backend instance. As in the SSL proxy load balancing example, 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.

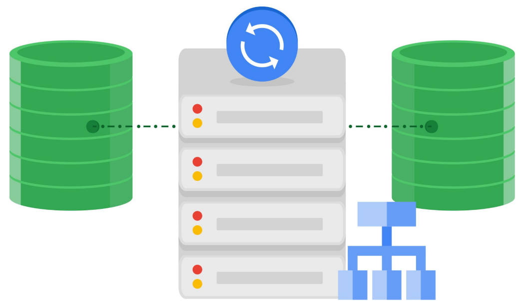
Now, the traffic between the proxy and the backends can use SSL or TCP, and I also recommend using SSL here.



Next, let’s talk about network load balancing, which is a regional load balancing service.

Network load balancing

- Regional, non-proxied load balancer
- Forwarding rules (IP protocol data)
- Traffic:
  - UDP
  - TCP/SSL ports
- Architecture:
  - Backend service-based
  - Target pool-based



Network load balancing is a regional, non-proxied load balancing service. In other words, all traffic is passed through the load balancer, instead of being proxied, and traffic can only be balanced between VM instances that are in the same region, unlike a global load balancer.

This load balancing service uses forwarding rules to balance the load on your systems based on incoming IP protocol data, such as address, port, and protocol type. You can use it to load balance UDP traffic and to load balance TCP and SSL traffic on ports that are not supported by the TCP proxy and SSL proxy load balancers.

The architecture of a network load balancer depends on whether you use a backend service-based network load balancer or a target pool-based network load balancer. Let's explore these in more detail.

Proprietary + Confidential

Backend service-based architecture

- Regional backend service
- Defines the behavior of the load balancer and how it distributes traffic to its backend instance groups
- Enables new features not supported with legacy target pools
  - Non-legacy health checks
  - Auto-scaling with managed instance groups
  - Connection draining
  - Configurable failover policy

Google Cloud

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 services enable new features that are not supported with legacy target pools, such as support for non-legacy health checks (TCP, SSL, HTTP, HTTPS, or HTTP/2), auto-scaling with managed instance groups, connection draining, and a configurable failover policy.

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?

Proprietary + Confidential

Target pool-based architecture

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

Google Cloud

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.

06

Internal Load Balancing

Next, let's talk about internal load balancing.

