

Networking Products

Guides

Reference

Support

Resources

[Contact Sales](#)
[Get started for free](#)

Cloud Load Balancing

[Documentation](#)
[Product overview](#)

How-to guides

[All how-to guides](#)

- [Setting up your load balancer](#)
- [Advanced settings, monitoring, logging, and troubleshooting](#)

Concepts

[All concepts](#)
[Load balancing overview](#)
[Choosing a load balancer](#)
[Load balancer features](#)

- [Access control](#)
- [Internal HTTP\(S\) Load Balancing](#)
- ▾ [External HTTP\(S\) Load Balancing](#)
 - [Overview](#)
 - [Traffic management overview](#)
- [Internal TCP/UDP Load Balancing](#)
- [External TCP/UDP Network Load Balancing](#)
- [External SSL Proxy and TCP Proxy Load Balancing](#)
- [Network endpoint groups](#)
- [Load balancing components](#)

[Networking Products](#) > [Load Balancing](#) > [Documentation](#)

[Send feedback](#)

External HTTP(S) Load Balancing overview

Contents ▾

Use cases

- [Load balancing using multiple backend types](#)
- [Three-tier web services](#)
- [Cross-region load balancing](#)
- [Content-based load balancing](#)

...

This document introduces the concepts that you need to understand to configure Google Cloud external HTTP(S) Load Balancing.

For information about how the Google Cloud load balancers differ from each other, see the following documents:

- [Load balancing overview](#)
- [Choosing a load balancer](#)
- [Load balancer features](#)

Use cases

The external HTTP(S) load balancers address many use cases. This section provides some high-level examples.

Load balancing using multiple backend types

External HTTP(S) Load Balancing supports the following backend types:

- [Instance groups](#)
- [Zonal network endpoint groups \(NEGs\)](#)
- [Serverless NEGs](#): One or more [App Engine](#), [Cloud Run](#), or [Cloud Functions](#) services
- [Internet NEGs](#), for endpoints that are outside of Google Cloud (also known as custom origins)
- Buckets in [Cloud Storage](#)

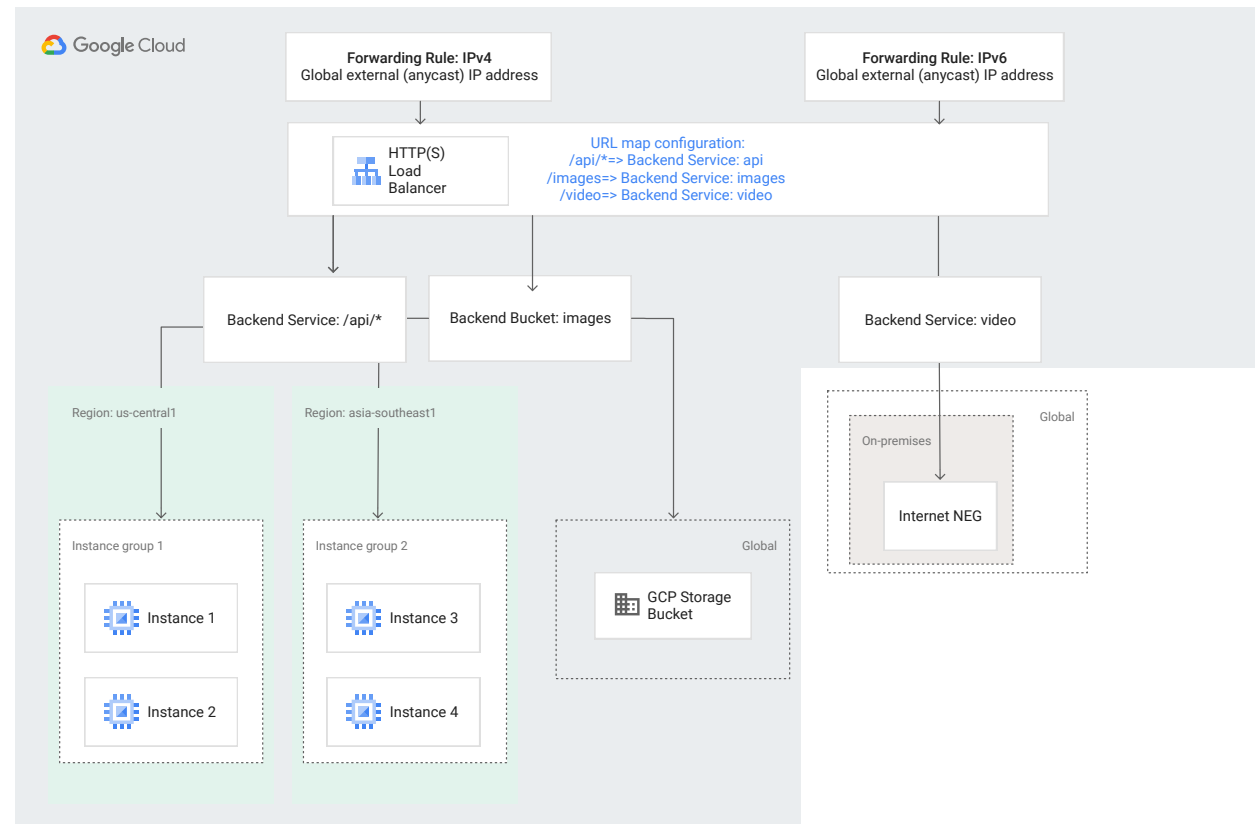
One common use case is load balancing traffic among services. In the following example, external IPv4 and IPv6 clients can request video, API, and image content by using the same base URL, with the paths `/api`, `/video`, and `/images`.

The external HTTP(S) load balancer's URL map specifies that:

- Requests to path `/api` go to a backend service with a VM [instance group](#) or a [zonal NEG](#) backend.
- Requests to path `/images` go to a [backend bucket](#) with a Cloud Storage backend.
- Requests to path `/video` go to a backend service that points to a [internet NEG](#) containing an external endpoint that is located on-premises outside of Google Cloud.

When a client sends a request to the load balancer's external IPv4 or IPv6 address, the load balancer evaluates the request according to the URL map and sends the request to the correct service.

The following diagram illustrates this use case.



Load balancing diagram with a custom origin (click to enlarge)

On each backend service, you can optionally enable Cloud CDN and Google Cloud Armor. If you are using Google Cloud Armor with Cloud CDN, security policies are enforced only for requests for dynamic content, cache misses, or other requests that are destined for the CDN origin server. Cache hits are served even if the downstream Google Cloud Armor security policy would prevent that request from reaching the CDN origin server.