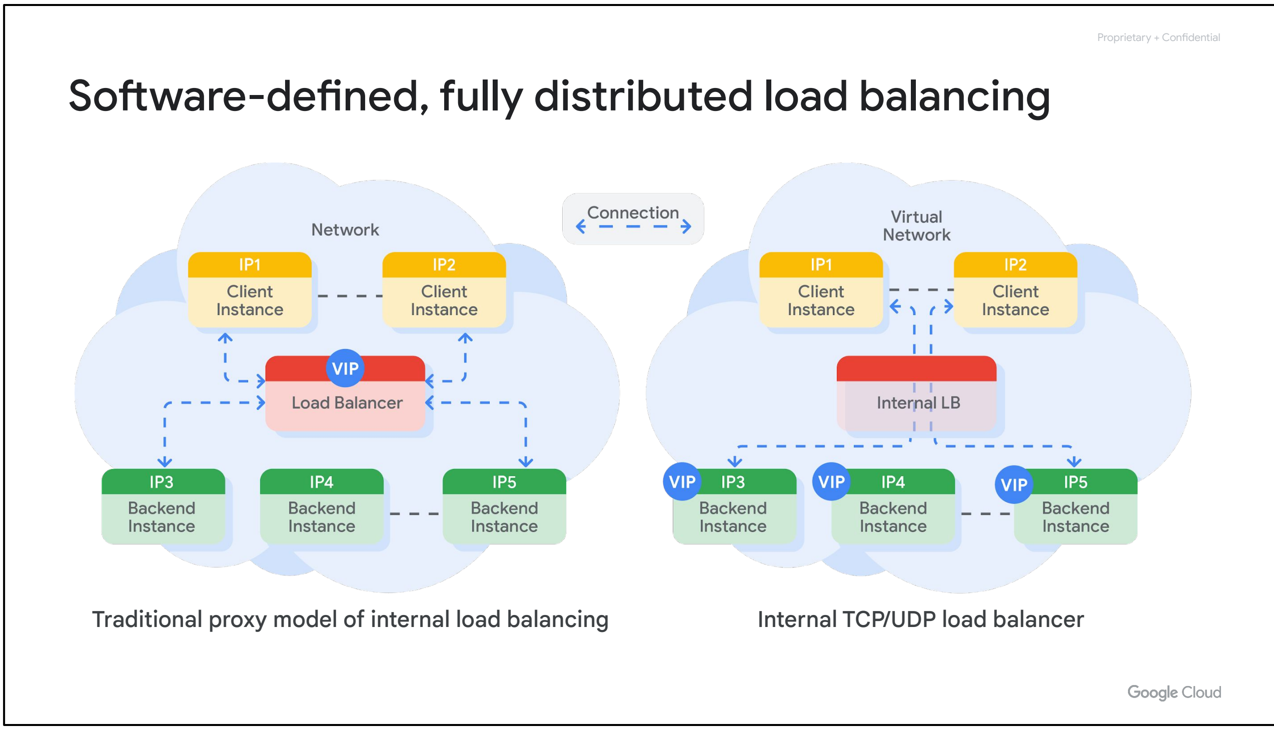
Internal TCP/UDP load balancing

- Regional, private load balancing
  - VM instances in same region
  - RFC 1918 IP addresses
- TCP/UDP traffic
- Reduced latency, simpler configuration
- Software-defined, fully distributed load balancing

The internal TCP/UDP load balancer is a regional, private load balancing service for TCP- and UDP-based traffic. 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 TCP/UDP load balancing 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 TCP/UDP load balancing service.



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.

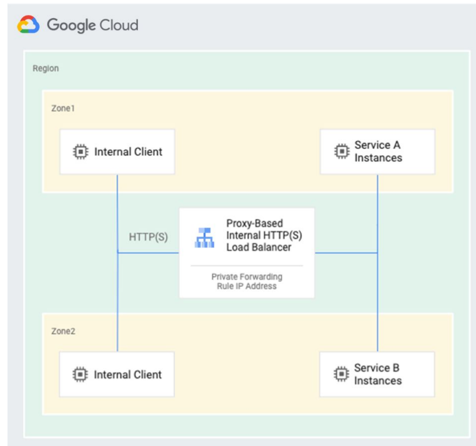
In the traditional proxy model of internal load balancing, as shown on the left, you configure an internal IP address on a load balancing device or instances, and your client instance connects to this IP address. Traffic coming to the IP address is terminated at the load balancer, and the load balancer 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 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, see the links section of this video. [https://cloudplatform.googleblog.com/2014/04/enter-andromeda-zone-google-cloud-platforms-latest-networking-stack.html]

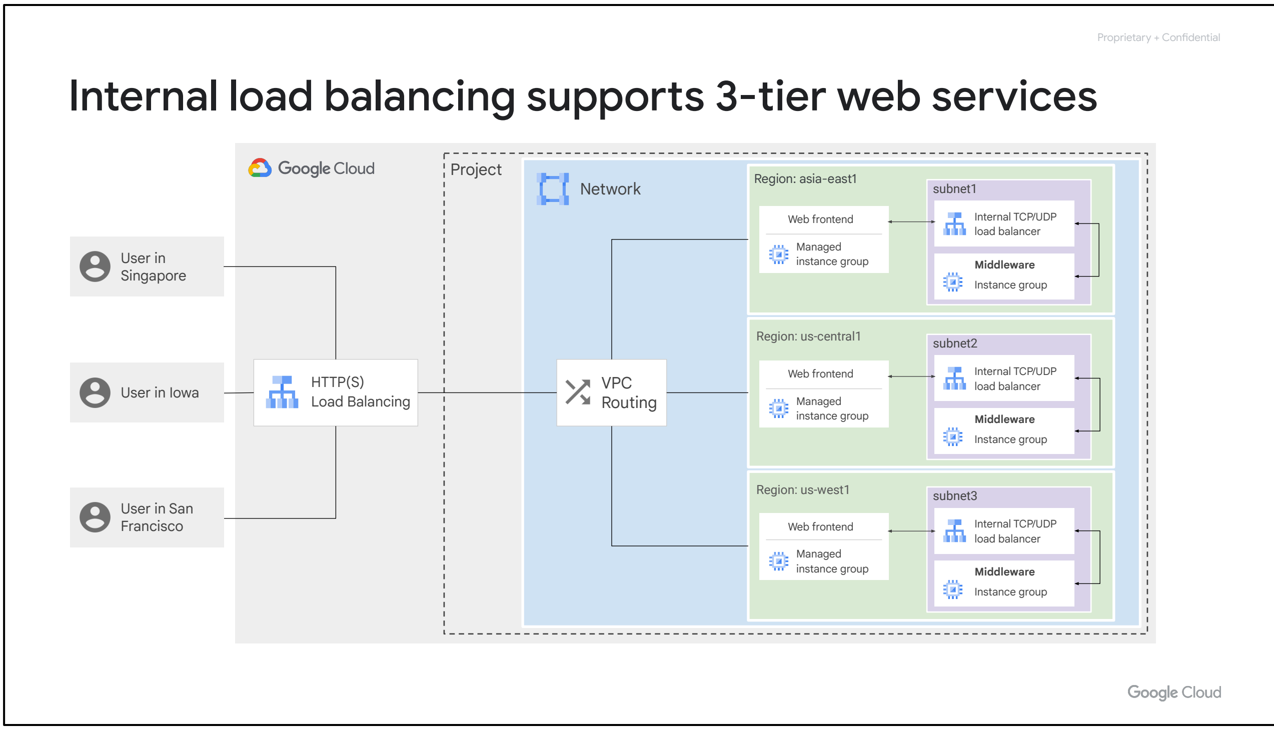
Internal HTTP(S) load balancing

- Regional, private load balancing
  - VM instances in same region
  - RFC 1918 IP addresses
- HTTP, HTTPS, or HTTP/2 protocols
- Based on open source Envoy proxy



Now, Google Cloud Internal HTTP(S) Load Balancing is a proxy-based, regional Layer 7 load balancer that also enables you to run and scale your services behind an internal load balancing IP address. Backend services support the HTTP, HTTPS, or HTTP/2 protocols.

Internal HTTP(S) Load Balancing is a managed service based on the open source Envoy proxy. This enables rich traffic control capabilities based on HTTP(S) parameters. After the load balancer has been configured, it automatically allocates Envoy proxies to meet your traffic needs.