On backend buckets, Cloud CDN is supported, but not Google Cloud Armor.

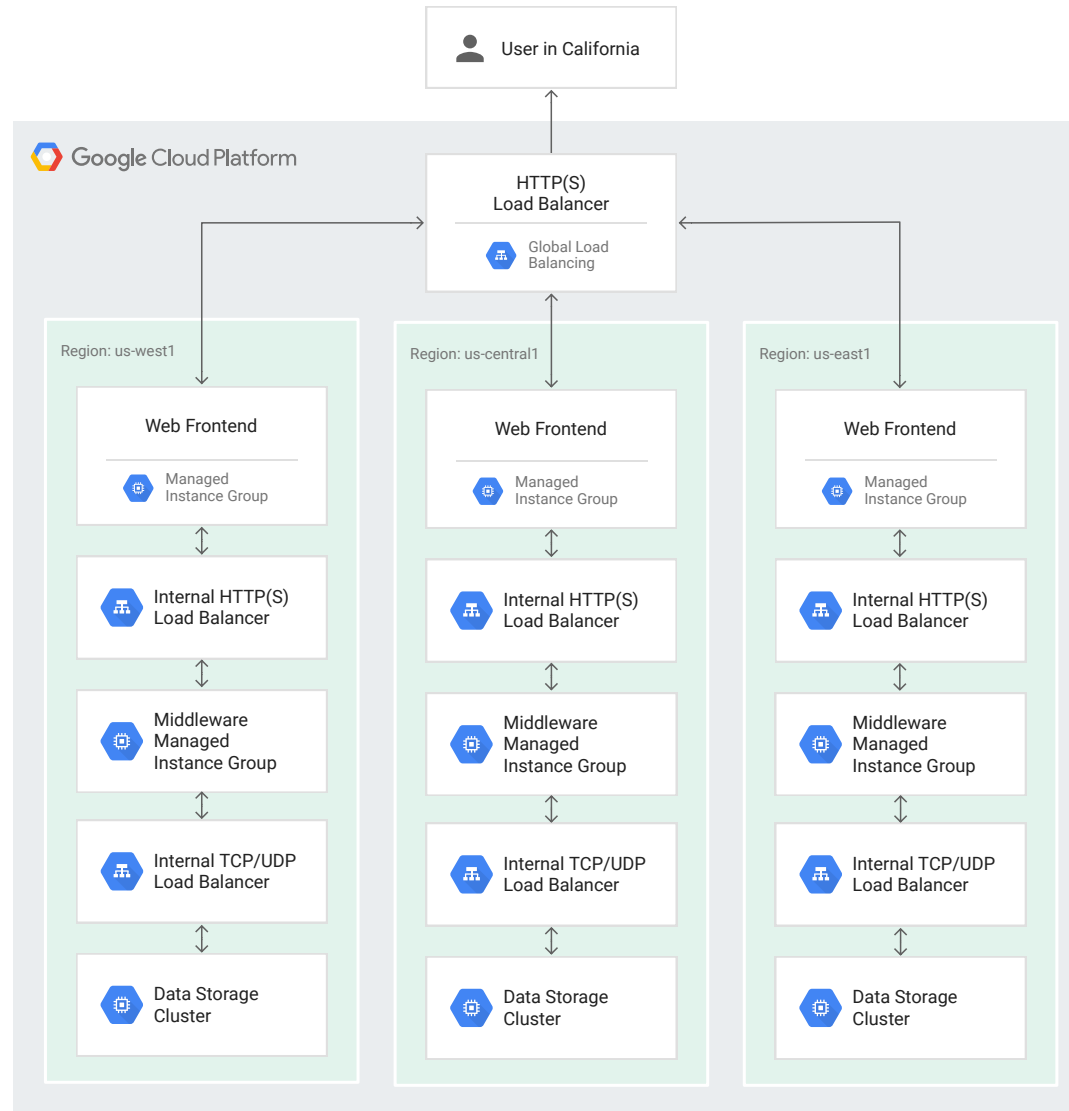
Three-tier web services

You can use external HTTP(S) Load Balancing to support traditional three-tier web services. The following example shows how you can use three types of Google Cloud load balancers to scale three tiers. At each tier, the load balancer type depends on your traffic type:

- **Web tier:** Traffic enters from the internet and is load balanced by using an [external HTTP\(S\) load balancer](#).
- **Application tier:** The application tier is scaled by using a regional internal HTTP(S) load balancer.
- **Database tier:** The database tier is scaled by using an [internal TCP/UDP load balancer](#).

The diagram shows how traffic moves through the tiers:

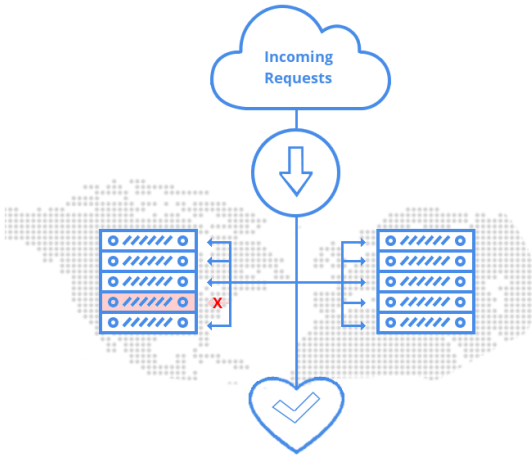
1. An external HTTP(S) load balancer (the subject of this overview) distributes traffic from the internet to a set of web frontend instance groups in various regions.
2. These frontends send the HTTP(S) traffic to a set of regional, internal HTTP(S) load balancers.
3. The internal HTTP(S) load balancers distribute the traffic to middleware instance groups.
4. These middleware instance groups send the traffic to internal TCP/UDP load balancers, which load balance the traffic to data storage clusters.