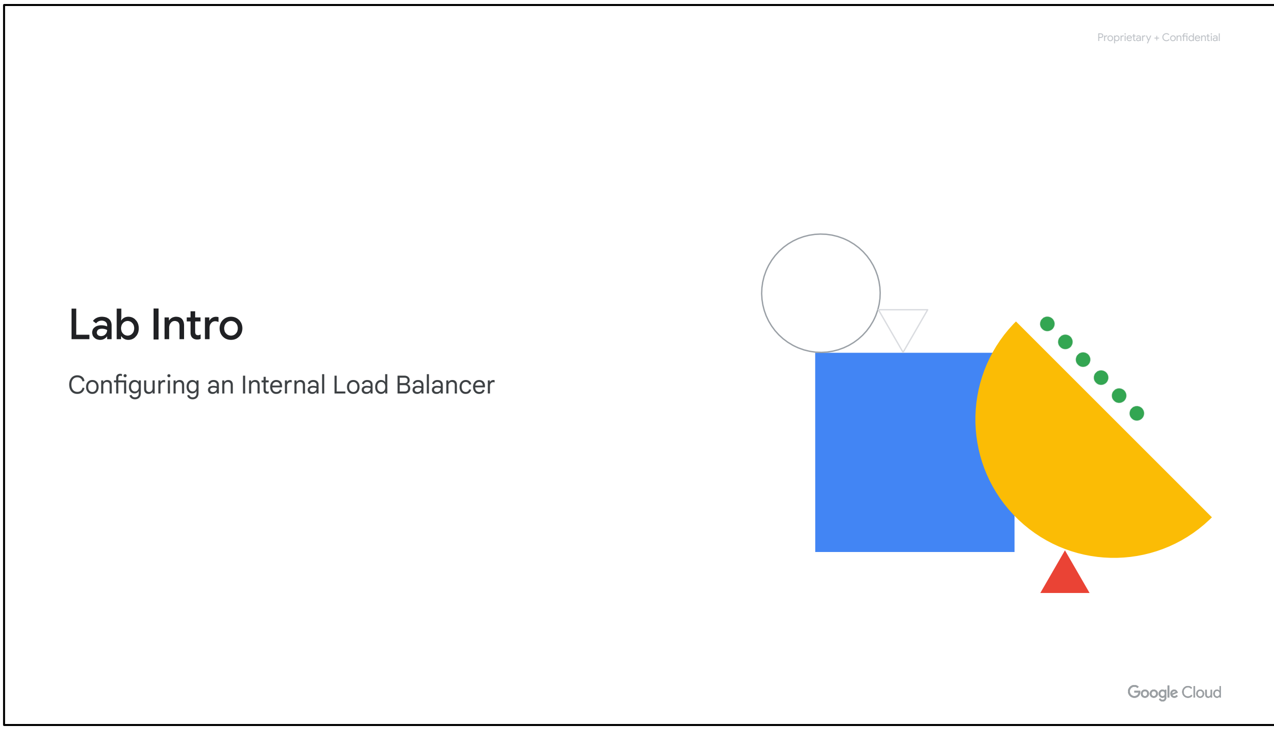


Now, internal load balancing enables you to support use cases such as the traditional 3-tier web services.

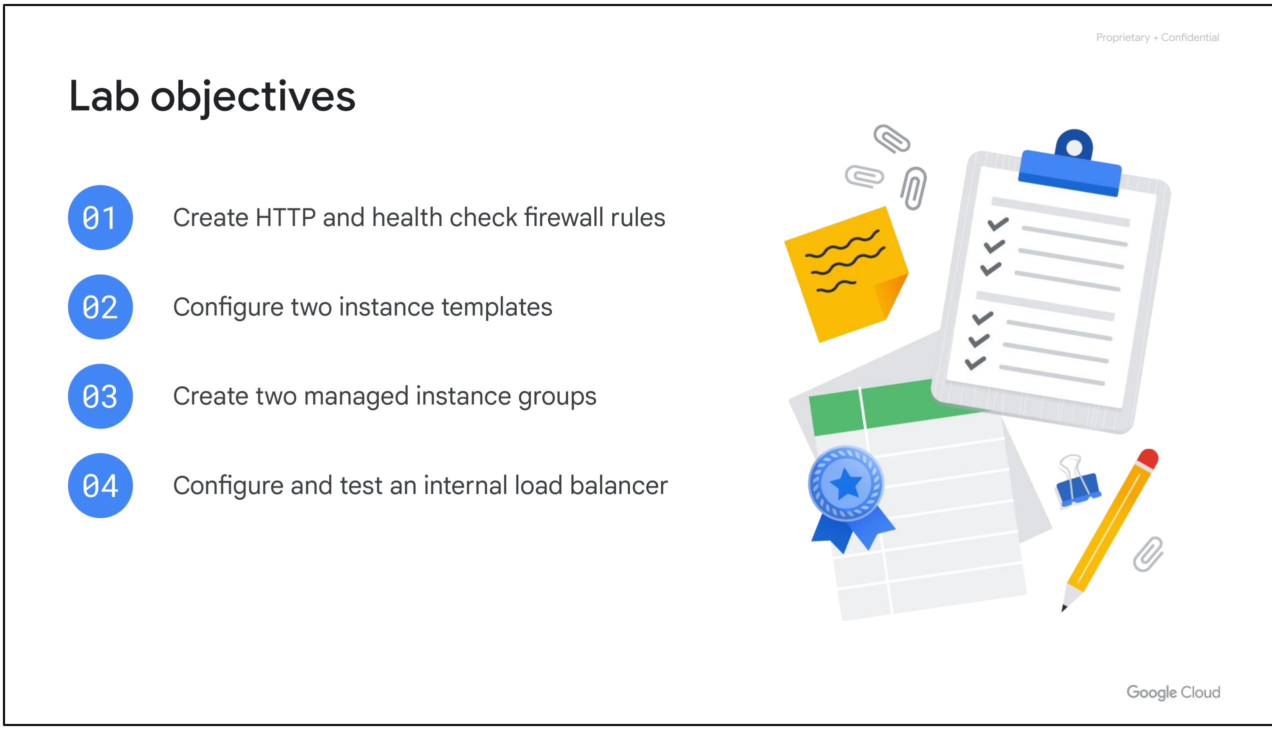
In this example, the web tier uses an external HTTP(S) load balancer that provides a single global IP address for users in San Francisco, Iowa, Singapore, and so on. The backends of this load balancer are located in the us-west1, us-central1 and asia-east1 regions because this is a global load balancer.

These backends then access an internal 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.

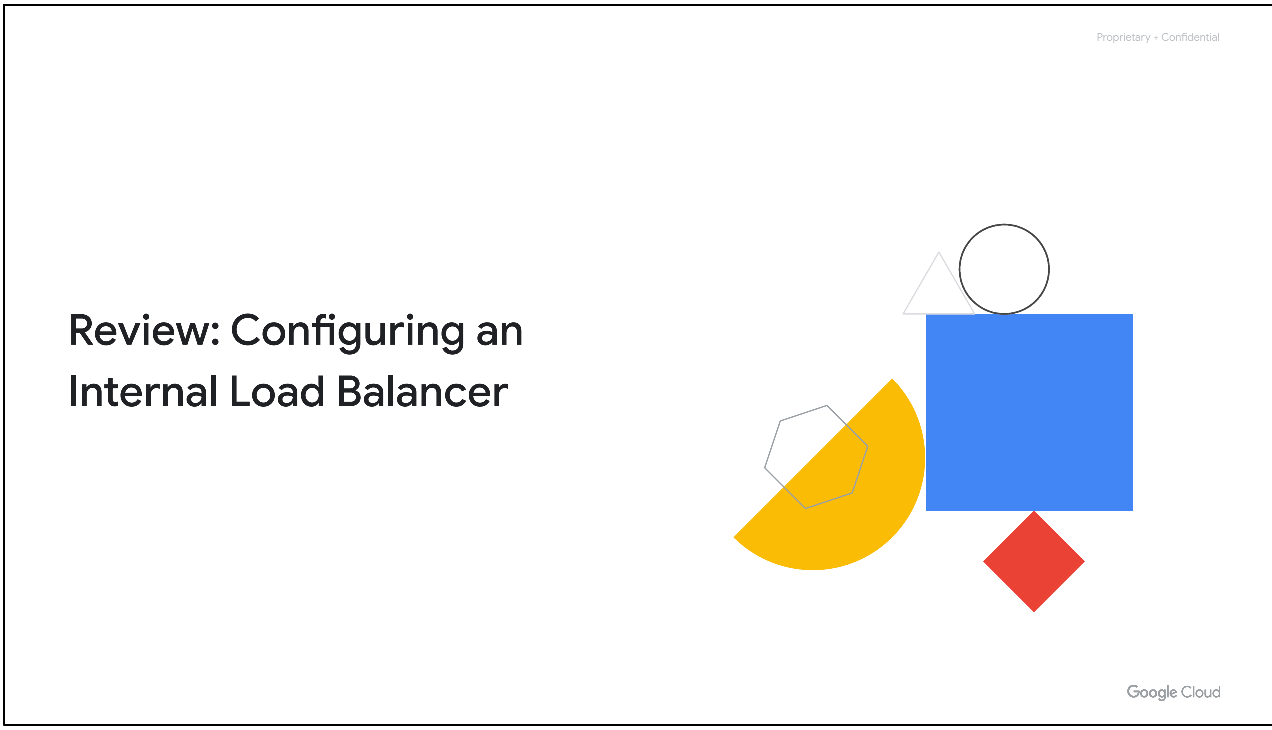
The benefit of this 3-tier approach is that neither the database tier nor the application tier is exposed externally. This simplifies security and network pricing.