Layer 7-based routing for internal tiers in a multi-tier app

Cross-region load balancing

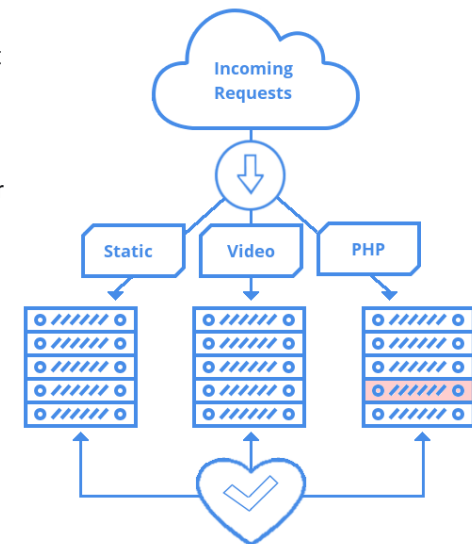
When you configure an external HTTP(S) load balancer in Premium Tier, it uses a global external IP address and can intelligently route requests from users to the closest backend instance group or NEG, based on proximity. For example, if you set up instance groups in North America, Europe, and Asia, and attach them to a load balancer's backend service, user requests around the world are automatically sent to the VMs closest to the users, assuming the VMs pass health checks and have enough capacity (defined by the balancing mode). If the closest VMs are all unhealthy, or if the closest instance group is at capacity and another instance group is not at capacity, the load balancer automatically sends requests to the next closest region with capacity.



Content-based load balancing

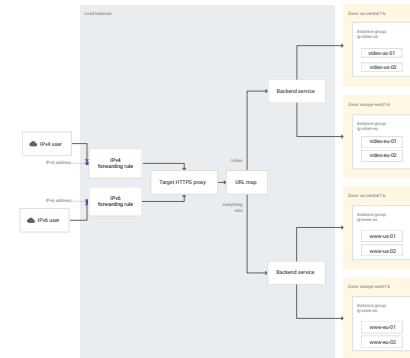
HTTP(S) Load Balancing supports content-based load balancing using URL maps to select a backend service based on the requested host name, request path, or both. For example, you can use a set of instance groups or NEGs to handle your video content and another set to handle everything else.

You can also use HTTP(S) Load Balancing with [Cloud Storage buckets](#). After you have your load balancer set up, you can [add Cloud Storage buckets](#) to it.



Creating a combined load balancer

You can configure an external HTTP(S) load balancer in Premium Tier to provide both content-based and cross-region load balancing, using multiple backend services, each with backend instance groups or NEGs in multiple regions. You can combine and extend the [use cases](#) to configure an external HTTP(S) load balancer that meets your needs.



Example configuration

If you want to jump right in and build a working load balancer for testing, see [Setting up a simple external HTTP load balancer](#) or [Setting up a simple external HTTPS load balancer](#).

For a more complex example that uses content-based and cross-region load balancing, see [Creating an HTTPS load balancer](#).

How connections work in HTTP(S) Load Balancing

External HTTP(S) Load Balancing is a service, implemented by many proxies called Google Front Ends (GFEs). There isn't just a single proxy. In Premium Tier, the same global external IP address is advertised from various points of presence, and traffic is directed to the client's nearest GFE.

Depending on where your clients are, multiple GFEs can initiate HTTP(S) connections to your backends. Packets sent from GFEs have source IP addresses from the same range used by health check probes: `35.191.0.0/16` and `130.211.0.0/22`.

Depending on the backend service configuration, the protocol used by each GFE to connect to your backends can be HTTP, HTTPS, or HTTP/2. If HTTP or HTTPS, the HTTP version is HTTP 1.1. HTTP keepalive is enabled by default, as specified in the HTTP 1.1 specification. The GFE uses a keepalive timeout of 600 seconds, and you cannot configure this.

You can, however, configure the request/response timeout by setting the backend service timeout. For more information, see [timeouts and retries](#).

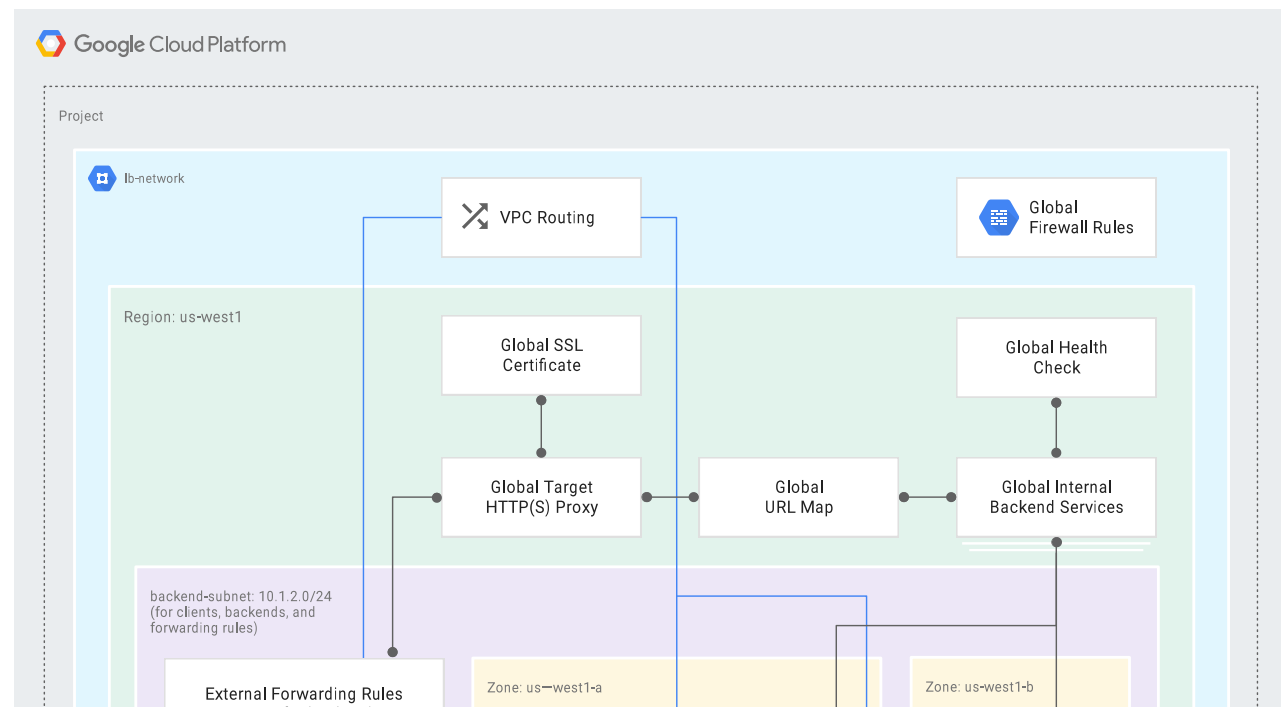
HTTP keepalives attempt to efficiently use the same TCP session; however, there's no guarantee. Though closely related, an HTTP keepalive and a TCP idle timeout are not the same thing.

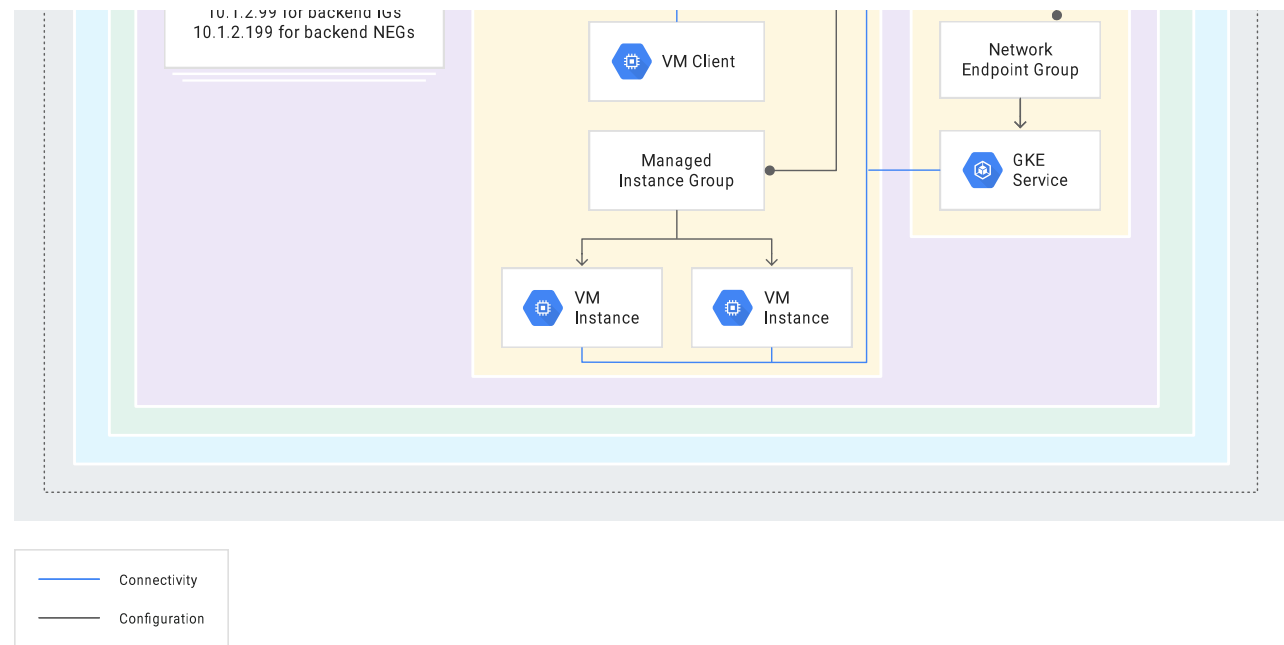
The numbers of HTTP connections and TCP sessions vary depending on the number of GFEs connecting, the number of clients connecting to the GFEs, the protocol to the backends, and where backends are deployed.

For more information, see [How HTTP\(S\) Load Balancing works](#) in the solutions guide: Application Capacity Optimizations with Global Load Balancing.

Architecture and resources

The following diagram shows the Google Cloud resources required for an external HTTP(S) load balancer.





HTTP(S) Load Balancing components

The following resources define an external HTTP(S) load balancer:

- An **external forwarding rule** specifies an external IP address, port, and global target HTTP(S) proxy. Clients use the IP address and port to connect to the load balancer.
- A **global 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.
- If you are using HTTPS load balancing, the target HTTPS proxy uses **global SSL certificates** to prove its identity to clients. A target HTTPS proxy supports up to a [documented number](#) of SSL certificates.
- The HTTP(S) proxy uses a **global URL map** to make a routing determination based on HTTP attributes (such as the request path, cookies, or headers). Based on the routing decision, the proxy forwards client requests to specific backend services or backend buckets. The URL map can specify additional actions, such as sending redirects to clients.
- A **backend service** or **backend bucket** distributes requests to healthy [backends](#).

- One or more **backends** must be connected to the backend service or backend bucket. Backends can be instance groups, NEGs or buckets in any of the following configurations:

- Managed instance groups (zonal or regional)
- Unmanaged instance groups (zonal)
- Network endpoint groups (zonal)
- Network endpoint groups (internet)
- Cloud Storage buckets

You cannot have instance groups and NEGs on the same backend service.

- A **global health check** periodically monitors the readiness of your backends. This reduces the risk that requests might be sent to backends that can't service the request.
- A **firewall** for your backends to accept health check probes.

Source IP addresses

The source IP addresses for packets, as seen by each backend virtual machine (VM) instance or container, is an IP address from these ranges:

- 35.191.0.0/16
- 130.211.0.0/22

The source IP address for actual load-balanced traffic is the same as the [health checks probe IP ranges](#).

The source IP addresses for traffic, as seen by the backends, is *not* the Google Cloud external IP address of the load balancer. In other words, there are two HTTP, SSL, or TCP sessions:

- Session 1, from original client to the load balancer (GFE):
 - **Source IP address:** the original client (or external IP address if the client is behind NAT).
 - **Destination IP address:** your load balancer's IP address.
- Session 2, from the load balancer (GFE) to the backend VM or container:
 - **Source IP address:** an IP address in one of these ranges: 35.191.0.0/16 or 130.211.0.0/22.

You cannot predict the actual source address.

- **Destination IP address:** the internal IP address of the backend VM or container in the Virtual Private Cloud (VPC) network.

★ **Important:** If you block *any* of the IP addresses in `35.191.0.0/16` or `130.211.0.0/22`, health checks might still pass. Multiple probes are used simultaneously. A few probes that fail can be overridden if the majority pass. However, this can block actual load-balanced traffic.

Client communications with the load balancer

- Clients can communicate with the load balancer by using the HTTP 1.1 or HTTP/2 protocol.
- When HTTPS is used, modern clients default to HTTP/2. This is controlled *on the client*, not on the HTTPS load balancer.
- You cannot disable HTTP/2 by making a configuration change on the load balancer. However, you can configure some clients to use HTTP 1.1 instead of HTTP/2. For example, with `curl`, use the `--http1.1` parameter.
- HTTPS load balancers do not support client certificate-based authentication, also known as mutual TLS authentication.

Open ports

The external HTTP(S) load balancers are reverse proxy load balancers. The load balancer terminates incoming connections, and then opens new connections from the load balancer to the backends. The reverse proxy functionality is provided by the Google Front Ends (GFEs).

The [firewall rules](#) that you set block traffic from the GFEs to the backends, but do not block incoming traffic to the GFEs.

The external HTTP(S) load balancers have a number of open ports to support other Google services that run on the same architecture. If you run a security or port scan against the external IP address of a Google Cloud external HTTP(S) load balancer, additional ports appear to be open.