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

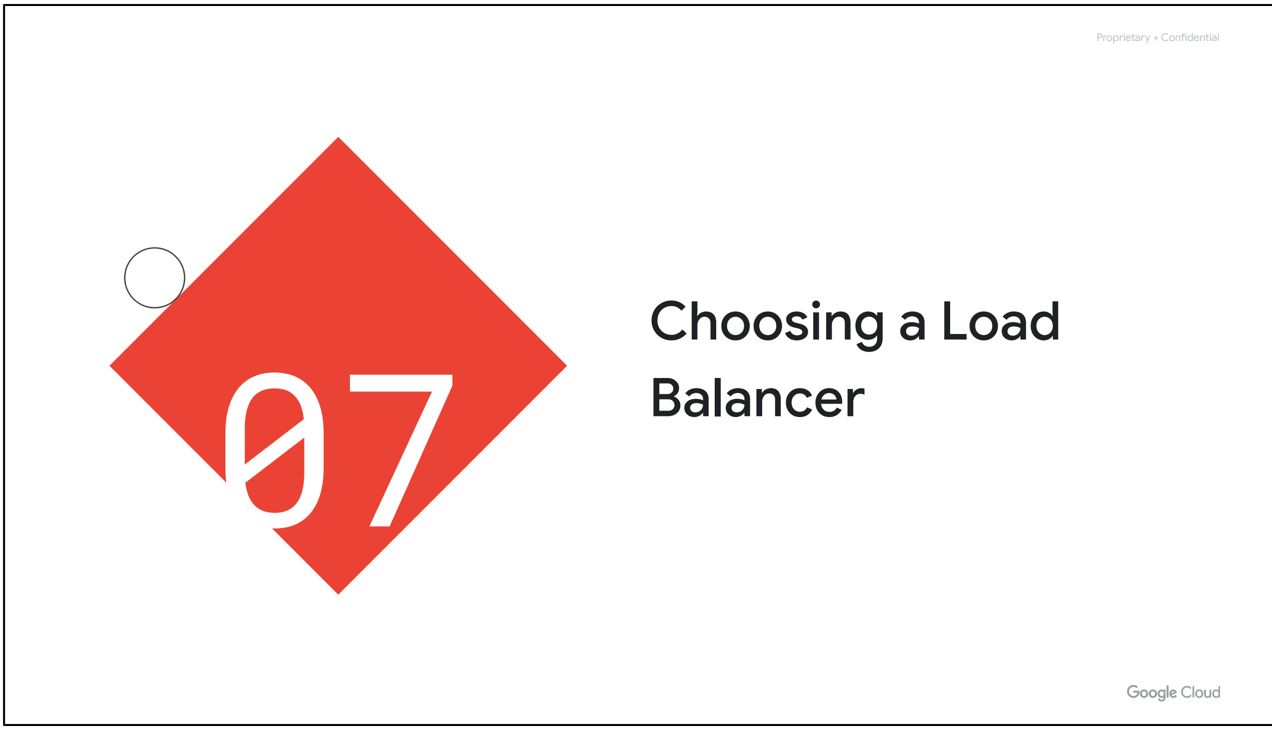


In this lab, you create two managed instance groups in the same region. Then, you configure and test an internal load balancer with the instances groups as the backends, as shown in this network diagram.