This does not affect external HTTP(S) load balancers. External forwarding rules, which are used in the definition of an external HTTP(S) load balancer, can only reference TCP ports 80, 8080, and 443. Traffic with a different TCP destination port is not forwarded to the load balancer's backend.

Components

The following are components of external HTTP(S) load balancers.

Forwarding rules and addresses

[Forwarding rules](#) route traffic by IP address, port, and protocol to a load balancing configuration consisting of a target proxy, URL map, and one or more backend services.

Each forwarding rule provides a single IP address that can be used in DNS records for your application. No DNS-based load balancing is required. You can either specify the IP address to be used or let Cloud Load Balancing assign one for you.

- The forwarding rule for an HTTP load balancer can only reference TCP ports 80 and 8080.
- The forwarding rule for an HTTPS load balancer can only reference TCP port 443.

The type of forwarding rule required by external HTTP(S) load balancers depends on which [Network Service Tier](#) the load balancer is in.

- The external HTTP(S) load balancers in the Premium Tier use global external forwarding rules.
- The external HTTP(S) load balancers in the Standard Tier use regional external forwarding rules.

Target proxies

[Target proxies](#) terminate HTTP(S) connections from clients. One or more forwarding rules direct traffic to the target proxy, and the target proxy consults the URL map to determine how to route traffic to backends.

The proxies set HTTP request/response headers as follows:

- `Via: 1.1 google` (requests and responses)
- `X-Forwarded-Proto: [http | https]` (requests only)
- `X-Cloud-Trace-Context: <trace-id>/<span-id>;<trace-options>` (requests only)
Contains parameters for [Cloud Trace](#).

You can create custom request headers if the default headers do not meet your needs. For more information about this feature, see [Creating user-defined request headers](#).

Do not rely on the proxy to preserve the case of request or response header names. For example, a `Server: Apache/1.0` response header may appear at the client as `server: Apache/1.0 ..`

Host header

When the load balancer makes the HTTP request, the load balancer preserves the Host header of the original request.

X-Forwarded-For header

The load balancer appends two IP addresses to the X-Forwarded-For header:

- The IP address of the client that connects to the load balancer
- The load balancer's IP address

If there is no X-Forwarded-For header on the incoming request, these two IP addresses are the entire header value. If the request does have an X-Forwarded-For header, other information, such as the IP addresses recorded by proxies on the way to the load balancer, are preserved before the two IP addresses. The load balancer does not verify any IP addresses that precede the last two IP addresses in this header.

If you are running a proxy on your backend instance, this proxy typically appends more information to the X-Forwarded-For header, and your software might need to take that into account. The proxied requests from the load balancer come from an IP address in the range `130.211.0.0/22` or `35.191.0.0/16`, and your proxy on the backend instance might record this address as well as the backend instance's own IP address.

URL maps

[URL maps](#) define matching patterns for URL-based routing of requests to the appropriate backend services. A default service is defined to handle any requests that do not match a specified host rule or path matching rule. In some situations, such as the [cross-region load balancing example](#), you might not define any URL rules and rely only on the default service. For content-based routing of traffic, the URL map allows you to divide your traffic by examining the URL components to send requests to different sets of backends.

SSL certificates

If you are using HTTPS-based load balancing, you must install one or more [SSL certificates](#) on the target HTTPS proxy.

These certificates are used by target HTTPS proxies to secure communications between the load balancer and the client.

For information about SSL certificate limits and quotas, see [SSL certificates](#) on the load balancing quotas page.

For the best security, use end-to-end encryption for your HTTPS load balancer deployment. For more information, see [Encryption from the load balancer to the backends](#).

For general information about how Google encrypts user traffic, see the [Encryption in Transit in Google Cloud](#) white paper.

SSL policies

[SSL policies](#) give you the ability to control the features of SSL that your HTTPS load balancer negotiates with HTTPS clients.

By default, HTTPS Load Balancing uses a set of SSL features that provides good security and wide compatibility. Some applications require more control over which SSL versions and ciphers are used for their HTTPS or SSL connections. You can define SSL policies that control the features of SSL that your load balancer negotiates and associate an SSL policy with your target HTTPS proxy.

Geographic control over where TLS is terminated

The HTTPS load balancer terminates TLS in locations that are distributed globally, so as to minimize latency between clients and the load balancer. If you require geographic control over where TLS is terminated, you should use Google Cloud [Network Load Balancing](#) instead, and terminate TLS on backends that are located in regions appropriate to your needs.

Backend services

[Backend services](#) provide configuration information to the load balancer. An external HTTP(S) load balancer must have at least one backend service and can have multiple backend services.

Load balancers use the information in a backend service to direct incoming traffic to one or more attached backends.

The backends of a backend service can be either [instance groups](#) or [network endpoint groups \(NEGs\)](#), but not a combination of both. When you add a backend instance group or NEG, you specify a *balancing mode*, which defines a method for distributing requests and a target capacity. For more information, see [Load distribution algorithm](#).

HTTP(S) Load Balancing supports [Cloud Load Balancing Autoscaler](#), which allows users to perform autoscaling on the instance groups in a backend service. For more information, see [Scaling based on HTTP\(S\) Load Balancing serving capacity](#).

You can enable connection draining on backend services to ensure minimal interruption to your users when an instance that is serving traffic is terminated, removed manually, or removed by an autoscaler. To learn more, see [Enabling connection draining](#).

Changes to a backend service associated with an external HTTP(S) load balancer are not instantaneous. It can take several minutes for changes to propagate throughout the network.

Behavior of the load balancer in different Network Service Tiers

HTTP(S) Load Balancing is a global service when the Premium [Network Service Tier](#) is used. You may have more than one backend service in a region, and you may create backend services in more than one region, all serviced by the same global load balancer. Traffic is allocated to backend services as follows:

1. When a user request comes in, the load balancing service determines the approximate origin of the request from the source IP address.
2. The load balancing service knows the locations of the instances owned by the backend service, their overall capacity, and their overall current usage.
3. If the closest instances to the user have available capacity, the request is forwarded to that closest set of instances.
4. Incoming requests to the given region are distributed evenly across all available backend services and instances in that region. However, at very small loads, the distribution may appear to be uneven.
5. 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.

HTTP(S) Load Balancing is a regional service when the Standard Network Service Tier is used. Its backend instance groups or NEGs must all be located in the region used by the load balancer's external IP address and forwarding rule.

Health checks

Each backend service also specifies which [health check](#) is performed against each available instance. For the health check probes to function correctly, you must create a firewall rule that allows traffic from `130.211.0.0/22` and `35.191.0.0/16` to reach your instances.

For more information about health checks, see [Creating health checks](#).


Protocol to the backends

When you configure a backend service for the external HTTP(S) load balancer, you set the protocol that the backend service uses to communicate with the backends. You can choose HTTP, HTTPS, or HTTP/2. The load balancer uses only the protocol that you specify. The load balancer does not fall back to one of the other protocols if it is unable to negotiate a connection to the backend with the specified protocol.

If you use HTTP/2, you must use TLS. HTTP/2 without encryption is not supported.

Although it is not required, it is a best practice to use a health check whose protocol matches the protocol of the backend service. For example, an HTTP/2 health check most accurately tests HTTP/2 connectivity to backends.

Using gRPC with your Google Cloud applications

[gRPC](#)  is an open-source framework for remote procedure calls. It is based on the HTTP/2 standard. Use cases for gRPC include the following:

- Low latency, highly scalable, distributed systems
- Developing mobile clients that communicate with a cloud server
- Designing new protocols that must be accurate, efficient, and language independent
- Layered design to enable extension, authentication, and logging

To use gRPC with your Google Cloud applications, you must proxy requests end-to-end over HTTP/2. To do this with an external HTTP(S) load balancer:

1. Configure an HTTPS load balancer.
2. Enable HTTP/2 as the protocol from the load balancer to the backends.

The load balancer negotiates HTTP/2 with clients as part of the SSL handshake by using the ALPN TLS extension.

The load balancer may still negotiate HTTPS with some clients or accept insecure HTTP requests on an external HTTP(S) load balancer that is configured to use HTTP/2 between the load balancer and the backend instances. Those HTTP or HTTPS requests are transformed by the load balancer to proxy the requests over HTTP/2 to the backend instances.

If you want to configure an external HTTP(S) load balancer by using HTTP/2 with Google Kubernetes Engine Ingress or by using gRPC and HTTP/2 with Ingress, see [HTTP/2 for load balancing with Ingress](#).

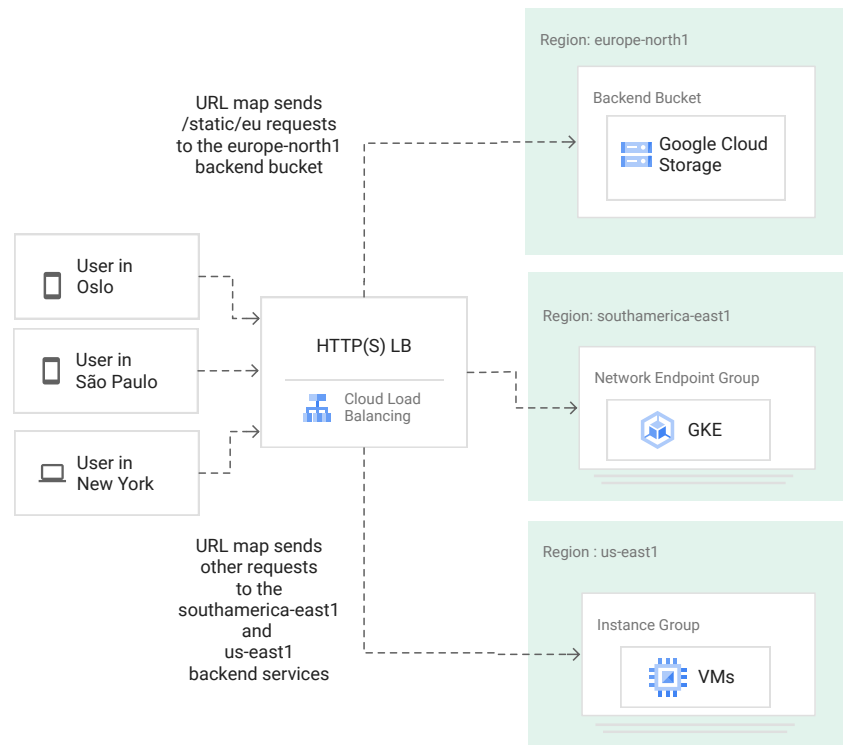
For information about troubleshooting problems with HTTP/2, see [Troubleshooting issues with HTTP/2 to the backends](#).

For information about HTTP/2 limitations, see [HTTP/2 limitations](#).

Backend buckets

Backend buckets direct incoming traffic to [Cloud Storage buckets](#).

As shown in the following diagram, you can have the load balancer send traffic with a path of `/static` to a storage bucket and all other requests to your other backends.



Distributing traffic to various backend types with HTTP(S) Load Balancing (click to enlarge)

For an example showing how to add a bucket to an existing load balancer, see [Setting up a load balancer with backend buckets](#).

Firewall rules

The backend instances must allow connections from the load balancer GFE/health check ranges. This means that you must [create a firewall rule](#) that allows traffic from `130.211.0.0/22` and `35.191.0.0/16` to reach your backend instances or endpoints. These IP address ranges are used as sources for health check packets and for all load-balanced packets sent to your backends.

The ports you configure for this firewall rule must allow traffic to backend instances or endpoints:

- You must allow the ports used by each forwarding rule
- You must allow the ports used by each health check configured for each backend service

Firewall rules are implemented at the VM instance level, not on Google Front End (GFE) proxies. You cannot use Google Cloud firewall rules to prevent traffic from reaching the load balancer.

For more information about health check probes and why it's necessary to allow traffic from `130.211.0.0/22` and `35.191.0.0/16`, see [Probe IP ranges and firewall rules](#).

Return path

Google Cloud uses special routes not defined in your VPC network for health checks. For more information, see [Load balancer return paths](#).

Load distribution algorithm

When you add a backend instance group or NEG to a backend service, you specify a *balancing mode*, which defines a method measuring backend load and a target capacity. External HTTP(S) Load Balancing supports two balancing modes:

- `RATE`, for instance groups or NEGs, is the target maximum number of requests (queries) per second (RPS, QPS). The target maximum RPS/QPS can be exceeded if all backends are at or above capacity.
- `UTILIZATION` is the backend utilization of VMs in an instance group.

Before a Google Front End (GFE) sends requests to backend instances, the GFE estimates which backend instances have capacity to receive requests. This capacity estimation is made proactively, not at the same time as requests are arriving. The GFEs receive periodic information about the available capacity and distribute incoming requests accordingly.