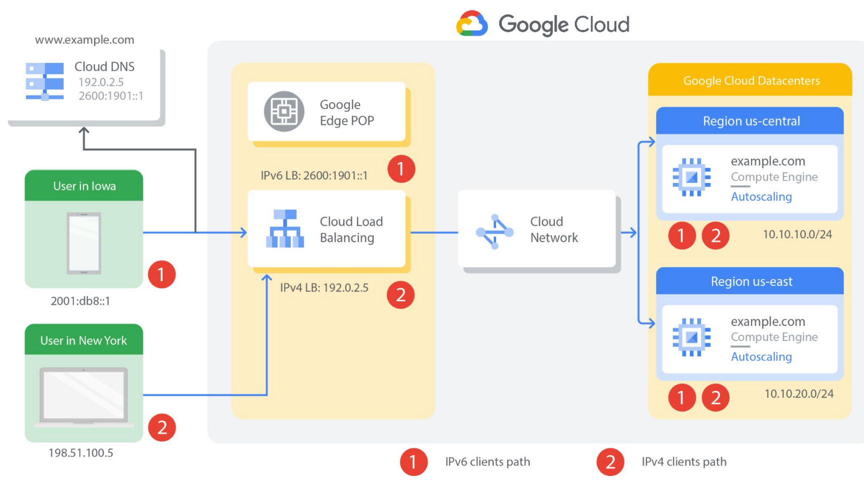
In this lab, you created two managed instance groups in the us-central1 region, along with firewall rules to allow HTTP traffic to those instances and TCP traffic from the GCP health checker. Then, you configured and tested an internal load balancer for those instance groups.

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



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.

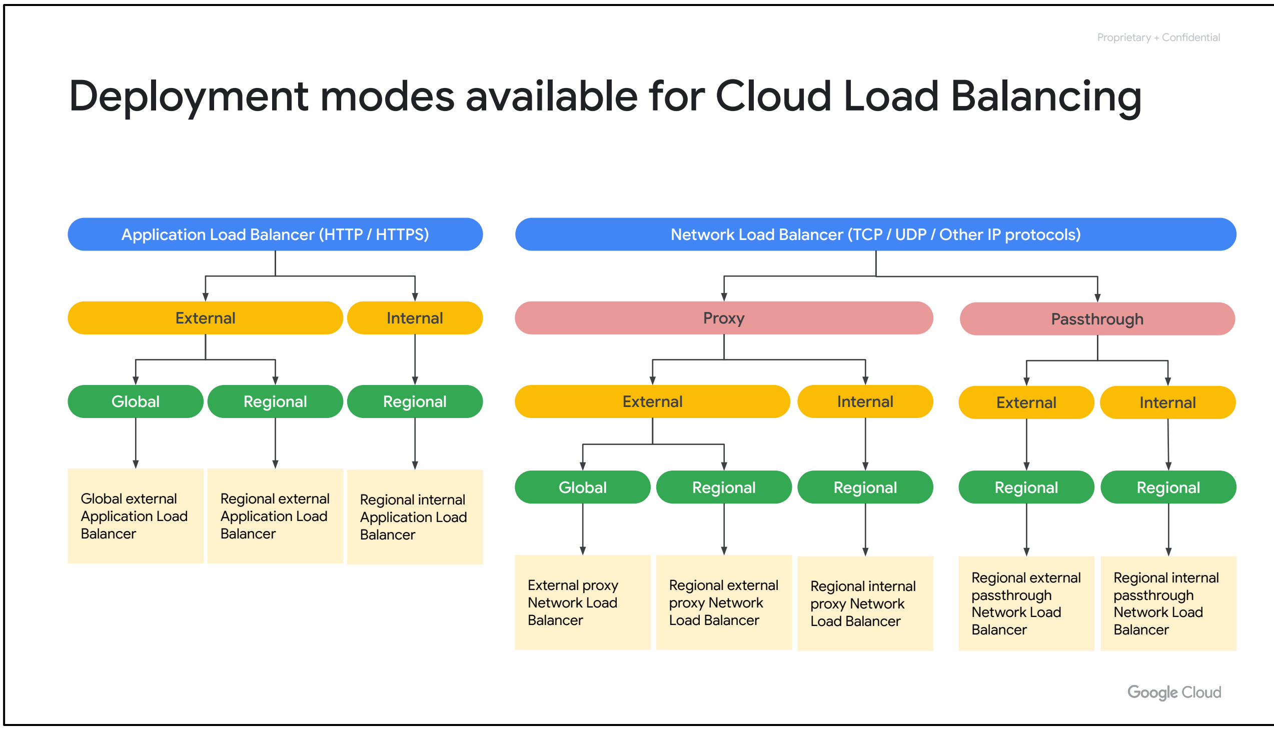
IPv6 termination for load balancing



One differentiator between the different Google Cloud load balancers is the support for IPv6 clients. Only the HTTP(S), SSL proxy, and TCP proxy load balancing services support IPv6 clients. IPv6 termination for these load balancers enables you to handle IPv6 requests from your users and proxy them over IPv4 to your backends.

For example, in this diagram there is a website www.example.com that is translated by Cloud DNS to both an IPv4 and IPv6 address. This allows a desktop user in New York and a mobile user in Iowa to access the load balancer through the IPv4 and IPv6 addresses, respectively. But how does the traffic get to the backends and their IPv4 addresses?

Well, the load balancer acts as a reverse proxy, terminates the IPv6 client connection, and places the request into an IPv4 connection to a backend. On the reverse path, the load balancer receives the IPv4 response from the backend and places it into the IPv6 connection back to the original client. In other words, configuring IPv6 termination for your load balancers lets your backend instances appear as IPv6 applications to your IPv6 clients.



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

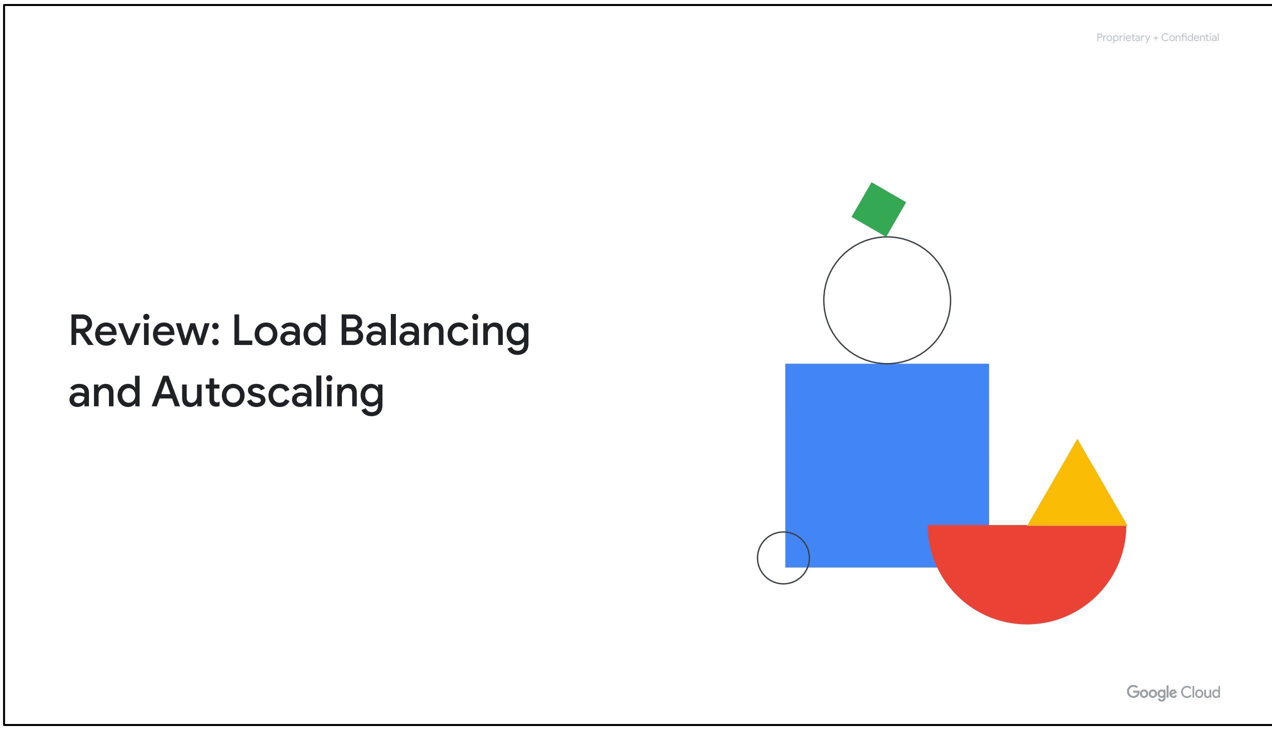
Google Cloud

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.



In this module, we looked at the different types of load balancers that are available in GCP, 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.