What *capacity* means depends in part on the balancing mode. For the `RATE` mode, it is relatively simple: a GFE determines exactly how many requests it can assign per second. `UTILIZATION`-based load balancing is more complex: the load balancer checks the instances' current utilization and then estimates a query load that each instance can handle. This estimate changes over time as instance utilization and traffic patterns change.

Both factors—the capacity estimation and the proactive assignment—influence the distribution among instances. Thus, Cloud Load Balancing behaves differently from a simple round-robin load balancer that spreads requests exactly 50:50

between two instances. Instead, Google Cloud load balancing attempts to optimize the backend instance selection for each request.

For more information, see [Traffic distribution](#).

For more information about the balancing modes, see [Balancing mode](#).

Network Service Tiers

When an external HTTP(S) load balancer is in Premium Tier, requests sent to the load balancer are delivered to backend instance groups or NEGs in the region closest to the user, if a backend in that region has available capacity. (Available capacity is configured by the load balancer's balancing mode.)

When an external HTTP(S) load balancer is in Standard Tier, its backend instance groups or NEGs must all be located in the region used by the load balancer's external IP address and forwarding rule.

Regions and zones

After a region is selected:

- When you configure backends in multiple zones within the region, an external HTTP(S) load balancer tries to balance requests as evenly as possible across the zones, subject to backend instance capacity and [session affinity](#).
- Within a zone, an external HTTP(S) load balancer tries to balance requests by using the load balancing algorithm, subject to available capacity and session affinity.

Session affinity

[Session affinity](#) provides a best-effort attempt to send requests from a particular client to the same backend for as long as the backend is healthy and has the capacity, according to the configured balancing mode.

Google Cloud HTTP(S) Load Balancing offers three types of session affinity:

- **NONE.** Session affinity is not set for the load balancer.
- [Client IP affinity](#) sends requests from the same client IP address to the same backend.

- [Generated cookie affinity](#) sets a client cookie when the first request is made, and then sends requests with that cookie to the same backend.

When you use session affinity, we recommend the `RATE` balancing mode rather than `UTILIZATION`. Session affinity works best if you set the balancing mode to requests per second (RPS).

WebSocket proxy support

HTTP(S) Load Balancing has native support for the WebSocket protocol when you use HTTP or HTTPS, not HTTP/2, as the protocol to the backend.

Backends that use the WebSocket protocol to communicate with clients can use the external HTTP(S) load balancer as a frontend for scale and availability. The load balancer does not need any additional configuration to proxy WebSocket connections.

The WebSocket protocol, which is defined in [RFC 6455](#), provides a full-duplex communication channel between clients and servers. The channel is initiated from an HTTP(S) request.

When HTTP(S) Load Balancing recognizes a WebSocket `Upgrade` request from an HTTP(S) client and the request is followed by a successful `Upgrade` response from the backend instance, the load balancer proxies bidirectional traffic for the duration of the current connection. If the backend does not return a successful `Upgrade` response, the load balancer closes the connection.

The timeout for a WebSocket connection depends on the configurable *response timeout* of the load balancer, which is 30 seconds by default. This timeout is applied to WebSocket connections regardless of whether they are in use. For more information about the response timeout and how to configure it, see [Timeouts and retries](#).

If you have configured either client IP or generated cookie session affinity for your external HTTP(S) load balancer, all WebSocket connections from a client are sent to the same backend instance, if the instance continues to pass health checks and has capacity.

The WebSocket protocol is [supported with Ingress](#).

QUIC protocol support for HTTPS Load Balancing

HTTPS Load Balancing supports the [QUIC protocol](#) in connections between the load balancer and the clients. QUIC is a transport layer protocol that provides congestion control similar to TCP and the security equivalent to SSL/TLS for HTTP/2, with improved performance. QUIC allows faster client connection initiation, eliminates head-of-line blocking in multiplexed streams, and supports connection migration when a client's IP address changes.

QUIC affects connections between clients and the load balancer, not connections between the load balancer and its backends.

The target proxy's QUIC override setting allows you to enable one of the following:

- Negotiate QUIC for a load balancer when possible.
- Always disable QUIC for a load balancer.

If you do not specify a value for the QUIC override setting, you allow Google to manage when QUIC is used. Google enables QUIC only when the `--quic-override` flag in the `gcloud` command-line tool is set to `ENABLE` or the `quicOverride` flag in the [REST API](#) is set to `ENABLE`.

For information about enabling and disabling QUIC support, see [Target proxies](#). You can enable or disable QUIC support as follows:

- In the load balancer's frontend configuration section of the Google Cloud Console
- By using the `--quic-override` flag with the `gcloud compute target-https-proxies update` command
- By using the `targetHttpsProxies.setQuicOverride` API method

QUIC has the following SSL certificate requirements:

- The SSL certificate's commonName (CN) attribute must match a DNS name that resolves to the load balancer's IP address. Otherwise, the load balancer doesn't serve QUIC content.
- The SSL certificate must be signed by a certificate authority (CA) that is trusted by the client. QUIC transport cannot be negotiated if your load balancer uses a self-signed certificate or one signed by a CA that isn't trusted by the client.

How QUIC is negotiated

When you enable QUIC, the load balancer can advertise its QUIC capability to clients, allowing clients that support QUIC to attempt to establish QUIC connections with the HTTPS load balancer. Properly implemented clients always fall back to

HTTPS or HTTP/2 when they cannot establish a QUIC connection. Because of this fallback, enabling or disabling QUIC in the load balancer does not disrupt the load balancer's ability to connect to clients.

When you have QUIC enabled in your HTTPS load balancer, some circumstances can cause your client to fall back to HTTPS or HTTP/2 instead of negotiating QUIC. These include the following:

- When a client supports versions of QUIC that are not compatible with the QUIC versions supported by the HTTPS load balancer.
- When the load balancer detects that UDP traffic is blocked or rate-limited in a way that would prevent QUIC from working.
- If QUIC is temporarily disabled for HTTPS load balancers in response to bugs, vulnerabilities, or other concerns.

When a connection falls back to HTTPS or HTTP/2 because of these circumstances, we do not count this as a failure of the load balancer.

Ensure that the previously described behaviors are acceptable for your workloads before you enable QUIC.

SSL certificates

You must reference one or more [SSL certificates](#) on the target HTTPS proxy.

These certificates are used by target HTTPS proxies to secure communications between a Google Front End (GFE) and the client. These can be self-managed or Google-managed SSL certificates.

For information about SSL certificate limits and quotas, see [SSL certificates](#) on the load balancing quotas page.

For the best security, you can also encrypt traffic from GFEs to your backends. For more information, see [Encryption from the load balancer to the backends](#).

For general information about how Google encrypts user traffic, see the [Encryption in Transit in Google Cloud](#) white paper.

TLS support

By default, an HTTPS target proxy accepts only TLS 1.0, 1.1, 1.2 and 1.3 when terminating client SSL requests. You can [use SSL policies](#) to change this default behavior and control how the load balancer negotiates SSL with clients.

When the load balancer uses HTTPS as a backend service protocol, it can negotiate TLS 1.0, 1.1, or 1.2 to the backend.

Timeouts and retries

HTTP(S) Load Balancing has two distinct types of timeouts:

- A configurable HTTP **response timeout**, which represents the amount of time the load balancer waits for your backend to return a complete HTTP response. The default value for the response timeout is 30 seconds. Consider increasing this timeout under any of these circumstances:
 - You expect a backend to take longer to return HTTP responses.
 - You see an HTTP 408 responses with the `jsonPayload.statusDetail client_timed_out`.
 - The connection is upgraded to a WebSocket (HTTP(S) Load Balancing only)





For WebSocket traffic sent through the load balancer, the backend service timeout is interpreted as the maximum amount of time that a WebSocket connection can remain open, whether idle or not. For more information, see [Backend service settings](#).

★ **Note:** The response timeout is not an HTTP idle (keepalive) timeout. It is possible that input and output (IO) from the backend is blocked due to a slow client (a browser with a slow connection, for example). This wait time isn't counted against the response timeout.

- A **TCP session timeout**, whose value is fixed at 10 minutes (600 seconds). This session timeout is sometimes called a keepalive or idle timeout, and its value is not configurable by modifying your backend service. You must configure the web server software used by your backends so that its keepalive timeout is longer than 600 seconds to prevent connections from being closed prematurely by the backend. This timeout *does not apply to WebSockets*.

This table illustrates changes necessary to modify keepalive timeouts for common web server software:

Web server software	Parameter	Default setting	Recommended setting
---------------------	-----------	-----------------	---------------------

Apache 	KeepAliveTimeout 	KeepAliveTimeout 5	KeepAliveTimeout 620
nginx 	keepalive_timeout 	keepalive_timeout 75s;	keepalive_timeout 620s;

The load balancer retries failed GET requests in certain circumstances, such as when the response timeout is exhausted. It does not retry failed POST requests. Retries are limited to two attempts. Retried requests only generate one log entry for the final response.

For more information, see [HTTP\(S\) Load Balancing logging and monitoring](#).

Illegal request and response handling

The external HTTP(S) load balancer blocks both client requests and backend responses from reaching the backend or the client, respectively, for a number of reasons. Some reasons are strictly for HTTP/1.1 compliance and others are to avoid unexpected data being passed to or from the backends. None of the checks can be disabled.

The load balancer blocks the following for HTTP/1.1 compliance:

- It cannot parse the first line of the request.
- A header is missing the `:` delimiter.
- Headers or the first line contain invalid characters.
- The content length is not a valid number, or there are multiple content length headers.
- There are multiple transfer encoding keys, or there are unrecognized transfer encoding values.
- There's a non-chunked body and no content length specified.
- Body chunks are unparseable. This is the only case where some data reaches the backend. The load balancer closes the connections to the client and backend when it receives an unparseable chunk.

The load balancer blocks the request if any of the following are true:

- The total size of request headers and the request URL exceeds the limit for the maximum request [header size for external HTTP\(S\) Load Balancing](#).
- The request method does not allow a body, but the request has one.

- The request contains an `Upgrade` header, and the `Upgrade` header is not used to enable WebSocket connections.
- The HTTP version is unknown.

The load balancer blocks the backend's response if any of the following are true:

- The total size of response headers exceeds the limit for maximum response header size for external HTTP(S) Load Balancing.
- The HTTP version is unknown.

Specifications and limitations

- HTTP(S) Load Balancing supports the `HTTP/1.1 100 Continue` response.

HTTP/2 limitations

- HTTP/2 between the load balancer and the instance can require significantly more TCP connections to the instance than HTTP(S). Connection pooling, an optimization that reduces the number of these connections with HTTP(S), is not currently available with HTTP/2.
- HTTP/2 between the load balancer and the backend does not support:
 - Server push
 - WebSockets

Restriction on using Cloud CDN

- You cannot enable Identity-Aware Proxy or Cloud CDN with the same backend service. If you try to do so, the configuration process fails.

What's next

- To learn about the external HTTP(S) load balancer setup, see [Setup overview for HTTP\(S\) Load Balancing](#).

- To build a working load balancer, see [Setting up a simple external HTTP load balancer](#) or [Setting up a simple external HTTPS load balancer](#).
- To create an HTTPS load balancer that uses content-based and cross-region load balancing, see [Creating an HTTPS load balancer](#).
- To find the locations for Google PoPs, see [GFE locations](#).
- To learn about Google Cloud Armor, which provides security for HTTP(S) Load Balancing at the network edge with security policies and rules that filter traffic based on layer 3, 4, and 7 attributes, see the [Google Cloud Armor security policy overview](#).
- To learn about Cloud CDN, which works with HTTP(S) Load Balancing to deliver content to your users, see the [Cloud CDN overview](#).
- To learn about capacity management, see [Capacity Management with Load Balancing tutorial](#) and [Application Capacity Optimizations with Global Load Balancing](#).
- To learn about serving websites, see [Serving websites](#).

Was this page helpful?



[Send feedback](#)

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see the [Google Developers Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2020-08-12 UTC.

Why Google

Choosing Google Cloud

Trust and security

Products and pricing

GCP pricing

G Suite pricing

Solutions

Infrastructure
modernization

Resources

GCP documentation

GCP quickstarts

Engage

Contact sales

Find a Partner

Open cloud

Global infrastructure

Customers and case studies

Analyst reports

Whitepapers

Maps Platform pricing

See all products

Data management

Application modernization

Smart analytics

Artificial Intelligence

Security

Productivity & work
transformation

Industry solutions

DevOps solutions

Small business solutions

See all solutions

Google Cloud Marketplace

G Suite Marketplace

Support

Tutorials

Training

Certifications

Google Developers

Google Cloud for Startups

System status

Release Notes

Become a Partner

Blog

Events

Podcast

Community

Press center

Google Cloud on YouTube

GCP on YouTube

G Suite on YouTube

Follow on Twitter

Join User Research

We're hiring. Join Google
Cloud!

[About Google](#) | [Privacy](#) | [Site terms](#) | [Google Cloud terms](#)

[Sign up for the Google Cloud newsletter](#)

[Subscribe](#)

[Language ▼](#)